Bastian Goldlücke

# Multi-Camera Reconstruction and Rendering for Free-Viewpoint Video

– Ph.D. Thesis –

November 29, 2006

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

**Bibliografische Informationen der Deutschen Bibliothek**
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

1. Aufl. - Göttingen : Cuvillier, 2005
   Zugl.: Saarbrücken, Univ., Diss., 2005
   ISBN X-XXXXX-XXX-X

Dissertation zur Erlangung des Grades
*Doktor der Naturwissenschaften (Dr. rer. nat.)*
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Eingereicht am 30. September 2005 in Saarbrücken durch

Bastian Goldlücke
MPI Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken

`mail@bastian-goldluecke.de`
`www.bastian-goldluecke.de`

**Betreuender Hochschullehrer – Supervisor**
Dr. Marcus A. Magnor, MPI für Informatik, Saarbrücken

**Gutachter – Reviewers**
Dr. Marcus A. Magnor, MPI für Informatik, Saarbrücken
Prof. Dr. Joachin Weickert, Universität des Saarlandes, Saarbrücken

**Dekan – Dean**
Prof. Dr. Jörg Eschmeier

## Abstract

While virtual environments in interactive entertainment become more and more lifelike and sophisticated, traditional media like television and video have not yet embraced the new possibilities provided by the rapidly advancing processing power. In particular, they remain as non-interactive as ever, and do not allow the viewer to change the camera perspective to his liking. The goal of this work is to advance in this direction, and provide essential ingredients for a free-viewpoint video system, where the viewpoint can be chosen interactively during playback.

Knowledge of scene geometry is required to synthesize novel views. Therefore, we describe 3D reconstruction methods for two distinct kinds of camera setups. The first one is depth reconstruction for camera arrays with parallel optical axes, the second one surface reconstruction, in the case that the cameras are distributed around the scene. Another vital part of a 3D video system is the interactive rendering from different viewpoints, which has to perform in real-time. We cover this topic in the last part of this thesis.

## Kurzfassung

Während die virtuellen Welten in interaktiven Unterhaltungsmedien immer realitätsnäher werden, machen traditionellere Medien wie Fernsehen und Video von den neuen Möglichkeiten der rasant wachsenden Rechenkapazität bisher kaum Gebrauch. Insbesondere mangelt es ihnen immer noch an Interaktivität, und sie erlauben dem Konsumenten nicht, elementare Parameter wie zum Beispiel die Kameraperspektive seinen Wünschen anzupassen. Ziel dieser Arbeit ist es, die Entwicklung in diese Richtung voranzubringen und essentielle Bausteine für ein Videosystem bereitzustellen, bei dem der Blickpunkt während der Wiedergabe jederzeit völlig frei gewählt werden kann.

Um neue Ansichten synthetisieren zu können, ist zunächst Kenntnis von der 3D Geometrie der Szene notwendig. Wir entwickeln daher Rekonstruktionsalgorithmen für zwei verschiedene Anordnungen von Kameras. Falls die Kameras eng beieinanderliegen und parallele optische Achsen haben, können lediglich Tiefenkarten geschätzt werden. Sind die Kameras jedoch im einer Halbkugel um die Szene herum montiert, so rekonstruieren wir sogar echte Oberflächengeometrie. Ein weiterer wichtiger Aspekt ist die interaktive Darstellung der Szene aus neuen Blickwinkeln, die wir im letzten Teil der Arbeit in Angriff nehmen.

# Summary

Interactive entertainment is starting to play a major role nowadays. Modern grapics hardware becomes more and more sophisticated, and lifelike dynamic scenes can be rendered, becoming virtually indistinguishable from their real-world counterparts recorded with cameras. In contrast, television and video do not make use of the new possibilities provided by modern technology yet. Both lack interactivity and do not allow the user to adjust important parameters to his liking, in particular the viewpoint from which a scene is being watched.

Our work aims at 3D video and 3D television, which enables the user to arbitrarily change the perspective during playback. A key requirement is to provide additional geometric information, which makes it possible to synthesize novel views. The primary goal, therefore, is to obtain a high-quality representation of the geometry visible in a scene. We describe 3D reconstruction methods for two distinct kinds of camera setups. If the cameras are closely packed in an array with parallel optical axes, we reconstruct dense depth maps for each camera image. If, on the other hand, the cameras are surrounding the scene, we recover dynamic surface models. We also discuss ways to generate novel views interactively from the geometry model and the source images on modern graphics hardware.

## Depth Reconstruction

Our goal in the first part of the thesis is to reconstruct by photometric means a dense depth map for each frame of multiple video sequences captured with a number of calibrated video cameras. A depth map assigns a depth value to each image pixel, determining its location in 3D space. Simultaneously, we want to decide for every pixel whether or not it belongs to the background of the scene, known from background images captured with the same cameras.

While previous work was restricted to static scenes, an important visual clue available in video sequences is temporal coherence. For example, it is highly unlikely that large background regions suddenly turn into foreground, or that the depth of a region changes dramatically without accompanying changes in color. We present a framework to consistently estimate depth exploiting spatial as well as temporal coherence constraints, which is based on a global minimization of a discrete energy functional via graph cuts. The minimum of the functional yields the final reconstruction result. Particularly important advantages we inherit from the underlying approach include that all cameras are treated symmetrically, and that visibility is handled properly.

## Surface Reconstruction

In the second part of the thesis, we take the reconstruction problem one step further and are not content with simple depth maps anymore. Instead, our aim is to recover the full 3D geometry of arbitrary objects, using multi-video data from a handfull of cameras surrounding the scene as input.

The geometry estimate is defined as a *weighted minimal surface*, which minimizes an energy functional given as a surface integral of a scalar-valued weight or error function. The variational formulation of these kinds of minimization problems leads to a partial differential equation (PDE) for a surface evolution, which can be explicitly solved using a level set technique. We derive this equation for arbitrary dimensional surfaces and a large class of error functions.

Our first method based upon the equation is a spatio-temporal 3D reconstruction scheme. The full space-time geometry of the scene is recovered for all frames simultaneously by reconstructing a single hypersurface photo-consistent with all input images, whose slices with planes of constant time yield the 2D surface geometry at each time instant. Because the reconstructed surface is continuous in the temporal direction as well, temporal coherence is intrinsic to our method. As a second method based upon our surface evolution theory, we describe an algorithm how the bodies of homogenous, refractive media like water can be reconstructed using a sophisticated error functional.

## Video-based Rendering

In the final stage of a free-viewpoint video system, the geometric data has to be exploited to create high-quality novel views of a scene. The two scenarios we analyzed require two distinct different kinds of rendering algorithms, which we present in the last part of the thesis. Both schemes have in common that the source video images are mapped onto the target geometry using modern graphics hardware.

Novel views from video streams with accompanying depth information are created by warping and blending the source images using an underlying triangle mesh. If one has a surface representation of the scene instead, the textures obtained from the input videos are projected onto the geometry using projective texturing with careful selection of input camera views and weights.

# Zusammenfassung

Interaktive Unterhaltungsmedien spielen eine immer grösser werdende Rolle. Mit Hilfe leistungsfähiger moderner Grafikhardware werden beinahe lebensechte Szenen dargestellt, die sich von ihren mit einer Kamera aufgenommenen Gegenstücken aus der realen Welt bald kaum noch unterscheiden werden. Im Vergleich dazu werden die durch neue Technologien gebotenen Möglichkeiten sowohl vom Fernsehen als auch Video bisher sträflich vernachlässigt. Beiden Medien mangelt es an Interaktivität, und sie erlauben dem Konsumenten nicht, wesentliche Parameter wie den Blickpunkt der Wiedergabe seinen Vorstellungen anzupassen.

Unsere Arbeit hat 3D-Video und 3D-Fernsehen zum Ziel, die es dem Benutzer ermöglichen sollen, während der Wiedergabe die Perspektive frei zu wählen. Eine wesentliche Voraussetzung ist dabei zusätzliche Geometrieinformation, welche die Synthese neuer Ansichten ermöglicht. Das Hauptziel dieses Textes ist daher die Entwicklung von Verfahren zur qualitativ hochwertigen Rekonstruktion einer Szene als dreidimensionales Modell. Wir beschreiben Rekonstruktionsverfahren für zwei wesentlich verschiedene Kamerakonfigurationen. Sind die Kameras in einem Gitter mit parallelen optischen Achsen montiert, so können Tiefenkarten für jedes Einzelbild in den Videodatenströmen erstellt werden. Für den Fall, daß die Kameras um die Szene herum verteilt sind, ist es sogar möglich, ein dynamisches Oberfächenmodell zu gewinnen. Schließlich stellen wir noch Verfahren zur Verfügung, aus der erhaltenen Geometrie und den Eingabevideos neue Ansichten interaktiv zu generieren.

## Tiefenkarten

Das Ziel im ersten Teil der Arbeit ist es, mit photometrischen Methoden Tiefenkarten für Videobilder zu gewinnen, die von mehreren synchronisierten Kameras aufgenommen wurden. Jedem Bildpunkt wird dabei eine Tiefe zugeordnet, welche seine Position im Raum festlegt. Gleichzeitig soll für jeden Bildpunkt entschieden werden, ob er zum bewegten Vordergrund der Szene gehört oder zum statischen Hintergrund, welcher durch ein Hintergrundbild für jede Kamera definiert wird.

Obwohl sich bisherige Arbeiten auf statische Szenen beschränken, gibt doch zeitliche Kohärenz in aufeinanderfolgenden Bildern in einer Videosequenz wichtige Hinweise für eine erfolgreiche Rekonstruktion. So ist es zum Beispiel sehr unwahrscheinlich, daß große Regionen des Hintergrundes plötzlich in Vordergrund mutieren, oder daß sich die Tiefe in einem Bildbereich drastisch ändert, ohne daß dabei auch die Farbe der Bildpunkte anders wird.

Wir beschreiben daher ein allgemeines Verfahren, um konsequent die Restriktionen auszunutzen, die sich durch Photokonsistenz und zeitlich-räumliche Kontinuitätsforderungen ergeben. Es basiert auf einem erfolgreichen Algorithmus, der Tiefenkarten durch Minimierung eines diskreten Energiefunktionals bestimmt. Das Minimum kann durch Minimalschnitte in Graphen berechnet werden. Wichtige Vorteile, die wir von der ursprünglichen Methode erben, ist Symmetrie in den Kameras und die korrekte Behandlung der Sichtbarkeit von Objekten.

## Oberflächenrekonstruktion

Im zweiten Teil der Arbeit betrachten wir das Rekonstruktionsproblem von einer höheren Warte aus und versuchen, die komplette Oberflächengeometrie der Szene zu erhalten, wobei die Videodaten einer handvoll Kameras verwendet werden, die um die Szene herum verteilt sind.

Die Zielgeometrie ist definiert als eine gewichtete Minimalfläche, die ein Energiefunktional minimiert, welches als Oberflächenintegral einer Gewichts- oder Fehlerfunktion gegeben ist. Die Variationsformulierung dieses Minimierungsproblems liefert eine partielle Differentialgleichung für eine Oberflächenevolution. Eine Lösung kann über eine Implementation der Evolution durch implizite Flächen gelöst werden. Wir leiten die Differentialgleichung für Flächen beliebiger Dimension und eine große Klasse von Fehlerfunktionen mathematisch her.

Unser erstes praktisches Verfahren, das auf dieser Gleichung basiert, ist die räumlich-zeitliche Rekonstruktion von Geometrie aus Multivideo-Sequenzen. Die komplette Raumzeitgeometrie der Szene wird als eine einzige Hyperfläche gewonnen, welche photokonsistent mit allen Eingabebildern zugleich ist. Schnitte der Hyperfläche mit Ebenen konstanter Zeit ergeben die 2D Oberflächengeometrie zum entsprechenden Zeitpunkt. Da die rekonstruierte Oberfläche stetig auch in der zeitlichen Richtung ist, ist zeitliche Kohärenz implizit in das Rekonstruktionsverfahren eingebaut. Als zweites Verfahren, welches auf der allgemeinen Evolutionsgleichung basiert, stellen wir einen Algorithmus vor, welcher Volumen aus homogenen Medien mit Brechung, wie zum Beispiel Wasser, rekonstruieren kann. Dies erfordert die Definition einer ausgeklügelten Fehlerfunktion, die für diesen Fall maßgeschneidert ist.

## Videobasiertes Rendering

Im der letzten Arbeitsphase eines Videosystems mit freier Wahl des Blickpunktes muß die Geometrieinformation ausgenutzt werden, um qualitativ ansprechende neue Ansichten der Szene zu generieren. Die beiden von uns untersuchten Szenarien erfordern dabei zwei grundverschiedene Verfahren, die im letzten Teil der Arbeit präsentiert werden. Beide Verfahren haben gemeinsam, dass die Eingabevideos auf die Zielgeometrie unter Verwendung moderner Grafikhardware abgebildet werden.

Für Videodatenströme mit Tiefeninformation werden neue Ansichten generiert, indem die Eingabebilder auf ein Dreiecksnetz “geklebt” und mit diesem entsprechend der Tiefe der Eckpunkte in die neue Ansicht projiziert werden. Die Beiträge verschiedener Kameras werden dabei gemäß ihrem Abstand zur neuen Ansicht gewichtet. Falls stattdessen ein Oberflächenmodell der Geometrie zur Verfügung steht, werden die Eingabebilder während des Renderns als Texturen auf das Modell projiziert. Die verwendeten Bilder und ihre Gewichtung bei der Überblendung müssen dabei sorgfältig ausgewählt werden.

# Preface

*He stood in the center of Heaven and looked about it, having decided to have four eyes today. He noticed that with less than two looking in any one direction, he couldn't see as well as he ought. He resolved to set someone to discover the reason for this.*

Steven Brust, "To Reign in Hell".

Nowadays, Computer Vision has progressed far beyond the relatively simple question posed in the quote. Not only do we know that two views are necessary for stereo vision in order to obtain correspondences for triangulation, we can even design algorithms which mimic this process. And although the human visual system still performs far superior to machines when it comes to scene reconstruction, the latter progress fast and well. Vision algorithms become more and more robust and sophisticated, and modern techniques can integrate the information from a lot more than just two cameras and views to create a geometric model of a scene. Recently, researchers also began to exploit temporal coherence in video sequences.

The intention of this thesis is to progress one step further towards accurate and reliable scene reconstruction. The focus is on being able to render the model from novel viewpoints. Each chapter after the introduction is a revised and extended version of a refereed conference paper I published during my three years as a researcher at the MPI Informatik in Saarbrücken. Chronologically, the first one was the Light Field Rendering System in Chapter 12, which was presented at the *Vision, Modeling and Visualisation* 2002 in Erlangen [37]. In its first incarnation, it was based on an existing depth map reconstruction algorithm. Shortly after, we developed our own technique, which performs an additional background subtraction and adds temporal coherence for better playback quality. Presented at the *Computer Vision and Pattern Recognition* 2003 in Madison [42] and in a sketch at Siggraph 2003 in San Diego, this technique now forms the basis for Chapters 5 and Chapter 6.

My second line of research also started with rendering, this time free-viewpoint rendering of the existing visual hull geometry provided by Christian Theobalt using projective texturing. The result was the billboard rendering technique presented first at the *Visual Communications and Image Processing* 2003 in Lugano [43], and in an extended version at the *International Conference on Image Processing* 2003 in Barcelona [38]. It can be found in Chapter 13 of this thesis. Subsequently, I developed the necessary 3D surface reconstruction algorithms, which I now consider my major contribution and the core of this thesis. Analytical techniques based on weighted minimal surfaces seemed most appealing to me because of the concise mathematical approach. After I had the idea to represent spatio-temporal geometry as a hypersurface, I started to develop the necessary mathematical tools, which were presented at the *European Conference on Computer Vision* 2004 in Prague, now forming Chapter 8 of this thesis. The reconstruction technique based on the mathematical method, Chapter 9, was later accepted for publication for the *Computer Vision and Pattern Recognition* 2004 in Washington [39]. Reconstruction of refractive materials, the latest method based on the surface evolution and joint work with Ivo Ihrke, will be presented at the *International Conference on Computer Vision* 2005 in Beijing [50].

I am very thankful for the excellent working conditions I was provided with here at the MPI. In particular, my gratitude goes to my supervisor Marcus Magnor, who expertly guided and supported my research. Thanks also to my friends and fellow researchers for helpful scientific discussions and lots of fun in pursuit of less serious matters. Finally, a special thanks to my family for their unconditional love and support over the years. I dedicate this thesis to Susanne, who makes my life so much better in countless ways.

Saarbrücken, Germany, *Bastian Goldlücke*
November 29, 2006

# Contents

# Notation

## Common Mathematical Symbols

| | |
|---|---|
| $\mathbb{R}^n$ | $n$-dimensional vector space over the real numbers. |
| $\mathbb{S}^n$ | $n$-dimensional sphere, unit sphere in $\mathbb{R}^{n+1}$. |
| $\lVert\cdot\rVert_2$ | Euclidean norm of a vector. |
| $\lVert\cdot\rVert_\infty$ | Maximum norm of a vector. |
| $\langle\mathbf{v},\mathbf{w}\rangle$ | Scalar product of $\mathbf{v}$ and $\mathbf{w}$. |
| $\chi_{\mathcal{A}}$ | Characteristic function of the set $\mathcal{A}$. |

## Cameras and Images

| | |
|---|---|
| $\{C_1,\dots,C_n\}$ | Set of $n$ cameras. $C_i$ is also identified with the center of projection of the $i$th camera. |
| $\mathcal{I}_i^t$ | Image recorded by camera $i$ at time $t$, a mapping from $\mathbb{R}^2$ into color space. |
| $\pi_i$ | The projection mapping from $\mathbb{R}^3$ into image space $\mathbb{R}^2$ of camera $i$. |

## Scene Geometry

| | |
|---|---|
| $\varSigma_t$ | The scene geometry surface at time $t$. |
| $\mathcal{O}_t$ | Scene geometry volume at time $t$, with $\partial\mathcal{O}_t = \varSigma_t$. |
| $\mathfrak{H}$ | The three-dimensional hypersurface traced by the $\varSigma_t$ as they evolve in space-time. |
| $\mathcal{V}_t$ | Image-based visual hull of $\mathcal{O}_t$ at time $t$. |
| $S_i^t$ | Silhouette of $\mathcal{O}_t$ in camera image $i$. |

## Surface Point Properties

| | |
|---|---|
| $\nu_i^t(p)$ | Denotes whether point $p$ is visible in camera $i$ at time $t$. |
| $c_i^t(p)$ | The color of point $p$ in camera $i$ at time $t$, equals $\mathcal{I}_i^t \circ \pi_i(p)$ if $p$ is visible. |

## Level Sets

| | |
|---|---|
| $u^t$ | Function $\mathbb{R}^3 \to \mathbb{R}$ whose zero level set is $\varSigma_t$. |
| $\varGamma$ | Regular grid discretizing the region to be reconstructed. |
| $u^{xyzt}$ | Value of $u$ in the grid cell $(x,y,z)$ at time $t$. |
| $u_i^{xyzt}$ | The same value at step $i$ of the evolution. |

# Part I

# Introduction

# 1

# Free-Viewpoint Video

## 1.1 Motivation

Nowadays, interactive entertainment plays a major role and is starting to outperform traditional media. The computer gaming industry had higher sales than Hollywood's movie industry for three successive years now, and online games attract millions of players worldwide. With the sophistication of modern grapics hardware, lifelike dynamic scenes can be rendered, which will soon be virtually indistinguishable from their real-world counterparts recorded with cameras. Many modern computer games more and more resemble interactive movies, with a professional storyline and character development.

In contrast, the television and video experience remains unchanging. Interactivity when watching TV or a video is restricted to adjusting the volume or switching to another channel. While this is probably desireable for certain movies, where perspective and lighting were carefully chosen by a director for dramatic effect, a sports documentation would be much more exciting if one could change the viewpoint interactively, for instance to get the desired view of the last goal during a soccer broadcast. Educational documentaries would benefit as well from more interactivity, since it is much easier to visualize complex structures, e.g. molecules or engines, if the viewer can rotate them by himself.

A growing crowd of researchers is therefore pursuing 3D video and 3D television, where modern computer graphics and vision techniques are to be combined to obtain a streamable medium which gives the user more control over the aspects of playback, in particular free choice of viewpoint. This thesis aims at providing some of the necessary tools required for a 3D video system. We start with an overview of such a system in the next section, and focus on specific aspects later in the thesis. Our primary goal is to obtain a high-quality representation of the geometry visible in a scene. This is a key requirement for being able to render novel views from arbitrary perspectives. We also discuss ways to generate these views interactively from the geometry model and the source images on modern graphics hardware.

## 1.2 Components of a Free-Viewpoint Video System

The full pipeline for a system capable of recording, encoding and high-quality playback of 3D video is very sophisticated and consists of a lot of different steps, each of which is a small challenge in itself. In this work, we can only adress a small subset of them in more detail. We chose two topics which are in our opinion the most interesting ones, 3D reconstruction and rendering, and we discuss them in the remaining parts of this thesis. For the sake of completeness, we briefly outline the other necessary steps here. Fig. 1.1 shows the whole 3D video pipeline at a glance.

**Camera Setup and Calibration.** Naturally, the first and foremost task is to acquire data. For our reconstruction algorithms, we require the internal and external camera parameters of each of the imaging devices to be available. In particular, we need to know the exact mapping of 3D points to 2D image coordinates. To achieve this, before recording any actual data, a calibration procedure is applied. A suitable one has to be chosen depending on the setup of the cameras. We review some choices when we present different recording setups in the following two parts.

**Data Aquisition.** To record multi-video sequences of a temporally varying scene, the necessary hardware effort is considerable. Multiple synchronized video cameras are needed to capture the scene from different viewpoints. Aside from the physical construction of the calibrated studio, the cameras have to be triggered simultaneously, and the huge amounts of data they produce must be stored on hard drives in real time.

With recording hardware becoming cheaper and cheaper, nowadays, several research labs around the world feature studios capable of recording multi-video sequences [137, 16, 46, 79, 84, 100, 136]. In this work, sequences from two different systems were used. The first is the Stanford Multi-Camera Array [137]. Its cameras are densely packed in one or more arrays, and the data is ideally suited for the depth reconstruction algorithms and light field rendering techniques investigated in Part II of this thesis. The second one is our own custom-made studio available at the MPI Informatik [130]. It captures wide-baseline multi-video sequences in a hemispherical setup around the scene. Thus, it is ideal to aquire data for the surface reconstruction techniques presented in Part III. Both capturing systems are described in the respective

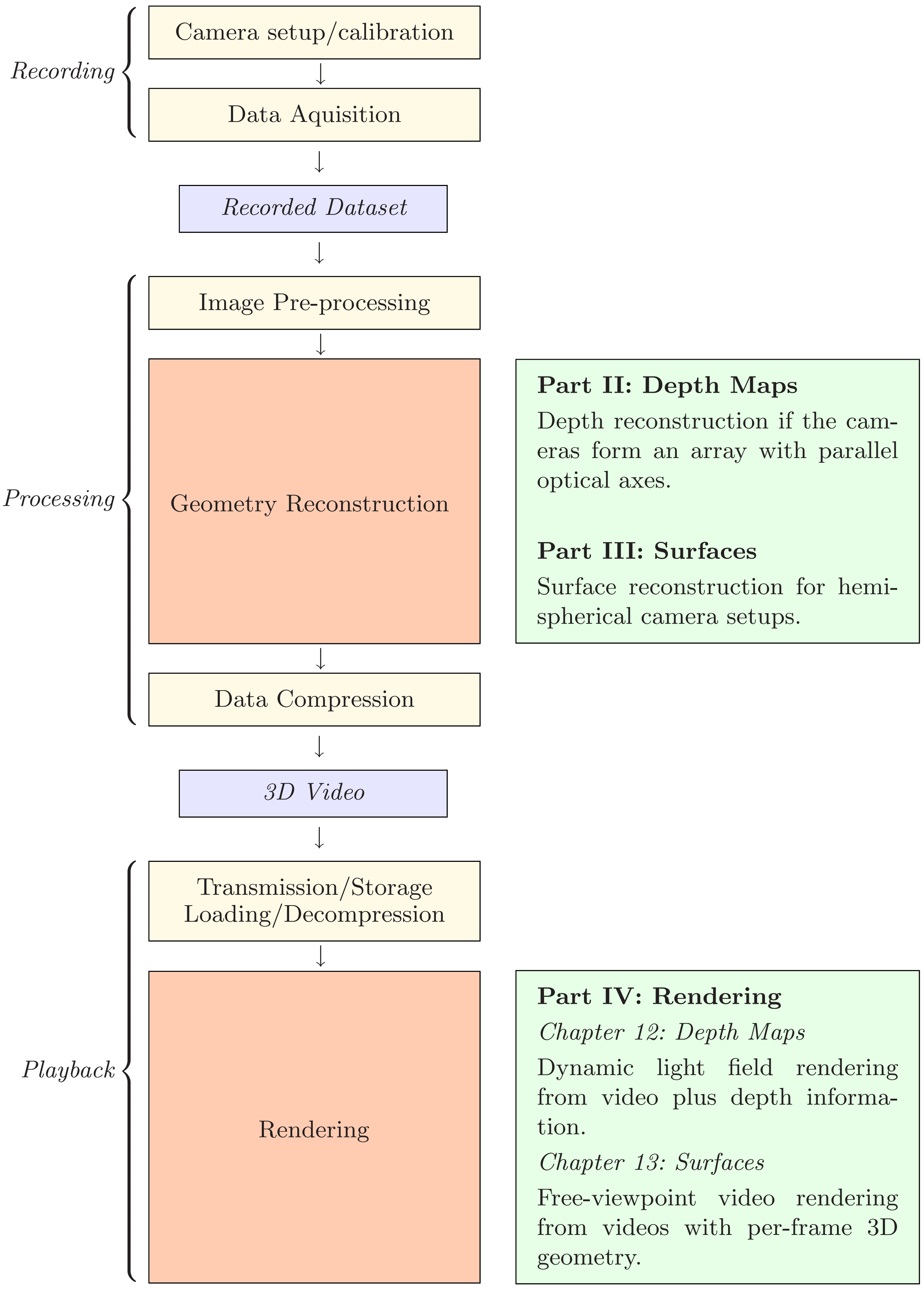


**Fig. 1.1.** The components forming the pipeline of a free-viewpoint video system, and where they are discussed in detail in this thesis.

part of the thesis where they are used, the Stanford Array in Sect. 4.2, and our in-house system in Sect. 7.4.

**Image Processing.** Before the reconstruction algorithms can be applied to the data, some image processing usually has to be performed. Colors have to be corrected in order to account for differences in the cameras' sensors, and for some vision techniques background subtraction is necessary. We do not discuss image processing techniques in more detail in this work. Instead, we assume that the input to our algorithms consists of sequences already pre-processed in a suitable manner.

**Geometry Reconstruction.** 3D video requires approximate scene geometry to be known for every frame of the sequence. Geometry reconstruction is one of the two major topics of this work. For setups with parallel optical axes, estimation of depth maps is the best one can hope for. We refer to these approaches as $2\frac{1}{2}$D. They are explored in Part II. In general camera setups, the recovery of true 3D surface geometry is desireable, as described in Part III.

**Data Compression.** Multi-camera video recordings produce huge amounts of data. Any useful 3D video format must include a very efficient way to reduce the redundancies in this data. It must be possible to load and decompress the video data in real-time for the rendering stage.

**3D Video Rendering.** Once the video data is uncompressed in main memory, it can be played back with the original frame rate. Ideally, the user is able to choose an arbitrary viewpoint without impacting the visual quality of the result. Real objects can be placed in virtual 3D environments, the environment can interact with them and vice versa. Rendering is the second main topic covered in this thesis, and discussed in detail in Part IV.

## 1.3 Outline of the Thesis

The focus of this work are methods for the reconstruction of $2\frac{1}{2}$D and 3D geometry in different camera setups, and photo-realistic rendering of the geometry from novel viewpoints, using the input images as texture. Thus, we only pursue the steps *Geometry Reconstruction* and *3D Video Rendering* of the free-viewpoint video pipeline.

In the next chapter, we introduce basic concepts and mathematical notation, which we use throughout the rest of the thesis. Afterwards, we present related work in Chapter 3.

This is followed by the analysis of the first kind of camera setups in Part II, where all cameras have parallel or near-parallel optical axes, as for instance in case of the Stanford aquisition system. For these setups, it is usually only possible to recover a $2\frac{1}{2}$D model of scene geometry in form of a *depth map* for each camera image. We present an algorithm which simultaneously estimates both a full set of dense depth maps for all cameras, as well as a segmentation into moving foreground and static background, Chapter 5. Temporal coherence can also be exploited in order to improve the accuracy of the estimate, Chapter 6.

Another kind of camera setup, which requires a substantially different representation of scene geometry as well as reconstruction and rendering techniques is the (hemi-)spherical setup, where the cameras are more or less evenly distributed around a moving object. This setup is analyzed in Part III. We present a space-time isosurface reconstruction technique in Chapter 9, which is based on the mathematical analysis of weighted minimal surfaces in Chapter 8. With a sophisticated choice of energy functional, the weighted minimal surface approach can also be employed to reconstruct solid transparent objects of homogenous index of refraction, Chapter 10.

The final Part IV is devoted to rendering techniques suitable for both kinds of setups. Rendering of dynamic light fields with depth information, which relies on the kind of data estimated in Part II, is presented in Chapter 12. Surfaces recovered from the 3D reconstruction techniques in Part III can be rendered in real-time and textured with the video data, as shown in Chapter 13.

# 2

# Basic Concepts and Notation

## 2.1 Scene Geometry

The goal of every reconstruction algorithm is to obtain an approximation to the exact scene geometry. At a single time-step $t \in \mathbb{R}$, we require the geometry to be a closed, piecewise differentiable two-dimensional manifold $\Sigma_t \in \mathbb{R}^3$. In particular, in this work we only deal with well-behaved, entirely solid objects with an open interior. Phenomena like smoke and foam, which consist of many small particles or are more of a fractal-like nature, are not being considered. However, we can work with large bodies of transparent, refractive materials, see Chapter 10.

The surface manifold is allowed to vary smoothly over time. If such a time-varying scene with moving objects is considered, the scene geometry $\Sigma_t$ evolves over time and traces a three-dimensional manifold $\mathfrak{H}$ in space-time, Fig. 2.1. The intersections of $\mathfrak{H}$ with planes of constant time $t$ yield the surface geometry $\Sigma_t$ at this moment.

The motion of surface points is described by the scene flow, which yields correspondences between points in surface manifolds at different time steps. It is a time-dependent vector field $\mathbf{v}_t : \Sigma_t \to \mathbb{R}^3$ which assignes to each point on the surface its current speed. Thus, the integral curves of the scene flow are the trajectories of surface points.

## 2.2 Multi-View Geometry

An essential idea of most 3D reconstruction schemes is that the location of 3D scene points can be computed from their projections. Indeed, in theory, if only two projections of the same 3D point $p$ in two different views are

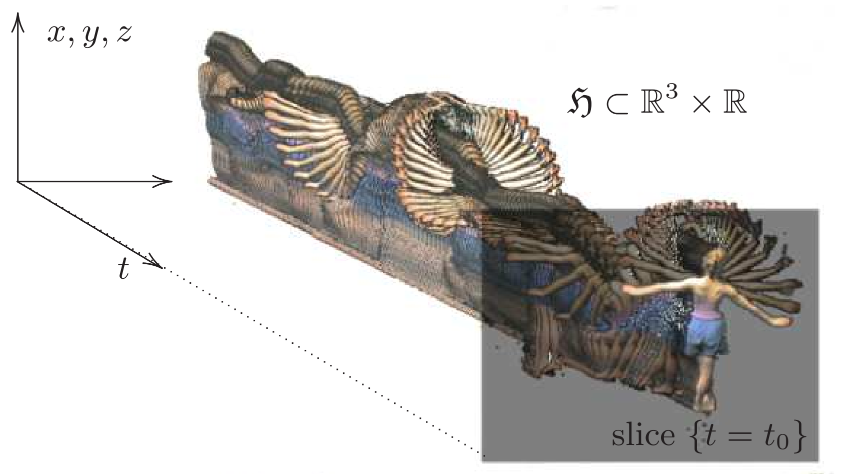


**Fig. 2.1.** A surface evolving over time defines a hypersurface $\mathfrak{H}$, the space-time geometry of the scene.

known, there is only one possibility for the location of $p$ if the cameras are in general positions. Fig. 2.2. Thus, one possible way to tackle the problem of 3D reconstruction lies in establishing those correspondences. In this section, we will explore a few basic mathematical concepts of multi-view geometry. In particular, we need the notion of the *Fundamental matrix*, which is a basic invariant of two-view geometry, relating a projected point in one view to its epipolar lines in another.

In the following, we always deal with multi-camera setups, and denote by $\{C_1, \ldots, C_n\}$ our set of $n$ cameras. At each time instant $t$, let $\mathcal{I}_i^t$ be the image of camera $C_i$. One of the main initial difficulties is to calibrate the camera system, i.e. estimate the mapping $\pi_i : \mathbb{R}^3 \to \mathbb{R}^2$ of 3D points to image coordinates for each of the cameras $C_i$. In practice, $\pi_i$ is usually not even projective linear, because of e.g. radial distortion one has to take into account. This work is situated one step further in the reconstruction pipeline, however, and we assume that all calibration has already been performed, and all source images have been rectified. Thus, we assume that we know all of the projection mappings, but keep in mind when designing our reconstruction algorithms that they are not perfect, as they were estimated with a computer vision technique themselves. The algorithms have to be robust enough to handle unavoidable small errors in camera calibration.

Let us now consider two of the cameras, and assume they are in general position. Their centers of projection are $C_i$ and $C_j$, through which 3D points are projected onto the image planes $\Pi_i$ and $\Pi_j$, respectively. If we fix $p_i \in \Pi_i$, then every 3D point which projects onto $p_i$ lies on $L(p_i) := \pi_i^{-1}(p_i)$, which is the line through $p_i$ and $C_i$, also called the optical ray of $p_i$. Fig. 2.2. Its projection into the $j$th view

$$F_{ij}(p_i) := \pi_j \circ \pi_i^{-1}(p_i),$$

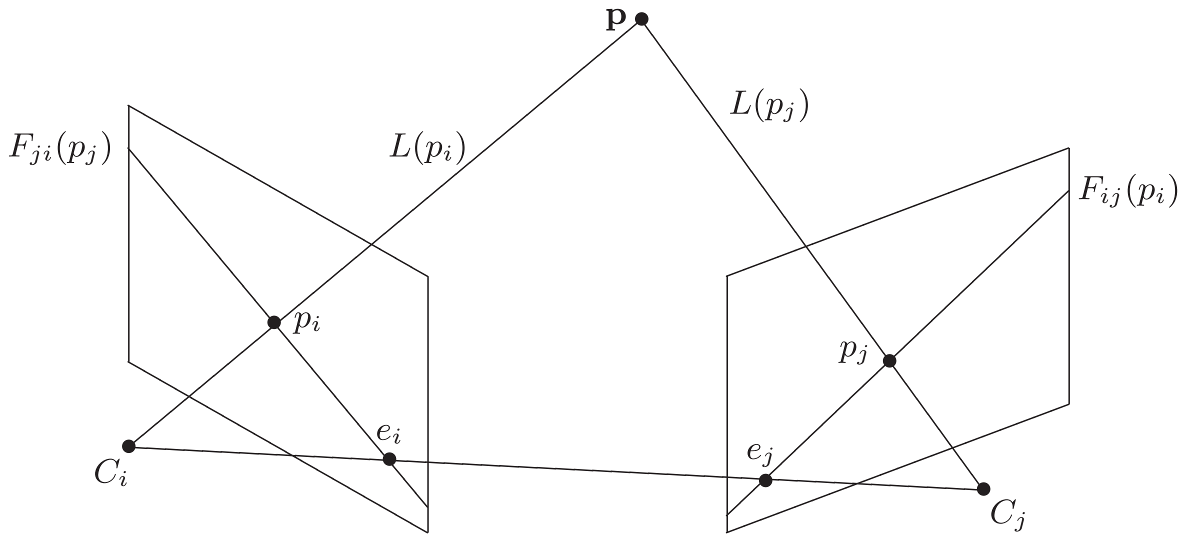


**Fig. 2.2.** *Epipolar Geometry.* Given $\mathbf{p}$, the point $p_j$ which corresponds to $p_i$ has to lie on its epipolar line $F_{ij}(p_i)$, and vice versa. Given a valid correspondence, the optical rays intersect in $\mathbf{p}$, and the optical rays are coplanar with the baseline $\overline{C_iC_j}$. The intersections $e_i$ and $e_j$ of the baseline with the image planes are called the epipoles.

is called the epipolar line of $p$ in camera $C_j$, and the mapping $F_{ij}$ the *Fundamental mapping* from $C_i$ to $C_j$. We assume that all images are rectified, so that the projections $\pi_i$ are projective linear and $F_{ij}(p_i)$ is indeed a line. Under these conditions, $F_{ij}$ can be written as a $3 \times 3$ matrix $\mathbf{F}_{ij}$, where the 3-vector $\mathbf{F}_{ij}p_i$ is exactly the projective representation of the epipolar line of $p_i$.

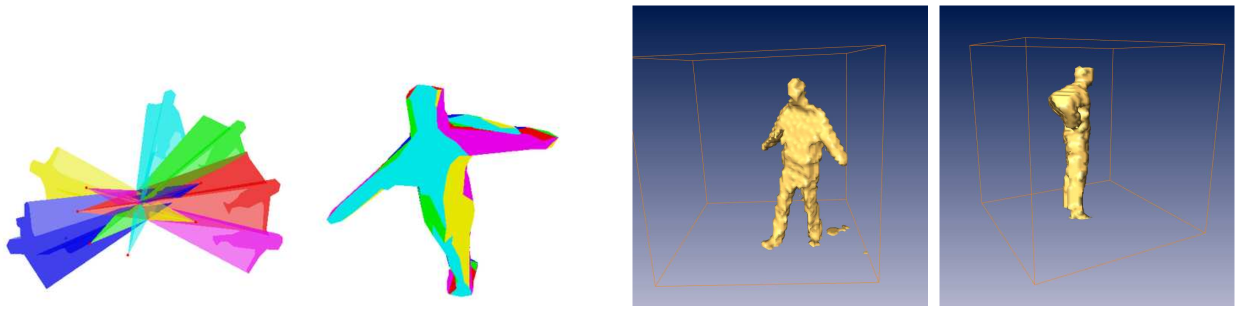
**Definition.** The $3 \times 3$ matrix $\mathbf{F}_{ij}$, which maps points in image $i$ onto the projective representation of their epipolar line in image $j$, is called the Fundamental matrix relating camera $C_i$ to camera $C_j$.

In particular, let the point $p_j \in \Pi_j$ correspond to $p_i \in \Pi_i$, which means that both are projections of the same 3D point $\mathbf{p}$. Then the optical rays $L(p_i)$ and $L(p_j)$ intersect in $\mathbf{p}$, and $p_j$ satisfies the *epipolar constraint*

$$p_j^T \mathbf{F}_{ij} p_i = 0. \tag{2.1}$$

This constraint holds for each and every pair of corresponding image points $p_i \in \Pi_i$ and $p_j \in \Pi_j$, so the Fundamental matrix is an invariant of the camera setup. Obviously, $p_i^T \mathbf{F}_{ij}^T p_j = 0$ as well, and thus $\mathbf{F}_{ji} = \mathbf{F}_{ij}^T$.

Knowledge of the Fundamental matrix is useful when searching for correspondences of pixels. Given a point $p_i$ in one image, it is only necessary to search for the corresponding pixel $p_j$ on the epipolar line $\mathbf{F}_{ij}p_i$. Thus, the two-dimensional search problem in the image plane $\Pi_j$ is reduced to a one-dimensional search. How far away one has to look is a function of the depth



(a) Extruding the foreground object silhouettes in the source images defines generalized cones in space. Their intersection is the visual hull.

(b) The voxel volume as defined by the visual hull and its bounding box for two different time frames.

**Fig. 2.3.** *Construction of the visual hull and the corresponding voxel volume.*

of $p_i$, i.e. its distance to the image plane. Many depth estimation algorithms are based on this observation.

For certain camera configurations, the Fundamental matrix attains a very special form, which will become important when we deal with rendering from depth maps. This will be detailed in Chapter 12.

## 2.3 The Visual Hull

The visual hull is the best conservative approximation one can get from the silhouettes of an object alone to the geometry of the object itself. Let $\mathcal{O}$ be an object whose boundary is the closed surface $\Sigma$. The projections $S_i := \pi_i(\Sigma)$ into the images are called the silhouettes of the object $\Sigma_t$. Their reprojections $\pi_i^{-1}(S_i)$ are generalized cones in space originating at the centers of projection of the cameras, Fig. 2.3(a). If only the silhouette $S_i$ in an image is known, then it is clear that the object itself has to lie within the reprojection $\pi_i^{-1}(S_i)$. This is true for every one of the cameras, so the *image-based visual hull*

$$\mathcal{V} := \bigcap_{i=0}^{n} \pi_i^{-1}(S_i)$$

is a conservative estimate of the object $\mathcal{O}$ in the sense that it is assured that $\mathcal{O} \subset \mathcal{V}$. In the literature where the concept of the visual hull was introduced, it was defined as the limit of $\mathcal{V}$ when $n$ converges towards the infinite number of possible views. Our image-based visual hull thus has a larger volume than the true visual hull, which in practice, however, only exists as a mathematical concept. Experiments performed by Matusik et al. [78] suggest that the volume

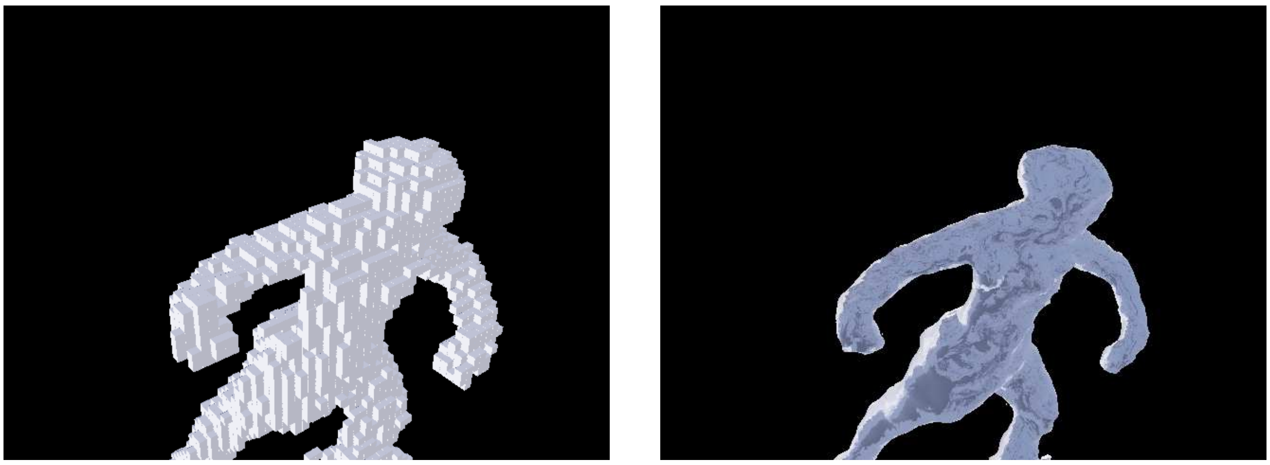


**Fig. 2.4.** *Voxels vs. Level Sets.* A voxel representation of the visual hull on a $32 \times 32 \times 32$ regular grid (left) is compared to the level set reconstruction of the same volume on the same grid (right).

converges quickly when the number of views is increased. When $n > 15$, the volume of the image-based visual hull is typically only 5% larger than that of the true visual hull.

The prerequisite for a succesful computation of the image-based visual hull is a background subtraction. We explain in more detail how we perform this image-processing step in our in-house system later in Chapter 7. In that chapter, we also explain how the image-based visual hull can be computed very efficienly, ideally even in real-time. For that reason, it is an excellent initial estimate to scene geometry, and we make frequent use of it. Together with the real-time rendering back-end from Chapter 13, we can even perform aquisition, 3D reconstruction and free-viewpoint rendering of the visual hull in an online system.

