


Graz University of Technology
Institute for Computer Graphics and Vision

Dissertation

Efficient Image-based Augmentations

Stefan Hauswiesner
Graz, Austria, January 2013

Advisor
Prof. Dr. Gerhard Reitmayr
Institute for Computer Graphics and Vision,
Graz University of Technology

Referee
Dr. Gabriel J. Brostow
Department of Computer Science, University College London

To Angelika and my parents

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, Austria	**January 4, 2013**	
Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, Österreich	**4. Jänner, 2013**	
Ort	Datum	Unterschrift

Abstract

Real-time 3D reconstruction and rendering of humans captured by a set of cameras is important in a number of applications, such as virtual try-on and interactive mixed reality systems. These applications create a virtual mirror by displaying a live representation of the user embedded in an artificial environment. Image-based methods are frequently used to perform reconstruction and rendering but suffer from poor performance at higher resolutions. However, the performance of these methods is particularly important because users perceive the latency between their own movements and the rendered images as disturbing. Previous implementations computed every output image from scratch and therefore did not reach their highest potential efficiency.

This thesis introduces novel methods to reuse information from previous images to improve the performance of image-based rendering. The image-based visual hull algorithm is a rendering method based on shape-from-silhouette reconstruction. This algorithm performs reconstruction and rendering simultaneously, which is more efficient than executing both steps separately. Combined with texturing, it generates novel viewpoints of the captured scene from arbitrary viewing angles. To improve the performance of the algorithm, we exploit our knowledge of the input data. When users move in a virtual mirror scenario, their motions are typically smooth and slow compared to the camera frame rate. Consecutive frames of such a video stream offer a certain amount of temporal coherence. To exploit the coherence during image-based rendering, we introduce methods to detect information in the video streams that remains valid over time. By reusing this information, the amount of required computations can be reduced considerably.

With the suggested efficient image-based rendering algorithms, various new applications become feasible, such as high fidelity telepresence systems or virtual mirrors. For advanced applications, like virtual try-on systems and mixed reality augmentations, the rendered image of a user must be augmented with virtual garments and accessories. This thesis introduces the *image-based augmentation* process to augment users with previously recorded image sequences. A major advantage of using recorded garment sequences over manually modeled meshes is the short content creation time, which is on the order of minutes. In combination with image-based rendering and image processing, we achieve realistic augmentations that blend seamlessly with images of the user.

Kurzfassung

Das Modell eines Benutzers in Echtzeit anhand von Kamerabildern zu rekonstruieren und gleichzeitig darzustellen ist essentiell für die Virtuelle Realität und zeigt ein breites Anwendungsspektrum. Virtuelle Umkleidekabinen und Mixed-Reality-Systeme basieren zum größten Teil auf diesen Techniken. Diese Anwendungen zeigen ein aktuelles Bild des Benutzers und erweitern es mit virtuellen, dreidimensionalen Inhalten. Bildbasierte Anzeigeverfahren kombinieren Rekonstruktion und Darstellung, sind aber bei höheren Auflösungen nicht schnell genug. Die Berechnungs- und Darstellungsgeschwindigkeit ist dabei ein aktives Forschungsgebiet, da Benutzer interaktiver Systeme keine Latenz, z.B. zur eigenen Körperbewegung, akzeptieren. Bisherige Implementierungen berechnen jedes Bild von Grund auf neu und schöpfen daher nicht alle Möglichkeiten zur Beschleunigung aus.

In dieser Dissertation beschreiben wir neuartige Verfahren um bereits berechnete Informationen zu nutzen um die Geschwindigkeit von bildbasierten Verfahren zu beschleunigen. Der Visual-Hull-Algorithmus ist ein solches Verfahren und basiert auf Silhouettenbildern. Zusammen mit Texturierungsverfahren erzeugt er neue Bilder einer Szene von beliebigen Blickwinkeln. Um die Effizienz dieses Algorithmus zu erhöhen, wenden wir das Wissen über die beobachtete Szene an. Benutzer bewegen sich gleichmäßig und langsam in Relation zur Kamerabildrate. Zwischen einzelnen Bildern der Sequenz kommt daher häufig eine so genannte zeitliche Kohärenz zustande. In dieser Dissertation beschreiben wir neuartige Verfahren um die zeitliche Kohärenz in den Videodaten auszunutzen um die Geschwindigkeit der Bildgebung zu erhöhen. Dazu wurden Verfahren entwickelt um über die Zeit stabile Daten zu identifizieren. Diese Daten können verwendet werden, um bildbasierte Verfahren substantiell zu beschleunigen.

Durch diese Entwicklungen wurde es im Laufe der Arbeit möglich eine Reihe von neuen Anwendungsgebieten, zum Beispiel Videokonferenzsysteme oder Unterhaltungsgeräte mit beweglichem Blickwinkel, zu erschließen. Fortgeschrittene Anwendungen, wie zum Beispiel virtuelle Umkleidekabinen oder realistische virtuelle Umgebungen, erfordern es, den Benutzer zusammen mit virtueller Kleidung und Gegenständen darzustellen. Zu diesem Zweck entwickelten wir den Prozess der *bildbasierten Augmentierung*. Dieser erlaubt es ein aktuelles Bild des Benutzers mit vorab erstellten bildbasierten Repräsentationen der gewünschten Objekte in Echtzeit zu erweitern. Ein Vorteil dieses Verfahrens ist die kurze Zeit, die benötigt wird, um ein Objekt, zum Beispiel ein Kleidungsstück, zu digitalisieren. Durch Kombination mit den im Laufe dieser Dissertation entwickelten bildbasierten Anzeigeverfahren geht ein virtuelles Objekt nahtlos in das Bild des Benutzers über und ermöglicht somit einen bisher unerreichten Grad an Realismus.

Acknowledgments

I am very grateful to my advisor, Prof. Gerhard Reitmayr, and to Prof. Dieter Schmalstieg for teaching me how to conduct research and develop my own ideas. I greatly appreciate the scientific and social atmosphere at the institute. Moreover, I am very thankful to my colleagues and co-authors, Matthias Straka, Bernhard Kainz and Markus Steinberger for their collaboration on numerous experiments and papers.

In particular, I want to thank my parents, Barbara and Kurt, who taught me the importance of education and encouraged and supported me throughout my entire life. Finally, I want to thank Angelika for her love and understanding. This thesis is dedicated to Angelika and my parents, who gave me the personal strength to succeed in life.

This work was supported by the Austrian Research Promotion Agency (FFG) under the BRIDGE program, project #822702 (NARKISSOS).

Contents

List of Figures

Chapter 1

Introduction

Contents

In the past few decades, the field of computer graphics has evolved tremendously. Computer-generated images can be indistinguishable from photographs. Even for interactive applications, where subsecond computation times are required, the visual quality reaches realistic levels. However, computer graphics are computed from synthetic models that are typically created by artists. While these models can be textured with photographs, the process is slow and is performed as part of the development of an application. Thus, it is typically not possible for users to incorporate their own appearance into such applications.

During the same period of time, the field of computer vision also progressed considerably. In particular, precise and robust methods of reconstructing surface models from photographs emerged. Among other applications, these algorithms can be used to create three-dimensional (3D) models of humans. Such models can be displayed with computer graphics methods to allow users to appear in virtual worlds or to be surrounded by virtual objects. Moreover, users can watch themselves from all sides with smooth viewpoint motions.

1.1 Problem statement

The goal of surface reconstruction is to compute 3D models from two dimensional (2D) images. This task can be regarded as the inverse of computer graphics and its image synthesis process called *rendering*. As an inverse problem, surface reconstruction is conceptually harder and more time consuming than image synthesis, among other issues. However, interactive reconstruction in combination with rendering increases the sense of

immersion by allowing users to see themselves and others moving and interacting in virtual worlds.

Interactive reconstruction and rendering of humans is an important technology for a number of applications, including interactive mixed reality systems that allow users to see their bodies embedded in virtual worlds for entertainment purposes. Alternatively, teleconferencing systems with 3D rendering of the participants, often referred to as telepresence, can create a more realistic form of remote communication. The major goal of this project was to create a *magic mirror* that enables a virtual try-on application. This application allows users to watch themselves from all sides wearing different garments on the screen. Despite the project's focus on a specific application, the methods described in this thesis are suitable for all of the aforementioned applications.

Due to the increased sense of immersion, these real-time systems are more useful than static reconstructions. However, the latency between capturing the real world and producing the output image must be as low as possible for such applications for a number of reasons. First, input precision benefits from a responsive system. Second, too much latency may cause motion sickness. Finally, jitter in the output is simply undesired. Thus, 3D surface reconstruction and rendering must be as fast as possible while the visual output quality must be sufficiently high to faithfully reproduce the user's appearance.

To achieve this goal, we develop reconstruction and rendering algorithms that quickly deliver images with high visual quality.

At the beginning of our project we installed ten color cameras to be able to obtain high-resolution images of the user from all sides. Later, we extended the setup with depth sensors and built another room equipped with up to seven Microsoft Kinect color and depth cameras. Based on these two setups, this thesis introduces a range of novel and improved rendering methods that create images of the user with seamless viewpoint motions.

We focus on image-based rendering methods that combine reconstruction and rendering in a single step. To achieve the highest efficiency, we suggest exploiting the temporal coherence in the videos to avoid recomputing every output image from scratch. Finally, for our virtual try-on application we want the rendered images of the user and virtual garments to blend seamlessly. Our goal is that it should not be possible to tell whether a garment is real or not. Moreover, we want to be able to digitize existing garment models with the same hardware setup to avoid the labor-intensive task of modeling clothes. Therefore, this thesis sets out to verify the following hypotheses:

1. **Image-based visual hull rendering** is suitable for free viewpoint rendering of moving users and outperforms voxel-based reconstruction and rendering methods. Moreover, it can be improved by depth sensors and lends itself to parallel execution.

2. Image-based visual hull rendering can be accelerated by exploiting the **temporal coherence** of the input data.

3. Previously captured images can be used to augment users with garments. These **image-based augmentations** can follow the user's motions and blend in seamlessly.

The following sections describe our hypotheses and how we evaluate these hypotheses by implementing technical solutions.

1.2 Image-based visual hull rendering

For surface reconstruction from color cameras we choose the shape-from-silhouette class of algorithms. Using silhouettes is a natural decision because we have a static background and the algorithms are much quicker than stereo matching based on color consistency. Voxel carving is a common method for shape-from-silhouette reconstruction due to its simplicity and high level of performance.

The chained execution of reconstruction and rendering creates a pipeline that takes images as the input and produces an image as the output. Image-based rendering methods achieve the same goal but circumvent intermediate data representations, such as mesh models or voxel grids. When immediate rendering is desired and a mesh or other data structure is not needed by other components, image-based rendering methods can be faster than a reconstruction and rendering pipeline.

From the class of shape-from-silhouette methods, we choose the image-based visual hull (IBVH) algorithm, which creates an output depth map directly from silhouette images without an intermediate representation. It therefore only computes the parts of the surface that can be seen from the desired viewpoint and computes exactly one surface sample per pixel. This behavior is particularly useful for processing video streams with low latency. We hypothesize that in low-latency applications, it is more efficient to use IBVH rendering than a sequence of voxel-based reconstruction and rendering for image generation. In Chapter 3 we describe and evaluate this method.

IBVH rendering is an output-driven algorithm similar to raycasting. Therefore, it is suitable for parallelization across multiple graphics processing units (GPUs). In addition to standard parallelization methods, we suggest multi-frame rate rendering for an even higher level of performance (see Chapter 4).

For view synthesis from a combination of color and depth cameras, we introduce the concept of visual-hull carved point cloud rendering. This method utilizes many of the advantages of depth sensors, including easier foreground segmentation and the ability to capture concave regions, while also benefiting from the robustness of visual hulls and their ability to hide the background scene (see Chapter 5).

1.3 Temporal coherence in IBVH rendering

The suggested view synthesis methods require quick IBVH rendering to avoid latency. Previous IBVH implementations computed every output image from scratch and thus did not maximize the efficiency of the system. When users pose and move in a multi-camera system, their motions are typically smooth and slow compared to the camera frame rate. This redundancy in the input video stream results in *temporal coherence*.

We hypothesize that exploiting the temporal coherence in input video streams makes IBVH rendering more efficient.

Our first method to exploit temporal coherence focuses on reusing visual hull patches that remain static between two consecutive output frames. The major challenge is to identify and distinguish static from moving regions before reconstructing the visual hull. We achieve this goal by applying the IBVH algorithm to silhouette difference images to create a motion detector that estimates a motion magnitude for each output pixel. This detector can be used to decide whether a block of pixels should undergo a full IBVH rendering pass or a more simple image-based rendering method. We use forward image warping as an efficient way of reusing previous IBVH rendering results (see Chapter 6).

Image warping is limited to static surface patches. Exploiting coherence for surface patches that move or deform over time is more difficult. Due to the unpredictability of deformation, previous depth maps can not be reused reliably. A large fraction of visual hull surface points (computed from the depth maps) can become invalid between two consecutive frames. The IBVH algorithm computes the depth map directly from the silhouette images, which means that without modifications of the original approach, there is no stored data that stays valid over two consecutive frames. We analyzed the IBVH algorithm and discovered several intermediate computation results that can be reused from the last frame to reduce the number of computations. The resulting IBVH rendering methods are more efficient than the original algorithm by exploiting temporal coherence (see Chapter 7).

1.4 Image-based augmentations

After developing efficient image-based rendering algorithms, a range of applications can be realized, such as telepresence systems, virtual mirrors or free viewpoint rendering in entertainment applications. For the advanced mixed reality scenario of our virtual try-on project, the rendered image of a user must be augmented with virtual garments and accessories.

We hypothesize that the combination of capturing garments with cameras and image-based rendering of the captured data enables a realistic virtual try-on application with free viewpoint rendering. To achieve this task, we introduce an image-based augmentation pipeline that uses previously recorded sequences to augment users. The appearance of a piece of clothing is transferred from one user to another. Such an approach is challenging in many respects. First, the best-fitting garment images must be found for the user's current pose. Next, the images must be rendered coherently and registered to overlap realistically. Finally, the images of the user and the garment must be composed to form a coherent output image.

A major advantage of using recorded garment sequences over manually modeled meshes is the short content creation time, which is on the order of minutes. Image-based rendering and image processing can be used to achieve a realistic augmentation quality at interactive frame rates (see Chapter 8).

With the image-based rendering and augmentation methods described in this thesis, we were able to verify these hypotheses. In the application section, Chapter 9, we show a selection of applications that use these methods. Chapter 10 concludes the thesis by summing up the contributions and achieved results.

1.5 Publications about this thesis

The content and contributions of this thesis are based on a number of publications with several co-authors. We grouped them into primary publications whose contents can be found in its entirety in this thesis. Secondary publications are partly contained in this thesis. The other publications of the author are not directly related to the thesis.

1.5.1 Primary publications

In this section the primary publications are listed along with their chapter in this thesis and their contribution to it.

S. Hauswiesner, M. Straka, G. Reitmayr. *Coherent image-based rendering of real-world objects.* Symposium on Interactive 3D Graphics and Games (I3D) 2011, San Francisco, California [90]

In this paper we introduced a motion-detector that discriminates static from moving visual hull patches before executing the IBVH algorithm. This allowed us to reuse static regions from the last frame by image warping (Chapter 6). As a result, the visual hull reconstruction process became more efficient by exploiting temporal coherence, particularly when the user moved rather slowly or only parts of his body. In addition, the paper contains several detail improvements of our efficient implementation of the IBVH pipeline (Chapter 3).

S. Hauswiesner, R. Khlebnikov, M. Steinberger, M. Straka, G. Reitmayr. *Multi-GPU Image-based Visual Hull Rendering.* Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization (EGPGV) 2012, Sardinia, Italy [89]

In this work we introduced several methods to parallelize the IBVH algorithm over two to four GPUs. To further increase the performance and achieve a better scaling behavior with the output resolution, we suggested a multi-frame rate approach that decouples viewing from rendering without synchronization (Chapter 4).

S. Hauswiesner, M. Straka, G. Reitmayr. *Temporal Coherence in Image-based Visual Hull Rendering.* Under major revisions at: IEEE Transactions on Visualization and Computer Graphics (TVCG), Year 2012 [93]

In this article we developed the temporal coherence idea further. In contrast to our previous method [90] we found ways to exploit coherence even for dynamic/moving visual hull patches. To achieve this we analyzed the IBVH pipeline to find intermediate computation results that stay valid between two consecutive frames. These results can be reused to avoid unnecessary computations (Chapter 7).

S. Hauswiesner, M. Straka, G. Reitmayr. *Image-based Clothes Transfer.* Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR) 2011, Basel, Switzerland [92]

and

S. Hauswiesner, M. Straka, G. Reitmayr. *Virtual Try-On Through Image-based Rendering.* Under major revisions at: IEEE Transactions on Visualization and Computer Graphics (TVCG), Year 2012 [94]

In this paper that was later extended to a journal article we introduced an image-based augmentation pipeline. It works by capturing users wearing a piece of clothing to augment other users. The capturing process is very quick and cheap compared to manual 3D modeling. The augmentations are created by IBVH rendering and image processing to adapt the garments shape to the new user. This way we achieve the realism that is inherent to image-based rendering and correct overlap between user and garment (Chapter 8).

1.5.2 Secondary publications

These publications are partly contained in this thesis.

S. Hauswiesner, D. Kalkofen, D. Schmalstieg. *Multi-Frame Rate Volume Rendering.* Eurographics Symposium on Parallel Graphics and Visualization (EGPGV) 2010, Norrköping, Sweden [88]

This paper introduces multi-frame rate volume rendering: a quick way of parallelizing the rendering of complex volumetric datasets at the cost of visual fidelity. The parallel, hardware-accelerated image warping implementation that is described in Chaper 6 was initially developed for this approach. Volume rendering in the form of raycasting shares important properties with IBVH rendering: it is an independent per-pixel operation and computationally expensive. Therefore, improvements like parallelization can be used for both algorithms (Chapter 4).

M. Steinberger, B. Kainz, B. Kerbl, **S. Hauswiesner**, M. Kenzel, D. Schmalstieg. *Softshell: Dynamic Scheduling on GPUs.* ACM SIGGRAPH Asia 2012 papers, Singapore [197]

For this work Markus Steinberger et al. developed a GPU-based scheduler that enables more complex execution configurations than the conventional Nvidia compute unified device architecture (CUDA). For the guaranteed frame rate approach described in Section 6.2 we used the scheduling framework's ability to make decisions depending on the elapsed time during kernel execution.

B. Kainz, **S. Hauswiesner**, G. Reitmayr, M. Steinberger, R. Grasset, L. Gruber, E. Veas, D. Kalkofen, H. Seichter, D. Schmalstieg. *OmniKinect: Real-Time Dense Volumetric Data Acquisition and Applications.* Symposium on Virtual Reality Software and Technology (VRST) 2012, Toronto, Canada [108]

For this work we designed a room equipped with up to seven Microsoft Kinect devices. To avoid interference between the infrared projectors, all devices are mounted on oscillating poles. We used this setup to compare point cloud rendering to visual hull rendering and to develop a combined algorithm (Chapter 5).

M. Straka, **S. Hauswiesner**, M. Rüther, H. Bischof. *A Free-Viewpoint Virtual Mirror with Marker-Less User Interaction.* Proceedings of the 17th Scandinavian Conference on Image Analysis (SCIA) 2011, Ystad Saltsjöbad, Sweden [200]

In this work the multi-camera setup used in this thesis is introduced along with the processing and display hardware. The low-level computer vision components are described, like background segmentation and compensation for radial image distortion (Chapter 3).

S. Hauswiesner, M. Straka, G. Reitmayr. *Free viewpoint virtual try-on with commodity depth cameras.* Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry (VRCAI) 2011, Hong Kong, China [91]

In this paper we suggested a simplified virtual try-on application that users can run on their home PCs. It extracts a virtual avatar mesh by using the multi-camera system that is used throughout this thesis. This avatar can be taken home by users and animated by skeleton tracking from a Microsoft Kinect sensor. In addition to the multi-camera setup, the application shares the view-dependent texturing implementation with this thesis.

1.5.3 Other publications

The author participated in different teams to research various topics that are not directly related to this thesis.

J. K. Mühl, B. Kainz, A. Bornik, M. Grabner, **S. Hauswiesner**, D. Schmalstieg. *The Future of Volume Graphics in Medical Virtual Reality.* World Congress on Medical Physics and Biomedical Engineering, IFMBE Proceedings 2009, Munich, Germany

B. Kainz, M. Grabner, A. Bornik, **S. Hauswiesner**, J. K. Mühl, D. Schmalstieg. *Efficient Ray Casting of Volumetric Datasets With Polyhedral Boundaries on Manycore GPUs.* Transactions on Graphics, ACM SIGGRAPH Asia 2009, Yokohama, Japan

A. Hartl, L. Gruber, C. Arth, **S. Hauswiesner**, D. Schmalstieg. *Rapid Reconstruction of Small Objects on Mobile Phones.* Proceedings of the Embedded Computer Vision Workshop 2011, Colorado Springs, USA

B. Kainz, M. Steinberger, **S. Hauswiesner**, R. Khlebnikov, D. Schmalstieg. *Stylization-based ray prioritization for guaranteed frame rates.* Proceedings of the Non-photorealistic Animation and Rendering Symposium (NPAR) 2011, Vancouver, Canada

M. Straka, **S. Hauswiesner**, M. Rüther, H. Bischof. *Skeletal Graph Based Human Pose Estimation in Real-Time.* Proceedings of the British Machine Vision Conference (BMVC) 2011, Dundee, Scotland

M. Steinberger, B. Kainz, **S. Hauswiesner**, R. Khlebnikov, D. Kalkofen, D. Schmalstieg. *Ray Prioritization Using Stylization and Visual Saliency.* Computers and Graphics, Year 2012

S. Hauswiesner, P. Grasmug, D. Kalkofen, D. Schmalstieg. *Frame Cache Management for Multi-frame Rate Systems.* Proceedings of the 8th International Symposium on Visual Computing (ISVC) 2012, Crete, Greece

M. Straka, **S. Hauswiesner**, M. Rüther, H. Bischof. *Simultaneous Shape and Pose Adaption of Articulated Models using Linear Optimization.* Proceedings of the 12th European Conference on Computer Vision (ECCV) 2012, Firenze, Italy

M. Straka, **S. Hauswiesner**, M. Rüther, H. Bischof. *Rapid Skin: Estimating the 3D Human Pose and Shape in Real-Time.* Proceedings of the International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT) 2012, Zürich, Switzerland

1.6 Collaboration statement

Aside from the supervisor Prof. Gerhard Reitmayr, a number of colleagues contributed to the hardware setup, software modules, publications and experiments that this thesis is based on. In this section the most important collaborators are mentioned.

Matthias Straka was the closest collaborator and involved in the design of the hardware setup and first computer vision modules. During the virtual try-on project, Matthias Straka and the author of this thesis worked closely on many software modules and most publications.

Dr. Bernhard Kainz designed and built the OmniKinect setup, a room equipped with up to seven Microsoft Kinect devices. We used it to compare our IBVH method to point cloud rendering and develop combined algorithms to utilize both color and depth sensors. The collaboration lead to a joint publication *OmniKinect: Real-Time Dense Volumetric Data Acquisition and Applications* [108].

Markus Steinberger developed a GPU-based work scheduler that allows CUDA programs to make decisions based on the elapsed computation time. The IBVH method described in this thesis can utilize this framework to guarantee a certain frame rate when quality degradation is acceptable. The collaboration lead to a joint publication *Softshell: Dynamic Scheduling on GPUs* [197].

Other collaborators include Prof. Dieter Schmalstieg, Prof. Horst Bischof, Dr. Denis Kalkofen, Dr. Matthias Rüther, Dr. Raphael Grasset, Dr. Hartmut Seichter, Dr. Eduardo Veas, Dr. Judith Mühl, Dr. Markus Grabner, Dr. Alexander Bornik, Dr. Clemens Arth, Rostislav Khlebnikov, Andreas Hartl and Lukas Gruber from the Institute for Computer Graphics and Vision. Moreover, some experiments and software modules were developed by the lab students Christoph Bauernhofer, Michael Kenzel, Bernhard Kerbl, Philipp Grasmug and David Fandler.

Chapter 2

Related work

Contents

Magic mirrors and virtual try-on systems need to sense the position and pose of the user and capture his or her body shape to react on motions and create mixed reality output images. Therefore, this chapter focuses on reconstruction and image processing methods.

Previous work in the field can be categorized according to the data representation that is reconstructed from the camera images. Frequently, an explicit data representation, like meshes or voxel grids is extracted. This is usually called 3D reconstruction and the topic of the next section. When immediate rendering of novel viewpoints is required, the detour of explicit geometry can often be avoided to reduce latency. Computing novel views directly from other images is called image-based rendering (Section 2.2). Reconstruction and image-based rendering methods are computationally expensive. When image sequences showing smooth object or user motions are processed, it is often possible to reuse information from previous images. This is called temporal coherence (Section 2.3). The last section in this chapter lists previous virtual try-on systems, which were mostly based on CAD-modeled meshes.

2.1 3D reconstruction

To display a user and virtual garments from all sides it is necessary to obtain a three-dimensional geometry that describes the user's body surface. Such a surface can be reconstructed from active or passive sensors [34].

Active sensors send a signal into the scene and measure its reflection to compute a depth map. These sensors can operate in a wide range of lighting conditions, but multiple sensors might interfere [147]. Unlike conventional cameras, they do not provide color images that are useful for texturing the reconstructed surfaces.

Conventional color cameras are passive sensors. Many of them can be operated at the same time because they do not interfere. Geometric information can be obtained when two or more sensors see a moving object at the same moment in time. The camera's relative location and viewing directions need to be known (*calibrated*) before geometry can be reconstructed. Given the calibration, pixel locations in both cameras that show the same point can be triangulated to obtain a 3D point on the surface. This is called shape-from-stereo [83, 181].

In shape-from-stereo, pixel locations showing the same part of the scene are called corresponding points or correspondences. These are usually found using matching algorithms that analyze the color values in the images or derived values, like gradients, histograms etc. Given enough computation time and texturedness of the scene, these methods reconstruct surfaces with remarkable quality [58, 139]. This method was used to reconstruct and track human body parts [33].

A shape-from-stereo variant called shape-from-silhouette reconstructs an object from its foreground/background segmentation [8]. While this method requires an additional segmentation step, it is usually quicker than shape-from-stereo and delivers water-tight and smooth surfaces called visual hulls. Visual hulls can be incorrect at concavities, which can be fixed by photo-consistency checks [182].

