Diss. ETH No. 17558

3D Video Acquisition, Representation & Editing

A dissertation submitted to
ETH Zurich

for the Degree of
Doctor of Sciences

presented by
Michael Waschbüsch
Diplom-Informatiker
born 15 January 1976
citizen of Germany

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner
Prof. Dr. Marcus Magnor, co-examiner
Dr. Stephan Würmlin, co-examiner

2007

Abstract

3D video is a new kind of media that captures three-dimensional appearance and dynamics of real-world scenes by combining data from multiple input video streams to a consistent model. During playback, it provides the viewer with the possibility of freely choosing the viewpoint of a virtual camera in space and time.

In this thesis, we present a 3D video system covering the full processing pipeline from acquisition over data representation, post-processing and editing up to high-quality display. The focus of our research lies on 3D video data representations. We investigate both image-based and geometric models. While the first show advantages in achievable output image quality, the latter provide more flexibility for applications beyond pure playback. In particular, we investigate methods for data streaming and compression, dynamic out-of-core data structures, and interactive 3D video editing.

For data acquisition we introduce a scalable system based on multiple sparsely placed 3D video bricks. Each brick captures high-quality depth maps of the scene from its respective viewpoint, using a combination of spacetime stereo vision and structured light projection. Texture images and pattern-augmented views are acquired simultaneously by time-multiplexed projections and synchronized camera exposures.

Our image space representation consisting of billboard planes augmented by detailed displacement maps combines the generality of acquired geometry with the regularization properties of purely image-based methods. Being placed in the disparity space of the acquisition cameras, the billboards provide a regular sampling of the scene with a uniform error model. Based on that, we propose a geometry filtering method which generates spatially and temporally coherent models and removes reconstruction noise as well as calibration errors. Rendering is performed using a GPU-accelerated algorithm which generates consistent view-dependent geometry and texture for each individual viewpoint.

Point samples are the fundamental primitive of our second representation. By carrying multiple surface attributes such as position and color, they provide a unified model of geometry and appearance of natural scenes. This representation allows for application of various post-processing algorithms for improving noisy input geometry. In particular, we propose a method that effectively removes outliers by enforcing photo consistency with all input views. By augmenting the points by a statistical model of acquisition noise, smooth images of the scene from novel viewpoints can be generated using a probabilistic renderer based on GPU-accelerated EWA volume splatting.

Point samples are ideally suited for streaming and compression. In contrast to mesh-based representations, no connectivity has to be encoded. We present a unified framework for compression of geometry and appearance of point-sampled models. Based on a multiresolution decomposition of the point cloud, it generates a data stream which easily allows for progressive decoding during transmission. Our method is generic in the sense that it can handle arbitrary point attributes using attribute-specific coding operators. In particular, we provide operators for point position, normal and color.

As an application beyond virtual playback we present a versatile system for interactive nonlinear editing of 3D video footage. It combines the advantages of conventional 2D video editing with the power of more advanced geometry-enhanced 3D video streams. Its underlying core data structure is a point-based representation of four-dimensional spacetime, which we call the video hypervolume. A dynamic out-of-core spatial indexing structure permits handling of large data sets that do not fit into main memory. Combined with a multiresolution hierarchy, access is possible at interactive rates. Conceptually, the editing loop comprises three fundamental operators: slicing, selection, and editing. The slicing operator allows users to visualize arbitrary subvolumes of the 4D data set. Visualization of both spatial and temporal domains allows for unified editing of all dimensions. The selection operator labels subsets of the footage for further modification. It includes a semi-automatic segmentation algorithm based on 4D graph cuts for convenient object selection. The actual editing operators include cut & paste, affine transformations, and shadow mapping. With those, a user is able to generate complex visual effects. Thus, 3D video provides a novel tool for video post-production, yielding possibilities for special effects that would be difficult or impossible to achieve with conventional two-dimensional imaging technologies.

Kurzfassung

3D-Video ist eine neue Medienform, die das dreidimensionale Erscheinungsbild sowie die Dynamik realer Szenen erfasst, indem bei der Aufnahme mehrere Videoströme zu einem konsistenten Modell zusammengefasst werden. Während der Wiedergabe bietet es dem Betrachter eine virtuelle Kamera, deren Position und Blickrichtung beliebig in Raum und Zeit festgelegt werden können.

In dieser Dissertation präsentieren wir ein 3D-Video-System, welches die vollständige Verarbeitungskette von der Aufnahme über die Datenverarbeitung und das manuelle Editieren bis hin zur hochqualitativen Bildgenerierung umfasst. Der Fokus unserer Forschung liegt auf 3D-Video-Datenmodellen. Wir untersuchen sowohl bildbasierte als auch geometrische Repräsentationen. Während erstere eine bessere Bildqualität ermöglichen, bieten letztere mehr Flexibilität für Anwendungen, die über das reine Abspielen hinausgehen. Dort untersuchen wir insbesondere Methoden zur Kompression und Datenübertragung, dynamische, externe Datenstrukturen sowie interaktives Editieren von 3D-Video.

Zur Datenakquisition stellen wir ein skalierbares System basierend auf mehreren, spärlich platzierten 3D-Video-*Bricks* vor. Jeder *Brick* erfasst hochqualitative Tiefenkarten der Szene aus seiner individuellen Blickrichtung mit Hilfe eines raumzeitlichen Stereo-Algorithmus in Kombination mit Projektionen strukturierten Lichts. Durch eine spezielle Synchronisation der Projektionen mit den Kameraverschlüssen können sowohl Texturen der Szene als auch die Lichtmuster fast gleichzeitig im Zeitmultiplex aufgenommen werden.

Unser bildbasiertes Datenmodell besteht aus *Billboard*-Ebenen und dazugehörigen Texturen mit detaillierten Farb- und Höheninformationen. Es kombiniert die Allgemeingültigkeit individuell rekonstruierter Geometrie mit den Regularisierungseigenschaften rein bildbasierter Verfahren. Durch ihre Definition im Raum der Disparitäten der Aufnahmekameras bieten die *Billboards* eine reguläre Abtastung der Szene mit einem uniformen Fehlermodell. Basierend darauf kann ein Geometrie-Filter erstellt werden, der sowohl Rauschen als auch Kalibrierungsfehler räumlich und zeitlich kohärent beseitigt. Zur Bildgenerierung wird ein hardwarebeschleunigter Algorithmus eingesetzt, der individuell von der Blickrichtung abhängige, konsistente Geometrie und Textur erzeugt.

Punktprimitive bilden das fundamentale Element unseres zweiten Datenmodells. Sie modellieren dreidimensionale Oberflächen durch diskrete Punkte, indem sie gleichzeitig verschiedene lokale Attribute wie z.B. Position und Farbe speichern, und ermöglichen somit eine einheitliche Repräsentation von Geometrie und Erscheinungsbild natürlicher Szenen. Sie erlauben die Anwendung einer

Vielzahl von Algorithmen zur Nachbearbeitung und Verbesserung der rekonstruierten Geometrie. Insbesondere stellen wir eine Methode zur effektiven Entfernung von Ausreißern vor, indem die photometrische Konsistenz des Modells mit den Eingabebildern sichergestellt wird. Ein in der Repräsentation enthaltenes, statistisches Fehlermodell ermöglicht die Generierung sauberer Bilder der Szene aus neuen Blickrichtungen mit Hilfe eines probabilistischen, hardwarebeschleunigten Verfahrens basierend auf volumetrischem *EWA-Splatting*.

Punkte eignen sich ideal zur komprimierten Datenübertragung. Im Gegensatz zu Dreiecksnetzen muss keine Konnektivität gespeichert werden. Wir präsentieren ein einheitliches System zur Komprimierung von Geometrie und Erscheinungsbild punktbasierter Modelle. Durch eine hierarchische Dekomposition der Punktwolke wird ein Datenstrom generiert, der auf einfache Weise während der Übertragung progressiv dekodiert werden kann. Indem man ihr spezielle Kodierungsoperatoren zur Verfügung stellt, kann unsere generische Methode beliebige Punkt-Attribute komprimieren. Wir stellen Operatoren für Position, Farbe und Oberflächennormale zur Verfügung.

Als eine Anwendung, die über die reine Wiedergabe hinausgeht, präsentieren wir ein System zum interaktiven nichtlinearen Editieren von 3D-Video. Es kombiniert die Vorteile konventioneller 2D-Video-Nachbearbeitung mit denen der in 3D-Video enthaltenen Geometrieinformationen. Es basiert auf einer punktbasierten Repräsentation der vierdimensionalen Raumzeit, dem so genannten Video-Hypervolumen. Eine dynamische, externe Indexstruktur ermöglicht die Verarbeitung großer Datensätze, die nicht in den Hauptspeicher passen. Kombiniert mit einer Auflösungshierarchie ist ein interaktiver Zugriff möglich. Die Videobearbeitung läuft konzeptionell in einer Schleife bestehend aus drei fundamentalen Operatoren ab: *Slicing*, Selektion und Editieren. Der *Slicing*-Operator ermöglicht es dem Benutzer, beliebige Teilvolumen des vierdimensionalen Datensatzes zu visualisieren. Ansichten sowohl der räumlichen als auch der zeitlichen Domänen ermöglichen eine einheitliche Bearbeitung aller Dimensionen. Die Selektionsoperatoren markieren Bereiche des Datensatzes für die weitere Bearbeitung. Sie beinhalten einen halbautomatischen Segmentierungsalgorithmus basierend auf einem vierdimensionalen Graphenschnitt-Verfahren. Zum eigentlichen Editieren stehen unter anderem Operatoren zum Ausschneiden und Einfügen, affine Transformationen sowie Werkzeuge zum Einfügen künstlicher Schlagschatten zur Verfügung. Mit diesen kann ein Benutzer auf einfache Weise komplexe Effekte erzeugen. Damit eignet sich 3D-Video als ein neues Werkzeug zur Videoproduktion, das das Erstellen von Spezialeffekten ermöglicht, die mit konventionellem, zweidimensionalem Video nur schwierig oder überhaupt nicht zu erreichen wären.

Acknowledgments

I would like to express my gratitude to my advisor Markus Gross for giving me the opportunity to work in such a stimulating and challenging environment, and for his guidance and support during the development of this thesis.

My sincere thanks go to my co-advisors: to Stephan Würmlin for his continuous advice, insightful discussions, and critical observations, which helped to keep me on the right track during the last four years, and to Marcus Magnor for his helpful comments.

Many people assisted to successfully complete this project. I am especially obliged to Daniel Cotting for our enjoyable and fruitful collaboration, Filip Sadlo for helping with the camera calibration, Edouard Lamboray for contributing to the point compression project, Doo Young Kwon and Vanessa Stadler for acting in the 3D video recordings, and Christoph Niederberger for video cutting. Additionally, many thanks to all students who contributed to parts of the implementation (in alphabetical order): Rolf Adelsberger, Oliver Büechi, Michael Duller, Felix Eberhard, Claudio Hatz, Patrick Jenni, Peter Kaufmann, Roger Küng, and Stefan Rondinelli.

My special thanks go to all my colleagues and friends at ETH Zurich for giving me daily support and for creating such a pleasant working environment: Bernd Bickel, Mario Botsch, Manuela Cavegn, Gaël Guennebaud, Simon Heinzle, Richard Keiser, Silke Lang, Martin Näf, Miguel Otaduy, Mark Pauly, Ronny Peikert, Christian Sigg, Denis Steinemann, Bob Sumner, Nils Thürey, Tim Weyrich, Martin Wicke, and Remo Ziegler.

Last but not least, let me thank Alexander Keller, who sparked my interest in the fascinating world of Computer Graphics and who encouraged me to do a PhD in that field.

This work has been carried out in the context of the blue-c-II project, funded by ETH grant No. 0-21020-04 as an internal poly-project.

Contents

Figures

Tables

Chapter 1

Introduction

3D video extends conventional video technologies by capturing three-dimensional appearance and dynamics of real-world scenes. During playback, it provides the viewer with the possibility to change the viewpoint at will. Moreover, 3D video is a new tool for video post-production. Besides creation of novel visual effects, available information about the three-dimensional scene structure greatly simplifies conventional video editing such as scene compositing.

This chapter gives a motivation for our research on 3D video and discusses our novel contributions to that field. After summarizing the publications that have been released during our research, it closes with an overview over the remaining parts of this thesis.

1.1 Motivation

Being able to digitally experience real-world locations and events from arbitrary viewpoints and viewing directions has interesting applications in next-generation television, video games, virtual reality and movie production. Imagine watching a sports play on television always from the best perspective by interactively choosing any viewpoint in a sports stadium using your remote control. For such experiences the scene has to be captured first using a set of digital cameras. Then, the multi-view image or video data has to be processed to some representation of the scene that allows virtual views to be synthesized.

Such technology is usually referred to as 3D photography if the scene is static, or 3D video if the scene is captured in motion. It can be categorized as a discipline of visual computing [Gross, 1994] as it uses algorithms from both computer graphics and computer vision. It is closely linked to the area of computational photography [Raskar and Tumblin, 2007], where multiple images or are often used to manipulate the appearance of a scene viewed from a single position for example by modifying the illumination [Sen et al., 2005] or by changing the optical parameters of the camera [Ng et al., 2005; Wilburn et al., 2005]. In contrast to that,

we focus our research on the geometry of a scene and investigate applications like replay from a virtual camera position and interactive content manipulation. This field is also often referred to as free-viewpoint video.

Virtual replays as well as freeze-and-rotate visual effects become feasible with fully automated processes using video cameras as input only. However, high-quality results are not achieved easily for arbitrary setups. Today, such effects typically have to be planned precisely and changes are no more feasible after the scene has been shot. As an example, Digital Air's® Movia® digital camera systems comprise high-speed, high-definition digital cinema cameras which are placed accurately such that no software view interpolation is needed. On the other hand, LiberoVision's® DiscoverEye® camera system is able to use video streams of conventional television cameras. However, it is currently specialized to broadcasts of soccer games only.

In this thesis, we would like to alleviate those constraints. We present a 3D video system covering the full processing pipeline from acquisition over data representation and editing up to high-quality display. Our work is motivated by bringing 3D video to a new degree where not only capturing and subsequent high-quality re-rendering is cost-effective, convenient and scalable but also editing is easy to do.

The focus of our research lies on 3D video data representations. They strongly influence the final image quality. Geometry-based representations are typically view-independent and can be used for arbitrary camera configurations. However, they often do not allow for production-quality results. Image-based representations allow to synthesize high-quality images even of highly complex scenes. However, the lack of detailed geometry and the need for very dense camera setups limit its practical applicability.

Moreover, the data representation determines the applicability of 3D video for an end user. To bring the new media to the masses, efficient data distribution methodologies have to be developed, including algorithms for streaming and compression of 3D video content. Scale, resolution, and access patterns during playback impose new research challenges. 3D video in principle allows a user to choose both wide-angle views on large-scale scenes and closeup viewpoints showing high-resolution details. The sampling density required for display can change over several orders of magnitude in such situations.

In our work we investigate various representations considering aspects like effective post-processing, high-quality rendering, streaming, compression, and flexibility for editing. To a large part we use point samples. Being able to store a variety of attributes such as geometry and color, they provide a homogeneous, view-independent representation for natural scenes. Hence, storage, streaming, and compression algorithms can handle various kinds of scene data in a unified way. Because points do not explicitly store topology information, no connectivity has to be updated during post-processing and editing. On the other hand, better image quality can be achieved with our view-dependent representation based on billboards. Being defined in image space, post-processing of the raw acquired

data can be done more effectively, exploiting the regular structure of the sampling. Editing, however, would break the regularity. In that case, the billboards can be easily converted to point clouds.

To obtain the input data, we developed a mobile, scalable acquisition system consisting of multiple so-called 3D video bricks. They capture texture and depth images of dynamic scenes synchronously from multiple viewpoints. High-quality depth reconstruction is achieved using a combination of state-of-the-art stereo vision and image processing algorithms. A three-dimensional description of the dynamic scene is obtained by merging all information gathered by the different bricks.

On the application side, current efforts mainly focus on playback from virtual viewpoints. Relatively little research has been devoted to 3D video post-processing. Having the three-dimensional scene information available, 3D video cannot only provide novel special effects but also greatly simplifies conventional video post-production. In principle, tasks like video cutout, scene compositing, or relighting become more simple end less time-consuming. We developed a new concept for intuitive manipulation of the four-dimensional data stream, including a set of basic editing operators exploiting spatial and temporal coherence. Integrated in our 3D video editing framework, they allow for producing a variety of special effects which would be difficult or impossible to achieve with conventional 2D video. We envision 3D video authoring as convenient as in Apple's® iMovie® 2D video editing software, allowing for similar editing as carried out in 3D content creation and modeling tools like Autodesk® Maya® or Pointshop 3D [Zwicker et al., 2002a].

1.2 Contributions

In this thesis, we make the following contributions:

Modular 3D video acquisition system. We introduce portable 3D video bricks which act as low-cost z-cameras and allow for simultaneous texture and depth map acquisition using spacetime stereo on structured light. To improve reconstruction accuracy at depth discontinuities, we extend window-based spacetime stereo matching with an efficient sliding window algorithm. Full 360 degree views of a scene can be recorded with only a sparse arrangement of bricks. Together, they build a modular, scalable 3D video acquisition studio.

Probabilistic, point-based 3D video representation. We propose a view-independent representation of geometry information acquired by the 3D video bricks. Its point-based nature supports application of a variety of processing algorithms, e.g. for outlier removal, editing, or compression. A statistical model of the depth reconstruction process together with a probabilistic rendering technique

based on view-dependent EWA volume splatting enables image generation from novel viewpoints with appealing quality.

Image-based 3D video representation. As an alternative, we present a view-dependent 3D video representation based on displacement-mapped billboards. Being an image space description with a well-defined error model, it allows for efficient application of signal processing tools for post-processing the raw acquired geometry. In particular, we apply a spatio-temporal bilateral filter that successfully removes noise and errors, and results in a close alignment of overlapping surface patches. We propose a rendering technique that generates view-dependent, consistent geometry from multiple, overlapping input scans. Together with view-dependent blending of colors, it generates high-quality images of the scene from novel viewpoints. If required, the image space representation can be easily converted to our point-based representation to perform further editing operations.

Volumetric 3D video representation. We introduce the point-sampled video hypervolume as a unified representation of four-dimensional spacetime. It allows for space- and time-coherent application of processing operators and builds the basis for our 3D video editing framework. A hyperslicing operator selects three-dimensional subspaces of the four-dimensional volume and allows for visualizing and editing both spatial and temporal aspects of the 3D video.

Progressive compression scheme for point-sampled models. We present a generic framework for compression of point-sampled models. By exploiting local coherence in the point cloud, various point attributes can be stored efficiently in a progressive data stream containing a compact, multiresolution representation of the scene. We provide efficient compression operators for point positions, surface normals, and colors. Together, they allow for efficient, unified storage and transmission of three-dimensional natural scenes including geometry and appearance.

Out-of-core data management. We propose an out-of-core data structure for storage of large 3D videos in the hypervolume representation. It allows for efficient queries and for dynamic modifications of the point data with a limited amount of main memory. Combined with a multiresolution representation, editing of large data sets is possible at interactive rates.

3D video editing system. We define a set of basic spatio-temporal operators for editing 3D video footage. In particular, we present a semi-automatic segmentation algorithm as a tool for making complex selections of surfaces or objects of the 3D video. Similar to 2D image or video cutout tools, it is based on a global graph cut optimization process. By using not only colors but also the available geometry information, we are able to greatly improve the robustness of the optimization and, thus, require much less user interaction. By integrating all operators in a

nonlinear 3D video editing system, we demonstrate the suitability of 3D video for video post-production and generation of novel visual effects.

1.3 Publications

This thesis is based on the following publications that have been released during four years of research. For a complete list of papers please refer the author's resume in the appendix.

WASCHBÜSCH, M., GROSS, M., EBERHARD, F., LAMBORAY, E., and WÜRMLIN, S., 2004. Progressive compression of point-sampled models. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 95–102.

The paper presents an algorithm for progressive compression of static point-sampled models, including hierarchical prediction schemes for geometry, surface normals, and colors.

WASCHBÜSCH, M., WÜRMLIN, S., COTTING, D., SADLO, F., and GROSS, M., 2005. Scalable 3D video of dynamic scenes. *Proc. of Pacific Graphics '05*, pp. 629–638.

This work introduces the concept of 3D video acquisition bricks using active stereo for depth reconstruction. Furthermore it presents a three-dimensional probabilistic point-sampled scene model together with probabilistic rendering methods.

WASCHBÜSCH, M., WÜRMLIN, S., and GROSS, M., 2006. Interactive 3D video editing. *Proc. of Pacific Graphics '06*, pp. 631–641.

This paper presents our interactive 3D video editing system, including a novel four-dimensional video hypervolume representation and a semi-automatic 4D surface selection method based on graph cuts.

WASCHBÜSCH, M., WÜRMLIN, S., COTTING, D., and GROSS, M., 2007a. Point-sampled 3D video of real-world scenes. *Image Communication*, 22(2), pp. 203–216.

An extended version of [Waschbüsch et al., 2005], including a novel discontinuity-preserving stereo matching algorithm.

WASCHBÜSCH, M., WÜRMLIN, S., and GROSS, M., 2007b. 3D Video Billboard Clouds. *Proc. of Eurographics '07*, pp. 561–569.

This paper presents an image-space 3D video representation using displacement-mapped billboards. A spatio-temporal disparity filter and a rendering algorithm based on view-dependent geometry achieve high-quality output images.

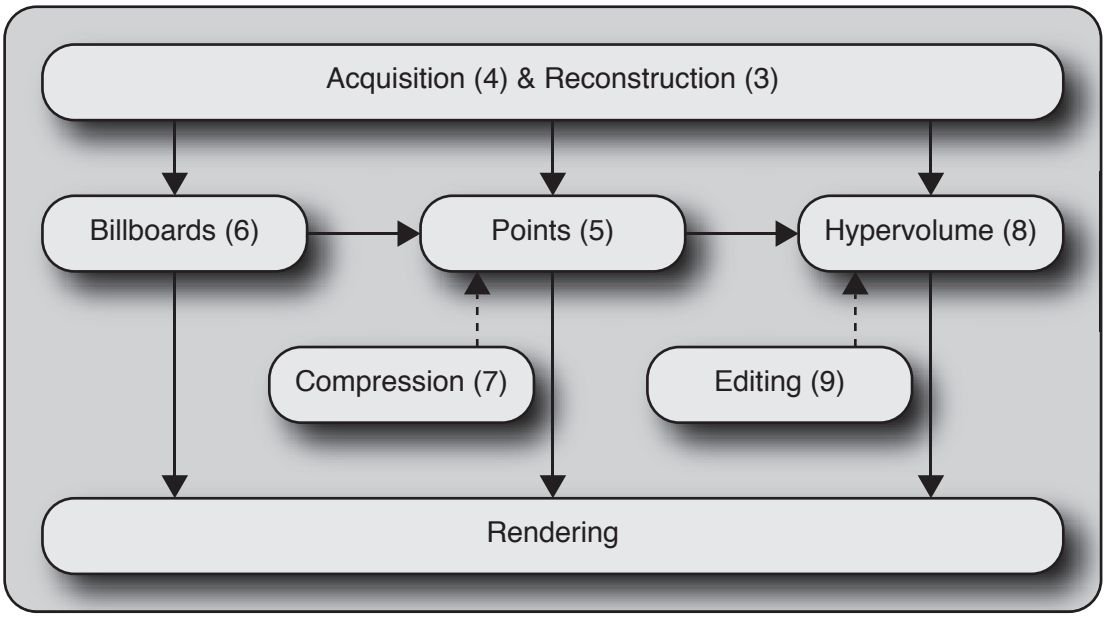


Figure 1.1: The 3D video pipeline as an underlying organization structure of this thesis. The numbers in brackets denote the corresponding chapters.

1.4 Organization

The remainder of this thesis is organized as follows: After surveying related work in chapter 2, the different aspects of our 3D video pipeline are described according to the organization illustrated in figure 1.1. Chapter 3 introduces the fundamentals of acquiring three-dimensional geometry of real-world scenes, which serves as the basic input data for 3D video. After surveying various computer vision algorithms for that purpose, it presents our implementation of stereo matching on structured light, which achieves the most robust results. Based on that method, chapter 4 presents our hardware setup for 3D video acquisition built on the modular concept of 3D video bricks. They serve as inexpensive capturing devices for recording geometry and color information of a moving scene synchronously from multiple viewpoints. To create a 3D video that can be played from arbitrary virtual viewpoints, the gained input data has to be transformed into a representation suitable for high-quality rendering. We present two basic representations together with appropriate image generation algorithms: A view-independent, point-based representation in chapter 5 and a view-dependent, image-based representation in chapter 6. While the second one achieves superior image quality, the point-based representation provides most flexibility for applications beyond free-viewpoint playback, such as editing of 3D video. Because the image-based representation can be easily converted to points, it can still be used as an intermediate model to perform high-quality post-processing of the raw geometry data. The power of points for 3D video transmission is presented in chapter 7. It discusses an

algorithm for compression of arbitrary attributes of point-sampled models such as geometry or appearance in a unified way, resulting in a compact, progressive data stream. Although it has been implemented for static models only, it can in principle be easily extended to dynamic data. 3D video editing is based on the point-sampled video hypervolume representation of chapter 8. By unification of space and time, it permits application of video processing operators that maintain both spatial and temporal coherence. Quick random access to large 3D video data sets is achieved by a multiresolution out-of-core spatial indexing structure. Being a dynamic representation, editing tools can be applied interactively. For that purpose, we integrated a fundamental set of manipulation operators in an interactive, nonlinear 3D video editing system presented in chapter 9. An included semi-automatic graph cut segmentation algorithm allows for quick, time-coherent selections of surfaces or objects in the video. Finally we conclude in chapter 10 with a validation of our claimed contributions.

Chapter 2

Related Work

This chapter presents the state of the art of research covered in this thesis. After summarizing publications in the topic of 3D video, it focuses on two main aspects of our work: point-based scene representations and video editing. The third important topic—reconstruction of depth maps—is covered in detail in section 3.1, as it directly influences the design of our 3D video acquisition system.

2.1 3D Video

In 3D video, multi-view video streams are used to re-render a time-varying scene from arbitrary viewpoints. There is a continuum of representations and algorithms suited for different acquisition setups and applications. Each approach provides a specific trade-off between image quality, number of acquisition devices, algorithmic complexity, and amount of generated data.

Purely image-based representations such as light fields [Levoy and Hanrahan, 1996] generally achieve very high image quality because they do not need any geometry at all. However, they require very densely-sampled input views [Chai et al., 2000; Zhang and Chen, 2001] and, thus, a large amount of densely-placed acquisition cameras. Examples include dynamic light field cameras [Yang et al., 2002; Wilburn et al., 2005] with camera baselines of a couple of centimeters. As a consequence, the amount of generated data is high, depending on the acquired viewing range. Thus, light field representations are most suitable for applications like 3D-TV [Matusik and Pfister, 2004] when only a very limited virtual viewing range has to be provided at maximum quality.

Camera configuration constraints can be relaxed by adding more and more geometry to image-based systems, as demonstrated by lumigraphs [Gortler et al., 1996; Buehler et al., 2001]. Depth image-based representations [Shade et al., 1998; Bayakovski et al., 2002] exploit depth information per pixel which can be computed either by shape-from-silhouettes [Matusik et al., 2000] or by stereo algorithms [Zitnick et al., 2004]. While the former is only applicable to stand-alone

objects, the latter still requires small baselines and does not integrate multiple cameras easily. Hence, both do not permit practical camera configurations for large viewing ranges.

Explicit 3D geometry allows for view-independent representations. Examples include implicit surfaces [Cockshott et al., 2003], triangular meshes [Kanade et al., 1997; Matusik et al., 2001] and 3D point samples [Würmlin et al., 2002]. However, most approaches are restricted to stand-alone objects due to the applied reconstruction algorithms.

Voxel-based methods [Vedula et al., 2002] are also suited. They represent the scene as a set of non-empty cells of a regular grid which can be reconstructed from the input camera images using voxel coloring [Seitz and Dyer, 1997; Culbertson et al., 1999] or space carving [Kutulakos and Seitz, 1999] algorithms. However, the regular sampling of the grid limits the resolution of the 3D video, preventing the user from moving the virtual camera to closeup views.

The use of geometric templates together with video textures alleviates the problem of robust geometry acquisition. By fitting prior triangular meshes into the images of the acquisition cameras [Carranza et al., 2003], high-quality output images can be achieved for almost all camera configurations. The approach can even be used for relighting the scene [Theobalt et al., 2007]. However, the template has to be known beforehand. As a consequence, complex scenes are difficult to model.

Besides virtual playback of prerecorded scenes, researchers also target online applications for virtual telepresence. Gross et al. [2003] use a 3D video system based on a point sample representation [Würmlin et al., 2004] for their telecollaboration system blue-c. By using the silhouette-based reconstruction algorithm by Matusik et al. [2000], they are able to achieve a full 360 degree viewing range in real-time. However, they are again limited to reconstruct foreground objects only. Mulligan and Daniilidis [2000] also target telepresence. They compute geometric models with multi-camera stereo and transmit texture and depth over a network. If only a constrained virtual viewing range is required, the hardware setup and algorithmic complexity can be dramatically reduced by using billboards instead of reconstructed geometry [Rhee et al., 2007].

2.2 Point-Sampled Geometry

Over the last years, the point sample gained more and more popularity as a fundamental primitive for representing and rendering three-dimensional models. Especially for the highly complex geometry of scanned real-world objects, points are often superior to triangle meshes, because no time-consuming computation of a consistent triangulation is needed. In the following, we summarize the research on the topics rendering, geometry processing, and compression, that are relevant for our work. For an overview over the whole field, please refer to Gross [2006]

or Gross and Pfister [2007].

2.2.1 Rendering

One of the first approaches using points for modeling three-dimensional objects was proposed by Levoy and Whitted [1985]. There, the authors concentrated on the special case of continuous, differentiable surfaces. Extensions to general surface geometry were proposed by Grossman and Dally [1998], and Pfister et al. [2000], using adequately sampled hierarchical data structures and forward warping for storage and rendering. Most current point rendering algorithms are based on the work of Zwicker et al. [2001a, 2002b]. Inspired by the EWA framework by Heckbert [1989], the authors formulate a forward splatting approach including an efficient screen space antialiasing filter for rendering high-quality images. The algorithm can be easily extended to volumetric rendering [Zwicker et al., 2001b].

With the increasing power of graphics hardware, GPU adaptations of point splatting are continuously developed and extended. Starting with the implementations by Ren et al. [2002] and Botsch et al. [2002], extensions include improved splatting kernels [Zwicker et al., 2004], clipped splats for sharp edges [Adams and Dutré, 2003, 2004; Wicke et al., 2004], per-pixel illumination [Botsch et al., 2004], and deferred shading [Botsch et al., 2005].

Besides splatting, alternative image generation methods such as adaptive resampling [Alexa et al., 2001] and ray tracing [Adamson and Alexa, 2003] have been proposed, too.

2.2.2 Geometry Processing

To post-process our 3D video data for outlier removal, smoothing, simplification, scan merging, and editing, we can use a great range of available processing algorithms for point-sampled geometry. Many of them are publicly available in the integrated editing application Pointshop 3D [Zwicker et al., 2002a].

Pauly and Gross [2001] propose a method for processing point-sampled surfaces in the Fourier domain. They decompose the surface into a set of patches and express their geometry at height fields on tangential planes. Blending at patch boundaries ensures global consistency.

Simplification of point clouds is introduced by Pauly et al. [2002]. Three different methods are proposed, using clustering, iterative decimation, and particle simulation. Measurements for local surface variation and quadric error permit exact control over the degree of simplification.

A practical tool set for post-processing the raw point clouds generated by range scanners is presented by Weyrich et al. [2004]. It includes various tools for outlier removal, smoothing, and hole filling. Merging of point sets from multiple scans can be done as proposed by Sadlo et al. [2005] who use a patch selection algorithm based on resolution and confidence measures.

Finally, surface editing operations such as deformations and Boolean operations are introduced by Pauly et al. [2003]. They are using a hybrid geometry description based on point samples and implicit moving least-squares surfaces [Levin, 2003]. Dynamic resampling retains surface quality even for large-scale modifications.

2.2.3 Compression

Because point sets do not contain any explicit connectivity information, they can be efficiently used for progressive streaming and compression.

In QSplat [Rusinkiewicz and Levoy, 2000], which is a multiresolution rendering system based on a hierarchical bounding sphere data structure and splat rendering, each node of the data structure is quantized to 48 bits, including color and surface normal data. By integrating view-dependent, progressive decompression [Rusinkiewicz and Levoy, 2001], an interactive viewer can request only the visible parts of the data stream over a network.

Botsch et al. [2002] use an octree data structure for storing point-sampled geometry as a binary vector representing the octree cell occupation. Surface normals are encoded using quantization schemes based on sphere subdivision. The authors show that the geometry of typical data sets can be encoded with five to ten bits per point. Compression performance can be further improved by predicting for each parent cell the occupancy of its child cells [Huang et al., 2006; Schnabel and Klein, 2006] using local surface statistics.

Instead of an octree, Bordignon et al. [2006] use a BSP-tree. By aligning its splitting planes along the principal components of local point neighborhoods, its structure better adapts to the object geometry. They are able to achieve a compression performance of 14 bits per point.

A similar performance is achieved by Fleishman et al. [2003] who propose a progressive point set coder based on the projection of points onto local polynomial surface approximations. However, their resampling method based on the moving least-squares (MLS) projection operator [Levin, 2003] tends to smooth out sharp features. Furthermore, the decompression is very time-consuming because the MLS projection also has to be applied during decoding.

For dynamic point clouds, temporal coherence can be exploited by encoding differential point updates [Würmlin et al., 2003]. By adding an acknowledgment protocol, such a scheme has been used in the blue-c project for real-time, fault-tolerant transmission of point-based 3D video over a network [Lamboray et al., 2004a]. Lamboray et al. [2004b] use multi-channel images from the input camera views to store the point information. Then, conventional image and video codecs can be used for compression [Würmlin et al., 2005].

2.3 Video Editing

To the best of our knowledge, no research has been done in the field of 3D video editing, yet. However, our work is based on several publications related to 2D video editing which we summarize here.

Our spatio-temporal data representation is inspired by the video cube that has been previously introduced by Fels and Mase [1999]. Intuitively, by stacking all successive two-dimensional frames on top of each other, one obtains a three-dimensional volumetric representation of the complete video stream. Such a structure allows for editing the video in both the spatial and temporal domain in a unified manner. For example Klein et al. [2002] use video cubes to produce a variety of visual effects. Bennett and McMillan [2003] provide a unified framework on the volumetric representation to elegantly formulate different editing tasks like object removal, camera stabilization and reformatting.

A fundamental task in editing of traditional 2D video is the computation of accurate alpha mattes of foreground objects. That problem has been initially covered for still images. Chuang et al. [2001] are using a Bayesian formulation based on color models of foreground and background. It has later been extended by Shum et al. [2004] who added a weak constraint ensuring boundary coherence. Similar results have been achieved by Sun et al. [2004] by solving a Poisson equation based on the image gradient. For video, Chuang et al. [2004] apply Bayesian matting on every frame while enforcing time coherence using optical flow [Barron et al., 1994]. As input, the user has to provide a coarse initial segmentation—a so-called graymap—for a single frame.

Video cutout techniques [Li et al., 2005; Wang et al., 2005] further simplify user interaction by automatically computing an initial binary segmentation based on some user-defined object markings using graph cuts [Boykov et al., 1999; Boykov and Jolly, 2001]. They extend two-dimensional image cutout techniques [Li et al., 2004; Rother et al., 2004] into the temporal domain by solving a three-dimensional optimization problem on the video cube. Wang et al. [2005] integrate the video cube into the user interface, allowing for conveniently marking of moving objects over time.

To completely avoid user input, fully automatic approaches have been proposed. They segment the foreground object based on its depth, using specialized video acquisition equipment. Gvili et al. [2003] use an additional time-of-flight camera for depth extraction. The approach by McGuire et al. [2005] uses multiple video cameras with different foal lengths and, thus, has the advantage of also working in large-scale outdoor environments.

Chapter 3

Reconstructing 3D Scenes from 2D Images

3D video captures two fundamental kinds of information about real-world scenes which dynamically changes over time: three-dimensional geometry and appearance of surface materials.

For a fixed viewpoint, appearance can be acquired completely using conventional video cameras. This may be simply extended to 3D video by recording all possible viewpoints at the same time, resulting in a so-called light field representation [Levoy and Hanrahan, 1996]. Although this is the most generic approach, yielding high-quality output images for most scenes, it has several practical limitations. The desired viewing range has to be densely sampled using a lot of cameras. As a result, the amount of produced image data is huge and difficult to handle. Moreover, no inherent knowledge about the scene besides images is captured, making post-processing difficult and constraining the application to virtual replay and synthetic aperture and lens effects only.

Instead, we opt for a different approach including explicit information about scene geometry. By acquiring not only images but also depth maps, novel views can be interpolated from a sparse set of input viewpoints. A consistent three-dimensional model can be built easily by backprojecting pixels from all views into three-dimensional space using their individual depth values. Such a scene model allows for applications beyond virtual camera control. As an example, we introduce a system for 3D video editing in chapter 9.

Our view interpolation requires one fundamental assumption: Each surface point has to have an identical appearance from all possible viewing directions. As a consequence, novel views of that point can be computed easily as soon as its position in three-space and its appearance from one viewing direction is known. Moreover, its appearance under the present lighting conditions can be captured from a single viewpoint by using a color camera. Thus, we impose the following requirements on the acquired scene and our applications:

- We are only interested in surfaces of objects, not in their interior structure.
- All surface materials have to be Lambertian, i.e. they scatter all incoming light evenly in all possible directions. There are no effects such as specular reflections or transparencies.
- There is no visible medium like smoke or fog between surfaces and acquisition cameras.

For such cases, reconstruction of depth maps is a heavily researched topic in the field of computer vision. A variety of methods exist, differing in the required hardware, reconstruction quality, and additional scene constraints. After surveying those approaches and discussing the needed theoretical concepts, we present our chosen method of depth from stereo on structured light. Especially, we discuss our extension for increasing fidelity at depth discontinuities. As a result, we are able to obtain decent-quality depth maps in a robust way.

3.1 Survey of Depth Acquisition Methods

All depth map acquisition methods we examine are based on some imaging device acquiring a two-dimensional view of light reflected or emitted by the scene. However, such information is generally not enough for reconstructing three-dimensional geometry. Thus, additional techniques are used for obtaining more data. These form the main differences between the examined methods and determine the complexity of the performed computations. Most approaches are based on triangulation. They can be separated in active methods which illuminate the scene with specialized light sources and passive methods which do not influence the environment. Different approaches are used by time-of-flight and shape-from-silhouette methods. The first performs temporal measurements instead of triangulation, the latter reconstructs an approximate geometry based on the silhouettes of an object seen from multiple viewpoints.

We define a set of requirements an optimal reconstruction algorithms should fulfill for recording 3D video of dynamic scenes:

- **Quality.** The acquired depth values should be of high quality, accurately representing the real scene without too much noise and outliers.
- **Density.** The depth maps should be dense, i.e. they should contain a valid depth value at almost every image pixel.
- **Dynamics.** The algorithm should be able to robustly capture moving scenes.

- **Scalability.** The system using the depth reconstruction algorithm should be scalable to an arbitrary number of acquisition viewpoints. As a consequence, multiple depth acquisition devices should not interfere with each other.

- **Generality.** The depth reconstruction method should be applicable to a large variety of scenes.

- **Practicability.** The acquisition system should comprise an acceptable amount of hardware that can be set up easily.

Existing methods are surveyed under these constraints.

3.1.1 Stereo Vision

Stereo vision algorithms are inspired by human depth perception. By finding pixel correspondences in two camera images captured from different but nearby viewpoints, the depth of the projected object point can be calculated via triangulation. The pixel search is essentially a minimization problem optimizing the matches for all pixels. A survey of different methods and their implementations can be found in the paper by Scharstein and Szeliski [2002]. They can be divided into two fundamentally different categories. Local methods do a matching search for each pixel individually, whereas global methods try to solve the problem for the whole image at once. Moreover, one could distinguish sparse methods providing depth values only at feature points from dense methods providing depth values at every pixel. As we require dense depth maps, we concentrate on the second category.

Local matching algorithms are based on correlation of small pixel neighborhoods within a usually rectangular window in both images. Due to their local view, they have difficulties at ambiguities occurring in regions with uniform color or repetitive texture. Hence, they require input images with highly varying textures. A second issue are depth discontinuities which cannot be modeled if they occur inside the correlation window. Thus, the core algorithms are not capable of accurately capturing object silhouettes without further extensions. On the other hand, they can easily find matches with subpixel accuracy, which is important for obtaining a high depth resolution.

Global stereo algorithms based on Markov random fields [Boykov et al., 1999; Sun et al., 2003] try to find all pixel matches at the same time with respect to some overall consistency constraints. By minimizing an energy composed of a correlation term and an edge-preserving smoothness term, they are able to handle both ambiguities and depth discontinuities. However, obtaining accurate depth values in texture-less regions still remains difficult because they are just interpolated by the smoothness function. In texture-rich regions, global algorithms are in general not better than local methods. Depth resolution can be even worse because many

global algorithms such as the one by Boykov et al. [1999] cannot easily handle subpixel accuracy.

More recently, segmentation-based stereo algorithms have been developed. Correlation of whole color segments [Zitnick et al., 2004] performs much better at depth discontinuities than pixel-based matching and is able to accurately reconstruct sharp boundaries. Global solvers [Hong and Chen, 2004] again produce the best quality. However, those methods require a color segmentation as input that reliably represents discontinuities and that is consistent in both camera images. This is an additional nontrivial computer vision problem.

Matching ambiguities in homogeneous regions can be avoided by producing artificial textures using structured light illumination [Kang et al., 1995]. Because the light patterns can be random, such an approach scales well with overlapping projections. Window-based algorithms show a great improvement with structured light. Their only remaining instability is in matching depth discontinuities. Global methods, on the other hand, showed to behave very unstable in our experiments. This is due to their build-in smoothness function which assumes correlation between depth and texture discontinuities, which is not true with the employed light patterns.

All stereo algorithms are in principle able to handle dynamic input data because they only require one image pair recorded at one point in time. To improve coherence, matching can be extended to the temporal domain by considering a whole stack of successive images at once [Davis et al., 2003; Zhang et al., 2003]. With local approaches, this usually improves stability at slow but not at fast motions, because the latter tend to create additional depth discontinuities in time.

Stereo vision can theoretically reconstruct arbitrary scenes as long as enough texture information is available. If the latter is generated by structured light projections, acquisition is naturally constrained to indoor scenes only. The acquisition setup consists of pairs of calibrated cameras and, optionally, uncalibrated structured light projectors, which have to be synchronized in the case of dynamic recording. The resulting matching quality largely depends on the number and complexity of occlusions which should be minimized. Besides changing the scene itself, this can be achieved by choosing small baselines between corresponding cameras of the stereo pairs. Between independent camera pairs, large baselines can be used to cover a large viewing range. Details can be found in chapter 4. On the other hand, specialized wide-baseline stereo vision methods exist [Pritchett and Zisserman, 1998], but they largely depend on distinct image features and, thus, rather belong to the category of sparse matching algorithms.

3.1.2 Structured Light

Structured light scanning systems [Salvi et al., 2004] are similar to stereo vision systems but replace one camera with a projector. Thus, they are naturally constrained to indoor scenes. Stereo matching is performed between camera and

projector pixels. In contrast to the aforementioned stereo vision on structured light, the projector has to be calibrated. The camera-projector baselines should be small in order to minimize occlusions. To find correct correlations, the algorithms have to uniquely identify light from distinct projector pixels in the camera images. Therefore, the projected patterns have to be well defined and should be more prominent than the scene textures. Compared to stereo vision, this improves the quality of correlation and generally yields superior results. On the other hand, it constrains the setup of multiple projectors and impairs scalability, as the pattern determinateness gets destroyed as soon as multiple projections overlap.

Structured light algorithms can be categorized into single-shot and multi-shot methods. Single-shot methods such as the one by Vuylsteke and Oosterlinck [1990] only need one image of the scene under one pattern illumination, making them suitable for capturing dynamic scenes. However, due to ambiguities in the pattern, they are only able to uniquely identify some specific projector pixels, yielding sparse depth maps only. Multi-shot methods [Inokuchi et al., 1984] resolve for those ambiguities by capturing multiple images of the scene under a sequence of different lighting conditions. They are able to reconstruct dense depth maps but cannot easily cope with motions. A special case of multi-shot approaches are laser range scanners which replace the pattern projection with a laser line that is constantly swept over the recorded object.

3.1.3 Time of Flight

Time-of-flight systems such as the one by Iddan and Yahav [2001] are based on a completely different approach. Instead of computing triangulations they measure the time light needs to travel from a calibrated source to an imaging sensor. This is achieved by measuring either directly the time of light pulses or phase shifts of modulated light. In the latter case, multiple devices can be combined without interference by using different modulation frequencies.

Today, integrated off-the-shelf products like the SR-3000 camera by MESA Imaging® or Canesta's™CanestaVision™camera become available. Their depth range is constrained to a couple of meters depending on the intensity of the emitted light, but they even work in moderately bright outdoor scenes. However, their depth images are still suffering from low resolution, noise, and outliers. In the future, they may provide a good alternative to our chosen approach.

3.1.4 Shape from Silhouettes

Shape-from-silhouette methods [Matusik et al., 2000, 2001] reconstruct an approximate geometry of distinct objects by backprojecting their silhouettes from multiple viewpoints into three-space and intersecting the resulting generalized cones. The acquisition setup consists of multiple calibrated cameras covering a convex viewing range. In order to be able to extract the silhouettes, only one or

	Stereo vision				Structured light		Time of flight	Shape from silh.
	Local		Global		Single-shot	Multi-shot		
	Passive	St. light	Passive	St. light				
Quality	−	+	○	−	+	+	○	○
Density	+	+	+	+	−	+	+	+
Scalability	+	+	+	+	−	−	○	+
Dynamics	+	+	+	+	+	−	+	+
Generality	+	○	+	○	○	○	○	−
Practicability	+	○	+	○	−	−	+	+

Table 3.1: Summary of the capabilities of various depth map acquisition methods (+ good, ○ average, − bad).

very few clearly separated objects but no complex scenes can be captured. They are usually placed in front of a background with a defined color. The reconstruction method is very robust but needs many cameras to create a good approximation to the real geometry. Even in the theoretical limit, when using an infinite amount of cameras covering all possible viewing directions, it cannot reconstruct the real geometry but only its visual hull which does not capture surface concavities.

3.1.5 Summary

Table 3.1 summarizes the capabilities of the discussed depth reconstruction methods. As we give the highest priority to reconstruction quality, local stereo matching supported by structured light projection fulfills our requirements at the best. Hence, this is the approach we have chosen for our 3D video acquisition system. A detailed explanation of the employed algorithms follows in the next sections.

3.2 Mathematical Concepts of Image Acquisition

All traditional image acquisition devices capture projections of light rays emanating from three-dimensional space onto a two-dimensional image plane. Methods reconstructing three-dimensional geometry from imagery perform the inverse projection based on the image data and known internal and external camera parameters. This section presents the mathematical foundations for computing those projections. The pinhole camera model provides a simple but powerful mathematical abstraction which can be applied to images of real-world cameras after compensating for effects of their optical systems. Epipolar geometry makes the transition from one to multiple cameras that acquire the same scene from different viewpoints by describing relations between the different images. For two cameras,

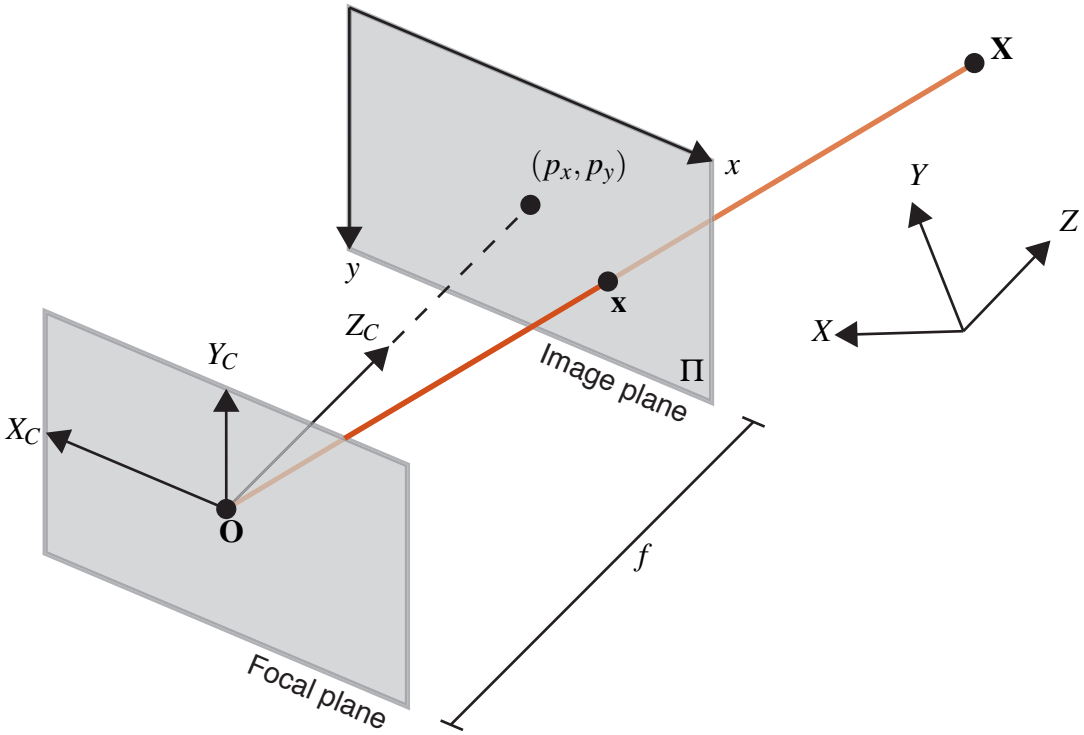


Figure 3.1: The pinhole camera model.

the epipolar geometry can be greatly simplified by post-processing their images with the presented rectification method.

3.2.1 The Pinhole Camera Model

The pinhole camera model [Faugeras, 1993], as illustrated in figure 3.1, describes how a point $\mathbf{X}$ in three-dimensional world coordinates (X,Y,Z) is projected on a two-dimensional image plane Π, yielding a pixel $\mathbf{x}$ with coordinates (x,y). It is an abstraction of real-world cameras, replacing the optical lens system by an infinitesimal small pinhole which is the center of projection $\mathbf{O}$ where all light rays pass through. $\mathbf{O}$ is lying on the focal plane which is parallel to the image plane. The distance f between both planes is called the focal length of the camera. In contrast to real cameras, we use the common approach of placing the image plane in front of the center of projection instead behind it. This is valid due to symmetry.