## 2.4 Geometry Models

An important design decision before implementing a reconstruction algorithm is to choose a suitable internal representation for the geometry. A common way to discretize a surface is to employ a triangle mesh. If the mesh encloses a volume, i.e. the surface equals the boundary of a solid object, a volumetric description either as a set of voxels or as a level set might be more appropriate.

Triangle meshes are the most widely used way to describe surfaces in computer graphics. Since graphics hardware is usually optimized for rendering triangles, they can be rapidly displayed without the need for fancy algorithms. Also, the exact location of surface points is immediately accessible, unlike for instance in the case of level sets. On the other hand, triangle meshes require non-trivial data structures for the edges and faces, which define the surface topology. In particular, it is not easy to implement a surface evolution based on meshes due to possible topology changes, which usually require complex local operations involving the creation or deletion of edges, faces and vertices. It is also difficult to avoid cases of mesh degeneration like unwanted self-intersections of the surface.

Since we aim at implementing surface evolutions, we vote for a volumetric representation instead. We express a surface $\Sigma$ as the zero level set of a regular function

$$\begin{aligned} &u : \mathbb{R}^n \to \mathbb{R},\ u^{-1}\{0\} \ = \ \Sigma, \\ &\text{i.e. } u(s) = 0 \ \Leftrightarrow \ s \in \Sigma. \end{aligned} \tag{2.2}$$

We require $u$ to be positive inside the volume enclosed by $\Sigma$, and negative on the outside. The space surrounding the surface is discretized into cells, for which we usually choose a regular grid in order to ease numerical computations. Values of $u$ in locations not on the grid are interpolated linearly.

A small drawback of the level set representation is that it requires a little bit more effort to render the surface. One can either resort to volumetric rendering, or apply an algorithm to extract the zero level-set as a triangle mesh, i.e. Marching Cubes. This disadvantage, however, pales in comparison to the gain, which is that one does not have to take care about topology changes anymore. Instead, the topology of the surface adjusts implicitly as the function values of $u$ evolve according to an evolution equation. Thus, a level set evolution is very easy to implement, although it requires some more computational effort, because new values of $u$ have to be computed in the space surrounding $\Sigma$, which is one dimension higher than $\Sigma$ itself.

Another widely used volumetric representation is to store the volume as a set of voxels, small cubes which make up the whole volume. However, this representation can just be viewed as a special case of a level set, if one defines $u$ on a grid made up of cubic cells, and allows only two values: $u = 1$ if that particular cell is occupied by a voxel, $u = -1$ if the cell is empty. An advantage is that some storage space can be saved, the drawback is that the volume is much coarser, Fig. 2.4.

Both voxels and level sets have in common that the computation of the visual hull is very easy and can be performed efficiently. Suitable algorithms will be presented in Sect. 7.4. The visual hull is important for our work since it forms a starting point for our reconstruction algorithms based on surface evolution.

## 2.5 Color Consistency and Constancy

In order to find matching pixels in different images or over time, almost all 3D reconstruction algorithms to date rely on some form of color consistency and constancy assumptions. We refer to *color consistency* when we mean that a point on an object which is observed in two different views has the same color in both views. Mathematically speaking,

$$\nu_i(p)\nu_j(p) > 0 \implies \|c_i(p) - c_j(p)\| = 0, \tag{2.3}$$

where $c_i(p) := \mathcal{I}_i \circ \pi_i(p)$ is the observed color of a 3D point $p$ in camera image $i$, and $\nu_i(p) \in \{0, 1\}$ denotes whether it is visible or not. Obviously, (2.3) is only valid for materials with special reflection properties. Usually, one has to assume Lambertian reflection only, which is rarely the case in real-world scenes. However, if the matching is robust in all areas without specular reflection, continuity assumptions can enforce correct matching of occasional highlights as well. But all in all, larger areas of non-Lambertian materials remain a challenging problem for 3D reconstruction.

*Color constancy* refers to the assumption that the same point on a surface which is observed over time leads to the same color in the image. This is obviously an idealization as well, which assumes that the lighting stays constant, and no other objects cast shadows on the observed point. For short video sequences or scenes with artificial lighting, the former requirement is in general at least approximately satisfied. The latter, however, can never be guaranteed, so that we can only safely assume that the color remains constant over certain connected parts of the trajectory, with occasional discontinuities.

More precisely, the trajectories of surface points are the integral curves $(p_t)$ of the ordinary differential equation given by the scene flow $\mathbf{v}_t$:

$$\frac{\partial p}{\partial t} = \mathbf{v}_t.$$

Thus, color constancy is equivalent to the requirement that

$$0 = \frac{\partial}{\partial t} c_i(p_t) = \frac{\partial \mathcal{I}_i^t}{\partial t} \left( D\pi_i \cdot \mathbf{v}_t \right)$$

almost everywhere on $\mathfrak{H}$.

In the discussion above, we neglect other problems with color. Sampling errors are not discussed, and we discarded the fact that different cameras might respond differently to the same input. Careful color calibration before or after a recording is mandatory. This, however, is also beyond the scope of this work, and we will from now on regard our image data as being perfect. Still, we keep in mind that our matching has to be robust not only with respect to camera calibration, but with respect to certain errors in color as well.

# 3

# Related Work

## 3.1 Overview

This work focusses on the 3D reconstruction part of the free-viewpoint video pipeline, so naturally we devote most of the discussion of related work to this topic. In the following three sections, we give an overview and classification of existing 3D reconstruction algorithms, focussing on the two main topics of depth reconstruction and surface reconstruction. Afterwards, we present previous systems for video-based rendering in Sect. 3.5, where authors address at least several parts, if not the complete pipeline. There, we also cover previous work on free-viewpoint and dynamic light field rendering techniques, since those algorithms are usually tied to a specific acquisition system and geometry representation.

## 3.2 Classification of 3D Reconstruction Techniques

At the heart of all computer vision research lies the analysis and interpretation of visual information. We can only envy the ease with which the human visual system performs these tasks, and indeed researchers base many algorithms on cues which give the ability to perceive depth to us as well. Some examples of these depth cues include binocular parallax, movement parallax, accomodation, convergence, shades and shadows, linear perspective, an many more [73].

While each of these cues is exploited by existing computer vision algorithms, we want to focus on the ones which are based on color consistency, as introduced in Sect. 2.5. A special incarnation is binocular parallax. From corresponding points in the image of the left and right eye, i.e. two retinal points

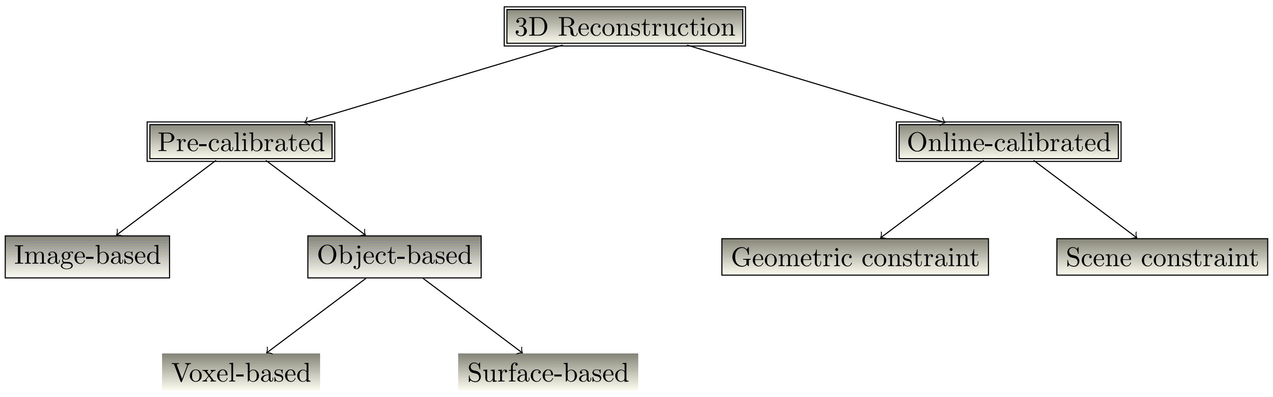


**Fig. 3.1.** Our classification scheme for 3D reconstruction techniques. It is based upon [70], but voxel models are merged into the branch of object-based methods, while this branch exhibits a further distinction into the different geometry representations.

which are the projection of the same 3D point, it is possible to infer the depth of the point by triangulation, see Sect. 2.2. This triangulation procedure is vital to the *image-based* algorithms for depth reconstruction we are going to discuss. They differ mostly in the way how correspondences are found, and how depth estimates for different pixels in different views are integrated into a single geometric representation. On the other hand, *object-based methods* are centered around the objects to be reconstructed. They try to find an optimal geometry which optimally fits all images simultaneously, usually optimizing a photo-consistency as well as a continuity constraint. Earlier approaches rely on silhouette information alone.

In the classification scheme of Zhang et al. [70], image-based and object-based methods belong to the category of pre-calibrated methods, which assume a prior calibration of the camera. The other major category is made up of online-calibrated methods. We decided to partly adopt their classification, but make a further distinction in the branch of object-based reconstruction, which we subdivide according to the representation of the geometry, Fig. 3.1. We discuss pre-calibrated methods in the next section, and review online-calibrated methods in Sect. 3.4.

## 3.3 Pre-calibrated Methods

All of our methods we present in this thesis are pre-calibrated. Prior calibration of the cameras is vital if one wishes to reconstruct a scene as accurately as possible. Since we record videos in a studio environment, pre-calibration is possible and beneficial. As stated before, we distinguish image-based and object-based methods.

### 3.3.1 Image-based methods

In image-based methods, one computes reconstructions from either sparsely matched image-features, or dense stereo correspondences. They work best if the cameras are densely packed and have near-parallel optical axes. We presend an image-based depth reconstruction method in Part II.

During our review of previous methods of image-based depth reconstruction, we follow the classification of Scharstein and Szeliski [108]. Their taxonomy is well known and widely used in computer vision, and in their work they also present an evaluation and comparison of the different algorithms. We classify new methods developed after publication of their report according to their scheme as well.

They observe that stereo algorithms generally perform a subset of four different tasks:

1. Matching cost computation
2. Cost support aggregation
3. Disparity computation and optimization
4. Disparity refinement

The sequence of steps as well as the exact method employed in each step is what defines each individual algorithm. We give an overview of strategies for the four steps in the next sections, and review methods which use them. A classification of existing algorithms according to their building blocks is given in [108].

**1. Matching Cost Computation**. Traditional pixel-based matching cost measures are squared intensity differences (SD), e.g. [47, 118], and absolute intensity differences (AD), e.g. [56]. In video processing, those are well-known as mean-squared error MSE and mean absolute difference MAD. The normalized cross-correlation measure is also ancient [47, 105], and performs similarly to SD.

More recent measures limit the influence of mismatches during aggregation, for instance truncated quadrics or contaminated Gaussians [10, 107]. Another class of interesting matching costs has the advantage of being insensitive to camera gain and bias, among them gradient-based measures [112] and rank or census transforms [144]. It can also be feasible to get rid of color calibration problems using a pre-processing step for bias-gain or histogram equalization [29].

More complex techniques compare phase or the response of filter banks [75, 54], or are insensitive to image-sampling [9]. The latter is achieved by not only comparing shifts of pixels by integer amounts, but also comparing the intensity of reference pixels against linearly interpolated intensity at non-integer positions in another image.

**2. Cost Support Aggregation**. The matching cost can be integrated by summing or averaging over a local support region. In a first class of algorithms, the support region is rectangular, and thus favoring fronto-parallel surfaces. There, evidence aggregation is implemented using square windows or Gaussian convolution, shiftable windows [12], and windows with adaptive sizes [87, 58]. In this case, the aggregation can be implemented efficiently using box filters.

The second class of methods uses a more general region instead of a rectangular window, and supports slanted surfaces. Proposed aggregation functions include limited disparity difference [45] and Pradzny's coherence principle [98].

A different aggregation method is iterative diffusion, where the averaging operation is implemented by repeatedly adding to each pixel's cost the weigthed costs of its neighbours [107].

**3. Disparity Computation and Optimization**. We distinguish *local methods* and methods which rely on *global optimization*. Computing final disparities in local methods is trivial, because all the work has already been done in the cost computation and aggregation steps. The final disparity chosen is just the one with the lowest cost, a scheme called "winner-take-all" optimization (WTA). Unfortunately, this scheme is difficult to extend to the case of multiple cameras, because a uniqueness of matches can only be enforced for one image.

Global methods perform almost all of their work in the optimization phase, often skipping the aggregation step. A popular formulation is an energy-minimization framework, where the objective is to find a disparity function $\lambda$ minimizing a global functional $E(\lambda)$. Our method, which we present in the next part, is a global optimization method, aggregating normalized cross-correlations over rectangular windows, and using graph cuts as an energy optimization technique. The use of graph cuts for this task has been proposed recently, and relies on the special form of the energy functional for disparity assignments [62, 61, 104, 13, 52].

The energy functional usually contains a smoothness term in order to enforce a regularization propery, which is a monotonically increasing function of disparity difference. [36] give a Bayesian interpretation of the relevant energy functionals, while [126] propose a discontinuity-preserving energy function based on Markov-Random-Fields and additional so-called line processes. Line processes in turn can be subsumed by a robust regularization framework [10]. The smoothness-enforcing term in the energy functional can be made to depend on intensity gradients, lowering the penalties for non-smoothness at likely edges [12, 13]. We also employ this idea in our method, and judge from experiments that it has a vital impact on the quality of the results.

Other than with graph cuts, one can also employ various other methods in order to find a minimum to the energy. Traditional methods include regularization and Markov Random Fields [11], simulated annealing [36, 4], highest confidence first [24], and mean-field annealing [35]. A different class of optimization techniques uses dynamic programming. Those techniques are based on the observation that a minimum for an independent scanline can be found in polynomial time, while the minimization problem for the total dense disparity map is NP-hard. The common idea is to compute the minimum-cost path through the matrix of all pairwise matching costs between two corresponding scanlines [6, 7]. Finally, cooperative algorithms, while performing local computations, use non-linear operations yielding an overall behaviour similar to a global optimization algorithm [107]. They are inspired by computational models of human vision, and among the earliest methods for disparity computation [74, 128].

A special class of global optimization methods rely on partial differential equations in order to compute the minimum of a non-discrete, continuous matching functional [3, 33, 101]. In various ways, they are more closely related to our surface reconstruction techniques presented Part III of the thesis. In particular, what they do can be thought of as fitting a deformable surface to the input images.

**4. Disparity Refinement**. With the exception of the PDE-based methods, all algorithms above compute the disparity assignments in a discrete disparity space. For image-based rendering, this is not a desireable feature, since the scene appears to be made up of several layers. To remedy this situation, a sub-pixel refinement stage can be applied after the initial discrete correspondence stage. To achieve this, several methods have been proposed. One possibility is iterative gradient descent, where a curve is fitted to the matching costs at discrete disparity levels [105, 55], requiring only minimal additional computation cost.

Other ways of post-processing the disparities include cross-checking of maps computed for several cameras [27, 34]. Outliers can be removed using median-filters, and holes in the disparity map due to occlusions can be filled by surface fitting or distributing neighbouring disparity estimates [7, 106].

**Background Separation**. Besides depth reconstruction, we address a second problem in Part II: separation of the foreground of a scene from a known background, which is another important prerequisite for several interesting vision algorithms. In particular, the computation of the visual hull relies entirely on object silhouette information [65, 78]. The kind of separation we have in mind is most closely related to video matting techniques, several of which are widely used. The *blue screen method* and *multi-background matting* rely on backgrounds with special mathematical properties and require a tightly controlled studio environment to be succesful [80, 120]. A common method in production is *rotoscoping*, where the user is required to draw curves around the foreground elements himself, assisted by several tools like automatic adherence of the curves to image contours, or the tracking of curves over time [25].

Our approach falls into a third category which uses *clean plates*, images of the static background of the scene. Methods to date obtain the foreground by subtracting or separating the known background from the current frame. Opacities are assigned to color differences at each pixel via some user-defined mapping [60]. It depends on statistically derived threshold values, and fails in regions where the foreground is similar in color to the background, mostly because it does not take into account spatial coherence. Making use of the spatio-temporal context to improve the segmentation was also investigated in [80, 94]. However, it has not yet been intertwined with simultaneous depth estimation.

## 3.3.2 Object-based algorithms

Object-based algorithms focus on reconstructing objects directly, without explicitly computing correspondences. Instead, they are based on photo-consistency and continuity assumptions, and try to find geometry which optimally satisfies the constraints. Older methods are based on silhouette information alone. We further classify object-based algorithms according to the internal representation they adopt for the geometry. Voxel-based methods recover a discrete, volumetric model, marking each cell in a reconstruction volume as either occupied or empty. In some cases, a color is associated to each cell as well. In contrast, surface-based methods recover a surface model directly, for instance in the form of a triangle mesh or as a level set. The transition between the different kinds of representations is smooth. As already pointed out in Sect. 2.4, voxel models are essentially equivalent to special level set representations. In turn, level sets and triangle meshes can be converted into each other at will, using marching cubes or signed distance transforms, respectively. The reconstruction algorithms, however, are usually tuned for a specific kind of representation.

### Voxel-based methods

Due to the relative ease with which they can be handled, voxel models of the scene have been quite popular for some time. In particular, the visual hull can be easily and quickly be computed as an approximate conservative model of scene geometry [65, 76], and has been widely used as a geometry proxy for image-based rendering. Improved approaches use an octree representation for the visual hull instead of voxels [97]. In voxel occupancy [122], the question of whether a voxel is occupied or empty is reformulated as a labeling problem, which is then solved via graph cuts. An additional smoothing term is provided as well.

While the visual hull relies on silhouette information alone, much refined voxel models can be obtained when color consistency is taken into account. In space carving or voxel coloring approaches [114, 64], one starts with a completely solid voxel model, and iteratively carves away voxels of bad photo-consistency.

### Surface-based methods

Surfaces can be represented in a variety of ways. Some researchers compute the polygonal visual hull instead of a voxel model [68, 77]. A triangle mesh optimizing photo-consistency for a number of views simultaneously using a hierarchic, multi-resolution approach is computed in Heidrich et al. [63].

If one employs a deformable surface model of the geometry, variational approaches are quite succesful. Here, the optimal surface is defined as the one minimizing a certain error functional, which is given as an integral over a surface. This functional is designed to enforce certain desireable properties like photo-consistency. We review these methods in more detail in Sect. 7.3, when we introduce our own surface-based reconstruction algorithm. The surface evolution PDE arising from variational approaches can be implemented using different geometry representations, among them level sets, triangle meshes and, as shown recently by one in our group [69], point based models.

## 3.4 Online-calibrated Methods

We finally give a brief overview of online-calibrated 3D reconstruction techniques, which we do not pursue further in this work. Those techniques are required for certain recording scenarios where we do not have access to the camera anymore, much less to its position in order to obtain an accurate calibration. This is always the case if one wants to reconstruct geometry from some previously recorded video sequence.

Online-calibrated algorithms are further distinguished according to how the calibration of the camera is computed. At present, two classes are most important. Algorithms which fall into the class *scene constraint* take advantage of special scene structures such as parallel lines within the scene. They compute vanishing points in the principle directions in order to determine the intrinsic and extrinsic calibration parameters. Beforehand, a fundamental matrix is estimated and a projective reconstruction is obtained by solving a triangulation problem. This can be performed for all feature correspondences and all views at once [132, 125]. Afterwards, closed form solutions for the intrinsic parameters of the cameras can be obtained as a function of the vanishing points [14]. Vanishing points can be estimated via an accumulation of detected parallel lines, followed by a search step [71, 102].

In contrast, algorithms belonging to the *geometric constraint* class do not assume prior knowledge about scene structure. Camera parameters are estimated using the projected image of the absolute conic, an abstract object known from projective geometry [133]. An earlier approach [81, 48] relies on Kruppa's equations, which has the advantage of not requiring an additional projective reconstruction, but has limitations in form of ambiguities in the case of specific camera motions [124]. Finally, stratified self-calibration upgrades from a projective to a metric reconstruction by estimating the infinite homography in a first step, yielding an affine reconstruction, i.e. via the modulus constraint [96]. In a second step, the calibration matrix is estimated using a linear algorithm. Compared to estimating the absolute conic directly, this method has the advantage of not having to compute all parameters at the same time, increasing numerical stability.

Unfortunately, self calibration and metric reconstruction is not possible for all video sequences. Degenerate camera configurations and motion sequences exist for which the process is ambigous. A complete catalog of critical motion sequences is given in [123].

## 3.5 3D Video Systems

The development of 3D Video and 3D-TV follows advances recently made in image-based rendering (IBR). In IBR, conventional photographs are used to capture the visual appearance, the *light field* of a scene. Given sufficiently many images from different viewpoints, any view of the scene from outside of the convex hull of the recording positions can be reconstructed [66]. Unfortunately, light field rendering quality depends on the number of photographs. Very large numbers of images are necessary to attain convincing rendering results [20]. Camera configuration constraints, however, can be relaxed by adding more and more geometry to image-based systems, as demonstrated by Lumigraphs [44]. Thus, a way to reduce the required number of images is to employ computer vision algorithms to reconstruct 3D scene structure. Hybrid model/image-based rendering methods based on the visual hull [78], per-image depth maps [110] or even a complete 3D scene geometry model [139] achieve realistic rendering results from only a relatively small number of images. Furthermore, programmable graphics hardware can be used to accelerate image-based rendering by warping and resampling the recorded images [119, 109]. Appropriate representations for coding 3D audio/visual data are currently investigated by the MPEG-4 committee [121]. The MPEG-4 multiple auxiliary components can encode depth maps and disparity information. But these are not complete 3D representations and possible shortcomings and artifacts due to DCT encoding and unrelated texture motion fields and depth or disparity motion fields still need to be investigated.

We roughly classify the multitude of existing systems according to the kind of geometry they are based upon. For almost every reconstruction method and geometric representation covered in Sect. 3.3, there exists a corresponding 3D video system. Note that a strict distinction is not always possible, since some systems work with several layers of different representations [28]. Purely image-based methods, which do not require any geometry at all, as well as depth image-based representations are covered first in the next subsection. These are followed by methods which recover a voxel model in subsection 3.5.2, mesh-based algorithms in subsection 3.5.3, and finally point-based and surfel models in subsection 3.5.4.

### 3.5.1 Image- and Depth Image-based systems

Purely image-based representations [66] need many densely spaced cameras for applications like 3D-TV [79]. Dynamic light field cameras [137, 143] which have camera baselines of a couple of centimetres do not need any geometry at all. Instead, novel views of sufficient quality can be generated by plain image warping and blending only.

Depth image-based representations [5, 116], on the other hand, employ depth maps in order to extrapolate novel views from the source images. Predominantly, those depth maps are computed by stereo algorithms [147, 37, 143]. All stereo systems still require the camera baselines to be reasonably small. Hence, scalability and flexibility in terms of camera configurations can not be achieved.

Pollard and Hayes [95] utilize depth map representations for novel view synthesis by morphing live video streams. This representation can suffer from inconsistencies between different views. Mulligan and Daniilidis [84] target telepresence. They compute geometric models from trinocular stereo depth maps in a multi-camera framework from overlapping triplets of cameras, and transmit texture and depth over a network.

A layered depth image representation for high-quality video view interpolation has been proposed Zitnick et al. [147]. In their approach, reconstruction errors at depth discontinuities are smoothed out by Bayesian matting. Again, this approach requires a quite dense camera setup to generate high-quality renderings in a limited view range. Scalability to larger setups is not addressed by the authors.

Good performance can be achieved by assisting the stereo reconstruction with specialized hardware. Cockshott et al. [28] propose a 3D video studio based on modular acquisition units and pattern-assisted stereo. For concurrent texture acquisition, the patterns are projected using strobe lights requiring custom-built hardware. Foreground objects are then modeled using implicit surfaces.

Recently, special-purpose hardware solutions for real-time depth estimation from video images have become available. 3DV Systems' ZCamTM[1], and Tyzx's DeapSea chips[2] solve the problem of real-time depth reconstruction and can be incorporated into most existing 3D video frameworks, including ours. ZCams are already being employed by Redert et al. [100] for 3D video broadcast applications.

### 3.5.2 Voxel-based systems

The methods described up until now did not estimate a full 3D model of scene geometry. In contrast, the following representations start to describe objects or the scene using time-varying 3D geometry, possibly with additional video textures. All can work with almost arbitrary camera configurations. However, since most existing systems reconstruct objects from their silhouettes, they are restricted to foreground objects only.

Frequently, voxel representations are derived by volume- or space carving methods [127]. Vedula et al. [135] achieve temporal coherence by introducing a 6D model which includes motion information about each voxel, and additionally carving away voxels inconsistent with the estimated scene flow. The 3D video recorder presented by Würmlin et al. [141] stores a spatio-temporal representation in which users can freely navigate.

Matusik et al. [77, 78] focus on real-time applications like 3D video conferencing or instant 3D replays. In [78], they present an image-based 3D acquisition system which computes the visual hull of an object in real time. It is build on epipolar geometry and outputs a view-dependent layered depth image representation. Their system neither exploits spatio-temporal coherence, nor is it scalable in the number of cameras.

[1] `http://www.3dvsystems.com`
[2] `http://www.tyzx.com`

### 3.5.3 Mesh-based systems

Similarly to their voxel-based representation, Matusik et al. also base polyhedral visual hulls [77] on epipolar geometry, and provide view-independent rendering through a mesh and texture representation. This approach shares the same limitations and furthermore introduces interpolation artifacts due to improper alignment of geometry and texture, a common drawback of mesh-based methods.

The photo hull, defined as the largest shape photo-consistent with the given images, is employed as a geometry proxy for their real-time system by Li et al. [67]. Their hardware-accelerated rendering algorithm reconstructs the photo hull implicitly during rendering.

Carranza et al. [16] present an offline 3D video system which employs an a-priori shape model which is adapted to the observed outline of a human. While this system is only able to capture pre-defined shapes, Theobalt et al. extend it to estimate the underlying skeleton in a previous step [129].

Kanade et al. [57] and Narayanan et al. [86] fuse a collection of range/intensity image pairs into a triangular texture-mapped mesh representation.

### 3.5.4 Point- or Surfel-based systems

A dynamic surfel sampling representation and algorithm for estimation of dynamic appearance and 3D motion is presented by Carceroni and Kutulakos [15]. In their work, a volumetric reconstruction is provided for a small working volume.

The 3D video recorder [141] handles point-sampled 3D video data captured by silhouette-based reconstruction algorithms and discusses data storage issues. Gross et al. [46] developed a 3D video system based on a point sample representation [140] for their telecollaboration system *blue-c*. No full scene acquisition is possible with the last two systems, but almost arbitrary camera configurations are possible. In a recent paper [136], however, they extend their work to a scalable 3D video framework for capturing and rendering dynamic scenes, which is not restricted to foreground objects anymore. Assisted by the projection of structured patterns, they obtain depth maps via space-time stereo, from which they construct a point-based model of the geometry.

# Part II

# Depth Reconstruction

# 4

# Overview

## 4.1 Depth Reconstruction

This part of the thesis deals with the reconstruction of dense disparity maps from multiple video streams. A key requirement is a suitable camera setup. Because local error measures are not robust against scene appearance variations recorded in wide-baseline camera setups, the proposed methods work best if the cameras are densely packed for instance in one or multiple arrays. Furthermore, all cameras should have parallel or at least near-parallel optical axes.

Our goal, as always, is to perform scene reconstruction in order to be able to render high-quality novel views of the scene. With this kind of camera setups, the novel viewpoints are allowed to lie within the convex hull of the recording camera's positions. Suitable rendering techniques are explored later in Chapter 12. In the following chapter, we discuss our basic algorithm of choice, which is a depth reconstruction technique based on the succesful graph cut method. We first extended it to perform a simultaneous segmentation into static background and moving foreground, an important requirement for crisp edges and thus realistic looking novel views. The final Chapter 6 of Part II further extends the basic technique in that it employs a pre-processing stage to improve local matching via a statistical analysis of the error distribution. Furthermore, temporal coherence of the depth maps is enforced, increasing the quality during playback by eliminating flickering artifacts due to temporal depth discontinuities.

All algorithms presented in the following are tested with data obtained with the Stanford Multi-Camera Array, which is described in Sect. 4.2, concluding this introductory chapter.

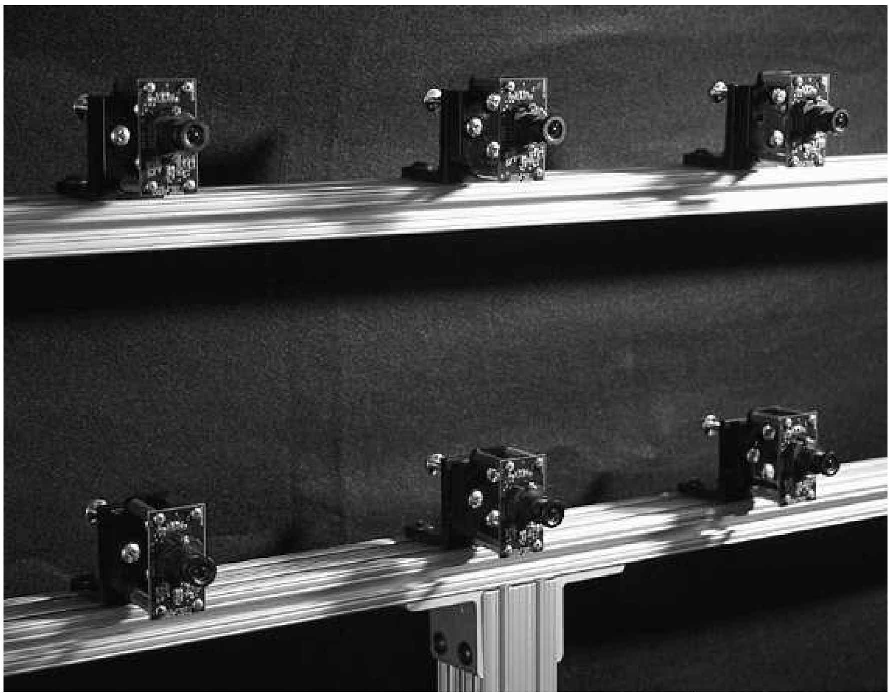


**Fig. 4.1.** The Stanford Multi-Camera Array consists of numerous CMOS camera heads, arranged in a planar matrix and with aligned optical axes.

## 4.2 Acquisition Hardware

To record the light field of a static scene, it is customary practice to use one single still-image camera and consecutively record multiple views by moving the camera around the scene. For acquiring the temporally varying light field of a dynamic scene, however, numerous video cameras are needed to record the scene from multiple viewpoints simultaneously. In addition, all video cameras must be synchronized to maintain temporal coherence among the recorded images.

Because of the high hardware requirements, only a few laboratories are currently able to record dense dynamic light fields [78, 89, 138]. The dynamic light field data used in this work has been captured with the Stanford Multi-camera Array (SMCA) [138] which is currently being built at Stanford University as part of the Immersive Television Project, Fig. 4.1. The SMCA consists of multiple low-cost CMOS imagers, each providing $640 \times 480$-pixel RGB resolution at 30 frames per second. The camera heads are aligned in parallel, capturing the scene's light field in the two-plane parameterization [66]. Custom-built driver boards enable on-line pre-processing as well as MPEG-2 compression of each video stream. At 5 MBytes/sec per camera, up to 60 MPEG-encoded video streams can be streamed to one PC via the IEEE1394 High Performance Serial Interface Bus where the data is stored on a SCSI hard drive.

The use of more than one camera inevitably leads to mismatches in hue, brightness and radial distortion among different camera images. These differences need to be minimized by careful calibration prior to further processing. In addition, due to the design of the Light Field Video Camera, only MPEG-compressed image data is available, causing quantization noise and blocking artifacts in the images. The depth estimation algorithm described in the following chapter must be robust against these artifacts.

# 5

# Joint Depth and Background Estimation

## 5.1 Introduction

In the remainder of this part, we consider a scenario where we have multiple views available from cameras which are packed in an array with parallel optical axes. If one wants to interpolate a novel view in between cameras, it is certainly not sufficient to just blend the two images together. Instead, one has to warp each pixel into the novel view along the projected epipolar line. Exactly by how much depends on the pixel's disparity, which is related to its depth in the scene, Fig. 12.1.

Thus, a pre-requisite for free-viewpoint video is to compute dense disparity maps for each view. Another desireable feature is a subtraction of the static background from the moving foreground of the scene. During rendering, this enables warping the foreground independently of the background, and preserves crisp contour curves along the foreground by avoiding blending with background pixels. The visual quality is greatly enhanced.

Clearly, 3D-reconstruction as well as background separation benefit greatly from a known solution to the respective other problem: If the static background pixels in an image are known, then these pixels must have the same depth as the background, while all other pixels must be less deep. On the other hand, if we know the depth of each pixel, then only pixels with a lesser depth than the background can belong to the foreground.

In the following sections we describe an algorithm which exploits this interdependency by addressing both problems simultaneously, assuming that we have a set of fully calibrated cameras and an image of the static background for each camera with at least approximate per-pixel depth information. Our method is a generalization of the successful multi-view reconstruction algorithm by Kolmogorov and Zabih [61]. Pixels are not only labeled by their depth, but also by an additional flag which indicates whether a pixel belongs to the background or not. As in [61], the result of our depth reconstruction and background separation algorithm is obtained as the minimum of an energy functional. Besides taking into account classic contraints from multi-view stereo, it regards also the new considerations related to background as well.

Sect. 5.2 outlines the problem we want to solve precisely and introduces the notation which is used throughout the rest of the chapter. The energy functional we minimize is defined in Sect. 5.3, while Sect. 5.4 is devoted to the method of graph cuts, which is used to perform this minimization. There, we also give proof that this method is applicable to our energy functional. Results we achieve by applying our algorithm to real-world as well as synthetic data are demonstrated in Sect. 5.5. Finally we conclude with a summary and some ideas for future work in Sect. 5.6.

## 5.2 Reconstruction Algorithm

We aim at reconstructing the 3D-geometry of a static scene captured by a number of calibrated cameras directly from the images. The goal is to retrieve depth maps, assigning a depth value to each pixel which defines its location in 3D-space. Simultaneously, we want to decide for every pixel whether it belongs to the background of the scene, known from *background images* captured with the same cameras. We assume that the depth of each pixel in the background images can be estimated at least approximately. Pixels belonging to objects present in the current image but not in the background image shall be tagged as foreground.

Our algorithm is a true generalization of the *multi-camera scene reconstruction via graph cuts* described in [61]. It shares all of its advantages:

- All input images are treated symmetrically,
- Visibility is handled properly,
- Spatial smoothness is imposed while discontinuity is preserved.

While our energy functional is different, we utilize a similar problem formulation and notation, which we introduce now.

**Input:** The input to the algorithm is the set of pixels $\mathcal{P}_k$ from each source camera $k$ together with the following mappings for every pixel $p \in \mathcal{P} := \bigcup_k \mathcal{P}_k$:

| | |
|---|---|
| $\mathcal{I}(p)$ | The color value of the input image. |
| $\Delta\mathcal{I}(p)$ | The value of the (discretely evaluated) Laplacian of the input image. |
| $\mathcal{B}(p)$ | The color value of the background image. |

**Output:** The goal is to find the "best" mapping $\lambda : \mathcal{P} \to \mathfrak{L}$ into a set of *labels* $\mathfrak{L}$. The precise definition of "best" is given later. To each pixel is assigned a label $\mathfrak{l} = (\mathfrak{l}_d, \mathfrak{l}_b)$, which is a *pair* of values. This is our first generalization: Labels not only encode depth, but also the new property of "backgroundness". The boolean value $\mathfrak{l}_b$ is true if and only if $p$ is a background pixel, while $\mathfrak{l}_d$ denotes the *depth* of $p$.

As is done in the original algorithm [61], the notion of "depth" we use is a somewhat abstract one: Depth labels correspond to level sets of a function $D : \mathbb{R}^3 \longrightarrow \mathbb{R}$ satisfying

- For all scene points $P, Q \in \mathbb{R}^3$ and all cameras $k$:

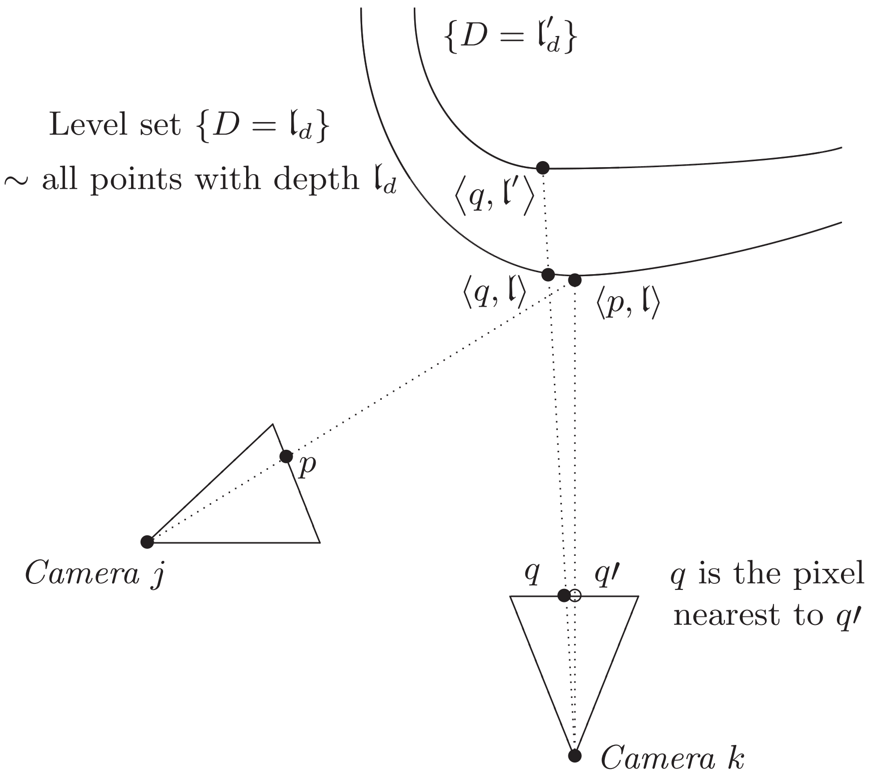
$$P \text{ occludes } Q \text{ in } k \;\Rightarrow\; D(P) < D(Q).$$

This is obviously a very natural requirement for a function indicating depth. The existence of such a function $D$ implies that there is a way to define depth *globally*, i.e. independent of a specific camera. The same constraint is postulated in the original algorithm [61] as well as in voxel coloring [113]. An important special case in which the constraint is automatically satisfied occurs when all cameras are located on one side of a plane $\Pi$ looking at the other side. The level sets of $D$ can then be chosen as planes which lie parallel to $\Pi$.

**Topology:** The definition of the algorithm includes the topological properties of the input images. A set-theoretic description is given by assigning to every $p \in \mathcal{P}$ the following sets of pixels:

| | |
|---|---|
| $\mathcal{N}_p$ | A set of neighbors of $p$ in $\mathcal{P}_k$ *excluding* $p$ where the energy functional will encourage continuity. |
| $\mathcal{C}_p$ | A neighborhood of $p$ *including* $p$. These regions will later be relevant for the computation of normalized cross correlations which are used as a criterion for photo-consistency. |

**Geometry:** Finally, the geometric relations between pixels in different images with regard to their current labels and the camera positions must be specified. We encode these in the set $\mathfrak{I}$ of *interactions*. First note that a pixel $p$ together with a label $\mathfrak{l}$ corresponds to a point in 3D-space via the projection parameters of the camera. This point is denoted by $\langle p, \mathfrak{l} \rangle$. The interactions now represent a notion of "nearness" of two 3D-points in the following sense, Fig. 5.1:



$\{D = \mathfrak{l}'_d\}$
Level set $\{D = \mathfrak{l}_d\}$
$\sim$ all points with depth $\mathfrak{l}_d$
$\langle q, \mathfrak{l}' \rangle$
$\langle q, \mathfrak{l} \rangle$
$\langle p, \mathfrak{l} \rangle$
$p$
Camera $j$
$q$
$q\prime$
$q$ is the pixel nearest to $q\prime$
Camera $k$

**Fig. 5.1.** *Interactions and Occlusions.* The 3D-points $\langle p, \mathfrak{l} \rangle$ and $\langle q, \mathfrak{l} \rangle$ interact, thus $\{\langle p, \mathfrak{l} \rangle, \langle q, \mathfrak{l} \rangle\} \in \mathfrak{I}$. On the other hand, $\langle q, \mathfrak{l}' \rangle$ is occluded by $\langle p, \mathfrak{l} \rangle$ in its camera image, so $\{\langle p, \mathfrak{l} \rangle, \langle q, \mathfrak{l}' \rangle\} \in \mathfrak{O}$.

- A pair $\{\langle p, \mathfrak{l} \rangle, \langle q, \mathfrak{l} \rangle\}$ belongs to $\mathfrak{I}$ if and only if
  1. $q \in \mathcal{P}_k$ and $p \notin \mathcal{P}_k$, i.e. $p$ and $q$ must come from two different cameras.
  2. $q$ is the pixel nearest to the projection of $\langle p, \mathfrak{l} \rangle$ onto the image of camera $k$.

  Note that interacting pixels always have the same label. In particular, foreground can only interact with foreground, background only with background, and both pixels must belong to the same level set of $D$.

The set $\mathfrak{O}$ of *occlusions* will be used to enforce visibility constraints. It also contains pairs of 3D-points and is defined as follows:

- A pair $\{\langle p, \mathfrak{l} \rangle, \langle q, \mathfrak{l}' \rangle\}$ belongs to $\mathfrak{O}$ if and only if $\{\langle p, \mathfrak{l} \rangle, \langle q, \mathfrak{l} \rangle\} \in \mathfrak{I}$ and $\mathfrak{l}_d < \mathfrak{l}'_d$. Geometrically this means that if $\langle p, \mathfrak{l} \rangle$ is projected onto $q$, then it will occlude $q$ if and only if the depth assigned to $p$ is smaller than the depth assigned to $q$.

**Energy minimization:** As stated before, the algorithm tries to find the "best" labelling $\lambda$ of pixels. Mathematically speaking, the best configuration corresponds to the one that minimizes an *energy functional* $E(\lambda)$. This functional encodes the high level knowledge about scene reconstruction: Unlikely or impossible assignments of labels must be penalized, while very likely configurations must be enforced. A precise definition of the energy functional we use is given in the next section.

## 5.3 The Energy Functional

The energy functional which is minimized by the algorithm can be written as a sum of contributions by every single pixel and every possible pair of pixels:

$$\begin{aligned} E(\lambda) \;=\; & \sum_{p,q\in\mathcal{P}} \left[ E^{p,q}_{\text{photo}}(\lambda) + \beta E^{p,q}_{\text{smooth}}(\lambda) + E^{p,q}_{\text{vis}}(\lambda) \right] \\ & + \alpha \sum_{p\in\mathcal{P}} E^{p}_{\text{background}}(\lambda). \end{aligned}$$

The terms on the right hand side will be different from zero only if $p$ and $q$ interact or occlude each other in certain configurations, or if $p$ and $q$ are neighbours. Thus, the sum runs in effect only over relatively few pairs of points, which is of course very important for fast performance of the algorithm. The positive weights $\alpha$ and $\beta$ are the only free parameters of our method. Good choices will be specified in Sect. 5.5. The goal of the graph cut algorithm in Sect. 5.4 is to find an assignment $\lambda$ of labels to all pixels that is a local minimum of $E$ in a strong sense. We now give a detailed description of the four contributing terms.