The general field of 3D reconstruction from images is very large. In addition to shape-from-stereo and shape-from-silhouette, it includes photometric stereo [21, 103] which requires a specific light setup for reconstruction. Shape-from-focus [49, 161] and shape-from-zooming [121] approaches need special camera hardware or a controlled motion of the reconstructed object. Marker or pattern-based methods [226] require a specific pattern on the reconstructed surface. Due to these limitations, this section describes previous work on selected topics like clothes and human body surface reconstruction. Then we describe the shape-from-silhouette approach, which is faster than shape-from-stereo and often used in interactive systems. Finally, we show previous work on active depth sensors which are becoming increasingly cheap and therefore popular.

2.1.1 Human body model reconstruction and motion capture

Reconstruction algorithms that are specialized on human bodies frequently adapt a template mesh to the camera images. At the same time, the user's body pose needs to be determined [4].

Motion capture, or human pose tracking, is the task of determining the user's body pose. It usually involves a pose and shape model that is fitted to sensor data and therefore comprises a model-based tracking problem. By using markers, tracking is possible from a single color camera [174]. Our system and the following related works only consider optical, marker-less pose tracking because we assume that any sort of markers is too obtrusive for virtual try-on applications.

Marker-less pose tracking and shape adaption is usually achieved by using multiple cameras [5, 7, 25, 37, 38, 59, 101, 102, 193, 204, 207, 215]. These systems first adapt a template skeleton to the observed image data by adjusting pose parameters and comparing the rigged template mesh to the images. Then the mesh shape is adapted to the observation, which in turn yields better a skeleton pose. The process is repeated until

convergence. Often, visual hulls are reconstructed from the camera images to guide the adaption. The shape and pose adaption is often formulated as an optimization problem to achieve robustness against noisy or wrong input data. For example, optical flow and silhouette constraints [205] can be used to refine the model.

Recent systems can track a user's pose even from single video streams [225], but typically can not provide accurately reconstructed geometry which makes them not relevant for our work.

Shape reconstruction and motion capture can be used to create new animations of humans [143, 195]. New animations can be embedded into the original video, which allows to change the shape of humans in videos [106]. Image-based rendering can be used for realistic rendering of such sequences [232]. A system for marker-less human motion transfer [28, 29] was suggested to transfer motions from one person to another. This can be seen inversely: the appearance of a user is transferred, which makes it similar to a virtual try-on application.

Most of the systems described above require a human template model. It usually consists of a mesh, a skeleton definition and skinning weights that describe how mesh and skeleton joints are connected. The SCAPE model is a popular template [4] and is used by some of the works above for adaption. Another parametric model was introduced [3] for real time animation. A humanoid animation standardization project called hAnim * provides a format to describe human joints, skeleton parts and feature points and sample datasets.

Recent developments in sensor technologies have enabled the acquisition of depth images in real-time, which opened up new possibilities for pose tracking with a single camera. Full body motions can be tracked by using a time-of-flight camera [60]. The more recent Microsoft Kinect camera allows for real-time recording of color and depth images at a very low cost, as well as high-quality real-time human pose estimation [183]. Tracking by synthesis [167] works by offline rendering of an articulated template mesh into a database and find the database entry with the lowest error to a range image.

Systems with laser scanners can provide a better avatar quality [2, 140, 233], but suffer from shortcomings. For example, most of these systems can only capture static models instead of motion sequences due to the slow scanning speed.

2.1.2 Visual hull reconstruction

Extracting geometry from a set of calibrated camera images when a foreground segmentation is available is a common technique in computer vision called shape-from-silhouette or space carving. Many systems use it to reconstruct a surface geometry estimation called the visual hull [120]. Visual hulls enclose the true object's surface, but do not fit tightly everywhere: concavities that are not contained in any of the silhouette images are filled. Therefore, visual hulls often serve as a first estimate of the surface and are refined by photoconsistency algorithms [182].

For a 3D production studio [194], the type and arrangement of cameras, calibration, background material and light setup were explored to optimize the quality of a visual hull

*`http://h-anim.org`, retrieved 2012-12-18

reconstruction system. Prior to visual hull reconstruction, the foreground or background of the camera images need to be segmented.

Foreground segmentation

Segmentation is the process of assigning labels to pixels. For shape-from-silhouette reconstruction, each pixel should be classified as either foreground or background. This can be achieved by simply subtracting the known background image from the current image. Differences indicate foreground and can be thresholded to obtain a binary classification. However, this method is not very robust, even when the operation is performed in illumination invariant color spaces. To achieve a more robust behavior in slightly moving background regions, several approaches have been suggested [48, 196] that estimate background models. Visual hulls can be computed despite static occluders in the scene [79]. Other robust techniques rely on optimization methods, for example, graph-cut segmentation [18, 148].

In a multi-camera scenario, it is possible to take advantage of the fact that all cameras observe the same foreground object. Visual hull reconstruction of intermediate segmentation results can be used to create priors that can be applied iteratively until the segmentation converged [42, 51, 67, 124, 125]. Visual hull approaches can be extended to detect and track multiple objects or users [77, 78] by learning their appearance. Automatic calibration and synchronization can be performed by analyzing the silhouette images [17, 97, 186].

Visual hull reconstruction methods can be divided into three groups. Voxel-based approaches are most common due to their simplicity. Mesh-based methods can provide higher resolutions because only the surface is calculated and stored. The third group consists of methods that utilize the conventional hardware-accelerated rendering pipeline of modern GPUs for reconstruction.

Voxel carving and voxel coloring

Voxel carving is a well known shape-from-silhouette technique [152]. It works by projecting all voxel locations onto all camera image planes. If any projected location of a voxel falls on a background pixel the voxel is discarded. The remaining voxels describe the object's spatial extent, including surface and interior. The interior of the object, however, is not of interest and subsequent processing usually skips the interior voxels. Similarly, all the discarded background voxels are unnecessary. This redundancy is a major disadvantage because the method requires much memory and even additional computations for skipping over interior and background voxels. Constructing octrees [170] was one of the first ways of alleviating the problem.

Many voxel carving systems utilize small clusters of personal computers (PCs) to achieve higher voxel resolutions by parallelization [68, 206, 230]. Other systems [165] are implemented as shader programs to use GPU hardware-acceleration. A recent voxel-based approach using GPUs on four PCs was shown [118]. Other systems use a PC network just for camera connectivity [211].

When voxel carving is extended by photoconsistency measurements, the result is often called voxel coloring [117, 182]. This method extends the voxel carving approach by com-

paring the color values at the projected voxel locations. Voxels are marked as background if these colors do not match. Recent implementations [104] also utilize GPU hardware.

Voxel-based visual hull reconstruction can be enhanced to help calibrate the cameras extrinsically [30]. Furthermore, motion capture of humans can be realized by fitting a skeleton [31] to the voxel grid.

Neither voxel carving nor voxel coloring can efficiently provide higher output resolutions [16]. Workarounds have been suggested, for example, using higher resolutions at a focus region like the user's head [115]. Other systems use low resolution voxel carving to create a rough proxy for further computations [191]. Instead of only alleviating the resolution problem, mesh-based methods solve it by computing and storing only the object's surface.

Mesh-based methods

Mesh-based shape-from-silhouette methods are more efficient at higher reconstruction resolutions because only the surface is stored (and not the background or interior of an object). The silhouette intersection tests can be performed in image space to ensure numerical stability [155]. A reconstruction algorithm [16, 50, 52, 53] triangulates the silhouette edges and discards non-surface triangles. An inverted CSG logic can be used to make the system robust to input cameras which do not capture the entire scene. Another system constructs the visual hull mesh from silhouette edges [122]. The Grimage system [54] utilizes a PC cluster for mesh-based visual hull reconstruction.

Visual hull reconstructions of complex objects can suffer from phantom surface parts. Phantoms [27, 159] can be removed by computing safe and unsafe regions for each camera image. Such regions can be identified by the number of their ray-silhouette intersections. This method requires a view-dependent visual hull reconstruction for each camera image, which makes it too slow for our purposes.

Similar to voxel carving, visual hull meshes can also be carved by photoconsistency measurements [57, 192]. The resulting image-based modeling process benefits from the robustness at the visual hull edges and from the color information between the edges. Phantoms can be removed and concavities can be restored, but the additional computations and memory transactions reduce the performance considerably.

Rendering pipeline-based

More specialized methods have been developed for visual hull rendering. The advent of powerful GPUs enabled interactive visual hull reconstruction applications. These systems usually do not use the standard space carving methods described above, but employ several hardware-specific solutions like proxy geometries and custom fragment shaders.

The work of Li [130, 134] categorizes two GPU-enabled visual hull rendering methods: CSG-based and texture-mapping based, and introduces a hybrid approach. The CSG-based approach [81, 235] uses depth-peeling of silhouette cones for correct results. The texture-mapping approach [131, 150] renders the cones and relies on alpha-tests to carve the final silhouette. The fidelity of the texture-mapping approach can be increased [123] by using silhouette maps. A visibility test was suggested [132] that works by reconstructing

the depth map for every camera image. A plane-sweep algorithm [63, 133, 137, 138, 234] can be utilized to find point-correspondences on the GPU and use it for triangulation.

2.1.3 Clothes reconstruction

Many virtual dressing applications display a textured clothes mesh over a camera image. Obtaining that mesh is a key aspect of such systems. Some approaches use meshes that were manually modeled with computer aided design (CAD) software. This process is labor intensive and requires physical simulation for animation (see Section 2.4). Clothes can also be reconstructed from images. Reconstructing the garment from a video sequence is a difficult task, especially because of occlusions and the non-rigid shape of cloth. Many approaches use markers on the cloth to tackle this problem [82, 179, 212, 226]. However, these methods can only capture the garment's shape and not its color, which makes them less suitable for our method.

More recent approaches do not require markers [19, 173, 176, 178]. They usually use a shape-from-stereo approach and apply complex processing to the data to account for occlusions. However, all approaches that rely on point correspondences that are computed from the image data assume a certain texturedness of the garment. By using a light dome [216] or a laser scanner [199] this limitation can be removed, but such hardware is expensive and processing can not be performed in real time. Some systems allow to capture garments from single cameras, but require a complex shape model to do so [209].

When garments are worn by a user during the reconstruction, a segmentation is needed to distinguish body parts and different pieces of clothing [56]. The segmentation can be guided by pose tracking. Reconstructed garment models can be used along with image-based representations [99] to increase the realism. The system selects the best fitting pose from the recorded sequence and deforms it according to a target mesh.

Once the shape of a garment is digitized, it needs to be fitted to the user's body model. This is a complex problem [129] that is usually not handled in real time. Volumetric Laplacian deformation [128, 223] can achieve this, but it requires correspondences which can be hard to define.

2.1.4 Interactive systems

Our mixed reality methods were implemented in an interactive system that reacts to user motion with little latency. Such systems have been built for the purpose of teleconferencing and 3D video capturing.

The Blue-C is an augmented reality system which includes 3D video acquisition and visual hull rendering [75]. It features of a small room consisting of display panels at all sides. The panels can become transparent for short periods of time to allow the user to be captured by outside cameras. The system can be used for telepresence, 3D video, entertainment etc. Moreover, a concept for a flexible virtual shopping system was introduced [119]. It includes ideas on how 3D remote viewing can be used for virtual try-on, car shopping etc.

The Grimage [1, 168] is a telepresence system which uses mesh-based visual hull reconstruction before rendering. The 3D Live system [172] builds on IBVH rendering, but

it uses a rather simplistic sampling approach along each viewing ray. This causes many projections and texture lookups, resulting in bad scaling with higher resolutions. Another approach is voxel-based [84, 85] and uses marching cubes to display the surface. Billboards [69] can also be used for display. Due to the voxel- or mesh reconstruction step, immediate 3D output with little end-to-end latency is difficult to achieve with these systems. Shape-from-silhouette was also used for rendering the head in a tele-presence application [12].

2.1.5 Active depth sensors

Active sensors send a light, laser or other electromagnetic signal into the scene and usually measure either the reflection time of the signal or a disparity vector to compute a depth map. These sensors do not rely on the existing scene light, which makes them robust even in dim lighting conditions. Integrated sensor systems perform the required time-of-flight or structured light computations on the device [110]. Due to onboard processing, they relief the central processing uint (CPU) from reconstruction computations.

However, it is often not possible to operate a number of sensors simultaneously because the emitted signals interfere with each other. The ability to use a number of sensors is particularly important if a user should be captured from all sides simultaneously. While some systems solve this by time-slicing or using different wavelength signals [76, 114], they tend to be expensive. Cheaper sensors, like the Microsoft Kinect, suffer from interference.

The capabilities of the Microsoft Kinect have been explored for a variety of applications. Kinect-based body scanning [208] enables virtual try-on applications at low costs. Newcombe et al. have shown with their work on KinectFusion [163] that dense volumetric reconstructions can be created in real time. Because the Kinect is so inexpensive, combining multiple devices has also been investigated for different research projects. For example, Wilson et al. [228] and Berger et al. [9] use up to four depth sensors to monitor a room. Both ensure that the reconstruction light patterns do not overlap, to avoid interferences of the structures light patterns emitted by multiple sensors. Maimone and Fuchs [145] propose advanced hole filling and meshing techniques to use a multi-Kinect setup for telepresence applications.

The problem of overlapping reconstruction light pattern has been solved by Maimone et al. [147] and Butler et al. [24] with a similar approach. Letting the whole RGB-D camera vibrate at a relatively high frequency blurs the light pattern for other, concurrently capturing sensors. The rigid connection of the vibrating sensor and the light pattern supports a clear reconstruction without interferences from other Kinects. In our setup, we altered this approach slightly to gain more flexibility, as detailed in Chapter 5. A similar setup was presented [146] with a modified KinectFusion algorithm.

Active sensors usually do not provide color information. Colors are useful for texturing the reconstructed surfaces. The Microsoft Kinect is equipped with an additional color camera, but it operates like a rigidly mounted external camera. The FreeCam system [116] combines color cameras and depth cameras in a system for free-viewpoint rendering. The depth hull rendering method [13] performs a depth enhanced visual hull reconstruction. A point-based variant [80] was shown. A similar method utilizes structured light [109]. Clothes texture overlay can also be achieved with commodity depth cameras [95].

The Kinect sensor and software package also features a built-in skeleton tracking algorithm. The companies AR Door [†] and Topshop use skeleton tracking to overlay virtual garments and display the result at a large TV screen. A similar system is Swivel from FaceCake [‡]. In these cheap and rather simple systems, garments and users do not have a coherent appearance and the garment motions look synthetic.

2.2 Image-based rendering

Image-based rendering (IBR) is the process of creating novel views on existing images. As such, it combines computer vision and graphics methods in a single step. The field covers a wide range of algorithms [184] that can be classified according to the image-based rendering continuum [126].

The continuum starts with purely image-based methods that do not reconstruct any three dimensional geometry or camera positions. Image morphing and view interpolation techniques are popular methods. On the other side of the continuum, an explicit geometry, like a mesh, is available or has previously been reconstructed. These methods include projective and conventional texture mapping.

Purely image-based methods require a large number of images to work properly. Conversely, geometry-based methods require a geometric description but only a low number of images [184]. In our system, we have a medium number of cameras (up to ten) and the need to operate in three dimensional space to allow for free viewpoint rendering and correct overlap between virtual objects and the user. Therefore, the image synthesis methods described below are focused on the center of the IBR continuum, where techniques like free viewpoint rendering, the image-based visual hull algorithm, image warping and view-dependent texture mapping are prominent.

2.2.1 Free viewpoint video rendering

Free viewpoint video rendering methods often operate on previously recorded video footage and add the ability to change the viewpoint [144].

Three-dimensional television (3D-TV) is an important application of free viewpoint video capturing and playback [189]. However, 3D-TV systems usually do not perform a 3D surface reconstruction that allows absolutely free viewpoint motions. Instead, only a small parallax effect is achieved by showing different images to the user's eyes. Advanced systems use camera arrays [157] for capturing and auto-stereoscopic displays. Lightfields or unstructured lumigraph rendering [22] can produce the small viewpoint and viewing angle offsets that are required to achieve a parallax effect.

Some methods are found on the image-based side of the IBR continuum, so the maximum change in viewpoint or viewing angle are usually restricted. They use image morphing and blending [46, 198, 238] between cameras for free viewpoint video. No geometry is reconstructed, even the cameras do not require calibration. These works are based on image correspondences [136] and are suitable for scenes that can not be reconstructed

[†] `http://ar-door.com/`, retrieved 2013-01-02

[‡] `http://www.facecake.com/`, retrieved 2013-01-02

reliably, like fire, fog etc. However, these methods are restricted to interpolate between existing viewpoints. Novel views outside of this range are not supported. Moreover, due to the lack of 3D information, a virtual try-on application with correct depth order is not possible.

By taking the principles of image morphing and view interpolation to the third dimension, an unrestricted image-based rendering can be built [213, 214]. These systems utilize an extended optical flow formulation called three-dimensional scene flow. It describes how surface patches move over time in 3D space. Unfortunately, these methods are computationally expensive.

Other methods are less restricted in viewpoint motion and include a three-dimensional surface reconstruction. 3D video recording and playback can be realized using point cloud reconstruction and rendering [231] or billboard clouds [62, 224] to create a remote viewing application that can display humans.

Smooth and reliable reconstruction can be achieved by a particle-based optimization approach [105]. This method benefits from GPU hardware acceleration.

A recent system combines reconstruction and video-based rendering [6] to navigate even casually captured videos in space and time. It relies on image-based methods to enable realistic transitions between the cameras. In another system, cameras can move during capturing [202].

Variational methods can be used for deblurring and super-resolution [66] or segmentation and surface reconstruction [64] in a multi-camera reconstruction and rendering setup.

Free viewpoint rendering methods allow users to view previously recorded sequences. Such sequences can be augmented with additional information, like overlays or textures.

2.2.2 Image and video retexturing and overlays

Augmented reality (AR) systems produce overlay images that are registered with the original image or video. This section focuses on a subclass of systems that change the appearance of clothes or augment humans with new virtual clothes. Image or video retexturing is a form of AR that uses a video background, but it does not insert new virtual objects to the scene. Instead, an object of the real world is augmented with new surface properties, like a new texture. This requires a very good spatial registration of the image and the new texture.

Completely retexturing a piece of clothing with realistic draping of textures to photographs [229] is a complex process and requires the knowledge of seams, contours and orientation of the desired garment. It is an artistic approach that is not suitable for interactive, automatic systems.

Simpler systems therefore make assumptions about the original garment that is worn by the user. Some use markers on the garment to replace the texture [20]. Later approaches do not need markers, but require a known texture [169, 227] of the original piece of cloth. Some systems can even handle self-occlusions of cloth [61]. Recent systems do not require a known texture, but a general texturedness for tracking. This enables interactive magic mirror applications where users wearing non-textured shirts with a printed region can see their shirt with a new printed logo and different color [98, 100]. These systems are

relatively cheap by using a single camera and operate in 2D, but they do not support arbitrary clothes or changing the view.

Video overlay systems are AR systems that insert new objects into a real scene that is captured by a camera. Garment overlays can be produced by capturing a user wearing the garment [43, 201] before augmentation. During runtime, garment images are selected from the recorded sequence and registered with the user by matching the silhouettes. For performance reasons, sparse features are extracted from the silhouette images. These features can be compressed to obtain even more descriptive feature vectors that allow for a real-time augmentation of moving users. This method inspired our image-based augmentation work. However, it operates in 2D on a single camera and therefore does not allow to change the viewpoint.

Other systems are specialized on specific tasks, like the augmentation of new shoes [47]. The system is interactive and relatively simple, but the analysis-by-synthesis approach that is taken here would not be suitable for more general purposes.

Retexturing and overlay systems can be used to augment users with garments, but do not provide free viewpoint rendering. As a consequence, we had to combine image-based rendering with overlays. A lightweight approach to free viewpoint IBR is image-based visual hull rendering.

2.2.3 Image-based visual hull rendering

When immediate rendering of novel viewpoints is required, the detour of explicit geometry can be avoided to reduce latency. To directly create novel views from silhouette images, the image-based visual hull (IBVH) [23, 156] method was introduced. It involves on-the-fly ray-silhouette intersection and CSG operations to recover a depth map of the current view.

Many extensions of the original approach have been developed. IBVH rendering can be improved by stereo matching [135, 187] to recover concavities. It can be enhanced to render transparent [158] objects. Other systems find body parts to improve the rendering of humans [236]. A GPU implementation and the use of silhouette segment caching can speed up intersection tests considerably [221]. Intersection testing then resolves to comparing angles, which also allows for a fast check of relevant pixels. Another method to speed up the ray-silhouette intersection testing is to chain-code the silhouette segments [113].

Recent GPU implementations [72] utilize Nvidia CUDA to avoid the overhead of a conventional rendering pipeline like OpenGL. To connect a larger number of cameras, a network of several PCs [73] can be used. Adaptive sampling was introduced to improve the performance of IBVH rendering [154].

Image-based visual hull rendering produces a depth map as the output. To produce the final color image that shows the user or a piece of clothing from a particular viewing angle, a texturing method needs to be applied to the depth map.

2.2.4 Projective and view-dependent texture mapping

Texture mapping is the process of adding details to a surface by using images [26, 96]. The mapping function defines how the image relates to the surface. In our multi-camera

scenario, the captured images are subject to a perspective distortion. Using camera images as textures can be performed by projective texturing [164, 180]: surface points are projected on the camera's image plane by applying the transformation matrix that contains the camera's intrinsic and extrinsic calibration. By looking up the color values from all the camera images at the projected locations, it is possible to determine the output color, for example, by averaging. Given the surface reconstruction of the object, it is also possible to detect occlusions and thus discard wrong color samples and improve the result. In our setup projective texture mapping is applied to the depth buffer that holds the frontmost surface points of the visual hull.

The geometric reconstruction that is textured is often not perfectly registered with the camera images for a number of reasons: calibration accuracy, noise, wrong point correspondences, insufficient sampling densities etc. To cope with these problems, view-dependent texture mapping (VDTM) [40] was introduced. It is an extension to projective texture mapping that favors images with a viewing direction similar to the currently desired view.

View-dependent texture mapping and lumigraph rendering were generalized [22] to obtain camera blending maps for texture mapping proxy geometries. This allows to compensate bad geometry by supplying more images (a more dense camera placement). Our texture mapping stage uses a variant of this technique.

Modern texturing approaches [44] are capable of compensating for an even worse geometric proxy by applying optimization procedures to align texture maps from different cameras. This can also compensate for calibration inaccuracies. It works by combining VDTM and optical flow for image registration to remove ghosting artifacts. A different approach uses an adaptive blur filter to make ghosting less apparent [45].

2.2.5 Image warping

Image warping is the process of moving image fragments according to a new projection matrix. It requires the depth values of all pixels and was introduced to compensate for jitter induced by network latency in a remote viewing application [151] and used for numerous applications.

Forward image warping uses the screen space coordinates of a pixel and its depth value to determine the position in world space, which then is projected using a new matrix. This way, pixels are moved independently of each other, which may result in holes and collisions in the output image. Image processing operations, like morphological closing, have been suggested to fix these artifacts [184].

Backward image warping operates differently. It computes the viewing ray of each pixel in the output image and intersects it with the source image plane by using the old projection matrix. In the source image the projected ray is traversed to find the surface intersection. This method does not suffer from holes, but due to the searching process it is considerably slower than forward image warping. Recent GPU implementations [71] have reached interactive frame rates. A different approach iteratively finds the correct backward warp for each pixel without searching the epipolar line [15].

2.3 Temporal coherence

Temporal coherence in the context of visual computing usually describes the situation that the rendered or captured scene changes smoothly over time [177]. For example, the viewpoint and the viewing angle in a virtual scene often move slowly. As a result, many pixels in a sequence of output images are identical and moved only slightly. This knowledge can be exploited to either reduce the runtime of an algorithm by reusing information over time [162], or to improve the quality of the result by distributing information across time [127].

Exploiting temporal coherence can improve the runtime performance of interactive raytracing. For example, the Tapestry system [185] builds a 3D mesh that acts as a cache for previous rendering results. The cache is updated incrementally and rendered instead of the scene. Adaptive frame less rendering [35] combines two efficient rendering techniques. Frame less rendering updates single pixels or patches instead of the whole image at each time step. Adaptive rendering guides this process to favor important pixels, for example, at depth discontinuities. The render cache [222] and the reverse reprojection cache [162] follow a similar notion. These systems are tailored to synthetic scenes and can not handle arbitrarily deforming objects.

In multi-camera systems that capture and reconstruct users, the input video streams often provide temporal coherence. Given a camera update rate of only a fraction of a second, the changes between two consecutive images are only minor.

For voxel-based visual hull reconstruction in multi-camera systems, temporal coherence has also been exploited [11] to improve the performance. It is the most similar method to our dynamic temporal coherence methods and works by incrementally updating a voxel grid for improved visual hull reconstruction performance. For each new frame, only the voxels whose corresponding silhouette pixels have changed are updated. This approach is limited in resolution due to using a voxel representation, and slower than our IBVH method despite the incremental update scheme.

To improve the quality of the reconstruction by temporal coherence, spatio-temporal texture atlases [107] were used to fill holes caused by occlusions and compress the camera video streams by exploiting the coherence as a form of redundancy over time. The animation cartography [203] is a similar approach. Other works use the temporal coherence [65, 70] to improve the quality of the visual hull reconstruction. These methods can fill holes in the reconstruction [127] with patches from other points in time.

2.3.1 Multi-frame rate rendering

Multi-frame rate rendering [190] decouples display updates from image generation in a pipeline with asynchronous communication. The display update stage can guarantee fast viewing frame rates and nearly latency-free response to interaction, while one or multiple rendering nodes in the backend stage can produce new high quality images at their own, usually slower pace. Rendering nodes can be multiple GPUs assembled in a single PC or workstations in a distributed system.

Multi-frame rate systems make extensive use of temporal/frame coherence because they use the last rendering result to synthesize novel viewpoints until a new image be-

comes available. Motion in the scene is allowed [188] as long as it can be described by a transformation matrix. Image warping is often used to compensate for jitter induced by latency.

2.4 Manual clothes mesh modeling

Garment mesh models can be reconstructed, as described above, or modeled manually. The modeling process can work in a very similar way to the real process of sewing garments. Two-dimensional cutting patterns are used to create patches that are connected at some of the edges. This can be achieved by a set of virtual cutting and sewing tools [112] to directly modify garments inside a framework for rendering and simulation. Physical simulation is important to transform the modeled planar patches into realistically looking meshes. The virtual sewing result can also be applied to the underlying planar cutting patterns, which makes user-driven customization possible. MIRACloth [218] is another cloth modeling application with integrated rendering and physical simulation. Special interfaces [210] facilitate clothes modeling.