For convenience, points $\mathbf{X}$ are first transformed into 3D camera coordinates $\mathbf{X}_C = (X_C, Y_C, Z_C)$ before performing the projection. The origin of the camera coordinate system is placed at the center of projection $\mathbf{O}$, its X_C- and Y_C-axes span the focal plane. Transformation of world coordinates into camera coordinates can be described by a 3×3 rotation matrix R and a translation to the center of projection $\mathbf{O}$ as

$$\mathbf{X}_C = \mathrm{R} \cdot (\mathbf{X} - \mathbf{O}) \tag{3.1}$$

or by using homogeneous coordinates as

$$\tilde{\mathbf{X}}_C = \begin{pmatrix} \mathrm{R} & -\mathrm{R}\mathbf{O} \\ 0 & 1 \end{pmatrix} \cdot \tilde{\mathbf{X}}. \tag{3.2}$$

R and $\mathbf{O}$ are the so-called extrinsic parameters of the camera. They encode all external characteristics like camera position and orientation. For convenience, we mainly use the notation of (3.1).

The main step for projecting $\mathbf{X}_C$ into a pixel $\mathbf{x} = (x, y)$ on the image plane is the perspective division by its third coordinate Z_C scaled with the focal length f:

$$x = f \cdot \frac{X_C}{Z_C}, \tag{3.3}$$

$$y = f \cdot \frac{Y_C}{Z_C}. \tag{3.4}$$

It can be written in homogeneous matrix notation as

$$\tilde{\mathbf{x}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{X}_C. \tag{3.5}$$

However, in this example, image coordinates are measured in the same length units as camera coordinates. For image processing, it is practical to transform them into pixel coordinates by scaling the focal length f differently in the x- and y-directions, yielding separate coefficients f_x and f_y:

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{X}_C. \tag{3.6}$$

Having separate scaling factors for the two image coordinate axes allows for modeling of non-square pixels as they occur in some image sensors. So far, the origin of the pixel coordinate system is located at the orthogonal projection of the pinhole onto the image plane. To comply with standard pixel coordinate notations that define the origin at an image corner, we add two more shifting coefficients p_x and p_y to equation (3.6) that describe the so-called principal point of the camera:

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & 0 & p_y \\ 0 & f_x & p_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{X}_C. \tag{3.7}$$

Ideally, the principal point is located in the image center. Thus for an image of width w and height h its coordinates would be $p_x = w/2$ and $p_y = h/2$. However, lens systems of real-world cameras often introduce some small displacement there which is measured along with the other parameters by standard camera calibration tools.

The final matrix of equation (3.7) encodes all internal parameters of the camera responsible for image projection. It is called the intrinsic matrix P. Putting it all together, any point $\mathbf{X}$ in Cartesian world coordinates can be projected to its homogeneous pixel coordinates $\tilde{\mathbf{x}}$ via

$$\tilde{\mathbf{x}} = \mathrm{P} \cdot \mathrm{R} \cdot (\mathbf{X} - \mathbf{O}). \tag{3.8}$$

As a side effect, the resulting homogeneous coordinate z of a pixel $\tilde{\mathbf{x}} = z \cdot (x, y, 1)^{\mathrm{T}}$ contains the so-called pixel depth that is the distance of the corresponding 3D point to the focal plane, i.e. $z = Z_C$. The three orthogonal coordinates x, y, and z define the so-called ray space of the camera. If, for a specific pixel, z is known, the 3D point can be reconstructed by inverting equation (3.8):

$$\mathbf{X} = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot z \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} + \mathbf{O}. \tag{3.9}$$

3.2.2 Real-World Cameras

Optical systems of real-world cameras show some additional effects that are not covered by the pinhole model. They include depth-of-field effects caused by the spatial extent of the aperture, chromatic aberration due to varying light refraction depending on the wavelength, and lens distortion. Chromatic aberration and depth-of-field effects cannot be handled easily without prior knowledge of the scene geometry. They have to be avoided during acquisition by choosing high-quality lenses and a small aperture. Lens distortion can be compensated by applying two-dimensional post-processing to the acquired image. As result, we obtain an image looking similar to one that would have been captured using a corresponding pinhole camera.

We use the common Brown-Conrady model [Brown, 1966] describing radial and tangential distortion of camera lenses. Given a pixel at coordinates (x, y) in an ideal image, the model describes the distorted pixel coordinates (x_D, y_D) as

$$x_D = (1 + r_1\rho^2 + r_2\rho^4) \cdot x + 2t_1xy + t_2(\rho^2 + 2x^2), \tag{3.10}$$

$$y_D = (1 + r_1\rho^2 + r_2\rho^4) \cdot y + t_1(\rho^2 + 2y^2) + 2t_2xy, \tag{3.11}$$

where $\rho^2 = x^2 + y^2$. The radial distortion coefficients r_1 and r_2 as well as the tangential distortion coefficients t_1 and t_2 can be estimated using the camera calibration routines provided by the OpenCV library [OpenCV]. It also includes a procedure for undistorting camera images.

3.2.3 Epipolar Geometry

Having two cameras with image planes Π_L, Π_R and centers of projection $\mathbf{O}_L$, $\mathbf{O}_R$ acquiring the same scene, there is a linear relationship between coordinates of

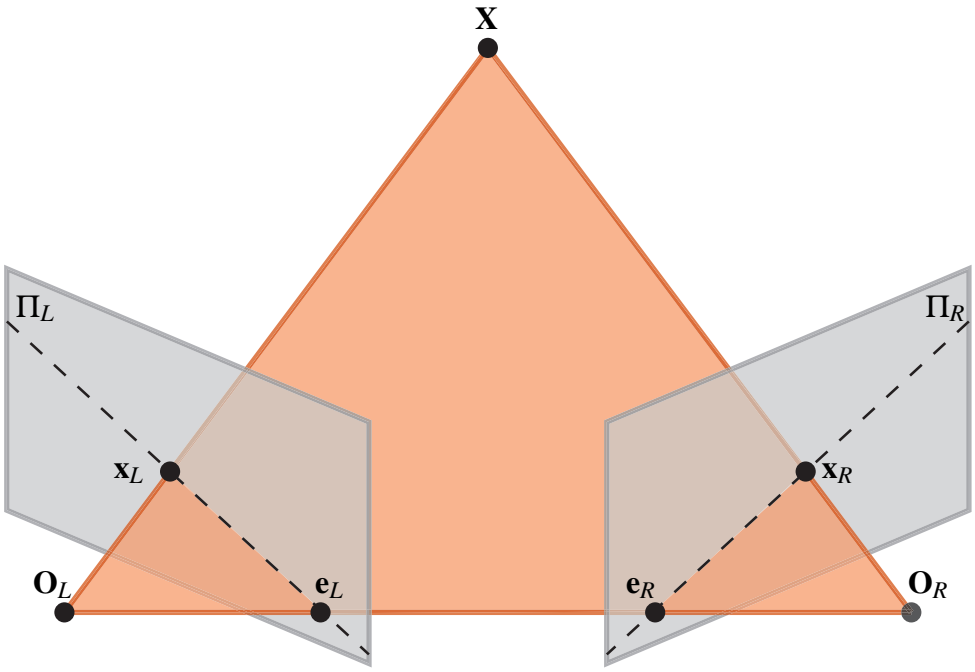


Figure 3.2: Epipolar geometry.

corresponding pixels that is described by the epipolar geometry. Consider an object point $\mathbf{X}$. Together with both centers of projections, the point forms a triangle $\Delta(\mathbf{X}, \mathbf{O}_L, \mathbf{O}_R)$ as shown in figure 3.2. Its edges $\overline{\mathbf{X}, \mathbf{O}_L}$ and $\overline{\mathbf{O}, \mathbf{O}_R}$ intersect the image planes at the respective projections $\mathbf{x}_L$ and $\mathbf{x}_R$ of $\mathbf{X}$. The projections of the epipolar plane into the camera images form so-called epipolar lines. For a varying $\mathbf{X}$ the epipolar plane rotates around the baseline of the camera pair building the epipolar pencil. All epipolar lines of the pencil intersect in one point $\mathbf{e}_L$ and $\mathbf{e}_R$, respectively, for each image, the so-called epipoles. The epipoles are located at the intersections of the camera baseline with the image planes.

Projections onto the two image planes corresponding to the same object point are always located on corresponding epipolar lines. This greatly reduces the search space in the stereo matching process. For a given pixel in the left image, the matching pixel in the right image can be found on the corresponding epipolar line. Intuitively, the search is performed on the projection of the viewing ray of the left image into the right image. Thus, stereo matching of two-dimensional images can be reduced to a one-dimensional search per pixel.

Mathematically, for each given camera setup, there exists a 3×3-matrix F such that for each object point $\mathbf{X}$ its two projections $\tilde{\mathbf{x}}_L$ and $\tilde{\mathbf{x}}_R$ fulfill the equation

$$\tilde{\mathbf{x}}_L^{\mathrm{T}} \cdot \mathrm{F} \cdot \tilde{\mathbf{x}}_R = 0. \tag{3.12}$$

F is the so-called fundamental matrix which even holds for uncalibrated cameras. It is of rank 2 having 6 degrees of freedom. In the calibrated case, F can be reduced

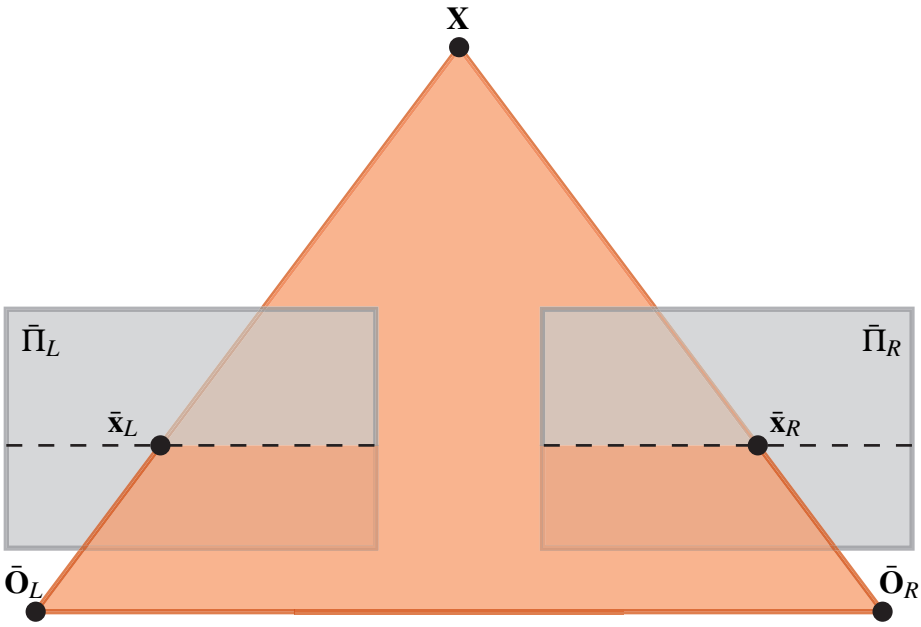


Figure 3.3: Epipolar geometry of a rectified camera pair.

to the special case of the so-called essential matrix describing the relationship of points in camera coordinates by encoding the difference of orientation and position of both cameras. Those matrices can be used to compute the epipolar lines. Details can be found in the book by Faugeras [1993].

For stereo matching, we use a different approach instead. By applying a rectification algorithm, the camera images are post-processed in a way that makes their epipolar geometry as simple as possible, alleviating the needed computations.

3.2.4 Image Rectification

Generally, epipolar lines form a pencil in each image. Image rectification warps and resamples both images in such a way that all epipolar lines are parallel and corresponding lines lie on the same scan line. In that case, the epipoles of both cameras are located at infinity and the image planes are coplanar and parallel to the baseline, as illustrated in figure 3.3. We use the rectification method by Fusiello et al. [2000]. Given the intrinsic and extrinsic matrices of the input cameras, it computes a new rectified camera configuration and a transformation that warps the images from the input cameras into the rectified cameras.

Consider a pair of pinhole cameras defined by their projection matrices $\mathrm{P}_L, \mathrm{P}_R$, rotation matrices $\mathrm{R}_L, \mathrm{R}_R$ and centers of projection $\mathbf{O}_L, \mathbf{O}_R$. They can be transformed into a rectified pair as follows:

First, the rectified cameras have to have the same intrinsic matrices $\bar{\mathrm{P}}_L = \bar{\mathrm{P}}_R$.

They can be defined arbitrarily, e.g.

$$\bar{\mathrm{P}}_L = \bar{\mathrm{P}}_R = \frac{1}{2} \cdot (\mathrm{P}_L + \mathrm{P}_R). \tag{3.13}$$

Second, their image planes should have the same orientation, parallel to their baseline. This is achieved by building a new orthonormal camera coordinate system defined by the three vectors

$$\mathbf{V}_1 = \frac{\mathbf{O}_L - \mathbf{O}_R}{||\mathbf{O}_L - \mathbf{O}_R||}, \tag{3.14}$$

$$\mathbf{V}_2 = \mathbf{V}_1 \times \mathbf{W}, \tag{3.15}$$

and

$$\mathbf{V}_3 = \mathbf{V}_1 \times \mathbf{V}_2, \tag{3.16}$$

where $\mathbf{W}$ is an arbitrary vector and $\times$ denotes the cross product. A new, common rotation matrix can then be built as

$$\bar{\mathrm{R}}_L = \bar{\mathrm{R}}_R = \begin{pmatrix} \mathbf{V}_1^{\mathrm{T}} \\ \mathbf{V}_2^{\mathrm{T}} \\ \mathbf{V}_3^{\mathrm{T}} \end{pmatrix}. \tag{3.17}$$

Finally, the center of projections remain unchanged, i.e.

$$\bar{\mathbf{O}}_L = \mathbf{O}_L \tag{3.18}$$

and

$$\bar{\mathbf{O}}_R = \mathbf{O}_R. \tag{3.19}$$

Keeping the camera positions permits the computation of rectified images using a 2D linear warping function. Warping matrices can be computed as

$$\mathrm{T}_L = \bar{\mathrm{P}}_L \bar{\mathrm{R}_L} \mathrm{R}_L^{-1} \mathrm{P}_L^{-1} \tag{3.20}$$

and

$$\mathrm{T}_R = \bar{\mathrm{P}}_R \bar{\mathrm{R}_R} \mathrm{R}_R^{-1} \mathrm{P}_R^{-1}. \tag{3.21}$$

Figure 3.4 shows the original camera images and the resulting rectified images with augmented epipolar lines.

From now on, if not mentioned otherwise, all matrices and images from stereo camera pairs are assumed to be rectified, without using the bar notation.

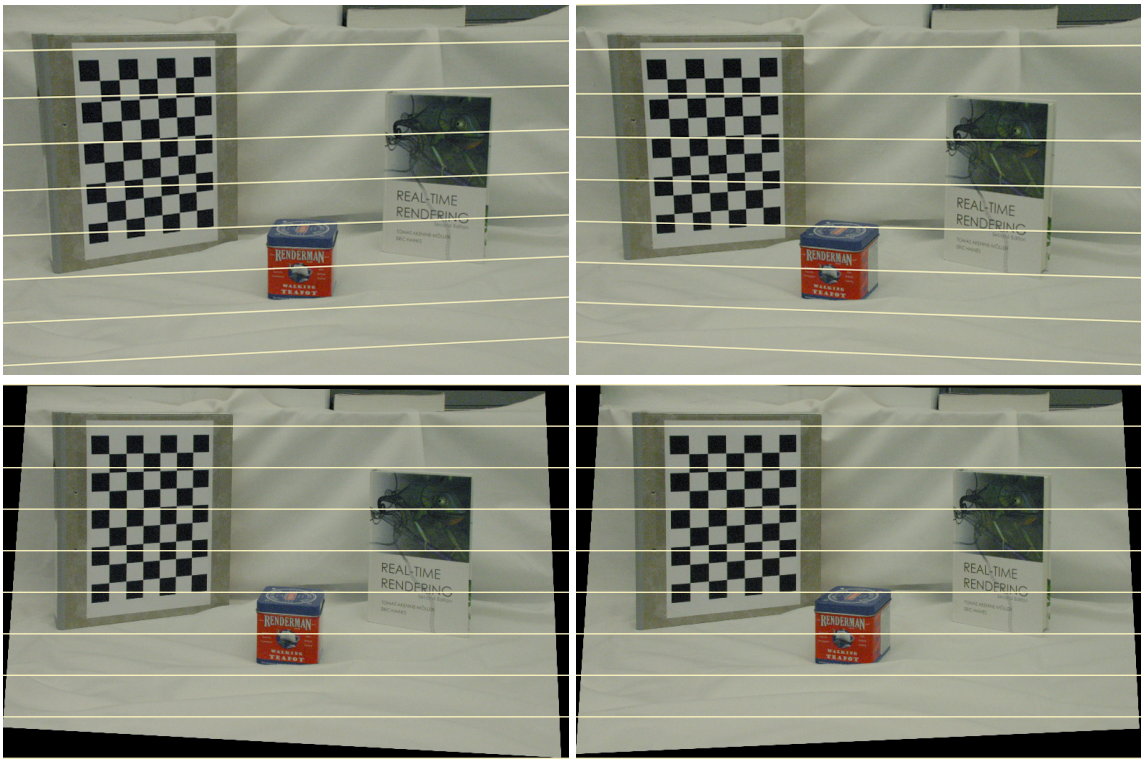


Figure 3.4: Stereo image pair with augmented epipolar lines. Top: original images. Bottom: rectified images.

3.2.5 Stereo Triangulation

If projections $\mathbf{x}_L$, $\mathbf{x}_R$ of an object point $\mathbf{X}$ onto two different camera planes Π_L, Π_R are known, one can reconstruct the coordinates of $\mathbf{X}$ via triangulation. Depth from stereo algorithms typically first compute pixel disparities, i.e. the coordinate deviations of corresponding pixels in two images. They can be directly transformed into depth maps that store for each image pixel the distance from its corresponding object point to the focal plane. Finally, three-dimensional geometry can be computed by backprojecting the pixels using the inverse camera projection matrix.

We only consider the case for two rectified camera images where corresponding pixels only differ in their x-coordinate. This reduces the problem to one dimension as depicted in figure 3.5.

The pixel disparity d is defined as the deviation of the pixel coordinates, measured relative to the principal points of the cameras:

$$d = (x_R - p_{x,R}) - (x_L - p_{x,L}). \tag{3.22}$$

To compute the pixel depth, note that the triangles $\Delta(\mathbf{X}, \mathbf{O}_L, \mathbf{O}_R)$ and $\Delta(\mathbf{X}, x_L, x_R)$

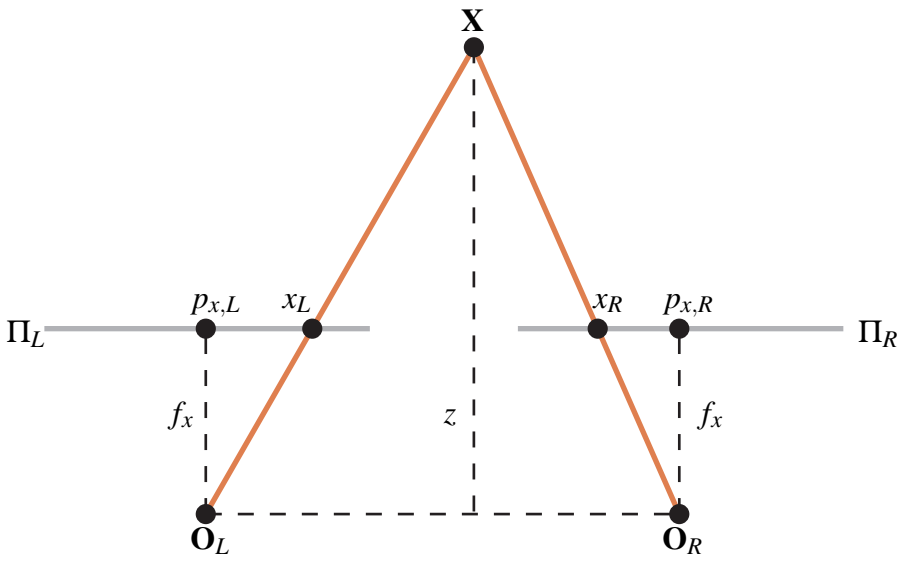


Figure 3.5: Stereo triangulation.

in figure 3.5 are similar and it holds that

$$\frac{||\mathbf{O}_R - \mathbf{O}_L||}{z} = \frac{||\mathbf{O}_R - \mathbf{O}_L|| + (x_R - p_{x,R}) - (x_L - p_{x,L})}{z - f_x}. \tag{3.23}$$

Simple calculations and substitution of d yields

$$z = -||\mathbf{O}_R - \mathbf{O}_L|| \cdot \frac{f_x}{d}. \tag{3.24}$$

With the gained pixel depths, the object point can be reconstructed from e.g. the left camera by backprojection:

$$\mathbf{X} = \mathrm{R}_L^{-1} \cdot \mathrm{P}_L^{-1} \cdot z \begin{pmatrix} x_L \\ y \\ 1 \end{pmatrix} + \mathbf{O}_L. \tag{3.25}$$

3.3 Stereo Matching

Stereo reconstruction is a one-dimensional search for corresponding pixels in two images along corresponding epipolar lines. Hence, for two rectified grayscale or color images $\mathcal{C}_L$ and $\mathcal{C}_R$ one has to find for every pixel (x,y) its disparity d such that $\mathcal{C}_L(x,y) = \mathcal{C}_R(x+d,y)$. This can be expressed as a minimization of an energy [Zhang et al., 2003]

$$E_M(d) = e_M(\mathcal{C}_L(x,y), \mathcal{C}_R(x+d,y)), \tag{3.26}$$

where e_M is an error function describing the difference between two pixels, for example the absolute difference metric $e_M(a,b) = |a-b|$. In our implementation

we use the error metric of Birchfield and Tomasi [1998]. It employs absolute differences on intensity functions that are linearly interpolated from small pixel neighborhoods and is therefore independent from image sampling.

Because all real images are augmented by acquisition noise, the colors of two corresponding pixels will never be the same and the above pixel-wise minimization will not succeed. This problem is solved by minimizing over a small rectangular window $\mathcal{W}(x_0,y_0) = \{x_0-\Delta x,\ldots,x_0+\Delta x\}\times\{y_0-\Delta y,\ldots,y_0+\Delta y\}$ of size $2\Delta x+1\times 2\Delta y+1$ around one pixel of interest (x_0,y_0):

$$E_M(d) = \sum_{(x,y)\in\mathcal{W}(x_0,y_0)} e(\mathcal{C}_L(x,y),\mathcal{C}_R(x+d,y)). \tag{3.27}$$

However, this assumes a constant disparity over the whole correlation window which is only correct for planar surfaces parallel to the image plane. If this is not the case, the computed disparities may deviate a few pixels from the ground truth. To obtain more accurate results, Zhang et al. [2003] extend the minimization by also searching for the gradient (d_x,d_y) of the disparity:

$$E_M(d,d_x,d_y) = \sum_{(x,y)\in\mathcal{W}(x_0,y_0)} e(\mathcal{C}_L(x,y),\mathcal{C}_R(x+d',y)), \tag{3.28}$$

where

$$d' = d+d_x(x-x_0)+d_y(y-y_0). \tag{3.29}$$

This corresponds to linearly changing disparities in the matching window which is exact for all planar surfaces and a good local approximation of the scene geometry.

If not only still images but videos are used as input, temporal coherence can be improved by extending the matching window over multiple successive frames, yielding a so-called spacetime stereo algorithm [Davis et al., 2003; Zhang et al., 2003]. Furthermore, it can be used to reconstruct static scenes under varying illumination conditions, increasing the robustness of the matching. In both cases, equation (3.28) is extended to

$$E_M(d,d_x,d_y,d_t) = \sum_{(x,y,t)\in\mathcal{W}(x_0,y_0,t_0)} e(\mathcal{C}_L(x,y,t),\mathcal{C}_R(x+d',y,t)), \tag{3.30}$$

with

$$d' = d+d_x(x-x_0)+d_y(y-y_0)+d_t(t-t_0). \tag{3.31}$$

There, t_0 denotes position of the current frame along the temporal axis t. Minimizing for the temporal derivative d_t of the disparities permits linear motions of objects perpendicular to the image plane.

According to Scharstein and Szeliski [2002], most stereo matching algorithms can be subdivided into the following successive stages:

S1	Matching cost computation
S2	Cost aggregation
S3	Minimization
S4	Disparity refinement

For efficiency, steps 1 to 3 usually only compute disparities at integer pixel accuracy. They are finally refined in the optional stage 4 up to a desired subpixel level. In the following, we explain those stages for our stereo matching implementation for a single pair of images. Extension to spacetime stereo is straightforward by adding the additional temporal domain.

S1. Matching cost computation does a pixel-wise correlation by evaluating equation (3.26) for all pixels (x, y) and a set of integer disparities d within a specific range. For each disparity d, all pixels of the left image are compared with those of the right image that has been shifted about d pixels along the x-direction. The result of the comparison using the error metric e_M is stored in a new image. Iteration over all disparities produces a whole stack of difference images, a so-called three-dimensional disparity space image (DSI).

S2. Cost aggregation does the averaging over the matching window $\mathcal{W}$ by computing the sum of equation (3.27). For a rectangular window, this can simply be achieved by applying a box filter to each DSI layer of constant disparity.

S3. Minimization computes the final pixel-accurate disparities by identifying for each pixel in the DSI the layer with minimal cost.

S4. Disparity refinement computes a subpixel-accurate result by minimizing equation (3.28). By constraining the search space of d to a small region around the previously computed coarse disparities, the optimization problem can be solved efficiently using the Levenberg-Marquardt algorithm [Press et al., 1992]. Besides allowing for skew surfaces by optimizing for d_x and d_y, we also handle discontinuities by masking those pixels in the correlation window whose coarse disparities differ too much from the one of the center pixel. This yields a non-rectangular window at depth boundaries. The additionally computed disparity gradients can be also used later to compute the 3D scene representation (see chapter 5).

3.4 Stereo on Structured Light

The presented stereo matching algorithm requires a highly textured scene to find good correlations between different views. It generally fails in reconstructing simple geometry of uniformly colored objects, e.g. white walls. Additionally, the textures should be non-periodic to guarantee unique matches. As a consequence, we add artificial textures to the scene by projecting structured light patterns, as originally proposed by Kang et al. [1995]. We use a binary vertical stripe pattern with randomly varying stripe widths. It yields strong and unique correlations in the horizontal direction of the stereo baseline and is at the same time insensitive to vertical deviations which may occur from inaccuracies in the camera calibration.

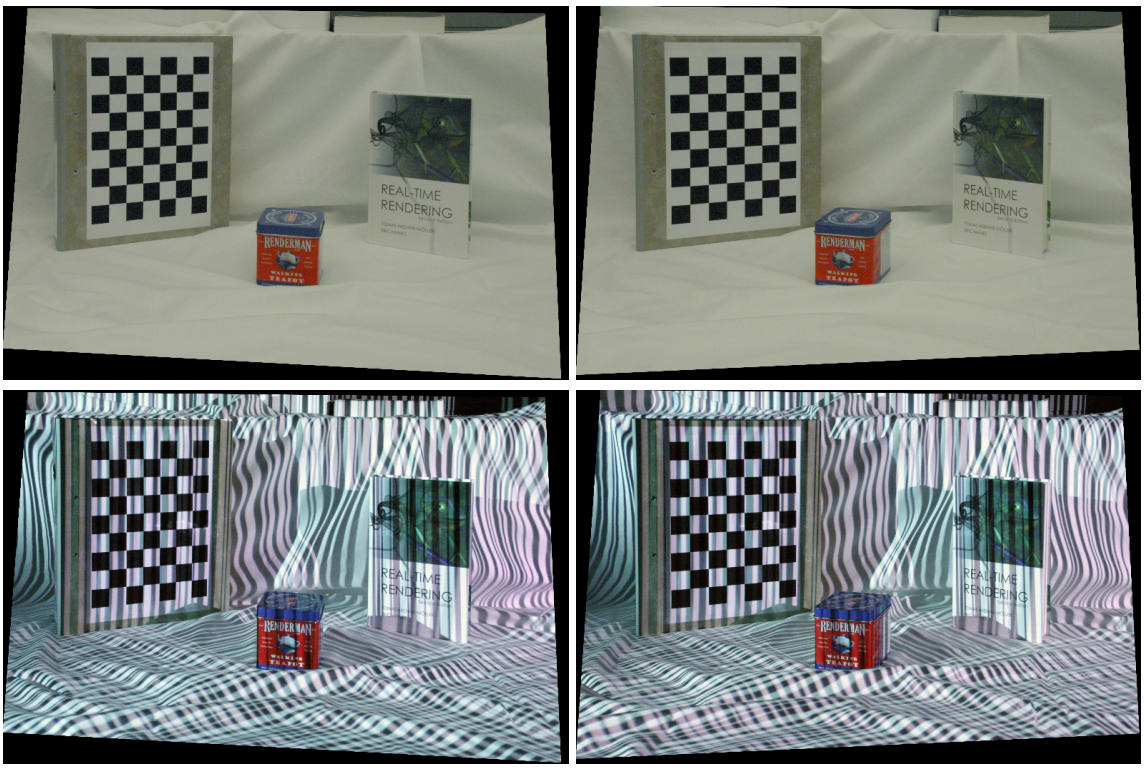


Figure 3.6: Rectified stereo camera images of a scene under uniform illumination (top) and under structured light illumination (bottom).

If spacetime stereo is used, temporally changing patterns can further increase the matching quality. To avoid untextured shadows, multiple overlapping projections can be used for illumination. Unlike pure structured light approaches or stereo matching between a single camera and a projector, our approach has the advantage of also working robustly within those overlap regions up to a certain extent: The dynamic range of the imaging sensors should be high enough to cover both single and overlapping projections with a sufficient contrast. Due to the linearity of light reflection, the minimum pattern contrast of an overlap region is actually equal to the one of a single projection. Practical experience has shown that our acquisition equipment is able to robustly handle up to four overlapping projections, which should be sufficient to cover a whole scene. Moreover, stereo on structured light does not need a projector calibration. Figure 3.6 shows a stereo pair of images from the same scene under uniform and under structured light illumination.

As a result, stereo matching is much more robust, even in untextured regions of the scene. Figure 3.7 shows a comparison of reconstructions of the same scene under uniform illumination and with structured light projections.

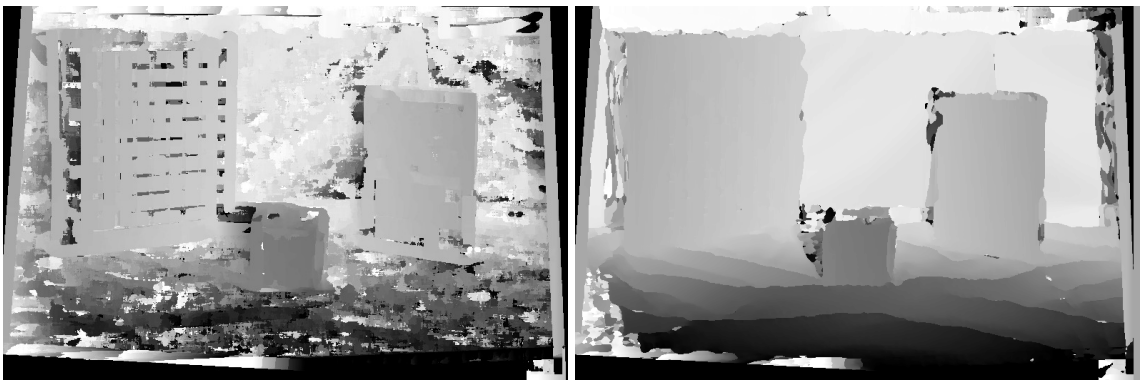


Figure 3.7: Depth map of the scene shown in figure 3.6 reconstructed by local stereo matching under uniform illumination (left) and by stereo on structured light (right). In both cases, a matching window of 25×25 pixels was used.

3.5 Handling Depth Discontinuities

Local stereo matching algorithms generally have difficulties in properly handling depth discontinuities, as illustrated in figure 3.8. Due to occlusions, pixels in one image may not have corresponding partners in the other image. Furthermore, disparities are assumed to change only linearly within the matching window, which is not the case at discontinuities. In the following, two extensions for handling those issues are presented.

3.5.1 Symmetric Stereo Matching

The algorithm presented so far is asymmetric in a sense that it performs matching search only in the right image for fixed pixels of the left image. As a consequence, it assumes that each pixel in the left image should have a corresponding partner in the right image. This is, however, not the case at occlusions. In those situations, the algorithm finds matches that do not exist in reality. A good stereo algorithm should detect those situations and mask all occluded pixels.

This can be achieved by a simple extension called cross-checking. There, the coarse stereo matching in steps one to three is performed twice: search in the right image for pixels corresponding to left image, and vice versa. Corresponding pixels in both disparity maps should have the same values, otherwise they belong to occlusions. A result of symmetric stereo matching is presented in figure 3.9.

As a drawback, time for computing the coarse matching is doubled. This can be overcome by symmetric stereo algorithms [Zitnick and Kanade, 1999; Sun et al., 2005] which search for matches on both sides in one step. However, because processing time of coarse matching can be neglected over the time for subpixel matching, we have chosen cross-checking for simplicity.

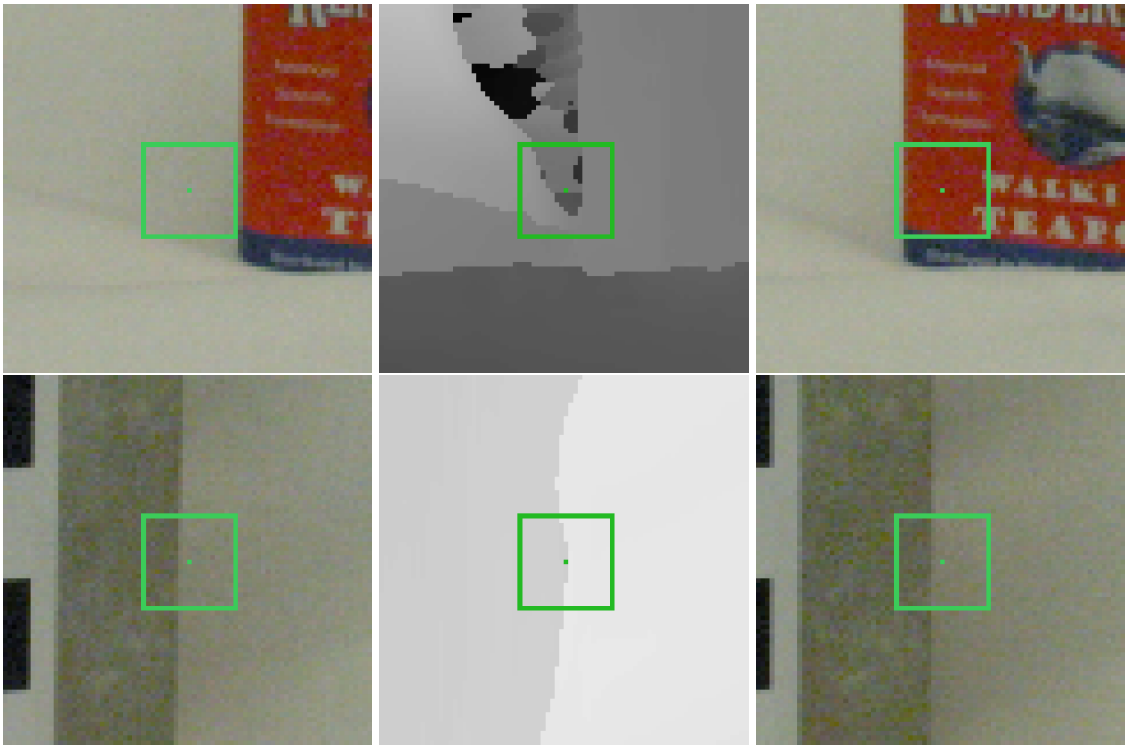


Figure 3.8: Illustration of difficulties in correlating depth discontinuities. Left to right: correlation window in left image, disparity image and right image. Upper row: occlusion. Lower row: sharp color boundary.

Figure 3.9: Symmetric stereo matching. Original scene (left) and computed occlusion mask (right).

3.5.2 Multi-Window Matching

Due to the spatial extend of the correlation window, a second issue arises at discontinuities. Because discontinuities generally come along with sharp color boundaries, they cause a strong correlation. This yields a low matching cost as soon as the pixel window overlaps the color edge, no matter if the center pixel itself is part of that edge. Thus, the matching algorithm tends to find similar disparities in neighborhoods on both sides of color boundaries, which cannot be correct at depth discontinuities.

To solve that, we extend our matching algorithm to a multi-window approach. At each pixel we consider all matching windows of equal size that still contain the pixel of interest and choose the one with the best correlation by extending equation (3.27) to

$$E_M(d) = \min_{(x',y')\in\mathcal{W}(x_0,y_0)} \sum_{(x,y)\in\mathcal{W}(x',y')} e(\mathcal{C}_L(x,y),\mathcal{C}_R(x+d,y)). \tag{3.32}$$

The chosen window usually has minimal overlaps with possible depth discontinuities, yielding a more reliable disparity value. A similar algorithm has already been proposed by Fusiello et al. [1997] who considered five different dilated windows around each pixel at the cost of a five times higher computation time. By sharing computations among neighboring pixels, our algorithm is able to consider all possible dilated windows containing the pixel of interest with low computational costs. It is implemented as a minimum filter on the DSI layers of equal disparity, using the shape of the matching window as structuring element. The filter is applied after the cost aggregation, yielding a new five-step stereo pipeline:

S1	Matching cost computation
S2	Cost aggregation
S3	Minimum filter for discontinuity preservation
S4	Minimization
S5	Disparity refinement

As a result, our algorithm achieves a better reconstruction quality at depth discontinuities with only a small overhead in computational costs. A comparison of conventional window-based stereo and our multi-window approach is presented in figure 3.10.

3.6 Conclusion

We presented a robust depth map reconstruction algorithm based on stereo vision, which builds the basis for our 3D video acquisition studio described in chapter 4. Ambiguities in originally untextured regions of the scene are resolved by generating artificial textures with structured light projections. That way, a local matching algorithm based on pixel windows can robustly find unique correlations in regions

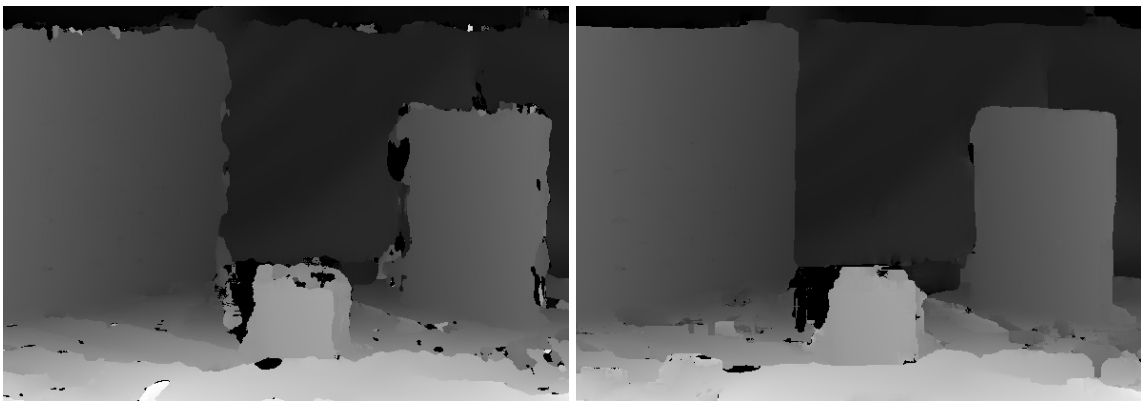


Figure 3.10: Comparison of disparity maps computed by a conventional window-based stereo (left) and our multi-window algorithm (right). Occluded pixels are colored in black. In both cases, a matching window of 25×25 pixels was used. As can be seen, the latter yields better reconstructions at depth discontinuities.

S1	Matching cost computation	9.0 s
S2	Cost aggregation	4.7 s
S3	Minimum filter for discontinuity preservation	9.3 s
S4	Minimization	1.2 s
S5	Disparity refinement	348 s

Table 3.2: Stereo matching performance for an image pair of 791×524 pixels, using a 25×25 pixels correlation window. Timings were measured on a 3 GHz Pentium-4 PC. Note that steps S1 to S4 denote the overall time for two matching processes which are necessary to perform cross checking.

of continuous depth. Discontinuities are handled by using multiple matching windows for each image pixel. This can be implemented with a low additional computational cost by applying a simple minimum filter on the disparity space image. Finally, the initially computed disparities are refined to subpixel accuracy using nonlinear optimization.

The computational effort of our C++-implementation of the whole stereo pipeline is summarized in table 3.2. In the example, a correlation window of 25×25 pixels has been used. Steps S1 and S2 compute the disparity space image using a single matching window for each image pixel in an overall time of 13.7 s. A naive multi-window approach for discontinuity preservation would use for each pixel 25^2 windows, yielding a computation time of $25^2 \cdot 13.7\,\text{s} = 2.4\,\text{h}$. Instead, our equivalent, optimized implementation using the minimum filter in step S3 only increases the time by a factor of 1.5.

Most of the computational time is used by the disparity refinement step. There is still room for improvement because we are currently using a non-optimized standard implementation for the involved nonlinear optimization. However, depending on the employed 3D video data structure and post-processing algorithms,

we do not necessarily need subpixel-accurate data for our application. In chapter 6, we propose a computationally more efficient disparity filter that generates visually pleasing results from only pixel-accurate disparities.

Chapter 4

A 3D Video Acquisition System with Active Illumination

In this chapter, we present a 3D video recording system based on so-called 3D video bricks. They act as low-cost z-cameras capturing color videos and depth maps of the scene from their respective viewpoints. Depth reconstruction is performed using stereo vision on structured light as presented in chapter 3. For that purpose, structured light patterns are captured concurrently to color texture images. Multiple bricks can be placed at will to achieve a large viewing range and to resolve for occlusions. Together, they build a scalable, mobile acquisition studio.

A very similar setup is used by Cotting [2007]. There, multiple bricks are combined to a system in which the projectors collaboratively contribute to a large-scale interactive display. A depth reconstruction method based on imperceptible structured light patterns embedded in the projected images is employed to adapt the display to the projection surface and to recognize interaction gestures. Our system is based on the same hardware components but uses a custom software with different data acquisition, synchronization, pattern projection and depth reconstruction methods, which are specifically tailored to our task of 3D video acquisition.

4.1 Acquisition Hardware

This section presents the complete hardware setup of our 3D video recording studio.

4.1.1 3D Video Bricks

Each 3D video brick concurrently captures stereo images of the scene illuminated by structured light and texture images under constant illumination. For stereo vision, the bricks are equipped with two grayscale cameras and a projector for structured light illumination. Textures are captured concurrently with a color camera. Using an appropriate synchronization, the projectors can optionally be used as a constant light source for texture acquisition. An illustration and a prototype of a 3D video brick is shown in figure 4.1.

In our setup, we use single-chip CCD cameras of the type Dragonfly® from Point Grey Research® which are capable to capture videos in XGA resolution of 1024×768 pixels at 15 frames per second. Their video streams are transmitted via standard IEEE-1394a FireWire interfaces which are merged to a single connector per brick using FireWire hubs. The camera shutters can be synchronized in hardware via a TTL signal over a custom two-pin connector. In that case, however, the maximum achievable frame rate drops to 12 Hz.

Structured light patterns are generated by single-chip DLP projectors of the type NEC® LT240K. They produce images in XGA resolution at 60 frames per second. In contrast to most similar products, those projectors generate their frames synchronously to the VGA input signal which makes them suitable for synchronization with the acquisition camera shutters.

Cameras and projector are mounted on a portable aluminum rig standing on two tripods. The interface of such an integrated system consists of three ports for data transfer and synchronization: one FireWire port outputting the video streams of the cameras, one two-pin TTL port for camera shutter synchronization, and one VGA port for input of the structured light signals and for projector synchronization. Nowadays, as small-size projectors and cameras are available, it would be possible to integrate all components into a small, portable device.

4.1.2 Recording and Projection Infrastructure

A schematic overview over the whole studio infrastructure is depicted in figure 4.2. Each 3D video brick is connected via FireWire to its own computer for recording and processing the video streams of the three cameras. It is a standard PC equipped with two fast hard disks working in parallel in a RAID-0 compound which guarantees the data transfer rate needed for real-time video recording. Besides frame acquisition, the PCs can perform all post-processing steps including depth from stereo extraction. The resulting color and depth videos can finally be collected via Ethernet.

Depth from stereo vision does not require specific textures or structured light images. The patterns can—and should—be random. Our system makes use of that fact by providing the same image data to all projectors. The structured light patterns are generated by one additional dedicated PC and distributed to all 3D

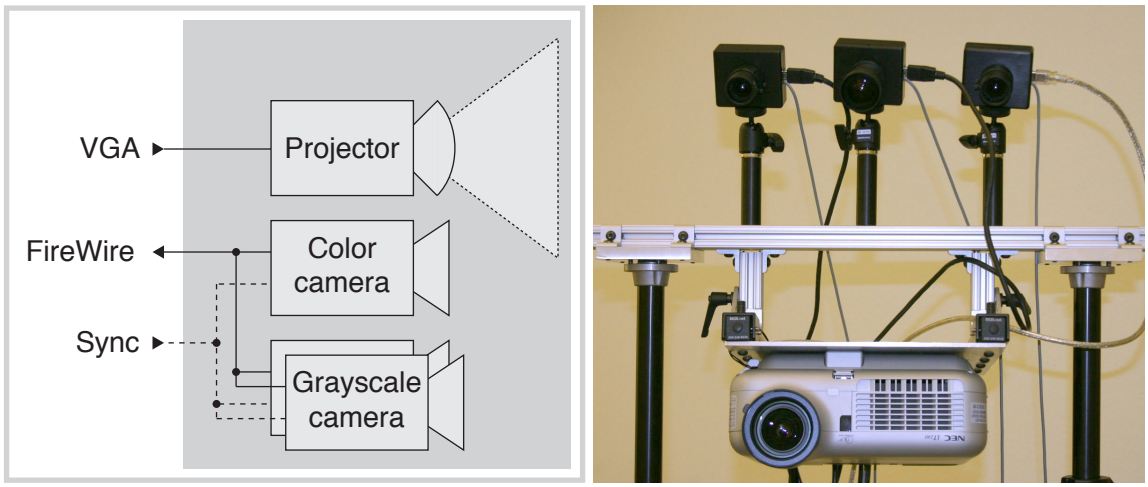


Figure 4.1: Left: Concept of a 3D video brick equipped with a color camera for texture acquisition, a stereo pair of grayscale cameras for structured light acquisition, and a projector for structured light illumination. Right: Prototype implementation.

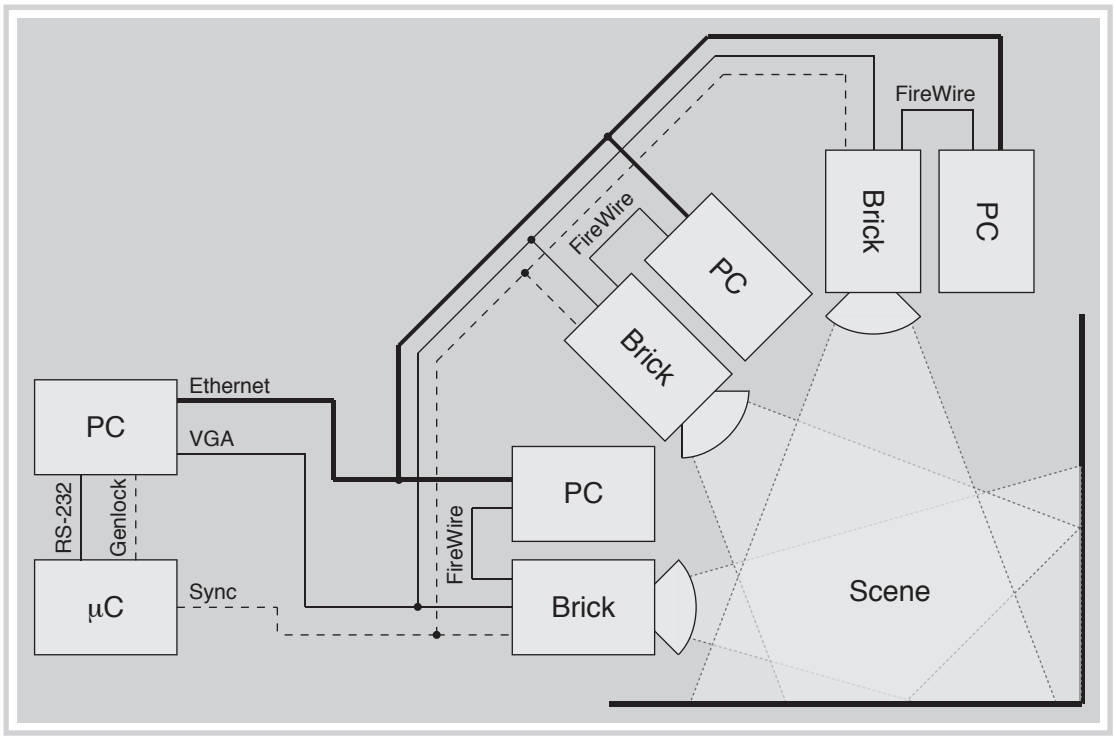


Figure 4.2: Infrastructure of the 3D video studio consisting of multiple acquisition bricks. Each brick transmits its video streams to a dedicated recording PC. A structured light video signal is distributed to all projectors from an additional PC. All components are synchronized to a common reference clock signal generated by a microcontroller. The structured light PC additionally acts as a command console which accesses all other computers via Ethernet.

Figure 4.3: Genlock connector at an NVIDIA® Quadro® FX 3000G graphics board for external synchronization of the video signal.

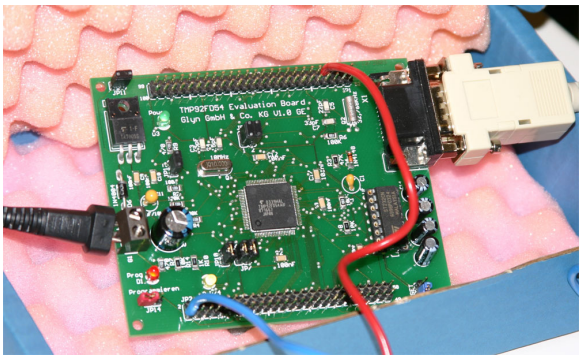


Figure 4.4: Toshiba® TMP92FD54 microcontroller board generating the synchronization signals for the whole 3D video studio.

video bricks via a VGA splitter, which simplifies synchronization because only one image signal has to be generated. This is achieved by a genlock graphics board (NVIDIA® Quadro® FX 3000G) which is able to synchronize its video signal to a clock signal fed into an external connector (cf. figure 4.3). Alternatively, to save additional costs for the genlock hardware, software synchronization [Allard et al., 2003; Waschbüsch et al., 2006] can be used. The pattern generation PC furthermore acts as a master console for controlling the whole acquisition studio via Ethernet.