### 5.3.1 Photo-consistency term

For interacting pixels sharing similar characteristics, we issue a photo-consistency bonus. This reflects the fact that if a 3D-point is projected onto a pixel $p$ in one image and a pixel $q$ in another and is visible in both images, then pixels in the neighbourhoods $\mathcal{C}_p$ and $\mathcal{C}_q$ should be similar. Mathematically, we set

$$E^{p,q}_{\text{photo}}(\lambda) := \begin{cases} -C(p,q) & \text{if } \{\langle p, \lambda(p)\rangle, \langle q, \lambda(q)\rangle\} \in \mathfrak{I}, \\ 0 & \text{otherwise.} \end{cases}$$

The *correlation term* $C(p,q) \in [0,1]$ must be small if $\mathcal{C}_p$ differs from $\mathcal{C}_q$ and large if the local pixel neighbourhoods are very similar. We found experimentally that a very good criterion is the statistical measure obtained by computing

- The normalized cross-correlation[1] between the sets of color values $\mathcal{I}(\mathcal{C}_p)$ and $\mathcal{I}(\mathcal{C}_q)$, taking the minimal correlation among the three color channels, and

[1] Cross-correlations in our sense are always positive numbers. If the result from the computation is negative, it is set to zero.

- The normalized cross-correlation between the sets of Laplacians $\Delta\mathcal{I}\,(\mathcal{C}_p)$ and $\Delta\mathcal{I}\,(\mathcal{C}_q)$, again computing the three color channels separately and taking the minimum.

A weighted average of these two values is then assigned to $C(p,q)$. In both cases the neighborhoods we use are square $3 \times 3$ pixel windows surrounding the points.

Indeed, this scheme has theoretical advantages as well. Especially in real-world data, correlations are much more robust than some kind of distance measure between the color values: Stereo images taken simultaneously by different cameras often have significantly different color values even for corresponding pixels, because the response of the cameras to the same signal is not identical. This effect can be somewhat reduced by careful calibration, but it remains a principal problem. Since correlation measures statistical similarity, not absolute similarity in values, it yields more reliable results even with uncalibrated images. This is especially true for neighbourhoods containing edges, which are generally more easily matched.

To further encourage that image features like edges and corners are matched with their counterparts in other images, we include the correlation of the Laplacian of the image into $C(p,q)$. Small additional improvements in quality can also be achieved by matching other characteristics like partial derivatives or even the coefficients of local Fourier expansions, see related work on local matching in Sect. 3.3.1. Possible benefits, however, are found to be very small when compared to the increase in computational cost.

### 5.3.2 Smoothness term

Drastic changes in depth or transitions from background to foreground are usually accompanied by image features. We transfer this simple observation into the smoothness energy

$$\begin{aligned} E^{p,q}_{\text{smooth}}(\lambda) &:= V^{p,q}\big(\lambda(p),\lambda(q)\big), \\ \text{where } V^{p,q}(\mathfrak{l},\mathfrak{l}') &:= \begin{cases} 0 & \text{if } q \notin \mathcal{N}_p \text{ or } \mathfrak{l}=\mathfrak{l}', \\ 2L_{\max} - \|\Delta\mathcal{I}\,(p)\|_\infty - \|\Delta\mathcal{I}\,(q)\|_\infty & \text{otherwise.} \end{cases} \end{aligned}$$

If the pixels are neighbors, it penalizes changes in depth or “backgroundness” if image colors vary only slightly in the neighborhood of $p$ or $q$. We enforce smoothness only in the four nearest neighbors, of which the set $\mathcal{N}_p$ consists in our case. The Laplacian of the image is used as a simple edge detector. Exchanging the Laplacian for a more sophisticated edge detector is, of course, conceivable. The maximum norm in the above definition denotes the maximum of all color channels, so a change in any channel is sufficient for the presence of a feature, which is a natural assumption. $L_{\max}$ is the largest possible absolute value for the Laplacian, which depends on color encoding and level of discretisation. It is thus assured that $E^{p,q}_{\text{smooth}}(\lambda) \geq 0$, an intuitive requirement since discontinuity should never result in an energy bonus, but which is also important for technical reasons described in the proof later on.

### 5.3.3 Visibility constraints

Certain configurations of labels are impossible because of occlusions. If camera $j$ sees pixel $p$ at depth $\mathfrak{l}_d$, and the projection of $\langle p, \mathfrak{l} \rangle$ into another image is pixel $q$, then it is of course not possible that $q$ has a larger depth than $p$. These illegal configurations are precisely the ones captured by the set of occlusions, so we forbid them by assigning an infinite energy

$$E^{p,q}_{\text{vis}}(\lambda) \ := \begin{cases} \infty & \text{if } \{\langle p, \lambda(p)\rangle, \langle q, \lambda(q)\rangle\} \in \mathfrak{O}, \\ 0 & \text{otherwise.} \end{cases}$$

### 5.3.4 Background term

For the classification of pixels as background pixels we again use normalized cross-correlations $C_b(p)$, this time computed between the ordered sets of image colors $\mathcal{I}(\mathcal{N}_p)$ and background colors $\mathcal{B}(\mathcal{N}_p)$. We penalize good correlations of the image values with the background values if $\lambda$ does not classify $p$ as a background pixel. A second constraint is the background depth: If $\lambda_b(p) =$ true, i.e. $p$ belongs to the background, then $p$ must have the same depth $\mathfrak{b}_d(p)$ as the background. This results in the following formula:

$$E^{p}_{\text{background}}(\lambda) \ := \begin{cases} C_b(p) & \text{if } \lambda(p)_b = \text{false}, \\ \infty & \text{if } \lambda(p)_b = \text{true} \\ & \text{and } \lambda(p)_d \neq \mathfrak{b}_d(p), \\ 0 & \text{otherwise.} \end{cases}$$

In image areas with few texture information, it is often the case that the correlation $C_b(p)$ is low even if $p$ is really a background pixel. For this reason we do not penalize low correlations when the current labelling $\lambda$ classifies $p$ as background.

In the following section we reference an algorithm which efficiently computes a local minimum of the energy functional defined above.

## 5.4 Energy Minimisation

In this section we give a formal proof that graph cuts can be used to find a strong[2] local minimum of our energy functional. The algorithm works by iterating over all labels, deciding in each step which pixels have to be changed to the current label in order to reduce the energy. One can start with any valid configuration $\lambda_0$ with $E(\lambda_0) < \infty$. An obvious choice is to set each pixel to the maximum possible depth and tag it as foreground. Since the energy is always reduced and impossible configurations have infitine energy, only valid configurations can be generated. We will now investigate a single step of the iteration in more detail.

Let $\lambda$ be the current label configuration of all pixels and $\mathfrak{a}$ the current label considered. Any set of pixels $\mathcal{A} \subset \mathcal{P}$ determines a new labelling $\lambda_{\mathcal{A},\mathfrak{a}}$ via an $\mathfrak{a}$-*expansion*: Set for every $p \in \mathcal{P}$

$$\lambda_{\mathcal{A},\mathfrak{a}}(p) \; := \; \begin{cases} \mathfrak{a} & \text{if } p \in \mathcal{A}, \\ \lambda(p) & \text{otherwise.} \end{cases}$$

The goal of each step is to determine $\mathcal{A}$, i.e. the set of pixels to be assigned label $\mathfrak{a}$, such that the energy becomes smaller if at all possible, otherwise it should stay the same – formally we want $E(\lambda_{\mathcal{A},\mathfrak{a}}) \leq E(\lambda)$. A very efficient algorithm achieving this uses graph cuts and is described in detail in [62]. We do not repeat this construction here and only prove that it can be applied to our case.

First the energy funcional must be rewritten in a way which captures *energy changes* during the possible $\mathfrak{a}$-expansions. Therefore we number the pixels in $\mathcal{P}$,

$$\mathcal{P} =: \{p_1, \ldots, p_N\},$$

and define for each $i = 1, \ldots, N$ a function of a binary variable

$$\sigma_i : \{0,1\} \to \mathfrak{L}, \quad \sigma_i(x) \; := \; \begin{cases} \mathfrak{a} & \text{if } x = 1 \\ \lambda(p_i) & \text{otherwise.} \end{cases}$$

We can now define an energy $E_{\lambda,\mathfrak{a}}$ depending on $N$ binary variables which encode whether the label of the corresponding pixel is changed during the $\mathfrak{a}$-expansion or not:

$$\begin{aligned} &E_{\lambda,\mathfrak{a}} : \{0,1\}^N \to \mathbb{R}, \\ &\qquad E_{\lambda,\mathfrak{a}}(x) := E\big(\sigma_1(x_1), \ldots, \sigma_N(x_N)\big). \end{aligned}$$

[2] "strong" in the same sense as in [13]

The task of finding the set $\mathcal{A}$ is then equivalent to the task of finding a vector $x \in \{0,1\}^N$.

In consideration of Theorem 3 in [62], it is sufficient to prove the following lemma for the energy functional $E$ defined in the last section.

**Lemma.** *Determine functions $E^i$ and $E^{i,j}$ of one or two binary variables, respectively, such that for all $x \in \{0,1\}^N$*

$$E_{\lambda,\mathfrak{a}}(x) \;=\; \sum_{1\le i\le N} E^i(x_i) \;+\; \sum_{1\le i<j\le N} E^{i,j}(x_i,x_j).$$

*Then each term $E^{i,j}$ satisfies the condition*

$$E^{i,j}(0,0)+E^{i,j}(1,1) \;\le\; E^{i,j}(0,1)+E^{i,j}(1,0).$$

*Proof.* Since only terms depending on a single point or a pair of different points contribute to $E_{\lambda,\mathfrak{a}}$, rewriting the functional in the above way is possible. Indeed, it is easy to verify that the choice of

$$\begin{aligned} 2E^{i,j}(x_i,x_j) \;&=\; E^{p_i,p_j}_{\text{photo}}(\lambda_x)+E^{p_i,p_j}_{\text{smooth}}(\lambda_x)+E^{p_i,p_j}_{\text{vis}}(\lambda_x) \\ \text{with } \lambda_x(p_k) \;&:=\; \begin{cases} \lambda(p_k) & \text{if } x_k=0, \\ \mathfrak{a} & \text{otherwise,} \end{cases} \end{aligned}$$

together with the obvious choice for $E^i$ accomplishes the desired result and that all expressions are uniquely determined. The factor "2" stems from symmetry in $i$ and $j$, which is exploited to reduce the number of contributions to the ones where $i<j$.

Because of linearity it is sufficient to prove the inequality for the three different terms of the sum independently. The visibility term is the same as in [61], so there remains nothing to prove. The same applies to the photo-consistency term: Although ours is different, it is also non-positive, which was the only condition necessary. Thus we only have to investigate the smoothness term. Again in view of [61], two conditions are sufficient:

- $V^{p_i,p_j}(\mathfrak{l},\mathfrak{l})=0$ for any label $\mathfrak{l}\in\mathfrak{L}$, this is obvious by definition.
- The *triangle inequality*: for all labels $\mathfrak{l},\mathfrak{l}'\in\mathfrak{L}$,

$$V^{p_i,p_j}(\mathfrak{l},\mathfrak{l}')\le V^{p_i,p_j}(\mathfrak{l},\mathfrak{a})+V^{p_i,p_j}(\mathfrak{a},\mathfrak{l}').$$

Suppose it was wrong, then a necessary consequence is $V^{p_i,p_j}(\mathfrak{l},\mathfrak{l}')\neq 0$, $V^{p_i,p_j}(\mathfrak{l},\mathfrak{a})=0$ and $V^{p_i,p_j}(\mathfrak{a},\mathfrak{l}')=0$. But this implies $\mathfrak{l}\neq\mathfrak{l}'$ as well as $\mathfrak{l}=\mathfrak{a}=\mathfrak{l}'$, which is a contradiction.

The argumentation now proceeds as in the original proof [61, Sect. 4.2].

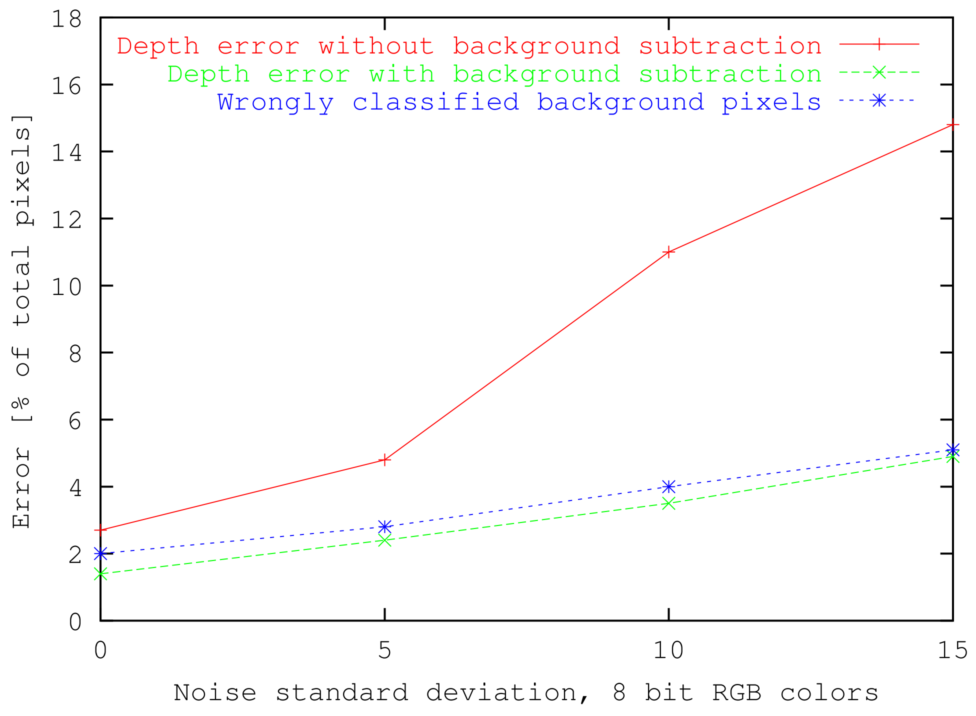


**Fig. 5.2.** Dependence of the depth estimation and background segmentation error on the amount of Gaussian noise added to the image in Fig. 5.4.

## 5.5 Results

We test the quality of the depth maps computed by our method in conjunction with our real-time dynamic light field rendering application presented in Chapter 12. The system is capable of rendering scenes from novel viewpoints inside the window spawned by the cameras. The quality of the rendering mainly depends on good per-pixel depth information. We use data from the Stanford light field camera, a $3 \times 2$ array of CMOS imagers, which we introduced in Sect. 4.2. The cameras are relatively far apart in our examples, which makes 3D-reconstruction more difficult due to the large disparity range from 3 to 34 pixels at an image resolution of $320 \times 240$ pixels. There are also dissimilarities in the color reproduction of the cameras, as well as artifacts due to MPEG compressiong during acquisition, imposing a further challenge onto color matching.

Fig. 5.3 depicts a frame of the sequence and the static background from one camera as well as the results from depth estimation and background separation. We extended our original rendering algorithm to make use of the additional background separation. It now renders first the constant background from the novel viewpoint, and then splats the foreground onto it. This method results in sharper edges and little bluriness in the final result. The overall sharpness in our rendering results indicates that the depth maps are in most

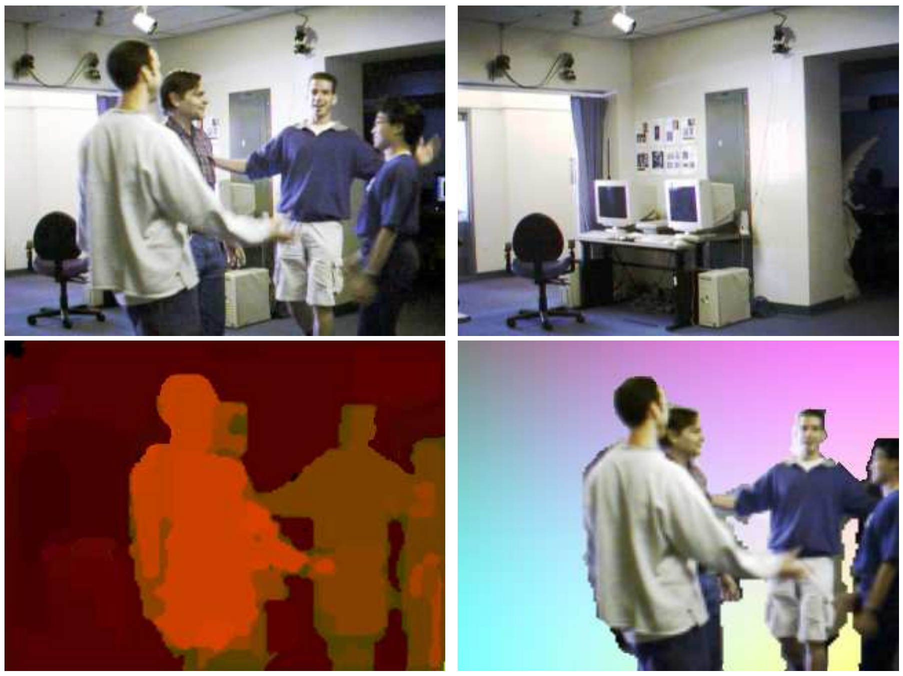


**Fig. 5.3.** *Top:* Real-world scene image and background image captured by a CMOS camera. *Bottom:* The reconstructed depth labels (brightness-coded) and the detected foreground.

areas very accurate, since each pixel is the result of a blending scheme where the two source images are weighted equally.

For a more formal verification of our method, we render a complex synthetic scene from four different viewpoints and use the Z-Buffer to obtain true per-pixel depth information. We run our algorithm to reconstruct depth and background information and compare the outcome with the known ground truth. Fig. 5.4 shows an image of the scene and some of the results. The reconstruction error is defined as the percentage of pixels for which a depth value is computed that is off by more than one pixel in disparity. Results from the new algorithm with background separation are compared to results with background separation turned off in order to demonstrate the benefits of our method in comparison to [61], Fig. 5.2. In the case with background separation, the percentage of pixels which are wrongly classified as background or foreground is also determined.

To verify the robustness of our algorithm, we perturb the color values of the input images with a preset amount of noise. To each color channel in each pixel we add a random number from a Gaussian distribution with mean zero and standard deviation $\sigma$. Here the true strength of our algorithm becomes evident. The residual error is already almost halved when compared to the original algorithm in the noiseless case, but the results of our new method remain well below 5% error even when a significant amount of noise is introduced. For the final case of $\sigma = 15$, the results from the algorithm

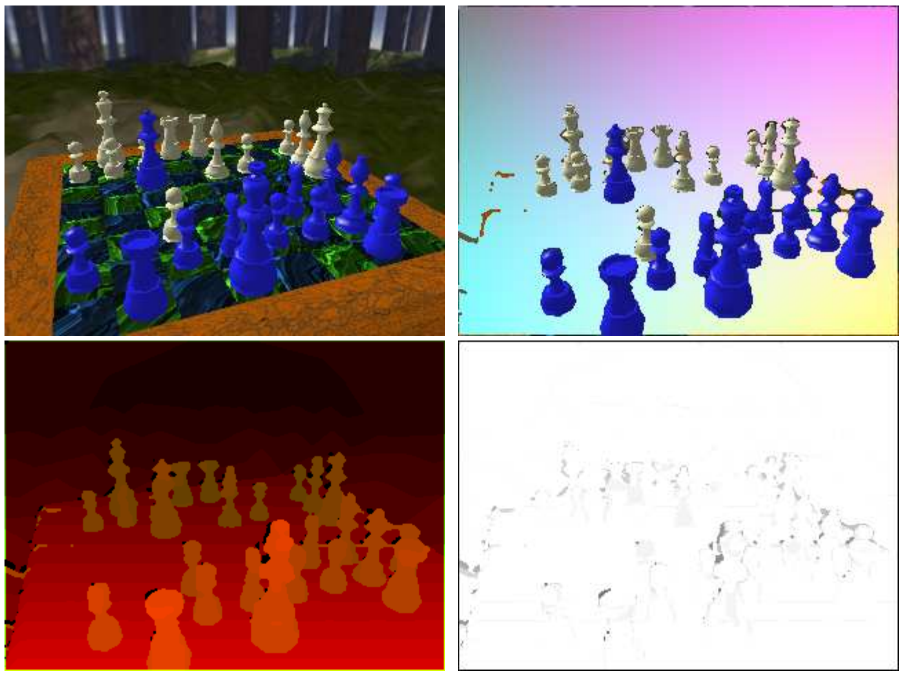


**Fig. 5.4.** *Top-left:* Synthetic scene of the position right before the famous combination in the *Immergrüne Partie*, Anderssen-Dufresne 1852. White to move and checkmate. *Top-right:* Result of the background subtraction. *Bottom:* Reconstructed depth labels and the distribution of the residual depth error compared to the known ground truth. The amount of Gaussian noise added was set $\sigma = 5$.

without background separation are almost useless, while our algorithm quite robustly gives only 4.9% faulty assigned pixels.

Both methods are running using optimal parameters, which are found to be the same in both cases - we experimentally determined $\alpha = 0.6$ and $\beta = 0.4$. Fig. 5.4 displays the result of our reconstruction with a noise standard deviation of $\sigma = 5$. Disparity values range from 2 to 20 pixels.

After 30 seconds of one-time initialization to precompute all correlations, one full cycle of iterations over all labels takes 65 seconds on a 1.8GHz Pentium III Xeon. We found that usually about four cycles are needed for convergence to the final result, so it takes a total amount of 290 seconds to compute all data for four $320 \times 240$ pixel images. Note that the number of labels and thus the iteration time is halved when background separation is turned off.

## 5.6 Conclusions

We have presented a homogenous approach to simultaneous 3D-reconstruction and background separation from multiple views of the same scene. Our results clearly demonstrate that a joint solution benefits both problems: The continous background feedback from the current estimate improves the reconstruction and vice versa. Moreover, it is a natural generalisation of an already very successful reconstruction method based on minimising a discrete energy functional via graph cuts. Existing code can be easily adapted to include background separation.

Since the algorithm is extremely flexible, it is possible to incorporate even more visual clues into its unifying framework. The next chapter will investigating how to exploit temporal coherence in video streams to further improve the reconstruction quality.

6

# Temporally Coherent Depth

## 6.1 Introduction

While the depth reconstruction technique presented in the last chapter was restricted to static scenes, an important visual clue available in video sequences is *temporal coherence*. For example, it is highly unlikely that large background regions suddenly turn into foreground, or that the depth of a region changes dramatically without accompanying changes in color. In this chapter, we present a way to consistently estimate depth by exploiting spatial as well as temporal coherence constraints. Our general framework for the reconstruction is still the global minimization of a discrete energy functional. The minimum of the functional yields the final reconstruction result. In particular, the important advantages we inherit include that all cameras are treated symmetrically, and that visibility is handled properly.

Our method automatically integrates several visual clues into one common mathematical framework. We take into account local criteria like correlations between pixel neighborhoods, as well as groupings of pixels with regard to color values and background similarity. Temporal coherence of the resulting depth maps is enforced and is found to greatly increase estimation accuracy and robustness.

The algorithm relies on a number of different visual clues. In order to handle them in a structured way, our system is separated into two components. The first and central component pre-processes the input data. All images are statistically analyzed in order to estimate the significance of each clue. Based on a few obvious assumptions, the clues are automatically weighted against each other. No user-defined parameter is required during the whole process, in contrast to the ealier method. The pre-processing stage works in part on a purely local basis comparing pixel neighborhoods, but it also exploits non-local attributes relying on an initial grouping of pixels.

As the second key component of our algorithm, we utilize the global reconstruction method based on an energy minimization via graph cuts. Both components are connected by letting the initial pre-processing determine the parameters of the energy functional to be minimized. That way, both components work hand in hand, and a unifying mathematical formulation can be found for the description of the algorithm. A nice side effect is that the original static version of the algorithm from the last chapter is also improved by eliminating the need for user-defined coupling constants in the energy functional. The new energy functional consists of five contributions:

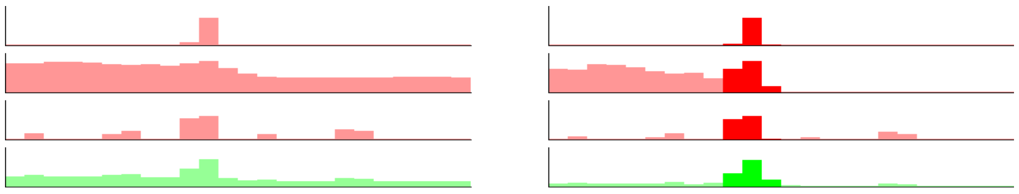
$$E = E_{\mathrm{photo}} + E_{\mathrm{vis}} + E_{\mathrm{smooth}} + E_{\mathrm{bg}} + E_{\mathrm{alloc}}, \tag{6.1}$$

describing photo consistency, visibility constraints, smoothness constraints, background similarity and the initial allocation, respectively. The *photo-consistency term* issues a bonus if two pixels which are the projections of the same 3D point have similar neighborhoods. *Visibility constraints* rule out illegal configurations and guarantee that all depth maps in a single time-step are consistent with each other. Spatial smoothness in low-gradient regions is enforced by the *smoothness term*, while the *background term* encourages pixels with high similarity to the background to be tagged as background. A novel term is the last one, which introduces an *initial allocation* for each pixel's depth.

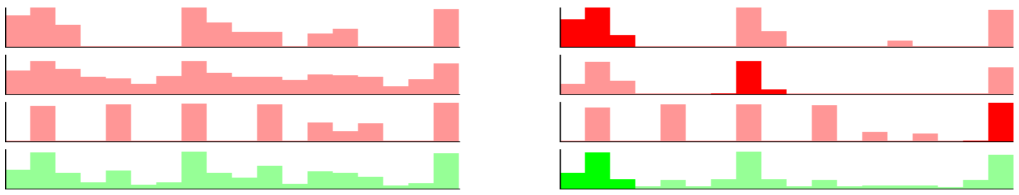
We proceed by introducing the pre-processing stage in the next section. All results of this pre-processing stage are encoded into the global energy functional defined in Sect. 6.3. We show that it can be minimized with Kolmogorov and Zabih's powerful graph cut technique [61] as well. Results presented in Sect. 6.4 demonstrate significant improvements in accuracy and robustness when compared to our previous method. We conclude with some ideas for future work in Sect. 6.5.

## 6.2 Pre-processing

The image properties taken into account in our approach can be divided into two categories. *Local data* denotes pixel characteristics that depend only on single pixels plus a $3 \times 3$ neighborhood on which normalized cross correlations are computed. Local constraints are derived from photo-consistency requirements between pixels which are projections of the same 3D point. Pixels in different images corresponding to the same 3D point are called *interacting* pixels.



(a) Example tuples of correlations with high statistical significance. The top three rows show the $k = 3$ input tuples, the bottom row the final tuple after weighting. The left result is obtained by simple averaging with equal weights. On the right side, the result is obtained using the more sophisticated scheme described in the text. Darker bars mark the ratio of mass used to compute the weight. The maximum is clearly enhanced, while insignificant noise is supressed.



(b) The same graphs for example tuples with low statistical significance. The three potential matches are enhanced, but the final confidence is low, because only a third of the mass is centered around the maximum.

**Fig. 6.1.** *Depth estimate and confidence from local characteristics:* The graphs on the left show initial tuples of correlations with depth marked on the x-axis. Depicted on the right are graphs of the same tuples with enhanced local maxima, illustrating the weighting process described in the text.

On the other hand, *group data* is non-local in nature and denotes properties that are averaged over a larger region of pixels. An initial grouping of pixels into regions is performed with respect to similar color and background difference characteristics of pixels. It tends to stabilize the result and to eliminate outliers, as explained later.

The following subsections focus on these two different kinds of pixel characteristics, explaining in detail how we obtain them and how they are embedded in the larger context. It is very important to note that the local properties of pixels as well as regional properties of groups computed in this section are only an *initial estimate.* This is also true for the grouping itself: Pixels within the same group can and often will have different depths in the end. The final step in scene reconstruction is truly global in nature. The global integration based on minimizing the energy functional constructed from the local data accomplishes this task, Sect. 6.3.

### 6.2.1 Local Data

Fix a pixel $p$ in one of the source images and regard its correspondence in a different source image $\mathcal{J}$. Let $[m, M]$ be the disparity search range. Each depth value $d \in [m, M]$ defines a 3D point $\langle p, d\rangle$, which has in turn a projection $q$ in $\mathcal{J}$. We aim at finding a measure $\chi_{pq}$ for the similarity between the neighborhoods of $p$ and $q$ as well as a measure $\alpha_{pq}$ for the confidence of this estimate. If both are high, it is likely that the correct depth for $p$ is $d$.

To this end, we first compute $k$ different similarity criteria as are widely used in stereo matching. Typically, we use $k = 3$ criteria: The first one is the correlation between the color values, the second one the inverse maximum difference between the color values. As a third one, we compute the correlation between the Laplacians of the images in order to encourage high-frequency image features to be mapped onto their counterparts in $\mathcal{J}$. Experiments with correlations between partial derivatives indicate that they do not further improve the result. We end up with tuples $(\chi_p^i(d))_{d=m}^M$ of values between 0 and 1 for each depth label and for each criterion $i = 1, \ldots, k$.

For the final similarity measure $\chi_{pq}$, these tuples have to be weighted depending on their relative relevance. Instead of demanding user-supplied global weights, we estimate the statistical significance of each tuple individually. To this end, we first linearly scale the tuple, mapping the range as follows:

$$\left[\mu^i, \nu^i\right] \hookrightarrow\!\!\!\!\rightarrow \left[0, \nu^i\right],$$

$$\text{where } \mu^i := \operatorname*{avg}_{d=m}^{M}(\chi_p^i(d)) \text{ and } \nu^i := \max_{d=m}^{M}(\chi_p^i(d)).$$

Results are clamped to $[0, 1]$. This step emphasizes maxima in a tuple while discarding correlations with high uncertainty, Fig. 6.1. Afterwards, we compute a weight $\alpha^i$ for each tuple. We assume that a tuple is statistically more significant if most of the mass is centered around its maximum:

$$\alpha^i \; := \; \frac{\chi_p^i(d_\mu^i - 1) + \chi_p^i(d_\mu^i) + \chi_p^i(d_\mu^i + 1)}{\sum_{d=m}^{M} \chi_p^i(d)},$$

assuming that the maximum $\nu^i$ is attained in $d_\mu^i$. Using these weights, the final similarity for each interacting pixel $q$ is computed. With $q$ being the projection of $\langle p, d\rangle$ into image $\mathcal{J}$, we define $\chi_{pq}$ to be the weighted average

$$\chi_{pq} \; := \; \frac{\sum_{i=1}^{k} \alpha^i \, \chi_p^i(d)}{\sum_{i=1}^{k} \alpha^i}.$$

The confidence $\alpha_{pq}$ of the final estimate is set to the maximum weight $\max_i(\alpha^i)$. Note that it depends only on $p$, and $\alpha_{pq} = \alpha_{qp}$.

So far, we have compared only small neighborhoods of individual pairs of pixels at a single time step. We now turn our attention to data which is non-local in nature and depends on larger regions in the image. In addition, we take into account the estimates from the previous time step to enforce temporal consistency.

### 6.2.2 Group Data

In previous algorithms, outliers are eliminated by the smoothness term in the energy functional whose influence has to be high to supress outliers with high statistical evidence. As a consequence, the result is often oversmoothed, since the parameter determining smoothness is defined globally. Instead, we eliminate outliers locally at an early stage during reconstruction. This is achieved by an initial grouping of the pixels with respect to their similarity in color and background difference. Averaged values are then assigned to the resulting groups.

**Grouping pixels.** The grouping is performed separately for each input image $\mathcal{I}$ by a variant of the Recursive Shortest Spanning Tree (RSST) algorithm [83]. Initially, each group contains only one pixel. Hence we have $\mathcal{G}_p = \{p\}$, where we denote with $\mathcal{G}_p$ the group a pixel $p$ belongs to. The following characteristics of each group $\mathcal{G}$ are used during the clustering process:

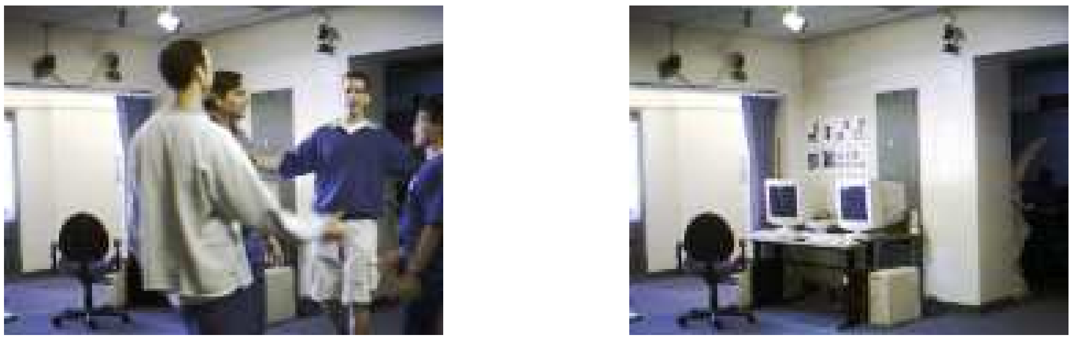
| | |
|---|---|
| $\mu_c(\mathcal{G})$ | Average color value of all pixels in the group |
| $\sigma_c(\mathcal{G})$ | Standard deviation of color values |
| $\sigma_b(\mathcal{G})$ | Standard deviation of the error between the current frame and the background frame, averaged over all pixels in the group |

Only adjacent groups are merged based on a *cost function* $\kappa$. The adjacent group with the lowest cost is selected for merging. This cost function keeps the statistical significance of the grouping as high as possible and minimizes the standard deviation of the resulting merged group:

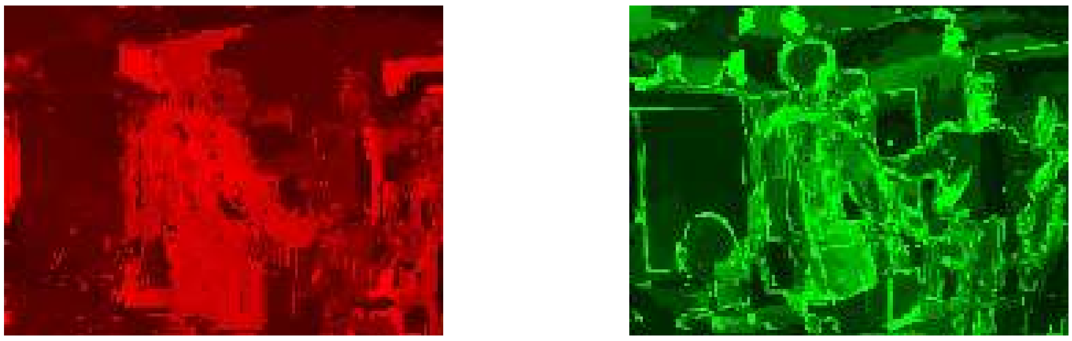
$$\kappa(\mathcal{G}, \mathcal{H}) := \sigma_c(\mathcal{G} \cup \mathcal{H}) + \sigma_b(\mathcal{G} \cup \mathcal{H}).$$

A merge is rejected when the cost function exceeds a threshold value. Merging of groups stops when the cost for merging any adjacent pair of groups exceeds this threshold. We have to choose the threshold such that a good compromise between over- and undersegmentation is obtained. A threshold value of $0.3\sigma_c(\mathcal{I}) + 0.1\sigma_b(\mathcal{I})$ works well in practice, and it is used for all our test sequences presented.

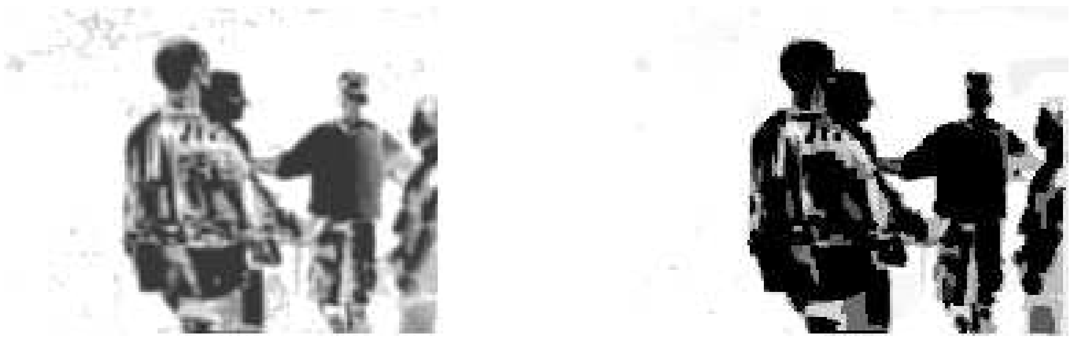
**Most likely depth of the group.** After the grouping is complete, we compute some more characteristics for each group $\mathcal{G}$. The first one is the initial



(a) Single video frame and its background.



(b) *Left:* Initial depth allocation based on local correlations alone. *Right:* Estimated confidence of the initial depth allocation: Bright areas indicate high confidence. Note that most areas in which depth is grossly wrong have only a very low estimated confidence. Accordingly, a depth map computed only from this local evidence alone exhibits inaccuracies.



(c) *Left:* Difference between current frame and background frame. Note the white regions within the people's silhouettes. *Right:* Total accumulated foreground evidence. Although the result is far more decisive than the difference image alone, it is still not possible to correctly retrieve the foreground region by simple thresholding.

**Fig. 6.2.** Results after pre-processing the real-world sequence.

depth allocation $d_c(\mathcal{G})$ of the group based on the local correlation data $\chi_{pq}$ and $\alpha_{pq}$. Choose a pixel $p \in \mathcal{G}$. For each depth $d$, let $Q_p^d := \{q_1, \ldots, q_n\}$ be the set of pixels which interact with $p$ at depth $d$, i.e., which are projections of $\langle p, d \rangle$ into any other of the source images. We assign to depth $d$ the likelihood

$$\hat{\chi}_p^d := \sum_{q \in Q_p^d} \alpha_{pq} \cdot \chi_{pq}.$$

The depth $\hat{d}_p$ with maximum likelihood is selected as the depth estimate for $p$ with confidence

$$\hat{\alpha}_p := \max_{q \in Q_p^d} (\alpha_{pq}).$$

We finally compute the initial depth allocation of the group $\mathcal{G}$ as the weighted average of the group's pixel depths:

$$d_c(\mathcal{G}) := \frac{\sum_{p \in \mathcal{G}} \hat{\alpha}_p \hat{d}_p}{\sum_{p \in \mathcal{G}} \hat{\alpha}_p}.$$

The confidence $\nu_c(\mathcal{G})$ of the group's estimate is set to the average of $\hat{\alpha}_p$ over all $p \in \mathcal{G}$. If the estimate is smaller than the depth of the background, it is discarded by setting its confidence to zero.

**Temporal Coherence.** In order to enforce temporal coherence, we attempt to find for each group $\mathcal{G}$ pixels in the previous time-step which belong to the history of $\mathcal{G}$. We use a very conservative estimate and check for each pixel $p \in \mathcal{G}$ whether or not $c^-(p)$ fits into $\mathcal{G}$. If it does, it is likely that it either belongs to the past version of $\mathcal{G}$ or to an object which $\mathcal{G}$ is part of. In both cases, the labelling $\lambda^-(p)$ from the previous time-step is a good estimate for the correct label of pixels in $\mathcal{G}$ in the current time-step.

We use the following function to determine whether a color value $c$ fits into a group $\mathcal{G}$:

$$\text{fits}(c, \mathcal{G}) := \max\left(0, \sigma_c(\mathcal{G}) - \|\mu_c(\mathcal{G}) - c\|_\infty\right).$$

It is positive whenever the error between $c$ and the average color $\mu_c(\mathcal{G})$ is smaller than the standard deviation $\sigma_c(\mathcal{G})$ of the colors. The set of pixels which is assumed to lie in the history of $\mathcal{G}$ is then

$$\mathcal{G}^- := \left\{p \in \mathcal{G} \,:\, \text{fits}(c^-(p), \mathcal{G}) > 0\right\}$$

We can now write down an equation for another initial estimate $d_t(\mathcal{G})$ for the depth of $\mathcal{G}$ which relies on temporal coherence:

$$d_t(\mathcal{G}) := \text{avg}\left\{\lambda^-(p)_d \,:\, p \in \mathcal{G}^-\right\}.$$

It has an associated confidence of $\nu_t(\mathcal{G}) := \|\mathcal{G}^-\| \,/\, \|\mathcal{G}\|$, i.e., it is equal to the ratio of pixels contributing to the estimate.

We now compare the disparity estimate $d_c(\mathcal{G})$ obtained from local neighborhood correlations with the estimate $d_t(\mathcal{G})$ from temporal coherence. First, the final estimate $d(\mathcal{G})$ for the group's depth is set to

$$d(\mathcal{G}) := \begin{cases} d_c(\mathcal{G}) & \text{if } \nu_c(\mathcal{G}) \geq \nu_t(\mathcal{G}), \\ d_t(\mathcal{G}) & \text{if } \nu_t(\mathcal{G}) > \nu_c(\mathcal{G}). \end{cases}$$

If $d_c(\mathcal{G})$ and $d_t(\mathcal{G})$ differ by at most one pixel, we award it with a synergy bonus and set the final confidence to

$$\nu(\mathcal{G}) := \nu_c(\mathcal{G}) + \nu_t(\mathcal{G}).$$

Otherwise, the confidences cancel each other out:

$$\nu(\mathcal{G}) := \max(\nu_c(\mathcal{G}), \nu_t(\mathcal{G})) - \min(\nu_c(\mathcal{G}), \nu_t(\mathcal{G})).$$

Temporal coherence also plays a role in background subtraction, as described later.

**Background and Foreground Evidence.** Using the background images and the data computed in the previous steps, the total evidence for each group $\mathcal{G}$ to belong to the background or to the foreground is now accumulated in the values $\beta(\mathcal{G})$ and $\phi(\mathcal{G})$, respectively. Let $\Delta_b$ be the error between current image and background image, averaged over all pixels belonging to $\mathcal{G}$. Initially, the evidence values are then set to

$$\begin{aligned} \beta(\mathcal{G}) &:= (2 - 2\Delta_b)^2 \\ \text{and } \phi(\mathcal{G}) &:= (4\Delta_b)^2, \end{aligned}$$

indicating strong foreground evidence if the pixel is very different from the background and somewhat smaller evidence for background otherwise. Afterwards, we increase the evidence for foreground if the group estimate for the depth differs significantly from the depth of the background. Let $d_b$ be the averaged depth of the group's background, then

$$\phi(\mathcal{G}) \longleftarrow \phi(\mathcal{G}) + \begin{cases} \nu(\mathcal{G}) & \text{if } |d_b - d(\mathcal{G})| > 1, \\ 0 & \text{otherwise.} \end{cases}$$

Additional strong evidence for foreground can be derived from the history of $\mathcal{G}$. Let

$$\mathcal{G}_b^- := \{p \in \mathcal{G} : \lambda^-(p)_b = \text{true}\}$$

be the set of pixels belonging to $\mathcal{G}$ which were background in the previous frame. It is clear that if $\mathcal{G}$ is also background, then all pixels in $\mathcal{G}_b^-$ must fit into $\mathcal{G}$ since they are static, thus $\mathcal{G}_b^- \subset \mathcal{G}^-$. We increase the foreground evidence by the ratio of pixels not satisfying this relation:

$$\phi(\mathcal{G}) \longleftarrow \phi(\mathcal{G}) + \frac{\|\mathcal{G}_b^- \setminus \mathcal{G}^-\|}{\|\mathcal{G}_b^-\|}.$$

Note that all of the accumulated foreground evidence is compulsive in the sense that if it is accurate, the pixel must actually belong to the foreground. On the other hand, the pixel does not necessarily have to belong to the background even if it has similar color and depth. Because of this, we let very strong foreground evidence always overrule background evidence:

$$\beta(\mathcal{G}) \longleftarrow \max(0, \beta(\mathcal{G}) - \phi(\mathcal{G})^2).$$

Now all local and regional statistical data is computed. Of course, all the different visual clues described in this section have numerous other interrelationships not covered by our equations. However, our experiments indicate that those we present here have the strongest impact. All of them were tested individually for their ability to improve the final result.