The *Virtual Try-On* project was conducted [41, 219] to provide a set of applications for various tailoring, modeling and simulation tools. 3D scans of real garments with color-coded cloth are also used. Cloth properties are measured from real samples. Rule-based morphing techniques were employed to obtain different sizes of a garment piece. A surface reflectance measurement device called gonioreflectometer was used to digitize clothes very realistically. Two companies called Human Solutions and Assyst offer similar tools [§].

The OptiTex company maintains an application called *3D Runway Designer - 3D Draping Solution* [¶]. It allows the user to model and simulate garments. Moreover, it offers solutions for fabric analysis, cutting pattern creation from 3D models etc. The focus of this application lies on 3D modeling and prototyping of cloth pieces for later production.

Physical simulation of cloth patches is important for a realistic appearance as well as believable dynamic effects, like swinging and folding [217]. The simulation parameters, like cloth stretch, friction etc. can be measured from real cloth samples [142]. Information about cut and seams, however, need to be known [166].

The physical simulation process is computationally expensive. The most prominent feature of clothes are wrinkles of different sizes. A bilayered approach was proposed [111] to reduce the number of computations by approximating the wrinkles. First, a coarse mesh is simulated with a stable but slower algorithm, while an in-between mesh creates realistic wrinkles at lower cost. Other systems add the wrinkles after the physical cloth simulation [175] to achieve better performance. Another multi-layer approach [32] simplifies the simulation for tight-fitting garments. A very detailed analysis on cloth seams and wrinkles and a realistic implementation [141] was shown, but this approach does not run in real-time. Physical simulation parameters, like cloth stiffness and friction, can be estimated from videos [10]. Other systems extract linear transformation models [36] from clothes simulations to approximate the complicated cloth behavior for real time rendering.

[§] `http://www.human-solutions.com/`, retrieved 2013-01-02

[¶] `http://www.optitex.com/products/3DRunway_Tools/`, retrieved 2013-01-02

Before a physical clothes animation can be performed, the garment mesh model needs to be positioned on a body model that serves as a collision and friction proxy. Automatic prepositioning algorithms [55, 74] were developed to relieve the user from manually placing the planar patterns around the virtual model prior to the physics simulation. The method relies on a segmentation of the major body parts that is given by a shape prior and several feature points. The feature points allow for matching the pattern parts to body parts.

Chapter 3

Image-based visual hull rendering

Contents

Many mixed reality systems require realtime view synthesis from camera images, for example, telepresence applications or magic mirrors such as our virtual try-on system. To build an interactive reconstruction and rendering device, we decided to equip a room with ten color cameras to be able to capture the user from all sides with high quality [90]. In our setup we assume a static background and a single or a small number of users. These assumptions allow us to use shape-from-silhouette algorithms for reconstruction. Processing silhouettes is more efficient than stereo matching based on color consistency. From the silhouette-based algorithms we chose the image-based visual hull (IBVH) method. Figure 3.1 shows the output of IBVH rendering visualized with different shading methods.

The IBVH algorithm creates an output depth map directly from silhouette images without an intermediate representation. As such, it can be classified as an image-based rendering algorithm. It only computes the parts of the surface that can be seen from the desired viewpoint, and it computes exactly one surface sample per pixel. This behavior is particularly useful for processing video streams with little latency.

The latency between capturing the scene and producing the output needs to be as little as possible for a number of reasons. If the user should interact with the system, the input precision benefits from a responsive system. During our experiments we experienced that too much latency to the body motion is irritating and may even cause sickness. Furthermore, jitter in the output is simply undesired and lowers the sense of immersion.

Visual hull-based methods have a major drawback in terms of the reconstructed surface quality: they can not cover all concavities of the object. At the same time, the visual hull is a good geometric proxy for projective texturing because it consists of large, smooth surface patches that can be textured with little distortions.

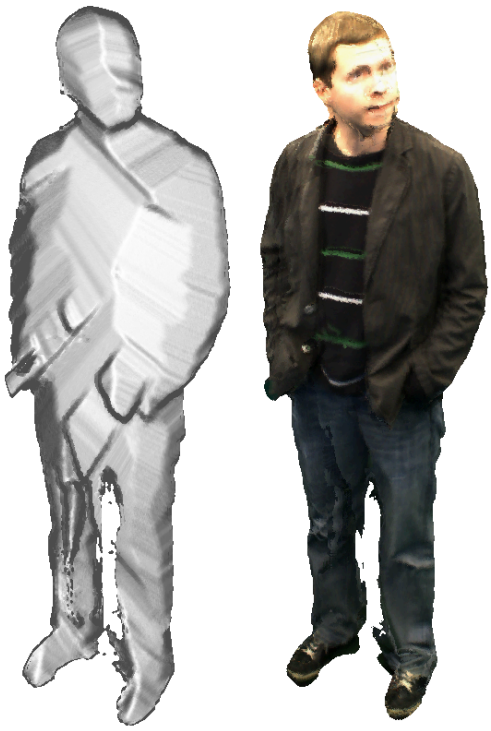


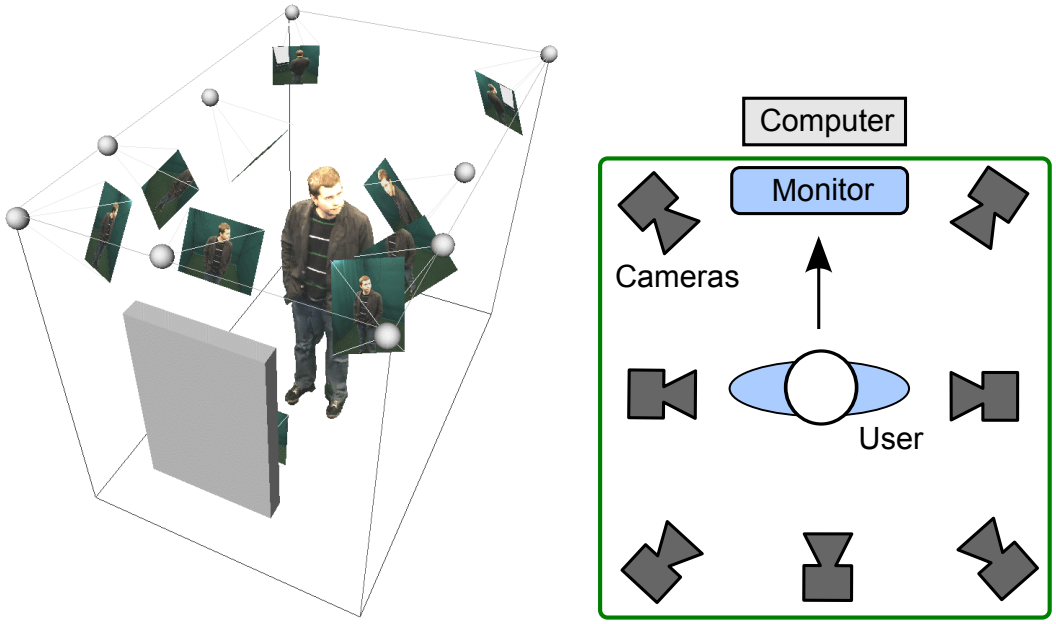
Figure 3.1: IBVH rendering of a user in our multi-camera system. Left: phong-shaded depth map, right: same depth map rendered with view-dependent texture mapping.

In this chapter, the multi-camera setup that is used for reconstruction and rendering is described and the IBVH pipeline's stages are explained in detail. The contribution that is described in this chapter includes improvements over the conventional pipeline. We describe efficient methods for silhouette segment caching, a quick interval intersection formulation that facilitates early background detection, an efficient visibility map generation method and a heuristic to determine a quick camera processing order.

3.1 The multi-camera capturing room

Our virtual mirror system requires three components: multiple cameras to capture images of the user from all sides, a computer to process the data and a monitor to show the output image to the user [200].

We needed to build a room that is sufficiently large to allow a limited number of cameras to capture the whole body of a user from all sides. At the same time, the room footprint should remain small to exploit the full camera image resolutions. The resulting proportions are a 3×2 meters footprint and a height of about 2 meters. The floor and the walls were covered with green cloth to facilitate background detection. The side walls are usually closed during capturing and rendering to guarantee a static background. Opening a side wall to allow for spectators is possible but may reduce the texturing quality for some views.



(a) (b)

Figure 3.2: An illustration of the camera arrangement used in our capturing and virtual-try on room.

At the walls we mounted ten cameras that are focused on the center of the room, where the user is allowed to stand and move. Figure 3.2 illustrates the room and the camera frustums. After a first prototype with cameras evenly spaced around the user, we mounted most cameras in the front half of the room and point at the user's head. We found that this arrangement yields a better visual quality for the user's head and face region. This was required because users were very sensitive to distortion or artifacts in this region.

The ten industrial grade color cameras are connected to a single PC via three separate FireWire buses. We decided to use a single PC to avoid the latency that would be introduced by a network of machines. Such a distributed computing network could deliver higher frame rates by pipelining all processing stages, but the end-to-end latency would increase. In early experiments we found that the latency to body motions was perceived negatively by all users, so we valued latency very highly in our design choices.

The cameras are synchronized and calibrated intrinsically and extrinsically before capturing and rendering. We obtain extrinsic camera parameters using a $1300mm$ high calibration target, shown in Figure 3.3 with StbTracker [220] targets. Each of the target's three sides ($400 \times 500mm$) shows four markers. The intrinsic calibration is obtained by capturing a checkerboard pattern [237]. The back-projection error of the resulting calibration is in the sub-pixel range, which is precise enough for image-based rendering.

They deliver 640×480 pixel images with 16 bit color resolution at 15 frames per second. The amount of data that needs to be transferred and processed can reach up to

(a) (b)

Figure 3.3: The StbTracker calibration target that is used to obtain the extrinsic camera parameters (a) and the calibration view (b) showing the detected markers. Please note that these images show the Kinect-based setup that is described in Chapter 5 which uses the same calibration target.

100 MB/s in this configuration. While the cameras could deliver higher resolutions, the buses and processors were at their limits. For processing, we exploit the computational power of modern GPUs and their programming capabilities. An Nvidia GTX 480 graphics card receives the camera images and performs all image processing and rendering stages. These stages are implemented as Nvidia CUDA kernels, which provided us with enough flexibility to realize even very complex algorithms.

The following section describes the general layout of our visual hull rendering pipeline. We then point out components which were improved over previous implementations.

3.2 The IBVH pipeline

The general architecture of our system consists of several main modules (see Figure 3.4). The pipeline starts with the segmentation module which reads the current image from all cameras in a synchronized fashion [89, 90].

Segmentation The camera images are uploaded to the GPU and segmented there by applying background subtraction. We assume a static background that is captured before the user enters the scene, and a contrasting foreground. This is achieved by using green walls inside our prototype, and users are not allowed to wear green clothing. We use background subtraction in normalized RGB space in order to handle shadows correctly. The subtracted images are thresholded and holes are closed by applying morphological dilation and erosion to the segmented image.

Edge extraction and segment caching From each silhouette image, all edges are transformed to 2D line segments. Every pixel is checked on whether it lies at the border

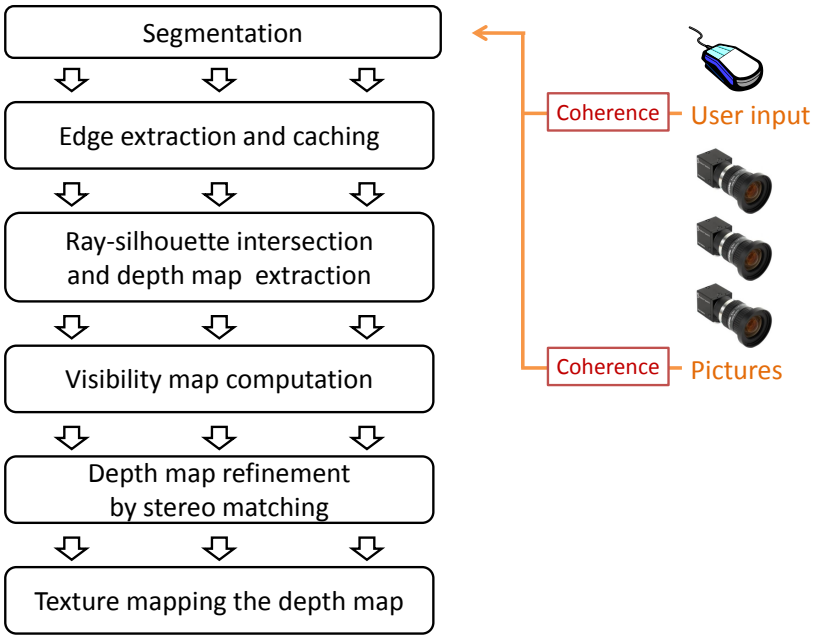


Figure 3.4: The image-based visual hull rendering process of our system without exploiting any coherence. Red shows where coherence can occur: successive camera frames may be similar, and viewpoint motion is usually smooth.

of the silhouette, and a line segment is emitted if it does. Pixels are processed in parallel, and interior silhouette edges are included because they help to identify regions where the user's body forms a loop, e.g., with his hands.

Since the IBVH algorithm performs many line-line intersection tests, a caching data structure can speed up the process of identifying potentially intersecting line segments [221]. Figure 3.5 illustrates the involved geometry. Every viewing ray can be projected onto the camera's image planes, where it can be described by an angle φ to one of the axes. The origin of the viewing rays projects to a point on every image plane, called the epipole. All projected rays emit from the epipole or converge to it. Similarly, every line segment of a silhouette lies on the image plane of the corresponding camera. It can therefore be described by two angles. Each endpoint of the line segment produces one angle: the angle between one of the axes and the vector between the endpoint and the projected virtual camera's center. If a viewing ray's angle lies between the two angles of a line segment, this line segment is hit by the viewing ray. For caching, the whole angular range is covered with a set of bins. The line segments are stored in all the bins which are covered by the range of their two angles.

Ray-silhouette intersection This is the main IBVH stage [90]. A viewing ray is created for every pixel in the output image. Rays start from the desired viewpoint and intersect the image plane at the pixel's location and end at the back plane. Each viewing ray is projected onto all of the camera images. The subsequent steps are performed for each camera image i. The line resulting from the projection is searched for intersections

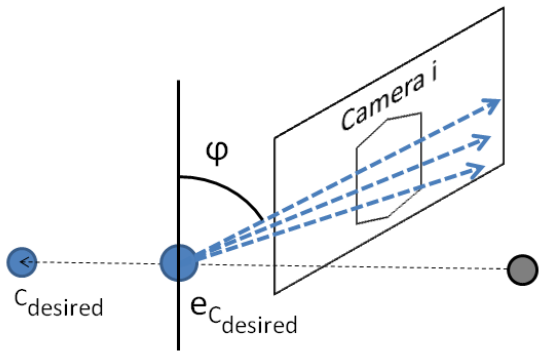


Figure 3.5: Angle φ which has been used by previous angle caching approaches.

with the user's silhouette. The angle of the line can be used to index the correct bin in the line segment cache. All segments in the bin are checked for 2D line intersection, using the line segment itself and the projection of the viewing ray onto the image plane of camera i. Intersections are sorted and lifted back onto the 3D ray by projection. Pairs of intersections build intervals that describe where the user is located along the viewing ray. These intervals have to be intersected by a set operation to compute the intervals along the ray that are located inside the object. From these intervals, the frontmost interval starts with the frontmost ray-object intersection which in turn yields the depth buffer value at this pixel. We call these values *geometry fragments*. See Figure 3.6 for an illustration.

Visibility map computation The subsequent stages for stereo matching and view-dependent texturing require a visibility map to avoid occlusion artifacts. Previous approaches either used a search along the epipolar line of each pixel to determine visibility or project the reconstructed depth map in a shadow-mapping fashion. In any case, the result of this stage is a buffer that encodes which pixel of the output image is visible for which camera.

Stereo matching Similar to other IBVH approaches, we use the visual hull as a search range for stereo matching. We calculate the similarity of pixel neighborhoods by building sums of absolute differences (SAD). For each reconstructed point on the visual hull, we calculate the SAD between pairs of nearby cameras in which the point is visible. We usually use a set of three cameras, whose viewing angle matches the surface normal best. This step greatly improves visual quality, but also has major impact on the performance (see Section 3.4). Fortunately, this step amortizes when exploiting temporal coherence because improved depth values help the image warping to produce high quality output.

Texture mapping In a final step, each point on the visual hull is textured from a set of three nearby camera images. We employ a variant of view-dependent texture mapping. Cameras which are not visible from the current point on the hull are skipped. Results from

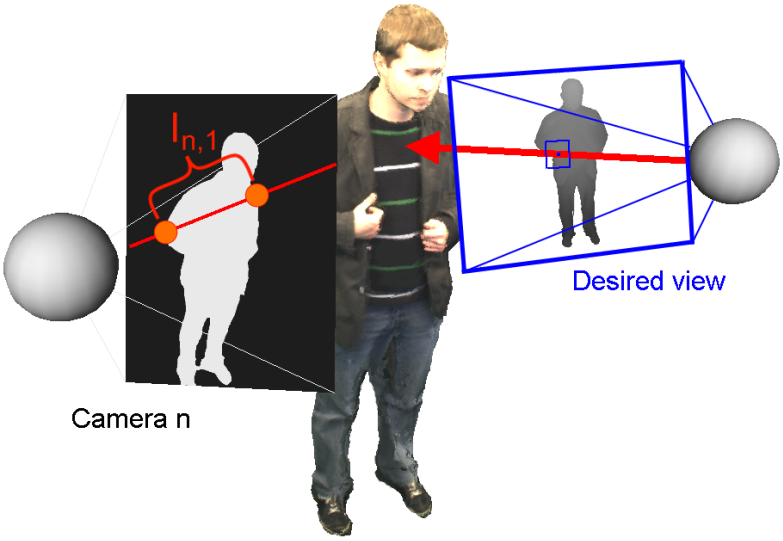


Figure 3.6: Illustrating the concept of image-based visual hull rendering. For every pixel its viewing ray (red arrow) is projected onto each camera plane (red line). All interior intervals $I_{n,1..m}$ on each ray are found and intersected with each other. The frontmost interval starts with the desired depth output value.

multiple cameras are blended according to the angle between the virtual and the physical camera's viewing directions to provide a smooth transition between camera images.

3.3 Improvements to the IBVH-pipeline

We improved several components of a conventional IBVH pipeline both in terms of robustness and performance. We describe efficient methods for silhouette segment caching, a quick interval intersection formulation that facilitates early background detection and an efficient visibility map generation method.

Finally, we observed that the angle between the cameras and the desired viewing direction has a strong impact on the foreground/background decision during rendering. Based on this observation we derived a method that reduces the execution time of the IBVH algorithm. We evaluated the effectiveness of the suggested methods under varying degrees of coherence and different resolutions.

3.3.1 Segment caching

Previous caching calculations have problems with their angle calculations when the desired viewpoint approaches a critical plane, where its epipole projects to infinity. Unfortunately, every camera in the system has such a plane. This formulation not only fails when the viewpoint is exactly on one of these planes, but also when it is close to any of the planes. There, the epipole lies at a finite point, but this point may be very distant from the image plane. As a result, the angular range of the whole silhouette falls within a tiny interval and precision is irreversibly lost. When line segments are added to the cache in such a scenario,

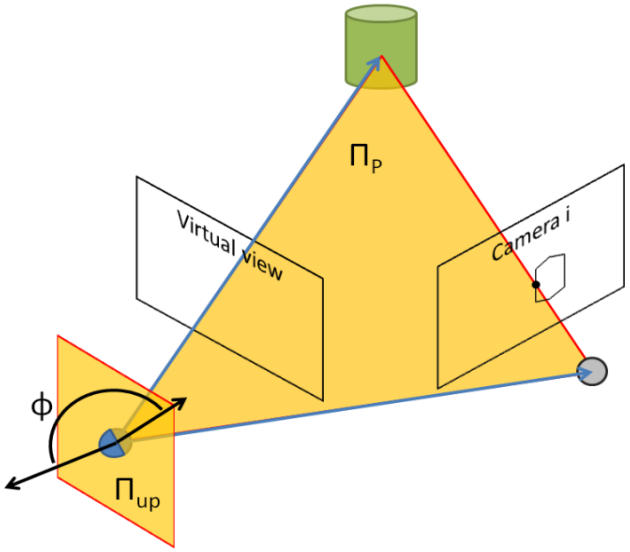


Figure 3.7: Our angle caching approach uses the dihedral angle ϕ which is spanned by the planes Π_P and Π_{up}.

many segments are assigned to the same bin which either causes an overflow of these bins, or in the case of very large bin sizes, in bad performance. Other than Waizenegger et al. [221], we observed this problem to be fairly common in our system.

The point where this formulation becomes unstable is the division by Z, when projecting the viewpoint. This division may project the viewpoint to its very distant epipole. Our formulation avoids this projection when computing the cache bins for a line segment by calculating the dihedral angle ϕ. ϕ is the angle between the two planes Π_P and Π_{up} which are defined by their normal vectors.

$$\phi(\Pi_P, \Pi_{up}) = acos(\frac{\Pi_P}{\|\Pi_P\|} \cdot \frac{\Pi_{up}}{\|\Pi_{up}\|}) \tag{3.1}$$

with

$$\Pi_P = (P - C_{desired}) \times (C_i - C_{desired})$$
$$\Pi_{up} = (0,0,0) - C_{desired}$$

and P being the 3D-projection of one of the endpoints of the line segment. The depth value, which is required for the projection, can be chosen arbitrarily, since it has no impact on the resulting angle. $C_{desired}$ is the desired viewpoint, C_i is the camera center of image i. Figure 3.7 illustrates the calculation. Because the whole computation is performed in 3D object space, no singularity problems can occur.

3.3.2 Ray-silhouette intersection

Most prior approaches compute the silhouette intervals $V_{x,y}$ of each ray according to:

$$V_{x,y} = \bigcap_{i \in I} \Omega_i \qquad with \qquad \Omega_i = \bigcup_{j \in J_i} [start_{i,j}, end_{i,j}] \tag{3.2}$$

Such approaches first enumerate all silhouette intersections J_i for image i along the viewing ray at pixel x,y and combine them in a set Ω_i. This set in turn is intersected over all camera images. As a result, only intervals which are identified to contain the object by all images remain. Unfortunately, when one or more cameras do not contain the whole object in their frustum, the partially invisible parts are removed. Since one requirement of our application setup is a small footprint, cameras often do not see the whole object because they are too close. Due to the lack of high quality wide angle lenses, we had to solve this problem algorithmically.

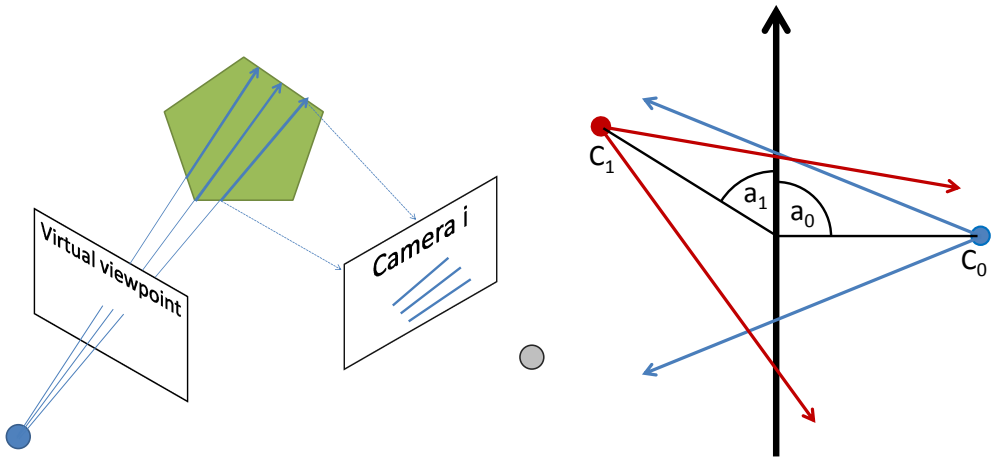
In the work of Boyer et al. [16], a dual formulation of equation 3.2 is used to accommodate for visibility problems during reconstruction:

$$V_{x,y} = \bigcup_{i \in I} D_i \setminus \Omega_i \tag{3.3}$$

D_i denotes the viewing frustum for camera i. This formulation computes the union of all intervals that describe the *outside* of the silhouette. This effectively prevents partially visible regions from being removed. Computing the result for each viewing ray can be performed particularly fast because no intermediate results need to be stored. Instead, inverted intervals can be added directly to the result list for each image. As a consequence, we need less memory per multiprocessor. We use the GPU's shared memory for storage of intervals because it is the fastest indexable memory available on CUDA-programmable GPUs. While it provides very high bandwidth, excessive shared memory usage decreases performance considerably when the GPU scheduler is no longer capable of switching execution blocks. Therefore, algorithms which use less memory are generally faster.

Another key performance factor for IBVH rendering is to determine quickly which viewing rays do not hit the foreground object. When interval intersection is based on equation 3.2, this is simple. The evaluation can be aborted as soon as the interval set has become empty. For our approach, which is based on equation 3.3, this is less trivial: the interval set has to be checked if it covers the whole ray, i.e., the whole ray is covered with empty space. To achieve this, we merge intervals during the computation. After computing $D_i \setminus \Omega_i$ for image i, we do not directly append the obtained intervals to the result list, but first check for overlapping intervals of previous images. If intervals overlap, we extend the existing interval to span over both intervals. Then, all intervals are again checked for overlap and merged in direction of the beginning of the list if possible. As a result, the first interval of each ray always contains the whole interval when a ray is empty. This condition can be quickly checked after each image is processed. Moreover, merged intervals require less shared memory than complete lists, thus resulting in an even greater speed up. During our tests, this early-termination method increased the speed of this stage to about six times as fast, which is roughly a 3.5 times overall speed up.

In implementations that allow for objects which are invisible to some cameras, phantom artifacts can occur. These artifacts show reconstructed geometry in regions where there is no foreground object. This happens when a camera's frustum is not sufficiently covered by other cameras. We implemented a counting algorithm to solve this problem. This algorithm eliminates intervals in $V_{x,y}$ which are not contained in a certain number of camera frustums D_i. The threshold should be selected according to the camera setup as the minimum number of cameras that see a point inside the intended bounding box.