For synchronization we employ the Toshiba® TMP92FD54 microcontroller unit shown in figure 4.4, running a custom embedded program. It derives two signals from the internal clock of the controller: one at 60 Hz for the projections, and one at 12 Hz for the cameras. The 60 Hz signal is fed into the genlock port of the graphics hardware. The 12 Hz signal is directly distributed in a star topology to all cameras. The response of the camera shutters to that signal can be adjusted using a camera-internal register controlling the shutter delay. This is used for concurrent texture and structured light acquisition, where the color cameras need a different delay than the grayscale cameras, as explained in section 4.2. The synchronization unit itself can be controlled from the master PC to which it is attached via an RS-232 port.

Figure 4.5: Panoramic view of our 3D video recording studio consisting of four 3D video bricks. The scene in the center is illuminated with structured light for depth acquisition. Two additional light sources are installed for capture of high-quality color textures.

4.1.3 Studio Setup

In our reconfigurable acquisition studio, four 3D video bricks are used to resolve for occlusions in the scene and to cover a greater viewing range. Scalability is guaranteed because overlapping projections are explicitly allowed by our depth reconstruction method. As each brick brings its own computer for recording and depth reconstruction, the overall computation load does not increase with the number of bricks.

The brick configurations we used for our data acquisition are described in appendix A. With four bricks, we were able to cover a working volume of $2.8 \times 3.2 \times 1.9\,\mathrm{m}^3$ at a horizontal viewing range of 71 degrees and a vertical range of about 51 degrees. The main constraint of the setup are the projections. They have to cover the whole working volume while also having a certain amount of overlap to resolve for shadows. Using wide angle lenses in front of the projectors would relax this issue and provide more freedom.

Figure 4.5 shows a panoramic view of the complete setup. The four 3D video bricks visible at both sides fully cover the scene in the center with structured light projections. Two studio light sources provide additional illumination for color texture acquisition.

4.2 Simultaneous Structured Light and Texture Acquisition

The various cameras of the acquisition system have to capture the scene under different lighting conditions. For depth extraction, the grayscale cameras need pictures of the scene under a structured light illumination. In contrast, the color cameras need a uniform, bright illumination for obtaining high-quality textures. Both image types have to be captured simultaneously in approximately the same time slots to guarantee a good registration between scene geometry and textures.

In our system, this is achieved by synchronization of the projectors and camera shutters.

All cameras are attached to the common 12 Hz reference clock for triggering their shutters. Additionally, a temporal delay between the release of the triggering signal and the real activation of the shutter can be defined via an internal camera register. Thus, the cameras can be activated at different points in time with only one common synchronization signal. Another camera register controls the shutter time, i.e. the duration of one image acquisition.

The projectors are able to generate 60 frames per second. They all receive the same analog video signal distributed via a VGA splitter, synchronized to the external 60 Hz reference clock using genlock-enabled graphics hardware. However, triggering the video signal is not sufficient. To achieve an accurate cooperation with image acquisition, it is important to understand the internal timing of the employed projectors, which can be uncovered by reverse engineering [Cotting et al., 2004]. The single-chip DLP projectors we use have an internal clock that triggers the frame drawing. They have been carefully chosen such that they are able to synchronize their clock to the VGA signal, which is not necessarily the case for other models. Hence, drawing of the frame is synchronous to the genlocking signal, delayed by a constant time of a few milliseconds. Because this delay is usually not specified, it is important to use identical projector models in the whole system.

As illustrated in figure 4.6, DLP projectors generate images using a digital micromirror device (DMD), a small chip containing one mirror for each image pixel. Each mirror can be flipped into two different positions, reflecting the light either through an optical system onto the screen or to an internal absorber. Varying light intensities are achieved by temporal modulation of the mirror flips. Colored pixels are generated with a rotating color wheel between DMD and light source. Besides red, green, and blue segments, the wheel in our system possesses a fourth, clear segment for boosting the brightness of the projection. At 60 Hz, generation of a whole image takes 16.67 ms. In that time, the color wheel performs two whole rotations. Because we are interested in acquiring only projections consisting of fully black or fully white pixels, the camera shutter times have to be either shorter than about 1 ms, corresponding to the exposure of the white color wheel segment, or they have to be a multiple of 8.33 ms, corresponding to one full rotation of the color wheel.

For concurrent capturing of textures and structured light, our system supports four different synchronization modes as depicted in figure 4.7, three of them additionally use the projections as constant white illumination for texture acquisition. They exploit the fact that the 12 Hz acquisition rate is only one fifth of the projection rate. Their main difference is in the sequence of patterns and the camera synchronization. In the first three modes, the shutters of the grayscale cameras are always exposed to the full time of one structured light projection lasting 16.67 ms.

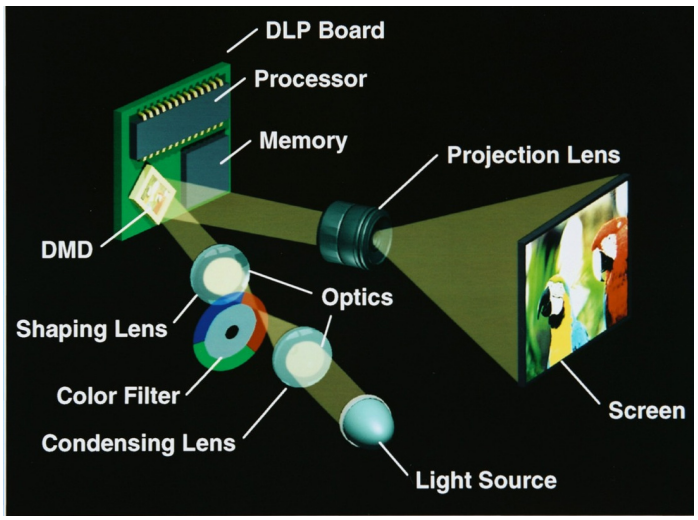


Figure 4.6: Schematic of a DLP projector.

Embedding of black frames. This mode exposes the color cameras to projections of black frames which are generated alternately with structured light patterns. For texture acquisition, the scene can be illuminated with external, high-quality constant light sources. However, their intensity has to be limited in order to maintain a good contrast of the captured structured light images. Moreover, there is a delay of 16.67 ms between a texture and a structured light acquisition.

Embedding of white frames. No external light sources have to be used if white frames are embedded instead of black frames. In that case, the projectors take over the illumination for texture acquisition. The timings are similar to the previous synchronization mode. However, the shutter time of the color cameras has to be exactly either 8.33 ms or 16.67 ms, corresponding to one or two rotations of the color wheel in the projectors.

Inverse pattern projections. By alternating projection of inverse patterns, the delay between texture and structured light acquisition can be reduced. The color camera shutters are triggered 8.33 ms after the beginning of the first projection and kept open for 16.67 ms. Thus, they acquire both a structured light pattern and its inverse, which sums up to an image of the scene illuminated by uniform white light. However, at surfaces moving very quickly towards or away from the camera, slightly visible patterns can be observed, because the projections are not anymore exactly inverse to each other. This effect usually appears in combination with common motion blur artifacts.

Imperceptible structured light. Imperceptible structured light [Cotting et al., 2004] exploits the light modulation of DLP projectors to embed invisible patterns

in any projection. It uses a time slot of 0.1 ms where each micromirror is in a constant position—either bright or dark. The color space of the projector is divided into two disjoint parts where all pixels appear either white or black within this time slot. By using a specialized color dithering approach, binary patterns can be embedded that are only visible to a specially synchronized camera. In our experiments, we used a constant projection of white with an embedded imperceptible pattern only visible to the grayscale cameras. However, due to the very short shutter time, the acquired images were too noisy to achieve a robust result from the depth reconstruction algorithm. But in principle, it would work if more light-sensitive cameras were used. The main advantages of this approach are the close alignment between texture and pattern acquisitions, as well as the constant illumination, producing no flickering visible for the human eye.

Figure 4.8 shows color and structured light images acquired by our system. White projections were used for scene illumination by running the capturing process in the inverse pattern projection mode. The visible projection boundaries in the color images are due to space restrictions of our acquisition studio. We had to position the projectors quite close to the scene. If we had been able to move the projectors further away or to equip them with wide angle lenses, the scene would have been covered completely in uniform white light. If more controllable lighting conditions are necessary, embedding of black frames in combination with studio light sources can be used, resulting in images like those of figure 4.9.

4.3 Camera Calibration

In order to compute valid depth maps and to merge the information gained from several bricks, all cameras in the 3D video system must be calibrated intrinsically and extrinsically. The projectors can remain uncalibrated because they just produce random textures supporting the depth from stereo algorithm.

For camera calibration, we use a custom software based on the calibration routines of the OpenCV library [OpenCV]. It assumes a pinhole camera model with a second order radial and tangential lens distortion, as explained in sections 3.2.1 and 3.2.2. All camera parameters are determined by acquiring images of planar checkerboard targets, as shown in figure 4.10. In a first step, the intrinsic parameters are determined for each camera separately, using a small DIN-A4 ($210 \times 297\,\mathrm{mm}^2$) checkerboard. It has to be acquired from about five different orientations for each camera. To obtain a good estimate of lens distortion, one shot should capture the checkerboard oriented parallel to the image plane, covering the whole field of view up to the image corners. Note that the intrinsic calibration of the whole system has to be performed only once. It remains stable as long as the lens parameters of the cameras are not changed. If a zoom-lens is readjusted, the calibration only has to be repeated for that particular camera. The extrinsic cali-

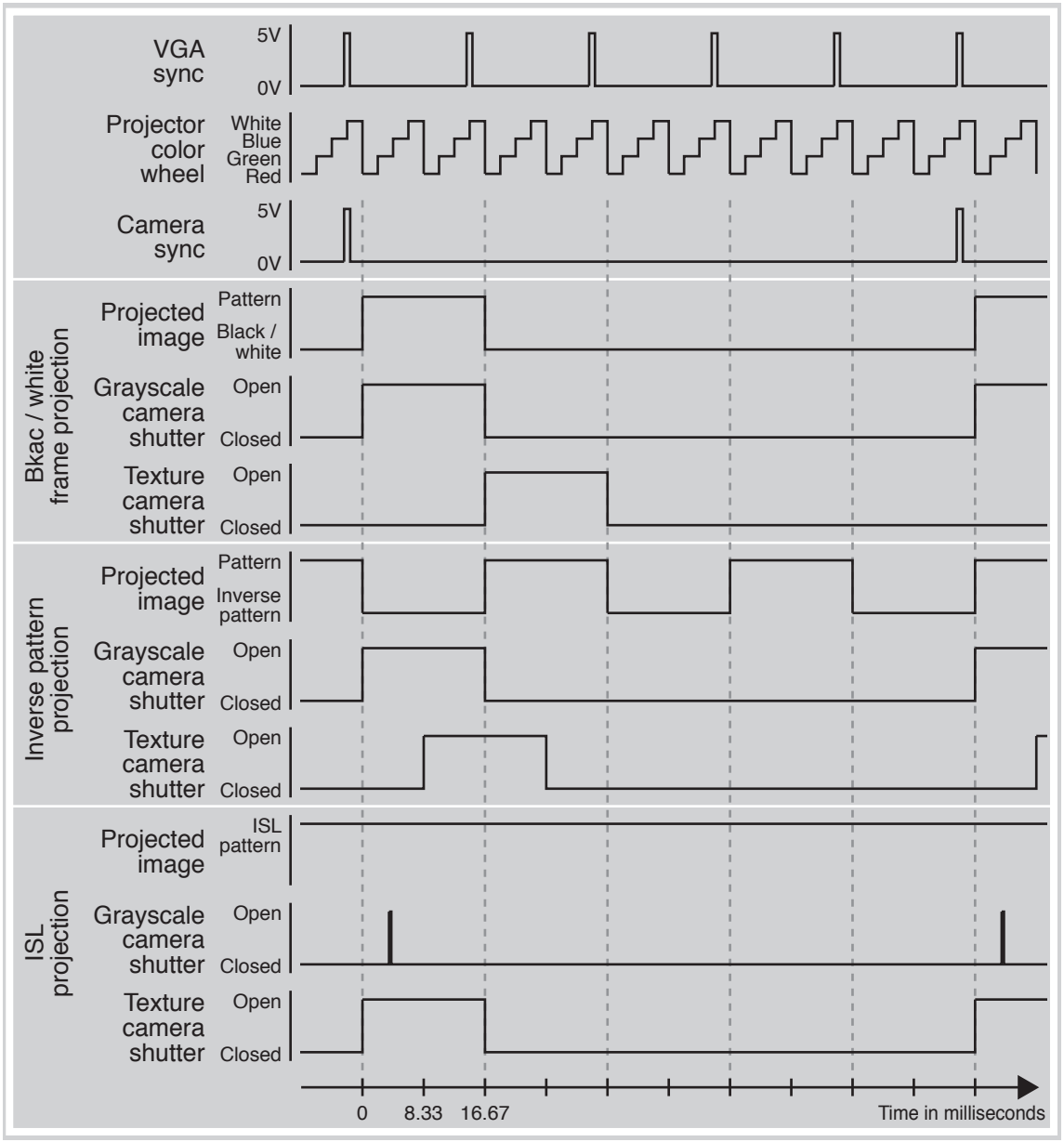


Figure 4.7: Timing diagrams of the different modes for simultaneous acquisition of color textures and structured light patterns supported by our recording system.

bration is determined for all cameras at once by a single shot of a large DIN-A0 ($841 \times 1189\text{mm}^2$) checkerboard in the center of the working volume. It can be performed conveniently before each recording. As a result, we achieve an error of reprojection of the calibration target into the camera images that is below one quarter of a pixel. This corresponds to a stereo triangulation accuracy below 1 cm in the three-dimensional volume at three meters distance to the camera.

Self-calibration methods such as the one by Svoboda et al. [2005] may be an

Figure 4.8: Camera images of our 3D video acquisition system captured using the inverse pattern projection mode. The scene has been recorded from three input views, using the projectors for texture illumination.

alternative to our approach, especially for a viewing range of 360 degrees, where our routine would require a more complicated three-dimensional calibration target that can be seen from all directions. However, in our experiments with our constrained viewing range, our method performed more robustly than self-calibration, yielding a similar accuracy. While the approach by Svoboda et al. [2005] in general needs less user interaction for intrinsic and extrinsic calibration together, our approach is less time-consuming if only the extrinsic parameters have to be updated.

4.4 Real-Time Recording

Each 3D video brick is equipped with three cameras for capturing frames of 1024×768 pixels at 12 frames per second. A recording device must be able to store a continuous data stream at 27 megabytes per second. We use a low-cost solution with a conventional office PC attached to each brick via the FireWire bus.

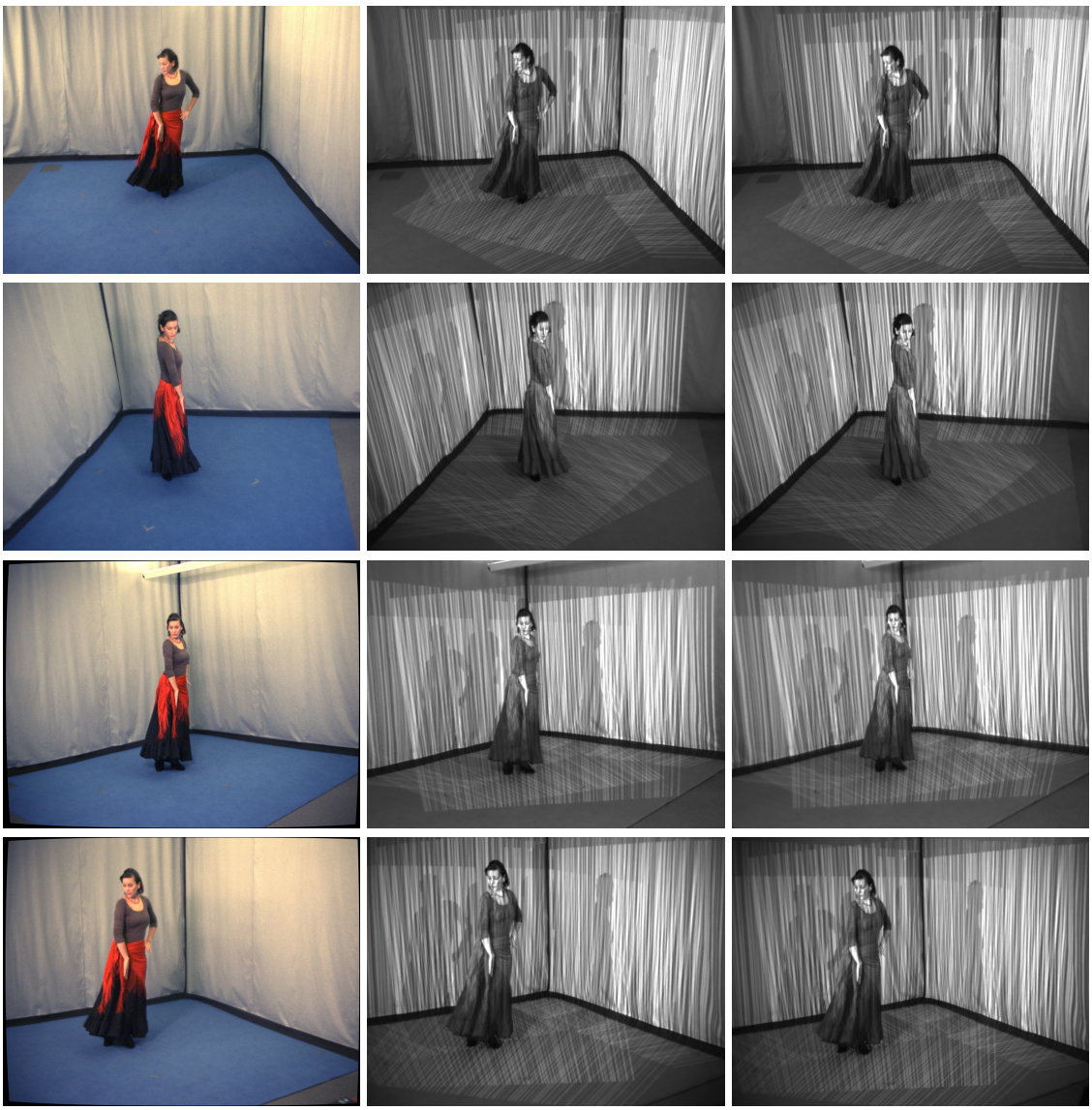


Figure 4.9: Camera images of our 3D video acquisition system captured using the black frame embedding mode. The scene has been recorded from four input views, using additional studio lights for texture illumination.

To guarantee the necessary data rate without interruptions, each PC is equipped with two IDE hard disks working in parallel in a RAID-0 compound.

However, even in such a setup, frame-losses couldn't be prevented in our first experiment that used conventional file I/O operations for data storage. It seems that the asynchronous write buffer or the disk block allocation routines of the file system are causing sparse but unpredictable delays. As a solution, our recording software preallocates a large continuous block as one file on a separate hard disk partition. This file is used as cache that receives the raw data stream using unbuffered write operations. With that method, our system is able to avoid frame

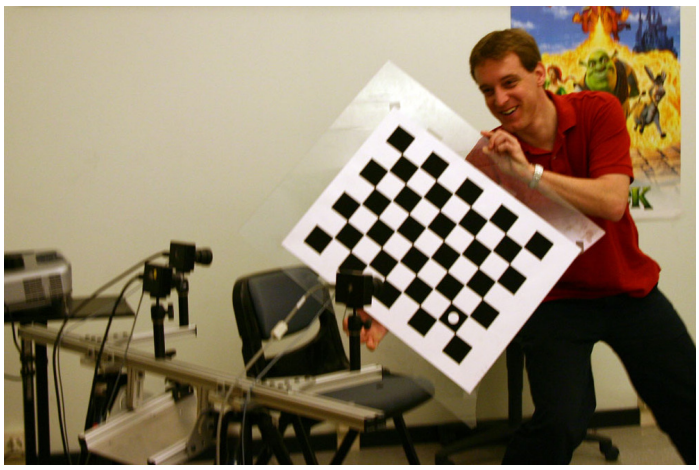


Figure 4.10: Camera calibration with a checkerboard target.

losses even when recording at 15 frames per second in the unsynchronized case. That could be shown in an empirical study where the recording system was running without interruptions for a whole day.

After recording is finished, the raw data is extracted offline from the cache and converted into sequences of image files. During that process, images of the color cameras are computed using the demosaicing algorithm of Malvar et al. [2004]. Furthermore, all images are undistorted using the parameters obtained in the camera calibration process.

4.5 Results

Figures 4.11 and 4.12 show color images acquired from multiple views with our 3D video recording system and corresponding depth maps reconstructed from the structured light pattern images using the stereo vision algorithm of chapter 3. In order to simplify the later processing steps, the depth maps have been warped into the views of the color cameras of the corresponding acquisition bricks, yielding one-to-one correspondences between depth and color pixels.

The taekwondo scene of figure 4.11 has been recorded at 10 frames per second using three bricks covering a horizontal angle of 61 degrees and a vertical angle of 40 degrees. Depth reconstruction has been performed without occlusion detection and discontinuity preservation. As a result, object silhouettes have a jagged appearance in the depth maps and contain many outliers. That issue has to be handled by later post-processing steps.

The flamenco scene of figure 4.12 has been recorded at 12 frames per second using four bricks covering a horizontal angle of 71 degrees and a vertical angle of 51 degrees. This time, the full depth reconstruction pipeline has been applied, including occlusion detection and discontinuity preservation. Depth pixels that

Figure 4.11: Color images and corresponding depth maps of the taekwondo scene acquired by three 3D video bricks.

are likely to be more inaccurate are marked as invalid. The resulting depth maps contain more holes and, thus, only provide a conservative representation of the scene geometry. The holes can be filled using the information from neighboring bricks, which is easier than removing invalid depth values in a later stage. Object silhouettes are represented quite accurately by the valid depth values.

An summary of all data sets that we acquired is given in appendix A.

4.6 Conclusion

Our 3D video acquisition system captures dynamic real-world scenes synchronously from a sparse amount of input viewpoints. The scalable setup of mobile 3D video acquisition bricks can be easily adapted to the scene complexity and the desired virtual viewing range during playback. Color textures are acquired concurrently with stereo grayscale images under structured light illumination, optionally using the projectors as white light sources. Based on the pattern images, a stereo matching algorithm computes dense depth maps from all input viewpoints.

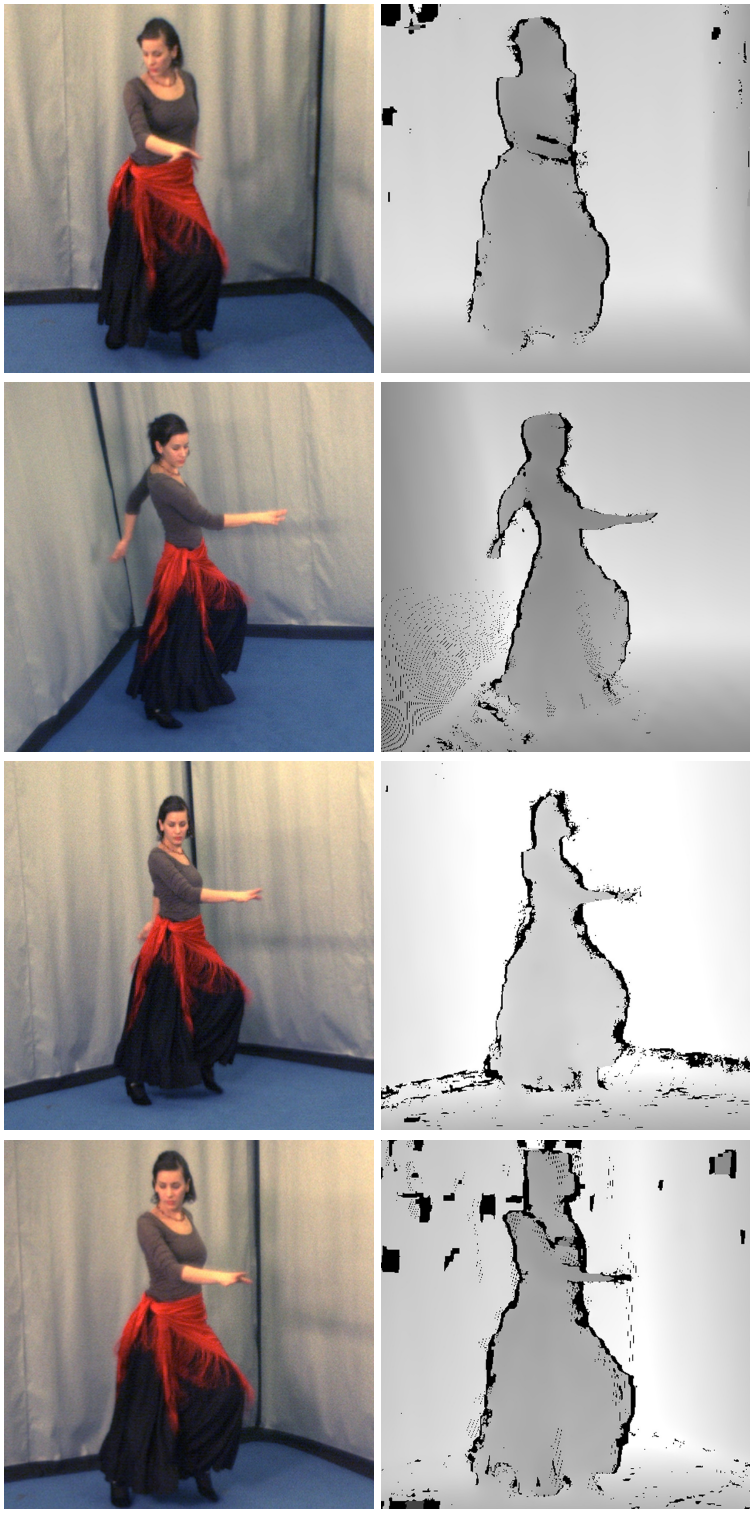


Figure 4.12: Color images and corresponding depth maps of the flamenco scene acquired by four 3D video bricks.

Chapter 5

Point-Sampled 3D Video

For conventional two-dimensional video, there exists a well-established, generic data representation: the image pixel. It is a simple yet powerful primitive capable of modeling 2D views of arbitrary complex scenes. It builds the basis for a large amount of applications ranging from image processing, video editing, streaming, and compression. To establish 3D video as a novel form of multimedia content, a similarly generic data primitive is needed.

In this chapter, we propose to use points that model all surfaces of a three-dimensional scene as an irregular cloud of samples. By sampling not only appearance but also geometry, points can be seen as a generalization of pixels to the third spatial domain [Würmlin, 2007]. The irregularity of the point cloud permits scalability: data from multiple 3D video acquisition bricks can be merged into one consistent, view-independent model of the scene. Sampling resolution can be locally adapted to the desired amount of detail. Unlike image-based structures such as such as those used by Zitnick et al. [2004], it is possible to keep the amount of data low by removing redundant points from the geometry [Pauly et al., 2002; Sadlo et al., 2005].

Compared to mesh-based methods, points provide advantages in terms of scene complexity because they reduce the representation to the absolutely necessary data and do not carry any topological information which is often difficult to acquire and maintain. For applications such as rendering or certain geometry processing tasks, a continuous surface is constructed locally on demand using efficient splatting (cf. section 5.1.2) or moving least-squares interpolation [Levin, 2003] algorithms. As each point in our model has its own assigned color, we also do not have to deal with texturing issues. The uniformity of the data structure allows for simple but efficient streaming and compression algorithms, as we show in chapter 7. Moreover, in combination with the spatio-temporal data structure of chapter 8, point samples are very suitable for 3D video editing applications as presented in chapter 9.

Specifically for scanned real-world data, we propose a probabilistic model where each point has an associated distribution describing the uncertainty of its position. This allows for modeling noise and errors introduced in the acquisition

and reconstruction process. In particular, we provide a method for modeling quantization noise of the image pixel grid and camera calibration errors. These errors are considered in our probabilistic rendering method which is able to generate smooth images out of noisy input data.

The next section introduces the fundamentals of point-based representations and rendering. Subsequently, our specialized probabilistic point-based 3D video data model is presented, followed by descriptions of post-processing algorithms and image generation methods. The chapter concludes with a discussion of the achieved results.

5.1 Introduction to Point-Sampled Geometry

In this section, we give an introduction into point-based representations and rendering algorithms that build the basis for the data models used in this thesis. For a more extensive overview over point-sampled geometry, including topics like surface analysis, filtering, resampling, and feature extraction, we refer to the book by Gross and Pfister [2007].

5.1.1 Data Model

As first introduced by Levoy and Whitted [1985], points can be used as an efficient representation for complex geometries alternatively to traditional triangles. They provide a compact model of geometry and appearance of three-dimensional scenes by storing local samples of specific attributes such as *position* or *color*. The complete scene is modeled by an irregular cloud of many points [Grossman and Dally, 1998]. Depending on the sampling, they can be used for modeling both surfaces [Pfister et al., 2000] and volumes [Zwicker et al., 2001b]. In contrast to more traditional meshes, points do not store any neighborhood information. Thus, they do not explicitly model topology.

All point representations used in this thesis follow the generic framework described by Zwicker et al. [2001b]. There, a point sample is defined completely by the values of its attributes. An example of a typical attribute for a k-dimensional geometry is shown in table 5.1. Usually, the geometry space is three-dimensional, i.e. $k = 3$, but we will later also define point samples in four-dimensional space-time.

The most important attribute is the position $\mathbf{X}$ which is generally used to identify a specific point in the cloud. Typically, all points are stored in a spatial query data structure like a kd-tree [Bentley, 1975] according to their position attribute. This is used as a basis for most algorithms processing the geometry.

Symbol	Data type	Description
$\mathbf{X}$	k-dim. vector	position
V	$k \times k$-matrix	covariance matrix
c	scalar triple	color

Table 5.1: Typical attributes of a point primitive.

During rendering, all points are projected onto the screen. In order to be visible at all, their projections have to cover a certain amount of image pixels. Therefore, points do generally not represent infinitesimally small Dirac samples but they describe small volumes of space around their center position. The volume is often modeled as a k-dimensional Gaussian normal distribution

$$N_k(\zeta;\mathbf{X},\mathrm{V}) = \frac{1}{\sqrt{(2\pi)^k|\mathrm{V}|}} e^{-\frac{1}{2}(\zeta-\mathbf{X})^{\mathrm{T}}\mathrm{V}^{-1}(\zeta-\mathbf{X})} \tag{5.1}$$

centered at the sample position $\mathbf{X}$. The covariance matrix V describes its ellipsoidal shape. Intuitively, the Gaussian blurs the point samples to some extent and interpolates their attributes in the space in-between. That way, points are able to model closed surfaces or volumes. Because its infinite extent is inefficient to handle, N_k is often only evaluated in a region up to a certain user-defined cutoff radius r_C, i.e. for all ζ with $(\zeta-\mathbf{X})\mathrm{V}^{-1}(\zeta-\mathbf{X}) < r_C^2$. Outside that region, the function is assumed to be zero.

The covariance matrix V provides a unified framework for the various specialized point primitives used in this thesis. It can be composed from k-dimensional vectors $\mathbf{T}_1,\ldots,\mathbf{T}_k$ that span the k-dimensional Gaussian ellipsoid as

$$\mathrm{V} = \Sigma \cdot \Sigma^{\mathrm{T}} \tag{5.2}$$

with

$$\Sigma = \begin{pmatrix} \mathbf{T}_1 & \cdots & \mathbf{T}_k \end{pmatrix}. \tag{5.3}$$

In practice, one or more of the spanning vectors may be zero, collapsing the Gaussian into a lower-dimensional ellipsoid.

A typical special case consists of two-dimensional elliptical discs, so-called surfels [Pfister et al., 2000], that are tangentially aligned to the three-dimensional surface they represent (cf. figure 5.1). Their non-zero spanning vectors $\mathbf{T}_1$ and $\mathbf{T}_2$ may be obtained by any of the available point-based algorithms such as those contained in Pointshop 3D [Zwicker et al., 2002a]. Alternatively, if the 3D video acquisition system is able to provide stable depth values z along with their gradients z_x and z_y, they can be computed by backprojecting the footprint of an image pixel into three-space. Based on the rotation matrix R, projection matrix P, and center of projection $\mathbf{O}$ of the acquisition camera, the center $\mathbf{X}$ of the surfel corresponding to a pixel (x,y) can be computed as

$$\mathbf{X} = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot (x,y,1)^{\mathrm{T}} \cdot z + \mathbf{O}. \tag{5.4}$$

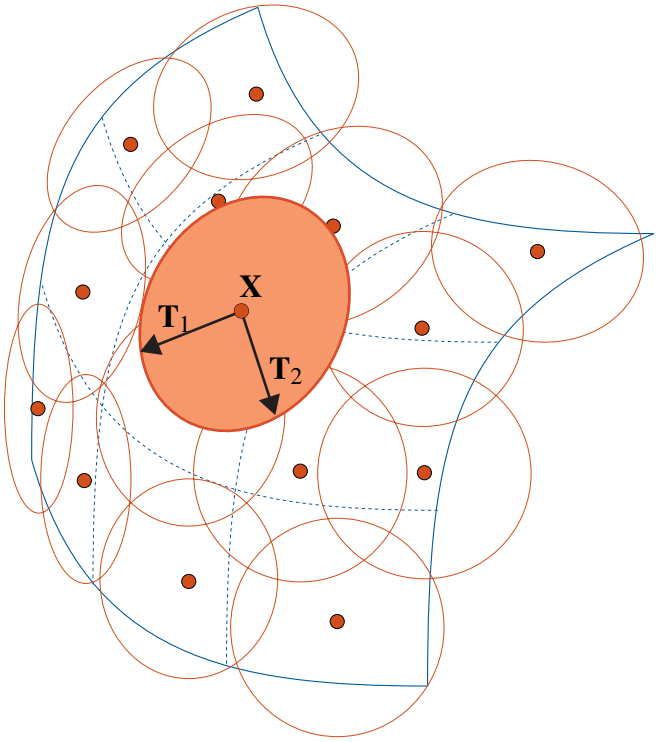


Figure 5.1: Surfels model three-dimensional surfaces as a set of small tangentially aligned ellipses.

Calculation of the spanning vectors follows from differentiation:

$$\mathbf{T}_1 = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot \left(z \cdot (1,0,0)^{\mathrm{T}} + z_x \cdot (x+1, y, 1)^{\mathrm{T}} \right), \tag{5.5}$$

$$\mathbf{T}_2 = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot \left(z \cdot (0,1,0)^{\mathrm{T}} + z_y \cdot (x, y+1, 1)^{\mathrm{T}} \right). \tag{5.6}$$

Table 5.2 lists all types of point samples that are used in this thesis.

5.1.2 Rendering

Images of point clouds can be generated in a forward rendering process called splatting. In the following, we give a summary of the basic algorithm. Besides its simplicity, it has the advantage of being based on a solid signal processing framework which allows for extensions such as high-quality texture antialiasing. Details can be found in the papers by Zwicker et al. [2001a, 2002b, 2001b]. Moreover, it can be easily implemented on the GPU [Ren et al., 2002; Botsch et al., 2002].

The algorithm in figure 5.2 shows the point splatting procedure in pseudocode notation. It consists of four basic steps:

Description	Dimensions of space	Dimensions of Gaussian	Representation	Usage in thesis
Circular disk	3	2	Normal vector, disk radius	Model compression (chapter 7)
Elliptical disk	3	2	2 spanning vectors	Surfel-based 3D video (section 5.1)
3D ellipsoid	3	3	3 spanning vectors	Probabilistic 3D video model (chapter 5.2)
Ellipsoid in 4D spacetime	4	4	3 spanning vectors in space, 1 in time	Video hypervolume (chapter 8)

Table 5.2: Different point types used in this thesis.

1. Projection onto image plane
2. Fuzzy depth test
3. Rasterization and blending
4. Normalization

Step one is performed in line 3 of the code listing. There, each point is projected onto the image plane together with its associated Gaussian normal distribution, yielding a screen space kernel $A_S(x,y)$ centered around a pixel (x_0,y_0), serving as an alpha mask for rasterization. Many point renderers use a local affine approximation of the projection operator [Zwicker et al., 2001a] for simplification of the rasterization stage because the screen space kernels are then again Gaussians in two dimensions.

The fuzzy depth test resolves for occlusions. Because Gaussians of neighboring points from the same surface tend to have some overlap, this step differs from traditional image generation approaches. Occlusions should only occur if screen space kernels from different surfaces contribute to the same pixel. If the kernels belong to neighboring points of the same surface, they should all contribute to the pixel color. This can be achieved approximately by a fuzzy depth test. Similar to conventional depth tests, depths z_S of fragments (x,y) are written into a depth buffer $\mathcal{Z}_S(x,y)$. A global value Δz_S defines a minimum distance that is considered as occlusion. If the depth z_S of a new fragment (x,y) differs no more than Δz_S from the depth buffer value $\mathcal{Z}_S(x,y)$ (line 9), there is no occlusion and the fragment color is blended with the color buffer value in the rasterization stage. Otherwise, an occlusion has occurred and the new pixel value is either rejected if $z_S > \mathcal{Z}_S(x,y) + \Delta z_S$ or depth and color buffer are overdrawn if $z_S < \mathcal{Z}_S(x,y) - \Delta z_S$ (line 5).

The third step performs the actual rasterization. Each incoming fragment (x,y) has an associated depth z_S, color c and transparency value $A_S(x,y)$. Depending on the result of the fuzzy depth test, rasterization either overdraws the framebuffers (lines 6 to 8) or blends their content with the attributes of the new fragment (lines 10 and 11) to obtain a smooth transition of neighboring surface samples.

```
1  foreach pixel (x,y) do A_S(x,y) ← 0, Z_S(x,y) ← ∞                ▷ Initialization
2  foreach point (X, V, c) do
3      compute screen-space kernel A_S(x,y) and depth z_S            ▷ Projection
4      foreach pixel (x,y) with A_S(x,y) > 0 do
5          if z_S < Z_S(x,y) − Δz_S then                             ▷ Fuzzy depth test
6              Z_S(x,y) ← z_S                                        ▷ Rasterize
7              C_S(x,y) ← c
8              A_S(x,y) ← A_S(x,y)
9          else if z_S ≤ Z_S(x,y) + Δz_S then                        ▷ Fuzzy depth test
10             C_S(x,y) ← A_S(x,y) · c + A_S(x,y) · C_S(x,y)         ▷ Rasterize and blend
11             A_S(x,y) ← A_S(x,y) + A_S(x,y)
12         end
13     end
14 end
15 foreach pixel (x,y) do                                           ▷ Normalization
16     C_S(x,y) ← C_S(x,y)/A_S(x,y)
17     A_S(x,y) ← 1
18 end
```

Figure 5.2: Point splatting algorithm.

Blending is performed in an additive way by setting the new pixel color $\mathcal{C}_S(x,y)$ to $A_S(x,y)c + \mathcal{A}_S(x,y)\mathcal{C}_S(x,y)$ and its alpha value $\mathcal{A}_S(x,y)$ to $A_S(x,y) + \mathcal{A}_S(x,y)$.

When all points have been rasterized, framebuffer pixels still have varying transparency values depending on the amount of overdraw caused by the additive blending function. To render constant opaque surfaces, the final color buffer is normalized in step four (lines 15 to 18) by dividing all pixel colors by their alpha values.

We did an efficient GPU implementation of the above splatting algorithm using OpenGL vertex and fragment shaders, similar to the one by Botsch et al. [2002], with the addition of a screen-space EWA texture antialiasing filter [Zwicker et al., 2001a]. Local affine projection [Zwicker et al., 2001a] is performed in a vertex shader, rasterization is done in a fragment program. The fuzzy depth test is not supported in current graphics hardware but can be simulated by a two-pass rendering approach called visibility splatting [Ren et al., 2002]: With the hardware depth test enabled, the scene is first rendered into the depth buffer only. In the second pass, the depth test is still enabled but the depth buffer is write protected. During rendering, all fragment depths are decremented by $2\Delta z_S$ to simulate the fuzzy depth test. The final normalization step is implemented as a fragment program running in one pass over all framebuffer pixels. Our renderer is able to achieve an average performance of about eight million splats per second on a NVIDIA® GeForce® 7800 GS graphics board.

5.2 Probabilistic Point Samples for 3D Video

While surfels provide a good model for artificial geometry, they are not an optimal solution for noisy, real-world data. To compute an accurate alignment of the surfels, stable surface normals are needed, which are difficult to obtain from 3D reconstruction methods. Disturbed normals produce visible artifacts in rendered images.

Therefore, we propose a different approach, similar to the one by Hofsetz et al. [2005]. Every point is modeled by a three-dimensional Gaussian ellipsoid with covariance matrix V spanned by the vectors $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ around its center $\mathbf{X}$. This corresponds to a probabilistic model describing the positional uncertainty of each point by a trivariate normal distribution.

To estimate V, Hofsetz et al. [2005] have chosen an approach based on the quality of the pixel correlation of the stereo matching. However, it turns out that those heuristic uncertainties are quite large compared to the high-quality disparities we are able to obtain from our structured light assisted approach, resulting in too blurry images. Consequently, we propose a different approach that constrains the uncertainties to cover only small but well-defined acquisition errors. We assume that most disparities are correctly estimated up to small errors caused by quantization noise of the image pixel grid and deviations in camera calibration.

Assuming a Gaussian model for each image pixel, we first compute the back-projection of the pixel into three-space which is a 2D Gaussian parallel to the image plane spanned by two vectors $\mathbf{T}_x$ and $\mathbf{T}_y$. Extrusion into the third domain by adding a vector $\mathbf{T}_z$ guarantees a full surface coverage under all possible views. This is illustrated in figure 5.3.

Each pixel (x,y) is spanned by orthogonal vectors $\sigma_x(1,0)^{\mathrm{T}}$ and $\sigma_y(0,1)^{\mathrm{T}}$ in the image plane. Assuming a positional deviation σ_C, the uncertainties of pixel width and height are $\sigma_x = \sigma_y = 1 + \sigma_C$. σ_C is estimated to be the average reprojection error of our calibration routine.

The depth z of each pixel is inversely proportional to its disparity d as defined by the equation

$$z = -\|\mathbf{O}_L - \mathbf{O}_R\| \cdot \frac{f_x}{d}, \tag{5.7}$$

where f_x is the x-component of the focal length of the rectified camera pair, $\mathbf{O}_L$ and $\mathbf{O}_R$ are the centers of projection, and $p_{x,L}$ and $p_{x,R}$ the x-coordinates of the principal points. The depth uncertainty σ_z is obtained by differentiating equation (5.7) and augmenting the derivative d_x of the disparity with the uncertainty σ_C:

$$\sigma_z = \|\mathbf{O}_L - \mathbf{O}_R\| \cdot \frac{f_x}{d^2} \cdot (d_x + \sigma_C)\,. \tag{5.8}$$

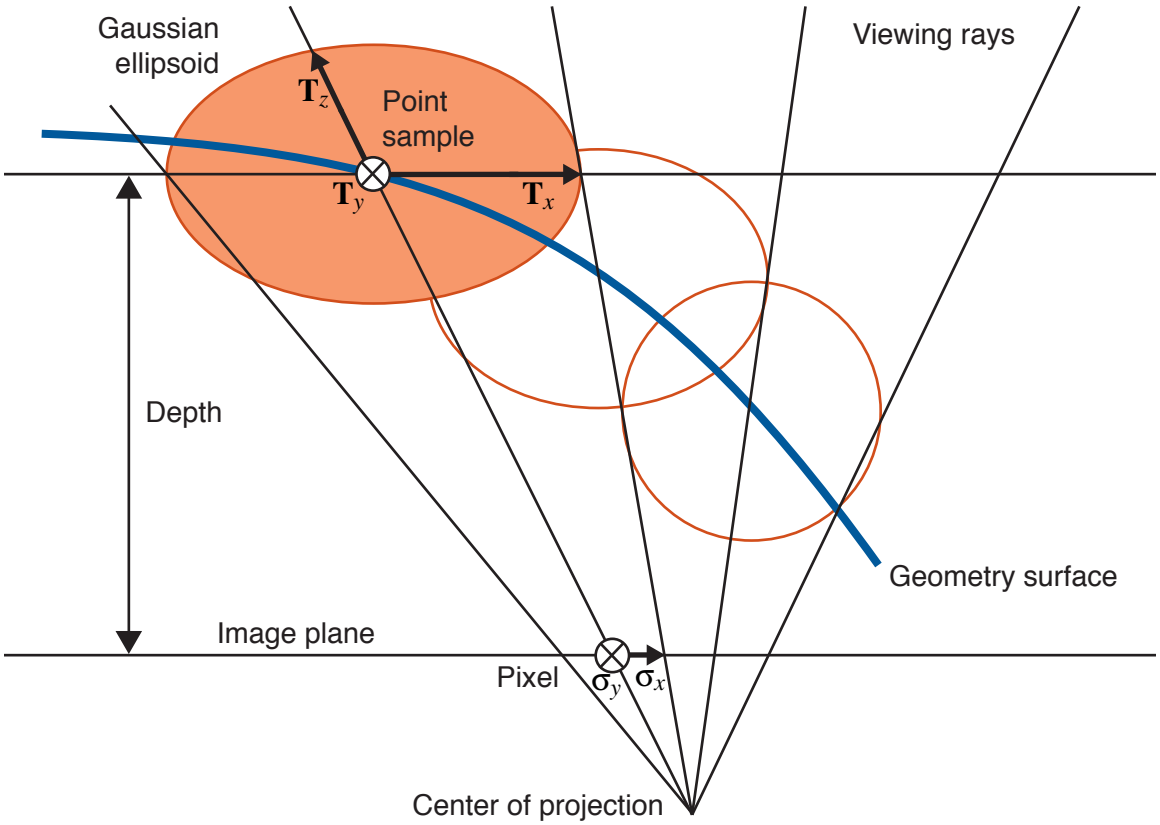


Figure 5.3: Construction of a three-dimensional Gaussian ellipsoid.

Now we can construct for each pixel its Gaussian covariance matrix

$$\mathrm{V}_R = \Sigma_R \cdot \Sigma_R^{\mathrm{T}} \tag{5.9}$$

in ray space with

$$\Sigma_R = \begin{pmatrix} \sigma_x \cdot z & 0 & \sigma_z \cdot x \\ 0 & \sigma_y \cdot z & \sigma_z \cdot y \\ 0 & 0 & \sigma_z \end{pmatrix}. \tag{5.10}$$

It is transformed into the world coordinate system by

$$\mathrm{V} = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot \mathrm{V}_R \cdot (\mathrm{P}^{-1})^{\mathrm{T}} \cdot (\mathrm{R}^{-1})^{\mathrm{T}} \tag{5.11}$$

using the camera rotation matrix R and projection matrix P.

The centers $\mathbf{X}$ of the ellipsoids are constructed by backprojection as

$$\mathbf{X} = \mathrm{R}^{-1} \cdot \mathrm{P}^{-1} \cdot (x, y, 1)^{\mathrm{T}} \cdot z + \mathbf{O}, \tag{5.12}$$

where $\mathbf{O}$ is the center of projection of the camera.

5.3 Scan Merging

Multiple scans from different views are merged into one consistent, view-independent 3D model of the scene by backprojecting all points into the same world reference frame. The scalability of our system permits simple addition of further input viewpoints in order to achieve a greater virtual viewing range and to resolve for occlusions.

After backprojection, the point model still contains outliers and falsely projected samples, especially at object silhouettes due to instabilities of the stereo reconstruction at depth discontinuities. While those points look correct from their originating viewpoint, they produce visible artifacts in renderings of other views, disturbing the overall appearance of the 3D video. Thus, we remove those points by checking the whole model for photo consistency with all texture cameras. This is similar to space carving [Kutulakos and Seitz, 1999] or generalized voxel coloring [Culbertson et al., 1999], but unlike those algorithms, our method works on the already reconstructed irregular point samples instead of a voxel grid.

The method, which is outlined in the algorithm of figure 5.4, works by successively rendering the complete point cloud in all input views using a modified version of the previously explained splatting algorithm. During rasterization, the colors c of the pixels (x,y) of each splat are compared with those of the corresponding input camera image $\mathcal{C}_v$. If the deviation exceeds a user-defined threshold Δc_P (line 15), the splat is inconsistent with the input image and the corresponding point is removed from the data model (line 16). Otherwise, the splat is rasterized into the fuzzy z-buffer (lines 18 to 22) for occlusion handling. Areas of the scene that are occluded in the current rendering view are generally inconsistent with the input image. Therefore, occluded splats should not be tested for consistency at all, which is achieved by rendering the points in increasing depth order (lines 3 to 6) starting with those that are closest to the camera.

As consistency measure, the color deviations of all unoccluded pixels of a splat are weighted by its alpha mask and summed up by computing

$$\Delta c = \sum_{(x,y)} A_S(x,y)\xi_v(x,y)(\mathcal{C}_v(x,y) - c), \tag{5.13}$$

in lines 9 to 14, where $\xi_v(x,y) = 1$ if fragment (x,y) of the splat is visible in the current view v and $\xi_v(x,y) = 0$ if it is occluded. The colors c, $\mathcal{C}_S(x,y)$, Δc, and Δc_P denote three-dimensional vectors in the employed color space. Therefore, the comparison in line 15 is actually performed component-wise. If a material is not perfectly Lambertian it is likely that the luminance of reflected light shows a higher variance over the reflection angle than its color. To cope with that, our algorithm expresses Δc in YUV color space which separates the luminance Y from chrominance U and V. We can then chose a higher threshold for the Y-component (e.g. 40% of the maximum possible value) than for the U- and V- components (e.g. 15% of the maximum possible value).

```
1  foreach input view v do
2      foreach pixel (x,y) do Z_S(x,y) ← ∞                          ▷ Initialization
3      foreach point i with attributes (X, V, c) do                  ▷ Depth sorting
4          Compute depth z_S in view v
5      end
6      Sort points according to increasing z_S
7      foreach point i with attributes (X, V, c, z_S) do
8          Compute screen-space kernel A_S(x,y) in view v            ▷ Projection
9          Δc ← 0                                                    ▷ Color deviation
10         foreach pixel (x,y) with A_S(x,y) > 0 do
11             if z_S < Z_S(x,y) + Δz_S then
12                 Δc ← Δc + A_S(x,y) · (c − C_v(x,y))
13             end
14         end
15         if ∃j ∈ {1,2,3} : |Δc_j| > Δc_Pj then                     ▷ Consistency test
16             remove point i                                        ▷ Delete inconsistent point
17         else
18             foreach pixel (x,y) with A_S(x,y) > 0 do              ▷ Rasterize...
19                 if z_S < Z_S(x,y) + Δz_S then                     ▷ ...consistent point
20                     Z_S(x,y) ← z_S
21                 end
22             end
23         end
24     end
25 end
```

Figure 5.4: Algorithm for photo consistency enforcement.