We have now derived all terms we need for Eq. (6.1) and are in the position to construct the energy functional whose minimum yields the final global depth and background separation estimate.

| *Variable* | *Meaning* |
|---|---|
| $\chi_{pq}$ | Similarity of the neighborhoods of pixels $p$ and $q$ |
| $\alpha_{pq}$ | Estimate for the confidence of $\chi_{pq}$ |
| $\mu_c(\mathcal{G})$ | Average color of the pixels in $\mathcal{G}$. |
| $\sigma_c(\mathcal{G})$ | Standard deviation of the color values. |
| $\phi(\mathcal{G})$ | Evidence for $\mathcal{G}$ to belong to the foreground |
| $\beta(\mathcal{G})$ | Evidence for $\mathcal{G}$ to belong to the background |
| $d\,(\mathcal{G})$ | Initial estimate for the groups depth |
| $\nu\,(\mathcal{G})$ | Estimate for the confidence of $d\,(\mathcal{G})$ |

**Table 6.1.** *Overview of the data used in the energy functional.*

## 6.3 The Energy Functional

In the global estimation step, the locally and regionally obtained estimates are integrated into one global reconstruction framework. This process is guided by a functional assigning a scalar energy value to any configuration of labels.

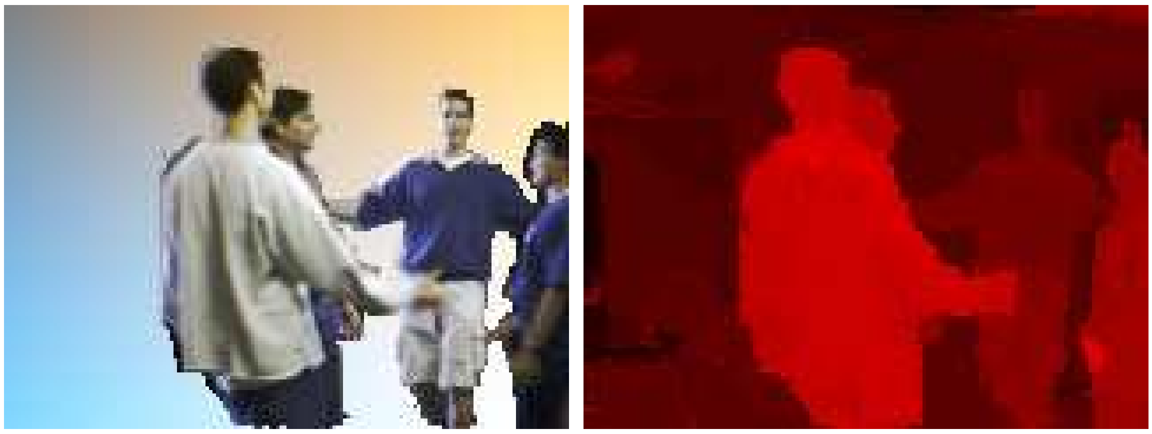


**Fig. 6.3.** Background subtraction and final depth map after global energy minimization. The result is improved compared to the one from the original algorithm, Fig. 5.3, in particular the depth estimates in the foreground region (head, legs, arms).

It can be written as a sum of contributions by every single pixel and every possible pair of pixels:

$$\begin{aligned} E(\lambda) \;=\; & \sum_{p,q\in\mathcal{P}} \left[ E^{p,q}_{\text{photo}}(\lambda) + E^{p,q}_{\text{vis}}(\lambda) + E^{p,q}_{\text{smooth}}(\lambda) \right] \\ & + \sum_{p\in\mathcal{P}} \left[ E^{p}_{\text{background}}(\lambda) + E^{p}_{\text{alloc}}(\lambda) \right]. \end{aligned}$$

The terms on the right hand side will be different from zero only if $p$ and $q$ interact or occlude each other in configuration $\lambda$, or if $p$ and $q$ are neighbors. In effect, the sum runs only over relatively few pairs of points. Note that there are no weights in the functional and thus we do not need any free parameters. The relative importance of the different terms is based on the statistical evidence derived in the pre-processing step. The minimization of the functional is performed by a graph cut method, which is thouroughly described and investigated in [61]. We will now explain the different contributions in more detail and continue with a discussion of the applicability of graph cuts at the end of the section. While a few terms are similar to the ones in the last chapter, there are subtle changes because of the pre-processing, so we discuss them here.

**Photo-consistency term.** If a 3D point is projected onto a pixel $p$ in one image and onto a pixel $q$ in another image, and if it is visible in both images, we issue a photo-consistency bonus. In all other cases this term is zero. The bonus is based on the similarity $\chi_{pq}$ and its confidence $\alpha_{pq}$ computed in the pre-processing step:

$$E^{p,q}_{\text{photo}}(\lambda) \;:=\; -(2\alpha_{pq})^2 \cdot \chi_{pq} \;-\; \mathit{\Gamma}_{pq}.$$

In order to render the estimate less local and thus more stable, we also award a bonus with regard to the characteristics of the pixels' groups, which is encoded in $\Gamma_{pq}$. We define

$$\Gamma_{pq} \;:=\; \text{fits}(c\,(p), \mathcal{G}_q) \;+\; \text{fits}(c\,(q), \mathcal{G}_p) \;\geq\; 0,$$

where the previously defined function *fits* measures how good the color in the first argument fits into the group in the second argument. If the error between the pixel's color and the group's average color is larger than the standard deviation of the colors in the group, there is no bonus anymore.

**Smoothness term.** As this term imposes spatial smoothness within the depth maps of the images, it is non-zero only if $p$ and $q$ are neighbors in the same image with respect to the standard four-connectivity. Drastic changes in depth or transitions from background to foreground are more likely at places where the characteristics of the pixels change significantly. We use the grouping of pixels based on their characteristics as a measure of their similarity. Changes within groups are very expensive, while the price for changes at group boundaries depends on the similarity of the colors of the pixel's groups. Mathematically, the *smoothness energy* is defined as

$$E^{p,q}_{\text{smooth}}(\lambda) := V^{p,q}\big(\lambda(p), \lambda(q)\big),$$

$$\text{where } V^{p,q}(\mathfrak{l}, \mathfrak{l}') := \begin{cases} 0 \text{ if } \mathfrak{l} = \mathfrak{l}', \\ 1 - \|\mu_c(\mathcal{G}_p) - \mu_c(\mathcal{G}_q)\|^2_\infty \text{ otherwise.} \end{cases}$$

**Visibility constraints.** Certain configurations of labels are impossible because of occlusions. If camera $j$ sees pixel $p$ at depth $\mathfrak{l}_d$, and the projection of $\langle p, \mathfrak{l}_d \rangle$ into another image is pixel $q$, then it is, of course, not possible that $q$ has a larger depth than $p$. If this is the case, an infinite energy is assigned to $E^{p,q}_{\text{vis}}(\lambda)$ to rule out illegal configurations. Otherwise, the term is zero.

**Background term.** The evidence for $p$ to be background or foreground is encoded into the values $\beta(\mathcal{G}_p)$ and $\phi(\mathcal{G}_p)$, respectively. They result in a background penalty

$$E^p_{\text{background}}(\lambda) \;:=\; \begin{cases} \infty & \text{if } \lambda(p)_b = \text{true} \\ & \text{and } \lambda(p)_d < \mathfrak{b}_d p, \\ \infty & \text{if } \lambda(p)_d > \mathfrak{b}_d p, \\ \beta(\mathcal{G}_p) & \text{if } \lambda(p)_b = \text{false} \\ \phi(\mathcal{G}_p) & \text{otherwise.} \end{cases}$$

Here $\mathfrak{b}_d p$ is the depth of the background at $p$. We forbid the impossible case that a pixel is assigned a depth larger than the background depth. Furthermore, a pixel classified as background can never have a different depth than the static background.

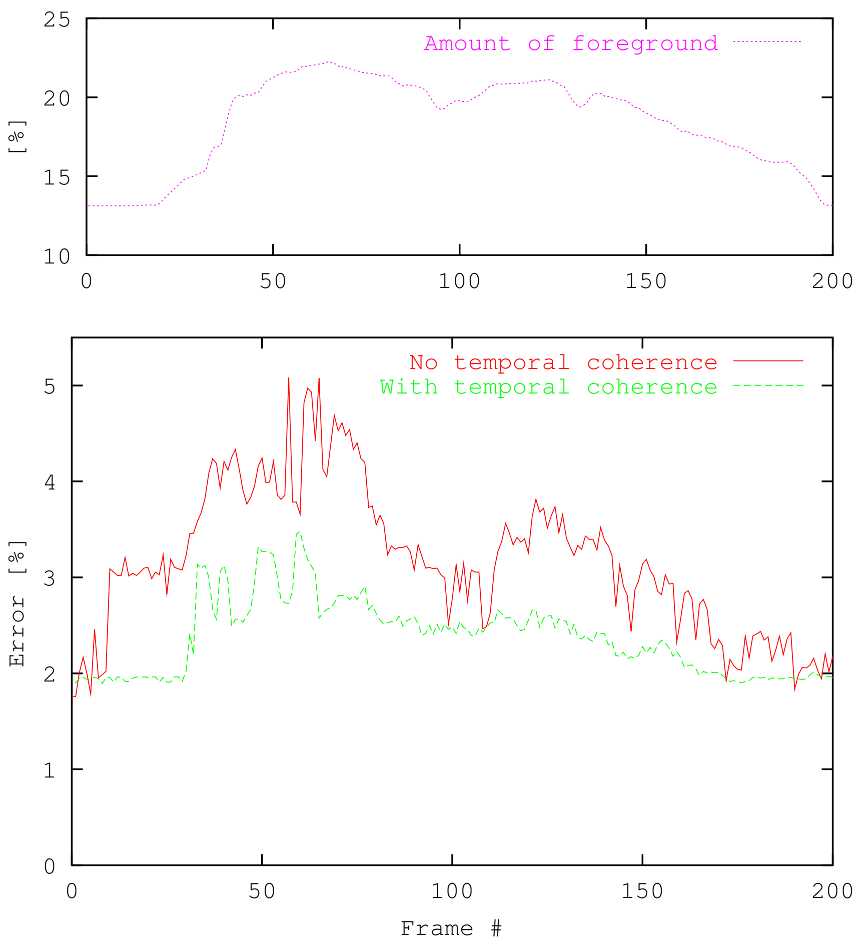


**Fig. 6.4.** Results of applying our algorithm to the synthetic test sequence and comparing the result to ground truth depth. The bottom graph depicts the percentage of erroneous pixels in each frame.

**Initial Depth.** An additional clue taken into account is the initial allocation $d(\mathcal{G}_p)$ for the depth of the pixel $p$'s group and its confidence measure $\nu(\mathcal{G}_p)$. Both are computed in the pre-processing step. We assign a penalty if the current label differs greatly from this initial allocation *and* this initial allocation is likely to be correct. More precisely,

$$E^p_{\text{alloc}}(\lambda) := (2\nu(\mathcal{G}_p))^2 \cdot \frac{|\lambda(p) - d(\mathcal{G}_p)|}{M - m},$$

where $[m, M]$ is the disparity search range.

Kolmogorov and Zabih [62] discuss which energy functions can be minimized via graph cuts. It is easy to verify that our functional is one of these. Indeed, taking into account the arguments in [42], all that remains to be proven is that $V^{p,q}$ is a metric with respect to arbitrary labels, which is a straightforward exercise.

## 6.4 Results

Our first example is a real-world sequence captured with six cameras of the Stanford Multi-Camera Array consisting of 250 frames. At the downscaled resolution of $160 \times 120$ pixels we use in our algorithm, maximum disparity is 30 pixels. The pre-processing results for single image are depicted in Fig. 6.2. It is evident from Fig. 6.2b and Fig. 6.2c that depth map estimation as well as background separation using local evidence alone leads to numerous outliers. The sequence exhibits foreground that consists mostly of color values which are also abundantly present in the background, and photometric as well as camera geometry calibration is not exact. When employing our temporally consistent method, however, the results after global energy minimization exhibit virtually no outliers anymore, Fig. 6.3.

Since ground truth data for the real-world sequence cannot be available, we also use a synthetic scene for quantitative evaluation of the algorithm. The sequence is rendered from four camera positions using the Maya rendering software including anti-aliasing in order to optimize quality. In addition, we add Gaussian noise of 5% to the images prior to running the reconstruction algorithm. A single frame from one of the four cameras is shown in Fig. 6.5. In total, the animation consists of 200 frames, which we also process at a resolution of $160 \times 120$ pixels. At that resolution, the maximum disparity measures 35 pixels.
Fig. 6.4 depicts the error in 3D reconstruction. We measure the ratio of pixels for which the depth estimate yields a depth different by more than one pixel in disparity than the ground truth depth. The total amount of foreground in each frame is also included in the graph. Background depth is obtained by first running the algorithm on the background images, with background subtraction turned off and at full resolution in order to increase accuracy. For the background, only 1.95% of all pixels are assigned erroneous depth.
The Fig. 6.4 compares the algorithm with and without temporal coherence activated. Results obtained with the full version of the algorithm including temporal coherence show a reduction of the average total error from 3.2% to 2.3%, equivalent to a reduction of 28%. Additionally, the error stays almost constant over long periods of time, indicating that the depth maps are temporally consistent. This is mandatory, for instance, in image-based rendering, when depth is used to display the sequence from a novel viewpoint. We also observe that the background subtraction greatly helps with the disparity estimate of the foreground: The percentage of erroneous foreground pixels in both methods is, on average, only 2.3% and 1.6%, respectively, about two third of the total error. The number of pixels erroneously detected to be background or foreground is in both cases already below 0.6% on average, with only a slight improvement when temporal coherence is turned on.

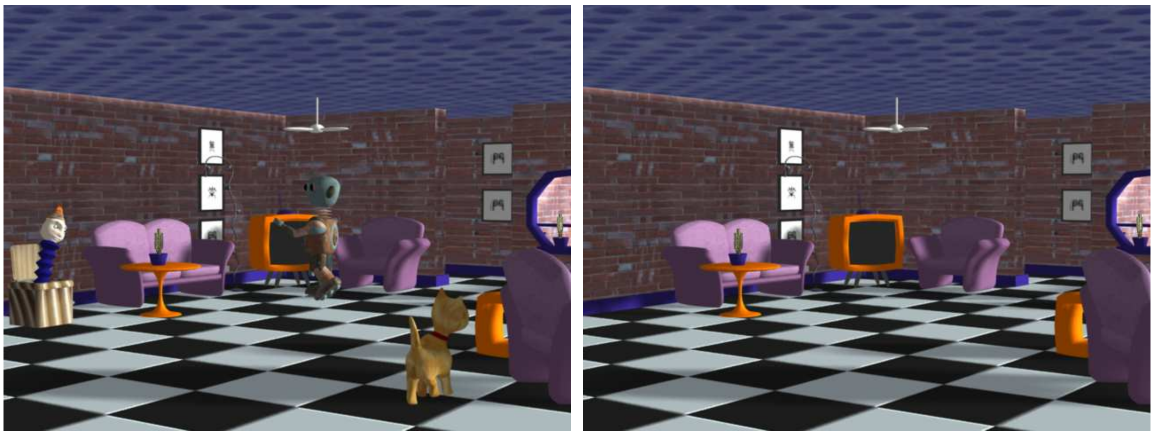


**Fig. 6.5.** A frame of the synthetic test sequence and its background. Moving objects are the robot, the cat and the jack-in-a-box.

Compared to the final energy minimization, pre-processing requires only a negligible amount of CPU time. For each frame of the sequence, we let the graph cut technique iterate five times over each depth and background label. At a downsampled image resolution of $160 \times 120$ pixels and a search range of 25 pixels, it takes about 4.5 minutes to perform all iterations on a 1.8 GHz Pentium IV PC. Afterwards, depth and background estimates for all four cameras are computed. The initial pre-processing takes only 21 sec. per time frame, most of which includes the pre-computation of all correlations which has to be performed anyway.

## 6.5 Conclusions

We have presented a novel technique to integrate several visual clues for 3D scene reconstruction into one common framework. The local significance of each clue is determined automatically through statistical analysis of the camera images. This pre-processing stage paves the way towards global reconstruction based on energy minimization via graph cuts. By explicitly enforcing temporal coherence, we obtain excellent depth maps as well as a separation into foreground and background in multiple video streams.

Some possible future work includes lifting constraints on camera geometry. Currently, our use of fixed rectangular windows to compute correlations assumes pure translatory difference between cameras, although the functional can handle more general geometries. It would also be nice to estimate an alpha matte instead of a binary separation into background and foreground, and to find a way to optimally handle non-Lambertian surfaces.

# Part III

# Surface Reconstruction

# 7

# Overview

## 7.1 Introduction

In this part of the thesis, we take the reconstruction problem one step further and are not content with simple depth maps anymore. Instead, our aim is to recover the full 3D surface geometry of arbitrary objects from multi-video data from a handfull of cameras surrounding them. The geometry models obtained that way enable us to render the dynamic scene from arbitrary viewpoints in high quality, using image-based rendering techniques we investigate in Chapter 13.

Our estimated geometry is defined as a *weighted minimal surface*, which minimizes an energy functional given as a surface integral of a scalar-valued *weight* or *error function*. The variational formulation of these kinds of minimization problems leads to a surface evolution PDE which can be explicitly solved using level set techniques. Other implementations use triangle meshes [30], which are more complicated to implement due to topology changes, but can be a lot faster [41]. Recently, we also started to investigate the use of surfel models as a base representation to implement the surface evolution [69]. Initial results seem to indicate that we get a performance compared to the triangle mesh implementation, while we do not have to worry about topology.

In the remainder of this chapter, we present the mathematical framework for the surface reconstruction problem in general, and discuss computer vision techniques which already rely on it. Chapter 8 presents the necessary mathematical analysis of the minimal surface problem, yielding the final surface evolution equation implemented for the reconstruction process. Chapter 9 then focusses on the aspect of spatio-temporal surface reconstruction from multi-video data, a computer vision problem that is a primary focus of research interest. Finally, in Chapter 10, we present a scheme to tackle the reconstruction of flowing, volumetric media like water. With a properly defined error density, the desired reconstruction result is also recovered as a weighted minimal surface.

## 7.2 Weighted Minimal Surfaces

A popular and successful way to treat many computer vision problems is to formulate their solution as a hypersurface which minimizes an energy functional given by a weighted area integral. In this part of the thesis, we want to expose, generalize and solve the mathematical problem which lies at the very heart of all of these methods.

Our aim is to find a $k$-dimensional regular hypersurface $\Sigma \subset \mathbb{R}^n$ which minimizes the energy functional

$$\mathcal{A}(\Sigma) := \int_\Sigma \Phi \, dA. \tag{7.1}$$

We will only investigate the case of codimension one, so throughout this text, $k = n - 1$. Such a surface is called a *weighted minimal hypersurface* with respect to the weight function $\Phi$. This function shall be as general as required in practice, so we allow it to depend on the surface point $s$ and the surface normal $\mathbf{n}$.

In Chapter 8, we derive an elegant and short proof of the necessary minimality condition:

**Theorem 7.1.** *A $k$-dimensional surface $\Sigma \subset \mathbb{R}^{k+1}$ which minimizes the functional $\mathcal{A}(\Sigma) := \int_\Sigma \Phi(s, \mathbf{n}(s)) \, dA(s)$ satisfies the Euler-Lagrange equation*

$$\langle \Phi_s, \mathbf{n} \rangle - Tr(\mathbf{S})\Phi + div_\Sigma(\Phi_\mathbf{n}) = 0, \tag{7.2}$$

*where $\mathbf{S}$ is the shape operator of the surface, also known as the Weingarten map or second fundamental tensor.*

Using standard techniques, a local minimum can be obtained as a stationary solution to a corresponding surface evolution equation. Since this surface evolution can be implemented and solved in practice, the Theorem yields a generic solution to all problems of the form (1) for practical applications. In this work, we set aside the problems of convergence and local minima, see e.g. [21] for a detailed analysis of those.
Our work has thus two main contributions:

*Unification*: We unite a very general class of problems into a common mathematical framework. This kind of minimization problem arises in numerous contexts in computer vision, with dimension $n \leq 3$ and various choices of $\Phi$. A few select examples are summarized in Sect. 7.3, among them the method of geodesic snakes for segmentation as well as a very general multi-view 3D reconstruction technique. Our theorem yields the correct surface evolution equations for all of them.

*Generalization*: Our result is valid in arbitrary dimension. We are not aware of a previously existing treatment in computer vision literature of this generality. Until now, the theorem has been proved separately in dimensions $k = 1$ and $k = 2$, using local coordinates on the surface[1]. The now freely selectable number of surface dimensions opens up new possibilities for future applications. As one example, we generalize the static 3D reconstruction of a surface towards a space-time reconstruction of an evolving surface in Chapter 9, which can be viewed as a 3D volume in 4D space. The proposed method treats all frames of multiple video sequences simultaneously in order to provide a temporally coherent result.

In the special case that $\Phi = 1$ is constant, the problem of minimizing (7.1) is reduced to finding a standard minimal surface, which is defined to locally minimize area. As we deal with a generalization, it seems reasonable to adopt the same mathematical tools used in that context [26]. A brief review of this framework, known as the *method of the moving frame*, is given in Sect. 8.2. However, we are forced to assume that the reader has at least some familiarity with differential geometry, preferably of frame bundles. The transition from the Euler-Lagrange equation to a surface and further to a level set evolution equation is reviewed in Sect. 8.4, where we also discuss some necessary implementation details.

## 7.3 Variational Methods in Computer Vision

Among the first variational methods which were successfully utilized for computer vision problems was the one now widely known as *Geodesic Active Contours* [17]. While originally designed for segmentation in 2D, it quickly became clear that it could be generalized to 3D [18], and also applied to other tasks. It is particularly attractive for modeling surfaces from point clouds [19, 145]. Geodesic contours were also employed for 2D detection and tracking of moving objects [91]. Also well analyzed in theory is how to employ minimal surfaces for 3D reconstruction of static objects from multiple views [31]. This technique

[1] At this point, a remark for the expert is necessary. Our result differs in the case $k = 2$ from previously reported ones in the way that it is considerably simpler, because terms depending on $\langle \Phi_{\mathbf{n}}, \mathbf{n} \rangle$ are missing. The reason for this is that we discovered this product to be *analytically zero*, which we also prove in Sect. 8.3

was recently extended to simultaneously estimate the radiance of surfaces, and demonstrated to give good results in practice [53].

We will briefly review the above methods to demonstrate that all of them fit into our framework. In particular, our theorem applies to all of them and yields the correct surface evolution equations.

## 7.3.1 Segmentation via Geodesic Active Contours

Gazelles, Kimmel and Sapiro realized that the energy which is minimized in the classical *snakes* approach [59] can be rewritten in terms of a geodesic computation in a Riemannian space by means of Maupertuis' Principle. The goal is to compute a contour curve $\mathcal{C}$ in an image $\mathcal{I}$ which is attracted by edges in the image while remaining reasonably smooth. Their final energy functional took the form

$$\mathcal{A}(\mathcal{C}) := \int_{\mathcal{C}} g \circ |\nabla \mathcal{I}| \, ds,$$
$$\text{where } g : \mathbb{R}^+ \to \mathbb{R}^+ \text{ is strictly decreasing with } \lim_{r \to 0} g(r) = 0.$$

It is of the desired form (7.1) in dimension $k = 2$, so (7.2) gives the correct Euler-Lagrange equation. $\nabla \mathcal{I}$ acts as an edge detector, while $g$ controls how image gradients are interpreted as energies. The main purpose of $g$ is to act as a *stopping function*: The flow of the curve should cease when it arrives at object boundaries. Because the integral is minimized, the contour will move towards regions of high gradient. The smoothness requirement is enforced by the curvature term in equation (7.2). Note that $g \circ |\nabla \mathcal{I}|$ depends only on the surface point and not on the normal, so the rightmost term in the Euler-Lagrange equation vanishes.

Essentially the same functional can be applied to 3D segmentation [18], where the source image $I$ is replaced by a volumetric set of data, and the unknown curve $\mathcal{C}$ by an unknown 2D surface.

## 7.3.2 Tracking

Paragios and Deriche combine geodesic active contours and a motion detection term in a single energy functional to track moving objects in a sequence of images [91]:

$$\mathcal{A}(\mathcal{C}) := \int_{\mathcal{C}} \gamma \underbrace{G_{\sigma_D} \circ \mathcal{I}_D}_{Motion} + (1 - \gamma) \underbrace{G_{\sigma_T} \circ |\nabla \mathcal{I}|}_{Contours} ds,$$

where $G_\sigma$ is a Gaussian with variance $\sigma$. The user-defined parameter $\gamma$ weights the influence of the motion detection term against the boundary localization. The Gaussians play the same role as $g$ in geodesic contours, their variances $\sigma_T$ and $\sigma_D$ are derived from the image statistics. The image $\mathcal{I}_D$ is designed to detect boundaries of moving regions in the current image $\mathcal{I}$ of the sequence, and constructed using a Bayesian model which takes into account the pixel differences to the previous frame.

### 7.3.3 Surface Modeling from Unstructured Data Sets

Let $S \subset \mathbb{R}^3$ be a general set of points in space which is to be approximated by a surface $\Sigma$. The following functional was proposed in [145] and [146]:

$$\mathcal{A}(\Sigma) := \int_\Sigma d_S^p \, dA,$$

where $d_S : \mathbb{R}^3 \to \mathbb{R}^+$ computes the distance of points to the data set.

That way, points in the data set attract the surface. In regions with high sampling density, the curvature term is more easily outweighed by the distance function, so the surface becomes more flexible in these regions and more rigid where sampling density is low. This is a desired property of the regularization term. The parameter $1 \leq p \leq \infty$ also influences the flexibility to some extent.

### 7.3.4 3D Reconstruction

As a first step, Faugeras and Keriven [31] give a simple functional in dimension $n = 3$ for static 3D scene reconstruction which does not depend on the surface normal. It can be viewed as a space-carving approach [64] generalized from discrete voxels to a continuous surface model.

Let $C_1, \dots, C_l$ be a number of cameras which project a scene in $\mathbb{R}^3$ onto images $\mathcal{I}_k$ via projections $\pi_k : \mathbb{R}^3 \to \mathbb{R}^2$. For each point $s \in \mathbb{R}^3$, let $\nu_k(s)$ denote whether or not $s$ is visible in camera $k$ in the presence of a surface $\Sigma$. $\nu_k(s)$ is defined to be one if $s$ is visible, and zero otherwise. For simplicity we must assume that $\nu$ does not change with respect to local variations of $\Sigma$, which is physically not entirely true. A measure of how good a surface $\Sigma$ as a model of the scene geometry really is in accordance with a given set of images can be obtained as follows: Each surface point is projected into the set of images where it is visible, and the differences between the pixel colors for each pair of images are computed and summed up to get an error measure for the surface point. This error is integrated over the surface to get the total error. In mathematical notation,

$$\begin{aligned}\mathcal{A}(\Sigma) &:= \int_{\Sigma} \Phi^S dA, \text{ where} \\ \Phi^S(s) &:= \frac{1}{V_s(V_s-1)} \sum_{i,j=1}^{l} \nu_i(s)\nu_j(s) \cdot \|\mathcal{I}_i \circ \pi_i(s) - \mathcal{I}_j \circ \pi_j(s)\|_{\infty}.\end{aligned}$$

The number $V_s$ of cameras able to see a point $s$ is used to normalize the function.

Clearly, the above model is too simple to be of much use in multi-view reconstruction, since only single pixels with no regard to their neighborhoods are compared. A better functional was therefore suggested by Faugeras and Keriven, and can be applied using the results on how the evolution depends on the current normals. We present a slight modification of their original approach here. Our functional only depends on invariant surface properties and does not make use of geometric objects in the source camera views.

To each surface point $s$, we associate a small rectangle $\square_{s,\mathbf{n}}$ in the tangent plane $T_s\Sigma$. In order to invariantly determine its orientation within the plane, we align the sides with the principal curvature directions. This rectangle is then projected into the images, and the normalized cross-correlation over the projected areas is computed. We choose the length of the rectangle sides to be inversely proportional to the curvature in the corresponding direction, up to a certain maximum, because the first order approximation of the surface by its tangent plane is valid over a larger region if the curvature is low. The corresponding functional can be written as

$$\begin{aligned}\mathcal{A}(\Sigma) &:= \int_{\Sigma} \Phi^C dA, \text{ where} \\ \Phi^C(s,\mathbf{n}) &:= -\frac{1}{V_s(V_s-1)} \sum_{i,j=1}^{l} \nu_i(s)\nu_j(s) \cdot \chi_{i,j}(s,\mathbf{n}) \text{ and} \\ \chi_{i,j}(s,\mathbf{n}) &:= \frac{1}{A(\square_{s,\mathbf{n}})} \int_{\square_{s,\mathbf{n}}} \left(\mathcal{I}_i \circ \pi_i - \overline{\mathcal{I}}_i^{s,\mathbf{n}}\right) \cdot \left(\mathcal{I}_j \circ \pi_j - \overline{\mathcal{I}}_j^{s,\mathbf{n}}\right) dA.\end{aligned}$$

The correlation integral has to be normalized using the area $A(\square_{s,\mathbf{n}})$ of the square. The mean values are computed using

$$\overline{\mathcal{I}}_i^{s,\mathbf{n}} := \frac{1}{A(\square_{s,\mathbf{n}})} \int_{\square_{s,\mathbf{n}}} \mathcal{I}_i \circ \pi_i \, dA.$$

When this functional is minimized, not only the position, but also the surface normal is adjusted to best match the images. This approach can also be employed to improve the normals for a known geometry approximation, i.e., the visual hull. When a segmentation of the images into background and foreground objects can be obtained, the visual hull also constitutes a good initial surface $\Sigma_0$ for the evolution equation (8.14), since it is by construction a conservative estimate of the object regions.

### 7.3.5 Reflectance Estimation

Jin, Soatto and Yezzi combine the reconstruction framework with a simultaneous reflectance estimation [53]. They use the functional

$$\mathcal{A}(\Sigma) := \int_\Sigma \left\| \tilde{R} - R \right\|_F^2 \, dA,$$

where the Frobenius norm $\|\cdot\|_F$ is employed to compute the difference of the measured radiance tensor field $\tilde{R}$ to an idealized $R$ obtained from a reflection model, which depends on the surface $\Sigma$.

As claimed previously, all of the problems reviewed in this section are of the form required by the main theorem, and can thus be subsumed under the unifying framework presented in this thesis. Before we proceed with explaining our framework, we briefly introduce our recording setup in the remainder of this chapter and sketch how we obtain segmented images and approximate starting volumes.

## 7.4 Acquisition Hardware

All multi-video sequences we use for surface reconstruction and free-viewpoint video were acquired on our in-house system. This system was originally designed and built for online human motion capture by Christian Theobalt [131]. It is capable of performing image processing and even basic 3D reconstruction tasks online. Using a client-server architecture with one client PC per two cameras, we can compute the voxel-based visual hull at a rate of 15 frames per second. Thus, if we combine this system with a real-time rendering backend, we can set up a basic free-viewpoint live capturing and playback system. However, the geometry is hardly perfect, since the visual hull usually exhibits severe artifacts if the number of cameras is as small as in our case. Offline post-processing is therefore recommended.

Our system acquires synchronized video streams via pairs of cameras connected to client PCs, Fig. 7.1. Each client consists of a 1GHz single processor Athlon PC connected to two Sony DFW-V500 IEEE1394 video cameras that run at a resolution of $320 \times 240$ pixels in RGB color mode. For synchronization of the cameras, a control box was built that distributes an external trigger

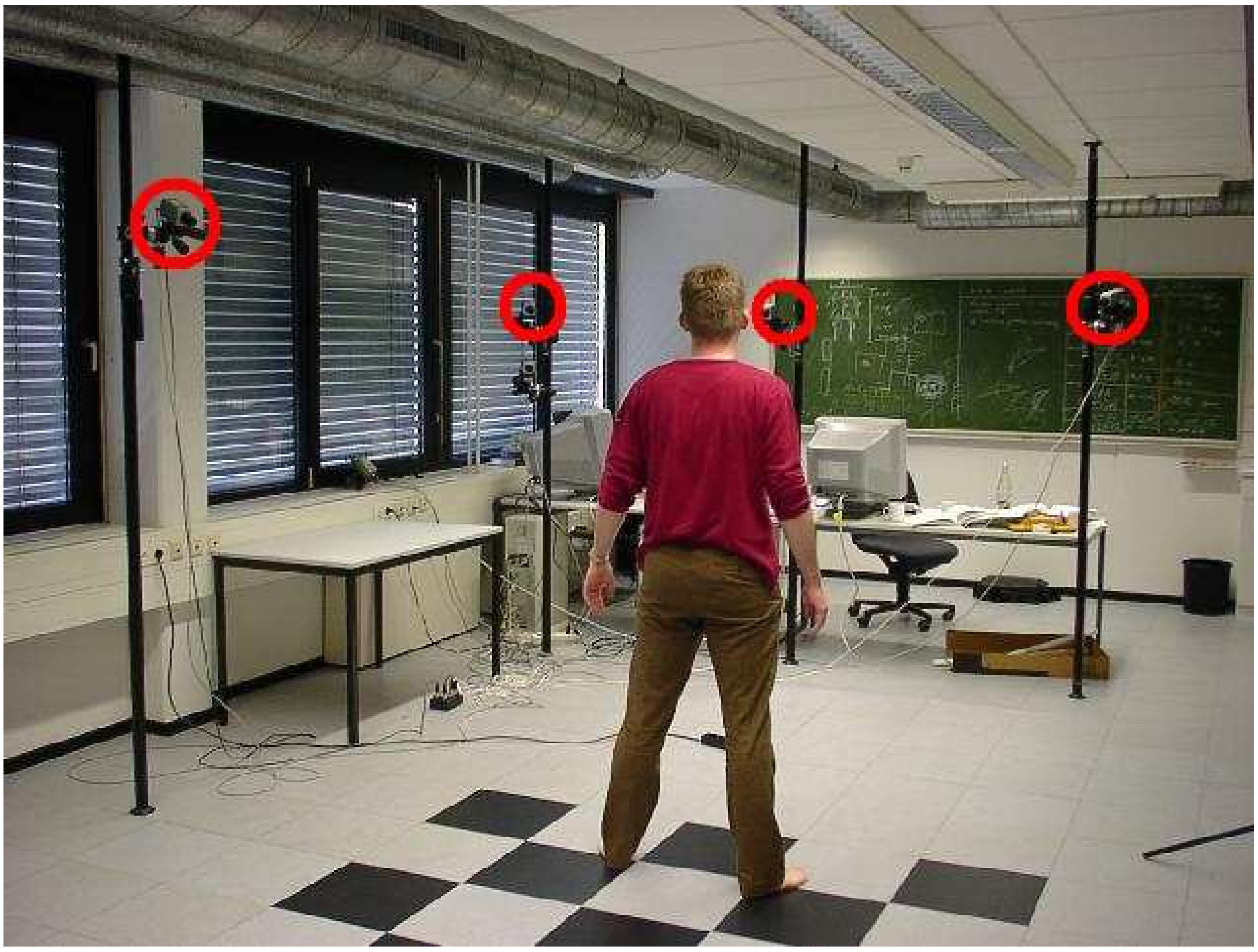


**Fig. 7.1.** A photo of our vision studio. Red circles denote recording camera positions.

pulse from the parallel port of a host computer to all 6 cameras. The clients are connected to a 1 GHz server PC by a 100 MBit/s Ethernet network.

Before recording any data, a calibration procedure based on Tsai's algorithm is applied [134]. The external camera parameters are estimated using a 2x2 m checkerboard calibration pattern, which also defines the bounding box of the voxel volume. The corners of the pattern are detected automatically by employing a sub-pixel corner detection algorithm on the camera images showing the pattern. The internal camera parameters can in theory also be calculated from the large pattern using Tsai's calibration method, but we achieve a more accurate calibration of the internal parameters by using a small checkerboard pattern that is positioned to cover a large part of each camera view.

During actual recording, each of the clients separates the foreground from the known static background of the scene for both of the cameras connected to it. The partial visual hulls are computed from the two silhouettes obtained from the background subtraction. The partial voxel model is then RLE-encoded and transmitted to the server PC, which intersects the par-

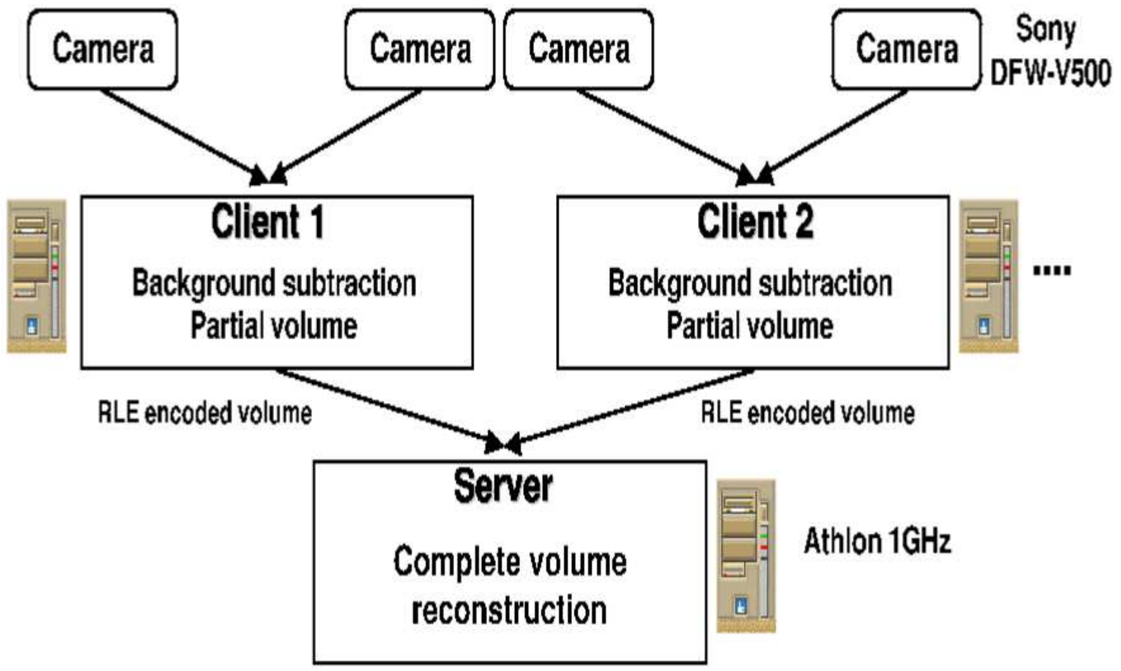


**Fig. 7.2.** *Architecture of the system used for video acquisition and online visual hull reconstruction.* The image-based visual hull is reconstructed online using several clients to compute partial volumes, which are combined on a server into the final voxel model of the scene.

tial models to obtain the final voxel model of the visual hull, Fig. 7.2. The server also sends the trigger signals to the cameras for synchronization. The whole architecture scales easily to more cameras and more clients.

We now describe segmentation and computation of the image-based visual hull in more detail. The background of the scene is static, so we can employ the segmentation method based on image statistics proposed by Cheung et al. [22]. This algorithm computes color mean and standard deviation of each background pixel from a sequence of images with no foreground object present. In the source frames, foreground objects are then identified by a large deviation from their background statistics. A pixel is classified as foreground if its color $p$ differs in at least one color channel by more than an upper threshold factor $\tau$ from its background distribution with mean $\mu$ and standard deviation $\sigma$,

$$\|p(x,y) - \mu(x,y)\|_\infty \geq \tau \cdot \sigma(x,y).$$

One principal problem remains: If no additional criterion is applied, shadow pixels are wrongly classified as foreground. We solve this problem by characterizing shadows as the set of pixels with a large intensity difference compared to the background, but only a small difference in hue. Pixels classified as shadow are removed from the foreground silhouette.

After the background subtraction, the system approximates the visual hull in the form of a cubic binary voxel volume, where a voxel value of 1 indicates that the corresponding scene point might be occupied. We assume that the foreground objects are contained in a cubic bounding box uniformly subdivided into voxels. The computation of the partial visual hull is then performed as follows: Let $C_i, i = 0, \ldots, N$ be a number of cameras. Consider an object which is projected onto the silhouettes $S_i$ by the respective cameras. For each voxel $v$ in the volume and each camera $C_i$, we can therefore compute the projection $v_i$ in the camera's image. Note that because $v$ is a cube, in general $v_i$ is a polygon. If $v_i \subset S_i$ for all $i$, the voxel $v$ is marked as occupied, otherwise it is marked as empty. Two example results are shown in Fig. 2.3(b) in the introduction.

With only slightly more effort, one can achieve a far superior result yielding a more general level set instead of a binary voxel volume. For this, we not only project the center of each voxel into the images when performing the silhouette test, but a number of sample points distributed in the voxel. For the $i$th camera, let $s_i$ equal the percentage of sample points of a voxel $V$ whose projection lies inside the silhouette $S_i$. The value assigned to the level set function $u$ in the grid cell corresponding to $V$ is then set to

$$\min_{1 \leq i \leq n} (s_i).$$

A comparison of this technique with a conventional voxel representation is shown in Fig. 2.4.

The visual hull serves as an approximate starting volume to initialize the following surface reconstruction algorithms. Another alternative is to place a real-time rendering backend directly behind the online visual hull reconstruction stage, for instance the one described in Chapter 13. Thus, we have a fully interactive free-viewpoint video acquisition and rendering system available, however, with low-quality geometry. Means to get far superior geometry offline are introduced in the next chapters.

# 8 Mathematical Foundations

## 8.1 Introduction

In order to compute a weighted minimal surface in practice given an arbitrary error density, some theoretical work is necessary. Previous work did not cover the general case for arbitrary dimension and weight function we aim at. The treatment of this case requires mathematical insights into the differential geometry of hypersurfaces. The aim is to derive a necessary minimality condition, which in this context is usually called the error functional's Euler-Lagrange equation. Our work is organized in a way that the impatient reader not interested in mathematical detail can skip this chapter entirely. Important equations are referenced later when needed.

The correct framework for dealing with minimal surface problems are frame bundles of a variation of the surface. We introduce those and a few of their differential geometric properties in the next section. After all tools have been assembled, we proceed with the derivation of the Euler-Lagrange equation in Sect. 8.3. The equation leads to a surface evolution, which can implemented using a level set technique, as shown in Sect. 8.4. We conclude with a summary in Sect. 8.5.

## 8.2 Some Background in Differential Geometry

Our goal is to give a general proof that surfaces minimizing (7.1) can be obtained as a solution of the Euler-Lagrange equation (7.2) for the energy functional. The mathematical tool of choice is called the *method of the moving frame*. This section is intended to give a brief overview of this framework.

Any minimal surface $\Sigma$ of the functional $\mathcal{A}$ is a *critical point* of the functional, i.e., in first order the value of the functional does not change under a small variation of the surface. This restriction is known as the functional's *Euler-Lagrange equation.* We are now going to give a, necessarily brief, overview of the mathematical framework in which this equation can be derived. For an excellent and thorough introduction, the reader is referred to [26].

We have to investigate how the functional behaves with respect to first order variations of the surface. To this end, let

$$X : \Sigma \times (-\epsilon, \epsilon) \to \mathbb{R}^n$$

be a variation of $\Sigma$ with compact support, then for each $\tau \in (-\epsilon, \epsilon)$ a regular surface $\Sigma_\tau \in \mathbb{R}^n$ is given by $X(\Sigma, \tau)$. For each $(s, \tau) \in \Sigma \times (-\epsilon, \epsilon)$, let

$$\{\mathbf{e}_1(s,\tau), \dots, \mathbf{e}_n(s,\tau) =: \mathbf{n}(s,\tau)\}$$

be an orthonormal frame for the surface $\Sigma_\tau$ at $s$ with $\mathbf{e}_n = \mathbf{n}$ normal to the tangent plane $T_s\Sigma_\tau$. The restrictions $\omega^i$ of the Maurer-Cartan forms of $\mathbb{R}^n$ to this frame are defined by

$$dX \;=\; \mathbf{e}_i\, \omega^i. \tag{8.1}$$

Throughout this text we use the Einstein convention for sums, which means that we implicitly compute the sum from 1 to $n$ over all indices appearing twice on the same side of an equation. Because the frame is adapted to $\Sigma_\tau$ in the above sense, the forms $\omega^1$ to $\omega^k$ are its usual dual forms on the surface. The *connection 1-forms* $\omega_i^j$ are defined by

$$d\mathbf{e}_i \;=\; \mathbf{e}_j\, \omega_i^j \tag{8.2}$$

and satisfy the *structure equations*

$$d\omega^i \;=\; -\omega_j^i \wedge \omega^j \qquad d\omega_j^i \;=\; \omega_k^i \wedge \omega_j^k, \tag{8.3}$$

which can be deduced by differentiating the definitions.