(a) (b)

Figure 3.8: (a) visibility map computation based on rasterization of lines. Ray segments of the virtual viewpoint are projected onto the image plane of each camera i and rasterized with interpolated depth values of the ray start and endpoints. (b) illustrates the camera angle a_i. Two cameras (red and blue) with their frustums and a viewing ray (black arrow) are illustrated. Camera 1 (red) covers a larger interval on the viewing ray and therefore can carve more empty space than camera 0. Camera 1 is therefore more useful for skipping background pixels. The angle a_i can be used to sort cameras accordingly.

3.3.3 Visibility map computation

The epipolar search [156] is bound to be quite slow because for every pixel in the desired view, for each camera image the epipolar lines have to be traversed and compared with the ray intervals. The shadow-mapping visibility test can be implemented more efficiently on current GPU hardware: the depth map of the previous step is projected onto all camera image planes which results in a depth buffer for each of the cameras. However, a depth map is not a full description of the foreground object, and therefore produces visibility artifacts. To remove most of them, we again exploit the IBVH rendering algorithm. When calculating ray-silhouette intersections, the backmost intersection can be found at no additional cost. Each pair of front and back intersections describe a 3D ray which is located inside the object. Of course, this is just an approximation: more correct results can be achieved by using all intervals along the rays, but this also induces more overhead.

After projecting the front and back point of a 3D ray onto a camera's image plane, we rasterize the line and write interpolated depth values into the visibility map using the atomic minimum operation of CUDA. See Figure 3.8(a) for an illustration. Low sampling densities can occur, for example, when the desired viewpoint is far away from the object. We counter this by using a visibility map resolution which is smaller than the scene's projected footprint from what we assume to be the maximum distance to the object.

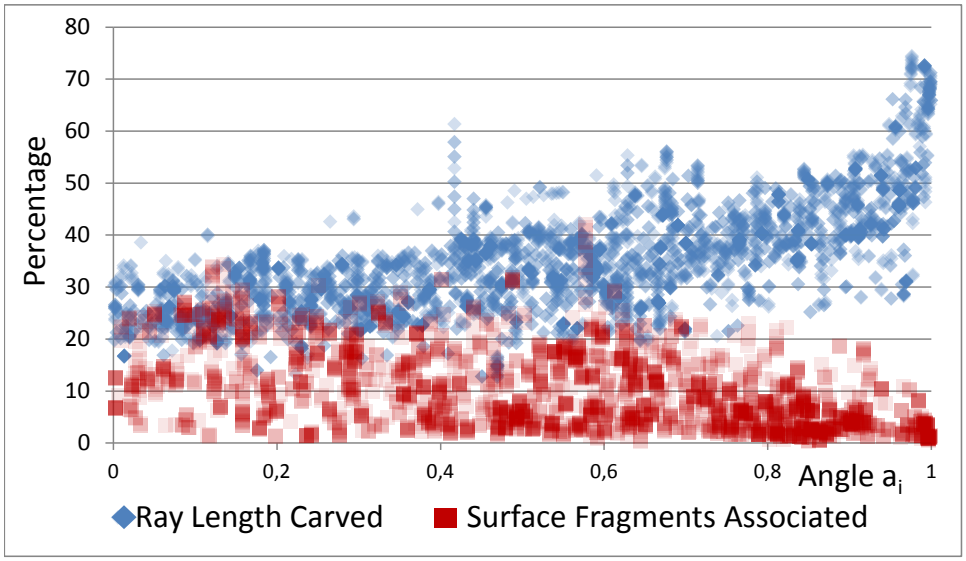


Figure 3.9: The plot shows statistical data that was captured over a user motion sequence with a rotating viewpoint. The x-axis denotes the angle a_i between a camera's optical axis and the viewing direction. The y-axis indicates the average percentages that are carved away from the viewing rays and the relative number of surface point associations.

3.3.4 Camera order

The IBVH algorithm computes every pixel's depth value by iterating over all silhouette images. For each image the viewing ray is intersected with the silhouette and resulting intervals are stored in a list. Finally, these intervals are intersected to find the frontmost ray-object intersection - the surface. Previous IBVH implementations process the cameras in an arbitrary order because the result is independent of it. However, the execution time is not.

To detect background pixels quickly, the interval finding and intersection steps should be interleaved. Once the intersection operation results in an empty set, processing can be stopped and the pixel can be marked as background. As was shown in this chapter, this early termination strategy can speed up the algorithm considerably. To improve the performance even further, the order in which cameras are processed should facilitate early termination.

When cameras that cover large intervals of a viewing ray are processed first then empty intersection sets are found earlier. We found that the optical axes of the cameras can be used to sort them. Figure 3.8(b) illustrates how the optical axis of a camera relates to the size of the interval it covers: cameras with steep angles relative to the current viewing direction tend to cover larger intervals. Therefore, to quickly approximate the percentage of average ray coverage of a camera i, we use the angle a_i.

$$a_i = abs(dot(axis_i, axis_{view})) \tag{3.4}$$

It is the angle between the normalized camera's optical axis $axis_i$ and the normalized current viewing direction $axis_{view}$. An *arccos* is not required because we use the value only for sorting. The absolute value of the dot product computes the angle of the shortest arc between the axes.

Figure 3.9 shows how the angle a_i relates to the relative number of surface points that are associated to a camera i. From the measurements we can conclude that most surface points are associated to cameras with low to medium angles. From this we can derive that cameras with low a_i are more likely to produce a surface fragment. Conversely, the most space along the viewing rays is carved away by cameras with relatively high a_i. This means that cameras with high a_i's are likely to be more powerful at identifying background pixels.

All our methods therefore sort the cameras according to their a_i in descending order before rendering. The average performance improvement over several tested camera orders reached up to 12%, depending on the amount of background surrounding the visual hull.

3.4 Results

We analysed the IBVH pipeline in terms of memory transfers and processing [89, 93] to find out how it scales with different input and output resolutions and numbers of cameras.

3.4.1 Data flow

Before the first stage begins, the camera images are read from the driver and uploaded to the GPU. There, the images are segmented and passed to the angle binning kernels. The amount of data that is passed is proportional to the camera resolution and camera count. After angle binning, the bins are passed to the intersection kernel. The size and number of the angle bins must be chosen such that the overall capacity (size multiplied by number of bins) is sufficient to hold all line segments of a camera. Choosing many small bins is beneficial for performance, while few large bins require less memory. Usually, this part of the pipeline creates the largest data flow, which makes it a bad spot for sharing data between GPUs. After the IBVH kernel, the desired depth map is computed and passed to a shading kernel. This final stage and the associated data flow has only minor impact on the overall performance and therefore marks the end of the pipeline that is analyzed here. Figure 3.10 shows the amount of data that flows through our pipeline.

3.4.2 Performance of the stages

Figure 3.11(a) illustrates the execution times of all stages. The system was configured to use ten cameras at a 640×480 resolution and 15 Hz. The output resolution was 1600×900 with about 1550×750 pixels covered with foreground. It can be observed that the main IBVH computation (the ray-silhouette intersection) is the most time consuming step.

Figure 3.11(b) compares IBVH rendering and voxel carving and rendering. Execution times are given for three different resolutions. The voxel grid resolution is set such that every visible voxel projects to one pixel to make the two approaches comparable. It can be observed that voxel carving and raycasting can only keep up with IBVH rendering at

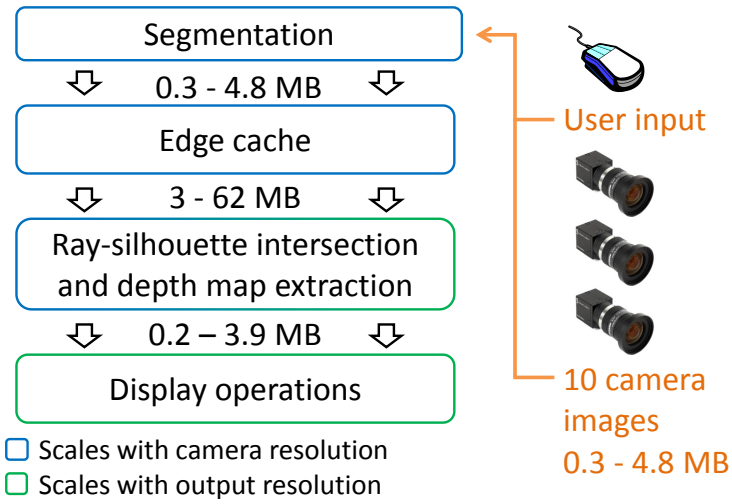


Figure 3.10: A typical image-based visual hull rendering pipeline. The first part consists of segmentation and undistortion. The second part builds a cache data structure to speed up the third part, the ray-silhouette interval extraction and interval intersection. The last part is responsible for texturing and display. The data traffic between stages is indicated for a single rendering pass. The value ranges cover camera resolutions from 320×240 to 1280×960 and a maximum output resolution of 2 Megapixels.

very small resolutions. The voxel resolutions were $700 \times 350 \times 1400$, $450 \times 225 \times 900$ and $250 \times 125 \times 500$ for the three test runs.

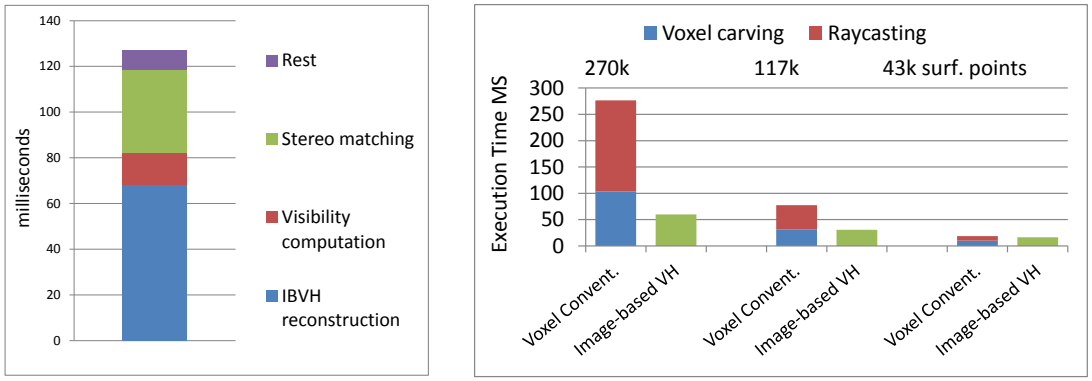
3.4.3 Scaling with inputs

First, the segmentation and edge cache generation extract information from the camera images. These tasks are defined per camera image, which means the runtime is proportional to the number of cameras and their resolution. Next, the IBVH core computes a depth map from the cached edges. This stage scales with the output resolution, and, to a lesser extent, with the number of edges. The number of edges is driven by the number of cameras and their resolutions. The final display step scales only with the output resolution. Figure 3.12 shows the runtime of the stages for different resolutions and number of cameras.

From this data we can derive that for increasing display resolutions, the computation of the depth map (the core IBVH step) becomes the dominant performance factor. This part also scales with the number of cameras. The resolution of the camera images on the other hand does not have a strong impact. This means that optimizing the IBVH method and workload is more important than the other stages.

3.4.4 Improved camera order

For this test we compared the IBVH pipeline with and without improved camera order. Figure 3.13 shows the results for a recorded sequence of various user motions. Averages



(a) (b)

Figure 3.11: (a) shows the kernel execution times of the IBVH pipeline's stages on a single GPU at an output resolution of 1600×900. (b) gives a comparison of IBVH rendering and voxel carving and rendering at different resolutions. The number of surface points equals the number of foreground pixels, where the voxel resolution is set such that each voxel projects to the area of one pixel.

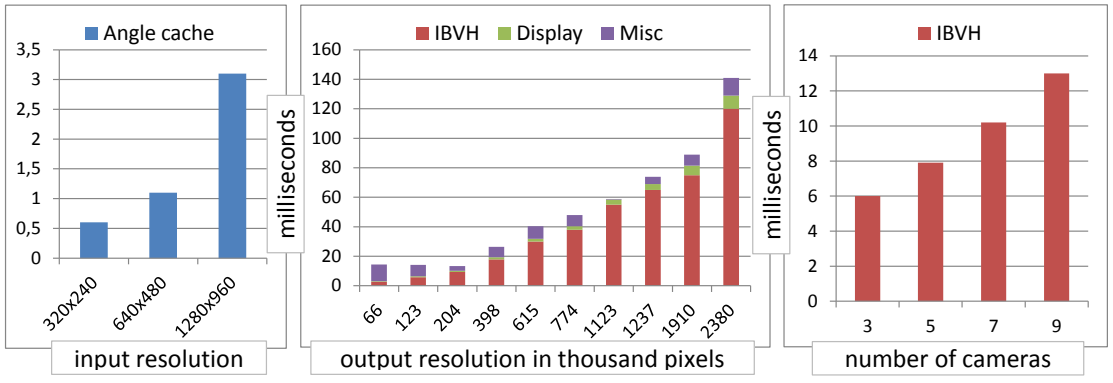


Figure 3.12: Kernel execution times on a single GPU with different numbers of cameras and different input (=camera) and output (=screen) resolutions. Angle caching scales with the camera resolution, while the other kernels scale with the output resolution. The IBVH algorithm also scales with the number of cameras. Unless specified otherwise, the timings are given for 10 cameras, an input resolution of 640×480 pixels and an output resolution of 480×870 pixels.

are built over the whole sequence, over rather static frames with slow user motions and over dynamic frames with much user motion. The improved camera sorting by angle a_i reduces execution times by around 7%.

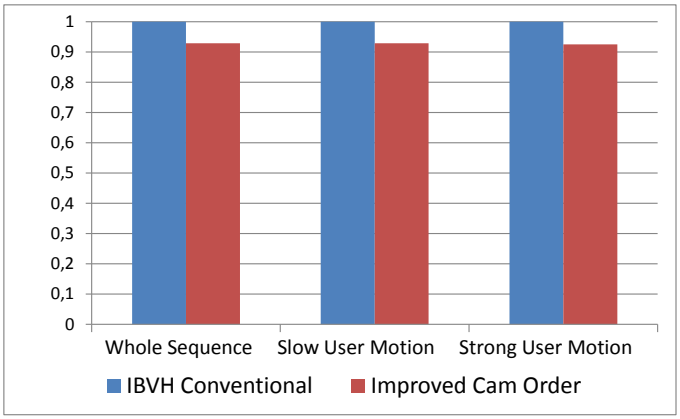


Figure 3.13: Evaluation of the suggested improved camera order.

3.5 Summary

In this chapter we introduced our multi-camera hardware setup. Next, we have shown how a full IBVH pipeline with visibility, texturing and stereo matching can be implemented. We introduced several extensions to the original approach to achieve better performance for GPU implementations and robustness of the angular formulation. Furthermore, we introduced the angle between a camera's optical axis and the viewing direction as a reliable statistic for skipping empty space and thus unnecessary computations.

We analyzed all stages of the pipeline in terms of processing time and the amount of data flow between them. The measurements showed that the resulting pipeline is much faster than the popular voxel carving method, which verifies the first part of our image-based rendering hypothesis. The evaluation also indicates that the IBVH pipeline scales better with resolution than voxel carving. The result of such a hard- and software arrangement is an interactive device that displays the image of a user from arbitrary viewpoints.

Chapter 4

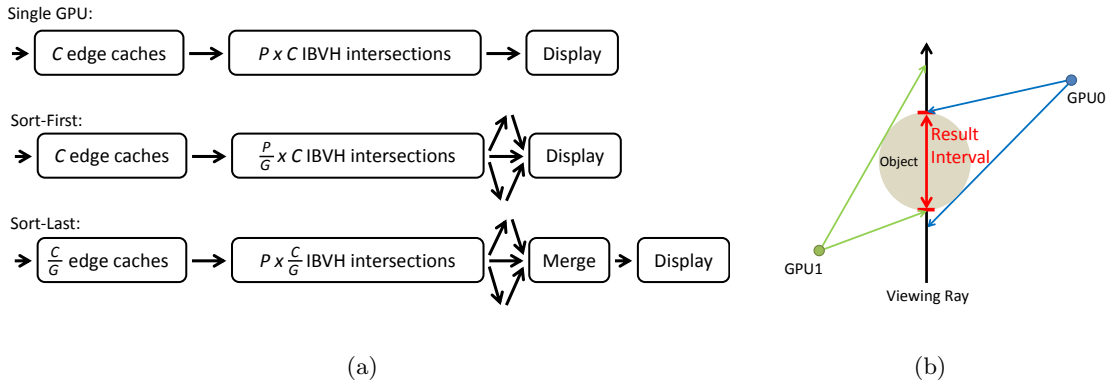
Multi-GPU IBVH rendering

Contents

In our free viewpoint rendering system, a set of cameras is mounted around the area where the user is allowed to move. The cameras provide a color video stream. As we described above, the image-based visual hull (IBVH) algorithm can be applied for the purpose of rendering the user from an arbitrary viewpoint. It performs a depth map reconstruction directly from the silhouette images [90]. Similar to raycasting it is a ray-based algorithm: for every pixel a viewing ray is computed. It therefore scales badly with an increasing output resolution. As the user immediately sees the rendering of himself, high computational performance and low latency are crucial for a satisfactory user experience. This is also important to avoid simulator sickness during extended sessions. Moreover, user-interaction through gestures benefits from low latency.

The high performance to cost ratio of a single PC equipped with multiple graphics processing units (GPUs) makes it an attractive platform for such complex interactive visualization tasks. Moreover, data transfers from main memory to GPUs are fast compared to network transfers. Several visual hull algorithms have been modified in order to run on several PCs or GPUs in parallel. These algorithms usually involve explicit, view-independent intermediate results in the form of meshes or voxel grids that can be computed independently before merging them in a final step. In contrast, IBVH rendering does not have such a representation, and is therefore not parallelizable with standard methods.

In the previous chapter we analyzed the computation times and data flow of an existing single-GPU IBVH pipeline. From these evaluations we derive which stages are suitable for parallelization. In this section we suggest three ways of distributing the workload of an IBVH pipeline over several GPUs. We start with a sort-first approach, which is simple but has drawbacks. Then, we introduce a sort-last approach by regarding cameras as scene objects. In addition, we suggest a compact buffer representation that reduces the bus



(a) (b)

Figure 4.1: (a) different configurations of a synchronized multi-GPU IBVH pipeline and a single-GPU pipeline as a reference. C denotes the number of cameras, P the number of output pixels, G the number of GPUs. Sort-last configuration (b): a viewing ray intersects silhouette cones from two GPUs. The intervals are intersected again to compute the result interval.

traffic. The last approach is a multi-frame rate setup that decouples the viewing process from the image generation to achieve high frame rates even for large resolutions. To improve visual quality, we suggest a combined forward- and backward warping method. We evaluated all approaches by measuring and comparing execution times for different configurations [88, 89].

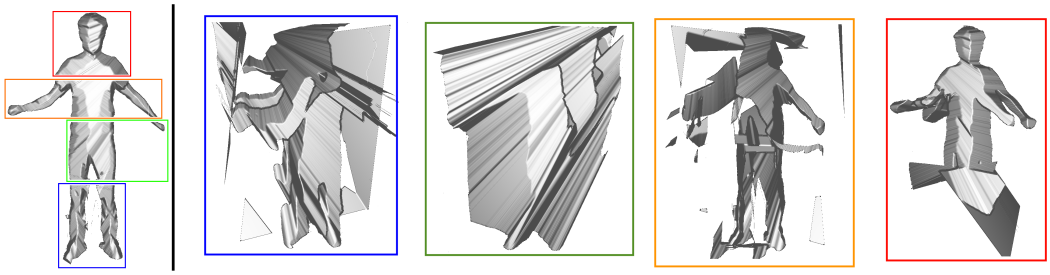
4.1 Parallelizing the image-based visual hull algorithm

We suggest several multi-GPU configurations of our pipeline. The main difference between them is the number and placement of synchronization points. At a synchronization point, the GPUs wait until the current stage of the pipeline has been completed on all GPUs. Afterwards, data is shared between the GPUs to allow later stages to have access to all intermediate results. Figure 4.1 (a) shows these synchronization points and the amount of work that needs to be performed by each stage.

The core part of the pipeline is the IBVH stage that produces a depth map from a set of angle bins. It requires most of the computation time. All parallelization configurations therefore focus on how to split and distribute this stage. The segmentation, angle binning and shading step are the same for all configurations.

4.1.1 Sort-first configuration

The first configuration uses a sort-first arrangement. The IBVH algorithm is defined as a per-pixel operation, which makes it very similar to raycasting in terms of how independent the computations are from each other. The workload can be easily split between the GPUs by dividing the output image into equally sized boxes. The IBVH stage can run in parallel. After IBVH computation, the synchronization point is reached and data is shared. The



(a) (b)

Figure 4.2: Sort-first (a) and sort-last (b) configuration intermediate results of four GPUs rendered with phong shading.

final display stage joins the subimages to form the output image. See Figure 4.2 (a) for an illustration of the workload distribution.

Sort-first parallelization is easy to implement, but suffers from the memory transfer that is required to pass the input data to the computing nodes. In our case, this means that all camera images have to be uploaded on all GPUs. Moreover, to achieve maximum scalability, the exact location where the screen is split needs to be determined by a robust load balancing mechanism [160].

4.1.2 Sort-last configuration

Sort-last approaches usually distribute the scene objects (models, or triangles, or bricks of volumetric data) across the computing nodes. Nodes render their chunk of data in parallel, and send the result image plus depth information to one or more compositing nodes. Compositing is more complex than the merging step of sort-first approaches: it involves testing or sorting all fragments according to their depth.

Visual hull rendering is usually focused on a single scene object: a person in our case. Even systems that can capture multiple objects in front of their cameras usually can not distinguish between the objects before actually rendering them. To distribute workload across multiple nodes, we therefore suggest to assign a subset of the cameras to each node (=GPU).

For example, when two GPUs are used, each can process five camera images to achieve a total of ten. Each GPU computes the visual hull of only a subset of all available cameras (see Figure 4.1 (b)). However, unlike conventional rendering, such a subset is not a stand-alone rendered image with depth. Instead, it is only an intermediate state in the sequence of ray-silhouette intersections that make up the IBVH algorithm. This state consists of a list of intervals along each viewing ray that must not be collapsed to a single depth value until all subsets are intersected. See Figure 4.2 (b) for an illustration of the intermediate results.

In our system we have ten cameras, which means that each GPU has to handle a maximum of five when multi-GPU computing is desired. For the application of rendering

humans, we found that two intervals along each viewing ray are sufficient to capture the geometry of up to five cameras. Each interval can be described by two float values that denote the boundaries of the interval along the viewing ray. This means that each GPU produces a buffer that has the resolution of the output image and four float values per pixel.

The suggested method corresponds to a sort-last approach, where cameras are regarded as scene objects. In contrast to conventional sort-last approaches, the compositing step is more involved than testing or sorting depth values. The intervals at each pixel need to be intersected to produce the final visual hull. After intersecting, the first interval that describes the object surface can be used to produce the output depth value.

Compact interval representation Sort-last approaches can be optimized by only transferring image parts that contain an object [160]. This is called *sparse* sort-last and usually achieved by tight fitting bounding boxes. Unfortunately, for sort-last IBVH all pixels within the viewing volume produce data. As a result, the data traffic after rendering is considerable and prevents the algorithm from scaling well with the output resolution and the number of GPUs.

The interval data that are transfered between the GPUs correspond to depth values generated from a subset of the cameras. While depth buffer compression techniques are often tuned for triangle data [86], the interval data for IBVH rendering shows a different structure, as depicted in Figure 4.2. Nevertheless, more general depth buffer compression algorithms can also be used for the ray-like depth structure found in the interval data buffers.

To efficiently compress and uncompress the data without hardware supported compression, we use a method similar to Orenstein et al. as described in [86]. We divide the interval data into blocks of $8x8$ pixels. For each block we pick a representative pixel and distribute its interval values across the entire block. Every other pixel within the block computes the difference of its values to the representative values. The differences are stored with reduced bit length into a global buffer.

To tune the compression ratio, we support different bit lengths per value. To decide which bit length to use, all pixels in the block publish their required bit length and the maximum is taken. With lossless compression, data traffic rates can be reduced by approximately 33% to 45% depending on the resolution. But there is no need for full precision as long as there is no perceivable difference in the results. Therefore, we drop the four least significant bits, reducing the bus load by 60% to 70%.

To handle the entire memory transfer with a single transaction, we pack the encoded data compactly in memory. We use a single atomically modified counter which is increased by the required amount of memory for each block. In this way, every block retrieves its individual offset in memory placing blocks with different compression rates next to each other. The block offset is stored together with information about the used bit length in a header structure. Blocks are decompressed independently of each other. Each block reads its offset and bit length from the header. Then, the representative values and differences are read and the decompressed values can be generated.

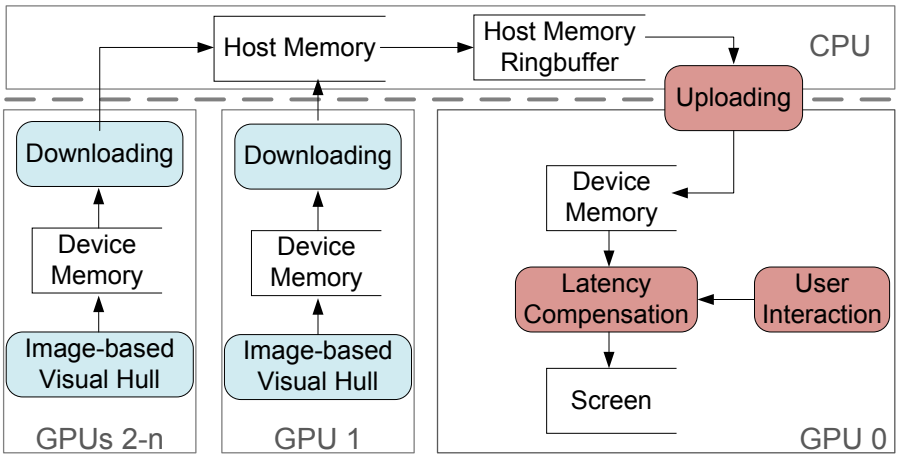


Figure 4.3: Multi-frame rate configuration that uses GPU0 for viewing and the other GPUs for parallel image-based visual hull rendering.

4.1.3 Multi-frame rate configuration

Multi-frame rate (MFR) rendering decouples viewing from image generation and typically uses separate computing nodes or GPUs for each of the tasks. Viewing means linear camera or object transformation, as it is common in many applications. The image generation method can be arbitrary, as long as a depth value can be computed for each pixel. Generated images are transferred to the viewing node, but the viewing node does not wait for images to become available. Instead, it uses the last known image for viewing. In general there is a slight difference in the desired viewing transformation and the one that was used to generate the last image. This difference can be covered by image-warping. Hardware-accelerated image-warping and the asynchronous communication behavior guarantees very high frame rates at the viewing node. The advantage over synchronized parallel rendering grows with increasing scene complexity and output resolution. See Figure 4.3 for a diagram of the architecture.