As a result, enforcing photo consistency considerably improves the seamless fit of multiple acquired depth maps in our model. The reduction of artifacts can be clearly seen in figure 5.5. Nevertheless, there still remain some issues with mixed pixels, i.e. silhouette pixels possessing a color that is interpolated from different surfaces. These tend to produce holes in the cleaned model. This may be solved using boundary matting techniques which are introduced in the next chapter. Currently, we apply our consistency check conservatively and tolerate remaining outliers that are not detected.

The resulting point cloud still contains redundancies, especially in overlap regions of surfaces reconstructed from neighboring input views. They can be removed using any of the available point cloud simplification [Pauly et al., 2002] or merging [Sadlo et al., 2005] methods.

Figure 5.5: Enforcing photo consistency during view merging: without (left) and with (right) enforcement.

5.4 Rendering

The uncertainties modeled by the Gaussian ellipsoids are used in our probabilistic renderer for generating smooth images of the scene from novel viewpoints. Our method combines the advantages of the probabilistic image generation approach described by Broadhurst et al. [2001] with those of visibility splatting [Ren et al., 2002]. Additionally we perform a view-dependent blending similar to method used by Hofsetz et al. [2005].

5.4.1 Probabilistic Rendering

Broadhurst et al. [2001] use probabilistic volume ray casting to generate smooth images. Each ray is intersected with the Gaussians of the scene model. At a specific intersection point $\mathbf{X}$ with the sample i, the evaluation $N_k(\zeta;\mathbf{X}_i;\mathrm{V}_i)$ of the Gaussian describes the probability that a ray hits the corresponding surface point. To compute the final pixel color, the algorithm employs the Bayes rule: It integrates all colors along each ray weighted by the probabilities without considering occlusions. Thus, the color c_R of a ray $\mathcal{R}$ is computed as

$$c_R = \frac{\int_{\zeta\in\mathcal{R}} \sum_i c_i N(\zeta;\mathbf{X}_i,\mathrm{V}_i)\partial\zeta}{\int_{\zeta\in\mathcal{R}} \sum_i N(\zeta;\mathbf{X}_i,\mathrm{V}_i)\partial\zeta}. \tag{5.14}$$

That approach produces very smooth images, but is incapable of handling occlusions and rendering solid surfaces in an opaque way.

Occlusions can be resolved via visibility splatting, as done in our point renderer of section 5.1.2. In combination with the final alpha normalization step, this method is able to generates crisp images containing opaque surfaces. On the other hand, it also sharply renders noise in the geometry.

Figure 5.6: Comparison of visibility splatting (left) and Bayesian rendering (center) with our approach (right).

We propose a rendering method that combines both approaches in order to benefit from their respective advantages. Our idea is to accumulate the colors along each ray like in the Bayesian setting, but to stop as soon as a maximum accumulated probability has been reached. Thus, we accumulate the solution of the integrals of equation (5.14) by traversing along the ray from the camera into the scene and stop as soon as the denominator reaches one. Assuming that solid surfaces are densely sampled, the probabilities are high enough so that the rays will stop within the front surface. This is still not guaranteed, but in practice, surfaces in the generated images appear much more opaque compared to the Bayesian renderings.

Such a renderer can be implemented by a modified version of our splatter. First, the Gaussians have to be presorted by the CPU according to their depths such that the GPU renderer splats them from front to back. As a consequence, no fuzzy depth test is needed for occlusion handling. The blending function is modified such that it stops accumulation when the accumulated values in the alpha buffer reach a level of saturation. This is directly supported by the OpenGL blending function GL_SRC_ALPHA_SATURATE.

We compare visibility splatting and Bayesian rendering with our approach on noisy data in figure 5.6. Notice the large distortions in the image generated by visibility splatting, which get smoothed out by the other two methods. However, the Bayesian renderer blends all the points including those from occluded surfaces, while our method renders opaque surfaces and keeps the blending. Thus, our renderer provides the advantages of both previous methods.

5.4.2 View-Dependent Blending

One specific sample usually looks most accurate from the view it has been acquired from. As the deviation between the acquisition and the virtual view be-

comes larger, the quality decreases. Thus, the contribution of a specific point to the generated image should be weighted by the deviation of its acquisition viewpoint from the virtual viewpoint. We achieve this by computing a weight for the alpha mask of each Gaussian using the view-dependent criterion of the unstructured lumigraph framework introduced by Buehler et al. [2001].

Because the camera of our acquisition system all have the same resolution and a similar distance from the recorded scene, we only use the angular penalty. If a different camera setup was used, implementation of the other penalties would be straightforward.

Assuming that the reconstructed scene looks best from the view of an acquisition camera and worse from other viewing angles, the algorithm compotes for each input view $v \in \{1,\ldots,m\}$ a penalty of the form

$$\omega_v'' = \arccos(\mathbf{R} \cdot \mathbf{R}_v), \tag{5.15}$$

where $\mathbf{R}$ and $\mathbf{R}_v$ are the viewing rays of the virtual and the acquisition cameras, respectively. For efficiency, we approximate those vectors with the viewing directions of the respective cameras: Given a camera rotation matrix R, they can be computed as $\mathbf{R} = \mathrm{R}^{\mathrm{T}} \cdot (0,0,1)^{\mathrm{T}}$. Instead of using the scalar product only as done by Hofsetz et al. [2005] and many other people, we found that our penalty which is linear to the deviation of viewing angles generates a smoother transition in the rendering when the viewing direction of the virtual camera is changed.

As suggested by Buehler et al. [2001], a blending weight for each input camera is then computed by

$$\omega_v' = \left(1 - \frac{\omega_v''}{\max_{u\in\{1,\ldots,m\}} \omega_u''}\right) \cdot \frac{1}{\omega_v''}. \tag{5.16}$$

This equation ensures epipole consistency: if an acquisition camera is identical with the virtual camera, its weight goes to infinity and dominates the weights of all other input cameras. For practical calculations, the weights have to be clamped to a large, finite number. Finally, they are normalized to

$$\omega_v = \frac{\omega_v'}{\sum_{u=1}^{m} \omega_u'}. \tag{5.17}$$

During splatting, weighted Gaussian screen-space kernels $\omega_v A_S(x,y)$ are used, yielding splats that are more transparent the more their acquisition camera view deviates from the rendering viewpoint.

5.5 Results and Discussion

The following results have been created using the taekwondo data presented in chapter 4 and appendix A. It has been recorded in our acquisition studio with three 3D video bricks covering an overall viewing angle of 61 degrees horizontally and 40 degrees vertically.

5.5.1 Image Quality

Figure 5.7 shows novel views of the scene generated from our reconstructed point-based 3D model. Our re-renderings have a decent look with a high-quality texture. Acquisition noise is smoothed out by our probabilistic data model in combination with the view-dependent rendering algorithms. This is clearly visible in comparison with a conventional point model based on surfels, as shown in figure 5.8.

Our depth map acquisition system based stereo vision and structured light projection together with our resolution-adaptive, irregularly sampled scene representation allows for capturing highly detailed textures and geometry. As shown in figure 5.9, we are able to reconstruct the texture details of the small-scale box on the table as well as the complex geometry of the tablecloth.

A remaining issue are outliers especially at object silhouettes which couldn't be removed by the photo consistency check. They are artifacts of the depth reconstruction process due to inaccurate stereo matches at depth discontinuities. Our time-coherent filtering and alpha matting approaches introduced in chapter 6 help to further improve image quality.

5.5.2 Scalability

With our system we are in principle able to acquire a large viewing range with a relatively low amount of cameras. To support increasingly large ranges, our system is scalable up to full spherical views. To fully cover 360 degrees in all dimensions about 10 to 12 3D video bricks are needed. Data of additional bricks is just added to the global scene model, as shown in figure 5.10. Note that this is in principle not constrained to convex views. Although overlaps in the geometry can help to improve the overall quality due to the photo consistency enforcement, they are not required as each brick reconstructs its own scene part independently.

A practical limitation is the size of the acquired data. For example, each 3D video frame of the taekwondo data set consists in average of 1.21 million points, which corresponds to a raw data size of 45 megabytes for point positions, covariance matrices and colors together. Although it can be reduced by simplification, this comes at the cost of resolution, which may be undesirable if the user should have the possibility of free camera control including closeup views. Therefore, we investigate appropriate compression algorithms in chapter 7, as well as out-of-core data management in chapter 8.

5.5.3 Editing Capabilities

Our view-independent data model provides possibilities for novel effects and 3D video editing. Due to its point-based structure we are able to employ any kind of available point processing algorithms [Gross and Pfister, 2007]. Once the three-dimensional information is available, selection and compositing issues become

Figure 5.7: Re-renderings of the 3D video from novel viewpoints.

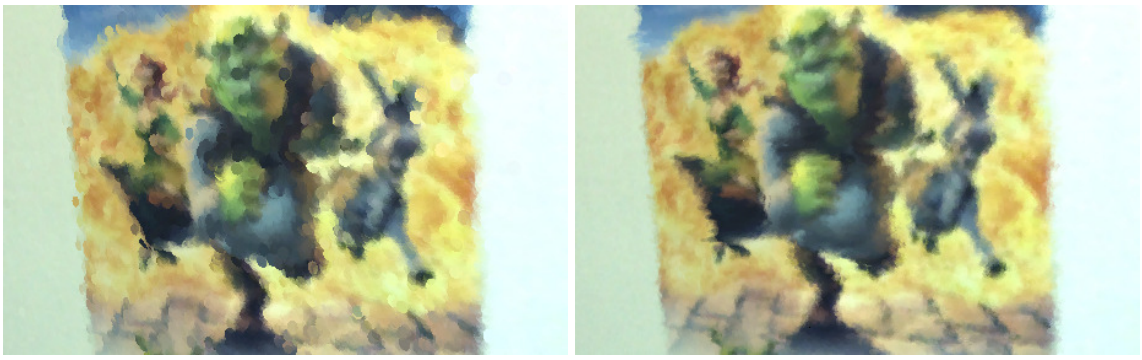


Figure 5.8: Rendering using surfels (left) and our view-dependent uncertainty blending (right).

Figure 5.9: Geometric detail in the tablecloth. For illustration we recomputed smooth surface normals and rendered the scene with Phong lighting under two different illumination conditions.

Figure 5.10: Scan merging: reconstructions from one (left), two (center) and three bricks (right).

Figure 5.11: Special effects: actor cloning (top), motion trails (bottom).

straightforward and can be easily implemented using spatial clustering or bounding box algorithms. Such tasks are much harder to achieve on both conventional 2D video and view-dependent 3D video approaches based on light fields or depth maps only. Apart from the well-known time freeze we show two example effects in figure 5.11. We clone the actor by copying its corresponding point cloud to other places in the scene. Motion trails are generated by compositing semi-transparent renderings of moving objects from previous time steps.

To further investigate novel possibilities of video post-production, we developed a 3D video editing system which is introduced in chapter 9. It provides a convenient user interface for navigating in the four-dimensional spatio-temporal data set, as well as a toolbox of manipulation operators such as semi-automatic graph cut segmentation for object selection.

5.6 Conclusions

Our point-based data model together with our scalable concept of modular 3D video acquisition bricks allows for capturing of a large viewing range with sparsely placed components. Decent-quality images of novel views are generated using Gaussian ellipsoid rendering with view-dependent blending methods.

The view-independent data representation is well suited for spatio-temporal video editing. It can directly benefit from a large variety of available point-based processing algorithms for simplification, compression, multiresolution rendering or out-of-core storage, which we show in the following chapters of this thesis.

The resulting image quality, however, still has room for improvement. Filtering the point cloud for smoothing or outlier removal remains difficult due to its irregular nature. Therefore, the next chapter introduces an alternative image-based representation that is able to significantly improve visual quality. It can either be used exclusively for re-rendering acquired depth and texture images or as an intermediate representation for efficient application of post-processing algorithms before conversion into a point-based model.

Chapter 6

3D Video Billboard Clouds

Although the previously introduced point-based model has its advantages in generality, scalability, and flexibility, post-processing of the raw scanned data for removing noise and outliers remains difficult. The irregular and three-dimensional nature of the point cloud requires complex error models, query data structures, and processing algorithms.

In this chapter we present an alternative representation called 3D video billboard clouds. They combine the generality of geometry-based representations with the regularization properties of image-based representations. As an extension to the original billboard clouds representation [Décoret et al., 2003], displacement-mapped billboard clouds (DMBBC) have been recently introduced by Mantler et al. [2007] as a new image-based rendering primitive. They represent parts of a geometrically complex scene as a set of proxy planes augmented with detailed displacement maps. We exploit DMBBCs for 3D video by constructing both their proxy planes and displacement maps from depth images of the scene. Those can be acquired using stereo matching algorithms, which are usually subject to noise and errors. The billboard planes with their approximate geometry are used to regularize this noisy, detailed geometry. By placing the billboard representation in the disparity space of the acquisition cameras, they provide a regular sampling of the scene with a uniform model of acquisition error. This allows to apply signal processing algorithms to generate smooth models with space and time coherence. The application of bilateral filters can successfully remove reconstruction and quantization noise as well as calibration errors and, thus, allows for higher quality renderings compared to those of reconstructions from complex and time-consuming subpixel stereo matching algorithms. Our GPU-accelerated rendering algorithm is able to further improve the final image quality by generating consistent view-dependent geometry and textures for each individual frame. To handle not only single objects with our representation, we also present a semi-automatic approach for modeling complete dynamic three-dimensional scenes

with a set of multiple 3D video billboards clouds.

With this framework, we are able to achieve a significantly higher image quality given the same input data. For 3D video editing, the billboard clouds can be easily converted into our point-based representation. The resulting 3D model benefits from the post-processing filters that have been applied in the image space domain of the billboards.

6.1 Data Model

A 3D video billboard cloud models a single 3D video object and comprises a collection of multiple 3D video billboards. A 3D video billboard represents the 3D structure and texture of an object at a specific point in time as observed from a single viewpoint. It consists of an arbitrarily placed and oriented texture mapped rectangle or proxy $\hat{\Pi}$ approximating the real geometry of the object. Its associated textures are a displacement map $\hat{\mathcal{D}}$ for adding fine scale geometric detail, a color map $\hat{\mathcal{C}}$ modeling the surface appearance, and an alpha map $\hat{\mathcal{A}}$ representing the object's boundary. The latter is employed for seamless blending with the background of the scene.

Let us first assume that the required input data to generate such a billboard is available. Figure 6.1 shows the input data consisting of color images, alpha mattes, and depth maps of a single object. Color images and alpha mattes directly serve as texture maps on the billboard planes. They can be recorded e.g. by standard cameras in front of a green screen or using segmentation and matting algorithms [Wang et al., 2005; Li et al., 2005]. In section 6.4 we propose a method to construct this data from multi-view recordings of real-world scenes with multiple objects. The depth maps can be reconstructed using e.g. our stereo vision algorithm of chapter 3. They are transformed into displacement maps for the billboards, which are placed in disparity space of the acquisition cameras as explained in section 6.1.1. Figure 6.2 illustrates the billboard planes and their composition to a billboard cloud, as well as displacement-mapped billboard planes.

We impose a set of requirements for an optimal 3D video billboard clouds representation:

1. **Simple geometric proxy.** The geometric proxy should be as simple as possible, i.e. a rectangle. This permits an easy parametrization for texture mapping.

2. **Regular sampling**. By ensuring a regular sampling we can exploit standard signal processing methods for post-processing of the geometry without the need of resampling. In particular, we would like to directly exploit the existing regular sampling from the acquisition cameras.

3. **Uniform error model.** 3D reconstruction introduces noise that is usually not uniform in world coordinates. The uncertainty of depth values recon-

Figure 6.1: Billboard input data: colors (left), alpha matte (middle), and depth map (right).

Figure 6.2: Illustration of the billboard cloud for one object: billboard plane from one input view (left), composition of planes from multiple input views to a billboard cloud (middle), displacement-mapped billboard plane from one input view (right).

structed by triangulation increases with their absolute value. Our billboard representation should be defined in a space where the reconstruction error is approximately uniform, independent from the distance of a surface from the camera. Thus, uniform, linear filters can be applied for smoothing the acquired geometry.

4. **Minimal displacements.** A minimal displacement of the proxy to the real surface ensures a good approximation of the geometry and can improve future compression and level-of-detail algorithms.

Requirement 1 can be guaranteed by construction. Requirements 2 and 3 are fulfilled by defining the billboards not in conventional 3D space of the scene but in the so-called disparity space of the acquisition camera. This is described in sec-

tion 6.1.1. Finally, a minimization algorithm introduced in section 6.1.2, ensures the last requirement.

6.1.1 Scene Sampling and Error Model

Consider an input depth map $\mathcal{Z} = \{(x_i, y_i, z_i)\}$ which has for each pixel at coordinates (x_i, y_i) a unique depth value z_i. The pixels are sampled on a uniform, regular grid. This is a representation of the scene in the ray space of an acquisition camera, as each pixel corresponds to a unique viewing ray. Assume a pinhole camera model with projection matrix P. Camera space coordinates (X_{Ci}, Y_{Ci}, Z_{Ci}) are projected into ray space by

$$\begin{pmatrix} z_i x_i \\ z_i y_i \\ z_i \end{pmatrix} = \mathrm{P} \cdot \begin{pmatrix} Z_{Ci} \\ Y_{Ci} \\ Z_{Ci} \end{pmatrix} \tag{6.1}$$

followed by the division by the homogeneous coordinate z_i. This is a nonlinear transform, i.e. linear functions in camera space are not linear in ray space anymore, as illustrated in figure 6.3, and vice versa. Hence, if we defined the billboard plane in ray space and used the depth values as displacements, it would not be planar in world coordinates and thus it would be difficult to use it as an approximation for the real geometry. On the other hand, if we placed it in camera space, the sampling would become irregular.

Instead, we define a disparity space of a camera as coordinates (x_i, y_i, d_i) with d_i being inversely proportional to z_i, i.e. $d_i \propto \frac{1}{z_i}$. Using this representation, we can observe that planes in disparity space stay planar in camera space (cf. figure 6.3). Moreover, sampling in disparity space is identical to the regular sampling of the acquisition cameras. Thus, requirement 2 is fulfilled if we define the billboard planes in these coordinates.

This representation is directly motivated from the fact that most 3D scanners based on triangulation do not compute depth maps but disparity maps $\mathcal{D} = \{(x_i, y_i, d_i)\}$. For example, our depth from stereo algorithm of chapter 3 without subpixel matching computes disparities as the difference of the x-coordinates of two corresponding pixels in two different, rectified camera images. Due to the spatial extent of the pixels, this produces a quantization error of σ_z which is constant for each sample. Also with subpixel matching, there is a remaining uncertainty in the disparity values. In camera space it can be observed that the resulting uncertainty of the geometry is not constant anymore but depending on the absolute value of the disparity. This is illustrated with the red error bars in figure 6.3. By defining the billboards in disparity space we can thus use a uniform model for the reconstruction error and fulfill requirement 3.

In conclusion, we obtain an image-space representation of a billboard using pixel disparities by modeling the plane

$$\hat{\Pi}(x, y) = \hat{\pi}_x \cdot x + \hat{\pi}_y \cdot y + \hat{\pi}_0 \tag{6.2}$$

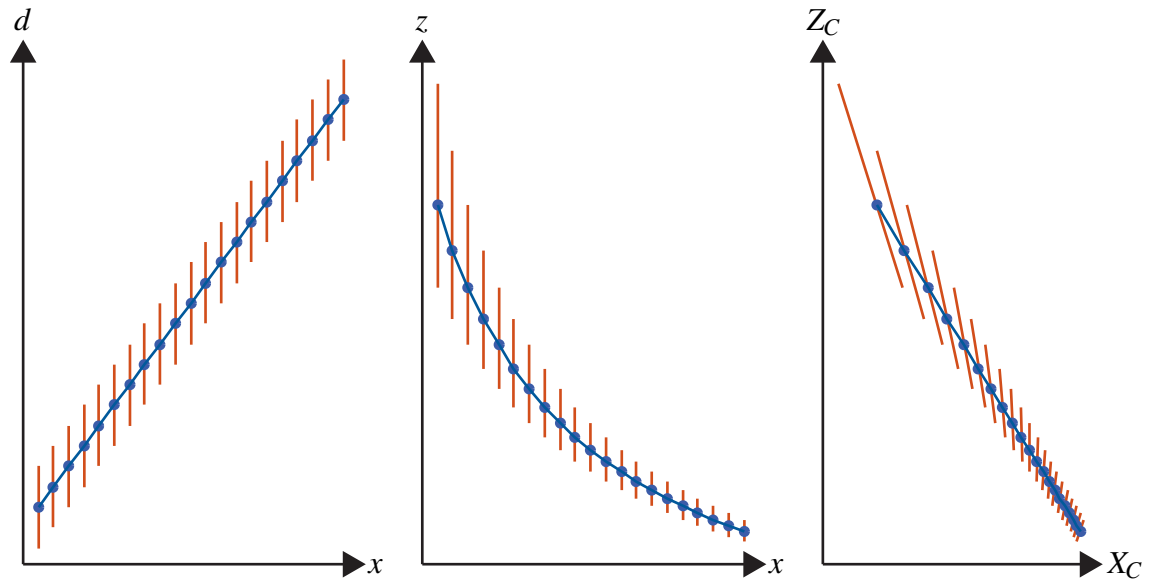


Figure 6.3: Illustration of the sampling of a plane (blue) and the triangulation error model (red) in disparity space (left), ray space (middle), and camera space (right).

as a linear scalar function over the pixel coordinates x and y, and the displacement map $\hat{\mathcal{D}} = \{(x_i, y_i, \hat{d}_i)\}$ with

$$\hat{d}_i = d_i - \hat{\Pi}(x_i, y_i) = \frac{1}{z_i} - \hat{\Pi}(x_i, y_i). \tag{6.3}$$

Using stereo vision as input, we can compute the displacements directly from the disparities d_i.

6.1.2 Optimal Billboard Placement

We are still free to choose the position $\hat{\pi}_0$ and orientation $\hat{\pi}_x$ and $\hat{\pi}_y$ of the billboard plane. A bad choice of these values can lead to arbitrarily large displacements in world coordinates. This becomes an important issue as soon as the values of the displacement map should be processed. E.g. by applying filters for improving the surface geometry, already very small errors due to numerical instabilities can become very large in world coordinates and produce large geometric artifacts. Another example is lossy compression of displacement maps, e.g. if they are stored as compressed textures in the GPU. While compression algorithms try to minimize artifacts appearing in the displacement maps it should also be ensured that visible artifacts on the actual surfaces also remain small. In these terms, we are looking for an optimal choice of the plane parameters.

Note that constructing a least-squares plane in disparity space by computing

$$\arg \min_{\hat{\pi}_x, \hat{\pi}_y, \hat{\pi}_0} \sum_i ||\hat{\pi}_x x_i + \hat{\pi}_y y_i + \hat{\pi}_0 - d_i||^2 \tag{6.4}$$

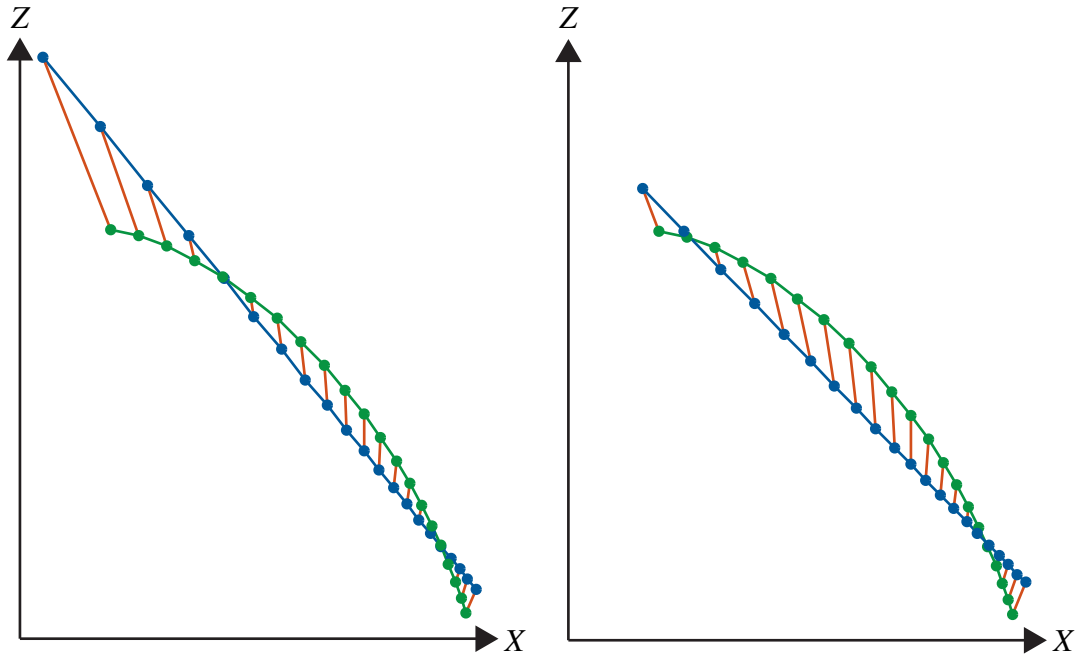


Figure 6.4: Illustration of surface (green), billboard plane (blue) and displacements (red) in world space. Left: displacements optimized in disparity space. Right: displacements optimized in world space.

is not sufficient. Although it minimizes the displacements in disparity space they can grow arbitrary large in world coordinates depending on the magnitude of the present disparities. A result of such a minimization in 2D is illustrated on the left side of figure 6.4. Instead the optimization has to be carried out directly in world coordinates by solving the nonlinear least squares problem

$$\arg\min_{\hat{\pi}_x,\hat{\pi}_y,\hat{\pi}_0} \sum_i \left\| \mathrm{P}^{-1} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \cdot \left(\frac{1}{\hat{\pi}_x x_i + \hat{\pi}_y y_i + \hat{\pi}_0} - \frac{1}{d_i} \right) \right\|^2 . \tag{6.5}$$

Usually, five to ten iterations of the Levenberg-Marquardt algorithm [Press et al., 1992] are sufficient for convergence to a relative error below 10^{-6}. The right side of figure 6.4 shows the result of such an optimization.

The billboard planes may be additionally stabilized over time by incorporating additional disparity space coordinates (x_i, y_i, d_i) from previous and successive frames into equation (6.5), weighted by their temporal distance from the current frame. However, our experiments have shown already a good temporal stability without this approach for our application. This can be explained with the temporal invariance of acquisition noise and the robustness of the least-squares fit against outliers. For different data sets that for example contain large clusters of outliers, temporal stabilization may be necessary.

6.2 Filtering Framework

The displacement values generated by the acquisition system are subject to quantization errors, noise, and calibration inaccuracies, resulting in several kinds of artifacts in the re-rendered image: The object surfaces do not appear smooth and their geometry is very noisy, which is especially visible as flickering over time. Moreover, overlapping parts of surfaces from different scanning directions do not necessarily fit to each other. To improve this, we apply a four-dimensional smoothing filter yielding better spatial coherence within surfaces and between overlapping surfaces, and better coherence over time. This complements the spacetime stereo matching algorithm of chapter 3 which applies a spatio-temporal filter on the correlation function in order to enhance local maxima before they are extracted. The method described here filters the locations of these maxima after extraction to smooth errors of the previous optimization step, as well as errors like camera misalignment, which stereo matching cannot handle.

Our displacement map representation has some nice properties allowing us to use standard signal processing tools to filter the geometry. The billboard plane serves as a parameter domain for scalar data sampled on a regular grid. Over time, it provides a parametrization along object trajectories because each billboard represents a best fit to the local geometry. Additionally, the invariant error model in disparity space allows for using a uniform filter kernel.

Our method is based on the bilateral filter [Tomasi and Manduchi, 1998], which is a well-established tool in image processing for smoothing noise while retaining sharp features. It can be shown [Elad, 2002] that it is based on the same Bayesian framework as traditional methods for additive noise removal such as anisotropic diffusion [Weickert, 1998]. But unlike those iterative approaches, the bilateral filter is a very simple single-pass algorithm with low computational effort, yielding similar results. Section 6.2.1 introduces the concept and our application to the displacement maps of the billboards. We call it intra-view filter because it processes the different input views separately. In section 6.2.2, we describe our extension to the so-called inter-view filter which processes all input views together to improve geometric consistency. Both approaches are finally compared in section 6.2.3.

6.2.1 Intra-View Filter

Let $\hat{\mathcal{D}}(\dot{\mathbf{x}})$ denote all displacements reconstructed from a single acquisition view, where $\dot{\mathbf{x}} = (x, y, t)$ contains the pixel coordinates x and y, as well as the acquisition time t. A general bilateral spatio-temporal filter that computes new displacements $\hat{\mathcal{D}}'$ is of the form

$$\hat{\mathcal{D}}'(\dot{\mathbf{x}}) = \frac{\int \hat{\mathcal{D}}(\zeta) \cdot k_D(\zeta, \dot{\mathbf{x}}) \cdot k_R\left(\hat{\mathcal{D}}(\zeta), \hat{\mathcal{D}}(\dot{\mathbf{x}})\right) \cdot \partial\zeta}{\int k_D(\zeta, \dot{\mathbf{x}}) \cdot k_R\left(\hat{\mathcal{D}}(\zeta), \hat{\mathcal{D}}(\dot{\mathbf{x}})\right) \cdot \partial\zeta}. \tag{6.6}$$

It is a combination of a domain filter with kernel k_D and a range filter with kernel k_R [Tomasi and Manduchi, 1998]. The denominator is a normalization term.

In our case, k_D is used to smooth the displacements over space and time while k_R retains geometric discontinuities. For that purpose, we employ for k_D a cubic b-spline low-pass filter kernel B, which is additionally weighted by the alpha values $\hat{\mathcal{A}}(\dot{\mathbf{x}})$ of the current billboard:

$$k_D(\zeta, \dot{\mathbf{x}}) = \hat{\mathcal{A}}(\zeta) \cdot B(\zeta - \dot{\mathbf{x}}). \tag{6.7}$$

This alpha weight ensures that the filter only accumulates points belonging to the current billboard. Moreover, it provides a local extension to our uniform error model by considering points at the surface boundary as less important, because they are likely to be more inaccurate. The range filter kernel k_R is a simple step function

$$k_R(\hat{d}_1, \hat{d}_2) = \begin{cases} 1 & \text{if } |\hat{d}_1 - \hat{d}_2| \leq \Delta d \\ 0 & \text{if } |\hat{d}_1 - \hat{d}_2| > \Delta d \end{cases} \tag{6.8}$$

which maintains discontinuities that are larger than a user-defined disparity threshold Δd.

As we are working with sampled data, the filter can be discretized. For ease of notation, let us omit the temporal and one spatial domain and consider only the case of one-dimensional displacement and alpha maps $\hat{\mathcal{D}}(x)$ and $\hat{\mathcal{A}}(x)$. Let those images be sampled at discrete positions x_i, yielding corresponding pixel values $\hat{d}_i$ and $\hat{\alpha}_i$ for $i \in \{1, \ldots, n\}$. After inserting equation (6.7) into (6.6), the filter can now be written as

$$\hat{d}'_i = \frac{\sum_{j=1}^{n} \hat{d}_j \cdot \hat{\alpha}_j \cdot B(x_j - x_i) \cdot k_R(\hat{d}_j, \hat{d}_i)}{\sum_{j=1}^{n} \hat{\alpha}_j \cdot B(x_j - x_i) \cdot k_R(\hat{d}_j, \hat{d}_i)}. \tag{6.9}$$

Because the image sampling positions are on a regular grid, we can further set $x_i = i$ and write

$$\hat{d}'_i = \frac{\sum_{j=1}^{n} \hat{d}_j \cdot \hat{\alpha}_j \cdot B_{j-i} \cdot k_R(\hat{d}_j, \hat{d}_i)}{\sum_{j=1}^{n} \hat{\alpha}_j \cdot B_{j-i} \cdot k_R(\hat{d}_j, \hat{d}_i)}. \tag{6.10}$$

There, we also discretized the b-spline kernel to $B_i := B(i) = B(x_i)$, which expands to an efficient separable filter matrix in the multi-dimensional case. Such regularity is, however, not given in our following extension, which therefore builds upon equation (6.9).

6.2.2 Inter-View Filter

When filtering all acquisition views independently, problems can arise in regions of overlapping geometry coming from different billboards. The filtering result may diverge such that the distinct surface patches will not fit to each other. Thus,

we extend the filter to an additional domain providing inter-view coherence information.

Let us consider again the one-dimensional case similar to equation (6.9). To distinguish the different views, we use an upper index in our following notation. Thus, $\hat{d}_i^v$ and $\hat{\alpha}_i^v$ denote the discrete pixel values of the displacement and alpha maps belonging to the billboard that has been generated from a specific view $v \in \{1,\ldots,m\}$. Those pixels are sampled at discrete positions x_i^v.

The filter is still applied successively to each single view. But before filtering a specific view v, the unfiltered disparities from all other views u are projected into v by using depth image warping: From view u, each pixel with position x_i^u and displacement $\hat{d}_i^u$ is first reconstructed in three-dimensional world coordinates and subsequently projected into v, yielding a new pixel there, with image-space position $x_i^{u \triangleright v}$ and displacement $\hat{d}_i^{u \triangleright v}$. With $u \triangleright v$ we denote the projection operator from u to v. Note that the pixel positions $x_i^{u \triangleright v}$ are irregular: They are coordinates on the image plane of view v, but they are not aligned to the pixel grid.

Now, the displacements $\hat{d}_i^v$ can be filtered, taking into account all other displacements $\hat{d}_i^{u \triangleright v}$ by extending equation (6.9) to

$$\hat{d}_i'^v = \frac{\sum_{u=1}^{m}\sum_{j=1}^{n} \hat{d}_j^{u \triangleright v} \cdot \hat{\alpha}_j^u \cdot B(x_j^{u \triangleright v} - x_i^v) \cdot k_R(\hat{d}_j^{u \triangleright v}, \hat{d}_i^v)}{\sum_{u=1}^{m}\sum_{j=1}^{n} \hat{\alpha}_j^u \cdot B(x_j^{u \triangleright v} - x_i^v) \cdot k_R(\hat{d}_j^{u \triangleright v}, \hat{d}_i^v)} \tag{6.11}$$

$$= \frac{\sum_{u=1}^{m}\sum_{j=1}^{n} \hat{d}_j^{u \triangleright v} \cdot \hat{\alpha}_j^u \cdot B(x_j^{u \triangleright v} - i) \cdot k_R(\hat{d}_j^{u \triangleright v}, \hat{d}_i^v)}{\sum_{u=1}^{m}\sum_{j=1}^{n} \hat{\alpha}_j^u \cdot B(x_j^{u \triangleright v} - i) \cdot k_R(\hat{d}_j^{u \triangleright v}, \hat{d}_i^v)}. \tag{6.12}$$

There, the range filter kernel k_R does not only maintain discontinuities but also ensures a correct handling of occlusions that occur during projection.

In contrast to the previous filter, the b-spline kernel cannot be discretized. But nonetheless, the filter can be implemented very efficiently via splatting, similar to our point renderer described in chapter 5. The algorithm is provided in pseudocode notation in figure 6.5. It maintains two pixel buffers $\{p_i\}$ and $\{q_i\}$ for the numerator and denominator of equation (6.12), which are initialized in line 1. All billboard pixels are successively projected into the current view v (line 4) and splatted (lines 5 to 9). In the real implementation, rasterization is only performed in the finite support region of the b-spline kernel. Finally, the fraction is computed in line 12.

Because the projection itself depends on the unfiltered displacements, it may be necessary to apply the filter iteratively. In each iteration, all views are filtered successively, using the displacements from the previous iteration for the projection. However, because the b-spline filter diminishes all frequencies to a certain extent, all displacements belonging to a continuous surface patch would converge to a smooth geometry after an infinite amount of iterations. Only larger discontinuities would be retained by the range filter kernel. Thus, the number of iterations has to be fixed beforehand and the size of the b-spline has to be adjusted appropriately. With our data sets, more than one iteration did not show a visible effect in the rendered images.

```
1  for i ∈ {1,...,n} do p_i ← 0, q_i ← 0                    ▷ Numerator & denominator
2  for u ∈ {1,...,m} do
3      for j ∈ {1,...,n} do
4          compute x_j^{u▷v} and d̂_j^{u▷v}                    ▷ Projection
5          for i ∈ {1,...,n} do                              ▷ Splat rasterization
6              a ← α̂_j^u · B(x_j^{u▷v} − i) · k_R(d̂_j^{u▷v}, d̂_i^v)   ▷ Screen-space kernel
7              p_i ← p_i + d̂_j^{u▷v} · a                      ▷ Accumulate numerator
8              q_i ← q_i + a                                 ▷ Accumulate denominator
9          end
10     end
11 end
12 for i ∈ {1,...,n} do d̂_i^{▷v} ← p_i / q_i                  ▷ Normalization
```

Figure 6.5: Splatting algorithm for inter-view filter (1D example).

6.2.3 Comparison of Filters

In figure 6.6, both the intra-view and the inter-view filter have been applied for comparison. The size of the b-spline kernel has been chosen as 21×21 pixels in image space and 5 frames over time. The raw displacements were calculated from disparity maps generated by our window-based stereo algorithm of chapter 3 without subpixel estimation. In contrast to subpixel stereo, which can also generate smooth displacements, our second filter can additionally correct for calibration errors. It can even outperform our subpixel stereo matching as shown in section 6.5.

6.3 View-Dependent Rendering and Blending

The billboards are directly rendered from the disparity space representation. Transformation into world coordinates is done during image generation in the GPU. We implemented a simple displacement mapping technique that stores a tessellated plane as vertex array and uses the CPU to set the z-coordinates of all vertices to the disparities. There also exist displacement mapping algorithms for the GPU [Donnelly, 2005; Mantler et al., 2007] that can be directly integrated in our framework.

Consistent images from multiple billboards from different views are generated by our view-dependent rendering approach. Each billboard from a view v gets an assigned weight ω_v according to the unstructured lumigraph framework [Buehler et al., 2001] presented in section 5.4.2, based on the orientation of its correspond-

Figure 6.6: Comparison of displacement filtering methods. Far left: all displacements set to zero. Middle left: unfiltered displacements. Middle right: displacements smoothed using our intra-view filter. Far right: displacements smoothed using our inter-view filter.

ing acquisition camera and of the current virtual camera. Thus, billboards closer to the current view have a larger impact on the image generation process.

In contrast to the original approach, our method does not only blend the colors but first reconstructs a consistent, view-dependent geometry of the scene, where each pixel has a uniquely assigned depth value. The procedure is illustrated in the rightmost part of figure 6.7.

If multiple fragments are rendered at the same pixel, a new depth buffer value z_S is computed in a fragment program by averaging all individual fragment depths z_{Si}:

$$z_S = \frac{1}{\sum_i \omega_i \hat{\alpha}_i} \sum_i \omega_i \hat{\alpha}_i z_{Si}. \tag{6.13}$$

The depths are additionally weighted with the values $\hat{\alpha}_i$ from the alpha matte of the billboard because they are likely to be more inaccurate at the billboard boundary.

The pixel color c_S is assigned afterwards according to the new depth, using projective texturing. It is determined by view-dependent blending all incoming texture values c_{Si} using

$$c_S = \frac{1}{\sum_i \omega_i \hat{\alpha}_i} \sum_i \omega_i \hat{\alpha}_i c_{Si}. \tag{6.14}$$

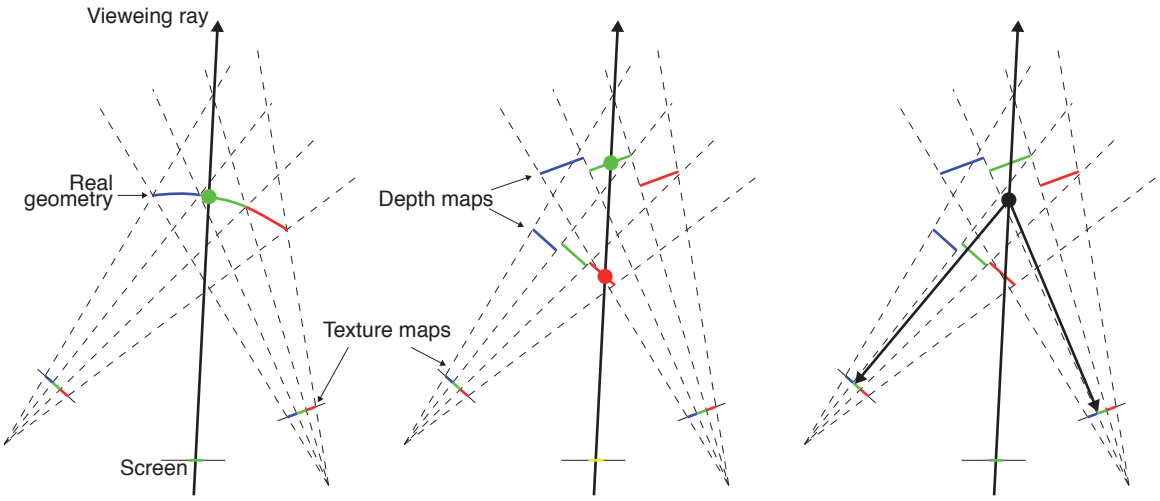


Figure 6.7: Illustration of rendering a surface patch textured with a red-green-blue pattern. While ray-casting the real geometry (left) always yields a correct result, view-dependent color blending (middle) may mix inconsistent texture values due to inaccuracies in the depth maps. By first constructing a consistent view-dependent depth value per screen pixel with successive view-dependent texturing, our approach (right) tends to better reproduce the original colors.

Furthermore, we compute the alpha value of the pixel as

$$\alpha_S = \max_i \alpha_{S_i} \tag{6.15}$$

to ensure that the transparencies of the alpha matte are maintained such that the billboard cloud still smoothly blends with the background.

Due to blending, occlusions cannot be handled using the conventional z-buffer algorithm. Instead of doing an expensive back-to-front rendering we implemented a fuzzy z-buffer via visibility splatting [Ren et al., 2002], similarly to the point renderer of chapter 5.

Compared to conventional unstructured lumigraph rendering, our approach tends to better reproduce the original textures by first generating a consistent, view-dependent depth map. As can bee seen in the comparison of figure 6.8, this results in much crisper images with less ghosting artifacts.

6.4 Handling Scenes

To model complete scenes, each view has to be decomposed into multiple billboards. We use a semi-automatic video cutout technique to segment the input videos into distinct objects. The segment boundaries are refined by a Bayesian matting algorithm yielding alpha mattes for the billboards.

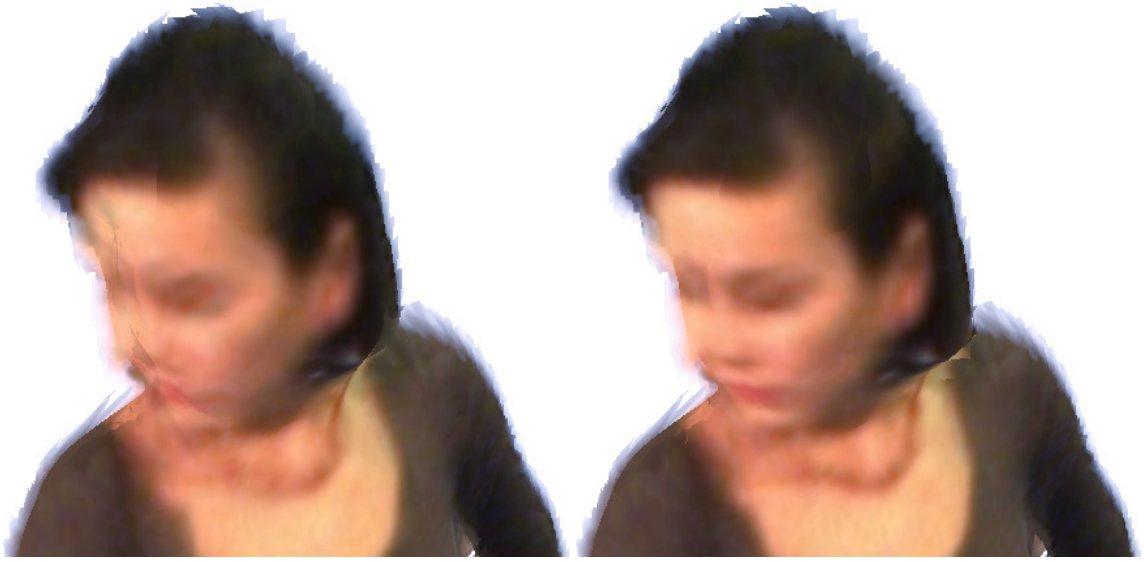


Figure 6.8: Comparison between unstructured lumigraph blending of colors only (left), and colors and depths (right).

6.4.1 Segmentation

Segmentation of the video is done interactively. The user marks objects in a single input frame of each view by applying a few brush strokes. A graph cut optimization then automatically computes the segments for each object over time. We extend the video cutout method by Wang et al. [2005] by including also the available depth values into the graph cut segmentation. This makes the minimization more robust and needs less user input. We now describe the segmentation of one single object. Segmentation of multiple objects can be achieved by running the algorithm multiple times on the same input, each time considering a different brush stroke as a mark of the object of interest.

For the further considerations, we call the image region containing the object of interest the *foreground* and the remaining region the *background*. To segment foreground from background, the algorithm minimizes an energy function

$$\arg\min_{\mathcal{A}} E_A(\mathcal{A},\mathcal{C},\mathcal{Z},\Theta_F,\Theta_B,\mathcal{R}_F,\mathcal{R}_B,\mathcal{R}_U), \tag{6.16}$$

which we denote shorter as $\arg\min_{\mathcal{A}} E(\mathcal{A})$. For a vector of input pixel colors $\mathcal{C} = \{c_1,\ldots,c_n\}$ and depths $\mathcal{Z} = \{z_1,\ldots,z_n\}$, it tries to find a binary labeling $\mathcal{A} = \{\alpha_1,\ldots,\alpha_n\} \in \{0,1\}^n$ such that all pixels i belonging to the foreground are labeled with $\alpha_i = 1$ and all remaining pixels with $\alpha_i = 0$. Note that the depths $\mathcal{Z}$ are not included in the originally proposed algorithm. The minimization is guided by three disjoint image regions $\mathcal{R}_i \subset \{1,\ldots,n\}$ with $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for $i \neq j$, $i,j \in \{F,B,U\}$, and $\mathcal{R}_F \cup \mathcal{R}_B \cup \mathcal{R}_U = \{1,\ldots,n\}$. $\mathcal{R}_F$ and $\mathcal{R}_B$ contain all pixels that have been marked by the user as belonging to the foreground and background, respectively. $\mathcal{R}_U$ contains all remaining pixels whose labeling is still unknown.

Furthermore, some prior statistical models Θ_F and Θ_B of the foreground and background are used that are automatically generated from the marked pixels.

The energy can be split into a weighted sum of two parts:

$$E(\mathcal{A}) = \lambda_D E_D(\mathcal{A}) + E_S(\mathcal{A}). \tag{6.17}$$

$E_D(\mathcal{A})$ is the so-called data energy that measures the compliance of the proposed segmentation with the foreground and background models. The smoothness energy $E_S(\mathcal{A})$ tries to keep the segmentation smooth in space and time.

Data Energy

The data energy measures how well the proposed segmentation $\mathcal{A}$ complies with the user marks by summing up energies for each pixel:

$$E_D = \sum_{i=1}^{n} D_i(\alpha_i) \tag{6.18}$$

with $D_i(\alpha_i)$ defined as follows:

$D_i(\alpha_i)$	$i \in \mathcal{R}_F$	$i \in \mathcal{R}_B$	$i \in \mathcal{R}_U$
$\alpha_i = 1$	0	∞	$\frac{\Theta_F(c_i,z_i)}{\Theta_F(c_i,z_i)+\Theta_B(c_i,z_i)}$
$\alpha_i = 0$	∞	0	$\frac{\Theta_B(c_i,z_i)}{\Theta_F(c_i,z_i)+\Theta_B(c_i,z_i)}$

(6.19)

In the marked regions, the pixel labels are already known and the energy values are assigned appropriately. For pixels in the unknown region the energies evaluate their compliance with the respective model Θ_F or Θ_B. As in the paper by Rother et al. [2004] they are normalized for determining the final energy.

In contrast to previous implementations of image or video cutout algorithms, our method incorporates a model $\Theta_{F,Z}$ of the available pixel depths in addition to a color model $\Theta_{F,C}$. Both prior models are built separately and combined in a weighted sum by letting

$$\Theta_F(c_i,z_i) = (1-\lambda_Z)\Theta_{F,C}(c_i) + \lambda_Z \Theta_{F,Z}(z_i) \tag{6.20}$$

and

$$\Theta_B(c_i,z_i) = (1-\lambda_Z)\Theta_{B,C}(c_i) + \lambda_Z \Theta_{B,Z}(z_i) \tag{6.21}$$

Each component is the negative logarithmic likelihood of a Gaussian mixture model [McLachlan and Basford, 1988] describing the probability if the current pixel fits to the respective foreground or background model. For example, the model of foreground colors is defined as

$$\Theta_{F,C}(c) = -\log \sum_{j=1}^{k} \nu_j N_3(c; \mu_{F,C_j}, V_{F,C_j}), \tag{6.22}$$

where $N_3(c;\mu_{F,C_j},\mathrm{V}_{F,C_j})$ denotes a Gaussian normal distribution in three-dimensional color space with mean value μ_{F,C_j} and covariance matrix V_{F,C_j}. They are fitted to the color distribution of the marked foreground pixels using k-means clustering [Jain and Dubes, 1988] followed by a principal component analysis [Keinosuke, 1990]. The number k of clusters is a user-defined value, commonly between five and ten. The convex weights ν_j are assigned proportionally to the number of pixels contributing to the individual clusters, such that $\nu_j > 0$ and $\sum_{i=1}^{k} \nu_j = 1$. The Gaussian mixture models of depths are built in a similar fashion. In contrast to the color models, they are only one-dimensional.

Smoothness Energy

The smoothness energy ensures that the segmentation follows object boundaries in image space and that it is smooth in time. Because the available depth maps do not contain pixel-exact representations of object boundaries, the energy equation is based on colors only.