From the connection forms stems the true power of this method. They allow us to express derivatives of the frame, in particular of the normal, in terms of objects which are *part of the frame bundle themselves.* This is the one reason why we will never need local coordinates, because all necessary information about the embedding of the surface in space is encoded in the connection forms.

From the Euclidean structure on $\mathbb{R}^n$ it follows that the connection 1-forms are skew-symmetric, $\omega_i^j = -\omega_j^i$. The connection forms $\omega_i^n$ can be expressed in

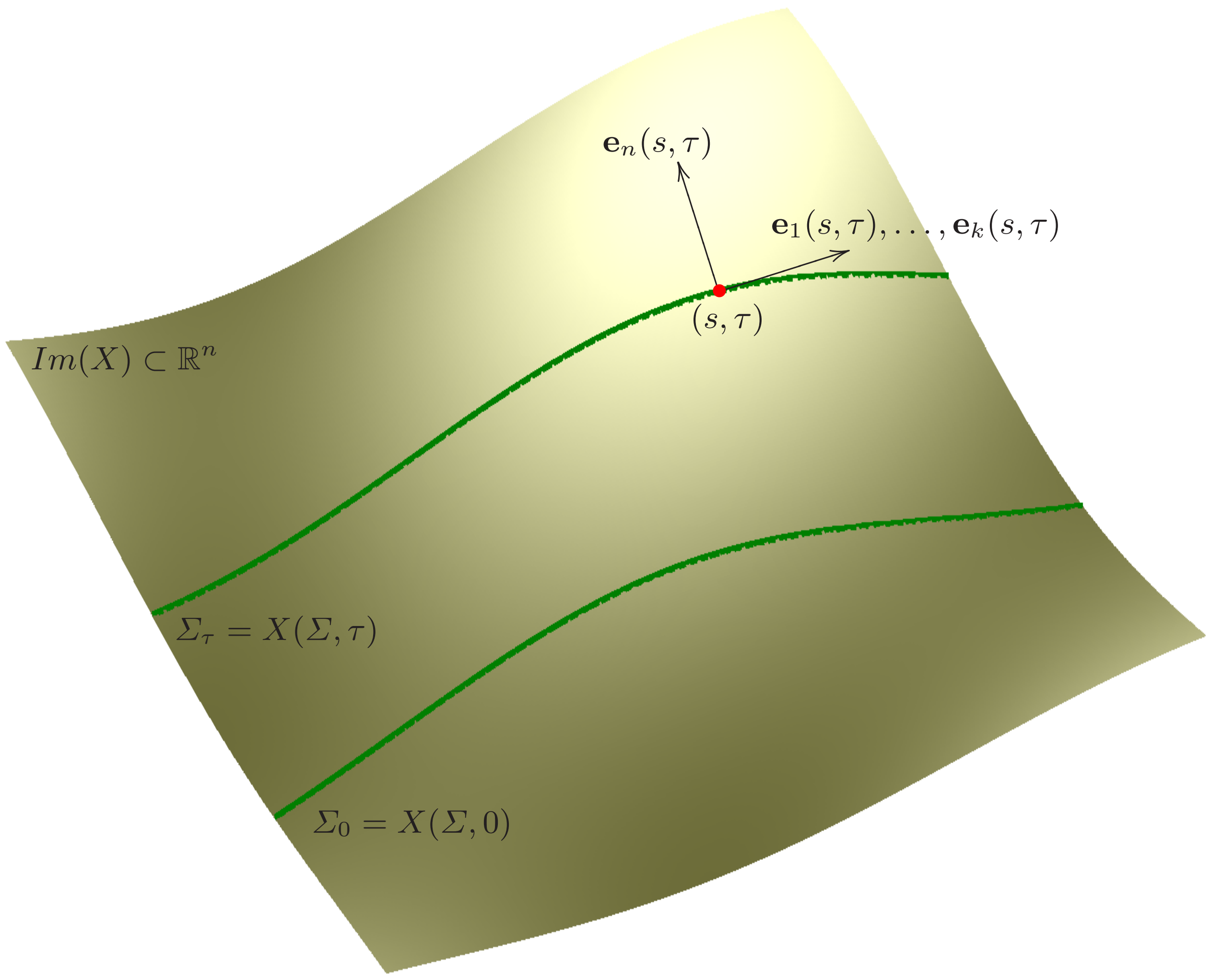

$\mathbf{e}_n(s,\tau)$
$\mathbf{e}_1(s,\tau),\ldots,\mathbf{e}_k(s,\tau)$
$(s,\tau)$
$Im(X) \subset \mathbb{R}^n$
$\Sigma_\tau = X(\Sigma,\tau)$
$\Sigma_0 = X(\Sigma,0)$

**Fig. 8.1.** *The frame bundle of a variation.*

the base $\{\omega^1,\ldots,\omega^k,d\tau\}$, courtesy of Cartan's Lemma [117]. To see this, first note that because of definition (8.1)

$$\omega^n \;=\; \langle dX,\mathbf{n}\rangle \;=\; \frac{\partial X}{\partial\tau}\,d\tau \;=:\; f\,d\tau. \tag{8.4}$$

Differentiating this equation yields together with (8.3)

$$df\wedge d\tau + \sum_{i=1}^{k}\omega_i^n\wedge\omega^i \;=\; 0,$$

therefore, by Cartan's Lemma, there exist functions $h_{ij}$ such that

$$\begin{bmatrix}\omega_1^n\\ \vdots\\ \omega_k^n\\ df\end{bmatrix} = \begin{bmatrix}h_{11} & \ldots & h_{1k} & f_1\\ \vdots & \ddots & \vdots & \vdots\\ h_{k1} & \ldots & h_{kk} & f_k\\ f_1 & \ldots & f_k & f_n\end{bmatrix}\begin{bmatrix}\omega^1\\ \vdots\\ \omega^k\\ d\tau\end{bmatrix}. \tag{8.5}$$

The top-left part $\mathbf{S} := (h_{ij})$ of this matrix is called the *shape operator*, and is closely related to the curvature of $\Sigma_\tau$. In the lower dimensional cases, its entries are commonly known as follows:

- If $k = 1$, i.e. $\Sigma_\tau$ is a curve in $\mathbb{R}^2$, the sole coefficient $h_{11}$ equals the scalar valued curvature usually denoted by $\kappa$.
- If on the other hand $k = 2$, i.e. $\Sigma$ is a regular surface in $\mathbb{R}^3$, the entries of $\mathbf{S}$ are the coefficients of the second fundamental form of $\Sigma_\tau$. More precisely,

$$II \;=\; \begin{bmatrix}\omega^1 & \omega^2\end{bmatrix} \mathbf{S} \begin{bmatrix}\omega^1 \\ \omega^2\end{bmatrix} \;=\; h_{11}(\omega^1)^2 + 2h_{12}\omega^1\omega^2 + h_{22}(\omega^2)^2.$$

  Thus $H = \frac{1}{k}\mathrm{Tr}\,(\mathbf{S}) = \frac{1}{k}\sum_{i=1}^{k} h_{ii}$ is the mean curvature of the surface.

The $f_i$ are just the directional derivatives of $f$ in the directions of the $\mathbf{e}_i$. Using the structure equations (8.3), we immediately deduce an important relation for the area form $dA$ on $\Sigma_\tau$:

$$dA =: \omega_A = \omega^1 \wedge \ldots \wedge \omega^k \;\Longrightarrow\; d\omega_A \;=\; -\mathrm{Tr}\,(\mathbf{S})\; \omega_A \wedge \omega^n. \tag{8.6}$$

We introduce the notation $\omega_A$ to remind the reader of the fact that the area element $dA$ indeed is a differential form of degree $k$. Note that area in our sense does not imply "two-dimensional".

Finally, we need a notion of an 'integration by parts' for a surface integral. First, we generalize the usual operators from vector analysis to vector fields $\mathbf{v}$ and functions $f$ on $\Sigma$:

$$\mathrm{div}_\Sigma(\mathbf{v}) \;:=\; \sum_{i=1}^{k} \frac{\partial v^i}{\partial \mathbf{e}_i} \text{ with the expansion } \mathbf{v} \;=\; v^i\,\mathbf{e}_i, \text{ and}$$

$$\nabla_\Sigma f \;:=\; \sum_{i=1}^{k} \frac{\partial f}{\partial \mathbf{e}_i}\,\mathbf{e}_i \;=\; \sum_{i=1}^{k} f_i e_i.$$

Using the definitions and the product rule, one obtains a generalization of an identity well-known from classical vector analysis,

$$\mathrm{div}_\Sigma(\mathbf{v}f) \;=\; \langle \mathbf{v}, \nabla_\Sigma f\rangle \,+\, \mathrm{div}_\Sigma(\mathbf{v})\, f, \tag{8.7}$$

which will be useful later as one possibility of shifting partial derivatives from one object to another. A second possibility is given by a general form of Gauss' Theorem for surfaces [2], which in our context reads

$$\int_\Sigma \mathrm{div}_\Sigma(\mathbf{v})\, dA \;=\; -\int_\Sigma \mathrm{Tr}\,(\mathbf{S})\,\langle \mathbf{v}, \mathbf{n}\rangle\, dA. \tag{8.8}$$

Note that $\mathbf{v}$ does not have to be tangential to $\Sigma$. Since we assume that all our surfaces are closed, the boundary term usually contributing to the formula has vanished.

We have now collected all the necessary tools to derive the Euler-Lagrange equation of $\mathcal{A}$, and do so in the next section. In Sect. 8.4, this will yield an evolution equation for the level sets of a function on $\mathbb{R}^n$.

## 8.3 Euler-Lagrange Equation

In this section we employ the mathematical framework to derive the Euler-Lagrange equation of the functional $\mathcal{A}$. The arguments can be followed just by abstract manipulation of symbols, without the need to understand all of the reasons which lead to the governing rules presented in Sect. 8.2.

The desired equation characterizes critical points of $\mathcal{A}$, and is given by the derivation of the functional with respect to $\tau$ at $\tau = 0$. We assume that $\Phi = \Phi(s, \mathbf{n})$ is a function of the surface point $s$ and the normal $\mathbf{n}(s)$ at this point. Since $\Phi$ maps from $\mathbb{R}^n \times \mathbb{S}^n$, $\Phi_{\mathbf{n}}(s, \mathbf{n})$ is tangent to the unit sphere of $\mathbb{R}^n$ at $\mathbf{n}$, so we have the important relation $\langle \Phi_{\mathbf{n}}(s, \mathbf{n}), \mathbf{n} \rangle = 0$. This fact was overlooked in previous publications, which is the reason why our final equation is considerably simpler. It seems reasonable to give a more formal proof. Note that $\Phi$ depends only on the *direction* of the normal, which has always unit length, so $\Phi(s, \mathbf{n}) = \Phi(s, \delta\mathbf{n})$ for all $\delta \neq 0$. Since for fixed $s$, $\langle \Phi_{\mathbf{n}}(s, \mathbf{n}), \mathbf{n} \rangle$ is the directional derivative of $\Phi(s, \cdot)$ at $\mathbf{n}$, evaluated in the direction $\mathbf{n}$, it indeed follows directly from the definition that

$$\langle \Phi_{\mathbf{n}}(s, \mathbf{n}), \mathbf{n} \rangle = \lim_{\epsilon \to 0} \frac{1}{\epsilon}(\Phi(s, (1+\epsilon)\mathbf{n}) - \Phi(s, \mathbf{n})) = 0. \tag{8.9}$$

Let us now turn to the computation of the Euler-Lagrange equation. Using the Lie-derivative

$$\mathcal{L}_{\mathbf{v}}\omega = \mathbf{v} \rightharpoonup d\omega + d(\mathbf{v} \rightharpoonup \omega) \tag{8.10}$$

of a differential form $\omega$ in the direction of $\mathbf{v}$, we obtain

$$\begin{aligned}
\left.\frac{d}{d\tau}\right|_{\tau=0} \mathcal{A}(\Sigma_\tau) &\overset{(a)}{=} \int_\Sigma \mathcal{L}_{\frac{\partial}{\partial\tau}} (\Phi\, \omega_A) \overset{(b)}{=} \int_\Sigma \frac{\partial}{\partial\tau} \rightharpoonup d(\Phi\, \omega_A) \\
&\overset{(c)}{=} \int_\Sigma \frac{\partial}{\partial\tau} \rightharpoonup (d\Phi \wedge \omega_A + \Phi\, d\omega_A) \\
&\overset{(d)}{=} \int_\Sigma \frac{\partial}{\partial\tau} \rightharpoonup \big( \langle \Phi_s, \mathbf{e}_i \rangle\, \omega^i \wedge \omega_A + \Phi_{\mathbf{n}}\, d\mathbf{n} \wedge \omega_A - \operatorname{Tr}(\mathbf{S})\, \Phi\, \omega_A \wedge \omega^n \big) \\
&\overset{(e)}{=} \int_\Sigma \left[ (\langle \Phi_s, \mathbf{n} \rangle - \operatorname{Tr}(\mathbf{S})\, \Phi)\, f\, \omega_A + \frac{\partial}{\partial\tau} \rightharpoonup (\Phi_{\mathbf{n}}\, d\mathbf{n} \wedge \omega_A) \right].
\end{aligned} \tag{8.11}$$

The five equalities above are justified by the following arguments:

a. A generalization of the 'Differentiation under the integral'-rule in classic calculus [26].
b. Cartan's rule (8.10) for expressing the Lie derivative and using the fact that $\omega^1(\mathbf{n}) = \cdots = \omega^k(\mathbf{n}) = 0$. Note that $\frac{\partial}{\partial\tau}$ is parallel to $\mathbf{n}$, so this equation also holds for $\frac{\partial}{\partial\tau}$.
c. Product rule for differential forms, note that $\Phi$ is a 0-form.
d. Expansion of $d\Phi = \Phi_s\, dX + \Phi_{\mathbf{n}}\, d\mathbf{n} = \langle \Phi_s, \mathbf{e}_i\rangle\, \omega^i + \Phi_{\mathbf{n}}\, d\mathbf{n}$. Here we inserted the definition (8.1) of the restrictions $\omega^i$. The last term under the integral is due to (8.6).
e. Linearity of the left hook and again $\omega^1(\mathbf{n}) = \cdots = \omega^k(\mathbf{n}) = 0$. From (8.4), it follows that $\omega^n(\frac{\partial}{\partial\tau}) = f d\tau(\frac{\partial}{\partial\tau}) = f$.

We now turn our attention to the second term of the last integral. Inserting the definition (8.2) of the connection 1-forms and afterwards the expansion of the connection forms (8.5) due to Cartan's Lemma, we get

$$\begin{aligned} \frac{\partial}{\partial\tau} \rightharpoonup (\Phi_{\mathbf{n}}\, d\mathbf{n} \wedge \omega_A) \;&=\; \frac{\partial}{\partial\tau} \rightharpoonup \big(\langle \Phi_{\mathbf{n}}, \mathbf{e}_j\rangle\, \omega_n^j \wedge \omega_A\big) \\ = \frac{\partial}{\partial\tau} \rightharpoonup (-\langle \Phi_{\mathbf{n}}, \nabla_\Sigma f\rangle\, d\tau \wedge \omega_A) \;&=\; -\langle \Phi_{\mathbf{n}}, \nabla_\Sigma f\rangle\, \omega_A \\ = \operatorname{div}_\Sigma(\Phi_{\mathbf{n}})\, f\, \omega_A \;-\; \operatorname{div}_\Sigma(\Phi_{\mathbf{n}}\, f)\, \omega_A. \end{aligned} \tag{8.12}$$

In the last equality, we have shifted derivatives using the product rule (8.7). We can finally compute the integral over the left term using Gauss' Theorem (8.8):

$$\int_\Sigma -\operatorname{div}_\Sigma(\Phi_{\mathbf{n}}\, f)\, dA \;=\; \int_\Sigma \operatorname{Tr}(\mathbf{S})\, \langle \Phi_{\mathbf{n}}, \mathbf{n}\rangle\, f\, dA \;=\; 0.$$

It vanishes due to (8.9). When we thus put equations (8.11) and (8.12) together, we see that we have derived

$$\left.\frac{d}{d\tau}\right|_{\tau=0} \mathcal{A}(\Sigma_\tau) = \int_\Sigma \left(\langle \Phi_s, \mathbf{n}\rangle - \operatorname{Tr}(\mathbf{S})\, \Phi + \operatorname{div}_\Sigma(\Phi_{\mathbf{n}})\right) f\, dA.$$

Since for a critical point this expression must be zero for any variation and hence for any $f$, we have arrived at the Euler-Lagrange equation of the functional

$$\langle \Phi_s, \mathbf{n}\rangle \;-\; \operatorname{Tr}(\mathbf{S})\, \Phi \;+\; \operatorname{div}_\Sigma(\Phi_{\mathbf{n}}) \;=\; 0, \tag{8.13}$$

and thus proved our Theorem (7.2).

## 8.4 Corresponding Level Set Equation

Level sets represent an efficient way to implement a surface evolution [90, 23], and are by now a well-established technique with a wide area of applications [115]. We will briefly review the transition from (8.13) to a surface evolution equation followed by one for a level set in this section. For the remainder of the text, let

$$\Psi := \langle \Phi_s, \mathbf{n} \rangle \; - \; \mathrm{Tr}\,(\mathbf{S})\,\Phi \; + \; \mathrm{div}_\Sigma(\Phi_\mathbf{n}).$$

A surface $\hat{\Sigma}$ which is a solution to the Euler-Lagrange equation $\Psi = 0$ is likewise a stationary solution to a surface evolution equation, where $\Psi$ describes a force in the normal direction:

$$\frac{\partial}{\partial \tau}\Sigma_\tau \; = \; \Psi \mathbf{n}. \tag{8.14}$$

If we start with an initial surface $\Sigma_0$ and let the surface evolve using this equation, it will eventually converge to a local minimum of $\mathcal{A}$. Instead of implementing a surface evolution directly, we can make use of the level set idea. We express the surfaces $\Sigma_\tau$ for each parameter value $\tau \geq 0$ as the zero level sets of a regular function

$$\begin{aligned} &u : \mathbb{R}^n \times \mathbb{R}^{\geq 0} \to \mathbb{R}, \; u(\cdot, \tau)^{-1}\{0\} \; = \; \Sigma_\tau, \\ &\text{i.e. } u(s,\tau) = 0 \; \Leftrightarrow \; s \in \Sigma_\tau. \end{aligned} \tag{8.15}$$

We require $u(\cdot,\tau)$ to be positive inside the volume enclosed by $\Sigma_\tau$, and negative on the outside. An immediate consequence is this

**Lemma.** Let $\nabla$ be the gradient operator for the spatial coordinates of $u$. Then we can compute the outer normal and the trace of the shape operator for $\Sigma_\tau$ using

$$\mathbf{n} \; = \; -\frac{\nabla u}{|\nabla u|} \quad \text{and} \quad \mathrm{Tr}\,(\mathbf{S}) \; = \; \mathrm{div}\left(\frac{\nabla u}{|\nabla u|}\right).$$

*Proof.* The relationship for the normal is obvious. By definition, the shape operator is given by $\mathbf{S} := -D\mathbf{n}$ and maps the tangential space $T\Sigma_\tau$ into itself. It indeed follows that

$$\mathrm{Tr}\,(\mathbf{S}) \; = \; \mathrm{Tr}\,(-D\mathbf{n}) \; = \; \mathrm{div}(-\mathbf{n}) \; = \; \mathrm{div}\left(\frac{\nabla u}{|\nabla u|}\right).$$

Note that we consider the normal to be defined on all level sets of $u$. ◇

Taking the derivative of (8.15) with respect to $\tau$ and inserting (8.14), we deduce the evolution equation for $u$ to be

$$\frac{\partial}{\partial \tau} u \; = \; -\left\langle \nabla u, \frac{\partial}{\partial \tau}\Sigma_\tau \right\rangle \; = \; -\langle \nabla u, \mathbf{n} \rangle\, \Psi \; = \; \Psi \; |\nabla u| \,. \tag{8.16}$$

Using the identities

$$\operatorname{div}\left(\Phi \cdot \frac{\nabla u}{|\nabla u|}\right) = \langle \Phi_s, \mathbf{n} \rangle + \Phi \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) \text{ and } \operatorname{Tr}(\mathbf{S}) = \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right)$$

for the curvature of the level sets of $u$ and the definition of $\Psi$, we arrive at the final reformulation of (8.14) in terms of a level set evolution:

$$\frac{\partial}{\partial \tau} u = \left[ -\operatorname{div}\left(\Phi \cdot \frac{\nabla u}{|\nabla u|}\right) + \operatorname{div}_{\Sigma}(\Phi_{\mathbf{n}}) \right] |\nabla u| \,. \tag{8.17}$$

Note that the derivatives of $\Phi$ can be computed numerically. Thus, it is not necessary to compute an explicit expression for them manually, which would be very cumbersome for more difficult functionals. Instead, in an existing implementation of the evolution for a general function $\Phi$, essentially any functional can be plugged in.

## 8.5 Conclusions

Using the mathematical tool of the *method of the moving frame*, we have derived the Euler-Lagrange equations for weighted minimal surfaces in arbitrary dimensions. We allowed for weight functions general enough to cover the variational problems encountered in computer vision research. Previously, existing proofs used local coordinates and were restricted to dimensions two or three, so our approach is more general. As demonstrated by several examples, weighted minimal surfaces lie at the heart of several well-established computer vision techniques. Our result for arbitrarily high dimensions paves the way for new, future research. In particular, we will employ it in a technique designed to achieve temporal coherence in 3D reconstruction from multiple video streams in the next chapter. With a sophisticated error functional, we can also reconstruct refractive, transparent bodies, to be analyzed in Chapter 10.

# 9

# Space-Time Isosurface Evolution

## 9.1 Introduction

In the previous chapter, we gave a mathematical analysis of weighted minimal hypersurfaces in arbitrary dimension and for a general class of weight functions. We derived the Euler-Lagrange equation yielding a necessary minimality condition. Our analysis covers all of the reviewed variational methods employed in computer vision. In this chapter, we present a variational method of a new kind, applying the freedom in dimensionality allowed by Theorem 7.1. A fourth dimension is introduced which represents the flow of time in the video sequence. Our goal is to reconstruct a smooth three-dimensional hypersurface embedded in space-time. The intersections of this hypersurface with planes of constant time are two-dimensional surfaces, which represent the geometry of the scene in a single time instant. Our approach defines an energy functional for the hypersurface. The minimum of the functional is the geometry which optimizes photo-consistency as well as temporal smoothness.

In Sect. 9.2, we will introduce the mathematical foundations of the algorithm and give a rigorous definition of our method in terms of an energy minimization problem. Implementation details are discussed in Sect. 9.3, where we describe our parallel scheme which computes the evolution equation using a narrow band level set method. We also propose algorithms necessary to evaluate the more involved terms of the equation. Results obtained with real-world video data are presented in Sect. 9.4.

## 9.2 Space-time 3D Reconstruction

In this section, we present the mathematical foundations of our 3D reconstruction algorithm. We assume that we have a set of fully calibrated, fixed cameras. The input to our algorithm are the projection matrices for the set of cameras, as well as a video stream for each camera. We want to obtain a smooth surface $\Sigma_t$ for each time instant $t$, representing the geometry of the scene at that point in time. The surfaces shall be as consistent as possible with the given video data. Furthermore, as in reality, all resulting surfaces should change smoothly over time.

### 9.2.1 Mathematical Foundations

To achieve these desirable properties, we do not consider each frame of the sequences individually. Instead, we regard all two-dimensional surfaces $\Sigma_t$ to be subsets of one smooth three-dimensional hypersurface $\mathfrak{H}$ embedded in four-dimensional space-time. From this viewpoint, the reconstructed surfaces

$$\Sigma_t \;=\; \mathfrak{H} \cap \left(\mathbb{R}^3, t\right) \subset \mathbb{R}^3$$

are the intersections of $\mathfrak{H}$ with planes of constant time. Because we reconstruct only one single surface for all frames, the temporal smoothness is intrinsic to our method.

However, we have to take care of photo-consistency of the reconstructed geometry with the given image sequences. We set up an energy functional

$$\mathcal{A}(\mathfrak{H}) \;:=\; \int_{\mathfrak{H}} \Phi \, dA. \tag{9.1}$$

defined as an integral of the scalar valued weight function $\Phi$ over the whole hypersurface. $\Phi = \Phi(s, \mathbf{n})$ measures the photo-consistency error density, and may depend on the surface point $s$ and the normal $\mathbf{n}$ at this point. The larger the values of $\Phi$, the higher the photo-consistency error, so the surface which matches the given input data best is a minimum of this energy functional. The Euler-Lagrange equation for the functional is given by Theorem 7.1, and we demonstrated in Sect. 8.4 how the Euler-Lagrange equation can be solved in practice using a surface evolution equation implemented via the level set method. In the remainder of this section, we present suitable choices for the error measure $\Phi$.

### 9.2.2 Continuous Space-time Carving

We need some additional notation for color and visibility of points in space-time first. Let $t$ denote a time instant, then a time-dependent image $\mathcal{I}_k^t$ is associated to each camera $k$. The camera projects the scene onto the image plane via a fixed projection $\pi_k : \mathbb{R}^3 \to \mathbb{R}^2$. We can then compute the color $c_k^t$ of every point $(s, t)$ on the hypersurface:

$$c_k^t(s) = \mathcal{I}_k^t \circ \pi_k(s).$$

Here, the image $\mathcal{I}_k^t$ is regarded as a mapping assigning color values to points in the image plane.

In the presence of the surface $\Sigma_t$, let $\nu_k^t(s)$ denote whether or not $s$ is visible in camera $k$ at time $t$. $\nu_k^t(s)$ is defined to be one if $s$ is visible, and zero otherwise.

The most basic error measure can now be defined as

$$\Phi^S(s,t) \ := \ \frac{1}{V_{s,t}} \sum_{i,j=1}^{l} \nu_i^t(s)\nu_j^t(s) \cdot \left\| c_i^t(s) - c_j^t(s) \right\| .$$

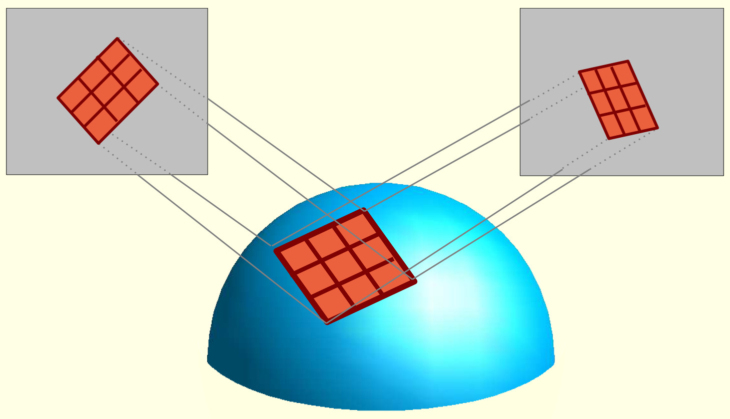
The number $V_{s,t}$ of pairs of cameras able to see the point $s$ at time $t$ is used to normalize the function.

If the error function $\Phi^S$ is used as the functional, the resulting algorithm is similar to a space carving scheme in each single time step. In that method, as introduced by Kutulakos and Seitz [64], voxels in a discrete voxel grid are carved away if $\Phi^S$ lies above a certain threshold value when averaged over the voxel. In our scheme, the discrete voxels are replaced by a continous surface. In the surface evolution introduced later, this surface will move inwards until photo-consistency is achieved. This process is analogous to the carving process [64]. The same functional for regular surfaces in $\mathbb{R}^3$ was introduced by Faugeras and Keriven [31] for static scene reconstruction. As an additional constraint, we enforce temporal coherence in the form of temporal smoothness of the resulting hypersurface, which makes our method ideal for video sequences.

### 9.2.3 Normal Optimization

Because the theorem also allows for error functions which may depend on the normal, we can take the scheme one step further to include an optimization for the surface normals as well. A similar idea was also presented in [31], however, we give a slightly modified version and still work in space-time to enforce temporal smoothness.

In order to set up an error function, we have to analyze how well a small surface patch at position $s$ with a given normal $\mathbf{n}$ fits the images at time $t$. To this end, we assign to each of these values a small patch $\square_{s,\mathbf{n}}$ within the plane orthogonal to $\mathbf{n}$, Fig. 9.1. How exactly this patch is chosen does not matter, however, the choice should be consistent over time and space and satisfy a few conditions which will become evident soon. In our implementation, we always choose rectangular patches rotated into the target plane by a well-defined rotation.



A small grid in the tangent plane is projected in both camera images. This leads to two columns of data containing the pixel colors of the grid corners in the respective video image. Between these two columns of data, the normalized cross correlation is computed.

**Fig. 9.1.** *Practical computation of the cross-correlation error term* $\varPhi^G$.

We will now define a measure how well the patch $\Box_{s,\mathbf{n}}$ is in accordance with the images at time $t$. For that, we employ the normalized cross-correlation of corresponding pixels in the images, a well-established matching criterion in computer vision. Mathematically, the resulting functional for a point $x = (s,t) \in \mathbb{R}^4$ with normal direction $\mathbf{n}$ is defined as follows:

$$\varPhi^G(x,\mathbf{n}) \;:=\; -\frac{1}{V_{s,t}} \sum_{i,j=1}^{l} \nu_i^t(s)\nu_j^t(s) \cdot \frac{\chi_{i,j}^t(s,\mathbf{n})}{A\left(\Box_{s,\mathbf{n}}\right)}$$

with the zero-mean cross-correlation

$$\chi_{i,j}^t(s,\mathbf{n}^t) \;:=\; \int_{\Box_{s,\mathbf{n}^t}} \left(c_i^t - \overline{\mathcal{I}}_i^{x,\mathbf{n}}\right)\left(c_j^t - \overline{\mathcal{I}}_j^{x,\mathbf{n}}\right) \, dA,$$

and the mean color value of the projected patch in the images computed according to

$$\overline{\mathcal{I}}_i^{x,\mathbf{n}} \;:=\; \frac{1}{A\left(\Box_{s,\mathbf{n}}\right)} \int c_i^t \, dA.$$

Some things have to be clarified. First, the correlation measure $\chi_{i,j}^t$ for a pair of cameras is normalized using the area $A\left(\Box_{s,\mathbf{n}}\right)$ of the patch. Second, it is now clear that we have to choose $\Box_{s,\mathbf{n}}$ sufficiently large so that it is projected onto several pixels. On the other hand, it should not be so large that only parts of it are visible in the images. As a compromise, we choose its diameter to be equal to the cell diameter of the underlying computation grid, as defined in Sect. 9.3. Third, the integration of $\Phi^G$ in the energy functional involves the normals of $\mathfrak{H}$ in 4D space, while $\mathbf{n}$ is supposed to lie in $\mathbb{R}^3$. For that reason, we project normals of $\mathfrak{H}$ into the tangent space of $\Sigma_t$ in order to get $\mathbf{n}$.

When this functional is minimized, two constraints are optimized simultaneously. Each surface $\Sigma_t$ together with its normals is selected to best match the images at that time instant. Furthermore, a smooth change of the surfaces $\Sigma_t$ with time is encouraged because of the curvature term in the Euler-Lagrange equation. The error functional can be minimized using a surface evolution implemented via a level set scheme, as derived in the Sect. 8.4. In the next section, we discuss the implementation details involved when the evolution equation is to be solved numerically.

## 9.3 Parallel Implementation

In order to implement the level set evolution equation (8.17), the volume surrounding the hypersurface $\mathfrak{H}$ has to be discretized. We use a regular four-dimensional grid of evenly distributed cells with variable spatial resolution of usually $64^3$ or $128^3$ cells. The temporal resolution is naturally equal to the number of frames in the input video sequences. One easily calculates that there is a massive amount of data and computation time involved if the sequence is of any reasonable length. In fact, it is currently not yet possible to store the full data for each grid cell together with all images of all video sequences within the main memory of a standard PC. A parallel implementation where the workload and data is distributed over several computers is therefore mandatory.

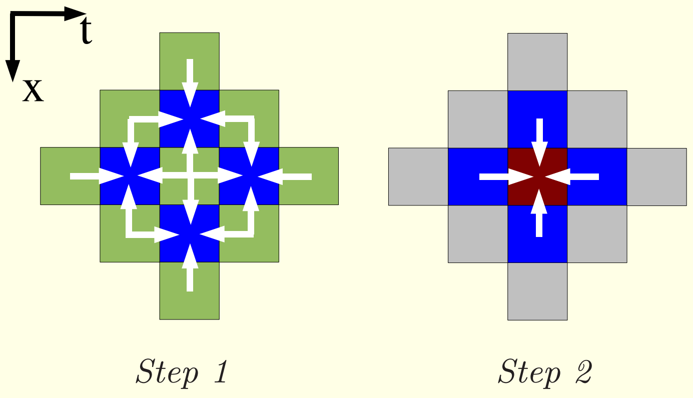
For that reason, we choose the narrow band level set method [115] to implement the evolution equation because it is straightforward to parallelize. We start with an initial surface $\mathfrak{H}_0$ and the values $u_0^{xyzt}$ of the corresponding level set function $u_0$ in the centers of the grid cells. A suitable initial surface for our case will be defined at the end of this section. Using the abbreviation

$$\Psi(u) := \left[ -\operatorname{div}\left( \Phi \cdot \frac{\nabla u}{|\nabla u|} \right) + \operatorname{div}_\Sigma(\Phi_{\mathbf{n}}) \right],$$

Eq. (8.17) simply reads

$$\frac{\partial}{\partial \tau} u = \Psi(u) \, |\nabla u| \, .$$

In the discretization, the values of the level set function are updated iteratively using the upwind scheme. At iteration step $i+1$, the new values $u_{i+1}^{xyzt}$ are



- In the first step, the values of $u_i$ in the green cells are used to compute the level set normal $\mathbf{n} \in \mathbb{R}^4$ in the blue cells using central differences. Having computed $\mathbf{n}$, we can also compute $\Phi$ for the blue cells. Note that for the purpose of the above 2D illustration, the three spatial dimensions are represented as one.
- For the second step, we compute the values for the central red cell, also using finite differences. The discrete formula for $\operatorname{div}(\Phi\mathbf{n})$ at position $p = (x, y, z, t)$ is

$$\sum_{i=1}^{4} \frac{\Phi^{p+e_i} n_i^{p+e_i} - \Phi^{p-e_i} n_i^{p-e_i}}{2}.$$

  We can also compute the curvature $\operatorname{Tr}(\mathbf{S})$ directly by omitting $\Phi$ in the above formula.
- The difficult part is to compute $\operatorname{div}_\Sigma(\Phi_\mathbf{n})$ for the red cell. It is equal to the trace of $\Phi_{\mathbf{n}s}$, restricted to the tangent plane $\Pi$ orthogonal to the normal at $p$. So we first compute $\Phi_\mathbf{n}$ for the blue cells using finite differences, taking the known normal $\mathbf{n}$ of the cell as the center point. With these values, we can set up the $4 \times 4$ matrix $U := \Phi_{\mathbf{n}s}$ for the red cell. We then choose an arbitrary orthonormal base $\{t_0, t_1, t_2\}$ of the plane $\Pi$. The entries for the $3 \times 3$ matrix $V$ of the mapping $\Phi_{\mathbf{n}s}|_\Pi$ can then be computed as

$$v_{ij} = t_i^T U t_j, 1 \le i, j \le 3.$$

**Fig. 9.2.** *Evaluation of the differential operator.*

obtained from the values $u_i^{xyzt}$ of the previous iteration step by a discrete version of equation (8.17) using an explicit time step:

$$u_{i+1}^{xyzt} = u_i^{xyzt} + \Psi\left(u_i^{xyzt}\right) |\nabla u_i| \cdot \Delta\tau. \qquad (9.2)$$

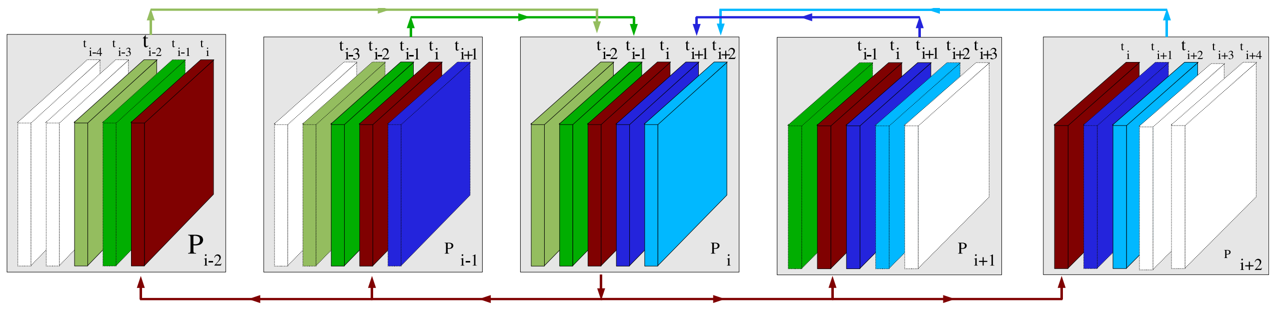

$t_{i-4}$ $t_{i-3}$ $t_{i-2}$ $t_{i-1}$ $t_i$
$P_{i-2}$
$t_{i-3}$ $t_{i-2}$ $t_{i-1}$ $t_i$ $t_{i+1}$
$P_{i-1}$
$t_{i-2}$ $t_{i-1}$ $t_i$ $t_{i+1}$ $t_{i+2}$
$P_i$
$t_{i-1}$ $t_i$ $t_{i+1}$ $t_{i+2}$ $t_{i+3}$
$P_{i+1}$
$t_i$ $t_{i+1}$ $t_{i+2}$ $t_{i+3}$ $t_{i+4}$
$P_{i+2}$

**Fig. 9.3.** *Data transmission of process $P_i$ for one iteration step. Each process stores five slices, corresponding to the geometry at five different time steps. It is responsible for the computation of the center slice. The slice computed by $P_i$ in the last iteration is transmitted to the four adjaced processes. In return, $P_i$ receives the other slices from its neighbours for the next iteration. In the figure, slices of the same color correspond to the same time instant and contain the same information.*

Here, $\Psi\left(u_i^{xyzt}\right)$ is the value of the discretized version of the differential operator $\Psi$ acting on $u_i$, evaluated in the cell $(x, y, z, t)$. Central differences on the four-dimensional grid are used to compute the derivatives involved in Eq. 9.2. The norm of the discretized gradient $|\nabla u_i|$ is calculated according to the up-wind scheme [115]. To ensure stability, the step size $\Delta\tau$ must be chosen such that the level sets of $u_i$ cannot cross more than one cell at a time, i.e. satisfy the CFL-condition

$$\Delta\tau \;\leq\; \max_{(x,y,z,t)\in\Gamma}\left(\frac{\text{diam cell}(x,y,z,t)}{\left|\Psi\left(u_i^{xyzt}\right)\cdot\nabla u\right|}\right). \tag{9.3}$$

The differential operator must be evaluated for each grid cell near the zero level set, and the computations that are necessary for each cell depend only on a local neighbourhood. Therefore, the computation of individual cells can easily be distributed over several processes. In our implementation, each process is responsible for the computation of one single slice of the grid of constant time $t_i$. This slice corresponds to the geometry of the $i$th frame of the video sequence. Fig. 9.2 shows in more detail how the value $\Psi\left(u_i^{xyzt}\right)$ is numerically evaluated from the values of $u_i$ in the grid cells. According to this figure, we need the values of grid cells up to two cells apart from $(x, y, z, t)$ in order to evaluate the operator. As a consequence, each process $P_i$ also has to know the slices of the four other processes $P_{i\pm1}, P_{i\pm2}$. These have to be communicated over the network. In addition, each process needs to store the image data of its own video frame and the two adjacent frames according to Fig. 9.2.

To summarize, one full iteration consists of the following four steps:

- Each process transmits its own slice $S_i$ to the adjacent processes and receives the other necessary slices from its four neighbours according to Fig. 9.3.
- Afterwards, each process computes $\Psi\left(u_i^{xyzt}\right)$ for all cells in its slice near the zero level set of $u_i$, using the scheme presented in Fig. 9.2.
- The maximum value of the operator for each process is transmitted to a special server process. From these maxima, the server calculates the optimal step size $\Delta\tau$ allowed by the inequality (9.3).
- The server broadcasts the maximum to all processes, which afterwards compute the evolution on their slice using equation (9.2).

After each iteration, the server process may poll the current geometry from any of the other processes in order to give the user feedback about the current state of the iteration. The iteration stops when the flow field is zero, or may be stopped by the user if the current geometry looks well enough for the purpose. In our final implementation, it is also possible to assign several processes to a single slice. In that case, they share the computation of the cells near the zero level set equally between each other, assuming that all processes run on similar machines.

We finally have to define an initial surface suitable $\mathfrak{H}_0$ to start the iteration process. For this purpose, we employ the visual hull, which by definition is always a superset of the correct scene geometry. In order to compute a level set representation, we have to choose suitable values of $u_0$ for each grid cell. For this purpose, we fix a grid cell $c$ and select a number of evenly distributed sample points $x_0, \ldots, x_k$ inside it. These points are projected into each source image, and we compute the percentage $p \in [0,1]$ of the projections which fall into the silhouettes of the object to be reconstructed. To the initial level set function $u_0$ is then assigned the value $2p-1$ at cell $c$. Since we only have to compute an approximate starting surface, this straightforward method gives sufficiently good results in practice. In particular, the projection of the zero level set of $u_0$ into the source images very closely resembles the silhoettes of the object if $k$ is sufficiently high.

## 9.4 Results

In order to test our algorithm, we run it on real-world $320 \times 240$ RGB video sequences of a ballet dancer. All input images are segmented into foreground and background using a thresholding technique. Consequently, we can compute the refined visual hull to get a starting volume for the PDE evolution, Fig. 9.4. For our test runs, we choose a 20 frame long part of the sequence with the depicted frame in the middle. As becomes apparent in Fig. 9.5, this

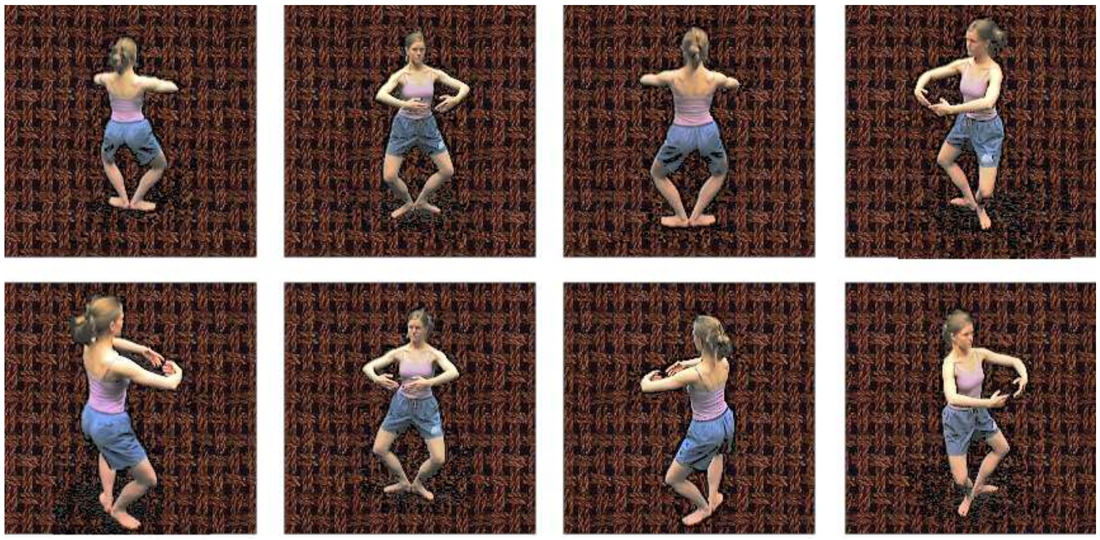
| Grid res. | # procs. | Time per iteration [s] without n.o. | with n.o. | Memory per proc. |
|---|---|---|---|---|
| $32^3$ | 60 | 0.9 | 25 | 80 MB |
| | 40 | 1.4 | 38 | |
| | 20 | 2.5 | 60 | |
| $64^3$ | 60 | 7 | 140 | 180 MB |
| | 40 | 11 | 210 | |
| | 20 | 17 | 360 | |
| $128^3$ | 60 | 30 | 510 | 535 MB |
| | 40 | 55 | 840 | |
| | 20 | 102 | 1200 | |

**Table 9.1.** *Time required for a single iteration step, depending on the resolution of the computation grid and the number of processors. Both the time with and without the normal optimization based on normalized cross-correlations is given.*

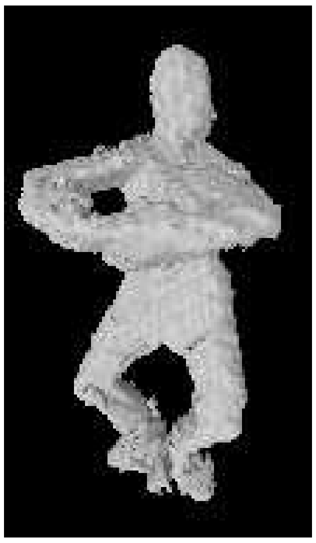
frame is particularly difficult to reconstruct, because we do not have a camera capturing the scene from above. For that reason, most of the area in between the arms of the dancer is not carved away in the initial surface.