IBVH rendering is a complex algorithm and therefore benefits from MFR rendering. However, the visual hull transforms non-rigidly every time a new set of camera images becomes available. Therefore, the high viewing performance of MFR rendering can only be exploited between camera image updates. The frame rate of the image generation node(s) must be higher than the update rate of the cameras. In practice we observed that this is not a limitation: the camera update rates are usually not as high as the desired viewing frame rates due to limited bus and network bandwidths and latencies that slow down camera image transfer. See Section 4.3 for a performance analysis.

Two-pass image-warping For image-warping we use a combination of forward and backward image-warping. Forward image-warping projects pixels from a source image to the current view. This is a fast operation, but suffers from holes in the result. Backward image-warping on the other hand does not produce holes, but involves a search in the original image to find the correct pixels and is therefore slow [184].

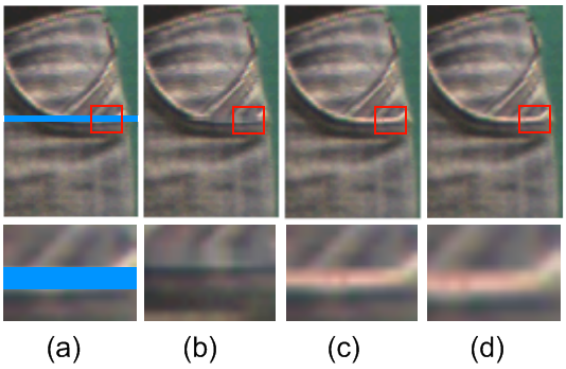


Figure 4.4: The effect of two-pass image warping: holes after forward image-warping (a, blue) can be filled by interpolating neighboring warp vectors as in (c) instead of interpolating color values as in (b). (d) shows the ground truth. The red boxes are magnified below.

Our two pass warping approach combines the advantages of both approaches. First, a forward warping step is used to project the last frame's pixel coordinates according to the current view. The holes in this buffer which arise from forward warping are closed using linear interpolation on the pixel coordinates (in contrast to color values in traditional approaches). Now, this buffer forms a lookup table into the last frame with sub-pixel accuracy. In a final step, this lookup is performed to compute the new color values, resulting in a backward warping approach. Figure 4.4 illustrates the differences between color value interpolation and our approach. Our approach does not blur the image. Implementation details are described in Section 4.2.

Combined synchronous and asynchronous rendering Note that large holes that come from disocclusions can not be filled in such an efficient way. Data that is not present can not be interpolated. Here we rely on rendering performance: disocclusions are less likely with a quick image source.

To achieve the required performance, we want to use multi-GPU rendering as the image source for multi-frame rate rendering. Our system allows to combine multi-GPU (synchronous) and multi-frame rate (asynchronous) IBVH rendering. For example, the image source can be a sort-first or sort-last IBVH renderer that uses all but one GPUs. The one remaining GPU is used for image-warping and display. See Figure 4.3 for an illustration of such a configuration and Section 4.3 for a performance evaluation.

4.2 Implementation

The whole pipeline is implemented in Nvidia CUDA [90] and uses streams to distribute command sequences to the GPUs. All memory copies are issued asynchronously, which makes the system more robust to differing workloads: when a GPU has finished processing it can already send the result and thus help to avoid later bus collisions. Memory is transferred between GPUs through host memory.

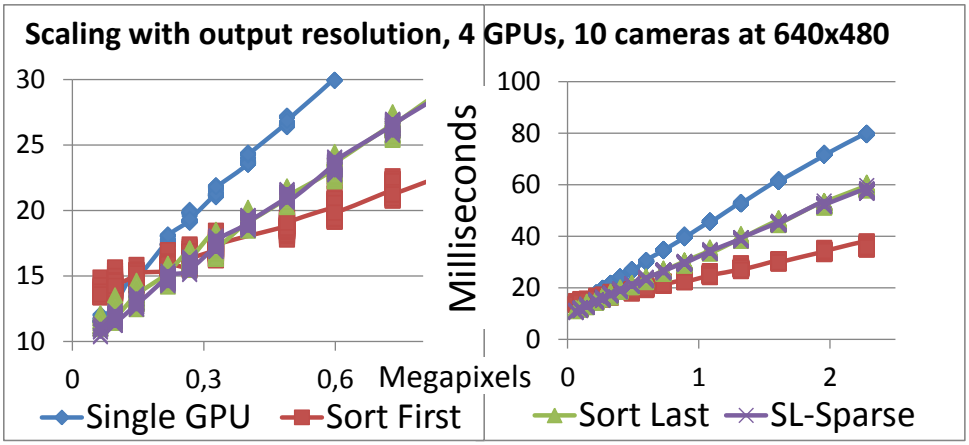


Figure 4.5: This performance evaluation of our pipeline shows how the suggested approaches scale with respect to output resolution. The left figure is a magnification of the first Megapixel range.

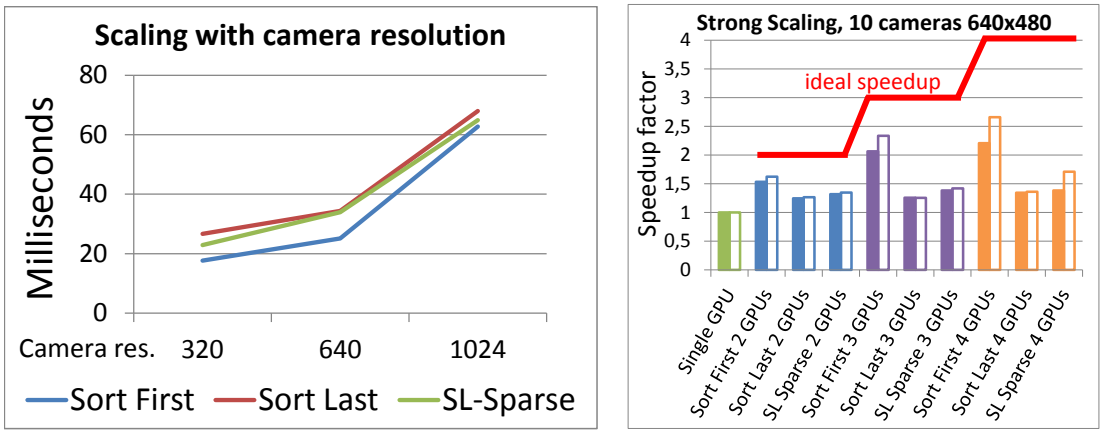
All nodes are provided with the camera images (4 bytes per pixel) that they need, state information in the form of a projection matrix (4x4 float) and bounding box information to avoid unnecessary data traffic. Pixels of depth image segments are either compressed to two bytes of precision (half float), or encoded per block for sparse sort-last with the compact interval representation (see Section 4.1.2).

In the case of multi-frame rate rendering, the memory transfers are double-buffered in host memory to facilitate asynchronous communication without stalls [88]. For image-warping the rendered images have the projection matrix attached that was used to create them.

The buffer that is used for two-pass image warping is a screen-sized array of unsigned integers. Screen coordinates are stored in the buffer by packing them with their depth value into a single unsigned integer. The depth values occupy the ten most significant bits, the screen coordinates share the rest. We can achieve efficient depth-buffering when such data packets are written to the buffer by using atomic minimum operations. For hole filling, the neighboring data packets are decoded, screen coordinates are averaged and stored at the hole pixel.

4.3 Results

In this section, the configurations suggested above are evaluated for their performance. The IBVH kernel is responsible for most of the computation time and is therefore the main focus of our parallelization methods and the corresponding evaluations. As stated in the beginning, the performance of this stage scales with the number of cameras and their resolution, the display resolution and the number of GPUs. For our evaluations we assume a fixed number of cameras of ten. While our system works with any number of cameras, this specific number proved to work best for the application of rendering users with high quality. We do, however, evaluate the system for different camera resolutions, display



(a) (b)

Figure 4.6: (a) shows the scaling with camera resolution for a fixed output resolution of 1 Megapixel. (b) illustrates how additional GPUs influence processing times at an output resolution of 2.3 Megapixels (filled bars) and 7.36 Megapixels (outlined bars).

resolutions and number of GPUs to find out how the suggested methods scale. The used camera images are read from the hard-disk for repeatability. Note that we excluded the hard-disk access times from the evaluation because in the live system the camera images are also not loaded from the disk. We therefore place the images in the host memory and upload them to the GPUs every frame, just like the live system would.

Scaling with display resolution In the first test series we rendered a representative IBVH scene (see Figure 4.2) from a viewpoint that is close enough to have the object fill the screen. We averaged the rendering times of several frames at varying display resolutions. Figure 4.5 shows performance measurements for four GPUs and a single-GPU as a reference. The multi-GPU approaches outperform the single-GPU approach especially for larger output resolutions.

Scaling with camera resolution For this test we rendered three scenes with different camera resolutions. The scenes consist of the same person standing in similar poses. All measurements are again averaged over multiple frames. See Figure 4.6(a) for results. While the resolutions quadruple every step, the execution times maximally double. This is promising for higher camera resolutions in the future.

Scaling with number of GPUs: strong scaling In this test series we use again a representative scene (see Figure 4.2) and render with one, two, three and four GPUs at a resolution of 2.3 and 7.36 Megapixels. Performance measurements are given as speedup factors. Speedup factors compare the performance of n GPUs to the performance of one GPU. A factor of n is considered ideal and is illustrated as a red line. All measurements

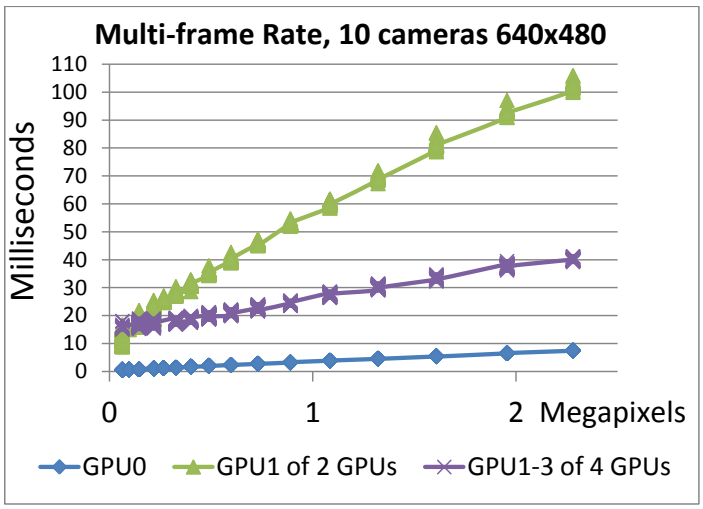


Figure 4.7: Multi-frame rate performance for a setup with two and four GPUs. GPU0 is always used for viewing. Its performance is comparable for both configurations. The quad-GPU setup uses three GPUs for sort-first parallel IBVH rendering.

are again averaged over multiple frames. Figure 4.6(b) illustrates how the performance benefit of adding an additional GPU declines with the total number of GPUs.

Multi-frame rate scaling The evaluation data of the multi-frame rate approach is shown separately in Figure 4.7. We used the scene from above and measured performance for two and four GPUs. The dual-GPU setup uses a viewing GPU (GPU0) and a single GPU for IBVH rendering (GPU1). The quad-GPU setup also uses GPU0 for viewing, but three GPUs for sort-first parallel IBVH rendering.

The GPUs that are responsible for image generation show a similar performance to a stand alone setup. The performance goal for them is to stay below the camera update rate in order to avoid missing any image. The camera update rate in our system usually is 15 Hz. The quad-GPU setup stays well below this mark.

The performance of the viewing GPU outperforms all other approaches easily because viewing is a less complex task than IBVH rendering by far. In combination with an image-generation backend that does not miss any camera image, this is a very powerful method for interactive IBVH rendering.

Interpretation and discussion In terms of performance we observed a behavior that is common to sort-first and sort-last parallel rendering. Sort-first nodes need to receive all the input data, whereas sort-last nodes need to send all the output data. Therefore, when the data transfer to the nodes - in our case the camera images - is dominant, then sort-last with compression is a good option. Figure 4.5 left shows such a setup. When traffic is output-heavy due to a relatively high output resolution then sort-first performs best. Figure 4.5 right illustrates this.

The performance measurements in Figure 4.6(b) reveal that the scaling of the pipeline with additional GPUs is far from optimal. To find the reason for this, we analyzed all stages of the pipeline. The IBVH kernel itself scales almost ideally with an increasing number of GPUs. However, the memory transfer times between the GPUs increase stronger. At 2 Megapixels output resolution and four GPUs, the sort-last approach uses 28% of the time for transferring output fragments, and we even subtracted the transfers that overlap with execution from this number. Only about 58% of the time was spent with actual processing. The sort-first approach has better scaling characteristics. For the same configuration, only 4.5% were fragment transfers and around 68% was processing. To increase the processing time relative to the overall execution time, we also evaluated the pipeline at 7.36 Megapixels output resolution. At this very high resolution the sort-first approach spent around 63% of the time with processing and 5.5% with fragment transfer. From this data we derive that it is mostly the memory traffic and the overhead computations (kernel launches, memory initializations) that prevent the presented pipeline from scaling well beyond two or three GPUs.

The multi-frame rate setup can utilize a sort-first or sort-last IBVH renderer as its image source. We found this configuration to be particularly useful because it provides over 100 frames per second on the viewing GPU even for output resolutions of 2 Megapixels. At the same time, a dual or triple-GPU image source can provide enough computation power to process every camera image. At these frame rates, the visual quality of image warping is high because disocclusion artifacts can hardly be observed.

4.4 Summary

In this chapter we introduced three methods to parallelize the image-based visual hull algorithm on multiple GPUs. First, we analyzed the pipeline for possible parallel execution. We identified two methods, following the common sorting-classification: sort-first and a sort-last. For sort-last we suggested to regard the cameras as scene objects, and introduced how the compositing step needs to be adapted. In addition, we suggested a block-based packing scheme that reduces memory traffic drastically. Finally, we enhanced the system by multi-frame rate rendering to achieve even higher frame rates for viewing applications. We introduced two-pass warping to achieve better hole-filling. We evaluated the performance of all approaches and were able to verify that a triple or quad-GPU multi-frame rate setup can achieve very high interactivity without sacrificing the visual quality.

As a result of the suggested parallelization techniques, our interactive free viewpoint capturing and rendering system can provide higher output resolutions. We achieve interactive frame rates even when computing two million pixels, which is enough to display every detail that our cameras are able to capture. This improvement in rendering efficiency supports our image-based rendering hypothesis.

Chapter 5

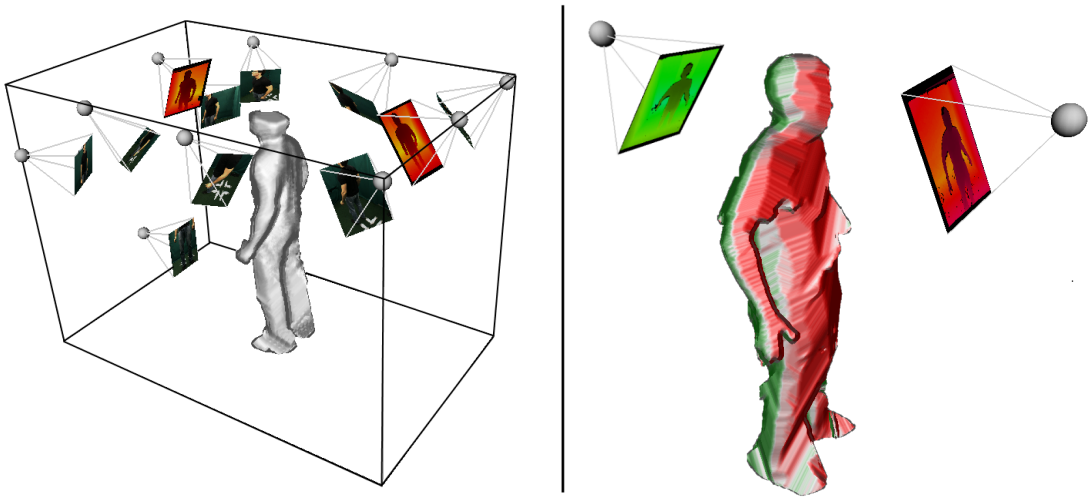
Combining color and depth sensors

Contents

The Microsoft Kinect device has profoundly changed the possibilities of sensing for games or virtual reality applications. Previously, depth sensing hardware was expensive, so computer vision applications were primarily based on affordable digital video cameras. However, video cameras do not directly provide depth information. Depth can be computed by utilizing two or more cameras and 3D reconstruction methods such as the IBVH algorithm. These methods are computationally expensive and therefore limited in terms of reconstruction quality and resolution.

With a Kinect device, direct depth sensing and video capture immediately deliver rich information on the scene structure at a competitive price without intensive processing. Not surprisingly, many researchers have taken advantage of this opportunity, and we see a proliferation of research projects that rely on this technology [108]. During our work on IBVH rendering, the devices became available and we decided to incorporate the new technology into our free viewpoint rendering pipeline to increase the reconstruction and display quality with little performance overhead.

In this chapter, we describe and compare two sensor arrangements that we installed and used for our experiments. The first is an extension of the multi-camera capturing room that was described in the previous chapters. It consists of two Microsoft Kinect sensors in addition to the ten color cameras. The other setup consists of up to seven Microsoft Kinects and is called OmniKinect. Building a setup with multiple active depth sensors is not necessarily trivial, as a number of conceptual and technical challenges in the system design must be overcome. Most importantly, multiple Kinects pointing at the same spot interfere. We describe solutions for these practical challenges, and also introduce rendering algorithms for each of the setups that leverage the different sensor configurations.



(a) (b)

Figure 5.1: The rendering in (a) shows how the multi-camera room was extended by two Microsoft Kinect devices, see the red-yellow color-coded depth maps. The right image (b) illustrates how most surface parts do not suffer from overlapping projector patterns. The bright regions on the shoulder and the head suffer from projector overlap and grazing angles and are therefore most likely not covered by a depth sensor.

5.1 Improving the visual hull with depth sensors

A single Kinect can deliver enough information to let a user control a character in a video game with body movements or resolve real time occlusion of video-see through Augmented Reality (AR) applications. However, a single sensor would limit the range of possible viewpoints in our application. We therefore need a number of sensors that capture the user from all sides.

A very common method to visualize the data from one or more depth sensors is point cloud rendering. It works by transforming every depth map pixel into a 3D point that can be projected onto the screen. With a sufficiently high depth map resolution this approach produces smooth rendered surfaces with high performance. However, it is very sensitive to the depth map quality.

Active depth sensors interfere with each other when the emitted signals are caught by the wrong device. As a result, the reconstructed depth maps have holes or outliers. This conceptual disadvantage also applies to Microsoft Kinect devices. These conflicts can be resolved by time-slicing the signals or shifting the signals to different wave lengths. However, such solutions are usually only available for very expensive devices. We alleviated this problem by extending our multi-camera setup with only two depth sensors mounted at opposite sides of the room. In this arrangement, the projected patterns on the user's body hardly overlap because the user is located between the sensors. Figure 5.1(b) illustrates the

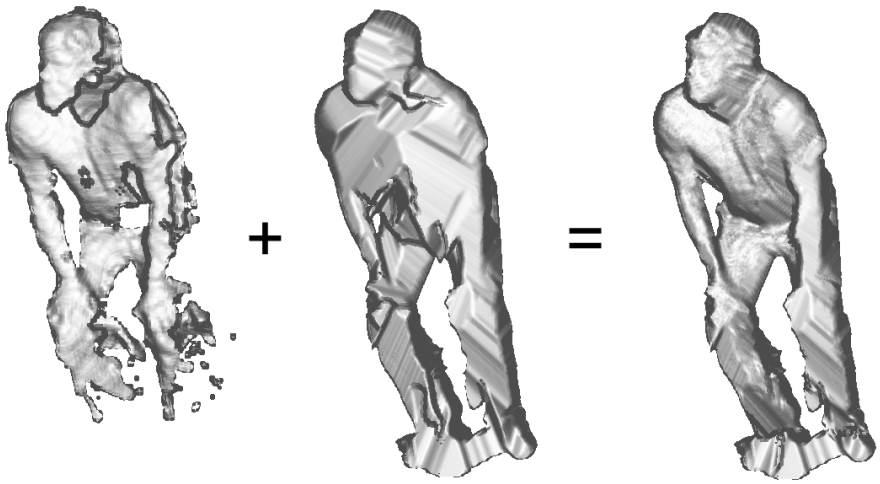


Figure 5.2: This figure illustrates the benefit of combining visual hull and depth map rendering. The left image shows a point cloud rendering of the two depth maps. The middle image shows the visual hull. The result on the right side shows the combined rendering.

concept. However, the two reconstructed surfaces are not dense and leave a gap between them.

Another problem is that active sensors that compute depth values or distances based on the disparity between a projected pattern and a camera, are subject to occlusion artifacts. These artifacts occur at depth discontinuities where the projector is blocked. Moreover, depth maps of Microsoft Kinect devices are often noisy, have holes and suffer from complex radial distortions. The distortions become apparent when more sensors are fused, see Figures 5.2 (left) and 5.4 for examples.

The suggested visual hull pipeline, on the other hand, is very robust. It produces images of a water-tight surface without holes. There is no perceivable distortion due to the high quality of our cameras and the calibration process. However, it suffers from artifacts that are inherent to shape-from-silhouette reconstruction, especially at concave surface regions (see Figure 5.2, middle image). Due to the real time constraint we preferred to avoid stereo matching to increase the visual quality. With additional depth data from active sensors, however, such improvements become possible.

5.1.1 Combined visual hull and depth map rendering

The visual hull of an object is a conservative surface estimation: the real surface is contained within the visual hull and touches it at some points. Therefore, only depth map points that are located inside the visual hull can be valid. We assume that these points are most likely a better surface estimate than the visual hull and should be preferred. Depth values outside the visual hull are outliers and can be removed. Holes in the depth map can be closed by visual hull patches.

The resulting algorithm is an extension to the IBVH rendering pipeline described above. In an additional stage, the visual hull is carved to fit the depth maps. First,

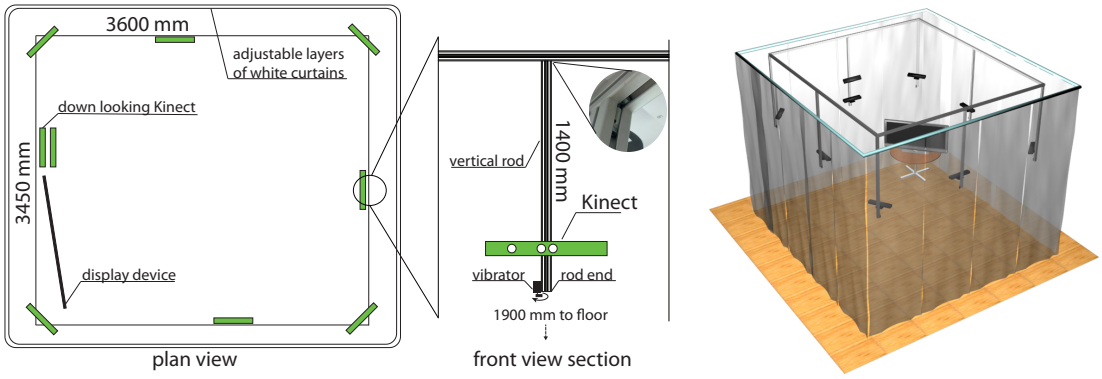


Figure 5.3: Plan views (a) and 3D overview (b) of our OmniKinect setup. In (a), possible mounting points for Microsoft Kinect devices are marked green.

the frontmost and backmost ray-visual hull intersection points are projected to the image planes of the depth sensors. There, the resulting epipolar lines are traversed to find the ray-depth map intersection. This point is used for the improved surface. If no such point is found, the visual hull is used. This way, holes in the sensor's depth map do not cause holes in the result, while valid points improve it. Figure 5.2 illustrates the advantage of the suggested method.

In our second sensor arrangement, the OmniKinect system, we developed the idea further and changed all the cameras in favor of Microsoft Kinect sensors. The next section describes how we built the new system.

5.2 The OmniKinect system

The OmniKinect system provides a way to capture, record and stream information using a multiple Kinect sensors infrastructure. However, multiple Kinect sensors produce overlapping infrared light patterns that interfere with each other and thus produce holes in the depth maps.

This problem has been solved by Maimone et al. [147] and Butler et al. [24] by letting the sensor vibrate at a relatively high frequency. This motion blurs the light pattern for other, concurrently capturing sensors. The Kinect device's own pattern does not blur because the projector and the sensor are connected rigidly. In OmniKinect, we altered this approach slightly to gain more flexibility.

Setup overview: Our basic setup consists of an extensible, ceiling mounted aluminum frame with rigidly fixed vertical rods at regular distances. We have attached Kinect for Windows devices to the rods with stiffened foot joints. To reduce interferences between the Kinects, the rods are equipped with vibrators. In contrast to previous work, we do not mount the vibrators directly onto the Kinects but on the supporting structure. This

has various advantages: First, we do not have to disassemble the Kinects and demount their foots to mount the vibrators at a center position and ensure a stiff mounting. We have also tried a mounting on top of the Kinect, which revealed to be hard to control and to mount because of the bent shape of the Kinect and which produces much more image blurring than in our setup. Second, we can adjust and fine-tune the vibration amplitude by the position of the Kinect. Since the rods are not mounted on the floor, they vibrate at a higher amplitude near to their end/bottom, where the vibrator is mounted. The vibrator frequency can be controlled by an adjustable power supply.

Currently, our setup uses eight vibrating rods. Additional rods can be inserted with just a few simple steps in less than five minutes. To reduce clutter and to allow defined lighting conditions, we have surrounded the setup by white curtains. Figure 5.3(a) shows a schematic illustration of our setup and Figure 5.3(b) shows the current implementation of this setup.

All vibrators are driven by a parallel circuit at slightly different frequencies, due to different cable lengths. In the final setup, we operate the vibrators between 7200 and 10200rpm.

Display device: To allow real-time visual feedback for various applications, we use a large TV LCD screen display, which can be freely positioned within our setup.