The energy term is based on the assumption that neighboring pixels of similar colors tend to belong to the same image segment. Thus, it penalizes neighboring pixel pairs having a similar color but a differing assigned labeling. It considers two kinds of neighborhoods: intra-frame links $(i,j) \in \mathcal{N}_I$ between neighboring pairs of pixels within the same image for ensuring spatial smoothness, and inter-frame links $(i,j) \in \mathcal{N}_T$ between pairs of pixels over successive frames for temporal smoothness. The overall energy is a weighted sum of both types:

$$E_S(\mathcal{A}) = \lambda_I \sum_{(i,j)\in\mathcal{N}_I} (1-\delta_{\alpha_i,\alpha_j}) S_{i,j} + \lambda_T \sum_{(i,j)\in\mathcal{N}_T} (1-\delta_{\alpha_i,\alpha_j}) S_{i,j} \tag{6.23}$$

with

$$\delta_{a,b} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}. \tag{6.24}$$

For $S_{i,j}$ we use an exponential function based on the Euclidean distance of colors according to Boykov and Jolly [2001]:

$$S_{i,j} = e^{-\beta \|c_i - c_j\|^2}. \tag{6.25}$$

The scaling factor β can be precomputed from the input images as explained in that paper.

Graph Cut Optimization

As has been shown by Kolmogorov and Zabih [2002], equation (6.17) can be efficiently solved using graph cut optimization. The image pixels $\{1,\ldots,n\}$ build vertices of a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Additionally, two so-called terminal nodes T_F and

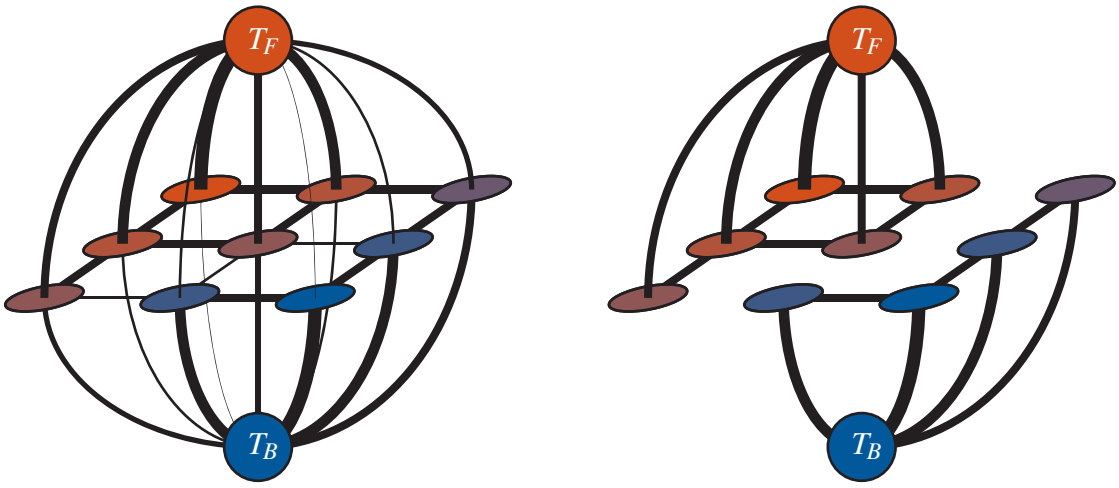


Figure 6.9: Graph cut segmentation of a 9×9 pixel image into foreground (red) and background (blue) regions. Left: Complete graph with pixel and terminal nodes, and smoothness and terminal edges. Edge costs are illustrated by line widths. Right: Graph after computing the optimal cut. The two separate subgraphs correspond to the resulting segmentation.

T_B are added, hence $\mathcal{V} = \{1, \ldots, n, T_F, T_B\}$. Edges are inserted between all neighboring pixels both in space and time, and additionally between each pixel and each of the two terminal nodes. This is illustrated in figure 6.9 for a single image only without temporal edges for the sake of clarity. Edge weights are assigned according to the following table:

Edge	Weight
$(i,j) \in \mathcal{N}_I$	$\lambda_I S_{i,j}$
$(i,j) \in \mathcal{N}_T$	$\lambda_T S_{i,j}$
(T_B, i)	$\lambda_D D_i(1)$
(T_F, i)	$\lambda_D D_i(0)$

Next, a minimum cut $\mathcal{E}_C \subseteq \mathcal{E}$ through the graph is computed such that in the new graph $\mathcal{G}_C = \langle \mathcal{V}, \mathcal{E} \setminus \mathcal{E}_C \rangle$ the terminal nodes T_F and T_B become separated and the sum of weights of all edges in $\mathcal{E}_C$ is minimal. This can be solved using the maximum flow algorithm [Boykov and Kolmogorov, 2004]. In the resulting graph, each pixel vertex is connected to exactly one terminal node. This can be interpreted as the optimal solution of the minimization problem in equation (6.17): A pixel i connected with terminal node T_F obtains a foreground label $\alpha_i = 1$, a pixel i connected with T_B obtains a background label $\alpha_i = 0$.

We further speed up the optimization by a two-pass process similar to method by Li et al. [2004]. Initially, an over-segmentation of all color images is computed in an offline process and stored together with the original videos. A first graph cut optimization is then run on the coarse segmentation. Due to its irregularity, smoothness links are introduced between each segment and its k nearest

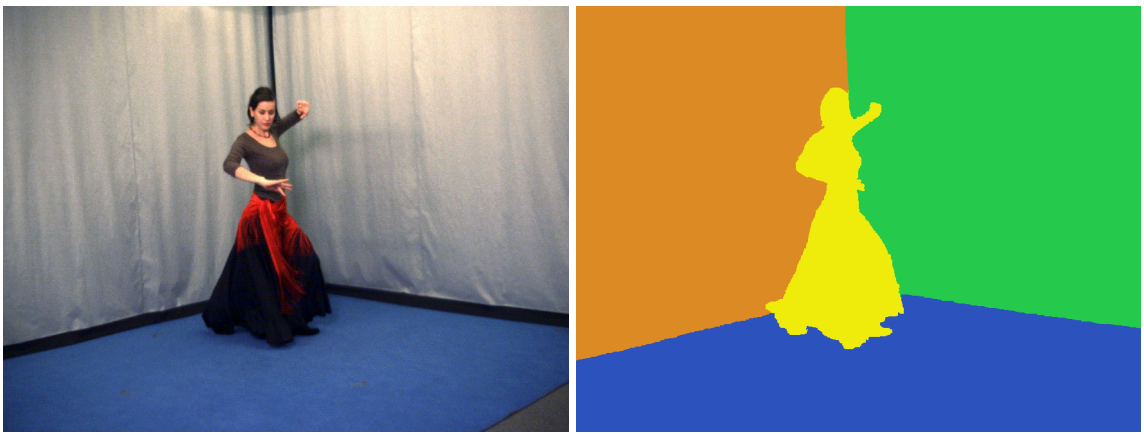


Figure 6.10: Illustration of a segmentation (right) of a scene (left) into four billboard clouds.

Figure 6.11: Graph cut segmentation using the input image with user markings on the left. Results of the optimization considering colors only (middle) and additional depths (right).

neighbors. The result is refined up to pixel level by running a second graph cut.

Segmentation Results

As result, the user can quickly generate a segmentation of a complete scene as illustrated in figure 6.10 with a few brush strokes. As shown in figure 6.11, inclusion of available depths increases the overall stability of the optimization and requires fewer input brush strokes from the user.

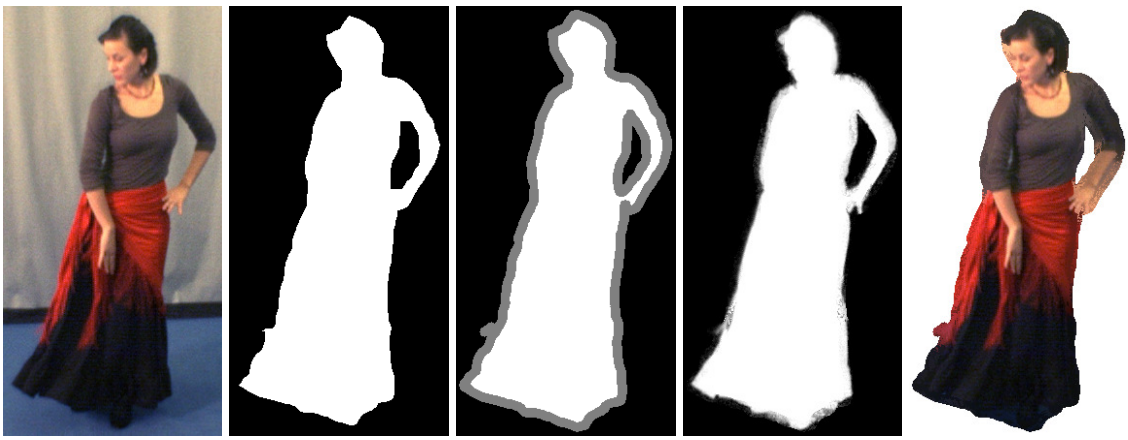


Figure 6.12: Object segmentation and matting. From left to right: original image, graph cut segmentation, graymap after dilation and erosion, alpha matte, foreground colors.

6.4.2 Matting

Pixels on the segment boundary typically cannot be assigned completely to either foreground or background. Their color is a mixture of both, i.e.

$$c = \alpha c_F + (1-\alpha)c_B, \quad 0 \leq \alpha \leq 1. \tag{6.26}$$

Matting algorithms can separate them again into foreground color c_F, background color c_B, and alpha weight α.

To solve that problem, we employ Bayesian matting [Chuang et al., 2001] which achieves robust results in an efficient way. As input it needs, similarly to the segmentation procedure, a color image and an initial segmentation into a known foreground region $\mathcal{R}_F$, a known background region $\mathcal{R}_B$, and an unknown region $\mathcal{R}_U$. Equation (6.26) is then solved for all pixels in $\mathcal{R}_U$. For stability, the unknown region has to be considerably smaller than in the segmentation problem. Thus, the regions are usually defined by a so-called graymap: a grayscale image using black, gray, and white pixel colors for $\mathcal{R}_B$, $\mathcal{R}_U$, and $\mathcal{R}_F$, respectively.

For each video object, we generate the graymap automatically by dilating and eroding the previously computed segmentation. The matting algorithm then computes an accurate alpha matte and foreground colors of the object, as shown in figure 6.12.

6.5 Results

For evaluating our 3D video billboard cloud representation we used the taekwondo and flamenco data sets presented in chapter 4 and appendix A. As input, we used depth maps computed by the stereo vision algorithm of chapter 3 without the

subpixel estimation stage. Instead, our bilateral filter was used to obtain accurate and smooth geometry.

Figures 6.13 and 6.14 show both sequences rendered from novel views. In comparison to previous approaches, we achieve better time coherence due to the spatio-temporal filtering. Furthermore, the alpha textures of the billboards smooth the appearance at object silhouettes and nicely blend the different segments of the scene.

The proxy geometry of the billboards introduces the possibility of manual user control. The flamenco data set is composed of four billboard clouds: one for the actor, two for both walls and one for the floor. For the background billboard, the proxy planes are already a very good approximation to the real geometry. Thus, all disparities can be set to zero. In the taekwondo data set the background is much more complex. Nonetheless, because the background billboards are known to have a static geometry, they can be stabilized by accumulating their displacements over time. This can be achieved by configuring the bilateral domain filter kernel to have a very long temporal extent. If manual intervention is not desired, one could also use automatic background estimation methods such as those described by Long and Yang [1990] or Colombari et al. [2006] on the input color videos to identify static scene parts.

In our application, our filtering framework can also be used as an alternative to the time-consuming subpixel estimation process of our stereo reconstruction algorithm. In the left part of figure 6.15 the disparity maps for the four billboard planes have been generated with the full pipeline of chapter 3, including subpixel matching. In contrast, the right part of figure 6.15 has been reconstructed with just the simple window-based stereo algorithm without subpixel matching, followed by bilateral filtering. Especially at surface boundaries, the filter shows a better numerical stability than the complex minimization algorithm of the subpixel matching, resulting in fewer outliers. On the other hand, the filter introduces more smoothing and discards some small geometric detail, but the overall visual appearance is still increased. Moreover, there is the temporal advantage of 1.5 minutes for filtering versus 28 minutes for the subpixel method. In principle, the filtering approach can be accelerated to real-time by using available GPU implementations for window-based stereo [Yang et al., 2006] and splatting [Botsch et al., 2005].

A current limitation of our method is shown figure 6.16. Thin, fast moving structures in the video like the arms or legs of an actor are difficult to handle. In such cases, the domain kernel of the bilateral filter does not have enough support by the data, neither in space nor in time. As a result, smoothing of those structures is not as good as in other surface regions. This may be improved in the future by using optical flow [Barron et al., 1994] or scene flow [Vedula et al., 1999] algorithms to obtain better temporal coherence.

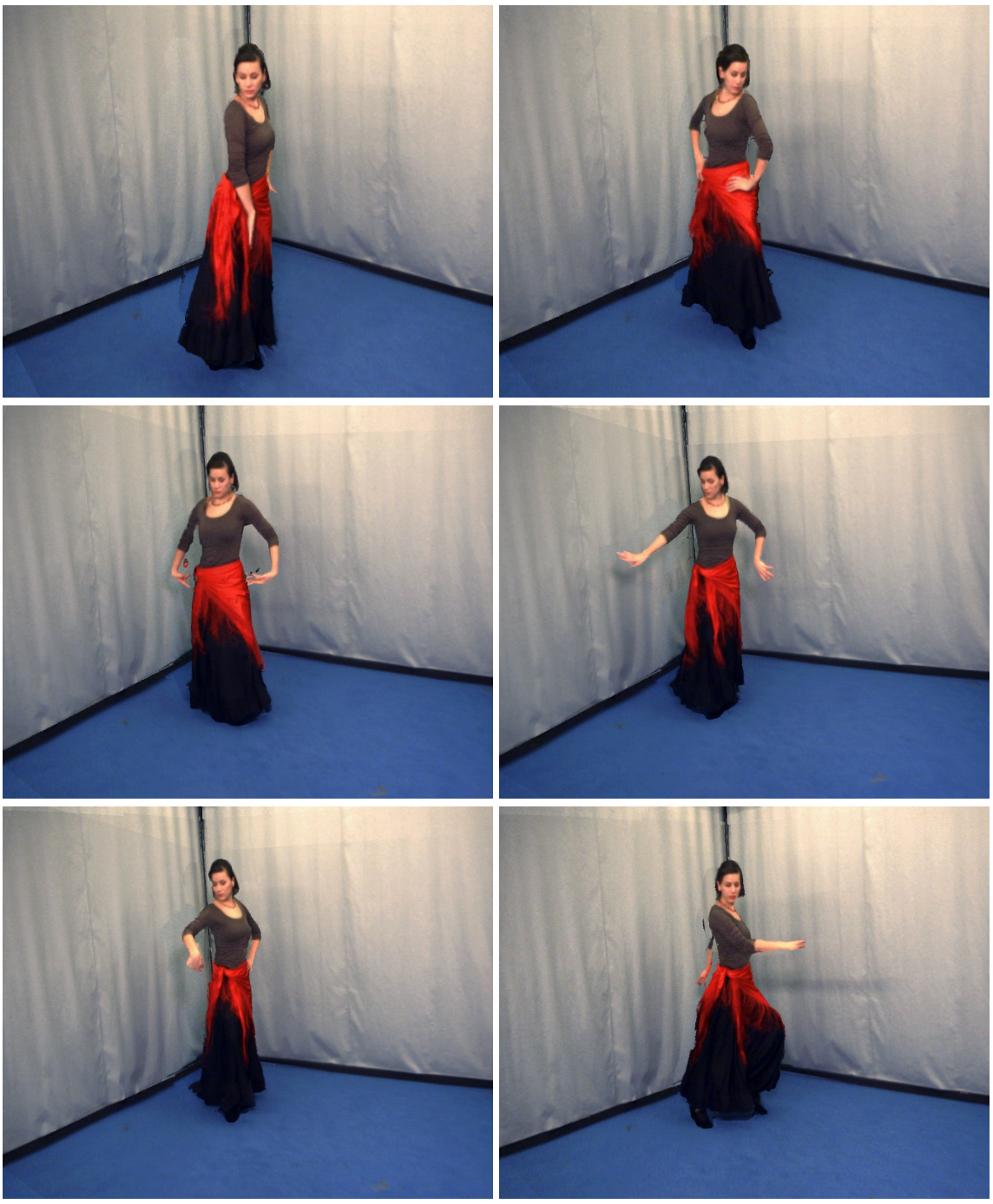


Figure 6.13: Image sequence of the flamenco data set rendered from novel views.

Figure 6.14: Image sequence of the taekwondo data set rendered from novel views.

Figure 6.15: Subpixel stereo versus bilateral filtering. Left: displacement maps generated by subpixel stereo matching without filtering. Right: displacement maps generated by simple stereo matching with bilateral filtering.

Figure 6.16: Thin, fast moving structures like the actor's arms cannot provide sufficient spatio-temporal support for high-quality filtering.

6.6 Comparison with Point-Sampled 3D Video

In table 6.1, we compare the 3D video billboard clouds with our point-sampled representation of chapter 5 using the following criteria:

- **Image quality.** Figure 6.17 directly compares the visual quality of both representations, using re-renderings of the scene from a novel viewpoint. There, billboards achieve a better image quality than points. The main reason for that is the decoupling of geometry and color in the billboard ren-

	Image quality	Scene complexity	Geometry filtering	Streaming & compression	Editing
Points	○	+	○	+	+
Billboards	+	○	+	+	−

Table 6.1: Comparison of point-sampled 3D video and 3D video billboard clouds (+ good, ○ average, − bad).

dering process. Because points have individually associated colors, noise in the geometry directly influences the texture. The projective texturing used for billboard rendering is less sensitive for such noise. For example, a geometric error that is exactly parallel to the image plane of the virtual camera would still produce correct view-dependent pixel depths and, thus, would not influence the texture at all. In principle, a similar approach could be implemented for point rendering, too, by splatting only pixel depths in a first pass and projecting the textures afterwards. Another reason for the superior image quality besides rendering is the different data processing pipeline. As discussed below, billboards support better methods for filtering reconstruction noise.

- **Scene complexity.** Because points make no assumptions about geometry and topology, they support in principle arbitrary scenes. Billboards, on the other hand, need a segmentation of the scene into distinct objects. Because the segmentation algorithm currently requires manual intervention, complex scenes with many moving objects are difficult to model. But there is still room for improvement. The segmentation algorithm could for example be guided by the input depths which contain strong hints about object positions.

- **Geometry filtering.** Being an image space representation, billboards naturally support a wide range of signal processing tools for post-processing the raw depth maps. Our bilateral filter is able to efficiently smooth reconstruction noise and errors. Powerful filtering algorithms exist for points, too [Pauly and Gross, 2001; Weyrich et al., 2004], but they are more complex because they have to deal with irregular sampling and the lack of local topology information.

- **Streaming & compression.** Both representations are very well suited for streaming and compression of 3D video. Both have the advantage of being a uniform storage container for geometry and appearance, which makes it easy to perform coding within a single framework. Because points do not store topology information, streaming is straightforward. Many point cloud compression algorithms are currently being developed. We introduce one in chapter 7. For compressing billboards, standard 2D video codecs can be used [Würmlin et al., 2005]. They can be in principle applied to the

Figure 6.17: Comparison of the visual quality of the point-based representation of chapter 5 (left) with our method based on 3D video billboards (right).

displacement maps, too, but their performance in that case still has to be examined, because they are originally tailored to natural color images.

- **Editing.** Being a view-independent representation, points are very well suited for 3D video editing. Their irregularity provides much flexibility for geometric deformations. Hence, we introduce a 3D video editor based on points in chapter 9. On the other hand, editing using the billboard representation would be difficult. Because billboards fix the data to regular grids, continuous resampling would be required. Moreover, editing operators would always have to assure consistency between multiple billboards representing the same object from multiple views.

6.7 Conclusion

We introduced 3D video billboard clouds as an image space representation for 3D video applications. Both the proxy billboard planes and displacement maps can be constructed from depth images of the scene acquired by a standard stereo matching algorithm. They provide a regular, uniform sampling of the scene in space and time which makes them suitable for standard signal processing methods. Application of a four-dimensional bilateral filter yields geometry with higher spatial and temporal coherence. Novel views are rendered using a GPU-accelerated method which generates consistent view-dependent geometry and textures for each individual frame. Modeling of dynamic three-dimensional scenes is achieved using a semi-automatic approach that generates a collection of 3D video billboard clouds.

Compared to our point-based data model, 3D video billboard clouds are able to achieve a higher image quality due to their regular structure that allows for more efficient post-processing. If, on the other hand, more flexibility for 3D video

editing is required, they can be used as an intermediate representation that can be easily converted into points after post-processing by backprojecting all pixels of their texture maps into three-dimensional space.

Possible extensions include fully automatic segmentation of complex scenes. This may be implemented by searching for planar patches in the input geometry using a 3D Hough transform similar to the approach by Décoret et al. [2003]. In contrast to the current system, this might lead to an over-segmentation that decomposes the scene into many very small billboards.

Chapter 7

Point Samples for Efficient Storage of 3D Scenes

The 3D video acquisition process generates a large amount of data. Assuming one texture and depth video per input view, it is a multiple of conventional 2D video data that scales linearly with the amount of input views and thus with the desired viewing range. The raw data has to be transformed into a representation suitable for interactive playback. To establish 3D video as a new form of digital multimedia content, this imposes novel challenges on the underlying data structure. Already conventional, two-dimensional digital video would not be possible today without suitable streaming and compression algorithms. There, however, representing the scene in a single resolution is sufficient, which does not hold for 3D video where the user can change his viewpoint at will. With 3D video, a user may want to view the scene as a whole, but, at the same time, he would like to be able to zoom his virtual camera to specific details. A 3D video data representation has to be flexible enough to cope with such situations.

Our point-based data model from chapter 5 builds an ideal basis for fulfilling those requirements. This chapter specifically demonstrates its strengths for streaming, multiresolution and compression. As the points do not store connectivity, streaming be can easily implemented by sequential transmission of single points. A multiresolution hierarchy can be built by successive merging of neighboring samples. Moreover, compression of a point-sampled scene can be implemented in a unified way. Because point samples integrate information about appearance and geometry, both data types can be handled in the same manner. In contrast to meshes, no separate data structures like textures have to be stored. Moreover, no additional topology information has to be compressed.

The compression approach presented in this chapter integrates all those aspects within one single algorithm. It processes various point attributes like position,

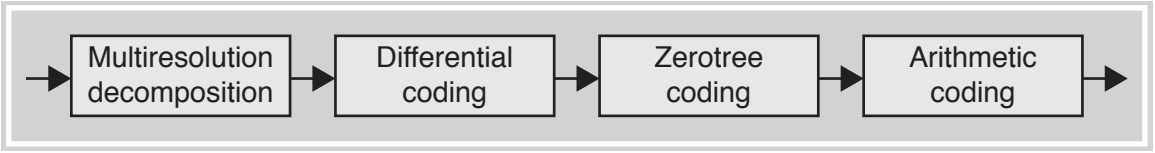


Figure 7.1: Compression pipeline.

color, and surface normal, and stores them as a compressed, progressive, multiresolution output stream. This data can be efficiently transmitted over the network. Resolution and detail of the point cloud progressively increase as more and more data is received. The presented algorithm concentrates on compression of static point models. Using a four-dimensional, spatio-temporal point cloud of a dynamic scene, as introduced in chapter 8, it can in principle be extended to 3D video.

7.1 Algorithm Overview

Our compression scheme comes as a general framework for coding a set of arbitrary point attributes. It is based on a multiresolution decomposition of the whole point set. Our predictive differential coding scheme has the ability of decorrelating the information contained in the model by exploiting local coherence. The multiresolution approach splits the information into several frequency bands. This property easily allows for progressive decoding by successively adding more detail to the model.

An overview of the whole compression pipeline is given in figure 7.1. In the first step, the multiresolution hierarchy is built up by recursively computing coarser approximations of the point set. This task requires local neighborhood relations between point samples that, unlike in triangle meshes, are not explicitly available in point-based models. To that end, we employ a search algorithm that delivers neighboring points. As a benefit, we can control the search in a way that gives us good correlations between neighbors in all attributes and not only in the geometry. Both the neighborhood search and the multiresolution decomposition are described in section 7.2.

Next, we compute the hierarchy of detail coefficients that describes how to successively reconstruct the original model from the lowest-resolution point set. The respective computations are driven by a prediction that is customized to the specific characteristics of each point attribute separately. After completing one hierarchy layer we immediately perform the quantization of the coefficients and propagate the quantization error into the computation of the next layer. This method prevents recursive accumulation of the quantization error over all multiresolution layers and thus minimizes the total error of our coding method. A detailed discussion follows in section 7.3.

We eventually end up with a set of quantized detail coefficients. They are not

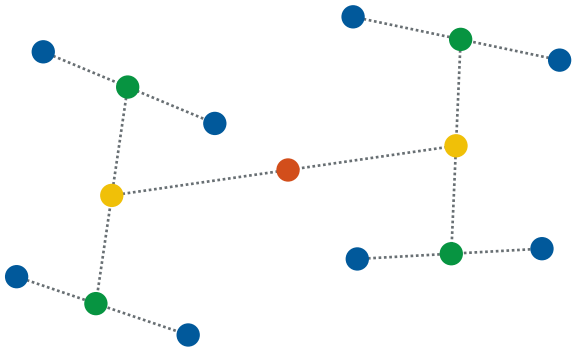


Figure 7.2: 8-neighborhood after 3 passes of hierarchical point contraction.

yet fully decorrelated but still contain some coherence which can be eliminated very efficiently by a zerotree coder. Finally, the data is further compressed by arithmetic coding. Both coders allow for progressive decoding. In addition to the conventional zerotree coder, we present a modified algorithm with a progressive behavior that better collaborates with our multiresolution framework. All those methods are further explained in section 7.4.

The first two stages of the pipeline have to be customized for the specific characteristics of the compressed point attributes. Stage one needs a distance measure for the neighborhood search, stage two requires a prediction operator and a coordinate transform. We provide specialized measures and operators for coding the point positions. We further show the extensibility of our framework towards other attributes by proposing coding methods for point colors and normals.

7.2 Multiresolution Decomposition

In the first stage, we compute a multiresolution hierarchy of the point cloud by recursively generating a sequence of subsampled versions $(\Lambda^{-1},\ldots,\Lambda^{-l})$ of the original point set Λ^0. We use here a notation related to the paper by Sweldens [1995] where Λ^{-k} denotes the point set at the k-th level of the multiresolution hierarchy. A specific point in a subsampled model is identified by λ_i^{-k}. We build the hierarchy bottom up, i.e. from high to low resolution, by decomposing in each step the point set into disjunctive point pairs and contracting each pair to an average point by averaging its associated attributes. In this way, we obtain a forest of binary trees of depth l, of which each layer describes a certain resolution. As the point pairs describe neighborhood relations between two points in the attribute space, each subtree in the hierarchy with depth k describes a neighborhood of 2^k points, as illustrated in figure 7.2. All those neighborhood relations are implicitly stored in the point ordering of the full resolution model and hence are still available after decompression.

Note that such a hierarchy can only be built if each layer in the forest but the root

layer contains an even number of points. Thus, the original model has to consist of a multiple of 2^l points. If this is not the case, the maximum $2^l - 1$ remaining points are stored separately and compressed by entropy coding only. For a moderate number of recursions, like $l = 10$, this number is relatively small compared to the overall size of typical models which often consist of several hundred thousand points.

The compression performance largely depends on the pair decomposition because it considers each average point as an approximation to each of its sons. Thus, we preferably want to contract "good" pairs of points with similar attribute values. Finding the best set of pairs is an optimization problem in the space spanned by all point attributes. This can be described as a minimum weight perfect matching problem [Lovasz and Plummer, 1986] in an undirected, weighted graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with vertices $\mathcal{V}$ and weighted edges $\mathcal{E}$. Its task is to find a set of edges $\mathcal{E}_M \subseteq \mathcal{E}$ of cardinality $|\mathcal{E}_M| = |\mathcal{V}|/2$ such that no two edges share a vertex in common and the sum of edge weights in $\mathcal{E}_M$ is minimal. In a complete graph with an even number of vertices, such a matching always exists. To speed up the matching process, we do not construct a complete graph but only an adjacency graph connecting each point with its $k = 12$ nearest neighbors. In the very unlikely event when no matching is found we can still increase k and let the algorithm run again, but this never happened in our experiments. A classical solution of the matching problem is provided by Edmonds' blossom shrinking algorithm [Edmonds, 1965]. In our implementation, we use the faster method by Cook and Rohe [1999].

The weight $w_{i,j}$ for each edge $\{i, j\}$ can be expressed as a sum of distance functions e_A, describing for each attribute A the difference of its values at the points λ_i^{-k} and λ_j^{-k} adjacent to that edge:

$$w_{i,j} = \sum_A e_A(i, j). \tag{7.1}$$

In practice, we incorporate here the point positions $\mathbf{X}_i^{-k}$ and normals $\mathbf{N}_i^{-k}$. The distance between normals corresponds to the cosine of their enclosing angle:

$$e_N(i, j) = \frac{1 - \mathbf{N}_i^{-k} \cdot \mathbf{N}_j^{-k}}{2}. \tag{7.2}$$

If we consider as a position weight only the Euclidean distance, we may end up after two recursive subsampling passes in 4-neighborhoods consisting of linearly arranged points. This will lead to instabilities in the subsequent detail coefficient computation. So we additionally include the previous direction of the contraction of the points λ_{2i}^{-k+1} and λ_{2i+1}^{-k+1} towards λ_i^{-k} and similarly for the contraction to

Figure 7.3: 64-neighborhoods after 16 matching passes.

λ_j^{-k}:

$$
\begin{aligned}
e_X(i,j) = &\left|\mathbf{X}_i^{-k} - \mathbf{X}_j^{-k}\right| \\
&+ \left|\frac{\mathbf{X}_{2i}^{-k+1} - \mathbf{X}_{2i+1}^{-k+1}}{|\mathbf{X}_{2i}^{-k+1} - \mathbf{X}_{2i+1}^{-k+1}|} \cdot \frac{\mathbf{X}_i^{-k} - \mathbf{X}_j^{-k}}{|\mathbf{X}_i^{-k} - \mathbf{X}_j^{-k}|}\right| \\
&+ \left|\frac{\mathbf{X}_{2j}^{-k+1} - \mathbf{X}_{2j+1}^{-k+1}}{|\mathbf{X}_{2j}^{-k+1} - \mathbf{X}_{2j+1}^{-k+1}|} \cdot \frac{\mathbf{X}_i^{-k} - \mathbf{X}_j^{-k}}{|\mathbf{X}_i^{-k} - \mathbf{X}_j^{-k}|}\right|.
\end{aligned}
\tag{7.3}
$$

figure 7.3 shows the resulting 64-neighborhoods after six recursive executions of the matching algorithm.

7.3 Predictive Differential Coding

Next, we successively compute a binary forest of detail coefficients $(\Gamma^{-l}, \ldots, \Gamma^{-1})$ of which each layer Γ^{-i} contains the information that is necessary to reconstruct Λ^{-i+1} from Λ^{-i}. The calculations are done with the help of two attribute-specific operators: a coordinate transform C and a prediction operator B. With regard to later decompression, C has to be invertible but not B. We finally end up in a set of coefficients $(\Lambda^{-l}, \Gamma^{-l}, \ldots, \Gamma^{-1})$ which is sufficient for reconstruction of the original model. Because each detail layer Γ^{-i} has the same size as its corresponding average layer Λ^{-i}, the whole coefficient set is still of the same size as the original

model. But the new data is much more decorrelated and therefore more suitable for compression.

Because high-resolution detail coefficients recursively depend on their lower-resolution counterparts, quantization errors that will be introduced due to compression would accumulate in each recursion. To prevent this, we compute the detail coefficients top down from Γ^{-l} to Γ^{1}, quantize them after each recursion and propagate the quantization error to the next higher resolution: After determining and quantizing the coefficients Γ^{-k} we first update the next average layer Λ^{-k+1} out of Λ^{-k} and the quantized Γ^{-k} and then recursively proceed computing the detail coefficients Γ^{-k+1} of the next higher resolution layer. The decoder later successively reconstructs higher-resolution models Λ^{-k+1} from the average points Λ^{-k} and the detail Γ^{-k}. There, all the data processing can be done in place by recursively substituting Λ^{-k} and Γ^{-k} by Λ^{-k+1}, similarly to the lifting scheme [Sweldens, 1995].

The actual computation of the detail coefficients is performed as illustrated in figure 7.4. For each contraction pair $(\lambda_{2i}^{-k+1}, \lambda_{2i+1}^{-k+1})$ and its corresponding average point λ_i^{-k}, we try to predict λ_{2i}^{-k+1} from λ_i^{-k} using a prediction operator B and store the prediction error in γ_i^{-k}. With this information, both higher resolution points can be reconstructed due to symmetry. In many cases, the prediction does not perform well if it is carried out in the global space of the model. So we apply beforehand a local coordinate transform C that depends on the points of Λ^{-k} and better characterizes the attribute space locally. Figure 7.5 illustrates the operator framework both for encoding and decoding. Note that the decoder performs the same prediction B as the encoder, whereas the coordinate transform C during decoding is inverse in order to obtain again the global model coordinates from the coefficients encoded in the local reference frames.

The whole transformation is recursively applied up to a depth l. As the coherence between neighboring points get worse for lower resolutions, a depth between $l = 6$ and $l = 8$ usually gives the best compression results. Because the coordinate transform C depends on neighborhood relations in the point set λ^{-k}, the multiresolution forest is built some recursions higher for getting neighborhood relations in λ^{-l}. We actually use there a depth of $l+2$ because our coordinate transforms rely on 4-neighborhood relations.

In the following, we provide operators C and B for the compression of the point positions, colors and normals.

7.3.1 Positions

For compression of the point positions, we want to exploit the fact that all points are arranged on a surface and not for instance in a volume. So we locally approximate the geometry by a least-squares plane and store the prediction error relative to that plane.

For an average point λ_i^{-k}, the operator C constructs the least-squares plane from

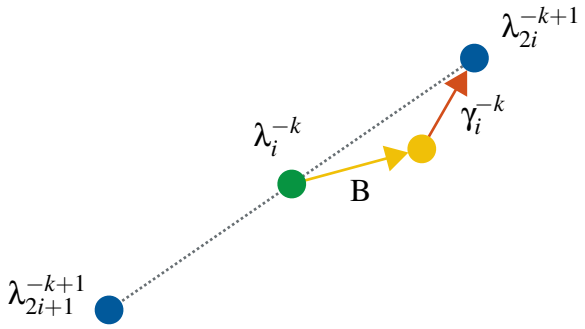


Figure 7.4: Prediction and detail coefficient.

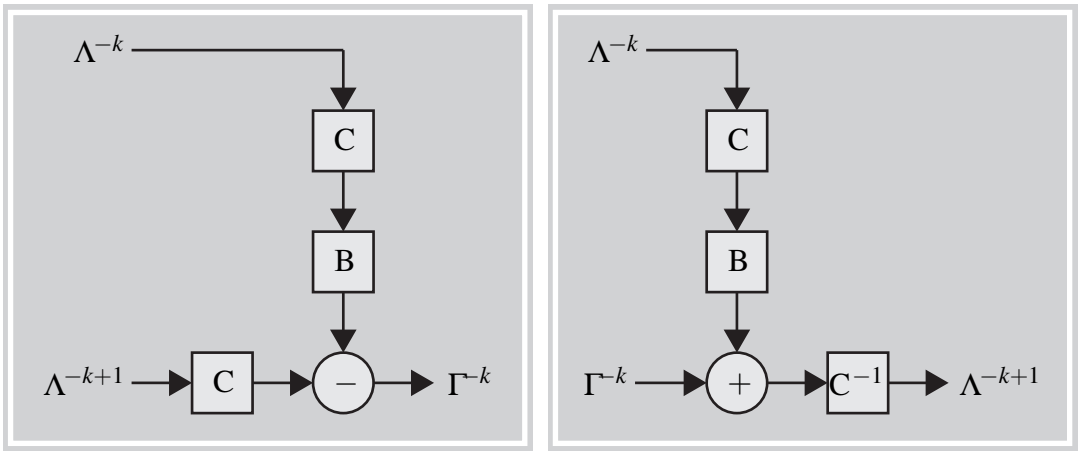


Figure 7.5: Operator framework. Left: encoding, right: decoding.

the point positions $(\mathbf{X}_{i-(i \bmod 4)}^{-k}, \ldots, \mathbf{X}_{i-(i \bmod 4)+3}^{-k})$ that are part of a neighborhood relation. The plane minimizes the squared distances to these four points and is therefore an approximation to the local geometry of the model. It is computed by a principal component analysis [Keinosuke, 1990] over those points which delivers an orthonormal coordinate system $(\mathbf{N}, \mathbf{V}_1, \mathbf{V}_2)$ describing the principal directions of the covariance ellipsoid of the point positions [Pauly et al., 2002]. This system is aligned to the least-squares plane of which $\mathbf{N}$ represents its normal vector. As depicted in figure 7.6, we translate the origin of that system to $\mathbf{X}_i^{-k}$ and represent all points by cylindrical coordinates (R, Θ, Z).

The operator B then predicts the coordinates (R, Θ, Z) of $\mathbf{X}_{2i}^{-k+1}$. Assuming that the least-squares plane is a close approximation of the geometry, we can predict the elevation Z of the point over the surface as $Z = 0$. Under the condition of an approximately regular sampling of the model, we can also make a prediction for R: As the average point density drops by a factor of $\sqrt{2}$ with each lower resolution step, R can be predicted from the density of the four average points divided by $2\sqrt{2}$. The density of the average points itself is estimated by their average distance. Finally, we also predict $\Theta = 0$. The prediction error for Θ then

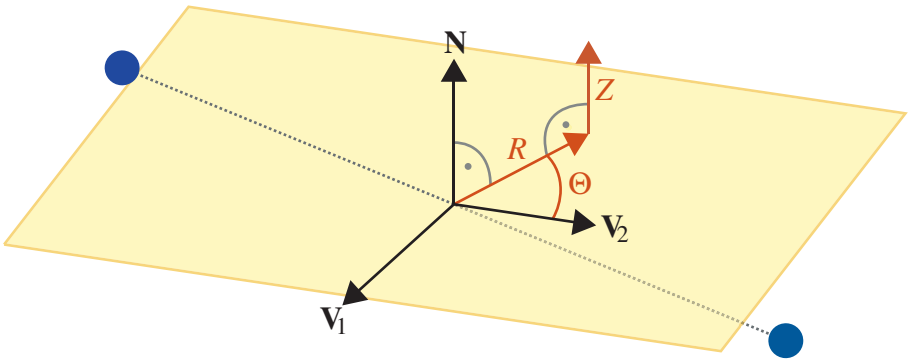


Figure 7.6: The reference frame for the position detail coefficients.

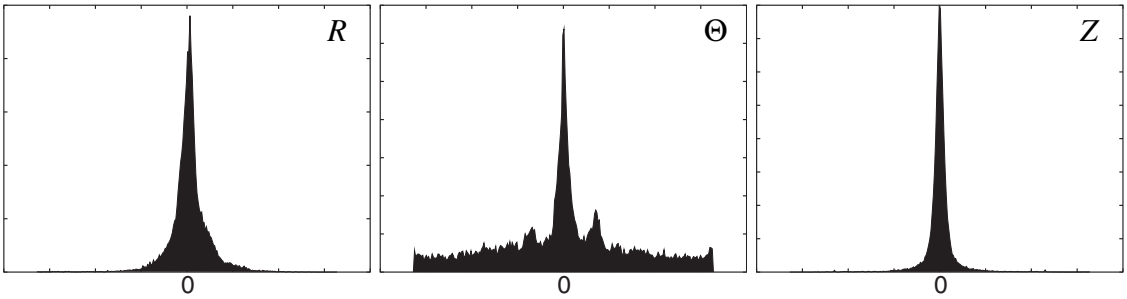


Figure 7.7: Histograms of the position detail coefficients R (left), Θ (middle), and Z (right).

lies in the interval $[0, 2\pi]$. It can be further constrained to $[0, \pi]$ by swapping the points λ_{2i}^{-k+1} and λ_{2i+1}^{-k+1} along with their associated subtrees in the multiresolution hierarchy in the case of an error greater than π.

Figure 7.7 shows histograms of the detail coefficients from the Chameleon model over all resolution levels. The peaks at zero prove the suitability of our predictions. It also shows an accumulation of the angular component around zero which is due to the alignment of our reference frame to the covariance ellipsoid.

7.3.2 Colors

All colors are first transformed from RGB into the global YUV space by the operator C. This representation is more closely related to the human perception as it splits the color information into a luminance part Y and a chrominance U and V. Because humans are more sensitive for the luminance than the chrominance, we spend more bits for Y than for U respectively V during quantization.

Assuming that neighboring points of the model have similar colors, the prediction operator B is zero. Hence, we store in the detail coefficients for each contraction pair the difference between the colors of one higher-resolution point and the average point. Histograms of those coefficients are depicted in figure 7.8.

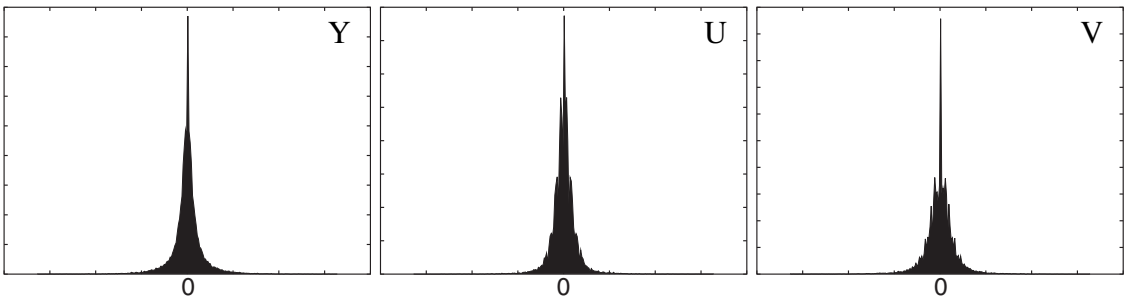


Figure 7.8: Histograms of the color detail coefficients Y (left), U (middle), and V (right).

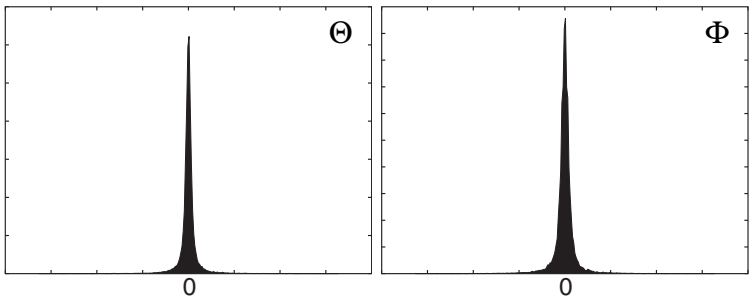


Figure 7.9: Histograms of the normal detail coefficients Θ (left), and Φ (right).

7.3.3 Normals

For each contraction pair we store the angle between the average normal and one of the higher-resolution normals in spherical coordinates. Hence, the prediction B is zero for both Θ and Φ. The coordinate transform C needs a local reference frame to align the spherical coordinate system. We use the average normal as the polar reference direction. To obtain an azimuthal reference vector, we project the vector with the biggest eigenvalue delivered by the principal component analysis from the position coder onto the plane perpendicular to the average normal. Figure 7.9 shows histograms of the detail coefficients.

7.4 Encoding the Detail Coefficients

The detail coefficients are eventually compressed by a zerotree coder [Shapiro, 1993] which is specialized on the coding of multiresolution coefficients and is able to encode them very efficiently. It exploits similarities between coefficients in subtrees of the multiresolution hierarchy. The output of the zerotree coder is a stream of symbols which is further compressed by arithmetic coding. We currently use the algorithm provided by Moffat et al. [1998]

Both zerotree and arithmetic coder behave progressively. A partial arithmetic decoding of the compressed data stream delivers a prefix of the zerotree stream

after arithmetic decoding. The zerotree decoder then produces a set of inexact coefficients which gets refined as more data is delivered to the decoder.

The conventional zerotree algorithm however successively refines the coefficients of all resolutions simultaneously during progressive decoding. This behavior does not blend well with our quantization error propagation because our coder assumes an ascertained accuracy of lower-resolution coefficients. Introducing a greater error after compression leads to instabilities in the least-squares planes and thus produces an inferior quality of decompressed point positions. Therefore we use the conventional zerotree coder only for fixed rate compression.

If real progressive decompression is desired, we suggest a modification of the zerotree encoder which consists in reordering its output stream in such a way that, during later decompression, the detail coefficients get refined successively from low to high resolution. The conventional zerotree coder does the encoding in several passes. In each pass, an additional bit for each significant coefficient is coded, successively gaining more accuracy. Within each pass, the data is processed from the low resolution coefficients up to the high resolution. Our modification consists in swapping those two orderings. Thus, we first arrange the data from low to high resolution coefficients and then do the ordering according to the bit significance. During progressive decoding of that stream, the high resolution coefficients all start with a value of zero and they will not be refined until all lower resolution coefficients are fully decoded. This manifests itself in increasing the resolution of the decoded model. In the following section, we provide experimental results for both approaches.

7.5 Results

We evaluate our coders with various point models of different sizes. The compression performance is quantified for each attribute by the average number of bits per point (bpp) in relation to the loss of quality, measured by the peak signal to noise ratio (PSNR).

In the field of mesh compression, the geometric error is usually calculated from the distance between the compressed and uncompressed mesh surfaces. A similar approach for point set surfaces is the comparison of the corresponding MLS surfaces [Pauly et al., 2002] which is also used by Fleishman et al. [2003]. However, this metric is unable to measure the sampling of the model, which is a crucial criterion for high-quality point-based rendering. In contrary to triangle meshes, a bad sampling would lead to cracks in the surface due to the lack of connectivity information. Furthermore, the MLS surface tends to smooth the quantization errors such that the measured error is usually lower than disturbance in the visual quality.

For these reasons we use the MLS metric for comparison with the algorithm by Fleishman et al. [2003] only. Whereas, in the main part of the following evalua-

	PSNR / dB			Bits per point	
	Y	U	V	Fixed	Progr.
Chameleon	29.5	39.5	38.8	2.5	2.7
(102k points)	24.4	38.1	37.3	0.9	1.0
Octopus	26.9	33.7	31.0	2.0	2.1
(466k points)	21.9	32.2	28.7	0.8	0.9
Face	35.5	47.4	42.3	1.7	1.8
(41k points)	31.9	45.1	39.7	0.8	0.8

Table 7.1: Compression performance of color coder.

tion, we concentrate on measuring the quantization error directly between discrete pairs of corresponding samples from the original and compressed model. The PSNR for the position attribute is evaluated using the Euclidean distance between the points. The peak signal is given by the length of the diagonal of the bounding box. The error between the normals is calculated from the enclosed angle with a peak angle of 180 degrees. For the colors, we compute the PSNR on the scalar values of each channel Y, U and V separately.

We first compare our fixed rate compression with our progressive scheme, using various coarse quantizations for the different point attributes. Figure 7.10 shows rate-distortion curves from the position coder. The progressive coding performs between 2 and 10 percent worse than the fixed rate coding. This is due to the fact that the compression gain of the arithmetic coding is worse for the progressively reordered zerotree stream than for the conventional stream. The behavior of the other pipeline stages is identical in both cases. Compared to state of the art mesh compression, the PSNRs given here are lower since they also measure the quality of the sampling. As shown by the images of figure 7.13, the compression artifacts differ from those of mesh compression due to the lack of connectivity information. For low bit rates, the sampling gets more and more irregular. We compensate for these irregularities by recomputing the splat radii during decompression which can be done very efficiently using the neighborhood relations that are inherently stored in the point ordering.

We further give some results for our color and normal coders. The color compression is evaluated in table 7.1. As can be seen in figure 7.14, it shows some blurry artifacts which are typical for high compression of images. Table 7.2 shows rate-distortion values of our normal coder; sample images are given in figure 7.15. In contrast to mesh compression, we are able to code the appearance of the model in the same way than the geometry, using the same data structures. Hence there is no need to store additional texture images and to associate and compress texture coordinates with each vertex.

Its multiresolution character makes our compression approach very suitable for streaming and progressive decompression. Figure 7.11 shows rate-distortion curves for progressive decoding of different models. The amount of data is de-

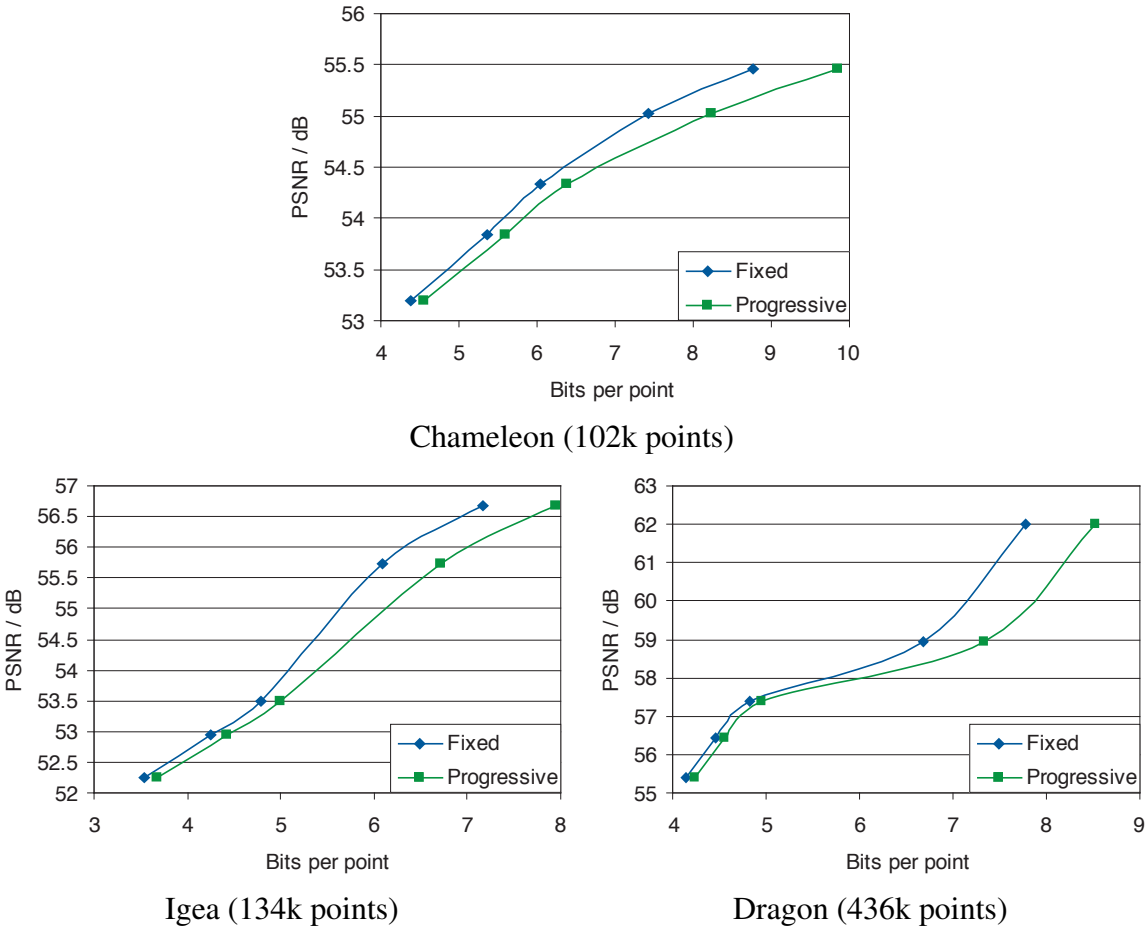
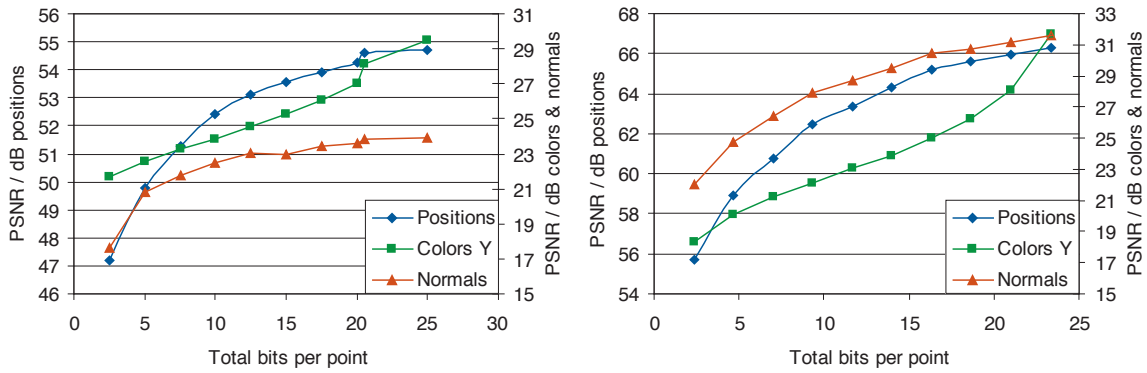


Figure 7.10: Compression performance of position coder.