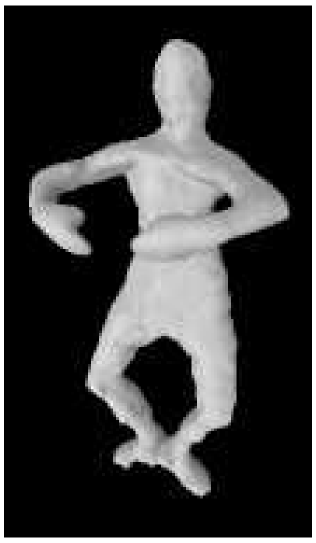
When we run a standard space-carving algorithm for this single frame alone, the situation improves. The shirt of the dancer contains not much texture information, however, so only part of the critical region is carved away as it should be. Only when we employ the full algorithm which takes into account temporal coherence between the geometry of the frames do we get the satisfactory result in Fig. 9.5 on the right. In Fig. 9.6, we show some more geometry results for several time slices of the sequence a few frames apart, and from several novel viewpoints. When textured with an image-based rendering algorithm using the source images, Fig. 9.7, it can be observed that the photo-consistency is indeed excellent.

Table 9.1 informs about the time and memory required by each of the slave processes for a single iteration. Our program ran on a Sun Fire 15K with 75 UltraSPARC III+ processors at 900 MHz, featuring 176 GBytes of main memory. It can be observed that the normal optimization requires a lot of computation time when compared to the standard version of our algorithm. For that reason, we first let the geometry evolve towards a surface which is very close to an optimal result, as assessed by the operator of the program. Afterwards, we switch on the normal optimization in order to improve the reconstruction of small surface details. In average, we need around one hundred iterations of the initial evolution and twenty more of the normal optimization until the surface has converged to the final result.



(a) Source silhouettes

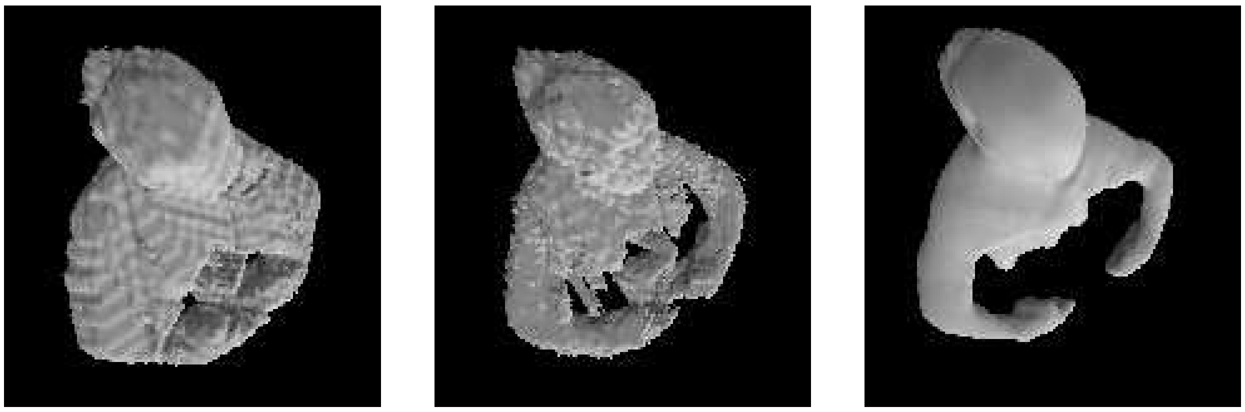




(b) Visual hull (c) Final result

**Fig. 9.4.** Top: *All eight source silhouettes from the cameras distributed around a ballet dancer. This is one single frame of a dynamic video sequence.* Below: *Seen on the left is the refined visual hull, representing the initial surface used to start the PDE evolution, on the right the final result after running the complete algorithm including normal optimization.*

In order to speed up the surface evolution, a further term is included in equation (9.2), as suggested in [31]. We subtract a multiple $\epsilon \mathrm{Tr}\,(\mathbf{S})$ of the curvature, where $\epsilon$ is a small user-defined constant factor. This forces the resulting hypersurface to be smoother, so larger steps $\Delta\tau$ can be taken to evolve the PDE.



(a) Visual hull (b) Space carving result (c) Weighted minimal surface

**Fig. 9.5.** *Comparison of different reconstruction schemes at a grid resolution of* $128^3$. (a) The visual hull, as seen from above. Since we do not have a camera capturing the scene from above, most voxels in the area between the arms are occluded and remain occupied. (b) The result obtained from static space carving. (c) When our algorithm using temporal information is employed, the reconstruction becomes almost optimal.

## 9.5 Conclusions

We have presented a novel 3D reconstruction algorithm which takes into account all frames of a multi-video sequence simultaneously. The idea is to optimize photo-consistency with all given data as well as temporal smoothness. Our method is formulated as a weighted minimal surface problem posed for a 3D hypersurface in space-time. Intersecting this hypersurface with planes of constant time gives the 2D surface geometry in each single time instant. The energy functional defining the minimization problem enforces photo-consistency, while temporal smoothness is intrinsic to our method. The functional can be minimized by implementing a surface evolution PDE using the narrow band level set method. Significant improments compared to space carving approaches which do not take temporal coherence into account can be observed in the results. In the future, we plan to include a global optimization of surface reflectance properties into the same unifying framework.

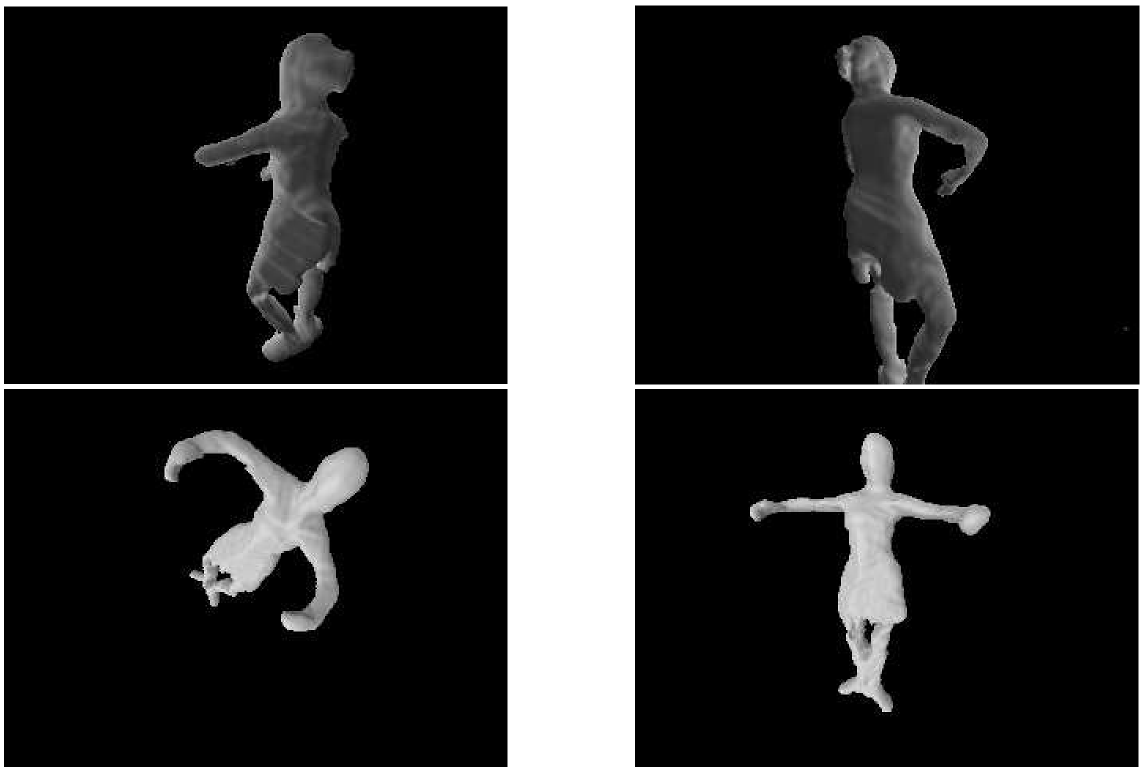


**Fig. 9.6.** *Reconstructed geometry for four more frames in the sequence. All viewpoints shown are far away from the original viewpoints of the source cameras.*

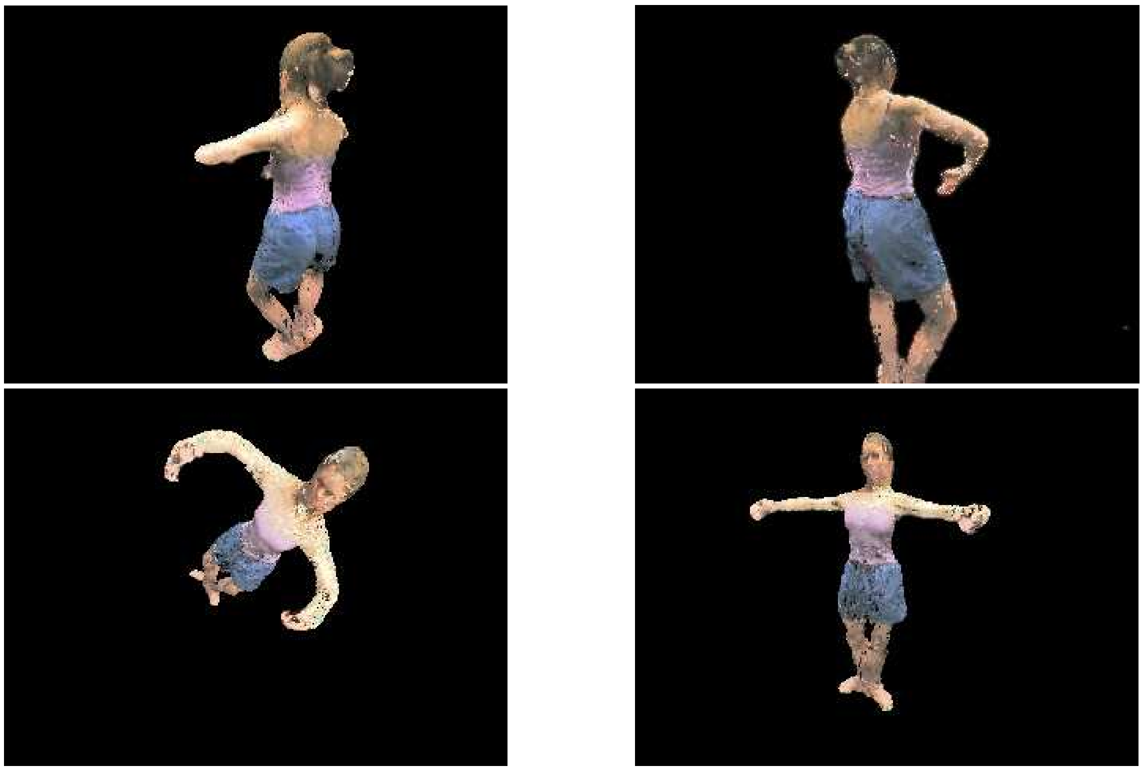


**Fig. 9.7.** *The same geometry rendered with our billboard rendering algorithm presented in Chapter 13 using the source images as textures.*

# 10

# Reconstructing the Geometry of Flowing Water

## 10.1 Introduction

Recently, new multi-view reconstruction problems, different from the traditional diffuse surface reconstruction, have emerged in the field of computer vision. These include multi-view reconstruction of time-varying, transparent, natural phenomena like fire and smoke [51, 49, 1].

The work so far concentrates on non-refracting media. In this chapter, we present a level set method for the reconstruction of a time-varying free flowing water surface. This problem arises in the context of free-viewpoint video, where we are concerned with the automatic acquistion of dynamic models for computer graphics purposes. The main problem here is that the surface structure can not be determined with traditional methods due to refraction effects, implying a complex image formation process. We alleviate this problem by dyeing the water with a fluorescent chemical. This allows us to directly measure the thickness of the water volume as a ray passes through it and hits the CCD-chip of the camera. In addition, a sophisticated energy minimization method is utilized for the reconstruction process, which is able to correctly incorporate error functions depending on surface normals. Obviously, this is a vital requirement if one wants to take into account refraction.

Image-based modeling of natural phenomena suitable for free-viewpoint video is performed using sparse view tomographic methods [51, 1] or surface based methods [49]. Reche et al. reconstruct trees from still images [99]. In [92], the geometry of hair is retrieved using a single camera and varying light source positions, exploiting the anisotropic reflectance properties of hair.

Only limited work has been done which directly addresses image-based reconstruction of water. In [85], a time-varying water surface is obtained by analyzing the distortion of a known texture beneath the water surface using

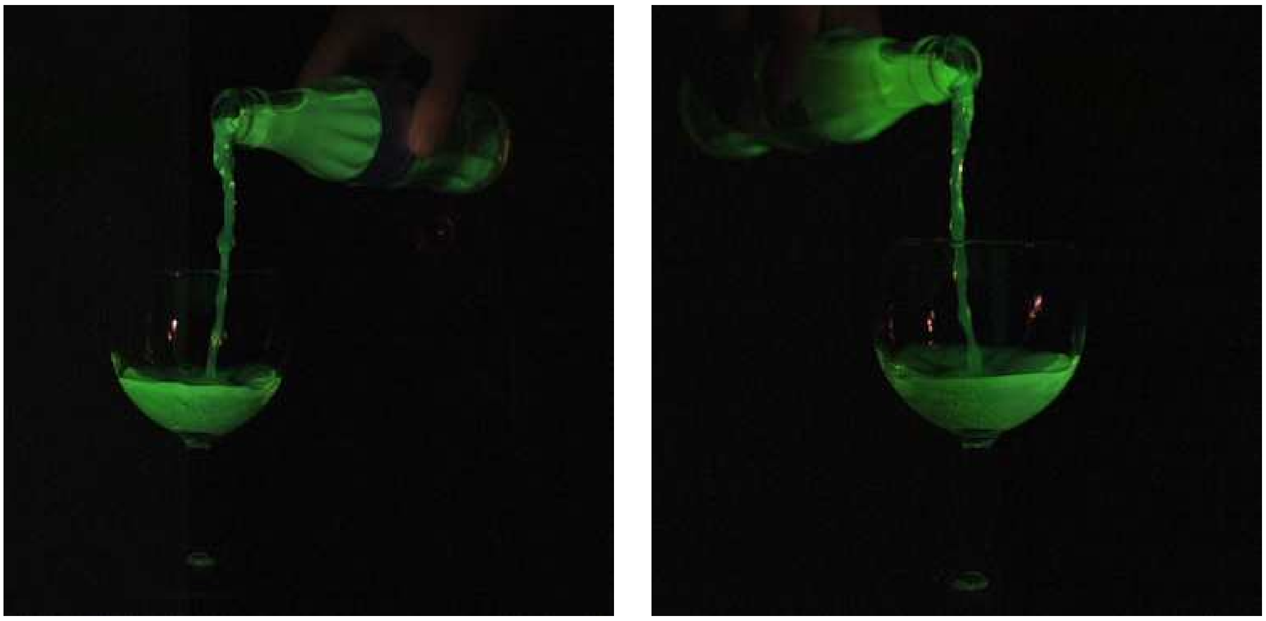


**Fig. 10.1.** Source images from two of the cameras for one frame of our test video sequence in which we pour dyed water from a bottle into a glass under UV-Illumination.

optical flow and shape from shading techniques. Schultz [111] studies the reconstruction of specular surfaces using multiple cameras. He reports good results on synthetic test data, a simulated water surface under known synthetic illumination. However, both of these methods can only determine a height field for a rectangular surface area, while we reconstruct fully three-dimensional bodies of water.

Another line of research is refractive index tomography e.g. [93, 148]. These methods usually need expensive apparatuses and do not lend themselves to image-based modelling. The goal of these methods is also quite different from ours. Whereas refractive index tomography attempts to reconstruct a field of *varying* refractive indices, we know that we have a constant refractive index and need to compute the surface of a volumetric body of water.

PDE based energy minimization methods are very popular in computer vision. Many of them naturally stem from weighted minimal surface approaches, where an error density is integrated over a surface in order to measure how well the surface fits the data [18, 91, 145, 53]. A general method how weighted minimal hypersurfaces can be computed via level set evolution equations in the case that the error function depends on the surface normal was presented in [40]. Earlier, Faugeras and Keriven analyze a special 3D case, where they define an error functional enforcing photo-consistency of recovered 3D geometry with multiple static views of a scene [31].

This chapter is organized as follows. Sect. 10.2 defines the reconstruction problem we want to deal with and presents a mathematical justification for the level set surface flow yielding an optimal solution. Details for the implementation using PDEs are discussed in Sect. 10.3. We present results obtained with both synthetic 2D data as well as recorded 3D data of flowing water in Sect. 10.4, and conclude with ideas for future work in Sect. 10.5.

## 10.2 General Reconstruction Problem

Our goal is to reconstruct the surface area of a possibly moving body of water, using recordings from only a handful of fully calibrated cameras distributed around the scene. In order to be able to work with a well-defined image formation model, special care has to be taken when acquiring the water video data. We employ a fluorescent dye which causes the water to emit visible light when exposed to UV radiation. An example input image from a single frame is shown in Fig. 10.1.

This section embeds the reconstruction problem we want to deal with in a rigorous mathematical framework. Subsection 10.2.1 discusses the image formation model underlying the optimization. It shows how to generate synthetic views given a certain reconstructed surface $\Sigma$, which can be compared to recorded real-world data in order to define a photo-consistency error measure. The 'best' surface is determined by minimizing an error functional optimizing photo-consistency. The functional is defined in subsection 10.2.2, while the mathematical foundations for its minimization using a level set surface flow were already adressed in Sect. 8.4. After the theoretical discussion in this section, we proceed with the details of the implementation in Sect. 10.3.

### 10.2.1 Image Formation Model

We dissolve the chemical Fluorescein in the water. Fluorescein exhibits a photo-luminescent behavior i.e. it has the ability to absorb light of higher energy and subsequently re-radiate light with a lower frequency than the light used for excitation. Fig. 10.2 explains this principle. The emission spectrum is independent of the excitation wavelength, only the amplitude of the emitted light changes. A schematic of our studio setup is shown on the right hand side. We place filters in front of the light source and the cameras, respectively. The two filters allow us to measure the emitted light only, which in turn lets us treat the body of water as a self-emissive medium.

We evenly dissolve the dye in the water and use a strong UV source to illuminate it. This allows us to assume a constant fluorescent emissivity throughout the volume. Thus, the accumulated light intensity along a ray traced through the water can be computed by multiplying its total length within the

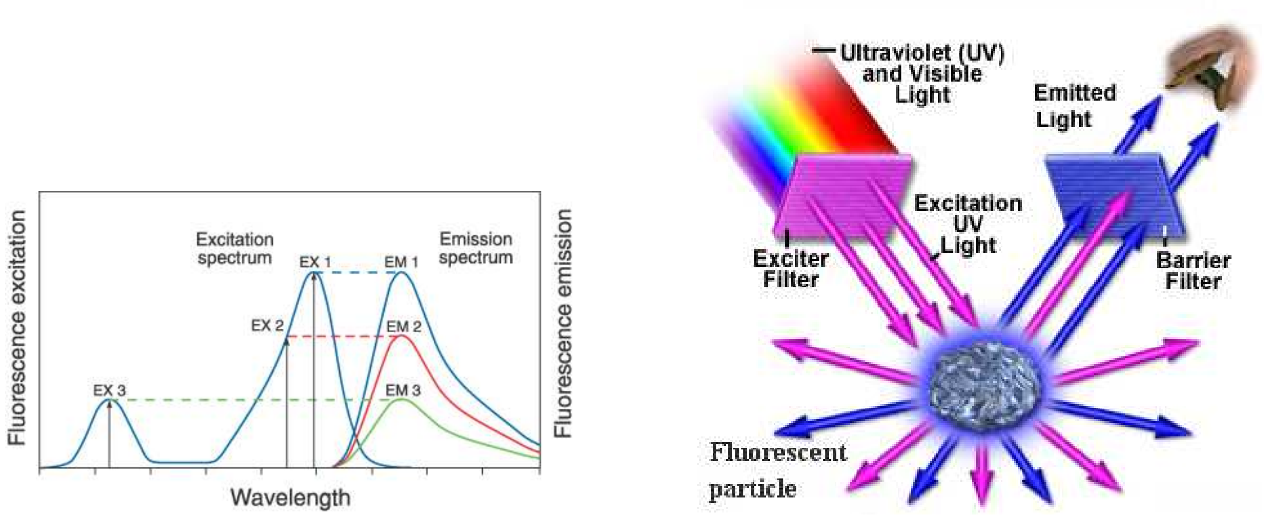


**Fig. 10.2.** *Left:* Excitation and emission in fluorophores: the excitation wavelength changes the amplitude of the emission spectrum only, the emission's maximum wavelength stays the same. *Right:* The use of filters generates a proper excitation light source, and allows the observer to measure the emitted spectrum without interference from the excitation light source.

volume with a constant emittance $\rho$. Furthermore, a color calibration on the cameras is performed, such that they exhibit a linear response to the incoming light intensity, scaling light intensity to image intensity by a factor of $\gamma$.

Now, let $p$ be a point in the image plane of camera $C$, and $C$ be the camera's center of projection. We want to compute the theoretical pixel intensity $I_\Sigma(p)$ in the presence of a surface $\Sigma$, enclosing a volume $\mathcal{O}_\Sigma$ of water prepared as above. Let $R(C,p)$ be the ray traced from $C$ in the direction of $p$ through the surface $\Sigma$, taking into account correct refraction, Fig. 10.4. We ignore scattering and extinction effects in the water volume. Then,

$$I_\Sigma(p) \;=\; \gamma \int_{R(C,p)\cap\mathcal{O}_\Sigma} \rho \, ds \;=\; \rho\gamma \int_{R(C,p)\cap\mathcal{O}_\Sigma} ds.$$

The last integral just measures the length the ray traverses through $\mathcal{O}_\Sigma$. In order to avoid having to determine the constant factor $\rho\gamma$ experimentally by acquiring and measuring a calibration scene, we implement an auto-calibration scheme. All image intensities are divided by the average intensity of the pixels in the image within the silhouette, and all ray-traced intensities by the average intensity of the rays corresponding to these pixels. The resulting quotients are independent of the quantity $\rho\gamma$.

Now that we are able to compute synthetic views given a surface $\Sigma$, we have to determine how well a reconstruced surface fits a given set of input views. If we are able to quantify the error, it can be used to define an energy functional mapping surfaces to real numbers, whose minimum yields an optimal reconstruction result. This aim is pursued in the next subsection.

### 10.2.2 Energy Minimization Formulation

We have to take care of photo-consistency of a reconstructed surface $\Sigma$ with the given source images. We set up an energy functional

$$\mathcal{A}(\Sigma) := \int_{\Sigma} \Phi(s, \mathbf{n}(s)) \, dA(s), \tag{10.1}$$

defined as an integral of the scalar valued weight function $\Phi$ over the whole surface. $\Phi(s, \mathbf{n})$ measures the photo-consistency error density, and may depend on the surface point $s$ and the normal $\mathbf{n}$ at this point. The larger the values of $\Phi$, the higher the photo-consistency error, so the surface which matches the given input data best is a minimum of this energy functional. Because refraction occurs frequently, the dependency of the error measure on the normal is a vital part of our method, in contrast to many other previous applications of weighted minimal surfaces in computer vision.

The question remains how to correctly choose the error measure. Ideally, we would want it to be the difference of the measured intensity in every camera with the theoretical intensity, which would look something like this:

$$\Phi_{\text{naïve}}(s, \mathbf{n}) := \sum_{i=1}^{n} \left( I_{\Sigma,i}(s) - I_i \circ \pi_i(s) \right)^2 ,$$

where $I_{\Sigma,i}(s)$ is the ray-traced image intensity assuming surface $\Sigma$, $I_i$ is the $i$th image, and $\pi_i$ the $i$th camera's projection mapping.

While the general idea is good and exactly what we implement, in this initial form it faces several problems, the worst of which is that we have to be able to evaluate the error function away from the surface in order to perform the surface evolution later. The exact technical definition is presented in Sect. 10.3.

As in the previous chapter, Theorem 7.1 yields the Euler-Lagrange equation of the functional, which leads again to the same surface evolution equation and level set implementation.

## 10.3 Implementation

In this section, we go into the details on how to implement our reconstruction scheme. Subsection 10.3.1 specifies the construction of the error function. For

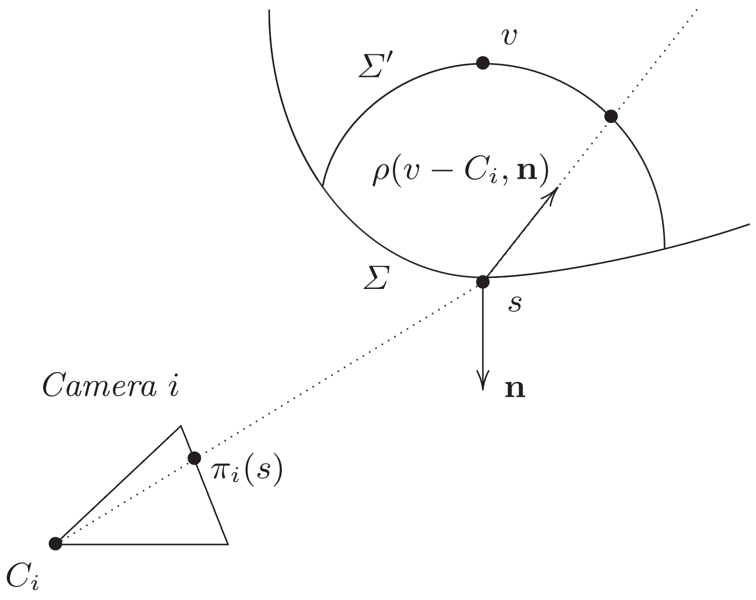


**Fig. 10.3.** *Evaluation of the partial error function $\phi_i$ for a single camera.*. The length difference between rays traced through the distorted surface $\Sigma'$ and the undistorted surface $\Sigma$ is just $\|s - v\|$. Note that $\mathbf{n}$ is not necessarily the exact surface normal, as it may slightly deviate from it in order to evaluate the derivative of $\Phi$ with respect to the normal.

a stable evolution, we have to make sure that the surface does not shrink below the image silhouettes. The boundary term in the evolution equation designed to accomplish this is introduced in subsection 10.3.2. We finally describe some nuts and bolts of the implementation of the PDE as a narrow band level set method in subsection 10.3.3.

### 10.3.1 Construction of the Error Function

Of particular difficulty is the evaluation of the error function $\Phi(s, \mathbf{n})$ for a given point $s$ and corresponding normal $\mathbf{n}$. The problem is that this term has to be evaluated away from the current surface $\Sigma$ in order to compute the derivatives in (8.17), i.e. for points that do not lie directly on the surface, and with a normal which may be different from the current surface normal. The particular question one asks in that case is what local error would arise *if the surface was distorted* such that it lies in $s$ with normal $\mathbf{n}$. For this reason, ray tracing in order to evaluate the error function has to be performed for a distorted surface $\Sigma'$. The computation of $\Phi(s, \mathbf{n})$ is thus performed in three steps.

In the first step, we construct the distorted surface $\Sigma'$ through which rays are traced. We have to change $\Sigma$ locally in a reasonably smooth manner such that the new surface passes through $s$. At this moment, we do not yet care about the normal. Assume for now that $s$ lies outside the volume $\mathcal{O}_\Sigma$ enclosed

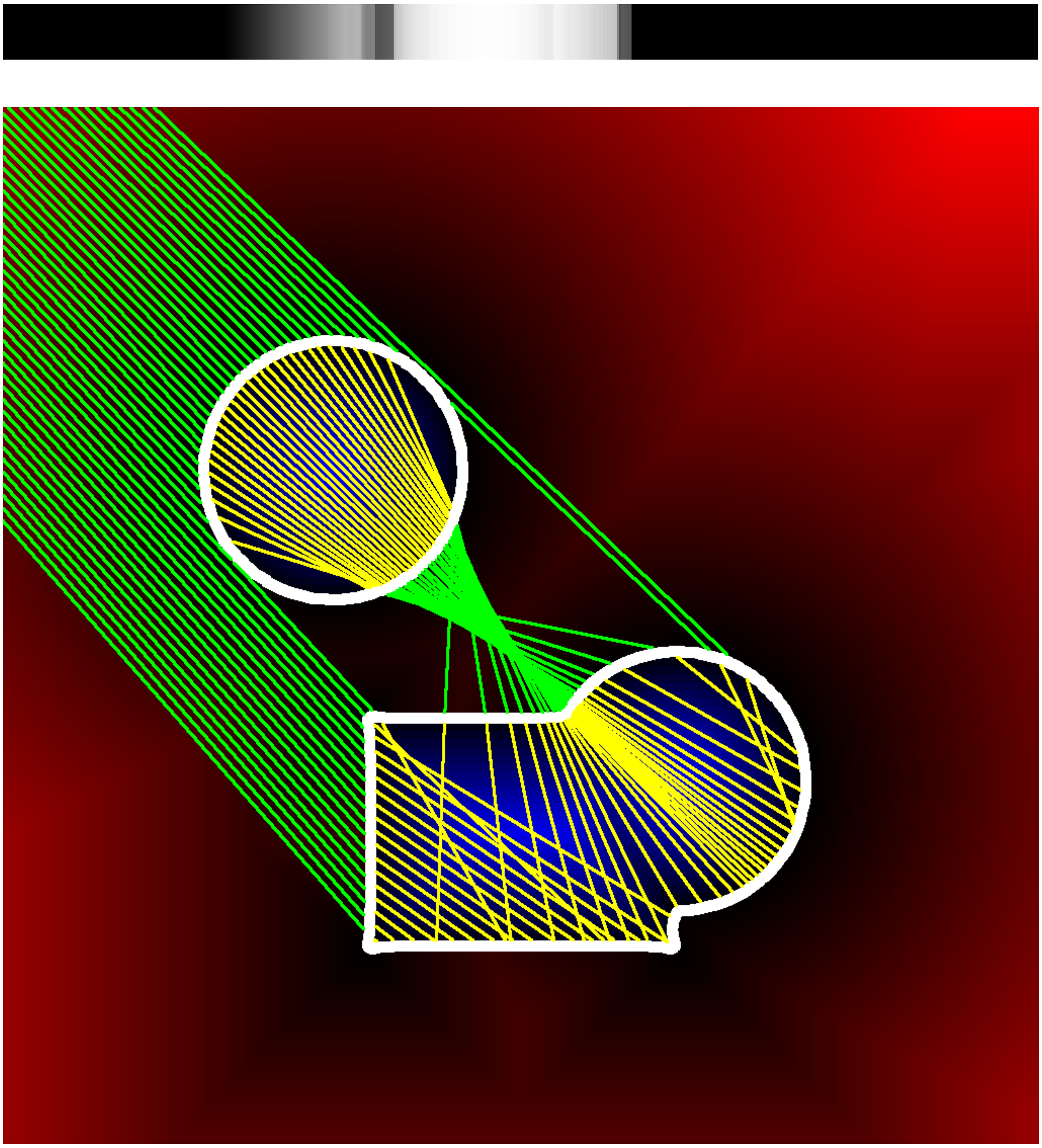


**Fig. 10.4.** Rays necessary to generate a view from the upper left direction, visualizing the complexity of the image formation process. Rays leaving the area without further intersections are not shown. On top is the resulting 1D view, where the intensity of each pixel is proportional to the length of the yellow segments for the corresponding ray.

by $\Sigma$. The desired result can then be achieved by uniting $\mathcal{O}_\Sigma$ with a ball $B$ centered in the point $v$ closest to $s$ on $\Sigma$, with radius $\|s - v\|$. Vice versa, if

$s$ lies inside $\mathcal{O}_\Sigma$, then we can achieve the result by subtracting $B$ from $\mathcal{O}_\Sigma$, Fig. 10.3.

The second step is to define the set of cameras $\mathcal{C} = \{C_1, \ldots, C_k\}$ which contribute to the error measure. Ideally, since the medium is transparent, we would like to consider all cameras we have available. Unfortunately, this would require to find for each camera the ray passing from the camera center to $s$, possibly refracted multiple times on the way. This computation definitely is too time-consuming. Instead, we only consider those cameras which have a good enough unobscured view of $v$ with regard to the original surface. More precisely, each camera $C_i$ belonging to $\mathcal{C}$ must meet the following two criteria:

- The straight line from $v$ to the center of projection $C_i$ must not intersect $\Sigma$, and
- The ray starting from $v$ in the refracted direction $\rho(v - C_i, \mathbf{n})$ must travel inside $\mathcal{O}_\Sigma$ in the beginning. $\rho$ is computed using Snell's law, using the index of refraction of water for inside the volume, and of vacuum for outside.

In the third step, we finally compute the photo-consistency error $\phi_i$ for each contributing camera $C_i$ and average those to get the total error $\Phi$. Each individual error is computed as follows: Let $\mathcal{I}_i \circ \pi_i\,(s)$ be the intensity of the projection of $s$ in image $\mathcal{I}_i$, and $r_i(s, \mathbf{n})$ be the accumulated intensity along a ray traced from $s$ into the refracted direction $\rho(s - C_i, \mathbf{n})$. Then

$$\phi_i(s, \mathbf{n}) \;:=\; \left(\mathcal{I}_i \circ \pi_i\,(s) - r_i(s, \mathbf{n})\right)^2 .$$

This corresponds to comparing the image intensity to the ray-traced intensity of a ray cast from the camera to $s$, refracted as if by a surface located in $s$ with normal $\mathbf{n}$. Thus, the desired normal $\mathbf{n}$ is also correctly taken into account.

## 10.3.2 Silhouette Constraints

An additional constraint on the photo-consistency of the reconstruction result is that the projection of the reconstruction in each camera image must match the silhouette of the object to be reconstructed [64]. This constraint yields both a stopping term in our evolution equation, as well as an initial surface for the evolution in form of the visual hull [65].

To this end, let $\sigma_i$ be the signed distance to the silhouette, defined in the image plane, negative inside the object silhouette. Then we obtain a good initial level set approximation to the image-based visual hull by defining

$$u_0(x) := \max_{i=0}^{n} \left(\sigma_i \circ \pi_i(x)\right)$$

for every $x \in \mathbb{R}^3$. We use this level set function as the starting point for the surface evolution after re-initializing it to a signed distance function.

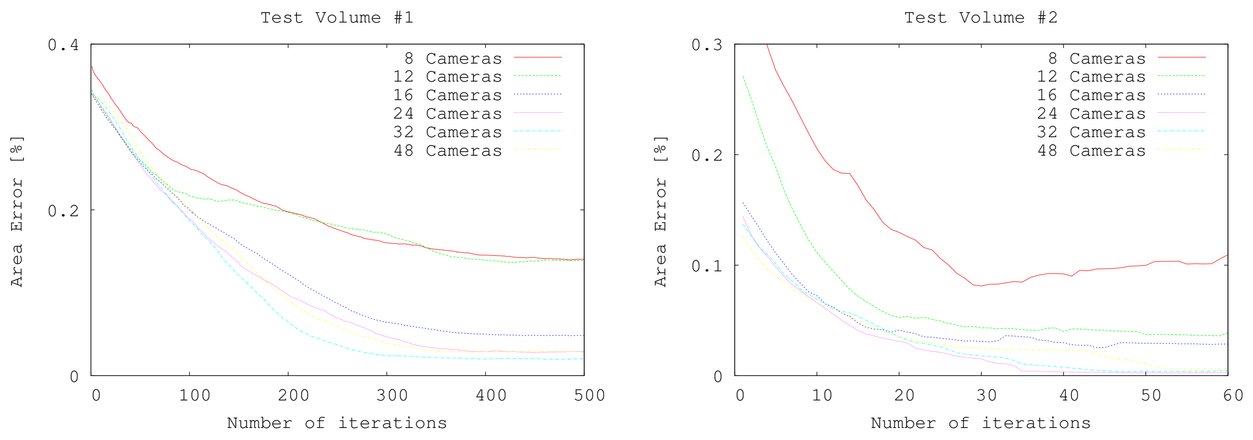


**Fig. 10.5.** Convergence for different numbers of input views.

Furthermore, we prohibit the projections to ever shrink beyond any of the silhouettes. A stopping term is added to the surface evolution, which grows very large if a point on the projected boundary of the surface lies inside a silhouette. When computing the visibility of a point $v$, we can extract from the set of unobscured views $\mathcal{C}$ the set of cameras $\mathcal{B} \subset \mathcal{C}$ in which $v$ lies on or very close to the boundary of the projection. The two criteria for camera $C_i$ in $\mathcal{C}$ to lie in $\mathcal{B}$ as well is that

- The angle between viewing direction $\mathbf{d}_i$ from $v$ to the center of projection $C_i$ and the surface normal $\mathbf{n}(v)$ must be close to ninety degrees, and
- The straight line from $v$ in the direction $\mathbf{d}_i$ away from the camera must not intersect the surface.

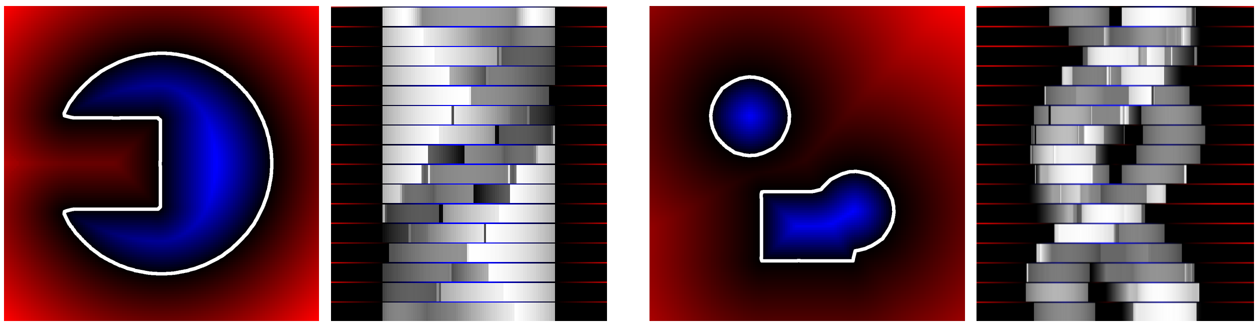
Then the boundary stopping term is defined as

$$B(s) := \sum_{C_i \in \mathcal{B}} \left[\exp\left(-\beta(\sigma_i \circ \pi_i)(v)\right) - 1\right],$$

where $v$ is again the point closest to $s$ on $\Sigma$, and $\beta > 0$ a user-defined weight, which should be set reasonably high. We use $\beta = 10$ throughout all of our tests, where the 2D images are defined to lie in the unit interval, and the signed distance is normalized accordingly.

### 10.3.3 PDE Discretization

In order to implement the level set evolution equation, the volume surrounding the surface $\Sigma$ has to be discretized. We use a regular three-dimensional grid of evenly distributed cells with variable spatial resolution of usually $64^3$ or $128^3$ cells. The surface is evolved according to the narrow band level set method [115], starting the evolution with the visual hull surface $\Sigma_0$ and the



(a) The first synthetic "volume" together with 16 input views stacked on top of each other to the right. Below each view is shown the signed distance transform $\sigma$ of the silhouette.

(b) The second synthetic "volume", again together with 16 input views and signed distance transform of the silhouette.

**Fig. 10.6.** Synthetic test volumes and ray-traced views. Red color denotes positive values of signed distance, blue color negative values.

values $u_0^{xyzt}$ of the corresponding level set function $u_0$ in the centers of the grid cells. Details on how the evolution equation is implemented were already presented in Sect. 9.3 of the previous chapter, to which the reader is referred. However, there are two optimization terms which are added to the values in the cells after each update step (9.2).

The first one is the boundary term $B(x, y, z)$ defined in subsection 10.3.2. The second term is designed to speed up the convergence and avoid local minima. It accelerates the shrinking process in regions where the error is excessively high. We add to $u_{i+1}^{xyzt}$ the value

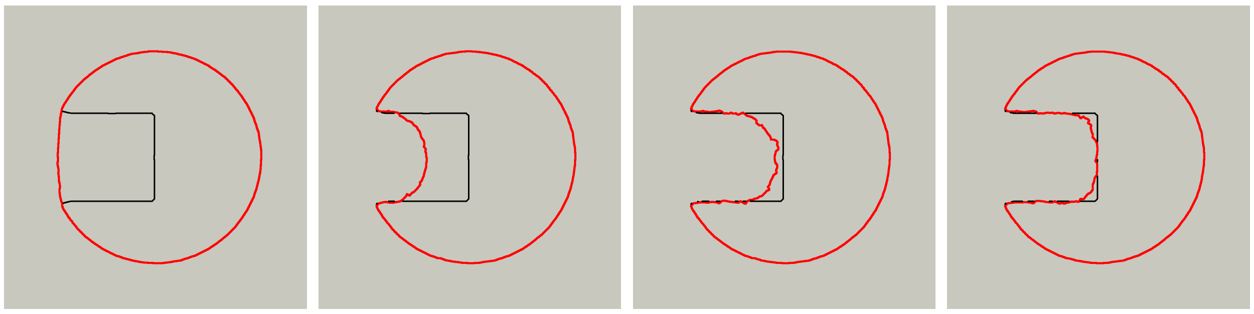
$$\epsilon_1 B(x, y, z) - \epsilon_2 L_{\sigma(\Phi)}(\Phi(x, y, z) - m_\Phi),$$

where $L_{\sigma(\Phi)}$ is the stable Leclerc M-estimator for the standard deviation of the error values of all cells, and $m_\Phi$ the mean value of the error. $\epsilon_1, \epsilon_2 > 0$ are two user-defined weights, the only free parameters of our method. Good choices and their influence on convergence behaviour is discussed in the next section, where we present results obtained with synthetic 2D as well as 3D real world data.

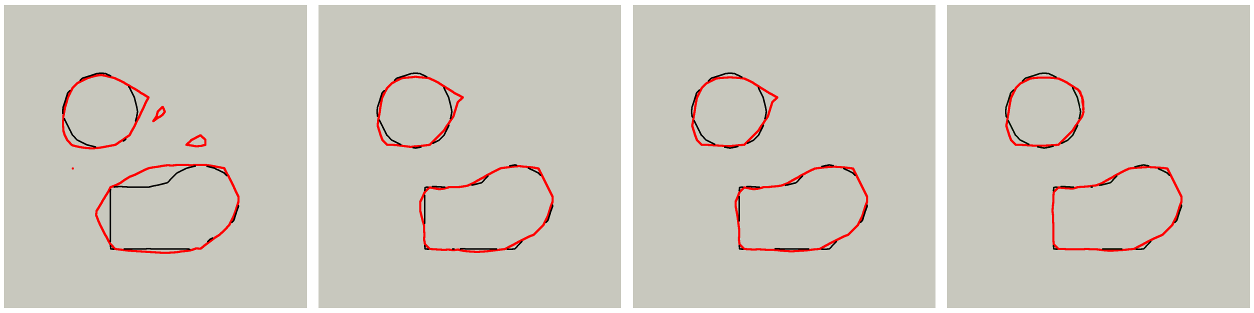
## 10.4 Results

### 10.4.1 Synthetic 2D Experiments

In order to verify that our surface evolution is capable of producing correct results despite the complex problem we want to solve, we first test it on



(a) Convergence towards the first test volume, after 0, 100, 200, and 300 iterations.