Processing: For reading the sensor data and processing it, we use an off-the-shelf PC similar to the PC used in the multi-camera setup. It is equipped with an NVIDA Quadro 6000 graphics card and four additional VIA USB 3.0 controllers. We also use powered USB extenders to connect all sensing devices. Note that for the setup of multiple Kinects, only the number of physical USB controller chips is important and not the number of USB ports. With this system, we can successfully operate up to seven Kinects if both the color- and the depth stream are used. This is mainly due to the limited bandwidth of the mainboard's south-bridge controller. As driver, we can use either the Microsoft Kinect SDK Driver, or, as for most of our example applications, the OpenNI * PrimeSense Driver.

Calibration: As intrinsic camera parameters, we use the values given in the Microsoft Kinect SDK. We obtain the extrinsic camera parameters by using the same calibration target as in the multi-camera room shown in Figure 3.3. The external calibration is computed from the RGB camera image. The depth image is transformed into the coordinate system of the RGB image by using the static transformation given by the OpenNI or Microsoft Kinect SDK.

5.2.1 Combined visual hull and depth map rendering

Capturing and rendering 3D videos is an important component of a variety of applications, including our virtual try-on system. When rendering from color cameras without depth information, we use the IBVH algorithm. It produces depth maps of the object with clear edges, watertight topology, and no noise, even when using cheap cameras. However, it does not incorporate the depth data that is available in our system: it was designed to work with standard color cameras. Therefore, it fails to reconstruct some of the concavities.

In contrast to color images, depth maps from Microsoft Kinect sensors are rather noisy and suffer from occlusions at depth discontinuities. At the same time, they convey more

*`http://www.openni.org`, retrieved 2013-01-03

information about the shape of the scene. To get the advantages from both depth map-based rendering and image-based visual hull rendering, we combine the strengths of both approaches as follows.

Silhouette-carved point clouds: A prerequisite for visual hulls are silhouette images. To obtain silhouette images, the scene needs to be captured without foreground objects before the visualization starts. We use background subtraction based on color- and depth values to segment foreground objects. This method is more robust than relying on color or depth alone.

Our first method to incorporate visual hull information into point-based rendering is to carve the point clouds on the image planes by only considering depth values that are inside the silhouettes. Silhouettes are calculated by using binary foreground masks, which can be generated by using the aforementioned background subtraction. During our experiments we observed that some of the depth values are outliers even though they are inside their respective silhouettes.

Visual hull-carved point clouds: The visual hull of an object is a conservative surface estimate: It contains the whole object plus space that does not belong to the object. To remove low-quality depth values that caused noise in the silhouette-carved approach, we restrict the point cloud from all Kinects to the 3D space that is covered by the visual hull.

To do so, we perform IBVH rendering followed by point splatting to get both surface estimates. Then, all point splats are culled against the visual hull surface at that pixel.

5.3 Results

To evaluate the quality of the OmniKinect setup, we captured an object of known size: a table with circular top (see Figures 5.4 and 5.5). We then reconstructed and rendered the object with a varying number of sensors with two methods: visual hulls and point cloud rendering.

Figure 5.4 shows the results. For two Kinects, the point cloud that is rendered from the depth data is already quite good when compared to the visual hull. One, two or even three silhouette images do not contain enough information to reconstruct such a surface in a meaningful way.

For a larger number of sensors, the rendering quality of the visual hull improves. It surpasses the quality of point cloud rendering at the edges of the object. The reason for this is that the color cameras can be calibrated intrinsically and extrinsically very accurately, which directly translates to precise visual hull edges. Moreover, the depth sensor has problems at some edges due to occlusions of the projected pattern.

On the other hand, the visual hull may miss concavities that a depth sensor can capture. To combine the strengths, we suggest the visual hull carved point cloud rendering algorithm for the OmniKinect setup. The result can be seen in Figure 5.5: concavities are reconstructed while sharp and precise edges are preserved.

IBVH rendering avoids explicit data representations and is output-driven: for every output pixel exactly one surface intersection is computed. Invisible parts or backsides of the object are not computed, which makes it very efficient. Point splatting is also a very

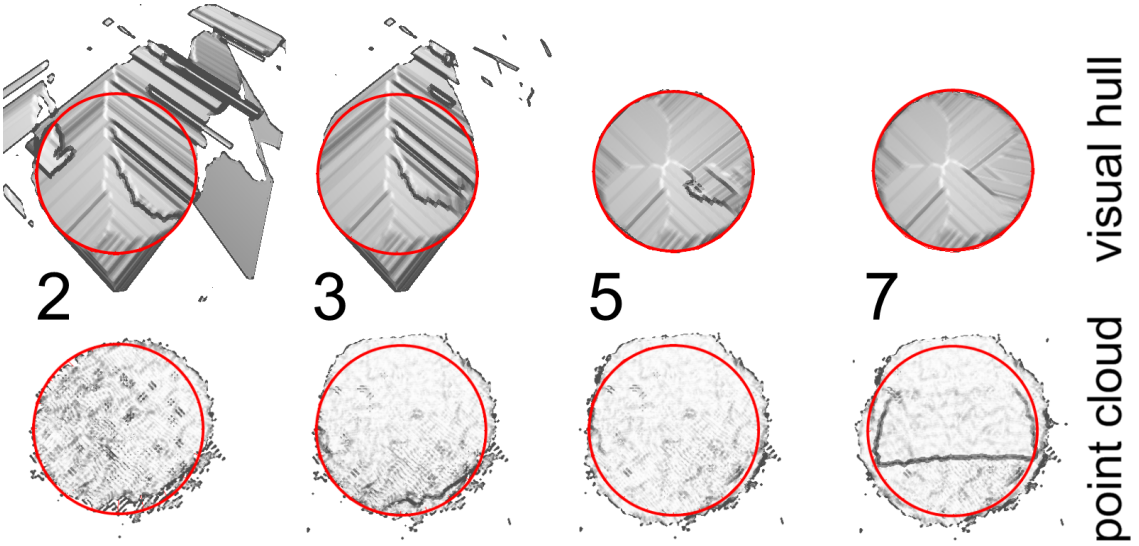


Figure 5.4: The top row shows visual hull rendering, the bottom row point cloud rendering for a varying number of Kinects. The red circle indicates the ground truth diameter of the reconstructed object.

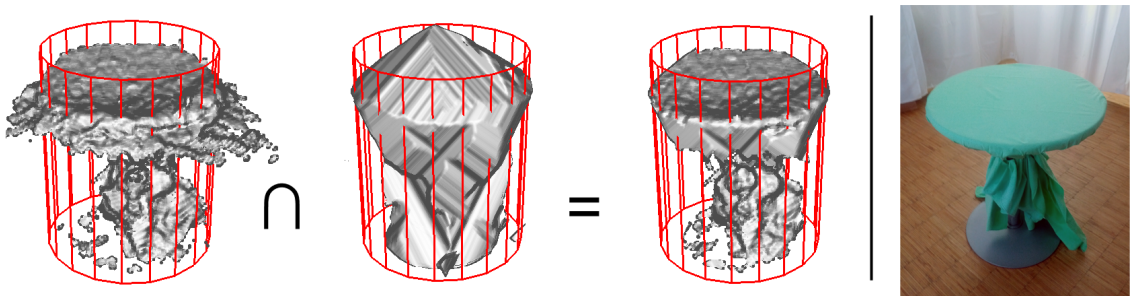


Figure 5.5: Free-viewpoint rendering of a static round table. From left to right: Point cloud rendering directly from Kinect depth maps suffers from low-quality edges. Visual hulls, on the other hand, have sharp edges, but lack concavities. By intersecting the two surface representations, we are able to achieve a much more desirable result. The red cylinders indicate the ground truth diameter of the table on the right.

efficient algorithm because modern GPUs are built for fast geometry transformations. The required scatter operation is also reasonably fast in recent CUDA versions. As a result, the visual-hull carved point cloud algorithm takes around 40 ms to compute on an Nvidia Quadro 6000 at a resolution of 1000×1000 pixels using seven Kinects.

The combined color and depth sensor setup using two Microsoft Kinects computes a similar surface to the OmniKinect setup, but fills missing depth information with visual hull patches. Figure 5.2 shows the results. Both approaches rely on the IBVH pipeline for stability and outlier removal, which results in comparable execution times. Based on surface quality and performance, we could conclude that the two suggested sensor arrangements are equally suitable for real-time surface reconstruction and rendering.

(a) (b)

Figure 5.6: A comparison of the resulting rendering quality of our two sensor arrangements. (a) shows a result image of the OmniKinect setup, (b) of the combined Kinect and color camera setup. Both screenshots were taken during user motion.

However, during our experiments the combined color and depth sensor setup achieved a superior visual quality. When texturing the resulting surface, the bad quality of the color cameras that are built into the Kinect becomes particularly apparent. Seams between the textures of different devices are visible due to the automatic exposure settings. The pictures are dark and have low contrast, which requires additional lighting. Most importantly, the lack of synchronization between the devices causes prominent halo artifacts during user motions. Figure 5.6 shows a side-by-side comparison of the rendering quality that can be expected when the user moves.

The results shown above illustrate that our methods succeed in removing artifacts from a visual hulls. Moreover, concave regions are restored and holes in the depth data are filled with the visual hull surface. Figure 5.7 shows an example of a removed artifact. However, due to the noise in the depth maps, the projective texturing can be distorted from some viewing angles.

5.4 Summary

In this chapter we presented two systems with different sensor arrangements for real-time 3D reconstruction and free-viewpoint rendering. The first setup uses two Microsoft

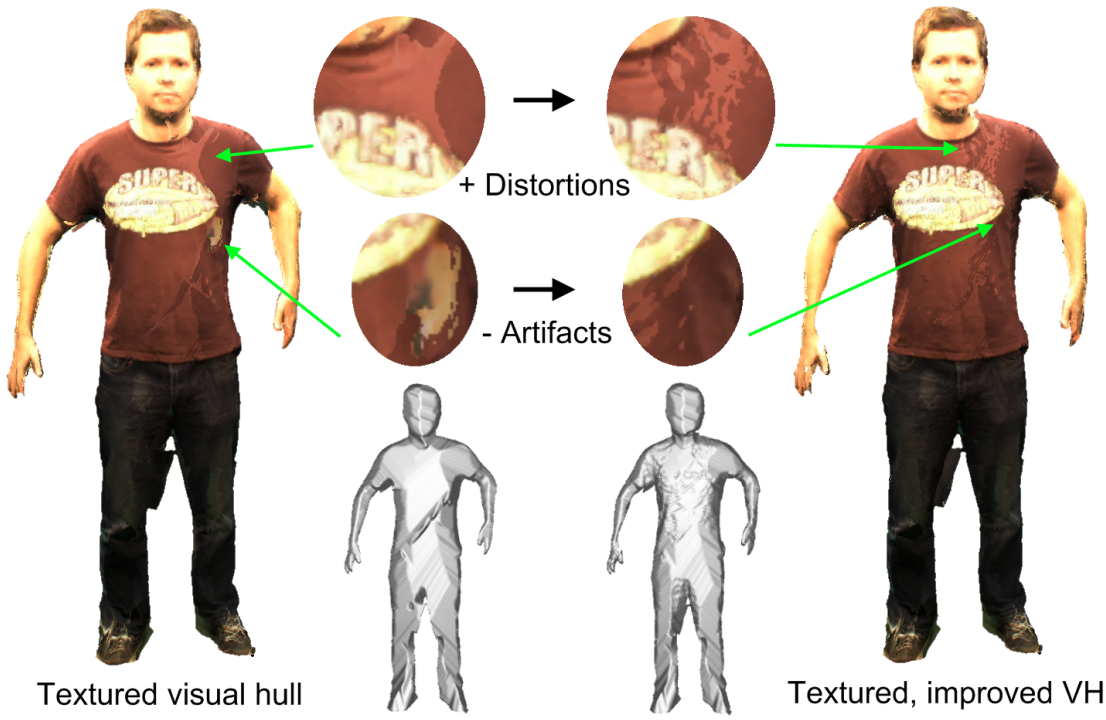


Figure 5.7: A side-by-side comparison of a textured visual hull and a visual hull improved with a Kinect depth map. It can be observed that by improving the visual hull, artifacts were removed but texture distortions were introduced.

Kinect active depth sensors and ten color cameras for capturing. Interferences between the sensors are avoided by mounting them on opposite walls of a room, pointing at the user from different sides. The resulting point cloud is not dense enough for immediate rendering due to a gap between the two reconstructed surfaces. To fill this gap and remove outliers that are caused by noise and occlusions, we suggested to combine IBVH rendering with depth map data. The resulting algorithm improves the visual hull by carving it with the depth measurements. At the same time, it retains the robustness of shape-from-silhouette by falling back to the visual hull surface when no adequate depth value is found.

The second arrangement called OmniKinect uses only Microsoft Kinect devices for capturing the scene. While conceptually simple, the calibration and fusion of data from multiple depth sensors requires careful design and deployment of the hardware setup. The cross sensor interferences were reduced by mounting the devices to vibrating poles. For 3D video capture and free-viewpoint rendering of moving and deforming objects, we introduced visual-hull carved point cloud rendering. It combines the advantages of image-based visual hull and point cloud rendering: precise edges and support for concave objects.

The suggested sensor arrangements deliver a comparable surface reconstruction quality in similar execution times, but differ in terms of quality when the user moves and the surface should be textured with camera images. The lack of synchronization between Microsoft Kinect devices and their inferior color cameras result in a reduced rendering

quality and even artifacts. This becomes especially apparent in the OmniKinect setup where both shape and texture are delivered by Kinect sensors. We therefore suggest to use the combined Kinect and color camera setup for free viewpoint rendering of captured users. To support our image-based rendering hypothesis, we demonstrated a considerable improvement of the reconstruction and image quality in this chapter.

Chapter 6

Temporal coherence of static visual hull patches

Contents

Our multi-camera reconstruction and rendering system utilizes the image-based visual hull algorithm to compute depth maps from silhouette images of calibrated cameras. While IBVH rendering was shown to be capable of meeting real time constraints for relatively small resolutions, our work emphasizes high resolutions with as little latency as possible. The process of IBVH rendering is complex, especially when output quality is improved by stereo matching, which prohibits a full computation every frame.

Users in front of a set of cameras usually move smoothly. As a result, the reconstructed and rendered output image does not change drastically over time. This is usually called temporal coherence and can be exploited to reduce the computation time of the IBVH algorithm. We distinguish two forms of temporal coherence: view coherence and frame coherence.

View coherence relies on the assumption that the viewpoint of the system does not change very fast in a virtual mirror system. Therefore, many of the output pixels of the previous frame can be reused, as long as the foreground object has not moved or deformed at these pixels.

Object or user motion is also usually relatively slow because it requires physical motion and is often restricted to parts of the user, like the arms or the head. As a consequence, the camera images also change slowly. We call this redundancy frame coherence. When the user does not move between two successive video frames there is a maximum of coherence. In such a situation, only the desired viewpoint may change. Most of the depth data from the previous frame can be reused by image warping.

An IBVH-setup is very well suited for exploiting these types of coherence because motion of the foreground object can be detected very efficiently: finding a bounding volume of changed parts is equivalent to computing a visual hull from differences in the silhouette images. The proposed solution leverages the coherence of static parts in free-viewpoint rendering of fully deformable and arbitrarily moving foreground objects. Frame-to-frame forward image warping is used along with an aging mechanism as an efficient algorithm for reusing previous rendering results.

6.1 Exploiting view- and frame coherence

While our IBVH rendering module is capable of rendering the video streams at interactive rates for resolutions up to 1000×1000 pixels, we attempt to scale the algorithm to Full-HD with as little latency as possible. By exploiting view- and frame coherence, we can speed up major stages. For this work, we define view coherence and frame coherence as follows:

View coherence relies on the assumption that the viewpoint of the system does not change very fast in a virtual mirror system. Therefore, many of the output pixels of the previous frame can be reused, as long as the foreground object has not deformed or moved at these pixels. We address this type of coherence with forward image warping which transforms pixels from the previous output frame to the current output frame.

Frame coherence assumes that the input camera images do not change drastically over time. User motion is usually relatively slow because it requires physical motion and is often restricted to parts of the user, like the arms or the head. In these cases, only the regions of the screen which show moving parts have to be redrawn. In addition, our cameras deliver new images 15 times per second, which means, that when we are able to render at higher frame rates, not every output image is based on a novel set of input images. In such cases, only viewpoint motion has to be accounted for.

The stages of the IBVH rendering pipeline (see Figure 3.4) have different input dependencies. First, the segmentation step depends on the color images from the cameras. Edge extraction and caching just depend on the geometric information which is computed from the camera images. Ray-silhouette intersection in addition is dependent on the viewing direction because it needs to reconstruct a depth value for each output pixel. The visibility maps on the other hand are not view-dependent except for little differences in sampling quality. Stereo matching again depends on view and geometry, whereas the final output stage depends on all inputs.

The segmentation step has to be performed every camera frame because it extracts the geometric information from the color images in the form of a foreground labeling. When new camera images are available and parts of the foreground object have moved, all stages after segmentation have to be recomputed, but only in the areas which are concerned by the change. Areas which have not changed, on the other hand, can stay completely untouched in steps which do not depend on the current viewpoint. In view-dependent steps, unchanged areas can be reused from the previous frame by applying an image warping algorithm which compensates for the relative viewpoint motion. The

Figure 6.1: Color- and silhouette difference between two consecutive frames of a camera. Color differences (left) are sensitive to, for example, changes in mimics. Resulting bounding rectangles are indicated in red.

texture mapping stage of our pipeline needs less than 3% of the overall rendering time, and therefore does not benefit noticeably from coherence. We therefore perform texture mapping every frame for the full image, whereas we exploit frame and view coherence for the underlying geometry. Computing the silhouette-ray intersections, stereo matching and visibility depend on the foreground object's shape and position, and therefore have to be performed in screen regions where the underlying geometry has changed. These stages are responsible for over 90% of the overall execution time, which makes exploiting frame coherence particularly rewarding for IBVH rendering.

6.1.1 Identifying moved or deformed regions

An easy way to identify changed camera image regions is to compute the difference between consecutive images. Figure 6.1 illustrates two types of difference images: color difference and silhouette difference. The color difference between images is sensitive to all kinds of changes in the scene, whereas the silhouette difference only shows changes in the projected geometry of the foreground object. Since we specifically aim at accelerating the geometry reconstruction stages of our pipeline, we only want to detect changes in geometry. Therefore, we use the silhouette difference as the detector. It does not contain changes in the foreground object's texture. Texture changes are accounted for in the texture mapping stage which is performed fully every frame.

In addition to building the difference image, this stage simultaneously counts the changed pixels for a block. When a block exceeds a certain fill rate, it is considered as being a changed part of the scene rather than noise in the segmentation, which is similar to a morphological erode operation. We use an axis aligned bounding rectangle to describe the region of each camera's image which contains change. By using CUDA's built-in atomic minimum and maximum operations, we extend the bounding rectangles on a per-block basis to obtain its extents BL_i, BR_i, BT_i, BB_i. Figure 6.1 illustrates the result of this stage. We have chosen axis aligned rectangles as the primitive because they can be fitted easily as described. Moreover, finding intersections with lines can be per-

formed efficiently using minimum/maximum functions and a low amount of floating point operations. This is important for the next stage.

6.1.2 Projecting the visual hull of changed regions

The multi-camera setup in combination with the IBVH rendering algorithm is very well suited for reconstructing geometry from its projections. This is also the idea behind our redraw volumes. The bounding rectangles from the previous step describe the projected boundaries of the changed parts of the object on the image planes of the input cameras. Therefore, the visual hull of the volume containing change can be extracted by applying the IBVH algorithm. To allow later stages to decide whether to recalculate a viewing ray or to apply image warping, each viewing ray (which equals a geometry fragment) needs to be classified as redraw or warp. Since rays can only be recalculated or warped as a whole, the decision is binary. Therefore, the exact depth interval, where each ray intersects the redraw volume, does not need to be extracted. This allows for a rough approximation of equation 3.2:

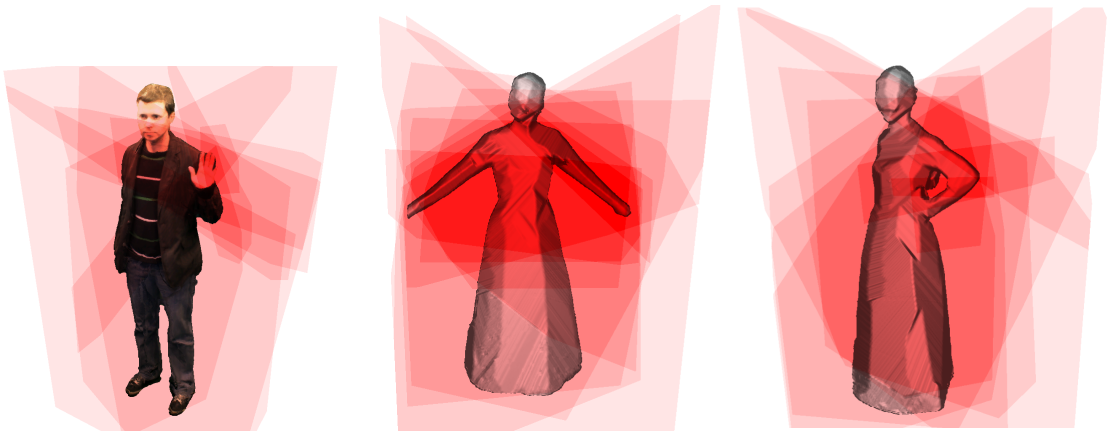
$$R_{x,y} = \left(\sum_{i \in I} hit(start_i, end_i, BL_i, BR_i, BT_i, BB_i) \right) > t_{redraw} \tag{6.1}$$

The binary redraw map $R_{x,y}$ which contains true when a geometry fragment needs to be recomputed is calculated from the sum of intersections of the projected viewing rays $start_i$ - end_i with the bounding rectangles BL_i, BR_i, BT_i, BB_i. *hit* returns 1 if the ray hits the rectangle, and 0 otherwise. This calculation does not correctly reconstruct the visual hull of the bounding rectangles, but an approximation which is good enough for estimating its projection. The computation does not require any indexable memory and only a few branches which makes it particularly fast. Figure 6.2 illustrates the state after the user moved his arms.

To account for occlusion in the scene, we mark a pixel as having changed, when more than a certain number t_{redraw} of the input cameras agree. We use a threshold of two third of the number of cameras, but the exact value may vary for foreground objects other than persons and different camera setups. To achieve more efficiency at this stage, we do not sample the redraw volume for every pixel of the output image, but rather for one representative of a block of 7×7 pixels.

6.1.3 Forward image warping

For transforming pixels from the previous output frame to the current output frame, we apply forward image warping as a CUDA kernel [88]. The algorithm computes the image-space motion between the two consecutive frames for each pixel. It is calculated by transforming a pixel from the previous frame to the current frame by using its depth value and the projection matrices of the current and previous frame. The result is called a warp map and has to be morphologically closed because it does not cover every output pixel reliably. Finally, the pixels are copied according to the map. Note, that we only warp fragments of the reconstructed geometry of the foreground object, but not actual color values because texture-mapping is applied every frame for the reasons mentioned above.



(a) (b)

Figure 6.2: This figure shows the redraw map $R_{x,y}$ which can be used to constrain the rendering to regions with user motion. The users moved their arms or upper body. Our motion detector indicates different motion magnitudes which are illustrated in shades of red. The threshold of Equation 6.1 was not applied in this figure for visualization purposes.

Only parts of the visual hull which have not changed from the previous frame to the current frame are subject to image warping. This is necessary to prevent false geometry fragments from being added to the scene. However, our motion detection algorithm, which triggers the redrawing of geometry fragments, is only a heuristic and may therefore introduce minor artifacts. For example, when very slow object motion which is is not detected by our redraw rectangle mechanism is present for several consecutive frames. Geometry fragments are warped from frame to frame. Therefore, when artifacts occur in one output frame, these artifacts stay in the output until the defective fragments are discarded, for example, by falling into a redraw region. Given enough view coherence and little object movement, fragments can have a life span of several seconds.

To allow image errors to eventually disappear from the output, we introduce an aging mechanism. Every time a geometry fragment is warped, its age is increased by one and stored along with the fragment. When the age of a fragment is too high, it gets discarded. Newly rendered fragments have their age reset. This way, all fragments in the system eventually get replaced. To prevent the algorithm from replacing too many pixels at once, newly created fragments have their age set to a small random value instead of 0. Of course, the age threshold has strong impact on the utility of view coherence. We usually choose age thresholds lower than the expected frames per second. This way, errors are only visible for a fraction of a second. At the same time, it allows fragments to be reused many times before being recomputed which results in a major speed up. Figure 6.3 (a) shows an age map which consists of color-coded ages for each output pixel. Red areas indicate low age which means the corresponding geometry fragments have been recomputed recently. Most

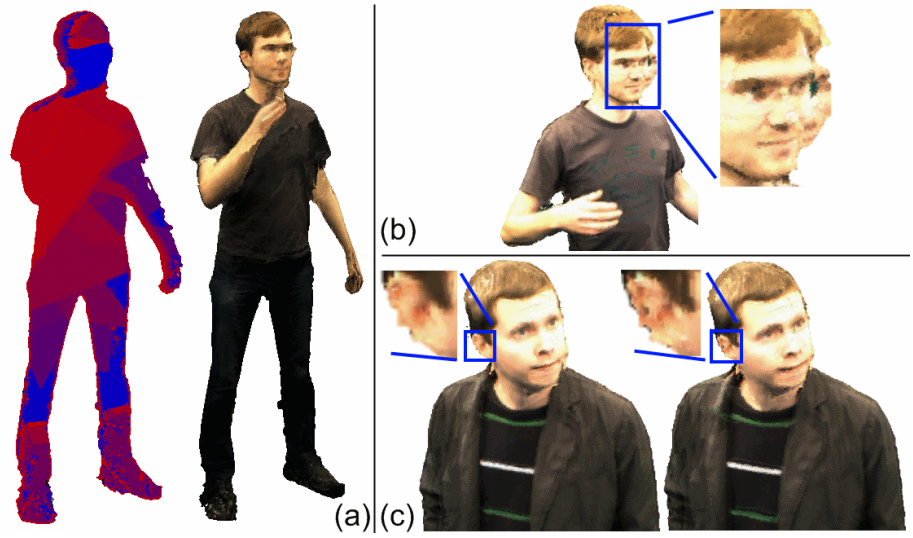


Figure 6.3: Illustration of warp-aging (a). Blue fragments indicate high age, red fragments indicate low age. The snapshot was taken when the user moved his right arm. (b) shows artifacts which can occur when a slow motion is not detected. (c) shows inaccuracies which are introduced by image warping. Warp-aging alleviates the problems of (b) and (c).

of these computations have been triggered by the user's movement of his right arm. Other fragments have been redrawn because their age-threshold has been reached (feet, hair).