	PSNR / dB	Bits per point	
		Fixed	Progr.
Chameleon	25.8	10.5	13.1
(102k points)	24.8	5.8	6.9
Igea	40.0	13.5	17.0
(134k points)	36.7	7.9	9.4
Dragon	31.1	10.5	12.7
(436k points)	27.8	5.6	6.4

Table 7.2: Compression performance of normal coder.



Chameleon (102k points) Octopus (466k points)

Figure 7.11: Rate-distortion curves for progressive decompression.

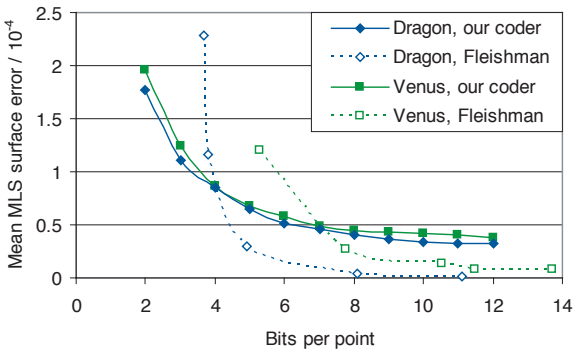


Figure 7.12: Comparison of our progressive position coder with the method by Fleishman et al. [2003]

scribed in total bits per point. One fifth of the bits is used for color compression. The rest is spread evenly over the position and normal coding. As more and more data is decoded, the resolution of the model successively increases. Visual results of this process are provided in figure 7.16.

Figure 7.12 shows a comparison of our progressive position coder with the one presented by Fleishman et al. [2003]. We use the error metric from their paper which measures the mean distance between the MLS surfaces of the original and compressed models. While Fleishman's coder introduces less error for high bit rates we are able to achieve superior results for lower rates of about 4 bits per point.

7.6 Conclusion

In this chapter we proposed a framework for compression of point-sampled models. In contrary to mesh compression we are able to code not only the geometry

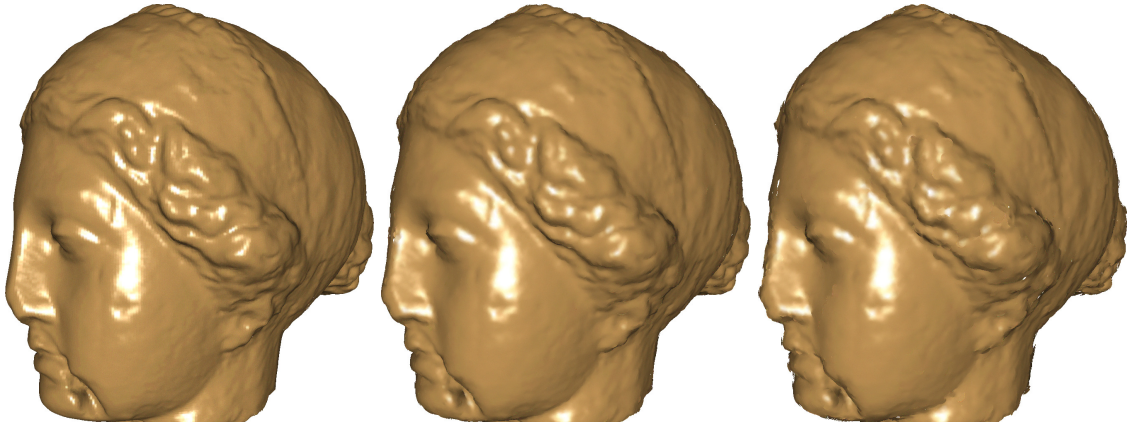


Figure 7.13: Comparison of position compression (uncompressed, 8.8 bpp, 4.4 bpp).

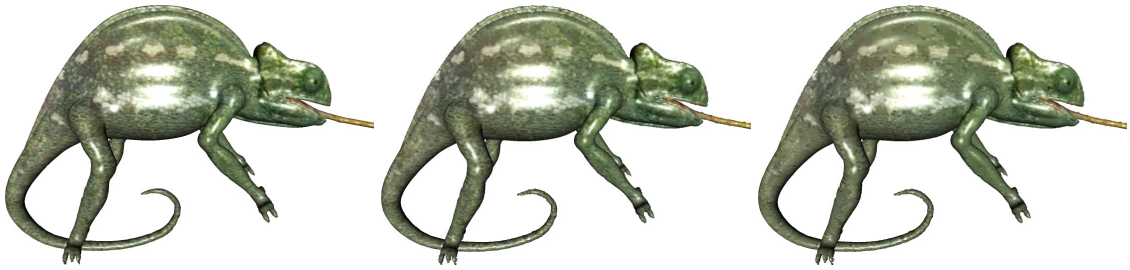


Figure 7.14: Comparison of color compression (uncompressed, 2.5 bpp, 1.0 bpp).

Figure 7.15: Comparison of normal compression (uncompressed, 12.7 bpp, 6.4 bpp).

Figure 7.16: Progressive decompression with total bit rates of 7, 14 and 23 bits per point.

but arbitrary appearance attributes associated with the point samples in a unified way. The lack of connectivity information further helps to save storage space.

Our method is based on multiresolution predictive coding that has the capability of exploiting similarities between neighboring points. By employing a graph matching algorithm we are able to find neighborhood relations that give us a maximum correlation in all attributes. The multiresolution approach naturally leads to a compression scheme that allows for progressive decoding.

In particular, we present a coder for the model geometry. By transforming the point positions into a local reference frame, we exploit the fact that all the points are living on a surface. We have shown by a set of various examples that this approach works quite efficiently. We complete our framework by also suggesting methods for compression of colors and normals.

By using the four-dimensional, spatio-temporal point cloud structure presented in the next chapter, our approach naturally extends to 3D video. In the temporal domain, the prediction would have to be changed from hierarchical perfect matching to a sequential approach that relates points from neighboring frames. This would allow streaming of successive frames during playback. Because many parts of the scene are usually static, predicting points as being constant would already yield a good compression rate. Further improvements may be achieved by additionally using scene flow algorithms [Vedula et al., 1999] that generalize the motion estimation used in 2D video codecs to three dimensions.

Chapter 8

The Video Hypervolume

By modeling both spatial and temporal aspects of real-world scenes, 3D video is in principle a four-dimensional structure. Playback generates a sequence of images of three-dimensional scenes and, thus, successively accesses local three-dimensional subspaces of the data stream at increasing temporal positions. Applications such as 3D video editing require more flexible access to the video stream. Operators that modify the data have to consider all four dimensions in order to avoid temporal or spatial discontinuities. Interactive editing should permit temporally coherent manipulation of complete frame sequences.

Inspired by research on visualization of time-varying volumetric data [Bajaj et al., 1998; Pasko et al., 2002], we introduce in section 8.1 the video hypervolume as a representation for 3D video. Similar to video cubes for 2D video [Fels and Mase, 1999; Klein et al., 2002], it considers space and time as compound entities in four dimensions. By applying hyperslicing and projection methods as described in section 8.2, we can exploit both spatial and temporal coherence. With these operations, the video hypervolume builds the basis for our nonlinear 3D video editing system introduced in chapter 9, which allows for operating in all four dimensions in a unified manner.

Based on the point samples of chapter 5, the hypervolume has some nice properties for representing 3D video data. Primarily, points do not need explicit topology and, hence, no connectivity information has to be constructed and maintained over time. Secondly, they encode explicit 3D scene geometry and color in a homogeneous way. Furthermore points can be flexibly extended to contain application-specific data like sophisticated material properties or object labels. In terms of storage efficiency, the hypervolume has some advantages over a dense regular grid due to the irregular sampling and the level of sparsity of 3D video data.

The point-sampled video hypervolume fits nicely into existing frameworks for processing point-sampled geometry [Zwicker et al., 2002a]. This allows for utilizing a variety of post-processing operations for outlier removal [Weyrich et al., 2004], redundancy elimination [Sadlo et al., 2005], simplification [Pauly et al., 2002], and geometry smoothing [Pauly and Gross, 2001]. Many of these algo-

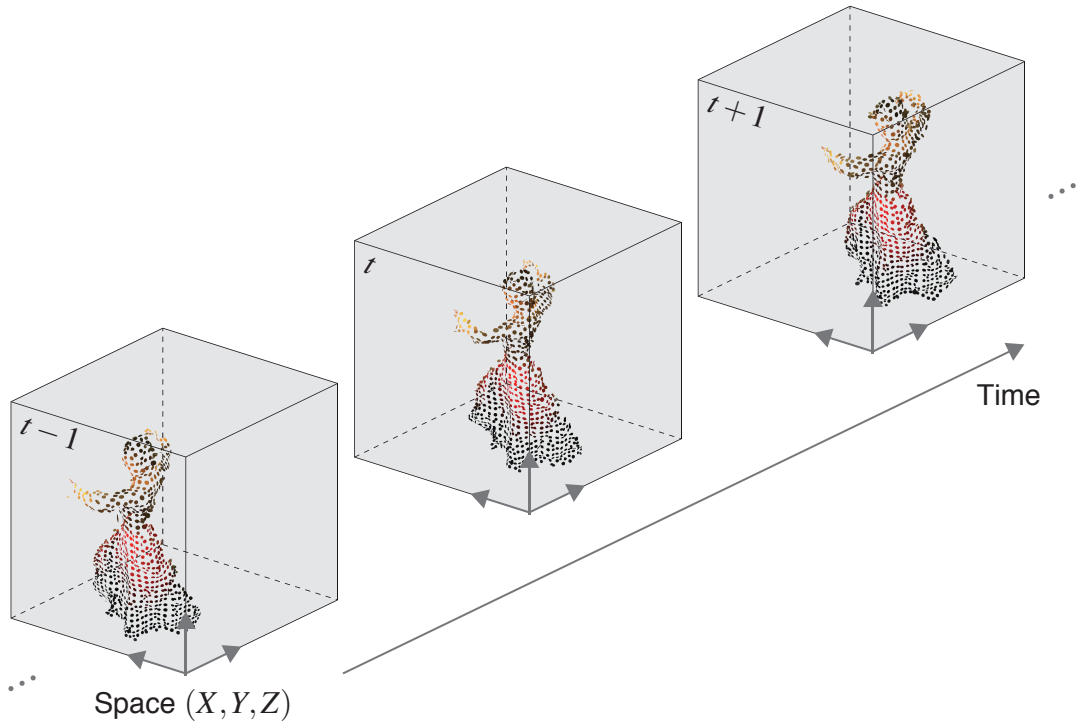


Figure 8.1: The video hypervolume.

rithms are independent of the number of dimensions and, thus, can naturally be extended to integrate time coherence.

However, high-quality 3D video still requires storage of several megabytes per frame. The complete hypervolume usually does not fit into main memory. 3D video editing does not only require interactive random access to the data but also has to allow for local manipulations. To handle that large amount of data, we propose in section 8.3 an out-of-core representation for the video hypervolume that permits random access and dynamic updates directly from the hard disk. Combined with the multiresolution hierarchy of section 8.4, this is possible at interactive rates.

8.1 Data Model

The video hypervolume is an irregular set of point samples in the four-dimensional spacetime domain, as illustrated in figure 8.1. Each sample represents a point on a scene surface with a positional coordinate (X, Y, Z) and a temporal coordinate T.

The video hypervolume can be constructed independently from the 3D video acquisition system, using e.g. depth and color images as an input. It does not impose any constraints on the setup of the acquisition cameras as long as occlusions can be resolved. Each point sample in the video hypervolume carries a set of

attributes describing local surface properties like position, orientation and color. Identification of a specific sample is done via its position attribute $\dot{\mathbf{X}} = (X, Y, Z, t)^{\mathrm{T}}$ which is a vector in Euclidean spacetime $\mathbb{R}^4$.

The point samples in the video hypervolume can be easily constructed by back-projecting the image pixels from the acquisition cameras using the corresponding depth information. To generate hole-free renderings as output, each point sample represents a small region in spacetime such that its projection onto the screen covers a certain area of pixels. Our point-based data model of chapter 5 already provides full surface coverage in the spatial domain. To also cover the time domain, we generalize it to 4D hyperpoints representing small ellipsoidal hypervolumes in $\mathbb{R}^4$. A hyperpoint is constructed from four orthogonal vectors $\dot{\mathbf{T}}_1, \ldots, \dot{\mathbf{T}}_4$, $\dot{\mathbf{T}}_i = (t_{ix}, t_{iy}, t_{iz}, t_{it})^{\mathrm{T}}$ spanning a 4D Gaussian ellipsoid with covariance matrix $\dot{\mathrm{V}} = (\dot{\mathbf{T}}_1, \ldots, \dot{\mathbf{T}}_n) \cdot (\dot{\mathbf{T}}_1, \ldots, \dot{\mathbf{T}}_n)^{\mathrm{T}}$. The first three vectors describe a conventional point sample in the spatial domain. They can be constructed using the methods of chapter 5 and by setting their fourth component to zero. To cover the time domain we define the fourth spanning vector as $\dot{\mathbf{T}}_4 = (0, 0, 0, \Delta t)^{\mathrm{T}}$, where Δt denotes the temporal sampling density corresponding to the frame rate of the video.

In the spatial domain, the samples are irregularly placed on the surfaces, whereas in time we usually deal with regular sampling resulting from distinct video frames of the acquisition system. Hence, there is some remaining redundancy at static parts of the scene as their samples are explicitly stored at each point in time. Being a general 4D point cloud, the video hypervolume in principle allows for completely irregularly positioned samples. Specifically, merging static samples into single ellipsoids elongated in time over multiple frames provides an easy way of compression by exploiting time coherence. However, such an approach introduces several drawbacks for nonlinear editing applications: The introduced inter-frame dependencies increase complexity for video cutting or editing of single frames. Moreover, time complexity of point queries in a completely irregular video hypervolume increases with the number of frames, making editing performance dependent on the length of a video. Therefore, we prefer regular temporal sampling at the cost of more storage space. This is similar to intra-frame-only coding used in most common formats for nonlinear 2D video editing, such as DV, DVCPRO [Uchida et al., 1996], or HDCAM [Wheeler, 2001]. Storage is handled by our out-of-core data structure presented in section 8.3.

8.2 Hyperslicing

Hyperslicing [Woodring et al., 2003] is the fundamental operation for visualizing parts of the video hypervolume, either for playback or for interactive editing. Projection onto the screen is performed in a two-stage process: First, hyperslicing selects a subset of all 4D point samples and projects them to three-dimensional

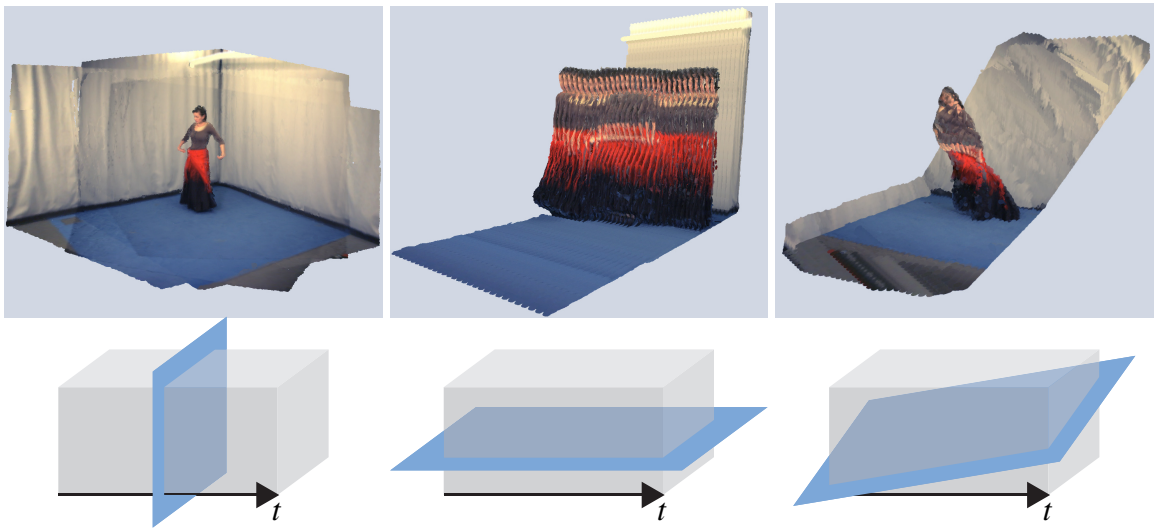


Figure 8.2: Different hyperslice orientations: orthogonal to the temporal axis for playback (left), parallel to the temporal axis for visualizing dynamic behavior (middle), and arbitrary orientation (right). The images of the upper row have been generated with the preview-renderer of our interactive 3D video editing system. For the slicing illustrations in the lower row, the number of dimensions of the hypervolume has been reduced to three.

space. The resulting 3D point cloud is then displayed using conventional point rendering methods (see chapter 5).

Intuitively, hyperslicing extracts a three-dimensional subspace from the 4D volume by intersection with a hyperplane. Depending on the orientation of the hyperplane, different views on the four-dimensional spacetime can be generated, as shown in figure 8.2. A hyperslice orthogonal to the time domain generates a conventional 3D video frame. Thus, 3D video playback is achieved by moving the position of such a slice continuously along the temporal axis. Conversely, a hyperslice that is parallel to the temporal axis visualizes the time domain. It can be used to perform 3D video editing over multiple frames with one single operation. Generally, our editing system allows to perform hyperslicing arbitrarily, providing the user with views of both spatial and temporal scene information. Applications for temporal slices are discussed in chapter 9.

Intersecting the hypervolume with a single hyperplane would select all points $\dot{\mathbf{X}} \in \mathbb{R}^4$ fulfilling the plane equation $\dot{\mathbf{N}}_H \cdot \dot{\mathbf{X}} - b_H = 0$, where $\dot{\mathbf{N}}_H \in \mathbb{R}^4$ is the normal of the hyperplane and b_H its distance from the origin. To comply with the sparse, irregular sampling of our video hypervolume, we extend this procedure as depicted in figure 8.3 and select all points within a specific distance Δb_H from the plane by solving

$$|\dot{\mathbf{N}}_H \cdot \dot{\mathbf{X}} - b_H| \leq \Delta b_H. \tag{8.1}$$

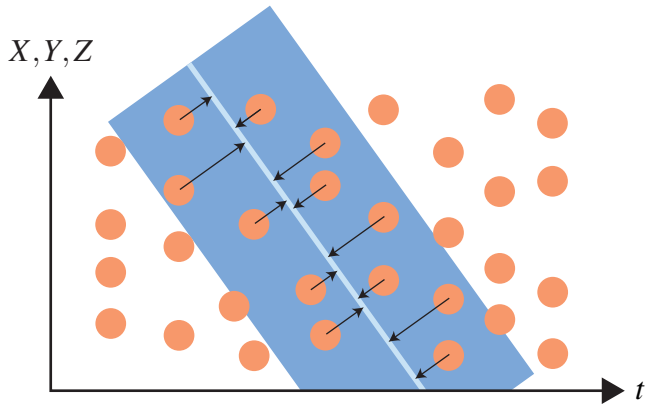


Figure 8.3: Samples of the video hypervolume (red) are intersected with the hyperslice (blue) and projected onto its center plane.

Three-dimensional positions

$$\mathbf{X} = \dot{\mathrm{P}}_H \cdot \dot{\mathrm{R}}_H \cdot \dot{\mathbf{X}} \tag{8.2}$$

are obtained by a rotation $\dot{\mathrm{R}}_H$ of $\dot{\mathbf{X}}$ into a coordinate system locally aligned at the hyperplane followed by a parallel projection $\dot{\mathrm{P}}_H$ along the hyperplane normal. The local coordinate system is spanned by four orthonormal column vectors $\dot{\mathbf{V}}_1, \ldots, \dot{\mathbf{V}}_4$ with $\dot{\mathbf{V}}_1 = \dot{\mathbf{N}}_H$, yielding $\dot{\mathrm{R}}_H = (\dot{\mathbf{V}}_1, \dot{\mathbf{V}}_2, \dot{\mathbf{V}}_3, \dot{\mathbf{V}}_4)^{\mathrm{T}}$. For 2D rendering of the projected 3D point samples using EWA volume splatting (see section 5.1.2), the covariance matrices have to be projected accordingly by computing

$$\mathrm{V} = \dot{\mathrm{P}}_H \cdot \dot{\mathrm{R}}_H \cdot \dot{\mathrm{V}} \cdot \dot{\mathrm{R}}_H^{\mathrm{T}} \cdot \dot{\mathrm{P}}_H^{\mathrm{T}}, \tag{8.3}$$

which results in descriptions of three-dimensional Gaussian ellipsoids.

To find the intersections with the hyperslice, the irregular point cloud has to be stored in a data structure that allows for efficient spatial queries. A well-established spatial indexing structure for irregular point clouds is the kd-tree [Bentley, 1975]. However, traditional kd-trees have to completely fit into the main memory of the computer, which is not possible with our large 3D video data sets. Thus, we employ an extension to an out-of-core structure, which is described in section 8.3. Apart from point queries, it also supports efficient dynamic modifications, which are necessary for 3D video editing. Moreover, to achieve interactivity, the structure is extended to a multiresolution hierarchy in section 8.4.

8.3 Out-of-Core Data Structure

Our out-of-core data structure is based on the Bkd-tree [Procopiuc et al., 2003]. It is a dynamic external-memory adaptation of the kd-tree [Bentley, 1975] for indexing multi-dimensional point data sets. Compared to octrees, kd-trees have

	60 frames at 200k surfels				2 frames at 1.75M surfels	
	Single 4D tree		One 3D tree per frame			
	R*	Bkd	R*	Bkd	R*	Bkd
Average insertion time per frame	140s	34s	64s	7s	1141s	112s
Average full frame query time	1.10s	1.17s	0.69s	0.61s	5.0s	4.6s
Total size on disk	1542MB	1033MB	1575MB	1134MB	415MB	285MB

Table 8.1: Performance comparison of the Bkd-tree with the R*-tree.

a better storage efficiency for sparse point clouds because they can be always fully balanced and thus contain less empty cells [Samet, 1984; Gross and Pfister, 2007]. This is especially important for out-of-core structures where the amount of necessary hard disk accesses should be kept as low as possible.

The traditional kd-tree is static: It has as to be completely rebuilt upon changes. To avoid that, the Bkd-tree uses the logarithmic method [Bentley, 1978; Overmars, 1987], a common approach to make originally static structures dynamic.

In prior experiments with unoptimized code, we compared the Bkd-tree with the famous R*-tree [Beckmann et al., 1990]. We considered both using one tree for the complete four-dimensional hypervolume, as well as multiple three-dimensional trees storing one frame each. Access time was measured by doing point queries returning one complete 3D video frame as result. To compare the performance of dynamic updates, we measured the average time needed for inserting surfels of each frame into initially empty trees. The results in table 8.1 show that the Bkd-tree outperforms the R*-tree in both I/O efficiency and space utilization.

To be independent from the duration of the 3D video, we exploit the regular structure of the video hypervolume in the temporal domain and use one Bkd-tree per frame. Thus, our data structure is a combination of a regular grid in the temporal domain and Bkd-trees in the spatial domains. In contrast to one single four-dimensional tree for the whole hypervolume, query performance with respect to the number of frames is $O(1)$.

The following sections give detailed descriptions of the building blocks of our structure: the out-of-core kd-tree, the logarithmic method, and the integration with the video hypervolume. The data structure was implemented based on the code by Procopiuc et al. [2003], using the TPIE library [Vengroff, 1994] which provides the basic C++ structures and abstractions for out-of-core programming.

8.3.1 External-Memory kd-Tree

A kd-tree [Bentley, 1975] is a binary space partitioning data structure. It is traditionally used for indexing multi-dimensional point data sets and can be used to perform axis-aligned bounding box or window queries at $O(\sqrt{n})$ [Lee and Wong, 1977], or nearest neighbor queries or $O(\log n)$ time complexity in a point cloud of

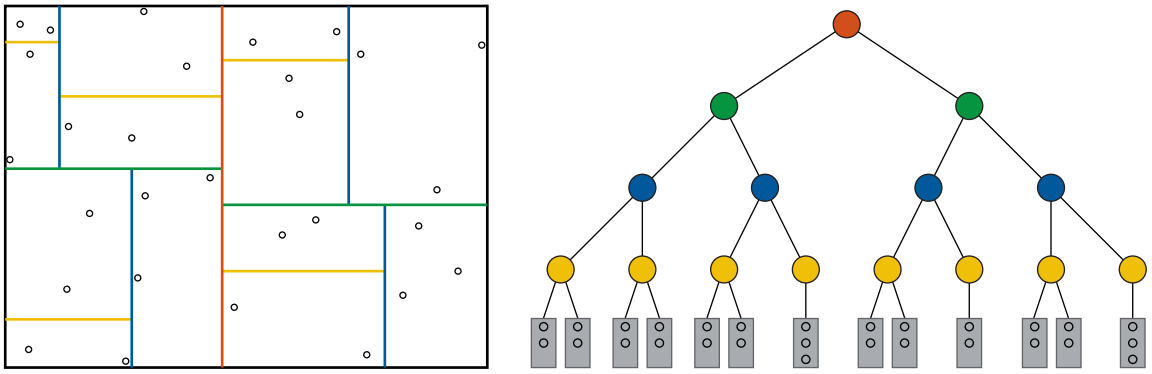


Figure 8.4: A blocked 2D kd-tree without block optimization. Most leaf blocks are not full, storage efficiency is approximately 72%.

size n. At each node, the k-dimensional space is split orthogonal to one dimension such that the point cloud is divided into two subsets of roughly equal size. Splitting is repeated recursively until the size of the point set contained in the current subspace reaches a lower bound—the so-called bucket size. The points are then stored in the leaves of the tree.

An out-of-core adaptation of the kd-tree has to amortize for the high access cost to external memory. For storage on the hard disk it should be optimized for the size of a disk block which is the indivisible amount of data that is transferred into main memory at each access. Choosing the bucket size of the kd-tree according to the size of one disk block is an obvious way: if one disk block is able to hold b points, the bucket size of the tree should be b. Thus, at each query, a whole leaf is transferred to main memory.

However, as illustrated in figure 8.4, choosing an inappropriate split criterion results in many half-empty leaves causing unnecessary transfer of unused data. For optimal space utilization, most leaves should be filled. Splitting the point set in such a way that the left subtree always contains a power of two of full disk blocks achieves that: Given q points to be split, $b \cdot 2^{\lfloor \log_2(q/b-1) \rfloor}$ points are put into the left child. As a result, the left subtree of the root node will be perfectly balanced and have perfect space utilization. This is true for the left subtree of any node. Thus, only a single block at most is not filled—the one in the rightmost path. Given a number n of points and a block size b, this is the smallest possible tree. Figure 8.5 shows such a tree for the point set of the previous example. Note that the number of internal nodes decreased from 15 to 9, and the number of leaf blocks decreased from 13 to 10. While the maximum height is the same as before, the average length of a path from root to leave decreased.

The inner nodes are also stored in a blocked fashion: Assume b_I is the number of inner nodes that can be stored in a single block. Starting from the kd-tree root, the tree is traversed depth-first until b_I nodes have been visited. All traversed nodes are then stored in a block. The remaining nodes are blocked recursively. The total number of blocks needed for all the internal nodes is $O(n/(bb_I))$ [Pro-

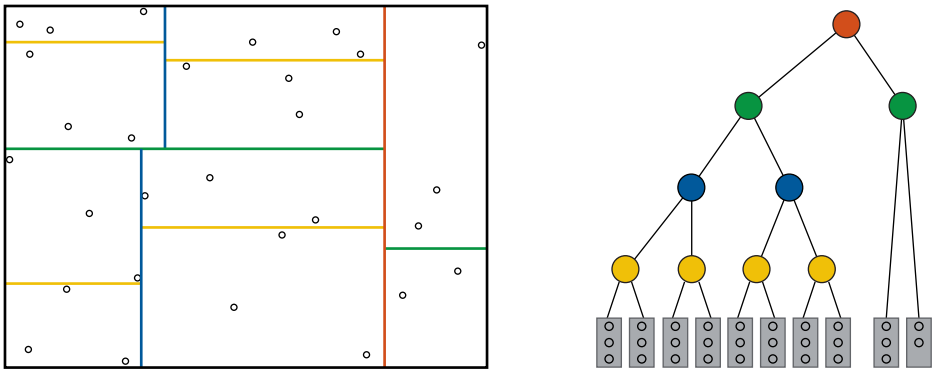


Figure 8.5: A blocked 2D kd-tree with block optimization. Most leaf blocks are full, storage efficiency is approximately 97% (optimal for this example).

copiuc et al., 2003].

In our implementation, each kd-tree consists of two collections of blocks, one for leaf blocks and one for node blocks. Each block can be accessed using memory-mapped I/O via a unique block ID that is managed by the TPIE library. The structure of such a tree on the hard disk is depicted in figure 8.6; figure 8.7 shows the respective data structures in C++ notation. A `LeafBlock` simply stores surface elements of type `Point`, whereas a `NodeBlock` stores binary nodes of type `BinaryNode` as well as links `blockLinks` of type `BlockID` to other blocks—either leaf blocks or node blocks. In addition to the value and direction of the splitting axis, each `BinaryNode` has two children. Besides a reference, it stores the type of each child, which can be `BINARY_NODE`, `NODE_BLOCK` or `LEAF_BLOCK`. Depending on this type, the child reference either directly points to a node in the same block or to the block ID of another node or leaf block, which is stored in the `blockLinks` vector.

8.3.2 The Logarithmic Method

The kd-tree traditionally is a static data structure. Insertion and deletion is in principle possible in $O(\log_2 n)$ time by, first, finding the appropriate leaf, and, second, performing leaf splitting or merging operations. However, the tree quickly degenerates after multiple of such operations, deteriorating its query performance. Thus, rebalancing has to be performed very often, which, however, requires reorganization of large parts of the tree [Samet, 1990].

Therefore, the authors of the Bkd-tree propose to use the logarithmic method [Bentley, 1978; Overmars, 1987], a generic approach to make static data structures dynamic. Rather than maintaining a single tree holding n points, a forest of up to $\log_2(n/p)$ smaller kd-trees $\mathcal{T}_i$ together with a main memory buffer $\mathcal{T}_0^{\mathrm{M}}$ of size $p \ll n$ is used. The kd-trees have geometrically increasing size: each tree $\mathcal{T}_i$ stores at maximum $2^i p$ points. Changes are applied to the main memory buffer and

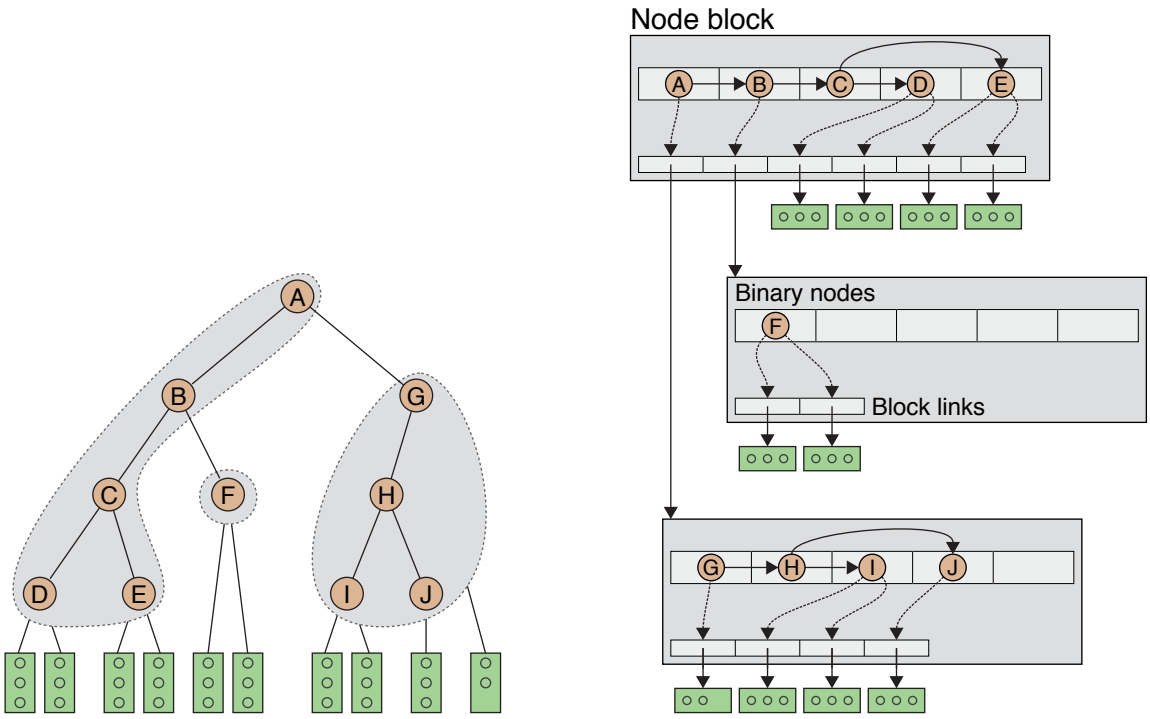


Figure 8.6: An example of how a kd-tree is blocked into ten leaf blocks and three node blocks. Leaf blocks have capacity for three points. Node blocks have capacity for five binary nodes.

occasionally written back into the forest. During a write-back, only one small tree has to be rebuilt completely. Thus, amortized insertion and deletion cost remains low, and query performance remains high. Figure 8.8 shows an example of a Bkd-tree. Operations for building, updating and querying Bkd-trees are described in detail by Procopiuc et al. [2003]. In the following sections, we give a short overview.

Bulk loading. Bulk loading builds a whole Bkd-tree from scratch from an unordered set of points. It basically generates all kd-trees of the log method using efficient kd-tree bulk loading algorithms. They are essentially out-of-core sorting algorithms optimized for best tree balance and minimum number of I/Os. With the algorithm by Procopiuc et al. [2003], a kd-tree of size n can be bulk-loaded in $O(\frac{n}{b}\log_{\frac{p}{b}}\frac{n}{b})$ I/Os.

Insert. Inserts are initially applied to the memory buffer $\mathcal{T}_0^{\mathrm{M}}$. When it becomes full, its content is written back to the forest using the following method: First, the smallest tree $\mathcal{T}_i$ that is completely empty is determined. Then, all trees $\mathcal{T}_j$ with $0 \leq j < i$ as well as $\mathcal{T}_0^{\mathrm{M}}$ are emptied by bulk-loading their content into a new tree T'. Its size is guaranteed to be smaller than the maximum possible size of

```
class BinaryNode
{
  enum {X, Y, Z, T}                                splittingAxis;
  float                                            splittingValue;
  size_type                                        leftChildIdx,
                                                   rightChildIdx;
  enum {BINARY_NODE, NODE_BLOCK, LEAF_BLOCK}       leftChildType,
                                                   rightChildType;
};

class NodeBlock
{
   vector<BinaryNode> binaryNodes;
   vector<BlockID>    blockLinks;
};

class LeafBlock
{
   vector<Point>   points;
};
```

Figure 8.7: Data structures for storing disk blocks of internal nodes and leaf nodes of the Bkd-tree.

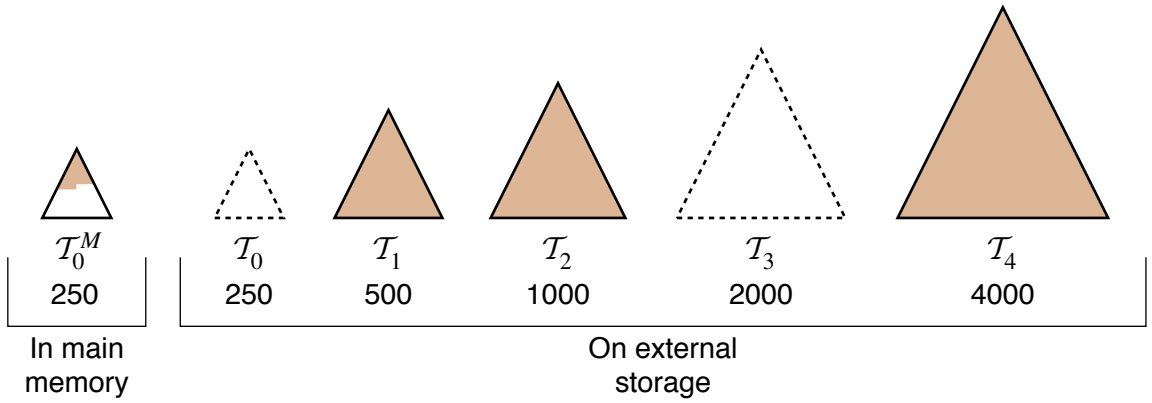


Figure 8.8: Bkd-tree example. $p = 250$ memory capacity. Three full trees are on disk, two trees are empty.

$\mathcal{T}_i$. Thus, T' can be reinserted into the forest at the lowest possible of the empty places $0 \leq j \leq i$. According to Procopiuc et al. [2003], the amortized cost of an insert operation is $O(\frac{\log_{p/b}(n/b)\log_2(n/p)}{b})$ I/Os.

Delete. Deletion is performed in a weak fashion: the point is just queried and marked as deleted. As a consequence, the Bkd-tree is not rebalanced for optimal query performance until insert operations cause again an overflow of the memory buffer. On the other hand, deletion is as fast as a single point query, requiring $O(\log_b(n/b)\log_2(n/p))$ I/Os.

Query. Window queries are performed in parallel in all $log_2(n/p)$ kd-trees as well as in the main memory buffer $\mathcal{T}_0^{\mathrm{M}}$. Due to the geometrically increasing size of the trees in the forest, its worst-case query performance is equal to the performance using one single kd-tree. For a window query returning k out of n points, $O(\sqrt{n/b}+k/b)$ I/O operations of disk blocks of size b are needed. As shown by Procopiuc et al. [2003], average performance is even much better.

8.3.3 External Storage of 3D Video Frames

In order to be independent of the length of the video, we store each frame in a separate Bkd-tree. For hyperslices that visualize a range of frames, data from multiple Bkd-trees has to be kept in main memory. Hence, in contrast to the prior description, we cannot use a separate in-memory buffer $\mathcal{T}_0^{\mathrm{M}}$ for each frame. As illustrated by figure 8.9, $\mathcal{T}_0^{\mathrm{M}}$ is also stored on disk. Instead, the editing system uses one in-memory cache as a common buffer for all Bkd-trees of currently visualized frames. The parameter p is chosen such that each $\mathcal{T}_0^{\mathrm{M}}$ exactly fits into one disk block. Typically, the cache is able to hold the complete buffers $\mathcal{T}_0^{\mathrm{M}}$ of many frames. E.g. for a typical cache size of 64 megabytes and a disk block size of 64 kilobytes, 1024 frames can be buffered. As a consequence, point insertions that do not trigger a reorganization of the kd-tree forest are effectively performed in-core.

Furthermore, we constrain the size of each Bkd-tree to completely fit into main memory. Because a whole frame has to be loaded for rendering anyway during playback, this is a reasonable assumption. As a consequence, we can speed up reorganization of the distinct kd-trees by using an in-core sorting algorithm instead of the out-of-core bulk-load method.

8.4 Multiresolution Hierarchy

The pure out-of-core representation is not yet sufficient for rendering the selected slices at interactive rates because all data still has to be streamed from disk. Therefore, we augment our data structure with a multiresolution point hierarchy. During

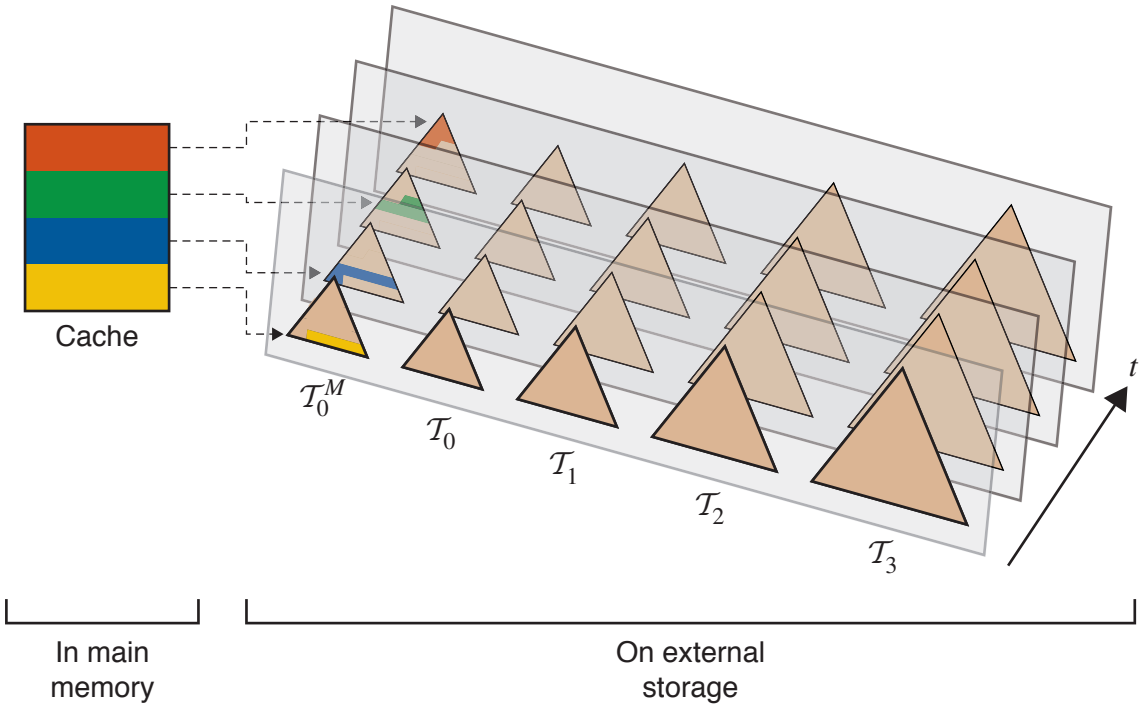


Figure 8.9: Multiple 3D video frames are stored in separate Bkd-trees. An in-memory buffer serves as a common cache for all currently accessed frames.

editing, a low-resolution model can be presented to the user instantaneously. By exploiting idle times of the hard disk, it is successively refined by a background thread running in the editing software.

Traditional multiresolution point hierarchies such as QSplat [Rusinkiewicz and Levoy, 2000] recursively merge high-resolution samples to larger points of lower resolution. All resolution levels are explicitly stored. Hence, the amount of data is increased if no compression is applied. For the sake of easy editing, compression is not desirable in our case. As the low-resolution levels are just used for preview rendering, we prefer a data structure that is inexpensive both in computational costs and storage amount. We simply subsample the high-resolution model by recursively omitting every second point. Thus, the complete multiresolution hierarchy is just an indexing scheme for the original point cloud. By presorting the points in a special way, a progressive data stream can be constructed that successively refines the model from low to high resolution. No additional data besides the original point cloud has to be stored.

The sorting procedure works as illustrated by figure 8.10. In a first step, pairs of similar points are found by constructing a balanced kd-tree with bucket size 1. Pairs are recursively simplified by deleting one of the two points. Thus, a multiresolution hierarchy of depth $l+1$ can be built by sorting the n leaves of the tree into $l+1$ sets $\Lambda^0,\ldots,\Lambda^{-l}$, where the upper negative index denotes the hierarchy level. For the lower levels $i \in \{0,\ldots,l-1\}$, they are built as

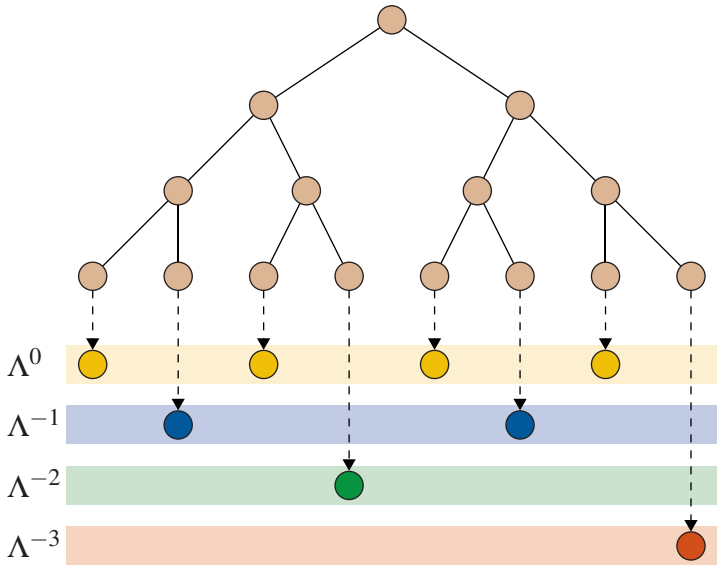


Figure 8.10: Construction of the multiresolution hierarchy. Similar point pairs are found by constructing a kd-tree. The multiresolution sets are then built by recursively assigning every second point to a higher level.

$\Lambda^{-i} = \{i + j \cdot 2^{i+1}\}, j \in \{0,\ldots,n/2^{i+1}\}$; the highest level is $\Lambda^{-l} = \{l + j \cdot 2^l\}$, $j \in \{0,\ldots,n/2^l\}$. The progressive point stream is finally built by concatenating the sets in ascending order $\Lambda^{-l},\ldots,\Lambda^0$. Note that the overall number of stored points is still the same as in the original model: $\sum_{i=1}^{l} |\Lambda^{-i}| = n$.

For displaying the model at resolution level i, where $i = 0$ is the highest resolution, all points of sets $\Lambda^{-i},\ldots,\Lambda^{-l}$ have to be rendered. Because the average sampling density is reduced by a factor of 2^i compared to the original model, the renderer increases the point size online by $2^{i/2}$.

We integrate the multiresolution representation into our out-of-core structure by constructing separate point hierarchies in all buckets of the Bkd-tree. The Bkd-tree is first built up to a bucket size of $2b$ points occupying the size of two disk blocks. Within each bucket, we then build the sets Λ^0 to Λ^{-l}. Λ^0 occupies exactly one disk block, the higher levels, however, only fill parts of a block. By merging the higher-level sets with those of the neighboring buckets, as depicted in figure 8.11, we are able to obtain a maximum space utilization.

Note that a query for a single point in this structure needs to access l leaf blocks. In contrast, the single-resolution representation only needs one block access. That drawback is amortized for practical range queries because they typically span multiple buckets whose low-resolution point sets are often stored in the same disk blocks. Moreover, there is the advantage of quick access to previews of the point cloud. Nonetheless, the height of the multiresolution hierarchy should be kept at a moderate level, typically $l = 5$.

The final data structure can be summarized as illustrated follows: There is one

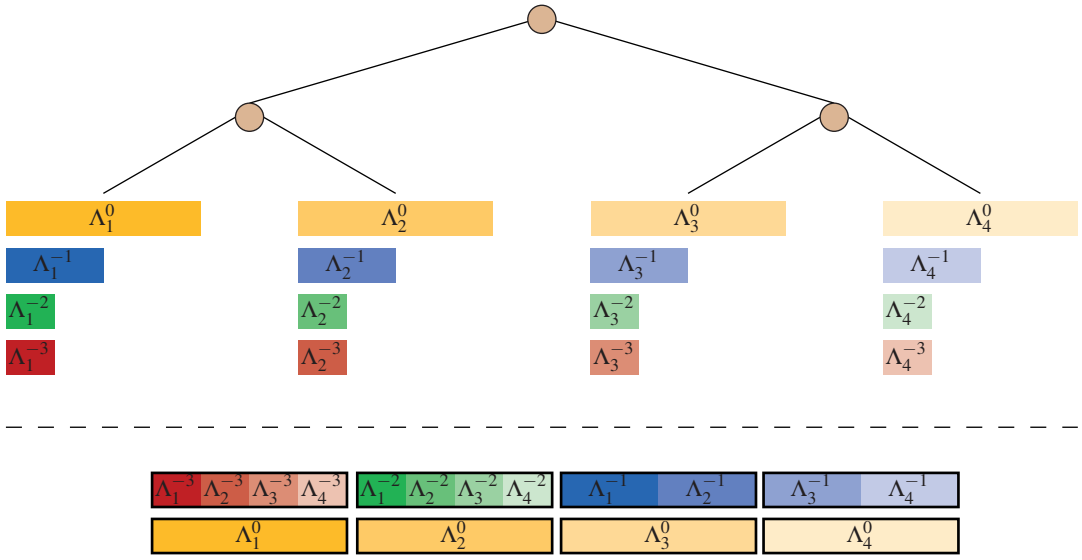


Figure 8.11: Distribution of multiresolution data into disk blocks. The point sets in each leaf of the Bkd-tree are subdivided separately into multiresolution layers (upper part). Those subsets are then sequentially written into disk blocks in order of increasing resolution.