(b) Convergence towards the second test volume, after 0, 15, 30, and 45 iterations.
**Fig. 10.7.** The best results we achieved with 24 input views, together with several in-between stages of the iteration.

synthetic 2D data. We ray-trace several views of two different test volumes using the image formation model presented in Sect. 10.2.1. The first volume is designed to test how well the algorithm can recover concavities, while the second volume is not connected and has a mixture of straight and round edges. Both test volumes and resulting 1D views are shown in Fig. 10.6. An exemplary trace through the volume can be found in Fig. 10.4. This trace gives a glimpse of the complexity of the reconstruction problem, and demonstrates how heavily the ray-tracing result depends on the normals.

We run our algorithm with different numbers of input views in order to test the dependence of convergence on this critical parameter. The results are shown in Fig. 10.5. Convergence becomes stable with eight or more cameras used, with twelve views required in the more complex second test case. We can also note that there is a quick saturation of reconstruction quality with respect to the number of cameras. The visual hull does not improve much more if more than 16 cameras are used, in accordance with earlier results [78]. In addition, the quality of the reconstruction peaks at around 24 cameras for both test volumes. Interestingly, more cameras do not necessarily imply a better result, which indicates that a good placement of the cameras is at least as important as their sheer number. The best reconstruction results were achieved with the moderate number of 24 cameras, shown in Fig. 10.7.

The computation of the normal term is relatively time-consuming, and our analysis of the convergence behaviour suggests that it becomes relevant only during the later iterations, when the finer details of the surfaces need to be carved out. For that reason, we adopt a hybrid model, where for the initial evolution the normal term is set to zero, and switched on as soon as the convergence starts to slow down.

In all cases, the algorithm runs with the same parameter values of $\epsilon_1 = 0.1$ and $\epsilon_2 = 100$. It exhibits a very stable behaviour against parameter changes, as Table 10.4.1 suggests. Here, 24 Cameras are used for the estimation of the first test volume, and the error after exactly 200 iterations depending on different parameter values is noted down. As a rule of thumb, there is a certain

| | | $\epsilon_1$ | | | | |
|---|---|---|---|---|---|---|
| | | 0.01 | 0.1 | 0.5 | 1 | 5 |
| | 1 | 0.07 | U | U | U | U |
| | 10 | 0.05 | 0.04 | 0.06 | U | U |
| $\epsilon_2$ | 50 | 0.16 | 0.07 | 0.03 | 0.04 | U |
| | 100 | 0.04 | 0.05 | 0.04 | 0.06 | U |
| | 1000 | S | S | S | S | 0.03 |

**Table 10.1.** *Final error depending on different settings of the parameters.*

threshold value for the speedup term above which it accelerates the evolution above a stable limit, causing the surface to shrink uncontrolled below the silhouettes. This is indicated by a “U” in the table. Too low a choice of $\epsilon_1$ has no ill effects on stability, but slows down the convergence a bit. $\epsilon_2$ can safely be chosen somewhere between 10 and 100 without much effect, but may cause the surface to be stuck at an undesireable spot if set too high, as indicated by the “S” in the table.

### 10.4.2 Real-world Water Videos

For the real-world tests, we use a multi-video studio consisting of 8 CCD-cameras with a resolution of $1004 \times 1004$ pixels. The cameras can record at a frame-rate of 45 frames per second. A 300W UV light source is employed to illuminate the Fluorescein-dyed water. Our setup is shown in Fig. 10.8. We acquire test sequences using a dark studio, the excitation light source and the fluorescent water being the only source of light. This measure allows for simple background subtraction. The reconstruction is performed on an equidistant, uniform grid of $128^3$ voxels. An example of a reconstructed water surface rendered in a virtual environment and with changed material properties is shown in Fig. 10.9.

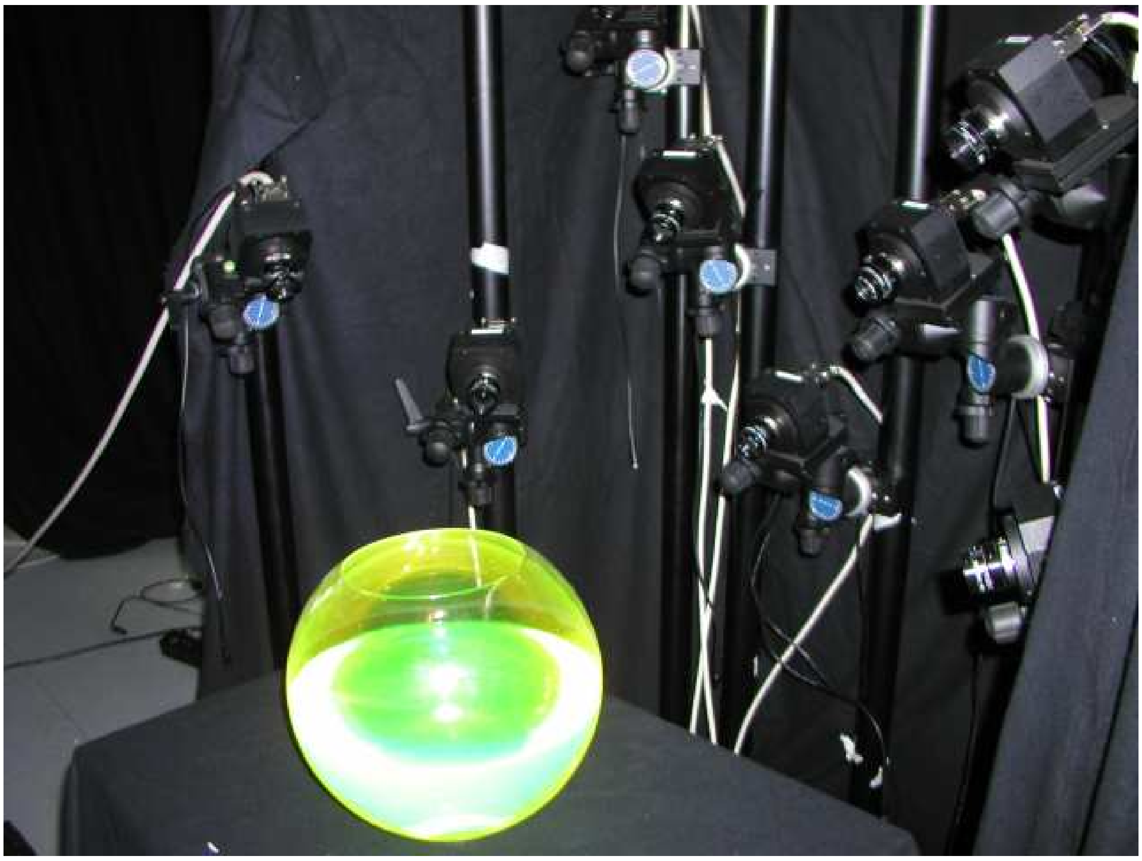


**Fig. 10.8.** Our studio setup consists of eight cameras, a fish bowl filled with dyed water and a UV light source (not visible).

## 10.5 Conclusions

We have presented a method for the reconstruction of flowing water surfaces suitable for free-viewpoint video. A novel recording methodology and a corresponding image formation model allow us to define a photo-consistency constraint on the reconstructed surface. We utilize weighted minimal surfaces to refine the visual hull of the water using constraints based on thickness measurements of the real surface. The resulting energy functional is minimized using the Euler-Lagrange formulation of the problem, leading to a partial differential equation. This PDE is solved by applying the well known level set method. Synthetic tests indicate that the solution of the equation is stable. Real-world tests demonstrate the suitability of our method for the reconstruction of water.

Our Future work includes research into the applicability of our method to the reconstruction of other refractive media. Additionally, we would like to develop a hierarchical representation of the underlying computational grid to achieve a higher resolution reconstruction which allows to resolve finer details.

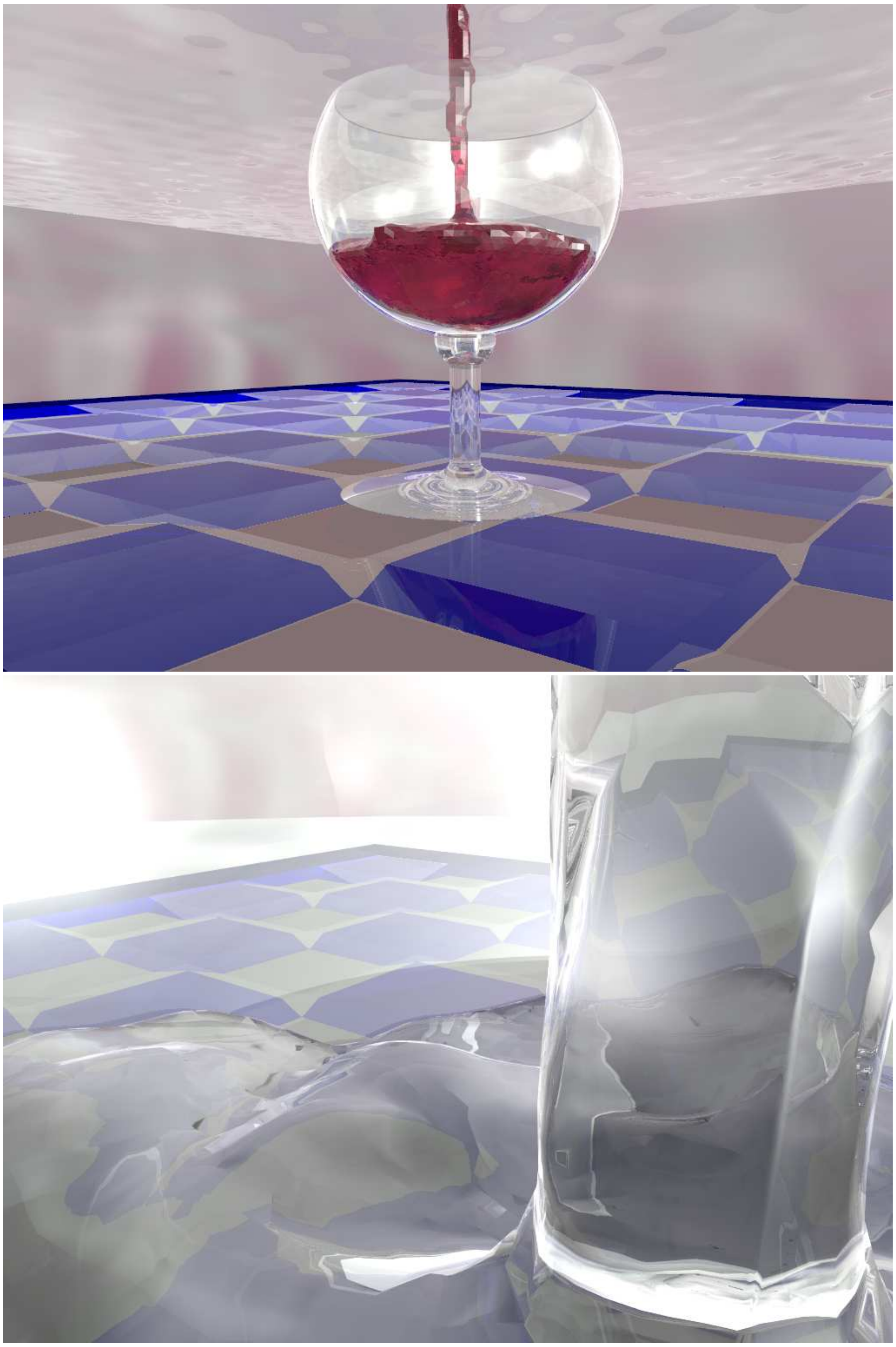


**Fig. 10.9.** Reconstructed stream of water placed in a virtual environment. *Top:* Turning water into wine - we changed the material properties of the water such that it resembles red wine. *Below:* Close-up of the water surface, showing the intricate details of the reconstructed geometry. All ray-tracing was performed with the *Persistence of Vision Ray Tracer*, available at `www.povray.org`.

# Part IV

# Video-based Rendering

# 11

# Overview

## 11.1 Introduction

In the last two parts, we described how to extract geometric information from multi-video sequences. In the final stage of a free-viewpoint video system, this geometric data has to be exploited to create high-quality novel views of a scene. The two scenarios we analyzed require two distinct different kinds of rendering algorithms, which we present in this part. Both have in common that the source video images are mapped onto the target geometry using modern graphics hardware.

In Chapter 12, we present a rendering system for dynamic light fields. It is suitable for a number of input streams with accompanying depth maps, which have to be estimated offline using for instance the techniques presented in Part II. The system achieves 20 frames per second on modern hardware when rendering a 3D movie from an arbitrary eye point within the convex hull of the recording camera's positions. Novel views are created by warping and blending the source images using an underlying mesh representation.

Rendering algorithms for the second case of a dynamically varying surface are presented in Chapter 13. We consider both the case of a triangle mesh representation and of a surface represented by a level set[1]. In a triangle mesh representation, the textures obtained from the input videos are just projected onto the geometry using projective texturing with careful selection of input camera views and weights. More sophisticated algorithms have to be considered for volumetric representations. Our method of choice is microfacet billboarding, which is both very fast and produces novel views of very good quality.

[1] or voxel volume, which is essentially a special case in the sense that a voxel volume is nothing more than a two-valued level set function

# 12

# Dynamic Light Field Rendering

## 12.1 Introduction

In this chapter, a rendering system for dynamic light fields is presented. A dynamic scene can be displayed interactively from novel viewpoints by warping and blending images recorded from multiple synchronized video cameras. The system enables interactive viewing of 3D movies from arbitrary viewpoint positions within the window spawned by the camera positions.

It is tuned for streamed data and achieves 20 frames per second on modern consumer-class hardware when rendering a 3D movie from an arbitrary eye point within the convex hull of the recording cameras' positions. The quality of the prediction largely depends on the accuracy of the disparity maps which are reconstructed off-line and provided together with the images.

In Sect. 12.2, we give some mathematical background on the relationship between disparity and depth, and show how vertices are projected into novel views using depth information. Sect. 12.3 presents our novel rendering algorithm, which maps input images onto an underlying triangle mesh and warps the textured mesh forward into novel views using modern graphics hardware. Sect. 12.4 shows our results, and we conclude in Sect. 12.5.

## 12.2 Disparity Compensation

In this section, we derive how disparity information is related to depth, and how it can be used to warp points in one image forward into novel views.

Let $\mathcal{I}$ be an image from a source camera $C$. For $k = 0, \dots, n$, let $\mathcal{I}_k$ be an image from a reference camera $C_k$ related to the source image by the fundamental matrix $\mathbf{F}_k$, see Sect. 2.2. If $P \in \mathbb{R}^3$ is a fixed point in world coordinates and $p$ its projection in the source image in homogenous coordinates, the corresponding point $p'_k$ in the reference image $\mathcal{I}_k$ lies on the epipolar line of $p$ satisfying the *epipolar constraint* (2.1),

$$p'^T_k \, \mathbf{F}_k \, p \;=\; 0. \tag{12.1}$$

We use the prime to denote points whose coordinates are given in the image coordinate frame of $C_k$, all others are given in the image coordinate frame of camera $C$.

If $C_k$ is obtained from $C$ by a pure translation $\mathbf{d}_k$ parallel to the image plane, the fundamental matrix is given by [32, Sect. 1.13]:

$$\mathbf{F}_k \;=\; [\mathbf{d}_k]_\times \;=\; \begin{bmatrix} 0 & 0 & d_{k,y} \\ 0 & 0 & -d_{k,x} \\ -d_{k,y} & d_{k,x} & 0 \end{bmatrix}$$

and the epipolar constraint (12.1) yields an equation for a line which can be rewritten as

$$p'_k \;=\; p \;-\; \lambda \mathbf{d}_k. \tag{12.2}$$

Note that $\mathbf{d}_k$ is not normalized. Here $\lambda = \lambda(P) \geq 0$ is called the *disparity of* $P$ and can be interpreted as the parallax of $p$ for a unit camera movement on the eye point plane. We also note that it is a bijective function of the *depth* $\delta(P)$, the distance of $P$ to the image plane: From Fig. 12.1, we deduce that *in the image coordinate frame of camera* $C$, $p_k$ has coordinates

$$p_k \;=\; p + \frac{\delta(P)}{\delta(P)+f}\,\mathbf{d}_k, \tag{12.3}$$

where $f$ denotes the camera's focal length. Since the coordinate frames in the image plane are related by $p'_k \;=\; p_k - \mathbf{d}_k$ we conclude by comparing (12.2) with (12.3) that

$$\lambda \;=\; 1 \;-\; \frac{\delta(P)}{\delta(P)+f} \;=\; \frac{f}{\delta(P)+f}$$

does not depend on the reference image, so it is well-defined. Equation (12.2) thus ensures that knowledge of $\lambda$ is sufficient to derive the location of $p$ in an image taken by a camera related to $C$ by a pure translation parallel to the image plane.

For the interactive rendering, we usually employ pre-computed disparity maps obtained with one of the reconstruction techniques presented in Part II. The depth maps visible on the next pages, however, were obtained with the PDE-based reconstruction technique by Alvarez et al. [3].

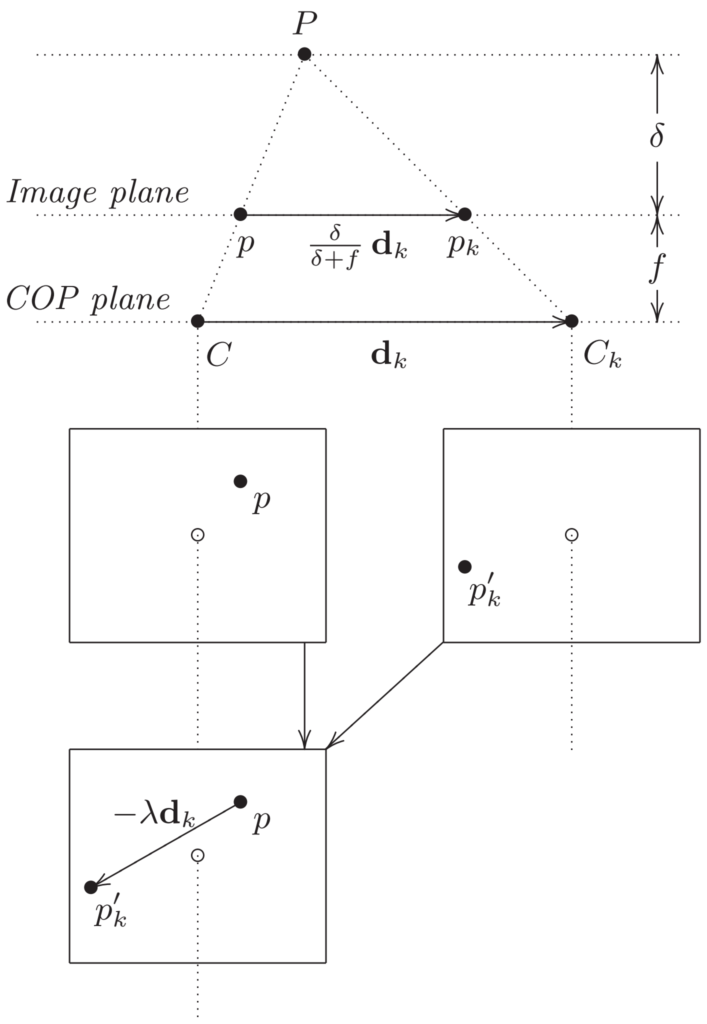


**Fig. 12.1.** *Top*: Plane spawned in $\mathbb{R}^3$ by a point $P$ and the cameras' centers of projection (COP). *Below:* Image coordinate frames of camera $C$ on the left and camera $C_k$ on the right side.

## 12.3 Interactive Rendering

In the rendering step, a number of images $\{\mathcal{I}_k\}_{k=1,\ldots,m}$ with precalculated dense disparity maps $\{\lambda_k\}$ are warped and blended in order to predict a view of the scene from a new viewpoint $C$. For interactive frame rates, one cannot transform each pixel seperately as this would consume too much time. The method described in this section exploits the polygon processing capabilities of OpenGL as well as hardware texturing and blending provided by modern graphics hardware.

We create a regular triangle mesh covering the area of the source image and assign to each vertex $\mathbf{v}$ of the mesh a disparity value $\lambda(\mathbf{v})$ computed as the average of its surrounding pixels in $\lambda_k$. This process essentially downscales the disparity maps and reduces the number of transformations required during rendering. The downscaled maps can also be precomputed and stored on hard drive for different resolutions of the triangle mesh. An additional benefit of

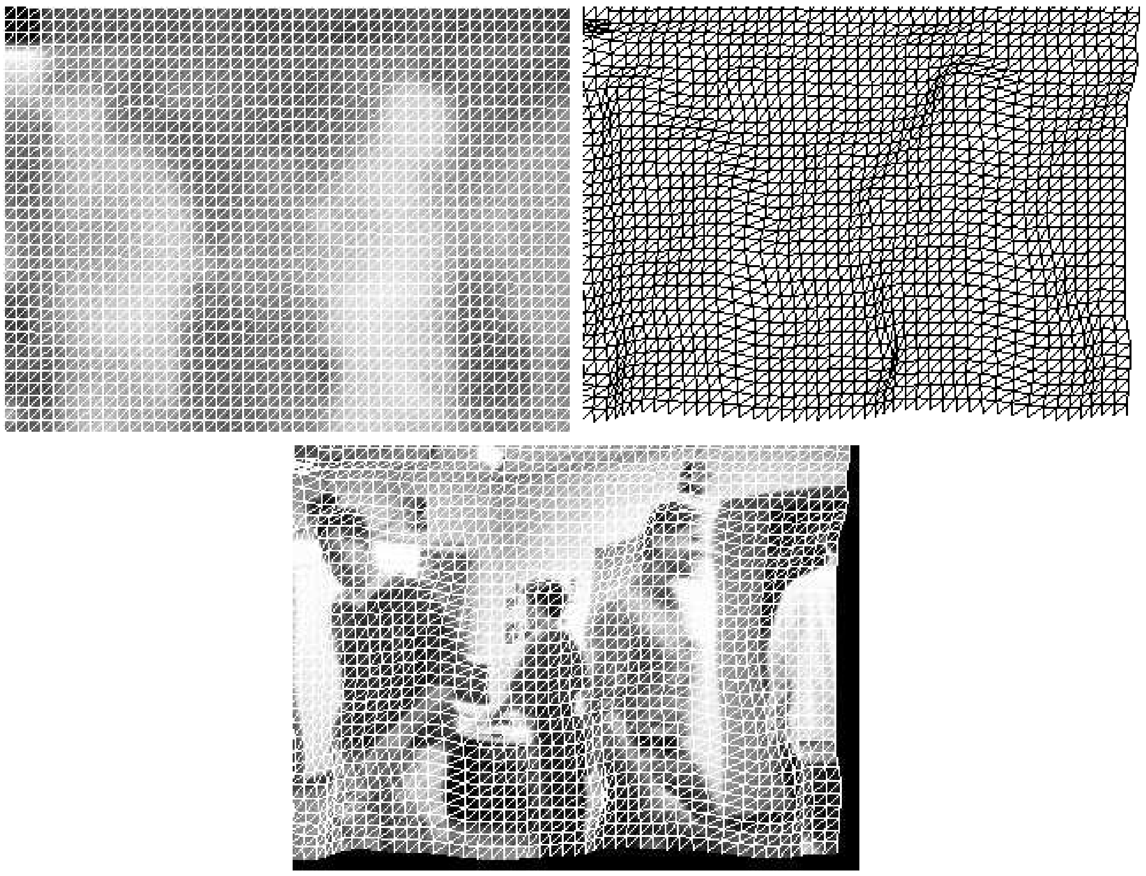


**Fig. 12.2.** Disparity map with triangle mesh, warped triangle mesh and resulting warped image.

downscaled disparity maps is their smaller size which speeds up loading while displaying sequences of movie frames.

The blending process requires $m$ passes of the scene. In the $k$th pass, the source image $\mathcal{I}_k$ is loaded to the texture target TEXTURE_RECTANGLE_NV. This OpenGL extension is required since the width and height of the source images is usually not a power of two. The mesh is rendered as a sequence of triangle strips, which gives far superior performance compared to single quads. Final vertex positions and texture coordinates are computed in hardware by a vertex program which performs the following tasks:

- Use the position of the vertex directly as the texture coordinates in the source image. Note that texture coordinates for rectangular textures are not homogenous.
- Compute the position of the vertex $\mathbf{v}$ in the warped image according to

  $$\mathbf{v}_{opos} = \mathbf{v} + \lambda(\mathbf{v})\,\mathbf{d}_k,$$

  where $\mathbf{d}_k$ is the translation from $C_k$ to the new viewpoint $C$.

The initial weight for each image is given by

| Block size [pixel] | Mesh res. [blocks] | Triangles [#] | Frame rate [Hz] |
|---|---|---|---|
| 8 | 40×30 | 9600 | 72.99 |
| 4 | 80×60 | 38400 | 32.15 |
| 2 | 160×120 | 145200 | 10.87 |

**Table 12.1.** Frame rates for different mesh resolutions (excluding loading times).

$$\omega_k \; := \; \exp(-c\, \mathbf{d}_k \cdot \mathbf{d}_k). \tag{12.4}$$

$\omega_k$ decreases with greater distance to the source image. The use of the Gaussian is not mandatory, it is but one of the functions having the desired property of smoothness and a function value of one when $\mathbf{d}_k$ equals zero. To further speed up rendering, images with a weight below a small treshold value $\epsilon > 0$ are not used since their contribution is too small to be visible. The constant $c$ is chosen so that $\omega_k$ falls just below $\epsilon$ when $\mathbf{d}_k$ equals the minimum distance between two cameras. Thus, if the position of the camera for the predicted view coincides with one of the source cameras, the original image of the camera is reproduced exactly without distortions from other source images.

After all initial weights are known, the final blending weight $w_k$ used in the OpenGL blend equation is then computed according to a cumulative normalization by

$$w_k \; = \; \frac{\omega_k}{\sum_{i=1}^{k} \omega_i}.$$

The stencil buffer is used to ensure that a weight of 1 is used in areas where no image data has yet been written. That way it is guaranteed that all pixels are blended with the correct weight relative to the images already warped and that for each pixel the sum of all weights after every pass is equal to one.

Backfacing triangles are culled during rendering since their pixels are obscured by a nearer object. An example of the original triangle mesh, the depth map and the resulting warped mesh is shown in Fig. 12.2.

## 12.4 Results

The measurements in the following tables were performed on a 1.7GHz Pentium Xeon with an nVidia GeForce 4 graphics card. Four source images with a resolution of $320 \times 240$ pixels taken from cameras in the corners of a square are warped together to render a predicted view with a resolution of $640 \times 480$ pixels, see Fig. 12.4 and Fig. 12.4. Frame rates achieved for a static image with

| Block size | Time per 100 frames used for Rendering | Loading images | Loading depth maps |
|---|---|---|---|
| 8 | 1.37 s | 3.68 s | 0.15 s |
| | 26.4 % | 70.8 % | 2.8 % |
| 4 | 3.11 s | 3.68 s | 0.61 s |
| | 42.0 % | 49.7 % | 8.3 % |
| 2 | 9.98 s | 3.68 s | 2.45 s |
| | 61.9 % | 22.8 % | 15.3 % |

**Table 12.2.** Profile for different tasks while displaying a movie, assuming the theoretical average transfer rate of 25 MByte/s.

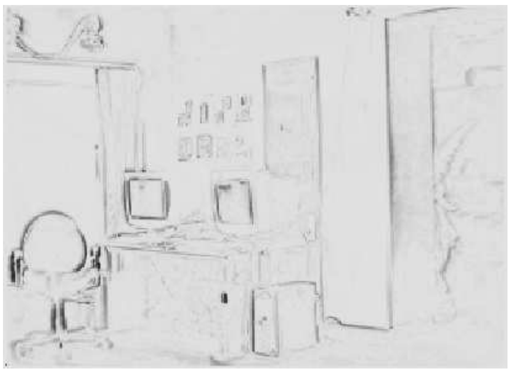


**Fig. 12.3.** Residual error of disparity compensation: The error is concentrated along edges in the image where small inaccuracies in disparity result potentially in large differences in color.

| Block size | Root mean squared error |
|---|---|
| 8 | 16.50 |
| 4 | 16.35 |
| 2 | 16.30 |

**Table 12.3.** Per-pixel error in a view warped from three other images. Pixel values range from 0 to 255.

different triangle mesh resolutions are denoted in Table 12.1, where block size corresponds to triangle leg length in pixel.

In the case of dynamic scenes, the task of loading the images and depth maps becomes the bottleneck, as can be concluded from Table 12.2. Indeed, at this resolution about 1 MByte of image data and 0.25 MByte of disparity data have to be transferred per frame from hard drive to the graphics card.

Modern standard IDE drives achieve average loading rates of 25 MByte per second, which limits the theoretically possible frame rate to 20Hz. In practice, the transfer rate on our system seldom exceeds 15 MByte/s, probably because the different data files to be read are not stored linearly on the drive.

Thanks to the use of graphics hardware for rendering, the predicted image can be scaled to an arbitrary size with no impact on performance. Smaller block sizes result in a more complex triangle mesh and require more bandwidth and rendering time, but improve the quality of the image only marginally. This is shown quantitatively in Table 12.3, where we predict the view from the top-left camera by warping the images of the other three cameras. The mean squared error per pixel between the original image and the predicted image serves as a measure for the warped image quality. Fig. 12.3 shows a visualization of the error distribution.

The predicted image in the worst possible case where the camera lies in the center of the square is displayed on the color plate. Some strong blurring artifacts are visible in the areas circled in red. The errors in the upper right corner result from the fact that some part of it is visible in only one image, so no correct depth information can be derived for it. In general, the depth information near the boundary is not as accurate as in central regions, which is a common problem in disparity map estimation [3]. The blurriness in the legs of the person is due to the motion blur already present in the source images, which leads to bad disparity estimates. However, the algorithm reconstructs well features such as the corners in the wall and the computer monitors, circled green. A movie showing our system in action is available for download on our web page[1].

## 12.5 Conclusions

The system we have presented is capable of rendering 3D movies from an arbitrary viewpoint within the recording window at interactive frame rates on today's consumer-class hardware. Image quality largely depends on the accuracy of the disparity maps provided with the recorded video streams.

In our current implementation the predicted view can only be rendered for translated cameras. The correlation algorithm used for preconditioning the disparity map estimation also assumes that the cameras in the array are related to each other by pure translations. However, it is possible to generalize our software towards arbitrary recording geometry and arbitrary positions used for prediction. The additional hardware-accelerated per-vertex computations will not decrease the overall frame rate significantly. Since the real

[1] *http://www.mpi-sb.mpg.de/∼bg/3dtv.html*

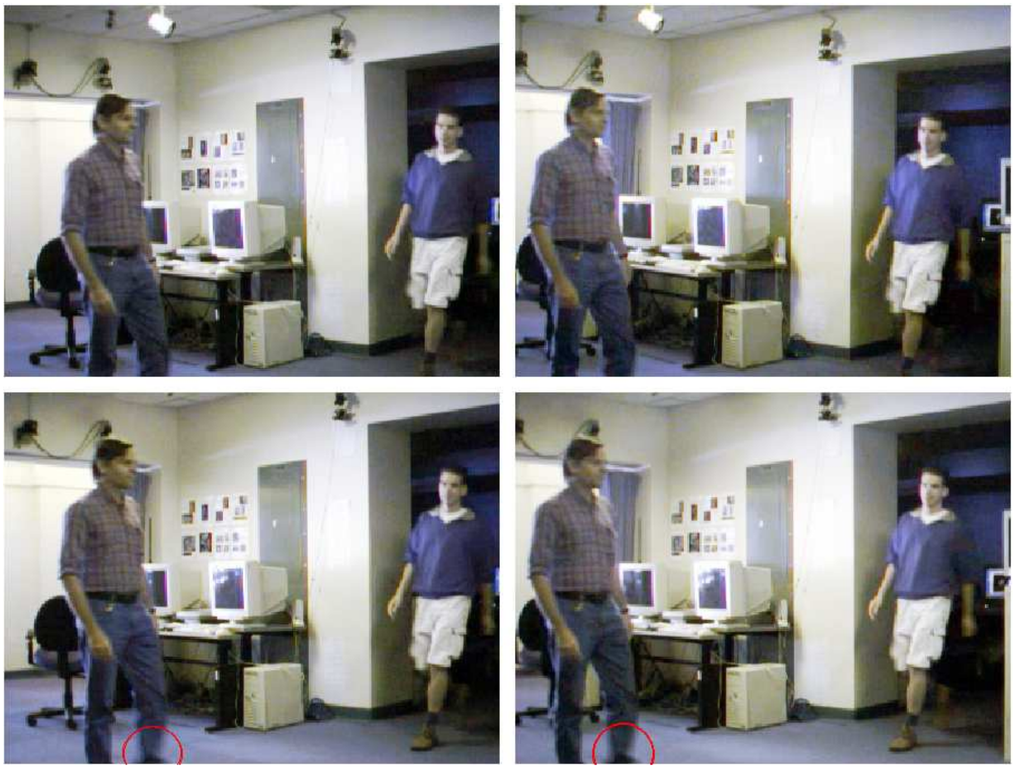


**Fig. 12.4.** Source images taken by four cameras positioned at the vertices of a square. Note the artifacts encircled red are caused by motion blur which lead to blurred reproduction in the predicted view below.

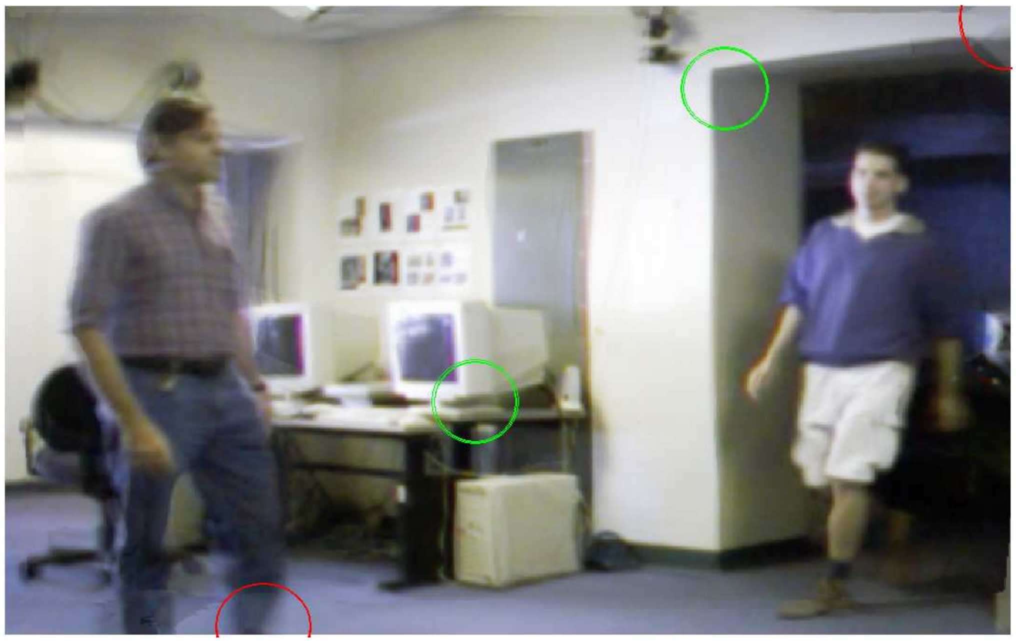


**Fig. 12.5.** The predicted view from a viewpoint in the center between four cameras. Note the sharp reproduction of edge features circled green. The visible blurriness in areas marked red is caused partly by motion blur in the source images, Fig. 12.4, and partly by inaccuracies in the depth maps as explained in the main text.

bottleneck is the time needed for data loading, compression techniques to speed up the transfer have to be investigated.

One could improve the quality of the prediction by using a triangle mesh adapted to image features instead of a regular mesh. By matching triangle edges with edges in the depth maps, pixels belonging to the same physical object will be bound to an independent subset of the mesh, which further improves rendering quality.

# 13

# Free-Viewpoint Video Rendering

## 13.1 Introduction

In this chapter, we present a hardware-accelerated image-based rendering system which uses multi-video sequences to texture a pre-computed approximation to scene geometry. The input images are placed into the texturing units, and projective texture coordinates are computed by a vertex/fragment program combination. Textures are blended with weights based on the visibility of each geometric primitive by a source camera. Rendering is performed at real-time frame rates, so our method is suitable for a free-viewpoint video system, if we can stream the geometry and the input images to the graphics hardware fast enough. Since AGP and in particular PCI Express data transfer rates are sufficient by a large margin, the bottleneck lies in hard drive transfer rate. It is therefore necessary to combine this rendering stage with a good compression algorithm with real-time decompression capability. We can also combine it with our studio setup, Sect .7.4, which can compute the visual hull interactively, in order to obtain a live recording and playback system.

Our algorithm can handle both direct triangle mesh representations as well as volumetric geometry defined as the zero level set of a function. In the latter case, we render *microfacet billboards*, small rectangles facing the viewer, which cover the whole zero level set. The former case is much simpler, since we can render the geometry directly. Because of this, in the following we only describe the case of volumetric geometry being rendered using billboards. Rendering of triangle meshes works exactly the same, except that the billboards are replaced by the triangle surface geometry. Sect. 13.2 is devoted to our novel rendering algorithm, whose results are presented in Sect. 13.3. We conclude with some plans for future work in Sect. 13.4.

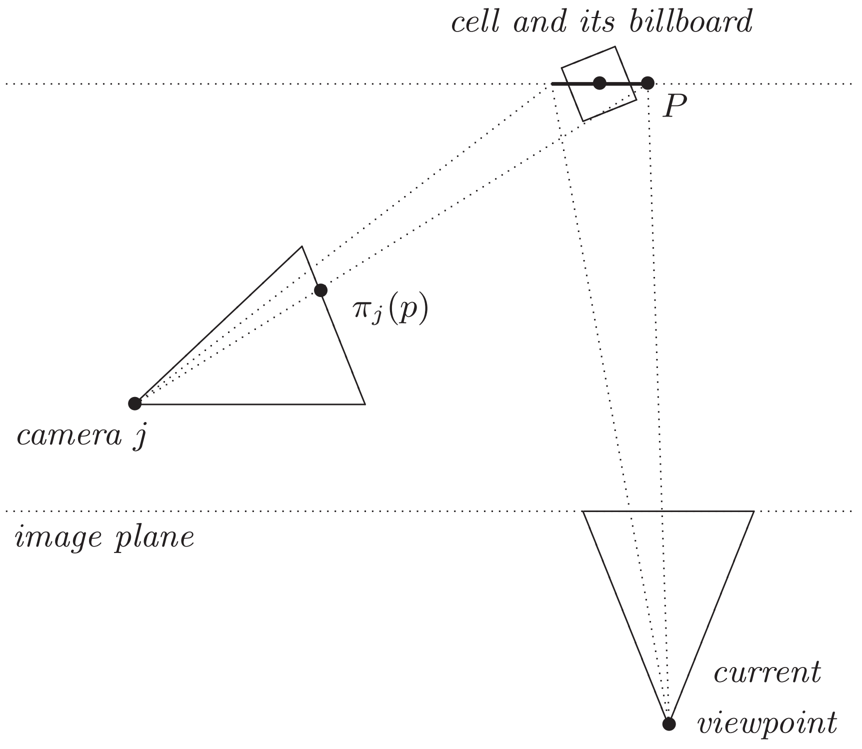


**Fig. 13.1.** The billboard associated with a cell is oriented always parallel to the image plane. Its corner $p$ has the texture coordinate $\pi_j(p)$ in the image of camera $j$. The projection $\pi_j$ can be pre-computed.

## 13.2 Hardware Accelerated Rendering

For each cell near the zero level set we render a single geometric primitive called a billboard, which is a rectangle parallel to the image plane having the same center as the cell. Since the coordinates of the billboard's corners in 3D-space are known, we can compute their locations in the camera views and use this information to texture the billboards, Fig. 13.1. The cameras are immobile, so this projection $\pi$ can be pre-computed and evaluated very efficiently using hardware-accelerated dependent texturing and customizable vertex transformations. In the remainder of the section we describe the texture setup in more detail.

For the sake of simplicity of notation we assume that the support $\Gamma$ of the level set function $u$ lies in the unit cube $[0,1]^3$. The formulas can easily be adapted to arbitraty positions by inserting additional transformations at the appropriate locations. In a preprocessing step, the unit cube is divided into $s^3$ smaller cubes. The 3D textures $T_i^\pi$ which discretize the mappings $\pi_i$ will be of size $s \times s \times s$ and are defined for the centers of the cubes. For each camera $i$ and each cube, we compute $\pi_i(p)$ for its center $p$, encode the resulting homogenous 2D texture coordinate into the alpha and red channel

| State | Value |
|---|---|
| *General* | |
| `BlendEquation` | `FUNC_ADD` |
| `BlendFunc` | `SRC_ALPHA`, `ONE_MINUS_SRC_ALPHA` |
| *Texture unit 1* | |
| `SHADER_OPERATION` | `TEXTURE_3D` |
| `TEXTURE_ENV_MODE` | `NONE` |
| *Texture unit 2* | |
| `SHADER_OPERATION` | `DEPENDENT_AR_TEXTURE_2D_NV` |
| `TEXTURE_ENV_MODE` | `COMBINE` |
| `COMBINE_RGB` | `REPLACE` |
| `SOURCE0_RGB` | `TEXTURE` |
| `OPERAND0_RGB` | `SRC_COLOR` |
| `COMBINE_ALPHA` | `REPLACE` |
| `SOURCE0_ALPHA` | `PREVIOUS` |
| `OPERAND0_ALPHA` | `SRC_ALPHA` |
| `SOURCE1_ALPHA` | `TEXTURE` |
| `OPERAND1_ALPHA` | `SRC_ALPHA` |

**Fig. 13.2.** OpenGL and texture stage state during billboard rendering.

of a texture and store them at the location $p$ of the current 3D texture $T_i^\pi$. This initial setup needs to be performed only once.

For each frame in the sequence, the images $\mathcal{I}_i$ currently viewed by the cameras are retrieved from hard disk and loaded as texture images $T_i^{\mathcal{I}}$ onto the graphics card. Since we want to use for texturing only those pixels within the silhouettes, we assign to all other pixels an alpha value of zero denoting full transparency, while an alpha value of one is assigned to all silhouette pixels. In order to smooth the blending in areas where the texture is in transition from one camera image to the next, an alpha value of 0.5 is assigned to pixels belonging to the boundary of the silhouettes.

During the last step the level set is rendered. This is the only part of the algorithm which depends on the current viewing direction. The following is performed for each cell: Let $d$ be the cell's diameter and $v$ its center. We select two cameras $j$ and $k$ according to the following criteria:

- The cell is fully visible in $\mathcal{I}_j$ and $\mathcal{I}_k$, and
- The angles $\alpha_j$ and $\alpha_k$ between the viewing direction and the optical axes of cameras $j$ and $k$, respectively, are minimal.