6.1.4 The complete coherence work flow

Our method allows for viewpoint- and object motion at the same time. In fact, this is a very common case when a mirror should be simulated. A movement of the user's head both changes the foreground object (the shape of the person) and the viewpoint. The viewpoint is determined by the position of the user's eyes to achieve the mirror effect. The eye position obviously changed along with the head position.

The final rendering pipeline which exploits coherence is illustrated in Figure 6.4. When new input images arrive, the segmentation is performed and silhouette edges are extracted and cached. Also, the changed parts of the scene are identified to yield redraw regions. The visual hull of the redraw regions is projected onto the previous and current frame's image planes, yielding two redraw maps. The warp map is computed in regions which are not classified as having changed by the redraw map of the previous frame.

Afterwards the main step is launched. Geometry fragments which are not labeled for redraw in either of the redraw maps can be warped from the previous frame. For all other pixels, IBVH depth map extraction is performed. These pixels are missing mostly because of motion in the scene, but also due to parts which were occluded in the previous frame. All parts of the user's body are connected, so it can be assumed that previously occluded parts only appear close to previously visible parts. This can be used to avoid IBVH extraction for pixels which have a certain distance to the next valid warp map entry and are not subject to redraw due to scene change. This distance defines the maximum

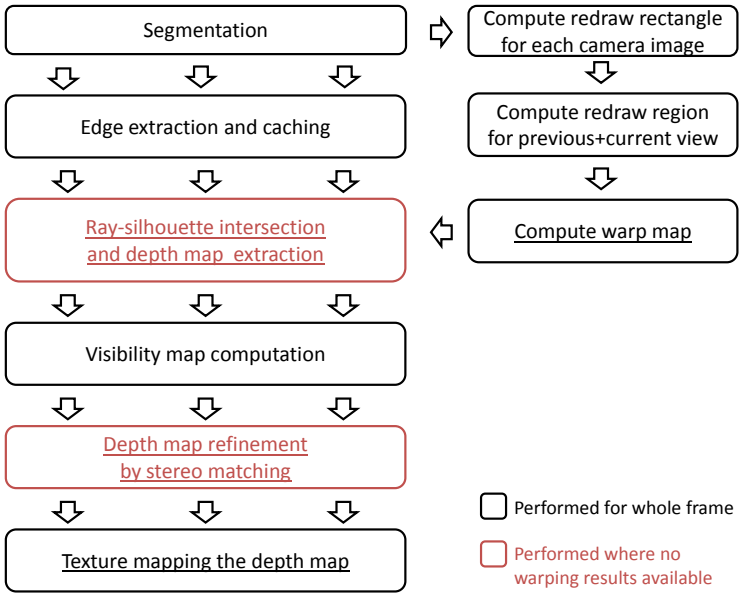


Figure 6.4: Our method which exploits frame- and view coherence. Red steps are only performed in regions where the scene changed or a novel view reveals previously unseen parts. Underlined steps are view-dependent.

allowed projected user motion per frame. We chose a distance of 8 pixels, which was rarely exceeded. Depending on the amount of background pixels, we observed that this effect can decrease runtime by up to 15%.

From there, visibility maps can be computed. Depth map refinement by stereo matching is performed only in regions where geometry changed. Other regions can be reused by utilizing the warp map. Finally, the texture mapping step is launched every frame for the whole output image, since changes in texture color are not detected by our redraw regions by design. This way, fine details like mimic and hair movement are updated as often as possible. Note, that the view-independent steps (Figure 6.4) only need to be performed when new camera images are available. When the rendering speed exceeds the camera update rate, many computations can therefore be avoided.

For interactive systems, it is desirable to directly constrain runtime and to maximize the image quality within the available time-frame. By using our temporal coherence method, a system implementing this behavior can easily be realized.

6.2 Guaranteed frame rate

The image-based visual hull (IBVH) algorithm estimates images at novel viewpoints directly from segmented camera images. It is particularly useful in interactive systems that must produce output images with little delay. In this chapter we showed that without sacrificing image quality, computing the IBVH can be constrained to those image areas in which the underlying geometry has changed [90]. After detecting changes in the scene geometry, only those image areas with a change magnitude above a threshold are recomputed

and the remaining areas are warped from the previous image. This threshold controls the output quality directly, but it only indirectly affects the execution time.

By exploiting coherence the rendering frame rate can vary drastically over time, depending on the user's motion. When immediate response to user motion is desired and should be guaranteed, a certain amount of rendering time must not be exceeded. Terminating the coherence-enhanced visual hull rendering process after a certain execution time leads to incomplete output images. Depending on the block scheduling order of the CUDA runtime, the top half of the image might be empty, for example.

To make most use of a fixed time window, the image regions that change the most between consecutive frames should be rendered first. We can approximate the change magnitude of an output pixel or a block of pixels by our change detector, the redraw map $R_{x,y}$ that can be seen in Figure 6.2. Image blocks with high motion magnitude should be subject to full IBVH rendering, while others may be approximated by image warping. After a certain fraction of the time window is over, all remaining image blocks should be filled by image warping to avoid holes or missing parts in the output.

However, this process is not possible using standard GPU programming facilities, like Nvidia's CUDA framework. We need a custom execution order of image blocks and a way of deciding when to switch from IBVH rendering to image warping. The Softshell [197] GPU scheduler adds these functionalities to CUDA.

The custom scheduler is implemented as an infinite loop that keeps all processors of the GPU busy (a so-called megakernel approach). Coherent blocks of threads query workpackages from a work queue. When finished, the block queries the next workpackage until no more work is to be done. The work queue is sorted before and during execution by a user-defined priority. The elapsed execution time starting when the first workpackage is issued can be queried to guide kernel execution.

Softshell workpackages are created for the entire image by the CPU. Their priority is set by our motion detector to the average change magnitude of their associated region, so that image areas with substantial changes will be processed first. During execution of each workpackage, the remaining time and remaining number of workpackages are queried. Based on the known, constant execution time of image warping, the kernel can decide whether to run the IBVH algorithm or image warping.

6.3 Quality discussion

The proposed image-based rendering architecture contains components which directly operate on camera images and therefore inherit several computer vision issues. Moreover, the redraw heuristics have an effect on the output quality. In this section, robustness and influence on quality of critical components are discussed.

The first step in the pipeline performs a foreground segmentation. We assume a rather controlled real environment which means in practice, the segmentation step is sufficiently robust. We experimented with garments of different colors and even with green stripes, but output quality remained stable. However, changes in the background scene produce intolerable visual artifacts.

We also observed the effect of morphological closing. This algorithm fills holes and cracks in the segmentation which usually results in a more correct segmentation and thus in improved visual quality. However, the operation may wrongly fill concavities, for example, face details. In practice, these artifacts are hardly noticeable in the output image because the stereo matching step shifts depth values to improve color consistency, and color outliers are also suppressed while texturing.

After the segmentation of all camera images, the redraw rectangles are computed by silhouette differences between the current and the previous image. While this step is robust to illumination because it operates on silhouettes instead of colors, the resulting bounding boxes are subject to segmentation noise. As described, we counter most noise by filtering, but bounding boxes may occasionally be too large. This results in more pixels which are reconstructed by IBVH, and less pixels which are warped form the previous frame. The visual quality therefore does not suffer from this, but rendering performance does. In our evaluation system, this was rarely a problem, but in less controlled environments, using more than just one bounding box per image may be more stable in this respect.

The computation of the redraw maps follows a heuristic. It contains a threshold which can be tuned in favor of performance or quality. By choosing a high threshold, minor motions of the user stay undetected and result in slight artifacts in the output image. To deemphasize this tradeoff, we introduced the concept of warp aging: all fragments are recomputed eventually. By carefully choosing the maximum warp age, artifacts are not visible long enough for users to notice, while still maintaining a frame rate which is considerably higher than the conventional IBVH approach produces. Finally, image warping itself adds a low amount of blur to the output which is mostly due to inaccuracies of repeated warping (see Figure 6.3).

The standard implementation sets a desired visual quality and lets the execution time vary freely. The guaranteed frame rate implementation follows the same quality/speed tradeoff, but the desired maximum execution time can be fixed while the quality varies. With a decreasing execution time limit, the quality degrades gracefully, as the rather static visual hull patches are approximated by image warping first. Obviously, setting the time limit too small leads to visible artifacts as moving visual hull patches do not receive full IBVH reconstruction.

6.4 Results

We use a recorded scene consisting of 10 uncompressed AVI videos with 640×480 resolution and 15 Hz. A frame of these videos is shown in Figure 6.2. The output resolution is set to 1600×900, with about 1550×750 effectively covered by rendered pixels.

Evaluating frame- and view coherence effects is not trivial because it strongly depends on the user's behavior in terms of physical movement and viewpoint movement. Therefore, for evaluating the rendering of a frame by exploiting frame coherence, we use three exemplary test scenarios. The first consists of a relatively strong user motion which involves the whole body. The second scenario only consists of the movement of an arm as shown in Figure 6.2. The third does not show any motion, and therefore leverages full frame coherence. For all scenarios, we moved the viewpoint slightly and averaged the

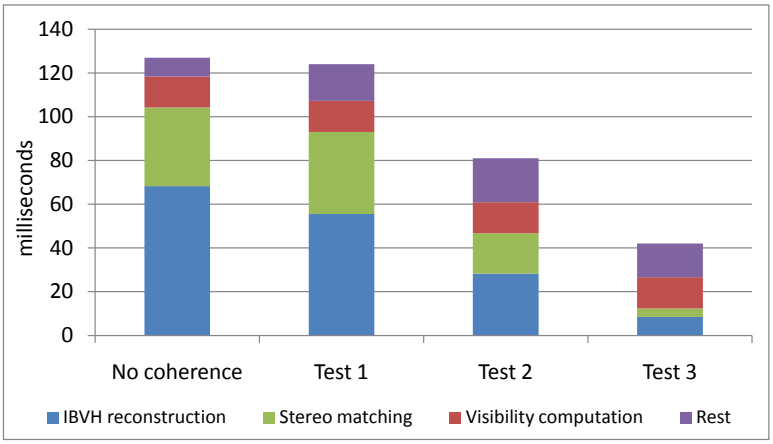


Figure 6.5: Comparative evaluation of kernel times in milliseconds: without using any coherence, weak coherence (test 1), medium coherence (test 2) and strong coherence (test 3).

recorded results. Together, these situations illustrate the range of possible performance improvements.

Figure 6.5 compares average kernel times of our IBVH rendering pipeline with and without exploiting frame- and view coherence. Test scenario one, which contained full motion of the user's body, resulted in an average rendering time of 124 ms. This is about as fast as without exploiting coherence. While the additional stages of the pipeline need time, the disadvantage is less than we expected: even with full user motion, the redraw volume still helps to speed up the detection of empty pixels. Scenario two already offers a substantially increased performance: the runtime is only 64% of the conventional pipeline. As we expected, test scenario three is fastest with a rendering time of only 33% of the original.

The benefit of exploiting frame- and view coherence with our algorithm improves with increasing output resolution (Figure 6.6) which makes it suitable for future high quality augmented reality systems. Moreover, we configured our system to the video conference setting of a recent CUDA-based IBVH implementation [221], where a performance of 25 frames per second for a 768×576 resolution using 9 cameras was reported. Our system delivered 52 to 63 frames per second in such a scenario. While we used a more recent GPU, our system additionally performed stereo matching, visibility tests and a more elaborate texture mapping.

The guaranteed frame rate implementation follows the same quality/speed tradeoff as the quality-driven execution scheme. The most important feature is therefore its ability to obey the given execution time limit. In the evaluation shown in Figure 6.7 we set the limit to 67 milliseconds, which equals the camera frame rate. We compare the execution times for both the guaranteed frame rate approach and the quality-driven implementation. It can be observed that the limit is exceeded rarely and only for a short time. The remaining variance in the execution times comes from the GPU-internal scheduler that is not fully controllable by our program.

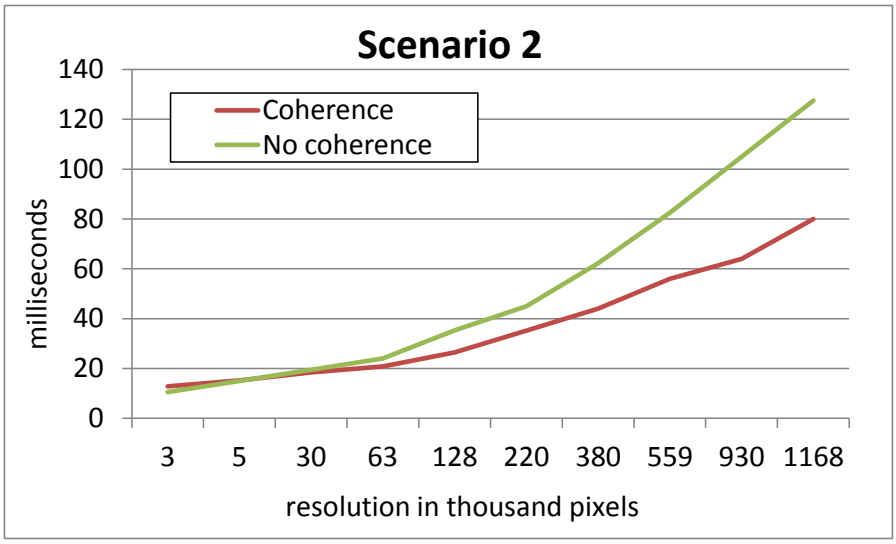


Figure 6.6: Execution times in relation to output resolution for scenario two. The benefit of exploiting coherence increases with increasing output resolution.

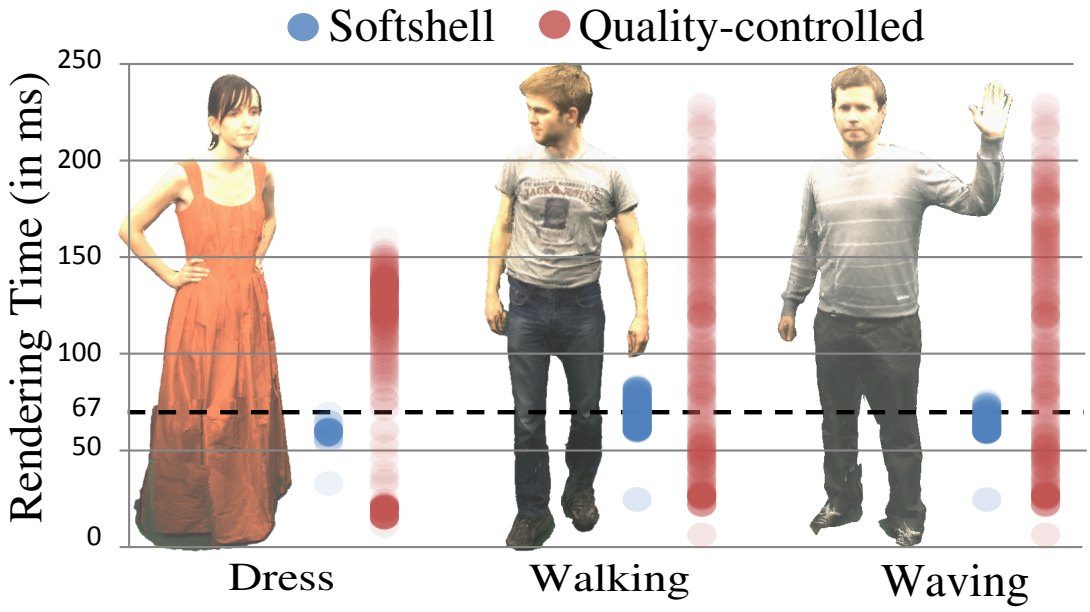


Figure 6.7: Guaranteed frame rate IBVH rendering with a target time-constraint of $67ms$ to match the camera frame rate. A fixed quality threshold determined prior to a kernel-launch leads to unpredictable execution times (red dots). The Softshell implementation controls the quality/speed tradeoff dynamically, based on the time remaining for the current frame. Work of higher-priority is scheduled first, generating the highest possible quality in the specified time (blue dots).

6.5 Summary

In this chapter we have shown how the performance of image-based visual hull rendering can be improved by leveraging frame- and viewing coherence. An effective and efficient motion detector was introduced that exploits the IBVH algorithm itself. It identifies pixels that can be reused by image warping. Image warping is a quick way to transfer surface points from the previous frame to the current frame, thus reducing the number of operations that are necessary to generate an output image. It does not restrict the flexibility, but improves performance considerably. Moreover, we demonstrated how to use such a combined IBVH and image warping system to guarantee frame rates. As a result, augmented reality applications which build upon our method can deliver high resolution output images with very low latency. The evaluation results verify our temporal coherence hypothesis. We evaluated the relative performance improvements for a range of output resolutions and use cases with differing amounts of coherence. The suggested coherence algorithm scales well with resolution, which makes it particularly useful for future interactive rendering setups. Performance improvements could be measured even for scenes with full body motions because the algorithm facilitates early background detection.

Chapter 7

Temporal coherence of dynamic visual hull patches

Contents

Exploiting the temporal coherence in the video streams during IBVH rendering can reduce the number of required calculations that are necessary for image generation. This is challenging because the user is reconstructed every frame and can move and deform arbitrarily. In the previous chapter we showed how to detect static visual hull patches that can be reused over time. Exploiting the coherence also for moving surface patches is challenging and requires modifications to the original IBVH algorithm.

The contribution described in this chapter is a method to utilize previous computation results to improve the performance of the IBVH algorithm even during user motion. To achieve this task, we first analyzed the IBVH algorithm to find intermediate computation results that are coherent over time and verified their persistence by measurements. We found that visual hull points have an association to silhouette edges that persists over several frames. This association is therefore more stable than the surface point locations themselves. This knowledge can be used to save computations in regions with stable camera associations (see Section 7.2).

Moreover, users usually do not move quickly in relation to the camera update rates which can be exploited. We analyzed the camera associations of ray-silhouette intersection intervals in the light of an assumed maximal user motion speed. Based on this constraint, we introduce methods to determine irrelevant and stable ray-silhouette intersections to reduce the amount of calculations over time (see Section 7.3).

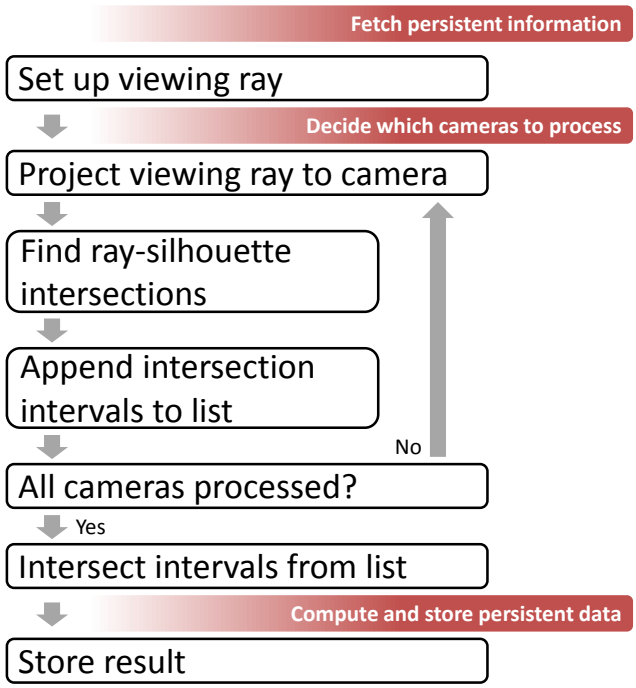


Figure 7.1: The black boxes describe the process of image-based visual hull (IBVH) rendering for a single viewing ray. The red boxes indicate where extensions are required to exploit temporal coherence.

7.1 Temporal coherence in IBVH rendering

The work presented in the previous chapter was limited to static surface patches. Exploiting coherence for surface patches that move or deform over time is more difficult. Due to the unpredictable deformation, previous depth maps can not be reused reliably. Figure 7.2 shows that a big fraction of visual hull surface points (computed from the depth maps) can become invalid between two consecutive frames. The IBVH algorithm computes the depth map directly from the silhouette images, which means that without modifications to the original approach there are no intermediate results that reliably stay valid over two consecutive frames.

We found such information and methods for determining its reliability. Figure 7.1 indicates how the conventional IBVH algorithm can be extended to exploit temporal coherence. First, the information that is persistent over consecutive frames is fetched. It helps to decide which cameras should be processed by the IBVH algorithm. Finally, the persistent data for the next frame needs to be extracted and stored.

The following sections measure the stability of intermediate computation results over time and derive strategies for exploiting coherence from the measurements. First, we describe how surface points of a visual hull are associated with the cameras and show that this association is more stable than the surface points themselves. This knowledge can be used to save computations in regions with stable camera associations.

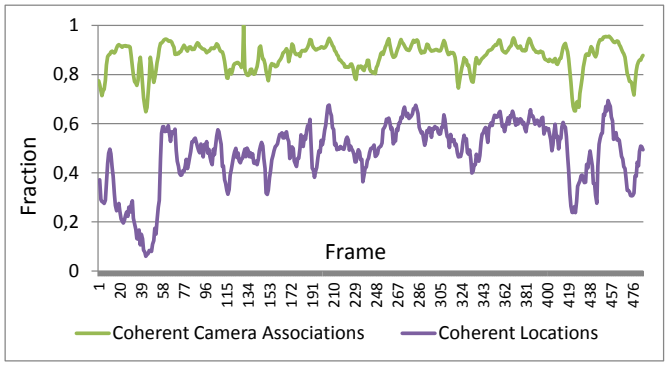


Figure 7.2: Coherence analysis: the fraction of camera associations and surface point coordinates that stay valid between two consecutive frames are compared. Camera associations show more coherence.

7.2 Camera association coherence

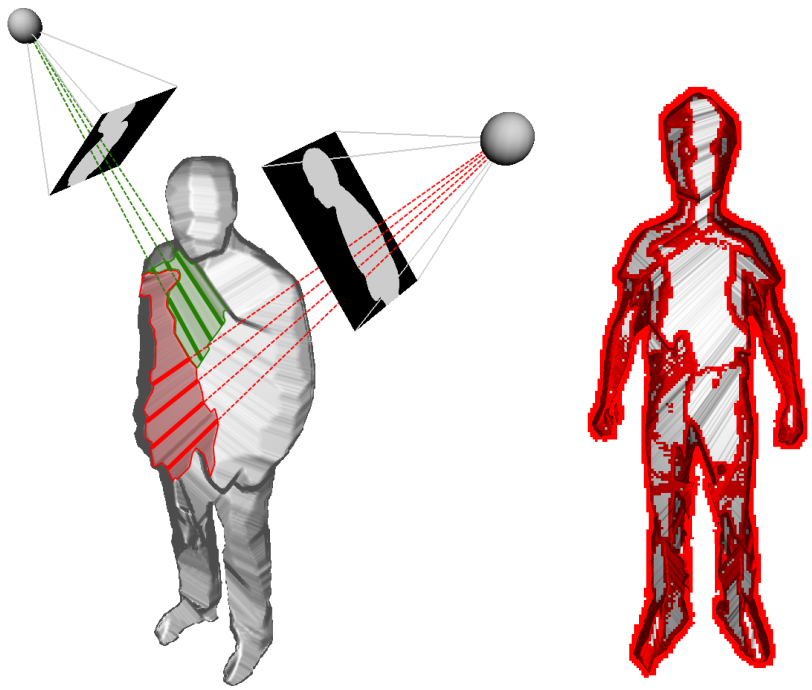
One important property of visual hulls is that every surface point projects to a silhouette edge in at least one of the camera images. Most surface points do not lie on visual hull ridges or junctions and therefore project to a silhouette edge in exactly one camera (Figure 7.3(a)). Therefore, most surface points can be computed by finding ray-silhouette intersections in only one instead of all camera images.

The association of a surface point with a single camera is not static over time. However, we found it to vary smoothly. To determine the amount of coherence, we therefore measured the percentage of surface points (equivalent to pixels for the IBVH algorithm) that do not change their association between frames over a whole sequence of user body motions.

Figure 7.2 shows the fraction of surface points that keep their camera association over a recorded sequence. It can be observed that most associations stay valid between consecutive frames. In contrast, the surface points' locations do not provide as much coherence. To account for numerical instability, we assumed that a surface point stays valid if it moves less than one millimeter between frames.

To quickly compute a surface point from a coherent camera association it is enough to execute a subset of the original IBVH algorithm: the ray-silhouette intersection. It returns a list of intervals that describe where the viewing ray runs through the reconstructed object. The frontmost interval is not necessarily the correct one. In complex cases where some of the intersection intervals would be carved by other cameras, it is necessary to know which of the intervals is correct. This means that the temporally persistent information of this method is a camera ID and an interval ID. In addition, we store a stability measure.

We therefore extended the conventional image generation algorithm by three stages (Figure 7.1). First, the two ID numbers and the stability value are fetched from a buffer for each pixel. The IDs can be used directly to decide which camera is processed. When a pixel is flagged as unstable, the conventional IBVH algorithm is executed by processing all cameras. After the surface point is computed, the associated IDs are stored as the new



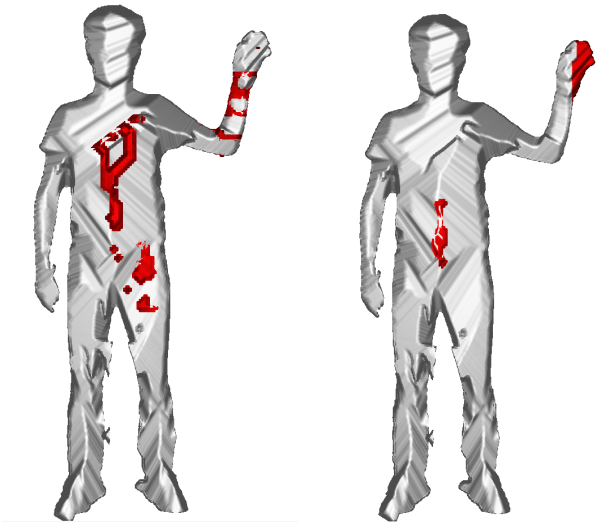
(a) (b)

Figure 7.3: (a) shows how silhouette edges generate surface points. (b) illustrates the stability estimation used in our method: blocks with inhomogeneous camera associations are marked as unstable (red).

persistent information. Finally, the expected stability needs to be computed for the next frame. The stability computation is subject to a tradeoff.