Bkd-tree per frame. Each Bkd-tree consists of multiple out-of-core kd-trees. Each kd-tree is saved to disk as a collection of disks blocks: one for internal nodes and one for leaves. The collection for the leaves is further distributed into $l+1$ sets of the multiresolution hierarchy.

Level-of-detail access to the temporal domain exploits its regular sampling: For a resolution of level i, only each 2^ith frame is accessed. The temporal extent of a point sample is multiplied by a factor of 2^i.

8.5 Results

We evaluate our data structure with a static scene consisting of 20 identical frames with approximately 1.5 million points each. Besides position, color and tangent vectors, the samples carry additional application-specific attributes such as user-defined object tags. Thus, they occupy 64 bytes each, which sums up to a storage amount of 98 megabytes per frame. With a block size of 64 kilobytes, each disk block is able to hold exactly 1024 points. They are stored on a RAID-0 compound of two hard disks that achieves an average linear transfer rate of 97 megabytes per second with an access time of 14 milliseconds.

Our test scene is stored in a multiresolution hierarchy consisting of 6 different levels of detail. Images of those levels generated by the preview renderer of our 3D video editing software are shown in figure 8.12. The loading times for each detail

Level	Surfels	Time
5	44700	38 ms
4	45652	62 ms
3	91304	86 ms
2	182608	140 ms
1	365215	320 ms
0	729480	515 ms

Table 8.2: Time for loading the distinct levels of the LOD hierarchy levels of the scene displayed in figure 8.12.

level are given in table 8.2. As can be seen, the lowest level already provides a sufficiently detailed view and allows for interactive navigation in the video stream at 26 frames per second. By sequentially reading the complete data stream, the model is successively refined. Its full resolution of 1.5 million points is reached after 1161 milliseconds.

To test the data access performance, we ran range queries on Bkd-trees of evenly distributed random points. The tree sizes were changed from 100000 to 2 million at increments of 100000. For each measurement, we chose the query range in such a way that the result consisted of approximately 10000 points. We measured the query time for the full resolution model as well as for the first level of detail. The results are displayed in figure 8.13. In contrast to streaming of complete frames, disk access time clearly dominates because blocks cannot be read in sequential order anymore. However, interactive rates can still be achieved by progressively querying the different levels in the multiresolution hierarchy. A result from the lowest level of detail is obtained already after less than 100 milliseconds. Successive high-resolution queries do not demand for new traversal of the Bkd-tree. It is just necessary to evaluate the points in the high-resolution leaves.

We measured the update performance of our data structure by successively inserting sets of 1000 random points into an initially empty Bkd-tree. The blue graph in figure 8.14 shows insertion times for individual sets. Its oscillating behavior is due to reorganizations of trees of the logarithmic method. As can be seen, most insertions only require reorganization of the smallest kd-trees in the forest of the logarithmic method, keeping insertion time below 120 milliseconds. The peaks indicate rebalancing of larger trees. Although it only happens rarely, those situations imply a longer waiting time for the user. This could be improved in the future by temporally increasing the size of the buffer $\mathcal{T}_0^{\mathrm{M}}$ in order to perform a delayed rebalancing in a background process. On average, however, the time for inserting 1000 points is staying below 110 milliseconds, as indicated by the red curve. It shows the average of all insertions that were used to create the Bkd-tree of the given size.

Our data representation scales well with the length of the video. Because each frame is stored in a separate Bkd-tree, both query and update time remains con-

44700 points.

90352 points.

181656 points

364264 points.

729479 points

1458959 points.

Figure 8.12: Scene displayed by the preview-renderer of the 3D video editor at six different levels of detail.

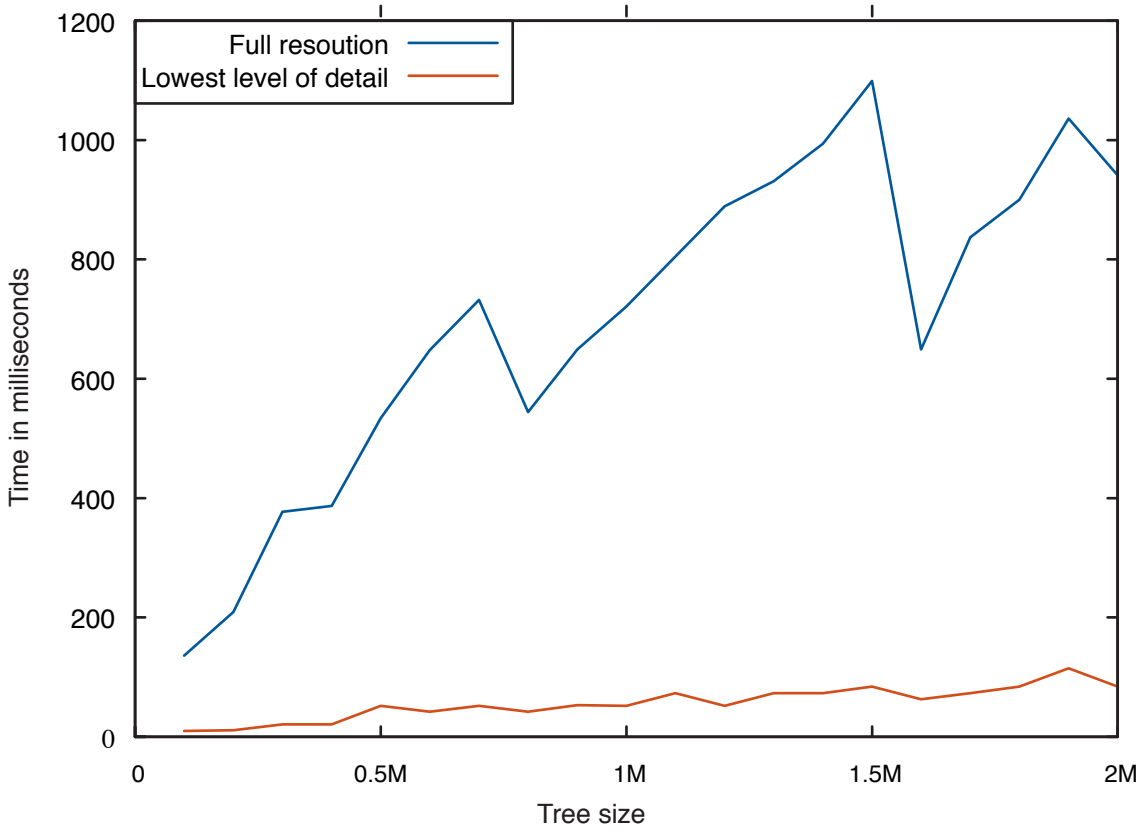


Figure 8.13: Time for window queries resulting in 10000 high-resolution points. Blue: time for complete full-resolution query. Red: time for low-resolution query.

stant. Queries that include several trees—i.e. slices that are not orthogonal to the t-axis—would have to access more trees. Search time for such slices would therefore approximately increase linearly with the number of frames. Because a user typically only edits a small number of successive frames at the same time, the number of frames visualized by those slices is bound with an adjustable parameter in order to prevent access to all frames over the full length of the video.

8.6 Conclusion

We presented the video hypervolume as a representation for 3D video data sets in four-dimensional spacetime . Consisting of point samples, it nicely fits into existing frameworks for processing point-based geometry. With its unification of space and time, processing operators are able to exploit and maintain coherence in all dimensions. The concept of hyperslicing extracts three-dimensional subvolumes for visualization and manipulation. Editing of large videos is possible using our dynamic out-of-core storage method. The integrated multiresolution hierarchy

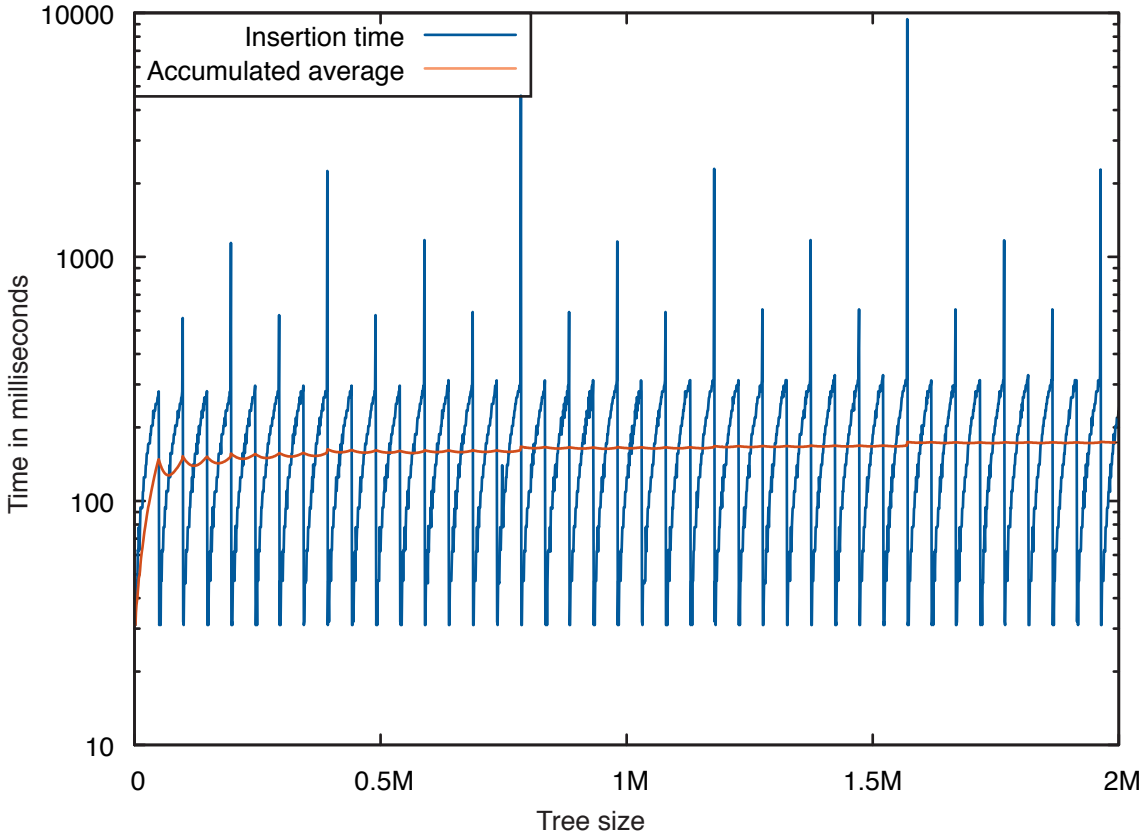


Figure 8.14: Time for inserting blocks of 1000 points into a Bkd-tree of a given size. Blue: Insertion time for a single block. Red: Accumulated average of previous insertion times.

achieves interactivity. The video hypervolume builds the basis for our nonlinear 3D video editing system presented in the following chapter.

Chapter 9

Interactive 3D Video Editing

In this chapter, we present a generic framework for interactive editing of 3D video footage. It extends existing concepts for two-dimensional video and combines their conceptual simplicity with the power of depth-enhanced video data. The multi-dimensional, spatio-temporal nature of 3D video leaves its editing highly non-trivial, but, at the same time, allows for a variety of novel features.

Our framework is based on explicit 3D geometry providing a view-independent scene model. Its point-sampled representation is independent of the used acquisition system as long as it captures depth information. It can be generated easily from depth maps using the approach presented in chapter 5. Built upon the video hypervolume of chapter 8 for representing the spatio-temporal video stream, our system allows for intuitive handling and editing of the four-dimensional domain.

We designed a concept for video editing which is based on three fundamental operators: slicing, selection, and editing. In particular, we present a 4D object selection algorithm based on graph cuts. To convey object boundaries, the user indicates object and non-object regions in the spatio-temporal domain by painting on the surfaces with a 3D paintbrush. In addition, we provide a set of spatio-temporal editing operations, such as cut & paste and affine transformations. By using the operators, processing of 3D video becomes easy and intuitive.

9.1 System Overview

Our system complements the 3D video acquisition and reconstruction pipeline with an editing framework for post-production as illustrated in figure 9.1. It is based on the four-dimensional video hypervolume which represents appearance and geometry of the scene as point samples in spacetime.

The editing framework is based on three operators: slicing, selection and edit-

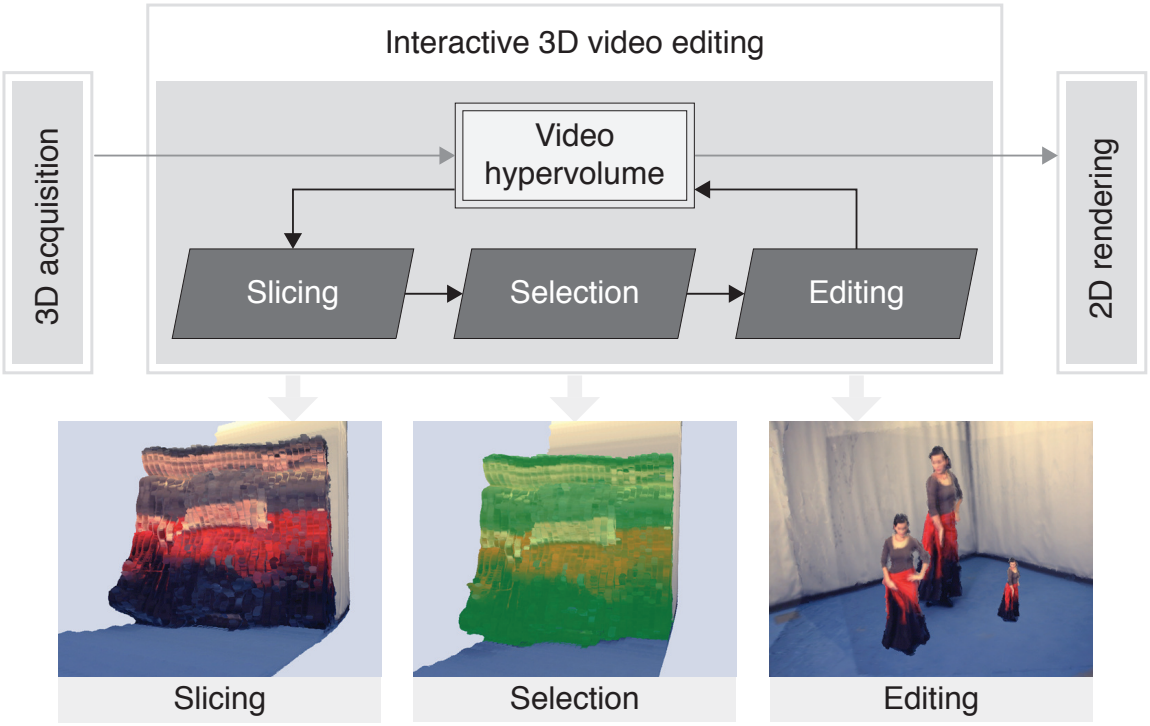


Figure 9.1: The 3D video editing framework.

Figure 9.2: Our interactive 3D video editing combines the advantages of 2D video editing with depth-enhanced 3D video streams. From left to right: Interpolated view of the 3D video input data; cutout of a 3D video object; composite 3D video with additional 2D and 3D objects, new background and shadow mapping.

ing. The slicing operator, which has already been introduced in section 8.2, provides an intuitive interface to interact with the four-dimensional domain. It transforms selected parts of the 4D data set from the video hypervolume to a cloud of 3D point samples. Slice orientation and position can be changed interactively (section 9.2). With the selection operator (section 9.3) the user can mark regions or objects of interest. While region selection can be performed using marquee, lasso or paintbrush tools, object selection requires the notion of boundaries which we introduce using a graph cut selection scheme. Users guide the selection process by

painting with an object brush and with a background brush. All selected parts can be modified by a set of editing operators (section 9.4). Operations make use of the explicitly modeled scene geometry and include cut & paste, spatial and temporal translations, rotations and scaling. During compositing, handling of occlusions is provided for free.

Our unified handling of space and time naturally supports editing operations exploiting both spatial and temporal coherence. Selection and editing are applied directly on a cloud of 3D point samples yielding from the slicing operation. The invisible, fourth domain can only be accessed by defining a different slice. Upon completion of an editing operation the data in the current slice is backpropagated into the video hypervolume. By operating on the slice only, we leverage interactive editing of the huge 3D video data sets. A typical editing session is illustrated in figure 9.2.

Interactive visualization and editing of the video hypervolume requires data structures providing efficient access to the samples. We implemented a two-level approach that relates to the general structure of our editing framework. The first level of the data structure represents the entire four-dimensional video volume. It is directly accessed from hard disk using our out-of-core representation presented in chapter 8. As typical access patterns do not select single points but whole subvolumes, the combination of spatial Bkd-trees and regular temporal sampling has proved to be very efficient. Moreover, the structure supports dynamic updates if the user adds, removes or transforms points during the editing session.

The editing itself only takes place in the 3D projection of a selected hyperslice. For efficient rendering, the 3D points are stored as vertex arrays in main memory. Editing operations typically need fast access to single points. Kd-trees are very efficient and widely used for that purpose in traditional point processing frameworks [Zwicker et al., 2002a]. In our implementation, we build and update a kd-tree on the fly as soon as a query for a specific point is performed. The kd-tree can be represented just as a reordering of the vertex arrays. Thus, no additional storage is needed.

9.2 Navigation

The slicing operator introduced in section 8.2 is used to navigate in the video hypervolume. In the most common case, the slice is orthogonal to the t-axis and corresponds to a single 3D video frame. Orientation of the 3D point cloud can be controlled interactively using an arcball interface. The user can select a specific frame using a slider to control the slice position b_H in time. Moreover, he can adjust its thickness Δb_H by defining in and out points—quite similar to 2D video processing—resulting in multiple frames getting displayed. This easily allows to identify static and dynamic scene parts.

For spatio-temporal editing it is also interesting to visualize the time domain on

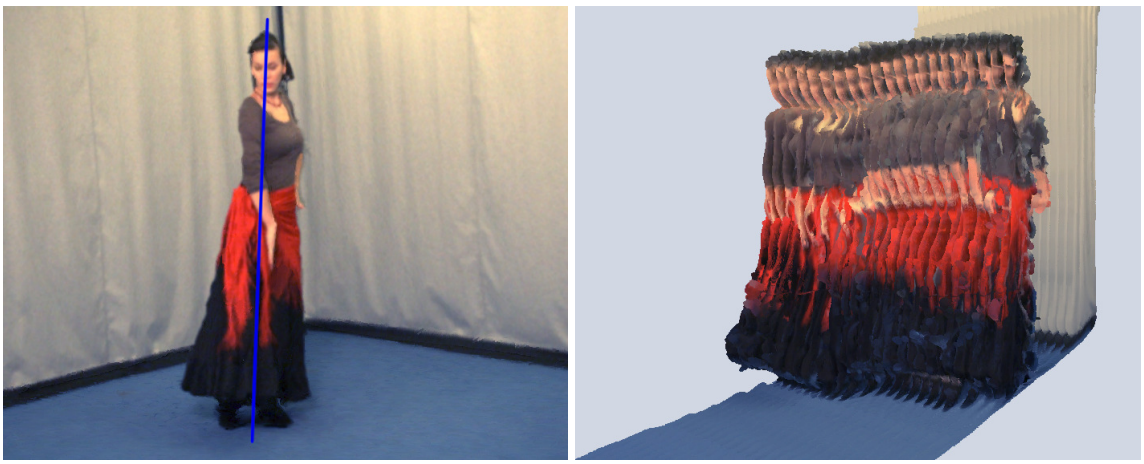


Figure 9.3: For defining a new hyperslice, the user draws a blue line onto the screen that represents the projection of its center plane (left). In this example, the new hyperslice reveals the time domain, allowing for easy selection and editing of objects over multiple frames.

the screen. This facilitates intuitive spatio-temporal selection as described in the next section. The user can define arbitrarily oriented slices by drawing a line on the screen representing the hyperplane. This conveniently allows to generate slices through a specific object of interest, as can be seen in figure 9.3. The slider now generally controls the movement of the slice through the video hypervolume. The slice thickness can be increased such that a greater part of the orthogonal, fourth dimension gets projected onto the screen. When the user defines the new slice, the system automatically computes its rotation matrix $\dot{\mathbf{R}}$ by determining its local coordinate system according to the drawn line and the current view. Figure 9.4 shows the vectors $\dot{\mathbf{V}}_1$, $\dot{\mathbf{V}}_2$ and $\dot{\mathbf{V}}_3$ which are all constructed within the current hyperslice. $\dot{\mathbf{V}}_1$ and $\dot{\mathbf{V}}_2$ are located in the current image plane, $\dot{\mathbf{V}}_2$ is orthogonal to the drawn line and defines the normal of the new slice. $\dot{\mathbf{V}}_3$ is orthogonal to the image plane. $\dot{\mathbf{V}}_4$ is not depicted as it is orthogonal to the current hyperslice.

9.3 Selection

The selection operator is the key to subsequent 3D video editing tasks. The 4D data does not feature object labels indicating conceptually connected data samples. For this purpose, our framework provides an algorithm based on graph cuts to associate such labels for further editing operations. But first we introduce some basic selection tools.

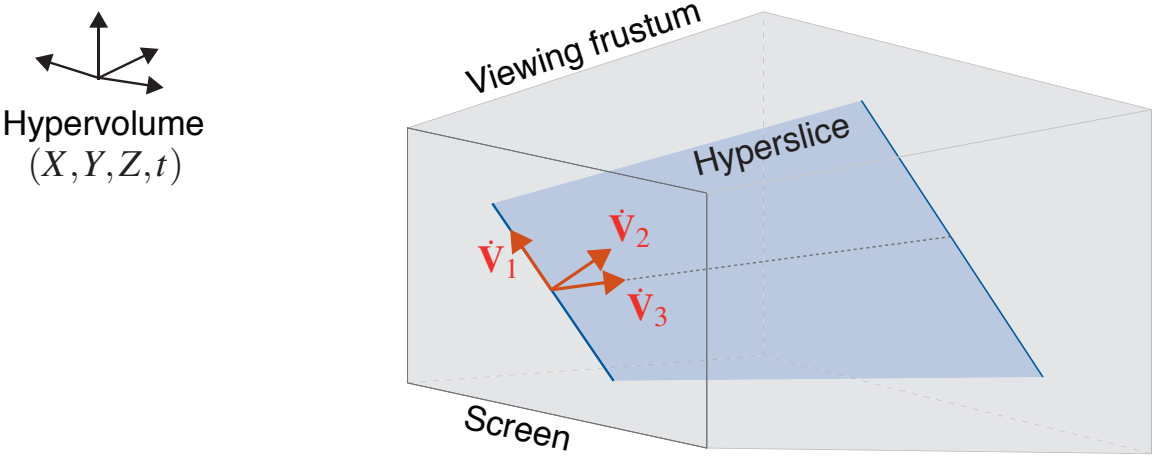


Figure 9.4: Non-orthogonal hyperslices are generated by calculating the required rotation matrix simply from a line drawn onto the screen.

9.3.1 Region Selection Tools

The user can view and select objects both in space and along trajectories in time using the slicing operator. By taking advantage of the underlying 3D geometry, accurate selection of objects and regions of interest is sometimes already possible by using basic selection tools.

Marquee and lasso selection tools. Similar to 2D photo editing applications our framework provides marquee and lasso selection tools. With these tools users are intuitively able to select large areas of the visualized slice. Users draw 2D regions on the screen which get extruded into the slice domain for 3D selection, using the current virtual camera parameters. In this way a whole subvolume and possibly hidden surfaces are selected. However, by rotating the viewed data, the user easily sees where hidden surfaces are selected and can work on the selection using different selection modes. Our framework provides addition, difference and intersection modes.

3D paintbrush. Another selection tool is the paintbrush also known from 2D photo and video editing applications to paint selections or colors. However, due to the additional dimension, we have to define a 3D footprint of the paintbrush. Intuitively we define the 3D paintbrush as a spherical volume in 3D. We determine the 3D center point by calculating the 3D position of the front surface data sample at the 2D screen space coordinate of the mouse pointer. 3D data points are selected if they are contained in the spherical volume around the center point and are determined using a range query in the underlying kd-tree structure. We calculate the 3D center point with help of the z-buffer. By considering z-values

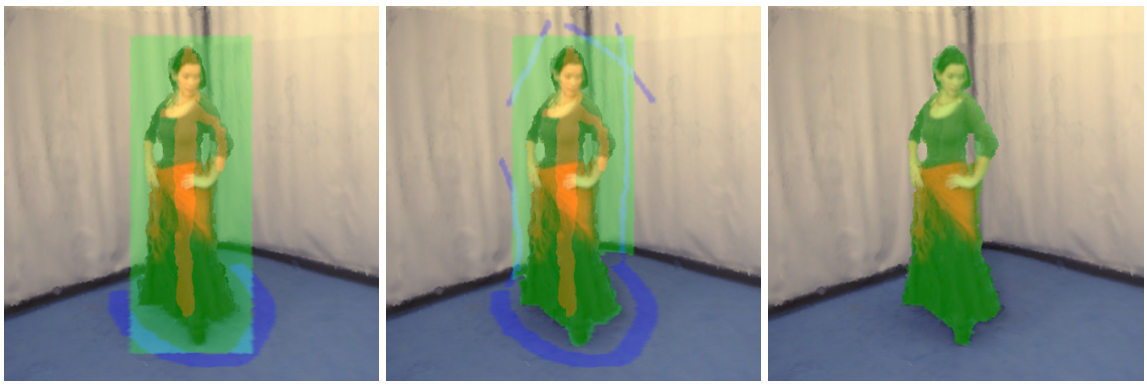


Figure 9.5: Object selection. Left: The user wants to select a person and marks her with red paint, the floor with blue paint and the region of interest with a rectangular marquee selection. Middle: The first invocation of the min-cut optimization yields wrongly marked samples on the wall behind the person. Right: After specifying more paintbrush strokes, the optimization completes with a satisfying selection.

of all pixels within the screen space footprint of the projected spherical volume, the intersection point of the picking ray with the scene can be determined very robustly. The user can chose the depth of the sphere as either determined by the nearest z-value or by a median filter over all z-values. The former can be used for selecting small surface patches in front of a bigger surface without the need to exactly click on them. The latter can be employed for selecting the densest surface without considering small surface patches.

9.3.2 Object Selection

The captured 4D data set does not supply the user with object boundaries or labels. However, for complex editing tasks, such a labeling is essential. Therefore, we introduce a graph cut algorithm to associate such labels. We use the 3D paintbrush and the marquee selection tools introduced in the previous section to specify the necessary constraints. With the paintbrush, the user can mark surface patches which should be selected (red paint) and patches which should not be selected (blue paint). The marquee selection tools (green paint) are used to define a spatio-temporal region of interest, thereby excluding large uninteresting regions and speeding up the min-cut optimization significantly. Performing the optimization on whole 3D video data streams is not feasible interactively.

The status of all data samples marked with green paint is then determined by invoking a min-cut optimization after hitting a button in the interface. Upon completion, the user can refine his markings using the region selection tools and run the optimization again. Figure 9.5 illustrates the object selection operator.

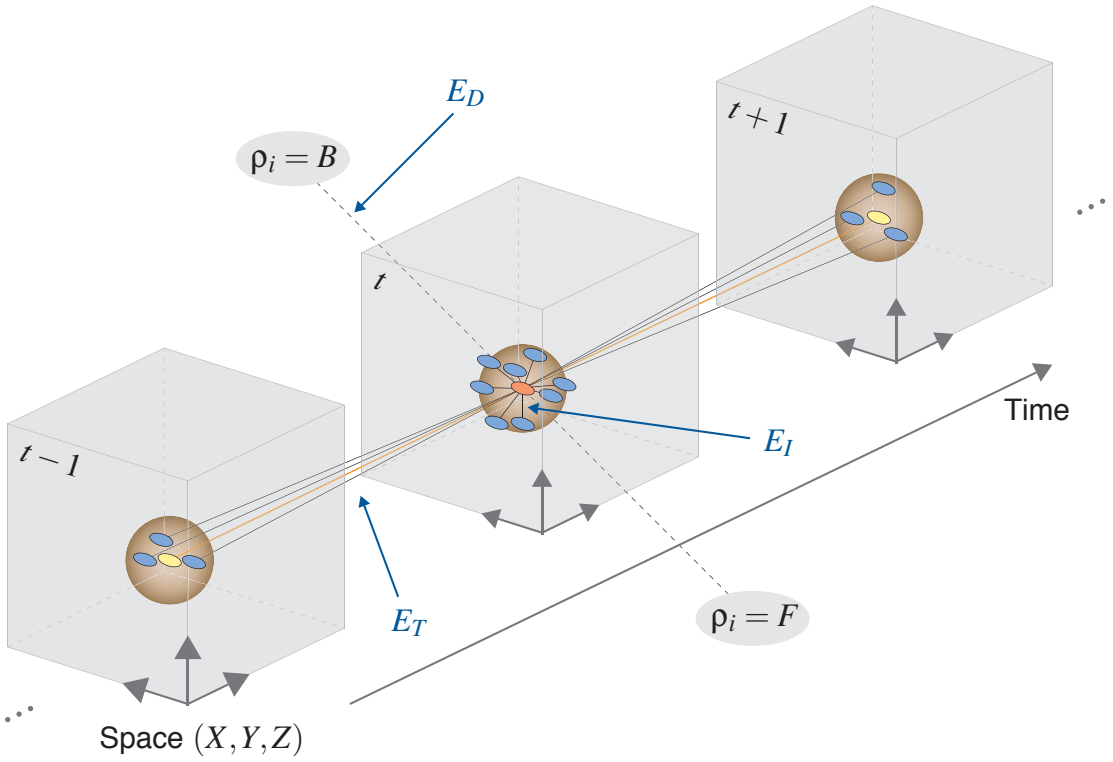


Figure 9.6: When constructing the 4D graph, any data sample or data region contributes to the graph according to intra-frame and inter-frame neighborhoods and energies, as well as to virtual nodes with data energies.

Graph Construction

The object selection problem is very similar to the video segmentation problem of section 6.4.1 but directly operates on the irregular four-dimensional point cloud. It can be interpreted as a graph labeling problem. Each data sample i is assigned a unique label $\alpha_i \in \{0,1\}$ where 1 means that the data sample belongs to the selection and 0 that it does not belong to the selection. We construct a 4D graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ on the 4D hypervolume bounded by the region of interest. The node set $\mathcal{V}$ contains all data samples that have been defined as the spatio-temporal region of interest. A node i represents a data sample with label α_i, color c_i, surface normal $\mathbf{N}_i$, as well as a user-assigned label $\rho_i \in \{F,B,U\}$. The latter is assigned depending on the input brush strokes: nodes marked with the foreground or background brush are tagged as $\rho_i = F$ or $\rho_i = B$, respectively, unmarked nodes are tagged as $\rho_i = U$. Furthermore, we define Λ_t as the scene at a time instant t. Figure 9.6 illustrates the 4D graph construction.

We construct the intra-frame arcs $\mathcal{E}_I$ by connecting spatially adjacent data samples in the same time instant Λ_t. The data samples are irregularly sampled in spacetime and do not feature connectivity. Hence, we have to calculate the spatially adjacent samples by using range queries—quite contrary to the 2D video

cutout of section 6.4.1 which uses explicit neighborhoods on the pixel grid. We apply a 3D kd-tree for this purpose and generate arcs for all data samples which lie inside a sphere with given radius (orange sphere in figure 9.6), typically 2 cm in our metric environment. We take the nearest k data samples and exclude evidently unrelated samples with mean color or normal differences over a certain threshold (usually 10%).

Inter-frame arcs $\mathcal{E}_T$ connect temporally and spatially adjacent point samples in adjacent time instants $\Lambda_{t\pm1}$ that are located within a given 3D radius, typically 4 cm. We also use a kd-tree for this purpose, created in the corresponding time instants $t \pm 1$. We initialize the range query by projecting the data sample (orange point in figure 9.6) from time t to $t \pm 1$ (yellow points in figure 9.6). Note that the radius has to be higher than for the intra-frame arcs due to non-regular sampling and motion. Furthermore, we only take the $k/2$ nearest data samples and exclude unrelated data samples too. In our current implementation we set k to 8.

4D Graph Cut Optimization

Similar to the 2D video cutout of section 6.4.1, we define a cost function E on the constructed graph $\mathcal{G}$. The 4D graph cut algorithm then solves the object labeling problem by minimizing the following energy:

$$\begin{aligned} E_A(\{\alpha_i\}) = \lambda_{\mathrm{D}} \sum_{i \in \mathcal{V}} E_{\mathrm{D}}(\alpha_i) \\ + \lambda_I \sum_{(i,j) \in \mathcal{E}_I} E_I(\alpha_i, \alpha_j) \\ + \lambda_T \sum_{(i,j) \in \mathcal{E}_T} E_T(\alpha_i, \alpha_j) \end{aligned} \tag{9.1}$$

The optimization assigns labels α_i for each data sample i represented by the graph nodes. Figure 9.6 illustrates the different terms of the energy function. E_{D} is the likelihood energy while E_{S} and E_T are the prior energies. E_{D} measures the similarity of the color of a data sample to the color models assembled from the user-assigned labels. E_{S} and E_T assess the color and geometry differences between spatially and temporally adjacent samples. They penalize strong color and normal deviations and ensure spatial and temporal coherence in the selection process. Consistency of point positions is assured implicitly by the graph arcs because they only connect points in local neighborhoods. We employ the max-flow algorithm by Boykov and Kolmogorov [2004] to minimize the energy $E_A(\{\alpha_i\})$ in equation (9.1).

Likelihood energy. We assemble two Gaussian mixture models by sampling the color c_i of the data samples with user-assigned labels. One model Θ_F is built for the "foreground" samples with $\rho_i = F$ and one for the "background" samples

with $\rho_i = B$. The likelihood energy $E_{\mathrm{D}}(\alpha_i)$ can then be defined similar to equation (6.19) as:

$$\begin{array}{c|ccc} E_{\mathrm{D}}(\alpha_i) & \rho_i = F & \rho_i = B & \rho_i = U \\ \hline \alpha_i = 1 & 0 & \infty & \frac{\Theta_F(c_i)}{\Theta_F(c_i)+\Theta_B(c_i)} \\ \alpha_i = 0 & \infty & 0 & \frac{\Theta_B(c_i)}{\Theta_F(c_i)+\Theta_B(c_i)} \end{array} , \tag{9.2}$$

where the color models are equivalent to those of section 6.4.1, e.g.

$$\Theta_F(c) = -\log \sum_{j=1}^{k} \nu_j N_3(c; \mu_{Fj}, \mathrm{V}_{Fj}) \tag{9.3}$$

using k Gaussian normal distributions $N_3(c; \mu_{Fj}, \mathrm{V}_{Fj}$ with mean values μ_{Fj}, covariance matrices V_{Fj}, and weights ν_j. We use $k = 5$ Gaussians which provides satisfying results in our editing framework.

Prior energies. The prior energies E_I and E_T are both defined in a similar fashion as

$$E(\alpha_i, \alpha_j) = (1 - \delta_{\alpha_i,\alpha_j}) S_{i,j} \tag{9.4}$$

with

$$\delta_{a,b} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}. \tag{9.5}$$

The $S_{i,j}$ are the intra- and inter-frame link costs, respectively, that are assigned to the actual graph edges. We adopt the global link costs from Wang et al. [2005] and extend them by geometry information. Besides considering color differences we weight the normal differences between adjacent data samples in a similar way. The link costs are defines as:

$$S_{i,j} = e^{(-\Delta_C^2/(2\beta_C^2))} + e^{(-\Delta_N^2/(2\beta_N^2))} \tag{9.6}$$

Δ defines the color or normal difference between two nodes i and j which the link connects. The β represent the intra-frame and inter-frame variance of colors or normals. In our current implementation we do not calculate the variances but empirically set them to $\beta_C = 0.08$ and $\beta_N = 0.12$.

9.4 Editing

Editing operations are leveraged using the slicing and selection operators described in the previous sections. Our supported set of editing operations is simple yet becomes very powerful in our framework and with the underlying 4D representation.

Cut & paste. After slicing and selection, the user can employ a clipboard to perform cut or copy operators for selected regions or objects. The data in the clipboard can then be used to paste objects to other hyperslices, other scenes or to clone objects. Compositing of multiple scenes can be done conveniently by first loading all the scenes together at different places in the video hypervolume and then moving their objects around. Objects can be easily removed without leaving holes in the background scene if the acquisition system was able to capture the background behind the object from a suitable viewing angle. Note that compositing in the spatial domain becomes very convenient because our representation explicitly stores the scene geometry.

Transformations. We can apply arbitrary affine transformations to the selection. Transformations are straightforward and intuitive in the case of a hyperslice orthogonal to the t-axis. On the other hand, by using other hyperslices we conveniently perform translations in time. To this end, the user simply generates a hyperslice non-orthogonal to the time axis. The translation operator nicely shows the possibilities of a uniform spacetime representation. Other transformation operators include rotation and scaling. The user can rotate objects freely in spacetime, even from the temporal into the spatial domain, creating interesting novel effects like visualization of movement trajectories. Note that in all cases, occlusions are correctly resolved for free by our explicit 3D geometry.

Compositing and shadow mapping. We provide various compositing operators with other media, e.g. images, videos, and virtual objects. They can be inserted into the video hypervolume by conversion to point samples. Alternatively, we allow for insertion of textured meshes during the final rendering phase. To seamlessly blend objects with new backgrounds we adopted a shadow mapping technique [Williams, 1978] to cast shadows of inserted objects into the new background. Again, this is leveraged by the underlying explicit 3D geometry. More realistic compositing can be achieved by adapting the scene's illumination conditions. However, for this purpose time-varying reflectance properties of the scene need to be calculated. This is an interesting challenge for future work.

9.5 Results

We recorded a number of 3D videos and performed editing tasks on the 4D data. The input consists of sequences with a flamenco dancer, an actor juggling a ball and a shot with a plant, a sofa and a sitting person. All sequences have been captured with our acquisition system using four 3D video bricks covering a convex horizontal viewing range of 71 degrees around a $2.8 \times 3.2 \times 1.9\,\mathrm{m}^3$ scene. They were recorded at 12 fps and their length was between 120 and 300 frames. Further

details can be found in appendix A. The interactive editing session took approximately one day to complete. To generate appealing videos, our editing system allows for content and viewpoint trajectory scripting. The final images have been rendered using our point splatter of chapter 5.

Figure 9.7 shows a scene with the flamenco dancer. The dancer was cut out of the original background and inserted into a new one. We used the object selection operator for this purpose. The sequence shows the generation of a "clone" in the same scene and subsequent scaling and transformation to the sofa. Shadow mapping and matting ensures that the dancer still blends in with the new background. Note the shadow in the fifth image which nicely shows the underlying geometry with the cast shadow of the small dancer on the sofa. The poster is also inserted onto the wall using the media import feature of our editor. Figure 9.8 shows an edited 3D video of the juggle sequence inserted into the environment with the sofa and the person. The plant shows the limitation of the employed 3D capturing system. Thin structures cannot be handled yet and the resulting geometry is not captured well. Nevertheless, unstructured lumigraph rendering reduces the resulting artifacts. In this sequence we also replaced the ball with a teapot. The trajectory was captured by cutting out the ball and calculating its center of gravity. We generated spin artificially since we could not capture this from the video footage. Figure 9.9 combines most of the editing operators in one shot. We placed a video trailer and the Pacific Graphics 2006 logo onto the walls.

9.6 Conclusion

We have demonstrated a system for interactive editing of 3D video footage. Based on our four-dimensional video hypervolume representation, it allows for unified handling of space and time. Using a three-staged processing loop, we support various editing tasks for post-production of 3D video. For future work we would like to improve our representation by explicitly modeling time coherence [Vedula et al., 1999]. To speed up the graph cut object selection, mean-shift presegmentation could be employed. Although the presented editing operators allow for the most common editing tasks, others can be envisioned, e.g. altering the motion of actors or retargeting of motion from one actor to another [Cheung et al., 2004]. In addition, illumination adaptation needs to be solved for application in productive environments. Finally, our system is still limited by the employed 3D video capturing system. Future development in depth video scanning systems will directly improve the resulting image quality.

Figure 9.7: The flamenco dancer is inserted into a new environment and cloned. Shadow mapping is applied for seamless blending with the scene.

Figure 9.8: The juggling actor is placed into a new environment. The ball can be replaced by a virtual object following the same trajectory.

Figure 9.9: The Pacific Graphics 2006 logo and a video trailer are placed onto the walls.

Chapter 10

Conclusion

In this thesis, we investigated various parts of a typical 3D video production pipeline, including acquisition, representation, editing, and display. Our focus lies in the 3D video data representations. While image space representations show their superiority in achievable rendering quality due to their regular structure, point-based representations with explicit geometry provide more flexibility for interactive applications beyond pure playback. We have demonstrated this with various algorithms and applications based on point-sampled geometry, such as streaming and compression, dynamic out-of-core access, spatio-temporal editing, and probabilistic image generation. Point samples can be easily constructed from color and depth images which we acquire with our modular system of 3D video bricks based on stereo vision on structured light. Together with our nonlinear editing system, point-sampled 3D video provides a novel tool for video post-production, yielding novel possibilities for special effects that would be difficult or impossible to achieve with conventional two-dimensional imaging technologies.

10.1 Review of Contributions

The main contributions of our work can be summarized as follows:

Modular 3D video acquisition system. We presented a modular 3D video acquisition system based on the concept of portable 3D video bricks. They acquire depth maps of the scene using spacetime stereo vision on structured light. An improved sliding window stereo matching algorithm achieves robustness at depth discontinuities. Specialized camera projector synchronization schemes permit simultaneous acquisition of color textures. The 3D video bricks are arranged in a scalable setup, allowing recording of full 360 degree views with a sparse arrangement of acquisition devices.

Probabilistic, point-based 3D video representation. We proposed a view-independent 3D video representation based on point samples that are generated by merging the depth information of all 3D video bricks into a common world reference frame. By applying a photo consistency check, we are able to remove outliers. The points describe a statistical model of the acquisition error, which is used in our probabilistic, view-dependent rendering technique to smooth geometric noise during image generation.

Image-based 3D video representation. As a second data model, we introduced 3D video billboard clouds as a view-dependent, image space representation based on displacement-mapped billboards. Being defined in the disparity space of the acquisition system, they provide a uniform model of reconstruction noise. Together with their regular sampling, they allow for efficient geometry filtering using signal processing operators. The presented spatio-temporal bilateral filter is able to successfully smooth quantization noise and outliers, and compensates for calibration errors in overlapping regions of different scans. Thus, the billboards can be used as an intermediate representation for efficient post-processing, and can be converted to point samples afterwards. Alternatively, images from novel viewpoints can be generated directly using our proposed rendering scheme. Our novel view-dependent geometry blending technique combined with projective texturing achieves high-quality output images.

Volumetric 3D video representation. To enable spatially and temporally consistent processing and editing of 3D video, we extended the point-sampled model to a for-dimensional volumetric representation, the so-called video hypervolume. As a fundamental operation, we introduced hyperslicing to select three-dimensional subspaces. By varying the orientation of the hyperslice, both spatial and temporal aspects of the video can be visualized and manipulated.

Progressive compression scheme for point-sampled models. For efficient storage and transmission of scanned three-dimensional scenes, we presented a generic framework for compression of point-sampled models. By exploiting local coherence in the point cloud, various point attributes can be stored efficiently in a progressive data stream containing a compact, multiresolution representation of the scene. In particular, we proposed efficient coding operators for point positions, surface normals, and colors, allowing for compression of geometry and appearance in a unified manner. Although being initially developed for static models, we discussed possible extensions of the compression algorithm towards dynamic point clouds.

Out-of-core data management. 3D video editing requires fast random access to large data sets. Therefore, we proposed an out-or-core storage scheme for our

video hypervolume representation. Based on external-memory Bkd-trees, it allows for efficient queries and for dynamic modifications of the point data with a limited amount of main memory. Combined with our multiresolution representation, editing of large data sets is possible at interactive rates.

3D video editing system. Based on the video hypervolume, we implemented a system for nonlinear 3D video editing, consisting of a pipeline of three fundamental operators: hyperslicing, selection, and editing. For complex selections of surfaces or objects of the 3D video, we presented a semi-automatic segmentation tool based on global graph cut optimization. In contrast to 2D video segmentation, we are able to improve the robustness of the optimization by using not only colors but also the available geometry information. Together with a set of manipulation tools in the editing stage, we demonstrated the suitability of 3D video for video post-production and generation of novel visual effects.

10.2 Outlook

In this thesis, we presented solutions to various problems in the fields of 3D video. Nonetheless, there is still room for improvement. In the following, we point out possible directions for future research.

High-quality 3D video acquisition. Achieving an image quality similar to today's 2D video or cinema is certainly the ultimate research goal. Although we have shown that quality is highly dependent on the employed representations and rendering algorithms, it is limited by the accuracy of the input data, especially of the depth reconstruction. In the near future, time-of-flight cameras may be a possible solution to that problem. They appear to achieve more robust results than traditional computer vision algorithms. On the other hand, they suffer from different limitations such as low resolution. Hence, novel algorithms to process that kind of data have to be developed.

Explicit temporal coherence. Our video hypervolume implicitly models temporal coherence by assuming smooth, slow movements of objects. However, as motions are often fast in relation to the sparse temporal sampling, algorithms for video post-processing, compression, and editing would largely benefit of having explicit correspondence information over successive frames, similar to motion vectors used in 2D video coding. Feature tracking or scene-flow algorithms can provide that information. However, robust results cannot be achieved easily. Those algorithms are still under heavy research in the field of computer vision. Another promising approach that reconstructs a spatio-temporal hypersurface has been recently proposed by Goldlücke et al. [2007].

Standardized coding formats. To bring 3D video to the masses, a compact data representation is necessary, both for storage and for streaming over networks. Solutions like ours developed in academic research are a first step. However, to achieve broad acceptance, a common industry standard is necessary, similar to existing standard formats for conventional digital video. The MPEG committee is currently investigating coding of multi-view video data. While this may be sufficient for pure playback, novel interaction metaphors may require more flexible representations that support explicit geometry information.

Interaction and editing. In our work we have shown that 3D video in principle provides novel possibilities besides virtual playback, for example simple replacement of video objects. With hyperslicing we also introduced a first novel interaction metaphor suitable for our needs. As a next step, concrete novel applications should be identified. For example one could study the cases in which creation of visual effects for 2D videos could benefit from available geometry. There, the right trade-off between simplicity of editing and the additional acquisition effort has to be found. Moreover, novel special effects could be crated that would not be possible with conventional methods. Furthermore, 3D video may be used as a completely new media type for entertainment applications, combining realistic image quality of movies with interaction possibilities of today's computer games.

Appendix A

3D Video Data Sets

This chapter provides an overview over the different 3D video acquisition setups and 3D video data sets that have been recorded for this theses.

A.1 Acquisition Setups

A.1.1 Three-Brick Setup

Figure A.1 shows a schema of our first acquisition setup. Three 3D video bricks cover a $2.4 \times 2.5 \times 2.0\,\mathrm{m}^3$ working volume in a corner of a room at a viewing range of 61° horizontally and 40° vertically. The setup was used to record the taekwondo data set.

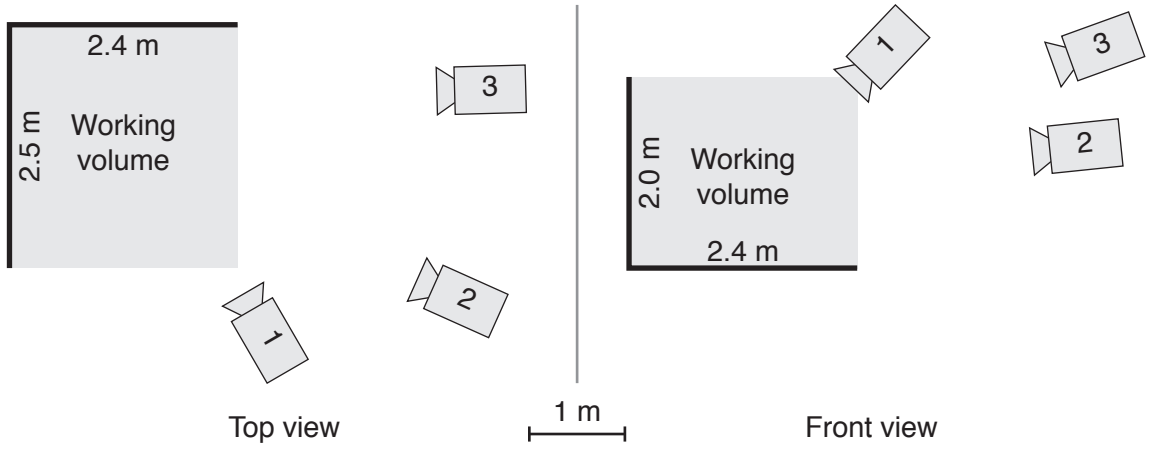


Figure A.1: Acquisition setup with three 3D video bricks.

A.1.2 Four-Brick Setup

Figure A.2 shows a schema of our second acquisition setup. Four 3D video bricks cover a $2.8 \times 3.2 \times 1.9\,\mathrm{m}^3$ working volume in a corner of a room at a viewing range of 71° horizontally and 51° vertically. The setup was used to record the flamenco, juggle, and sofa data sets.

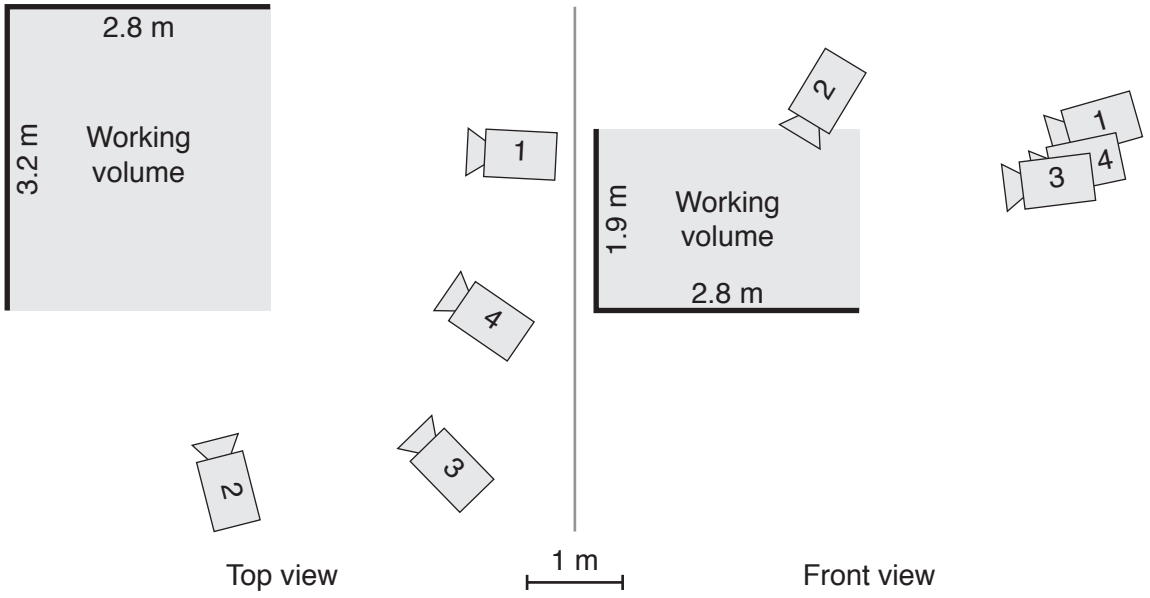


Figure A.2: Acquisition setup with four 3D video bricks.