Note that in general, the selection of the two different cameras depends on $v$, so different cells are textured by different cameras. The billboard will be textured

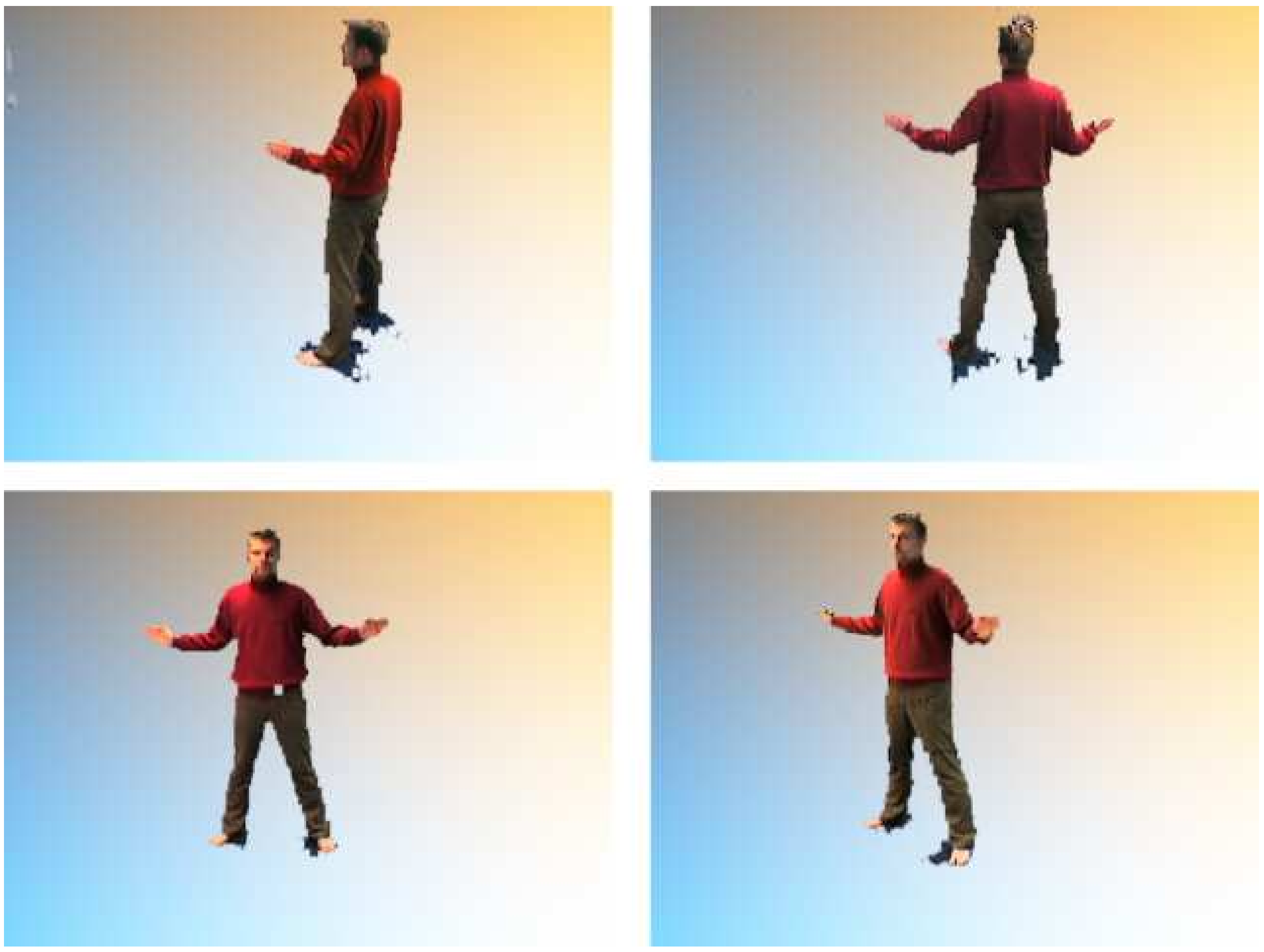


**Fig. 13.3.** *Segmented source images.* These are four of the six images used to construct the visual hull and texture the billboards. The cameras are spaced roughly 60 degrees apart.

with the images of these two cameras, which are blended depending on the similarity of the camera's optical axes to the current viewing direction. The blending weights $\omega_j$ and $\omega_k$ for the cameras are set to

$$\omega_{j,k} \;:=\; 1 - \frac{\alpha_{j,k}}{\alpha_j + \alpha_k}.$$

This way, a camera's image is reproduced exactly if the viewing direction coincides with its optical axis, and transitions are reasonably smooth when the selection of the two cameras changes due to a change in viewing direction – although not perfectly so, since to obtain smooth transitions everywhere requires knowledge about all the boundary lines between different camera selections, which is too much computational effort to determine.

Rendering the billboard requires two passes and at least two hardware texture units. In the first pass, the texture $T_j^{\pi}$ is set up in texture unit 1 with dependent texturing enabled. Unit 1 computes texture coordinates for texture unit 2, to which we assign the camera image $T_j^{\mathcal{I}}$. Blending is enabled to ensure that background pixels are transparent, and billboards have to be drawn in back-to-front order to correctly blend them one upon the other. The geometry transferred for each cell of $\Gamma$ is a single billboard centered on $V$ parallel to the current image plane and of size $\sqrt{3}d \times \sqrt{3}d$ in order to cover the projection of the cell with the projection of the billboard in the worst case. With texture coordinates set equal to the vertex positions, the graphics hardware now takes care of the rest.

Similarly, in the second pass, $T_k^{\pi}$ in unit 1 outputs texture coordinates for $T_k^{\mathcal{I}}$ in unit 2. We now have to blend correctly between the contribution of the first camera and this one, so the blending weight $\omega_k$ of the second image relative to the first one is stored in the alpha channel of the current color, which is modulated by the output of texture unit 2. The billboard is then rendered again. The correct setup for the texture shaders and environments in an example implementation using nVidia's OpenGL extension `NV_texture_shader` is summarized in Fig. 13.2. This setup can also be used for the first pass with the primary color's alpha value set to one, so no time-consuming state changes are required.

One also has to take great care to minimize the number of changes in texture images to optimize caching. In order to achieve this, a slight modification of the above scheme can be applied: The multiple textures for the source images are pasted on top of each other into one large single texture, where the $v$ coordinate is translated depending on the current camera. The same is done with the textures for the mappings. That way, texture images have to be selected just once before rendering all of the geometry.

## 13.3 Results

In our method, besides the negligible amount of geometry, data transfer from memory to the graphics card is limited to six very efficient texture loads per frame, one for each camera image. This amounts to 1.8MB of raw data in the case of 320 x 240 RGBA source images. It might be further reduced by first selecting the cameras which are able to contribute to the scene depending on the current viewpoint. However, it does not appear too much in view of the AGP 4x peak bandwidth, which lies in regions of 1GB per second. Instead, retrieving the data from hard drive fast enough is the bottleneck and requires a sophisticated compression scheme.

After all data has been transferred, our implementation is able to render a $64 \times 64 \times 64$ voxel volume from an arbitrary viewpoint at 30 fps. For each voxel, the contributions from the two nearest cameras are blended. The time to render each frame scales linearly with the number of visible voxels and almost linearly with the number of source cameras - note that the projected

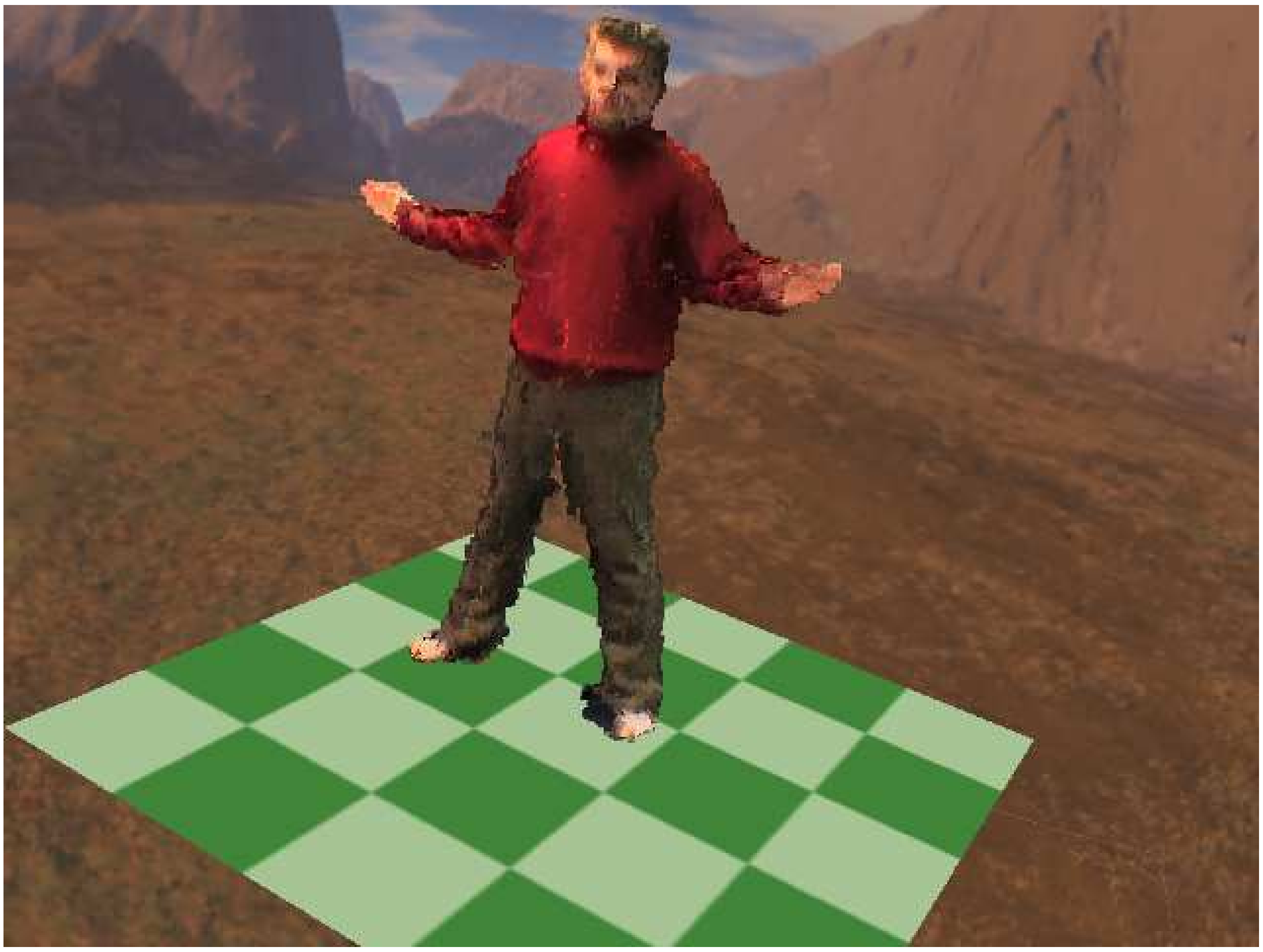


**Fig. 13.4.** *Rendered view.* The novel viewpoint lies in-between two of the source cameras of Fig. 13.3.

coordinates of each voxel have to be computed only once, regardless of the number of cameras. The program runs on a 1.8GHz Pentium IV Xeon with a GeForce 4 Ti 4600 graphics card. If no dependent texturing is available on a system, the mapping from 3D-space to the camera images can also be computed for each vertex by the CPU or a vertex program, decreasing performance slightly. The only real drawback of the algorithm is the limited resolution of the texture color channels: Since image coordinates are currently converted into 8-bit values, the discretized range of $\pi$ consists of only $256 \times 256$ distinct points. Higher image resolution gives more detail, nevertheless, because coordinates in-between arise from bilinear interpolation on the graphics card. Furthermore, already announced graphics hardware will support textures with higher dynamic range, probably up to floating-point accuracy, thus nullifying the problem. On even more modern hardware, it is likely that the texture coordinates can be computed entirely in the vertex program without notably impacting performance, making pre-computation unnecessary.

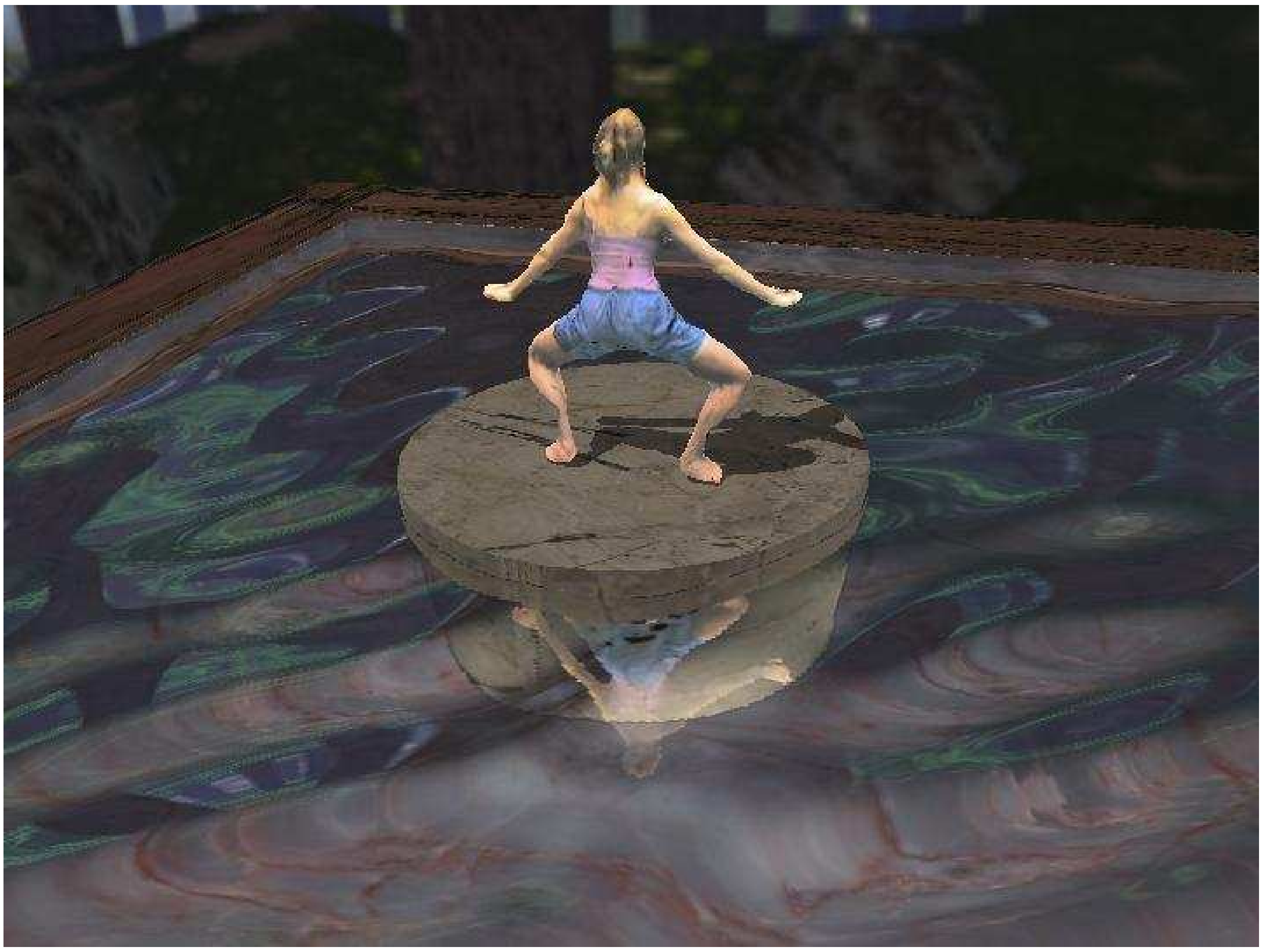


**Fig. 13.5.** *Rendered view from the ballet dancer sequence.* The novel viewpoint again lies in-between two of the source cameras. Geometry was reconstructed using our spatio-temporal reconstruction from Chapter 9.

Fig. 13.4 displays a rendered image of the person from a novel viewpoint directly in between two of the source cameras having an angular distance of about 60 degrees. Note that despite the low resolution of $64 \times 64 \times 64$ voxels, much finer details are visible. There are also few noticable transitions between voxels textured by different cameras, and the overall sharpness compared in view of sharpness and resolution of the source images in Fig. 13.3 is good. More rendering results were already presented in the previous part, Fig. 9.7.

It is important to note that although we use a voxel model of the visual hull to approximate the geometry, the actual rendering algorithm is not limited to this representation. Instead, it can use a polygonal model of the visual hull using exactly the same texture setup, or in fact almost any other geometry proxy.

## 13.4 Conclusions

The algorithm we presented employs a voxel-based representation of the visual hull as a geometry proxy for fast image-based rendering. By splatting small texture patches onto rectangular billboards, intricate details in the source images are preserved even in the case of coarse geometry data. The rendering algorithm can also handle other approximations to the scene geometry besides the visual hull. Real-time frame-rates of 30 Hz are achieved on current off-the-shelf hardware for $64 \times 64 \times 64$ voxel volumes.

Abundant applications exist in free-viewpoint television. The next step to make this rendering technique feasible in practice are novel compression methods which are especially tuned to the data required for the real-time rendering.

# 14

# Discussion and Conclusions

The driving motivation of this thesis is to be able to render novel views for video streams captured with a handfull of calibrated cameras. Many problems have to be solved in order to achieve this goal. We focused on the area of 3D reconstruction, since an approximation to the scene geometry is essential in order to be able to synthesise high-quality images from different vantage points. Interactive rendering methods which can be employed once suitable pre-computed geometry information is available were presented as well. Different problems have been identified which have to be solved depending on the kind of available camera setup. A common denominator is the question of how to adress temporal coherence, which is a major visual clue in video sequences. Any algorithm solving the reconstruction problem for videos can greatly enhance the quality of its results by incorporating ways to exploit this clue.

## 14.1 Summary

We will quickly summarize our algorithms, discuss their advantages and drawbacks, and show the advance over existing techniques.

### 14.1.1 Depth Reconstruction

For video sequences captured by cameras with parallel optical axes, we have presented an algorithm which simultaneously estimates the depth of each pixel in a scene, and whether or not it belongs to the moving foreground or static background. Unlike previous approaches, the system enforces temporal consistency among all video frames and requires no user-defined parameter values. All images are treated symetrically. Several visual clues, including pixel grouping and motion, are statistically analyzed by a pre-processing stage, and integrated into a global energy functional, whose minimum yields the reconstruction result. The minimization can be performed using the well-known, powerful graph cut technique.

For achieving optimal results, however, we currently require availability of background clean plates. We also cannot handle non-Lambertian surfaces properly, a common drawback of any approach based on local matching.

### 14.1.2 Surface Reconstruction

Our surface reconstruction algorithm is designed for setups where the cameras surround a dynamic object whose geometry is to be reconstructed for every frame of a video sequence. We introduce the idea, previously unexplored in computer vision, to represent the time-varying surface as a single hypersurface, whose slices with planes of constant time yield the geometry for each time instant. Our approach recovers the hypersurface as a weighted minimal surface of an error functional, which optimizes photo-consistency with all input images simultaneously. Temporal coherence follows implicitly from spatio-temporal continuity of the model. The Euler-Lagrange equation we derive for the general case of error function we require was to our knowledge previously unknown. It leads to a PDE evolution algorithm to derive a local minimum, which can be implemented for instance using a level set representation.

Since our photo-consistency criterion relies on pixel color values again, we have the same drawback of being only able to handle Lambertian surfaces. In addition, as the initial surface for the PDE evolution we employ the visual hull, which is prone to errors in the segmentation of the input images. Although we can start with a volume guaranteed to surround the whole scene instead, the algorithm will then require a lot more processing time until convergence. Because the computational complexity of our reconstruction scheme is high in any case, a parallel implementation and appropriate hardware is currently mandatory.

### 14.1.3 Video-based Rendering

For both kinds of geometry we reconstructed, depth maps and surfaces, respectively, we designed video-based rendering techniques specifically for the purpose of real-time rendering. Our dynamic light field rendering application can render from video streams with accompanying depth maps at a rate of 20 frames per second if the novel view is computed from four source images. When combined with the temporally coherent depth maps and background segmentation presented earlier, we can render foreground and background separately to achieve crisp contours and smooth viewing experience without virtually no flickering artifacts. The viewpoint, however, is limited to the window spawned by the recording cameras.

If the cameras surround the scene and we have per-frame geometry available, as for instance acquired with our surface reconstruction techniques, we can lift all constraints on the placement of the novel viewpoint. For triangle mesh representations, one can render the surface directly, using projective texturing to place details from the input images onto the surface. We also presented a billboard rendering technique to create novel views from level set representations.

Using optimizations we presented, one can reduce the effect of artifacts caused by texture boundaries. However, they cannot be completely eliminated, in particular if the resolution of the source videos is not high enough or there are errors in the segmentation. In general, all other errors in 3D reconstruction can naturally lead to severe artifacts as well, which cannot be alleviated in the rendering stage.

## 14.2 Future Work

We believe that this thesis has brought 3D reconstruction from multi-video footage a step forward, but still a lot of research remains to be done. Two issues in particular demand further attention. The first one is the current constraint of Lambertian reflection only, which clearly has to be lifted in order to be able to work with natural scenes instead of the studio recordings in controlled environments. Already, our algorithms could handle the matching problem if the BRDF or the material as well as the lighting is exactly known. However, lighting and/or BRDF estimation during reconstruction is something to be desired.

The second problem which is very important to solve is the recovery of the scene flow. So far, our spatio-temporal reconstruction technique only reconstructs the space-time geometry hypersurface, but not the actual correspondences between surface points at different time instants. We believe that this problem can be addressed by solving the partial differential equation for the scene flow on the surface in a similar way as the optic flow equation in a flat image, by adapting the best current PDE solvers for the flat geometry to arbitrary manifolds. In the end, we would like to formulate a coupled set of PDEs for combined recovery and optimization of surface geometry and scene flow. To be sure, this program will likely keep us and many others working in this exciting field busy for a long time to come.

# References

1. L. Ahrenberg, I. Ihrke, and M. Magnor. Volumetric reconstruction, compression and rendering of natural phenomena from multi-video data. In *Proc. International Workshop on Volume Graphics*, pages 544–552. Eurographics, 2005.
2. H. W. Alt. Analysis III. Lecture notes, Institut für angewandte Mathematik der Universität Bonn, 2002. *http://www.iam.uni-bonn.de/∼alt/ws2001/analysis3.html.*
3. L. Alvarez, R. Deriche, J. Sánchez, and J. Weickert. Dense disparity map estimation respecting image discontinuities: A PDE and scale-space based approach. *Visual Communication and Image Representation*, 13(1/2):3–21, 2002.
4. S. T. Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, 1989.
5. Y. Bayakovski, L. Levkovich-Maslyuk, A. Ignatenko, A. Konushin, D. Timasov, A. Zhirkov, M. Han, and I. K. Park. Depth image-based representations for static and animated 3D objects. In *Proc. International Conference on Image Processing*, pages 25–28. IEEE, 2002.
6. P. N. Belhumeur. A Bayesian approach to binocular stereopsis. *International Journal of Computer Vision*, 19(3):237–260, 1996.
7. S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. In *Proc. International Conference on Computer Vision*, pages 1073–1080, Bombay, India, 1998. IEEE.
8. S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Proc. International Conference on Computer Vision*, pages 489–495, Kerkyra, Korfu, Greece, 1999. IEEE.
9. Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401–406, 1998.
10. M. J. Black and A. Rangarajan. On the unification of line processes, outlier rejection and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19(1):57–91, 1996.

11. A. Blake and A. Zisserman. *Visual Reconstruction.* The MIT Press, Cambridge, Massachusetts, 1987.
12. A. F. Bobick and S. S. Intille. Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181–200, 1999.
13. Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.
14. B. Caprile and V. Torre. Using vanishing points for camera calibration. *International Journal of Computer Vision*, 4(2):127–139, 1986.
15. R. Carceroni and K. Kutulakos. Multi-view scene capture by sufel sampling: from video streams to non-rigid 3d motion, shape & reflectance. In *Proc. International Conference on Computer Vision*, pages 60–71, Vancouver, Canada, 2001. IEEE.
16. J. Carranza, C. Theobalt, M. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 569–577, 2003.
17. V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *Proc. International Conference on Computer Vision*, pages 694–699. IEEE, 1995.
18. V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert. Three dimensional object modeling via minimal surfaces. In *Proc. European Conference on Computer Vision*, pages 97–106, Cambridge, UK, 1996. Springer Verlag.
19. V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert. Minimal surfaces based object segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):394–398, 1997.
20. J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum. Plenoptic sampling. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 307–318, New Orleans, Louisiana, 2000.
21. Y.G. Chen, Y. Giga, and S. Goto. Uniqueness and existence of viscosity solutions of generalized mean curvature flow. *Journal of Differential Geometry*, 33:749–786, 1991.
22. K. M. Cheung, T. Kanade, J.-Y. Bouget, and M. Holler. A real time system for robust 3D voxel reconstruction of human motions. In *Proc. Computer Vision and Patter Recognition*, pages 714–720, Hilton Head, South Carolina, 2000. IEEE.
23. D. Chop. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106:77–91, 1993.
24. P. B. Chou and C. M. Brown. The theory and practice of Bayesian image labeling. *International Journal of Computer Vision*, 4(3):185–210, 1990.
25. Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David Salesin, and Richard Szeliski. Video matting of complex scenes. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 243–248, San Antonio, Texas, 2002.
26. J. N. Clelland. Msri workshop on Lie groups and the method of moving frames. Lecture notes, Department of Mathematics, University of Colorado, 1999. *http://spot.Colorado.EDU/~jnc/MSRI.html.*

27. S. Cochran and G. Medioni. 3D surface description from binocular stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):366–369, 1992.
28. W. P. Cockshott, S. Hoff, and J. C. Nebel. An experimental 3D digital TV studio. In *Proc. Vision, Image & Signal Processing*, pages 28–33, 2003.
29. I. J. Cox, S. Roy, and S. L. Hingorani. Dynamic histogram warping of image pairs for constant image brightness. In *Proc. International Conference on Image Processing*, pages 366–369, Washington, D.C., 1995. IEEE.
30. Y. Duan, L. Yang, H. Qin, and D. Samaras. Shape reconstruction from 3D and 2D data using PDE-based deformable surfaces. In *Proc. European Conference on Computer Vision*, pages 238–246, Prague, Czech Republic, 2004. Springer Verlag.
31. O. Faugeras and R. Keriven. Variational principles, surface evolution, PDE's, level set methods and the stereo problem. *IEEE Transactions on Image Processing*, 3(7):336–344, March 1998.
32. O. Faugeras and Q.-T. Luong. *The Geometry of Multiple Images.* The MIT Press, Cambridge, Massachusetts, 2001.
33. O. D. Faugeras and R. Keriven. Complete dense stereovision using level set methods. In *Proc. European Conference on Computer Vision*, pages 379–393, Freiburg, Germany, 1998. Springer Verlag.
34. P. Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6(1):35–49, 1993.
35. D. Geiger and F. Girosi. Parallel and deterministic algorithms for mrf's: Surface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):401–412, 1991.
36. S. Geman and D. Geman. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.
37. B. Goldluecke and M. Magnor. Hardware-accelerated dynamic light field rendering. In *Proc. Vision, Modeling and Visualisation*, pages 455–462, Erlangen, Germany, 2002.
38. B. Goldluecke and M. Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. In *Proc. International Conference on Image Processing*, pages 713–716, Barcelona, Spain, 2003. IEEE.
39. B. Goldluecke and M. Magnor. Space-time isosurface evolution for temporally coherent 3D reconstruction. In *Proc. Computer Vision and Pattern Recognition*, pages 350–355, Washington, D.C., 2004. IEEE.
40. B. Goldluecke and M. Magnor. Weighted minimal hypersurfaces and their applications in computer vision. In *Proc. European Conference on Computer Vision*, pages 366–378, Prague, Czech Republic, 2004. Springer Verlag.
41. B. Goldluecke and M. Magnor. Space-time continuous geometry meshes from multi-view video sequences. In *Proc. International Conference on Image Processing*, pages 516–522, Genova, Italy, 2005. IEEE.
42. Bastian Goldluecke and Marcus Magnor. Joint 3D reconstruction and background separation in multiple views using graph cuts. In *Proc. Computer Vision and Patter Recognition*, pages 683–694, Madison, Wisconsin, 2003. IEEE.

43. Bastian Goldluecke and Marcus Magnor. Real-time, free-viewpoint video rendering from volumetric geometry. In *Proc. Visual Communications and Image Processing (VCIP)*, volume 5150 (2), pages 1152–1158, Lugano, Switzerland, 2003. SPIE.
44. S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 43–54, New Orleans, USA, 1996.
45. W. E. L. Grimson. Computational experiments with a feature-based stereo algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(1):17–34, 1985.
46. M. Gross, S. Würmlin, M. Näf, E. Lamboray, C. Spagno, A. Kunz, A. Moere, K. Strehlke, S. Lang, T. Svoboda, E. Koller-Meier, L. van Gool, and O. Staadt. *blue-c*: A spatially immersive display and 3D video portal for telepresence. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 819–827, 2003.
47. M. J. Hannah. *Computer Matching of Areas in Stereo Images.* PhD thesis, Stanford University, 1974.
48. R. I. Hartley. Kruppa's equations derived from the fundamental matrix. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):133–135, 1997.
49. S. W. Hasinoff and K. N. Kutulakos. Photo-consistent 3d fire by flame-sheet decomposition. In *Proc. International Conference on Computer Vision*, pages 1184 – 1191, Nice, France, 2003. IEEE.
50. I. Ihrke, B. Goldluecke, and M. Magnor. Reconstructing the geometry of flowing water. In *Proc. International Conference on Computer Vision*, page *accepted for publication*, Beijing, China, 2005. IEEE.
51. I. Ihrke and M. Magnor. Image-Based Tomographic Reconstruction of Flames . In *Proc. Eurographics Symposium on Computer Animation*, pages 367–375, Grenoble, France, 2004. ACM Siggraph.
52. H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *Proc. European Conference on Computer Vision*, pages 232–248, Freiburg, Germany, 1998. Springer Verlag.
53. H. Jin, S. Soatto, and A. J. Yezzi. Multi-view stereo beyond Lambert. In *Proc. Computer Vision and Patter Recognition*, pages 171–178, Madison, Wisconsin, 2003. IEEE.
54. D. G. Jones and J. Malik. A computational framework for determining stereo correspondence from a set of linear spatial filters. In *Proc. European Conference on Image Processing*, pages 397–410, Italy, 1992. Springer Verlag.
55. T. Kanade. Development of a video-rate stereo machine. In *Image Understanding Workshop*, pages 549–557, Monterey, CA, 1994. Morgan Kaufman Publishers.
56. T. Kanade and M. Okutomi. Development of a video-rate stereo machine. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920–932, 1994.
57. T. Kanade, P. Rander, and P. J. Naranayan. Virtualized reality: Construction of virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–47, 1997.

58. S. B. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multi-view stereo. In *Proc. Computer Vision and Patter Recognition*, pages 280–288, Kauai, Hawaii, 2001. IEEE.
59. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1(2):321–331, 1988.
60. D. Kelly. *Digital Composition.* The Coriolis Group, Scottsdale, Arizona, 2000.
61. V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *Proc. European Conference on Computer Vision*, pages 82–96, Copenhagen, Denmark, 2002. Springer Verlag.
62. V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? In *Proc. European Conference on Computer Vision*, pages 65–81, Copenhagen, Denmark, 2002. Springer Verlag.
63. H. Kueck, W. Heidrich, and C. Vogelgsang. Shape from contours and multiple stereo: a hierarchical, mesh-based approach. In *1st Canadian Conference on Computer and Robot Vision*, pages 76–83. IEEE, 2004.
64. K. N. Kutukalos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):197–216, 2000.
65. A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Recognition*, 16(2):150–162, 1994.
66. M. Levoy and P. Hanrahan. Light field rendering. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 31–42, New Orleans, USA, 1996.
67. M. Li, M. Magnor, and H.-P. Seidel. Hardware-acccelerated rendering of photo hulls. *Computer Graphics Forum*, 23(3):635–642, 2004.
68. M. Li, M. Magnor, and H.-P. Seidel. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *Proc. Graphics Interface*, pages 41–48, London, Ontario, Canada, 2004.
69. C. Linz. PDE-based surface reconstruction from multiple views using a surfel model. Master's thesis, MPI Informatik, Saarbrücken, Germany, 2005.
70. Y. Lu, Z. Zhang, Q. M. Wu, and Z. Li. A survey of motion-parallax-based 3D reconstruction algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 34(4):532–548, November 2004.
71. M. J. Magee and J. K. Aggarwal. Determining vanishing points from perspective images. *Comput. Vis., Graph., Image Process.*, 26:256–267, September 1984.
72. M. Magnor and B. Goldluecke. Spacetime-coherent geometry reconstruction from multiple video streams. In *2nd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)*, pages 365–372, Thessaloniki, Greece, 2004. IEEE.
73. D. Marr. *Vision.* W. H. Freeman, 1982.
74. D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194:283–287, October 1976.
75. D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings Royal Society of London*, B204:304–328, 1979.

76. W. Martin and J. K. Aggarwal. Volumetric description of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158, 1983.
77. W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for realtime rendering. In *Proc. 12th Eurographics Workshop on Rendering*, pages 115–125, 2001.
78. W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 369–374, New Orleans, USA, 2000.
79. W. Matusik and H. Pfister. 3D TV: A scalable system for real-time acquisition, transmission and autostereoscopic display of dynamic scenes. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, 2004.
80. W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3D photography using opacity hulls. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 427–436, San Antonio, Texas, 2002.
81. S. J. Maybank and O. D. Faugeras. A theory of self-calibration of a moving camera. *International Journal of Computer Vision*, 8(2):123–151, 1992.
82. S. Moezzi, A. Katkere, D. Y. Kuramura, and R. Jain. Immersive video. In *Proc. Virtual Reality Annual International Symposium*, pages 17–24, Silver Spring, 1996. IEEE.
83. O. J. Morris, J. Lee, and A. G. Constantinides. Graph theory for image analysis: An approach based on the shortest spanning tree. In *IEE Proceedings*, volume F-133 (2), pages 146–152, 1986.
84. J. Mulligan and K. Daniilidis. View-independent scene acquisition for telepresence. In *International Symposium on Augmented Reality*, pages 105–110, 2000.
85. H. Murase. Surface Shape Reconstruction of a Nonrigid Transparent Object Using Refraction and Motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):1045–1052, 1992.
86. P. J. Narayanan, P. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *Proc. International Conference on Computer Vision*, pages 3–10, Bombay, India, 1998. IEEE.
87. M. Okutomi and T. Kanade. A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2):143–162, 1992.
88. P. J. Olver, G. Sapiro, and A. Tannenbaum. Invariant geometric evolutions of surfaces and volumetric smoothing. *SIAM Journal on Applied Mathematics*, 57(1):176–194, 1997.
89. R. Ooi, T. Hamamoto, T. Naemura, and K. Aizawa. Pixel independent random access image sensor for real time image-based rendering system. In *Proc. International Conference on Image Processing*, pages 193–196, Thessaloniki, Greece, 2001. IEEE.
90. S. Osher and J. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on the Hamilton-Jacobi formulation. *Journal of Computational Physics*, 79:12–49, 1988.

91. N. Paragios and R. Deriche. Geodesic active contours and level sets for the detection and tracking of moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):266–280, 2000.
92. S. Paris, H. M. Briceño, and F. X. Sillion. Capture of hair geometry from multiple images. *ACM Transactions on Graphics*, 23(3):712–719, August 2004.
93. C. Pintavirooj, A. Romputtal, A. Ngamlamiad, W. Withayachumnankul, and K. Hamamoto. Ultrasonic Refractive Index Tomography. *Journal of WSCG*, 12(2):333–339, February 2004.
94. R. Piroddi and T. Vlachos. Multiple-feature spatiotemporal segmentation of moving sequences using a rule-based approach. In *Proc. British Machine Vision Conference*, pages 353–362, Cardiff, UK, 2002.
95. S. Pollard and S. Hayes. View synthesis by edge transfer with application to the generation of immersive video objects. In *Proc. ACM Symposium on Virtual Reality Software and Technology*, pages 91–98, New York, USA, 1998. ACM SIGGRAPH.
96. M. Pollefeys and L. van Gool. Stratified self-calibration with the modulus constraint. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):707–724, 1999.
97. M. Potmesil. Generating octree models of 3D objects from their silhouettes. *Comput. Vis., Graph., Image Process.*, 40:1–29, January 1987.
98. K. Pradzny. Detection of binocular disparities. *Biological Cybernetics*, 52:93–99, 1985.
99. A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics*, 23(3):720–727, August 2004.
100. A. Redert, M. op de Beeck, C. Fehn, W. Ijsselsteijn, M. Pollefeys, L. van Gool, E. Ofek, I. Sexton, and P. Surman. ATTEST: Advanced three-dimensional television system technologies. In *3DPVT*, pages 313–319, 2002.
101. L. Robert and R. Deriche. Dense depth map reconstruction: A minimization and regularization approach which preserves discontinuities. In *Proc. European Conference on Computer Vision*, pages 439–451, Cambridge, UK, 1996. Springer Verlag.
102. C. Rother. A new approach for vanishing point detection in architectural environments. In *Proc. British Machine Vision Conf.*, pages 382–391, Bristol, UK, 2000.
103. S. Roy. Stereo without epipolar lines: A maximum flow formulation. *International Journal of Computer Vision*, 1(2):1–15, 1999.
104. S. Roy and I. Cox. A maximum-flow formulation of the $n$-camera stereo correspondence problem. In *Proc. International Conference on Computer Vision*, pages 492–499, Bombay, India, 1998. IEEE.
105. T. W. Ryan, R. T. Gray, and B. R. Hunt. Prediction of correlation errors in stereo-pair images. *Optical Engineering*, 19(3):312–322, 1980.
106. D. Scharstein. In *View Synthesis using Stereo Vision*, number 1583 in Lecture Notes in Computer Science. Springer Verlag, 1999.
107. D. Scharstein and R. Szeliski. Stereo matching with nonlinear diffusion. *International Journal of Computer Vision*, 28(2):155–174, 1998.

108. D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1–3):7–42, 2002.
109. H. Schirmacher, W. Heidrich, and H.-P. Seidel. High-quality interactive lumigraph rendering through warping. In *Proc. Graphics Interface*, pages 87–94, Montreal, Canada, 2000.
110. H. Schirmacher, M. Li, and H.-P. Seidel. On-the-fly processing of generalized lumigraphs. *Computer Graphics Forum*, 20:C165–C173;C543, 2001.
111. H. Schultz. Retrieving Shape Information from Multiple Images of a Specular Surface. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):195–201, 1994.
112. P. Seitz. Using local orientation information as image primitive for robust object recognition. In *Proc. Visual Communications and Image Processing (VCIP)*, volume IV, pages 1630–1639. SPIE, 1989.
113. S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Patter Recognition*, pages 1067–1073, San Juan, Puerto Rico, 1997. IEEE.
114. S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):1–23, 1999.
115. J. A. Sethian. *Level Set Methods and Fast Marching Methods.* Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, UK, 2nd edition, 1999.
116. J. Shade and S. Gortler. Layered depth images. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 231–242, 1998.
117. R. W. Sharpe. *Differential Geometry.* Graduate Texts in Mathematics. Springer, New York, 1997.
118. E. P. Simoncelli, E. H. Adelson, and D. J. Heeger. Probability distributions of optic flow. In *Proc. Computer Vision and Patter Recognition*, pages 310–315, Maui, Hawaii, 1991. IEEE.
119. P.-P. Sloan, M. Cohen, and S. Gortler. Time critical lumigraph rendering. In *Symposium on Interactive 3D Graphics*, pages 17–24, 1997.
120. A.R. Smith and J.F. Blinn. Blue screen matting. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 296–268, New Orleans, USA, 1996.
121. A. Smolic and H. Kimata. Description of exploration experiments in 3DAV. In *JTC1/SC29/WG11.* ISO/IEC, 2003.
122. D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *Proc. Computer Vision and Patter Recognition*, pages 345–352, Hilton Head, South Carolina, 2000. IEEE.
123. P. Sturm. Critical motion sequences for monocular self-calibration and uncalibrated Euclidean reconstruction. In *Proc. Computer Vision and Patter Recognition*, pages 1100–1105, San Juan, Puerto Rico, 1997. IEEE.
124. P. Sturm. A case against kruppa's equations for camera self-calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1199–1204, 2000.

125. P. Sturm and W. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *Proc. European Conference on Computer Vision*, pages 709–720, Cambridge, UK, 1996. Springer Verlag.
126. R. Szeliski. *Bayesian Modeling of Uncertainty in Low-level Vision.* Kluwer Academic Publishers, Boston, Massachusetts, 1989.
127. R. Szeliski. Rapid octree construction from image sequences. *Computer Vision, Graphics and Image Processing*, 58(2):23–32, July 1993.
128. R. Szeliski and G. Hinton. Solving random dot stereograms using the heat equation. In *Proc. Computer Vision and Patter Recognition*, pages 284–288, San Francisco, California, 1985. IEEE.
129. C. Theobalt, E. de Aguiar, M. Magnor, H. Theisel, and H.-P. Seidel. Marker-free kinematic skeleton estimation from sequences of volume data. In *Proc. of ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 57–64, Hong Kong, China, 2004.
130. C. Theobalt, M. Li, M. Magnor, and H.-P. Seidel. A flexible and versatile studio for synchronized multi-view video recording. Research Report MPI-I-2003-4-002, MPI Informatik, 2003.
131. C. Theobalt, M. Magnor, P. Schueler, and H. P. Seidel. Combining 2D feature tracking and volume reconstruction for online video-based human motion capture. In *Proc. Pacific Graphics*, pages 96–103, Beijing, China, 2002.
132. C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization approach. *International Journal of Computer Vision*, 9(2):137–154, 1992.
133. W. Triggs. Auto-calibration and the absolute quadric. In *Proc. Computer Vision and Patter Recognition*, pages 609–614, San Juan, Puerto Rico, 1997. IEEE.
134. R. Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. In *Proc. Computer Vision and Patter Recognition*, pages 364–374, Miami, Florida, 1986. IEEE.
135. S. Vedula, S. Baker, S. Seitz, and T. Kanade. Shape and motion carving in 6D. In *Proc. Computer Vision and Patter Recognition*, pages 592–598, Hilton Head, South Carolina, 2000. IEEE.
136. M. Waschbüsch, S. Würmlin, D. Cotting, F. Sadlo, and M. Gross. Scalable 3D video of dynamic scenes. In *Proc. Pacific Graphics*, 2005.
137. B. Wilburn, N. Joshi, V. Vaish, E. V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High-performance imaging using large camera arrays. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, 2005.
138. B. Wilburn, M. Smulski, K. Lee, and M. Horowitz. The light field video camera. In *SPIE Proc. Media Processors*, volume 4674, San Jose, California, 2002.
139. D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 287–296, New Orleans, USA, 2000.

140. S. Wuermlin, E. Lamboray, and M. Gross. 3D video fragments: Dynamic point samples for real-time free-viewpoint video. *Computers & Graphics*, 28(1):3–14, 2004.
141. S. Würmlin, E. Lamboray, O. Staadt, and M. Gross. 3D video recorder. In *Proc. Pacific Graphics*, pages 10–22, Beijing, China, 2002.
142. J. Xu and H. Zhao. An Eulerian formulation for solving partial differential equations along a moving interface. *SIAM Journal of Scientific Computing*, 19:573–594, 2003.
143. J. C. Yang, M. Everett, C. Buehler, and L. McMillan. A real-time distributed light field camera. In *Proc. 13th Eurographics Workshop on Rendering*, pages 77–86, Pisa, Italy, 2002.
144. R. Zabih and J. Woodfill. Non-parametric local transformations for computing cisual correspondence. In *Proc. European Conference on Computer Vision*, pages 151–158, Stockholm, Sweden, 1994. Springer Verlag.
145. H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *1st IEEE Workshop on Variational and Level Set Methods, 8th ICCV*, volume 80(3), pages 194–202, Vancouver, Canada, 2001.
146. H. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.
147. C. L. Zitnick, S. B. Kang, M. Uyttendale, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. In *Proc. ACM Conference on Computer Graphics (SIGGRAPH)*, Annual Conference Series, pages 548–558, 2004.
148. A. V. Zvyagin, D. Silva, S. A. Alexandrov, T. R. Hillman, and J. J. Armstrong. Refractive index tomography of turbid media by bifocal optical coherence refractometry. *Optics Express*, 11(25):3503–3517, December 2003.