A.2 Data Sets

A.2.1 Taekwondo

Figure A.3 shows sample images of the taekwondo sequence. It has been captured using our three-brick setup described in section A.1.1. Technical details are given in table A.1.

Figure A.3: Color camera images of the taekwondo sequence, captured by three 3D video bricks.

Duration	10 s
Frames	100
Frame rate	10 fps
Camera resolution	1024×768 pixels
Input viewpoints	3
Horizontal viewing range	61°
Vertical viewing range	40°
Camera projector synchronization mode	inverse pattern projection
Points per 3D video frame (average)	1.21 million
Overall 3D video data size (positions, colors, covariance matrices)	4.4 GBytes

Table A.1: Technical details of the taekwondo sequence.

A.2.2 Flamenco

Figure A.4 shows sample images of the flamenco sequence. It has been captured using our four-brick setup described in section A.1.2. Technical details are given in table A.2.

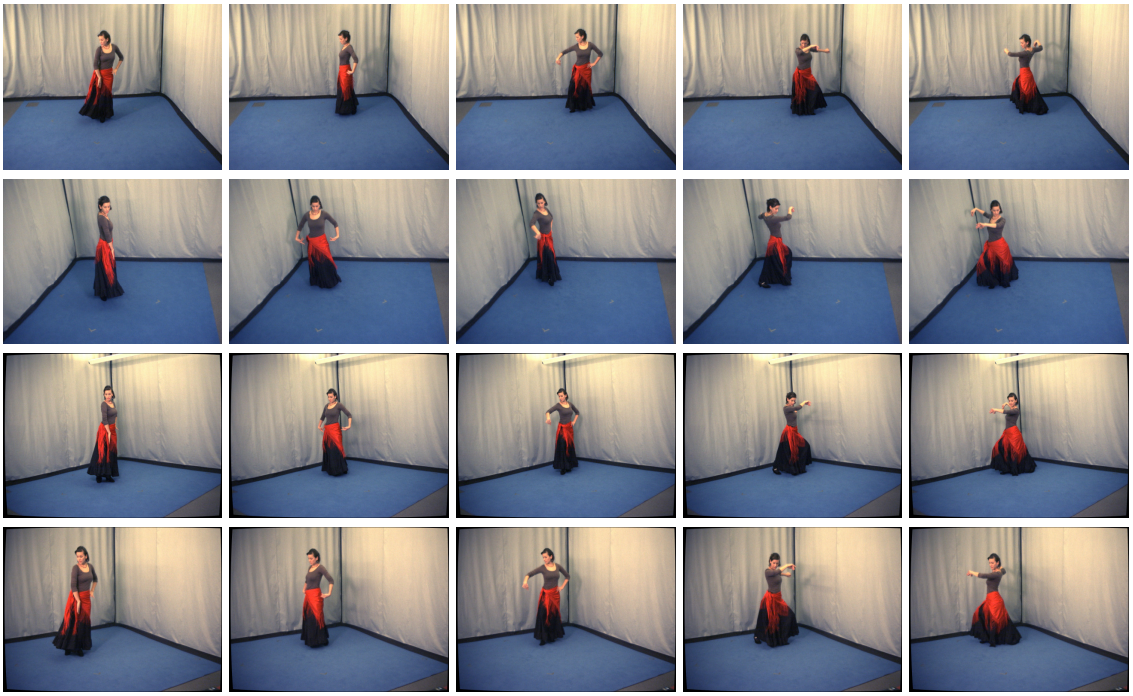


Figure A.4: Color camera images of the flamenco sequence, captured by four 3D video bricks.

Duration	25 s
Frames	300
Frame rate	12 fps
Camera resolution	1024×768 pixels
Input viewpoints	4
Horizontal viewing range	71°
Vertical viewing range	51°
Camera projector synchronization mode	black frame embedding
Points per 3D video frame (average)	1.47 million
Overall 3D data size (positions, colors, covariance matrices)	16.0 GBytes

Table A.2: Technical details of the flamenco sequence.

A.2.3 Juggle

Figure A.5 shows sample images of the juggle sequence. It has been captured using our four-brick setup described in section A.1.2. Technical details are given in table A.3.

Figure A.5: Color camera images of the juggle sequence, captured by four 3D video bricks.

Duration	20 s
Frames	240
Frame rate	12 fps
Camera resolution	1024×768 pixels
Input viewpoints	4
Horizontal viewing range	71°
Vertical viewing range	51°
Camera projector synchronization mode	black frame embedding
Points per 3D video frame (average)	1.43 million
Overall 3D data size (positions, colors, covariance matrices)	12.5 GBytes

Table A.3: Technical details of the juggle sequence.

A.2.4 Sofa

Figure A.6 shows sample images of the sofa sequence. It has been captured using our four-brick setup described in section A.1.2. Technical details are given in table A.4.

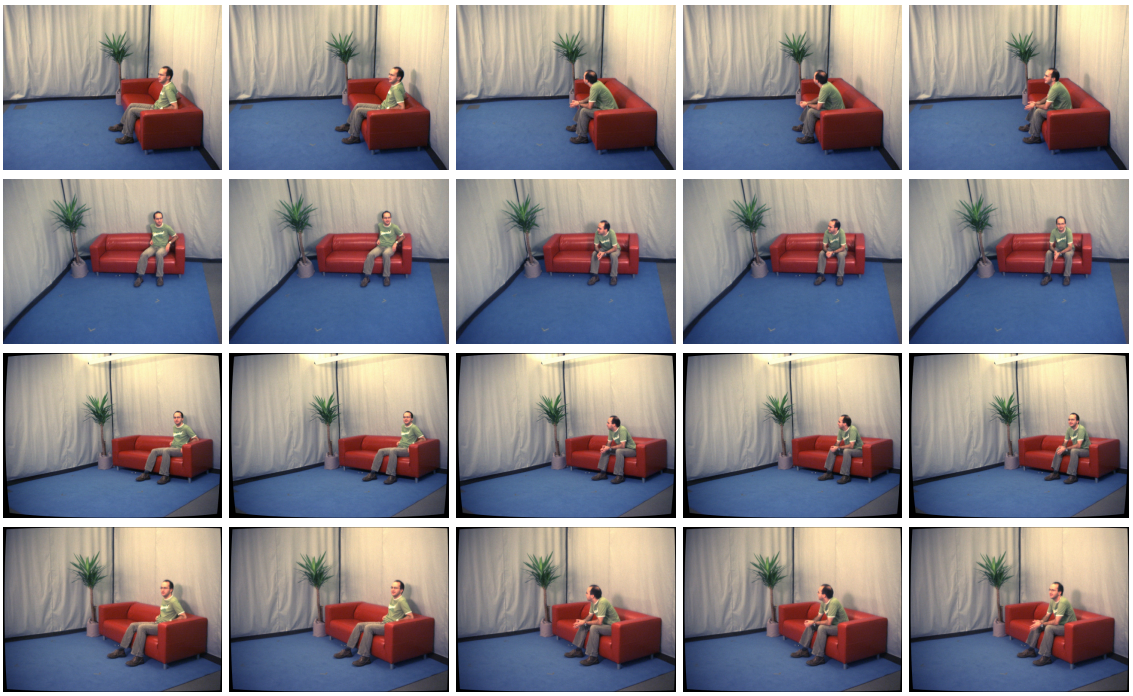


Figure A.6: Color camera images of the sofa sequence, captured by four 3D video bricks.

Duration	20 s
Frames	120
Frame rate	12 fps
Camera resolution	1024×768 pixels
Input viewpoints	4
Horizontal viewing range	71°
Vertical viewing range	51°
Camera projector synchronization mode	black frame embedding
Points per 3D video frame (average)	1.46 million
Overall 3D data size (positions, colors, covariance matrices)	6.4 GBytes

Table A.4: Technical details of the sofa sequence.

Appendix B

Glossary of Notations

A

A_S	screen space splatting kernel
$\mathcal{A}$	input alpha map
$\hat{\mathcal{A}}$	billboard alpha map
$\mathcal{A}_S$	alpha framebuffer

B

b	point capacity of a hard disk block
b_H	hyperslice distance from origin
Δb_H	hyperslice thickness
b_I	inner node capacity of a hard disk block
B	b-spline filter kernel
B	prediction operator of lifting scheme

C

c	point or pixel color
Δc	color deviation for photo consistency check
c_B	background color
c_F	foreground color
Δc_P	color deviation threshold for photo consistency check
c_R	ray color
c_S	framebuffer pixel color
C	coordinate transform of lifting scheme
$\mathcal{C}$	input color map

$\hat{\mathcal{C}}$	billboard color map
$\mathcal{C}_L$	input color map of left camera of a rectified stereo pair
$\mathcal{C}_R$	input color map of right camera of a rectified stereo pair
$\mathcal{C}_S$	color framebuffer

D

d	pixel disparity
Δd	disparity discontinuity threshold
$\hat{d}$	billboard texel displacement
$\hat{d}'$	filtered billboard texel displacement
d_t	derivative of pixel disparity in time
d_x	derivative of pixel disparity in x-direction
d_y	derivative of pixel disparity in y-direction
D_i	graph cut segmentation data term
$\mathcal{D}$	input disparity map
$\hat{\mathcal{D}}$	billboard displacement map
$\hat{\mathcal{D}}'$	filtered billboard displacement map

E

e	graph edge
e_M	stereo matching distance metric
e_N	distance of point normals used in lifting scheme
e_X	distance of point positions used in lifting scheme
$\mathbf{e}_L$	epipole of left camera of a stereo pair
$\mathbf{e}_R$	epipole of right camera of a stereo pair
E_A	graph cut segmentation energy function
E_D	graph cut segmentation data energy function
E_I	graph cut segmentation spatial smoothness energy function
E_M	stereo matching energy function
E_T	graph cut segmentation temporal smoothness energy function
$\mathcal{E}$	graph edge set
$\mathcal{E}_I$	intra-frame graph edge set
$\mathcal{E}_C$	graph cut edge set
$\mathcal{E}_M$	perfect matching edge set
$\mathcal{E}_T$	inter-frame graph edge set

F

f	focal length of camera
f_x	focal length of camera with scaling factor for image space x-axis
f_y	focal length of camera with scaling factor for image space y-axis
F	fundamental matrix

G

$\mathcal{G}$	graph
$\mathcal{G}_C$	cut graph

H

h	image height

I

i	index used for counting elements

J

j	index used for counting elements

K

k	natural number
k_D	domain filter kernel
k_R	range filter kernel

L

l	height of multiresolution point hierarchy

M

m	number of input views

N

n	number of points or pixels
N_k	k-dimensional Gaussian normal distribution
$\mathbf{N}$	3D point normal in world space
$\dot{\mathbf{N}}_H$	4D hyperplane normal in world space and time
$\mathcal{N}_I$	pixel neighborhoods in image space
$\mathcal{N}_T$	pixel neighborhoods in time

O

O	time complexity
$\mathbf{O}$	center of projection of camera
$\mathbf{O}_L$	center of projection left camera of a stereo pair
$\bar{\mathbf{O}}_L$	center of projection left camera of a rectified stereo pair
$\mathbf{O}_R$	center of projection right camera of a stereo pair
$\bar{\mathbf{O}}_R$	center of projection right camera of a rectified stereo pair

P

p	point capacity of main memory buffer
p_i	numerator of bilateral filter term
p_x	x-coordinate of principal point
$p_{x,L}$	x-coordinate of principal point of left camera of a rectified stereo pair
$p_{x,R}$	x-coordinate of principal point of right camera of a rectified stereo pair
p_y	y-coordinate of principal point
P	intrinsic matrix of camera
$\dot{\mathrm{P}}_H$	4D hyperplane projection matrix in world space and time
P_L	intrinsic matrix of left camera of a stereo pair
$\bar{\mathrm{P}}_L$	intrinsic matrix of left camera of a rectified stereo pair
P_R	intrinsic matrix of right camera of a stereo pair
$\bar{\mathrm{P}}_R$	intrinsic matrix of right camera of a rectified stereo pair

Q

q	natural number
p_i	denominator of bilateral filter term

R

r_1, r_2	radial lens distortion coefficients
r_C	splat cutoff radius
R	cylindrical world space coordinate
R	extrinsic matrix of camera
$\dot{\mathrm{R}}_H$	4D hyperplane rotation matrix in world space and time
R_L	extrinsic matrix of left camera of a stereo pair
$\bar{\mathrm{R}}_L$	extrinsic matrix of left camera of a rectified stereo pair
R_R	extrinsic matrix of right camera of a stereo pair
$\mathbf{R}$	normalized 3D viewing ray direction in world space
$\mathcal{R}$	viewing ray
$\mathcal{R}_B$	input background region for segmentation & matting
$\mathcal{R}_F$	input foreground region for segmentation & matting
$\mathcal{R}_U$	input unknown region for segmentation & matting

S

$S_{i,j}$	graph cut segmentation smoothness term

T

t	time
Δt	duration of a video frame
t_1, t_2	tangential lens distortion coefficients
T_B	background terminal node for graph cut segmentation
T_F	foreground terminal node for graph cut segmentation
$\mathbf{T}_i$	vectors spanning Gaussian point samples in world space
$\dot{\mathbf{T}}_i$	vectors spanning Gaussian point samples in world space and time
T_L	rectification matrix for left camera of stereo pair
T_R	rectification matrix for right camera of stereo pair
$\mathcal{T}_i$	kd-tree of logarithmic method
$\mathcal{T}_0^M$	main memory buffer of logarithmic method

U

u	natural number for counting input views

V

v	natural number for counting input views
V	covariance matrix
$\dot{\mathrm{V}}$	covariance matrix in world space and time
V_F	covariance matrix of foreground Gaussian mixture model
$\mathrm{V}_{F,C}$	covariance matrix of foreground color Gaussian mixture model
V_R	covariance matrix in ray space
$\mathbf{V}$	3D vector in world space
$\dot{\mathbf{V}}$	4D vector in world space and time
$\mathcal{V}$	graph vertex set

W

w	image width
$w_{i,j}$	graph edge weight
$\mathbf{W}$	3D vector in world space
$\mathcal{W}$	pixel window for stereo matching

X

x	Cartesian image space coordinate
x_D	x-coordinate of distorted pixel
x_L	x-coordinate of pixel in left image of a rectified stereo pair
x_R	x-coordinate of pixel in right image of a rectified stereo pair
$\mathbf{x}$	2D pixel position in image space
$\dot{\mathbf{x}}$	3D pixel position in image space and time
$\tilde{\mathbf{x}}$	homogeneous pixel position in image space
$\mathbf{x}_L$	2D pixel position in left image of a stereo pair
$\bar{\mathbf{x}}_L$	2D pixel position in left image of a rectified stereo pair
$\tilde{\mathbf{x}}_L$	homogeneous position of pixel in left image of a stereo pair
$\mathbf{x}_R$	2D pixel position in right image of a stereo pair
$\bar{\mathbf{x}}_R$	2D pixel position in right image of a rectified stereo pair
$\tilde{\mathbf{x}}_R$	homogeneous position of pixel in right image of a stereo pair
X	Cartesian world space coordinate
X_C	Cartesian camera space coordinate

$\mathbf{X}$	3D point position in world space
$\dot{\mathbf{X}}$	4D point position in world space and time
$\tilde{\mathbf{X}}$	homogeneous point position in world space
$\mathbf{X}_C$	3D point position in camera space
$\tilde{\mathbf{X}}_C$	homogeneous point position in camera space

Y

y	Cartesian image space coordinate
y_D	y-coordinate of distorted pixel
Y	Cartesian world space coordinate
Y_C	Cartesian camera space coordinate

Z

z	pixel depth
z_S	framebuffer pixel depth
Δz_S	threshold for fuzzy depth test
z_x	derivative of pixel depth in x-direction
z_y	derivative of pixel depth in x-direction
Z	Cartesian or cylindrical world space coordinate
Z_C	Cartesian camera space coordinate
$\mathcal{Z}_S$	depth framebuffer

A

α	pixel alpha
α_S	framebuffer pixel alpha
$\hat{\alpha}$	billboard texel alpha

B

β	weight in graph cut segmentation smoothness term
β_C	color variance in graph cut segmentation smoothness term
β_N	normal variance in graph cut segmentation smoothness term

Γ

γ_i^{-k}	individual detail coefficient in multiresolution hierarchy at level k
Γ^{-k}	set of detail coefficients in multiresolution hierarchy at level k

Δ

Δ_C	L2 color distance
Δ_N	L2 normal distance
$\delta_{a,b}$	Kronecker delta

Θ

Θ	cylindrical or spherical world space coordinate
Θ_B	Gaussian mixture model of background
$\Theta_{B,C}$	Gaussian mixture model of background color
$\Theta_{B,Z}$	Gaussian mixture model of background depth
Θ_F	Gaussian mixture model of foreground
$\Theta_{F,C}$	Gaussian mixture model of foreground color
$\Theta_{F,Z}$	Gaussian mixture model of foreground depth

Λ

λ_D	weight for data energy in graph cut segmentation
λ_I	weight for spatial smoothness energy in graph cut segmentation
λ_T	weight for temporal smoothness energy in graph cut segmentation
λ_Z	weight for depth data energy in graph cut segmentation
λ_i^{-k}	individual point in multiresolution hierarchy at level k
Λ_t	point cloud at time instant t
Λ^{-k}	point cloud in multiresolution hierarchy at level k

M

μ_F	mean value of foreground Gaussian mixture model
$\mu_{F,C}$	mean value of foreground color Gaussian mixture model

N

ν	Gaussian mixture model component weights

Ξ

ξ	fragment visibility

Π

$\hat{\pi}_0$	billboard plane offset
$\hat{\pi}_x$	gradient of billboard plane in x-direction
$\hat{\pi}_y$	gradient of billboard plane in y-direction
Π	image plane of camera
$\hat{\Pi}$	billboard plane
Π_L	image plane of left camera of a stereo pair
$\bar{\Pi}_L$	image plane of left camera of a rectified stereo pair
Π_R	image plane of right camera of a stereo pair
$\bar{\Pi}_R$	image plane of right camera of a rectified stereo pair

P

ρ_B	input background labels for graph cut segmentation
ρ_F	input foreground labels for graph cut segmentation
ρ_U	input unknown labels for graph cut segmentation

Σ

σ_x	uncertainty of pixel position in x-direction
σ_y	uncertainty of pixel position in y-direction
σ_z	uncertainty of pixel depth
σ_C	uncertainty of pixel position due to camera calibration error

Φ

Φ	spherical world space coordinate

Ω

ω	unstructured lumigraph weight

Copyrights

The following list specifies the sources of origin of all figures and models used in this thesis that have not been created by the author.

- Figure 4.6 is courtesy of Texas Instruments Incorporated.
- The chameleon model used in figures 7.3 and 7.14 is courtesy of Digimation Incorporated.
- The Igea model used in figure 7.13 is courtesy of Cyberware Incorporated.
- The dragon model used in figure 7.15 is courtesy of the Stanford Computer Graphics Group.
- The octopus model used in figure 7.16 is courtesy of the Computer Graphics Laboratory of ETH Zurich.

Bibliography

ADAMS, B. and DUTRÉ, P., 2003. Interactive boolean operations on surfel-bounded solids. *Proc. of SIGGRAPH '03*, pp. 651–656.

ADAMS, B. and DUTRÉ, P., 2004. Boolean operations on surfel-bounded solids using programmable graphics hardware. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 19–24.

ADAMSON, A. and ALEXA, M., 2003. Ray tracing point set surfaces. *Proc. of Shape Modeling International '03*, p. 272.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., and SILVA, C. T., 2001. Point set surfaces. *Proc. of Visualization '01*, pp. 21–28.

ALLARD, J., GOURANTON, V., LAMARQUE, G., MELIN, E., and RAFFIN, B., 2003. SoftGenLock: active stereo and genlock for PC cluster. *Proc. of Workshop on Virtual Environments '03*, pp. 255–260.

BAJAJ, C. L., PASCUCCI, V., RABBIOLO, G., and SCHIKORC, D., 1998. Hypervolume visualization: a challenge in simplicity. *Proc. of IEEE Symposium on Volume Visualization '98*, pp. 95–102.

BARRON, J., FLEET, D. J., and BEAUCHEMIN, S. S., 1994. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1), pp. 43–77.

BAYAKOVSKI, Y., LEVKOVICH-MASLYUK, L., IGNATENKO, A., KONUSHIN, A., TIMASOV, D., ZHIRKOV, A., HAN, M., and PARK, I. K., 2002. Depth image-based representations for static and animated 3D objects. *Proc. of International Conference on Image Processing '02*, vol. 3, pp. 25–28.

BECKMANN, N., KRIEGEL, H.-P., SCHNEIDER, R., and SEEGER, B., 1990. The R*-tree: An efficient and robust access method for points and rectangles. *Proc. of SIGMOD '90*, pp. 322–331.

BENNETT, E. P. and MCMILLAN, L., 2003. Proscenium: a framework for spatio-temporal video editing. *Proc. of ACM Multimedia '03*, pp. 177–184.

BENTLEY, J. L., 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), pp. 509–517.

BENTLEY, J. L., 1978. Decomposable searching problems. *Information Processing Letters*, 8(5), pp. 244–251.

BIRCHFIELD, S. and TOMASI, C., 1998. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4), pp. 401–406.

BORDIGNON, A., LEWINER, T., LOPES, H., TAVARES, G., and CASTRO, R., 2006. Point set compression through BSP quantization. *Proc. of Brazilian Symposium on Computer Graphics and Image Processing '06*, pp. 229–238.

BOTSCH, M., HORNUNG, A., ZWICKER, M., and KOBBELT, L., 2005. High-quality surface splatting on today's GPUs. *Proc. of Eurographics Symposium on Point-Based Graphics '05*, pp. 17–24.

BOTSCH, M., SPERNAT, M., and KOBBELT, L., 2004. Phong splatting. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 25–32.

BOTSCH, M., WIRATANAYA, A., and KOBBELT, L., 2002. Efficient high quality rendering of point sampled geometry. *Proc. of Eurographics Workshop on Rendering '02*, pp. 53–64.

BOYKOV, Y. and JOLLY, M.-P., 2001. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. *Proc. of International Conference on Computer Vision '01*, vol. 1, pp. 105–112.

BOYKOV, Y. and KOLMOGOROV, V., 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9), pp. 1124–1137.

BOYKOV, Y., VEKSLER, O., and ZABIH, R., 1999. Fast approximate energy minimization via graph cuts. *Proc. of International Conference on Computer Vision '99*, vol. 1, pp. 377–384.

BROADHURST, A., DRUMMOND, T., and CIPOLLA, R., 2001. A probabilistic framework for the space carving algorithm. *Proc. of International Conference on Computer Vision '01*, vol. 1, pp. 388–393.

BROWN, D. C., 1966. Decentering distortion of lenses. *Photogrammetric Engineering*, 32(3), pp. 444–462.

BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S., and COHEN, M., 2001. Unstructured lumigraph rendering. *Proc. of SIGGRAPH '01*, pp. 425–432.

CARRANZA, J., THEOBALT, C., MAGNOR, M., and SEIDEL, H.-P., 2003. Free-viewpoint video of human actors. *Proc. of SIGGRAPH '03*, pp. 569–577.

CHAI, J.-X., CHAN, S.-C., SHUM, H.-Y., and TONG, X., 2000. Plenoptic sampling. *Proc. of SIGGRAPH '00*, pp. 307–318.

CHEUNG, G., BAKER, S., HODGINS, J., and KANADE, T., 2004. Markerless human motion transfer. *Proc. of International Symposium on 3D Data Processing, Visualization and Transmission '04*, pp. 373–378.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., and SZELISKI, R., 2004. Video matting of complex scenes. *Proc. of SIGGRAPH '02*, pp. 243–248.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D. H., and SZELISKI, R., 2001. A Bayesian approach to digital matting. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 01*, vol. 2, pp. 264–271.

COCKSHOTT, W. P., HOFF, S., and NEBEL, J.-C., 2003. An experimental 3D digital TV studio. *IEE Proceedings Vision, Image and Signal Processing*, 150(1), pp. 28–33.

COLOMBARI, A., FUSIELLO, A., and MURINO, V., 2006. Background initialization in cluttered sequences. *Proc. of Conference on Computer Vision and Pattern Recognition Workshop '06*, pp. 197–202.

COOK, W. and ROHE, A., 1999. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2), pp. 138–148.

COTTING, D., 2007. Smart displays in interactive visual workspaces. Ph.D. thesis, ETH Zurich, Department of Computer Science.

COTTING, D., NAEF, M., GROSS, M., and FUCHS, H., 2004. Embedding imperceptible patterns into projected images for simultaneous acquisition and display. *Proc. of IEEE/ACM International Symposium on Mixed and Augmented Reality '04*, pp. 100–109.

CULBERTSON, W. B., MALZBENDER, T., and SLABAUGH, G. G., 1999. Generalized voxel coloring. *Proc. of Workshop on Vision Algorithms '99*, pp. 100–115.

DAVIS, J., RAMAMOORTHI, R., and RUSINKIEWICZ, S., 2003. Spacetime stereo: A unifying framework for depth from triangulation. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 03*, pp. 359–366.

DÉCORET, X., DURAND, F., SILLION, F., and DORSEY, J., 2003. Billboard clouds for extreme model simplification. *Proc. of SIGGRAPH '03*, pp. 689–696.

DONNELLY, W., 2005. Per-pixel displacement mapping with distance functions. PHARR, M., ed., *GPU Gems 2*, chap. 8, pp. 123–136. Addison Wesley.

EDMONDS, J., 1965. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17(3), pp. 449–467.

ELAD, M., 2002. In the origin of the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing*, 10(10), pp. 1141–1151.

FAUGERAS, O., 1993. *Three-Dimensional Computer Vision*. MIT Press.

FELS, S. and MASE, K., 1999. Interactive video cubism. *Proc. of Workshop on New Paradigms in Information Visualization aNd Manipulation '99*, pp. 78–82.

FLEISHMAN, S., COHEN-OR, D., ALEXA, M., and SILVA, C. T., 2003. Progressive point set surfaces. *ACM Transactions on Graphics*, 22(4), pp. 997–1011.

FUSIELLO, A., ROBERTO, V., and TRUCCO, E., 1997. Efficient stereo with multiple windowing. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 97*, pp. 858–863.

FUSIELLO, A., TRUCCO, E., and VERRI, A., 2000. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1), pp. 16–22.

GOLDLÜCKE, B., IHRKE, I., LINZ, C., and MAGNOR, M., 2007. Weighted minimal hypersurface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7), pp. 1194–1208.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., and COHEN, M. F., 1996. The lumigraph. *Proc. of SIGGRAPH '96*, pp. 43–54.

GROSS, M., 1994. *Visual Computing: Integration of Computer Graphics, Visual Perception and Imaging*. Springer-Verlag.

GROSS, M., 2006. Getting to the point. *IEEE Computer Graphics & Applications*, 26(5), pp. 96–99.

GROSS, M. and PFISTER, H., eds., 2007. *Point-based graphics*. Morgan Kaufmann.

GROSS, M., WÜRMLIN, S., NÄF, M., LAMBORAY, E., SPAGNO, C., KUNZ, A., MOERE, A. V., STREHLKE, K., LANG, S., SVOBODA, T., KOLLER-MEIER, E., GOOL, L. V., and STAADT, O., 2003. blue-c: A spatially immersive display and 3D video portal for telepresence. *Proc. of SIGGRAPH '03*, pp. 819–827.

GROSSMAN, J. P. and DALLY, W., 1998. Point-sample rendering. *Proc. of Eurographics Workshop on Rendering '98*, pp. 181–192.

GVILI, R., KAPLAN, A., OFEK, E., , and YAHAV, G., 2003. Depth keying. *Stereoscopic Displays and Virtual Reality Systems X (Proceedings of SPIE Volume 5006)*, pp. 564–574.

HECKBERT, P. S., 1989. Fundamentals of texture mapping and image warping. Master's thesis, CS Division, U.C. Berkeley.

HOFSETZ, C., NG, K., MAX, N., CHEN, G., LIU, Y., and MCGUINNESS, P., 2005. Image-based rendering of range data with estimated depth uncertainty. *IEEE Computer Graphics & Applications*, 24(4), pp. 34–42.

HONG, L. and CHEN, G., 2004. Segment-based stereo matching using graph cuts. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 04*.

HUANG, Y., PENG, J., KUO, C.-C. J., and GOPI, M., 2006. Octree-based progressive geometry coding of point clouds. *Proc. of Eurographics Symposium on Point-Based Graphics '06*, pp. 103–110.

IDDAN, G. J. and YAHAV, G., 2001. 3D imaging in the studio (and elsewhere...). *Three Dimensional Image Capture and Applications IV (Proceedings of SPIE Volume 4298)*, pp. 48–55.

INOKUCHI, S., SATO, K., and MATSUDA, F., 1984. Range imaging system for 3-d object recognition. *Proc. of International Conference on Pattern Recognition '84*, pp. 806–808.

JAIN, A. K. and DUBES, R. C., 1988. *Algorithms for clustering data*. Prentice-Hall Inc.

KANADE, T., RANDER, P., and NARAYANAN, P. J., 1997. Virtualized reality: Construction of virtual worlds from real scenes. *IEEE Multimedia*, 4(1), pp. 34–47.

KANG, S., WEBB, J., ZITNICK, L., and KANADE, T., 1995. A multi-baseline stereo system with active illumination and real-time image acquisition. *Proc. of International Conference on Computer Vision '95*, pp. 88–93.

KEINOSUKE, F., 1990. *Introduction to Statistical Pattern Recognition*. Elsevier Science Ltd, 2nd edn.

KLEIN, A. W., SLOAN, P.-P. J., FINKELSTEIN, A., and COHEN, M. F., 2002. Stylized video cubes. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation '02*.

KOLMOGOROV and ZABIH, 2002. What energy functions can be minimized via graph cuts? *Proc. of European Conference on Computer Vision '02*, vol. 3, pp. 65–81.

KUTULAKOS, K. N. and SEITZ, S. M., 1999. A theory of shape by space carving. *Proc. of International Conference on Computer Vision '99*, pp. 307–314.

LAMBORAY, E., WÜRMLIN, S., and GROSS, M., 2004a. Real-time streaming of point-based 3D video. *Proc. of IEEE Virtual Reality '04*, pp. 91–98.

LAMBORAY, E., WÜRMLIN, S., WASCHBÜSCH, M., GROSS, M., and PFISTER, H., 2004b. Unconstrained free-viewpoint video coding. *Proc. of International Conference on Image Processing '04*, vol. 5, pp. 3261–3246.

LEE, D. T. and WONG, C. K., 1977. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9, pp. 23–29.

LEVIN, D., 2003. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pp. 32–49. Springer-Verlag.

LEVOY, M. and HANRAHAN, P., 1996. Light field rendering. *Proc. of SIGGRAPH '96*, pp. 31–42.

LEVOY, M. and WHITTED, T., 1985. The use of points as display primitives. Tech. Rep. 85–022, The University of North Carolina at Chapel Hill, Department of Computer Science.

LI, Y., SUN, J., and SHUM, H.-Y., 2005. Video object cut and paste. *Proc. of SIGGRAPH '05*, pp. 595–600.

LI, Y., SUN, J., TANG, C.-K., and SHUM, H.-Y., 2004. Lazy snapping. *Proc. of SIGGRAPH '04*, pp. 303–308.

LONG, W. and YANG, Y.-H., 1990. Stationary background generation: an alternative to the difference of two images. *Pattern Recognition*, 23(12), pp. 1351–1359.

LOVASZ, L. and PLUMMER, M. D., 1986. *Matching Theory*. Elsevier Science Ltd.

MALVAR, H. S., HE, L.-W., and CUTLER, R., 2004. High-quality linear interpolation for demosaicing of Bayer-patterned color images. *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing '04*, vol. 3, pp. 485–488.

MANTLER, S., JESCHKE, S., and WIMMER, M., 2007. Displacement mapped billboard clouds. Tech. Rep. TR-186-2-07-01, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

MATUSIK, W., BUEHLER, C., and MCMILLAN, L., 2001. Polyhedral visual hulls for real-time rendering. *Proc. of Eurographics Workshop on Rendering '01*, pp. 115–125.

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., and MCMILLAN, L., 2000. Image-based visual hulls. *Proc. of SIGGRAPH '00*, pp. 369–374.

MATUSIK, W. and PFISTER, H., 2004. 3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *Proc. of SIGGRAPH '04*, pp. 814–824.

MCGUIRE, M., MATUSIK, W., PFISTER, H., HUGHES, J. F., and DURAND, F., 2005. Defocus video matting. *Proc. of SIGGRAPH '05*, pp. 567–576.

MCLACHLAN, G. J. and BASFORD, K. E., 1988. *Mixture models. Inference and applications to clustering*. New York: Dekker.

MOFFAT, A., NEAL, R. M., and WITTEN, I. H., 1998. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3), pp. 202–211.

MULLIGAN, J. and DANIILIDIS, K., 2000. View-independent scene acquisition for tele-presence. *Proc. of International Symposium on Augmented Reality '00*, pp. 105–110.

NG, R., LEVOY, M., BRÉDIF, M., DUVAL, G., HOROWITZ, M., and HANRAHAN, P., 2005. Light field photography with a hand-held plenoptic camera. Tech. Rep. CSTR 2005-02, Stanford University Computer Science.

OpenCV. Open source computer vision library. Available at http://www.intel.com/technology/computing/opencv/index.htm.

OVERMARS, M. H., 1987. *Design of Dynamic Data Structures*. Springer-Verlag New York, Inc.

PASKO, A., ADZHIEV, V., SCHMITT, B., and SCHLICK, C., 2002. Constructive hypervolume modeling. *Graphical Models*, 64(2).

PAULY, M. and GROSS, M., 2001. Spectral processing of point sampled geometry. *Proc. of SIGGRAPH '01*, pp. 379–386.

PAULY, M., GROSS, M., and KOBBELT, L., 2002. Efficient simplification of point-sampled geometry. *Proc. of Visualization '02*, pp. 163–170.

PAULY, M., KEISER, R., KOBBELT, L., and GROSS, M., 2003. Shape modeling with point-sampled geometry. *Proc. of SIGGRAPH '03*, pp. 641–650.

PFISTER, H., ZWICKER, M., VAN BAAR, J., and GROSS, M., 2000. Surfels: Surface elements as rendering primitives. *Proc. of SIGGRAPH '00*, pp. 335–342.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., and FLANNERY, B. P., 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edn.

PRITCHETT, P. and ZISSERMAN, A., 1998. Wide baseline stereo matching. *Proc. of International Conference on Computer Vision '98*, pp. 754–760.

PROCOPIUC, O., AGARWAL, P., ARGE, L., and VITTER, J., 2003. Bkd-tree: A dynamic scalable kd-tree. *Proc. of International Symposium on Spatial and Temporal Databases '03*, pp. 46–65.

RASKAR, R. and TUMBLIN, J., 2007. *Computational Photography: Mastering New Techniques for Lenses, Lighting, and Sensors*. A K Peters.

REN, L., PFISTER, H., and ZWICKER, M., 2002. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum*, 21(3), pp. 461–470.

RHEE, S.-M., ZIEGLER, R., PARK, J., NAEF, M.-M., GROSS, M.-M., and KIM, M.-H., 2007. Low-cost telepresence for collaborative virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 13(1), pp. 156–166.

ROTHER, C., KOLMOGOROV, V., and BLAKE, A., 2004. "GrabCut"—interactive foreground extraction using iterated graph cuts. *Proc. of SIGGRAPH '04*, pp. 309–314.

RUSINKIEWICZ, S. and LEVOY, M., 2000. QSplat: a multiresolution point rendering system for large meshes. *Proc. of SIGGRAPH '00*, pp. 343–352.

RUSINKIEWICZ, S. and LEVOY, M., 2001. Streaming QSplat: a viewer for networked visualization of large, dense models. *Proc. of SIGGRAPH '01*, pp. 63–68.

SADLO, F., WEYRICH, T., PEIKERT, R., and GROSS, M., 2005. A practical structured light acquisition system for point-based geometry and texture. *Proc. of Eurographics Symposium on Point-Based Graphics '05*, pp. 89–98.

SALVI, J., PAGES, J., and BATLLE, J., 2004. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4), pp. 827–849.

SAMET, H., 1984. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2), pp. 187–260.

SAMET, H., 1990. *The design and analysis of spatial data structures*. Addison Wesley.

SCHARSTEIN, D. and SZELISKI, R., 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1–3), pp. 7–42.

SCHNABEL, R. and KLEIN, R., 2006. Octree-based point-cloud compression. *Proc. of Eurographics Symposium on Point-Based Graphics '06*, pp. 111–120.

SEITZ, S. and DYER, C., 1997. Photorealistic scene reconstruction by voxel coloring. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 97*, pp. 1067–1073.

SEN, P., CHEN, B., GARG, G., MARSCHNER, S. R., HOROWITZ, M., LEVOY, M., and LENSCH, H. P. A., 2005. Dual photography. *Proc. of SIGGRAPH '05*, pp. 745–755.

SHADE, J., GORTLER, S., HE, L.-W., and SZELISKI, R., 1998. Layered depth images. *Proc. of SIGGRAPH '98*, pp. 231–242.

SHAPIRO, J. M., 1993. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 31(12), pp. 3445–3462.

SHUM, H.-Y., SUN, J., YAMAZAKI, S., LI, Y., and TANG, C.-K., 2004. Pop-up light field: An interactive image-based modeling and rendering system. *ACM Transactions on Graphics*, 23(2), pp. 143–162.

SUN, J., JIA, J., TANG, C.-K., and SHUM, H.-Y., 2004. Poisson matting. *Proc. of SIGGRAPH '04*, pp. 315–321.

SUN, J., LI, Y., KANG, S.-B., and SHUM, H.-Y., 2005. Symmetric stereo matching for occlusion handling. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 05*, pp. 399–406.

SUN, J., ZHENG, N.-N., and SHUM, H.-Y., 2003. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7), pp. 787–800.

SVOBODA, T., MARTINEC, D., and PAJDLA, T., 2005. A convenient multicamera self-calibration for virtual environments. *Presence: Teleoperators Virtual Environmants*, 14(4), pp. 407–422.

SWELDENS, W., 1995. The lifting scheme: A new philosophy in biorthogonal wavelet constructions. *Wavelet Applications in Signal and Image Processing III (Proceedings of SPIE Volume 2569)*, pp. 68–79.

THEOBALT, C., AHMED, N., LENSCH, H., MAGNOR, M., and SEIDEL, H.-P., 2007. Seeing people in different light—joint shape, motion, and reflectance capture. *IEEE Transactions on Visualization and Computer Graphics*, 13(4), pp. 663–674.

TOMASI, C. and MANDUCHI, R., 1998. Bilateral filtering for gray and color images. *Proc. of International Conference on Computer Vision '98*, pp. 839–846.

UCHIDA, H., ISAKA, H., YOSHIDA, T., and SAFAR, J., 1996. DVCPRO: A comprehensive format overview. *SMPTE Technical Conference and World Media Expo No137*, pp. 406–418.

VEDULA, S., BAKER, S., and KANADE, T., 2002. Spatio-temporal view interpolation. *Proc. of Eurographics Workshop on Rendering '02*, pp. 65–76.

VEDULA, S., BAKER, S., RANDER, P., COLLINS, R., and KANADE, T., 1999. Three-dimensional scene flow. *Proc. of International Conference on Computer Vision '99*, pp. 722–729.

VENGROFF, D. E., 1994. A transparent parallel I/O environment. *Proc. of DAGS Symposium on Parallel Computation*, pp. 117–134. Source code available at http://www.cs.duke.edu/TPIE.

VUYLSTEKE, P. and OOSTERLINCK, A., 1990. Range image acquisition with a single binary-encoded light pattern. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2), pp. 148–164.

WANG, J., BHAT, P., COLBURN, A., AGRAWALA, M., , and COHEN, M., 2005. Interactive video cutout. *Proc. of SIGGRAPH '05*, pp. 585–594.

WASCHBÜSCH, M., COTTING, D., DULLER, M., and GROSS, M., 2006. WinSGL: Software genlocking for cost-effective display synchronization under Microsoft Windows. *Proc. of Eurographics Symposium on Parallel Graphics and Visualization '06*, pp. 111–118.

WEICKERT, J., 1998. *Anisotropic diffusion in image processing*. Teubner Verlag.

WEYRICH, T., PAULY, M., KEISER, R., HEINZLE, S., SCANDELLA, S., and GROSS, M., 2004. Post-processing of scanned 3D surface data. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 85–94.

WHEELER, P., 2001. *Digital Cinematography*. Focal Press.

WICKE, M., TESCHNER, M., and GROSS, M., 2004. CSG tree rendering of point-sampled objects. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 160–168.

WILBURN, B., JOSHI, N., VAISH, V., TALVALA, E.-V., ANTUNEZ, E., BARTH, A., ADAMS, A., HOROWITZ, M., and LEVOY, M., 2005. High performance imaging using large camera arrays. *Proc. of SIGGRAPH '05*, pp. 765–776.

WILLIAMS, L., 1978. Casting curved shadows on curved surfaces. *Proc. of SIGGRAPH '78*, pp. 270–274.

WOODRING, J., WANG, C., and SHEN, H.-W., 2003. High dimensional direct rendering of time-varying volumetric data. *Proc. of Visualization '03*, pp. 417–424.

WÜRMLIN, S., 2007. Dynamic point samples as primitives for free-viewpoint video. Ph.D. thesis, ETH Zurich, Department of Computer Science.

WÜRMLIN, S., LAMBORAY, E., and GROSS, M., 2004. 3D video fragments: Dynamic point samples for real-time free-viewpoint video. *IEEE Computer Graphics & Applications*, 28(1), pp. 3–14.

WÜRMLIN, S., LAMBORAY, E., STAADT, O., and GROSS, M., 2003. 3D video recorder: A system for recording and playing free-viewpoint video. *Computer Graphics Forum*, 22(2), pp. 181–193.

WÜRMLIN, S., LAMBORAY, E., STAADT, O. G., and GROSS, M. H., 2002. 3D video recorder. *Proc. of Pacific Graphics '02*, pp. 325–334.

WÜRMLIN, S., LAMBORAY, E., WASCHBÜSCH, M., KAUFMANN, P., SMOLIC, A., and GROSS, M., 2005. Image-space free-viewpoint video. *Proc. of Vision, Modeling, and Visualization '05*, pp. 453–460.

YANG, J. C., EVERETT, M., BUEHLER, C., and MCMILLAN, L., 2002. A real-time distributed light field camera. *Proc. of Eurographics Workshop on Rendering '02*, pp. 77–86.

YANG, Q., WANG, L., YANG, R., WANG, S., LIAO, M., and NISTÉR, D., 2006. Real-time global stereo matching using hierarchical belief propagation. *Proc. of British Machine Vision Converence '06*, pp. 989–998.

ZHANG, C. and CHEN, T., 2001. Generalized plenoptic sampling. Tech. Rep. AMP 01-06, Electrical and Computer Engineering, Carnegie Mellon University.

ZHANG, L., CURLESS, B., and SEITZ, S. M., 2003. Spacetime stereo: Shape recovery for dynamic scenes. *Proc. of IEEE Conference on Computer Vision and Pattern Recognition 03*, pp. 367–374.

ZITNICK, C. and KANADE, T., 1999. A cooperative algorithm for stereo matching and occlusion detection. Tech. Rep. CMU-RI-TR-99-35, Robotics Institute, Carnegie Mellon University.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., and SZELISKI, R., 2004. High-quality video view interpolation using a layered representation. *Proc. of SIGGRAPH '04*, pp. 600–608.

ZWICKER, M., PAULY, M., KNOLL, O., and GROSS, M., 2002a. Pointshop 3D: an interactive system for point-based surface editing. *Proc. of SIGGRAPH '02*, pp. 322–329.

ZWICKER, M., PFISTER, H., VAN BAAR, J., and GROSS, M., 2001a. Surface splatting. *Proc. of SIGGRAPH '01*, pp. 371–378.

ZWICKER, M., PFISTER, H., VAN BAAR, J., and GROSS, M., 2002b. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3), pp. 223–238.

ZWICKER, M., PFISTER, H., VANBAAR, J., and GROSS, M., 2001b. EWA volume splatting. *Proc. of Visualization '01*, pp. 29–36.

ZWICKER, M., RÄSÄNEN, J., BOTSCH, M., DACHSBACHER, C., and PAULY, M., 2004. Perspective accurate splatting. *Proc. of Graphics Interface '04*, pp. 247–254.

Curriculum Vitae

Dipl.-Inform. Michael Waschbüsch

Personal Data

Date of birth	January 15, 1976
Place of birth	Wadern, Germany
Nationality	German
Civil status	Single

Education

2003–2007	ETH Zurich, Switzerland, Dep. of Computer Science Doctoral candidate in the Computer Graphics Laboratory Dissertation: *3D Video Acquisition, Representation and Editing* Referee: Prof. Dr. Markus Gross Co-referees: Prof. Dr. Marcus Magnor, Dr. Stephan Würmlin
1996–2003	University of Kaiserslautern, Germany, Dep. of Computer Science Diploma degree in computer science, minor in electrical engineering Diploma Thesis: *Efficient Data Structures for Animated Virtual Humans* Semester Thesis: *Rendering von Tageslicht*
1986–1995	Hochwald-Gymnasium Wadern, Germany Abitur (university entrance exam)
1982–1986	Grundschule Nunkirchen, Germany

Professional Experience

2003–2007	ETH Zurich, Switzerland, Dep. of Computer Science Research and teaching assistant in the Computer Graphics Laboratory
2001–2002	University of Kaiserslautern, Germany, Dep. of Computer Science Student research assistant in the Numerical Algorithms Group
2000–2001	University of Kaiserslautern, Germany, Dep. of Computer Science Student research assistant in the Artificial Intelligence and Knowledge-based Systems Group

1998–2002	University of Kaiserslautern, Germany, Dep. of Computer Science Student teaching assistant for various lectures in theoretical and technical computer science
1995–1996	Civilian service at Arbeiterwohlfahrt des Saarlandes, Wadern-Nunkirchen, Germany

Scientific Publications

WASCHBÜSCH, M., WÜRMLIN, S., and GROSS, M., 2007. 3D Video Billboard Clouds. *Proc. of Eurographics '07*, pp. 561–569.

COTTING, D., WASCHBÜSCH, M., DULLER, M., and GROSS, M., 2007. WinSGL: Synchronizing displays in parallel graphics using cost-effective software genlocking. *Parallel Computing*, 33(6), pp. 420–437.

WASCHBÜSCH, M., WÜRMLIN, S., COTTING, D., and GROSS, M., 2007. Point-sampled 3D video of real-world scenes. *Image Communication*, 22(2), pp. 203–216.

WASCHBÜSCH, M., WÜRMLIN, S., and GROSS, M., 2006. Interactive 3D video editing. *Proc. of Pacific Graphics '06*, pp. 631–641.

WASCHBÜSCH, M., COTTING, D., DULLER, M., and GROSS, M., 2006. WinSGL: Software genlocking for cost-effective display synchronization under Microsoft Windows. *Proc. of Eurographics Symposium on Parallel Graphics and Visualization '06*, pp. 111–118.

WASCHBÜSCH, M., WÜRMLIN, S., COTTING, D., SADLO, F., and GROSS, M., 2005. Scalable 3D video of dynamic scenes. *Proc. of Pacific Graphics '05*, pp. 629–638.

WÜRMLIN, S., LAMBORAY, E., WASCHBÜSCH, M., KAUFMANN, P., SMOLIC, A., and GROSS, M., 2005. Image-space free-viewpoint video. *Proc. of Vision, Modeling, Visualization '05*, pp. 453–460.

WÜRMLIN, S., LAMBORAY, E., WASCHBÜSCH, M., and GROSS, M., 2004. Dynamic point samples for free-viewpoint video. *Proc. of Picture Coding Symposium '04*.

WASCHBÜSCH, M., GROSS, M., EBERHARD, F., LAMBORAY, E., and WÜRMLIN, S., 2004. Progressive compression of point-sampled models. *Proc. of Eurographics Symposium on Point-Based Graphics '04*, pp. 95–102.

LAMBORAY, E., WÜRMLIN, S., WASCHBÜSCH, M., GROSS, M., and PFISTER, H., 2004. Unconstrained free-viewpoint video coding. *Proc. of International Conference on Image Processing '04*, vol. 5, pp. 3261–3264.

LAMBORAY, E., WASCHBÜSCH, M., WÜRMLIN, S., PFISTER, H., and GROSS, M., 2003. Dynamic point cloud compression for free-viewpoint video. Tech. Rep. 430, Institute of Computational Science, ETH Zurich. Also available as Tech. Rep. TR2003-137, Mitsubishi Electric Research Laboratories.

Industry Standardization Contributions

WÜRMLIN, S., LEE, C., ZWICKER, M., WASCHBÜSCH, M., GROSS, M., and PFISTER, H., 2004. Results on reference software implementation on point based rendering for MPEG-4 AFX. MPEG2004/m11038, Redmond, USA.

WÜRMLIN, S., WASCHBÜSCH, M., LAMBORAY, E., KAUFMANN, P., SMOLIC, A., and GROSS, M., 2004. Image-space free-viewpoint video. MPEG2004/m10894, Redmond, USA.

WÜRMLIN, S., ZWICKER, M., WASCHBÜSCH, M., GROSS, M., and PFISTER, H., 2004. Results on core experiments on point based rendering for MPEG-4 AFX. MPEG2004/m10579, Munich, Germany.

WÜRMLIN, S., WASCHBÜSCH, M., and GROSS, M., 2004. ETH-REAL: A real-world test data set for 3DAV EE2. MPEG2004/m10580, Munich, Germany.

WASCHBÜSCH, M., WÜRMLIN, S., LAMBORAY, E., GROSS, M., and PFISTER, H., 2004. Average coding of free-viewpoint video in MPEG-4. MPEG2004/m10581, Munich, Germany.

LAMBORAY, E., WÜRMLIN, S., WASCHBÜSCH, M., GROSS, M., and PFISTER, H., 2003. A compression framework for free-viewpoint video based on 3D video fragments. MPEG2003/m10339, Waikoloa, Hawaii, USA.

Patents

LAMBORAY, E., WASCHBÜSCH, M., WÜRMLIN, S., and GROSS, M., 2003. Method for encoding and decoding free viewpoint videos. U.S. Patent No. 10/723,035.