7.2.1 Quality/speed tradeoff

While many surface points keep their camera and intersection association between frames, not all of them do. Especially at the borders between different camera associations, surface points change their sides frequently. To avoid artifacts in these regions it is therefore important to identify and mark them as unstable, meaning that they require a full IBVH computation that considers all cameras. Figure 7.3(b) shows an IBVH rendered image with augmented stability information. The width of the unstable area around patches with inhomogeneous camera associations determines the execution time and the result quality. Increasing the width causes more full IBVH computations and therefore increases quality at the cost of performance. The result of setting a too small width can be seen in Figure 7.4(a).



(a) (b)

Figure 7.4: This visualization shows the impact of a too small instability width for camera association coherence (a), and a too small maximally expected motion for user motion coherence (b).

7.3 User motion coherence

Our application mainly focuses on reconstructing and rendering of people. In this case, we can safely assume a certain maximum velocity the user moves his body (parts). Most likely the hands will move the fastest. We currently assume a maximum velocity of 135 centimeters per second, which was sufficient during our experiments.

The method described above assumes that camera associations vary slowly over time. This proved to be true for large surface junks like the upper body of people. However, body parts like hands and legs can move much faster and are relatively thin. When considering a maximal motion of 135 centimeters per second around each surface point it becomes obvious that the association to a single camera can change quickly, especially for arms and legs of the user.

The camera association method covers these changes by a stability predictor that marks border regions in the association buffer as unstable. If we set the three dimensional extent of these border regions to the maximum motion distance that we expect in our system then most of the surface would be identified as being unstable. As a result, the method can be configured to favor speed or quality, but does not achieve an optimal tradeoff between these two goals. The situation is better for a rather low number of cameras, but for quality reasons we want to use as many cameras as possible.

The solution is to associate every surface point with a list of cameras that contribute important ray-object intersection intervals. This is the temporally persistent information of this method. A list containing all cameras is maximally stable because it allows to

compute the visual hull conventionally even without any temporal coherence. With an assumed maximum user motion, cameras can be removed from the list to reduce the processing time.

Again, we extended the conventional image generation algorithm by three stages (Figure 7.1). First, for every pixel the camera list is fetched in a certain neighborhood because camera associations should be distributed to all surface points inside the maximal motion range. When the new image is generated, only these cameras need to be processed. All other cameras can safely be skipped. During camera processing, we keep track of all cameras which generate intervals that lie within the maximal motion range around the surface point. After the surface point is computed, the new camera list is stored. As long as the expected range is sufficient, the resulting algorithm computes the exact same result as the conventional algorithm, but faster.

Compared to the single camera association method, we get less benefit per surface patch because the processing is reduced to a number of cameras instead of just one. However, more surface patches receive a reduced work load.

7.3.1 Quality/speed tradeoff

The maximum motion that we expect from the user has a strong impact on both the visual hull quality and the execution time. Usually it is estimated conservatively, which guarantees that the result will be the same as with conventional IBVH rendering. However, the maximum motion limit can be lowered to reduce execution times when certain errors in the resulting visual hull are tolerable during fast user motions. Figure 7.4(b) illustrates how such artifacts look like. In the evaluation section we provide measurements for a range of motion limits to illustrate the behavior of the algorithm.

Usually, different body parts and thus image regions are subject to different maximum velocities. For example, it is hard to move the head as quickly as the hands. Moreover, the motion velocity can vary over time as the user performs different actions. Therefore, the performance of exploiting temporal coherence can often be improved by applying different motion limits across space and time. To do so it is necessary to estimate the user's motion magnitude before reconstructing and rendering his body.

We achieved this with the motion detector (Chapter 6, Figure 6.2) which was used to identify surface patches suitable for image warping. It computes an approximate probability that a surface patch has moved between two consecutive camera image sets. Patches that move quickly are more likely to be detected and achieve higher scores, which allows us to estimate the velocity from this score. In our temporal coherence algorithm we adapt the maximally expected user motion according to it.

7.4 Implementation

The IBVH pipeline is implemented as CUDA kernels. It starts by uploading the camera images to the GPU, performs background segmentation and compensates for radial distortion. Then, an angular cache [90] is built that improves the performance of the subsequent IBVH step. This chapter focuses on the IBVH step, which we extended by three stages (see Figure 7.1): fetching the persistent information, deciding how to compute the pixel

and storing the results. In addition, background pixels that are outside the scene bounding box or too far from previous surface points are excluded from further processing.

All coherence methods use the CUDA warp voting functionality to find decisions that avoid diverging branches. Therefore, all decisions are made conservatively: if one thread in a warp requires full reconstruction then all receive full reconstruction. Shared memory is used when data needs to be available to all threads in a block for decision making.

7.4.1 Camera association coherence

The persistent information in this approach is a camera ID and an interval ID that are associated to the surface point at each pixel. The stability buffer only stores one value per block of 8×8 pixels to facilitate divergence free execution. Rendering with camera association coherence is a three-fold process. First, the modified IBVH rendering algorithm checks the stability buffer at each pixel. To compensate for viewpoint changes between two consecutive frames, the read location is transformed by image warping.

Second, the algorithm decides whether a patch of pixels should be reconstructed fully using the conventional IBVH algorithm or if it is safe to reuse information from the last frame. In the latter case, the ray-surface intersection is computed by only looking up the camera that each surface point is associated to. Otherwise the conventional algorithm is run. The new surface point location and its camera association are written to the result buffers.

Third, the persistent camera association buffer is checked for surface patches containing visual hull ridges, i.e. inhomogeneous blocks. These blocks are marked as unstable for future coherence decisions. Nearby blocks are also marked unstable. The object space range of this distribution is fixed before runtime. When transformed to image space, the range depends on the resolution and zoom level. It can be used to control the quality/speed ratio of the algorithm (see Section 7.5). In addition to inhomogeneous blocks, the background is also marked as being unstable to allow the visual hull to move into previously unoccupied space.

7.4.2 User motion coherence

This method also modifies the conventional IBVH algorithm at three points. First, the persistent information in the form of a camera list is fetched per pixel. The list describes which cameras should be traversed. We store the cameras's active/discarded status in a bit array per 8×8 pixel thread block. This ensures coherent warp decisions in the next frame. The bit mask also covers background regions and therefore helps to skip empty blocks quickly. To compensate for viewpoint changes between two consecutive frames, the read location is transformed by image warping. The camera IDs are aggregated by a logical or operation in an image space neighborhood that corresponds to the maximal expected user motion distance. This distance is defined in object space and therefore needs to be transformed to image space to account for varying zoom levels and resolutions.

The current implementation utilizes the a_i angle described in Section 3.4.4 to sort the cameras before processing. For pixels that showed relatively thick parts of the visual hull and that are not close to the background we assume that they will contain foreground in

the next frame too. There we sort the cameras with low a's first. This way, the resulting surface is found quicker and the remaining cameras can be analyzed for being necessary in the future. For pixels that are likely to show background, we sort the cameras with high a's first. This helps to discard such pixels quicker.

During IBVH processing the cameras in the list are traversed in the specified order. The algorithm keeps track of the ray-silhouette intersection intervals and their camera associations. Cameras that are not necessary to find the surface intersection along a viewing ray are discarded for the next frame. Reasons why a camera may be unnecessary for a ray are: the intervals do not cover and are far away from the surface. For example, a camera is unnecessary if it only carves space that is more than the maximal motion distance in front or behind the visual hull. Other reasons are that the ray does not intersect the viewing volume or the ray misses the object in this camera's silhouette image.

Finally, the new camera list is aggregated per block of pixels by an atomic or operation and stored in the buffer. Before the user enters the scene, all lists are empty. Therefore, when the user enters the scene, the algorithm would not notice. To bootstrap the process, we activate random camera IDs in random blocks every other frame.

7.5 Results

We evaluated our methods by measuring execution times and relating them to conventional IBVH rendering and voxel-based visual hull reconstruction. We measured how our methods scale with different resolutions and how their result quality degrades with different parameter settings. The test system is equipped with an Nvidia Quadro 6000 GPU and executes all tests on previously recorded video streams instead of life camera data to generate reproducible results.

7.5.1 IBVH rendering vs. voxel carving

As a first test, we compare IBVH rendering to voxel carving. Voxel carving is a popular method for shape-from-silhouette reconstruction. It consists of two stages: first, each voxel is projected into all camera images and labeled as *outside* if it projects to the background in any of the images. Second, the voxel grid needs to be rendered. We use raycasting in our tests because extracting a mesh from a frequently updated voxel grid is less efficient on our evaluation system.

For the tests we selected three different image and voxel resolutions. To allow for a fair comparison, the voxel resolutions are selected such that the visible voxels project to an equal number of pixels. This way, the voxel and the image-based methods compute geometries of comparable quality. Figure 7.5 shows how voxel carving and rendering compares to the IBVH approach: the voxel carving method is only competitive for small resolutions. Voxel resolutions were $700 \times 350 \times 1400$, $450 \times 225 \times 900$ and $250 \times 125 \times 500$ for the three test runs.

Voxel carving can be improved by exploiting temporal coherence. This approach is known as incremental visual hull reconstruction [11]. It works by casting rays through the volume at silhouette pixel locations that have changed between two consecutive frames. If a pixel becomes activated, the voxel grid along the according viewing ray is updated.

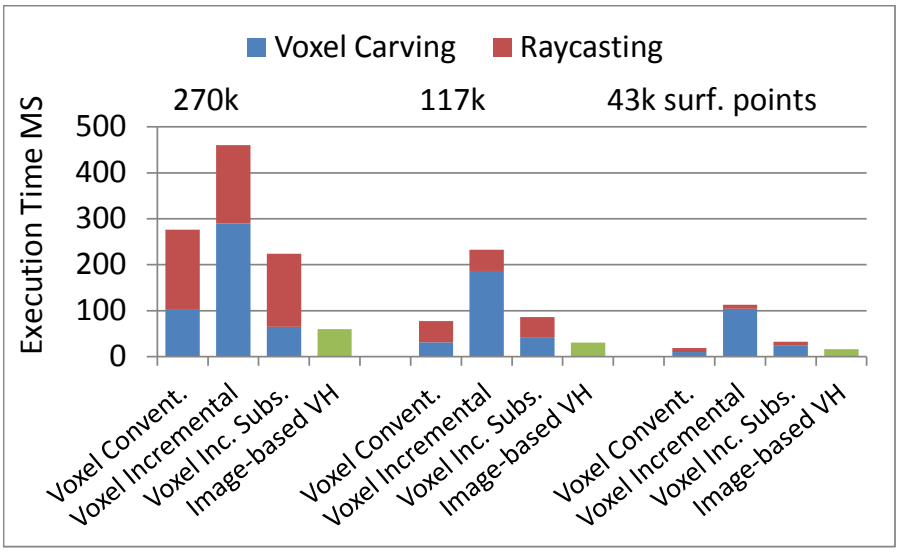
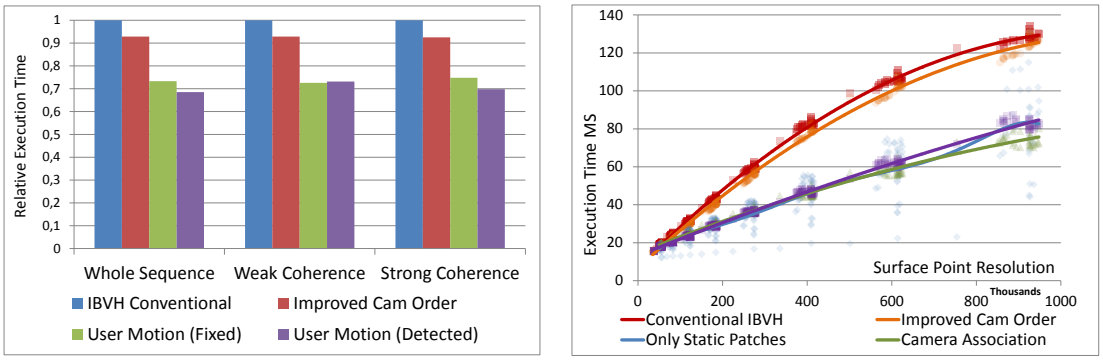


Figure 7.5: This evaluation data plot compares IBVH rendering with (incremental) voxel carving at three different image and voxel resolutions.

When a pixel gets removed, the voxel grid along the viewing ray gets carved. It is possible to subsample the silhouette images prior to carving to gain performance. We implemented it on our CUDA-based platform and compared our methods to it. Figure 7.5 illustrates that voxel carving, even when executed incrementally and subsampled to a quarter of the original resolution, does not match the IBVH rendering performance. We even observed that incremental updating can have a severe performance impact. The suggested subsampling scheme alleviates the problem but degrades the result quality. We assume that the reason for this is the scattered writing pattern in the voxel grid that is not efficient on the CUDA platform. This becomes particularly apparent at higher resolutions.

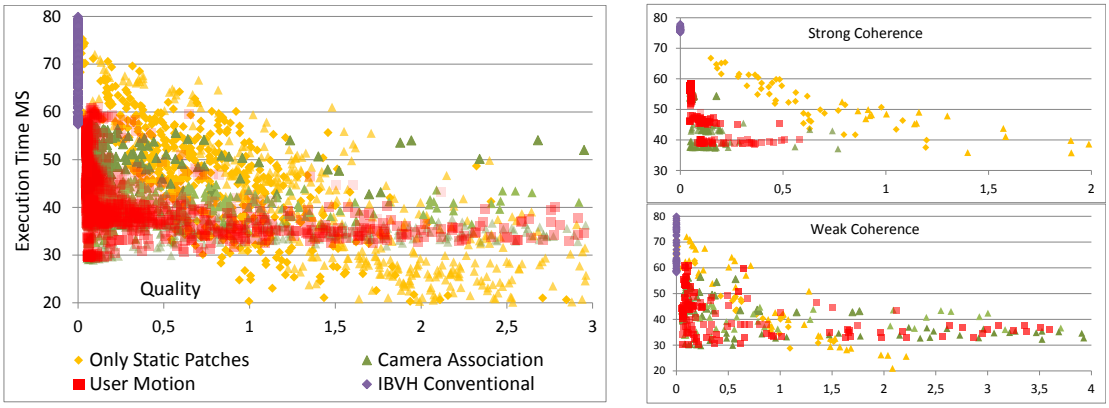
7.5.2 Loss-less methods

In our second test we compare execution times to the conventional IBVH algorithm. We use a method that does not degrade the visual hull quality: the user motion coherence approach with a sufficient maximal motion range. Figure 7.6(a) shows the results for a recorded sequence of various user motions. Averages are built over the whole sequence, over rather static frames with strong coherence and over dynamic frames with much user motion. The improved camera sorting by angle a_i reduces execution times by around 7%. Considering the maximally expected user motion (135 cm/s in this case), the run times can be decreased by another 20%. When estimating the maximum motion distance by utilizing our motion detector, we could decrease run times by up to another 5%. For frames with much user motion, however, the performance gains do not cover the additional computation time of the motion detector.



(a) (b)

Figure 7.6: Evaluation of the suggested methods. (a) shows the performance of the loss-less methods. (b) shows performance measurements for an increasing resolution.



(a) (b)

Figure 7.7: Quality versus execution time evaluation of the suggested methods. (a) shows a sequence with various user motions. (b) shows subsets of frames with weak coherence and strong coherence. Quality is measured in millimeters deviation from the ground truth (IBVH Conventional). The lower left corner therefore means *better*.

7.5.3 Scaling with resolution

The execution times of the IBVH algorithm and our temporal coherence extensions scale strongly with the output resolution [87] because for every pixel a surface point is computed. To draw conclusions from our evaluation runs it is therefore important to check whether the relative performance gains are representative for a larger resolution range. Figure 7.6(b) shows the execution time of all our temporal coherence methods for a wide resolution

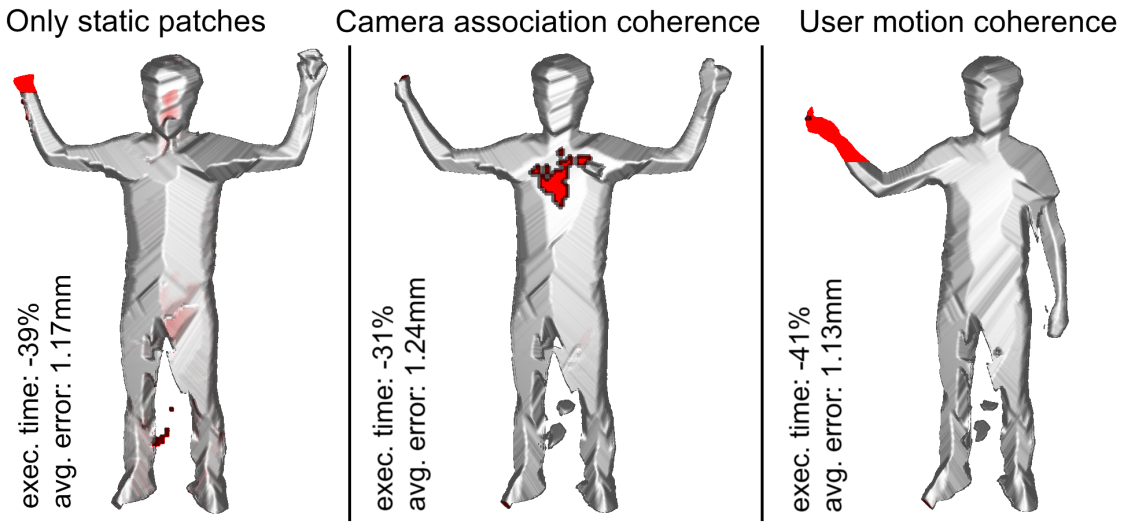


Figure 7.8: This figure shows output images with relatively bad quality, augmented with the difference to the reference image in red. It contains our previous approach for static visual hull patches (left) and the new temporal coherence methods. The relative performance improvement over conventional IBVH rendering is shown and error measurements are given.

range. Execution times scale sub linear with the number of pixels and maintain their relative order. It can be observed that the relative performance gains from the coherence methods and our previous approach (*Only Static Patches*) increase until around 700×700 surface points and then stays constant until it slightly decreases at roughly one million surface points resolution.

7.5.4 Quality/speed tradeoff

To evaluate temporal coherence methods it is not enough to measure execution times alone. It is also important to see how an algorithm performs when the coherence assumptions are broken. This is important when either the user's motion speed was underestimated or the parameters were set to intentionally favor speed over full quality.

We therefore decided to measure the execution times of the suggested methods and relate them to the output quality. To do so, we render the same video frame with and without exploiting coherence and measure the differences. We do this by finding pairs of closest surface points between ground truth and temporal coherence output and compute the average distance in millimeters. By using this metric error measure it is easier to interpret the evaluation data and it is possible to define a maximum deviation that is tolerable.

Figure 7.7 shows quality and speed measurements taken during an evaluation sequence of various user motions. The frames were grouped into all frames in Figure 7.7(a) and mostly dynamic frames and mostly static frames in Figure 7.7(b). In the dynamic frames, users entered the scene or moved their whole bodies. The static frames usually show users standing still and performing no intended motions. In this quality vs. speed scatter

plot, measurements in the lower left region have a good quality/speed tradeoff. The conventional IBVH algorithm is used as the ground truth and therefore has 0 mm error. The temporal coherence methods in this plot use the improved camera order and are executed for a wide range of quality/speed settings. It can be observed that the suggested methods outperform the conventional IBVH algorithm easily without quality loss (also see Figure 7.6(a)) and perform better than the previously used coherence method that only works on static surface patches. Figure 7.8 shows output images with a relatively high error level to illustrate the error metric. It can be observed that the camera association method is more prone to artifacts on the surface, while the user motion coherence method begins to fail when, for example, limbs move too quickly.

7.6 Summary

This chapter described a method to exploit temporal coherence in fully dynamic image-based visual hull rendering. We successfully showed that the surface point camera association has a relatively high stability over time and can be used to improve the performance of the original algorithm. We described how to set an upper limit to the expected user's body motions and how such a limit can be used to reduce the number of processed silhouette images. By using our motion detector, the motion limit can even be estimated before rendering.

We evaluated all suggested methods in terms of the achievable speed-up and illustrated the quality/speed tradeoff for a wide range of parameters. Our methods reduce the execution time by up to 50% when sub millimeter deviations in the visual hull are tolerable, which is usually the case for rendering applications. As a result, the achieved latency in our interactive free viewpoint rendering system can be lowered, especially for high resolutions. These results verify our temporal coherence hypothesis and extend our work on the temporal coherence of static visual hull patches.

Chapter 8

Image-based augmentations

Contents

With the free viewpoint rendering methods described in this thesis, we achieved a high quality image synthesis of humans moving in our multi-camera room at interactive frame rates. To create a virtual try-on application, the rendered images of users must be augmented with garments.

Obtaining the garment models is a key aspect. In computer graphics, these models are usually created manually by an artist and a 3D modeling tool. We wanted to avoid this labor intensive process and at the same time achieve a rendering quality that allows the garments to blend with the user seamlessly. We decided to tackle the problem by capturing garments worn by users in the multi-camera room that is also used for augmentation.

The goal of our image-based pipeline is interactive augmentation. Augmenting the visual hull of a user with virtual garments can not be achieved by conventional texture mapping. For example, a dress can be wider than the current clothes and therefore cover space that is not part of the original geometry. Moreover, a rigid assignment of texture coordinates is not enough because clothes can move relative to the user's body depending on the body pose. Finally, a garment's shape should adapt to the current user. Therefore, garment pixels should be allowed to move while satisfying a target objective that enforces fit and preserves the rough shape. This chapter describes an image-based augmentation pipeline that achieves these goals.

8.1 Virtual try-on through image-based rendering

Virtual try-on applications have become popular in recent years because they allow users to see themselves wearing different clothes without the effort of changing them physically. This helps users to quickly judge whether they like a garment or not, which in turn allows retail shops to sell more in less time. Web shops employ virtual try-on applications to enable users to better determine the size and fit of a garment [41, 75, 119]. Moreover, digital entertainment applications and games also aim at creating an image or a video of a user wearing different clothes. Digital content creation has become a major challenge in recent years as virtual worlds increase in size and complexity.

In our system, users standing inside a room equipped with cameras can see themselves on a large TV screen which shows them wearing different clothes. Users are allowed to move freely inside the room and can see themselves from arbitrary viewpoints. The system therefore needs to capture and render the user at interactive rates while augmenting his or her body with garments.

Some previous solutions are very simplistic: they display garment pictures as static overlays or retexture 2D surfaces. These approaches are clearly not capable of producing arbitrary viewing angles, which requires 3D information. Moreover, garments do not adapt their size and shape to the user due to their static nature. Other solutions address these issues by reconstructing 3D clothes models. These approaches can produce the desired output, but require motion capture to track the user's position over time. Motion capture is still a challenging task, and is likely to fail when limbs are not visible or in unusual positions.

We propose an approach where a user can be displayed wearing previously recorded garments. We achieve this by creating a garment database that stores the appearance of a worn garment over time. These recordings are not required to be from the same user and are performed using the same multi-camera device that is used for augmentation. The database can be queried by silhouette images. At runtime, the best fitting frame is picked for rendering. The image-based visual hull (IBVH) algorithm is used to render users and clothes from arbitrary viewing angles. To fit the transferred clothes to the user and adapt the garment's shape to its new body, rigid and non-rigid registration is performed. Both registration tasks are formulated as optimization problems.

Our approach is suitable for a wide range of clothing, and multiple garments can be combined. By using images of real garments instead of virtual models, a realistic rendering quality can be achieved. The complex appearance of clothes that comes from anisotropic materials and non-rigid deformations is already captured by the images and can therefore be reproduced with high performance. It does not require any physics simulation because the effect of gravity and cloth stretch is also present in the images. Due to image-based rendering, our method does not need an explicit 3D reconstruction of the garments or the user to produce views from all sides. For the same reason, it does not require manual 3D modeling by digital artists and it avoids motion capture. The whole process does not need manual interaction other than an initial garment segmentation.

To realize our approach, we introduce a garment augmentation pipeline [92, 94]. In this chapter we describe efficient methods for GPU-based silhouette matching by line-based and area-based silhouette sampling and evaluate their performance. Moreover, a novel method

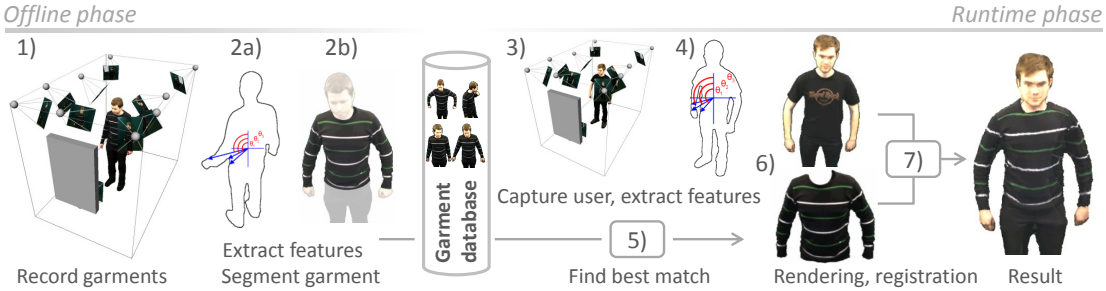


Figure 8.1: Overview of our pipeline. In the offline phase, garment image sequences are recorded and processed to create a database. At runtime, the database can be queried for a record that matches the current user's pose. The recorded images are used to augment the user with a garment.

to extend the pose space by combining multiple matches is introduced. We introduce an extended non-rigid registration formulation that adapts the garment shape by silhouette and depth data terms. The problem domain of the registration process is analyzed to derive an efficient non-linear conjugate gradient solver with proper initialization that exploits frame-to-frame coherence to improve quality and performance. Coherent visual quality over time is also enforced by a weighted infinite response filter that is capable of suppressing artifacts inherent to a discrete pose space. We evaluated all improvements for their impact on the visual quality by comparing them to ground truth images.

8.2 The augmentation process

Similar to previous work [201], our clothes augmentation process has an offline phase for recording garments and an online phase where users can be augmented. The stages of the recording process (see Figure 8.1) are:

1. A user wears a garment which should be used for future augmentations. He or she enters the dressing room and performs a short series of different poses while being recorded.

2. Garments are segmented and features are extracted from the recorded video streams. Results are stored in a garment database.

This phase can be controlled: it is possible to recapture the scene when incomplete, or switch segmentation strategies. From now on, users who enter the dressing room can be augmented with previously recorded garments. We call this the runtime phase:

3. Users can move freely inside the room while being captured.

4. Features from the captured images are extracted.

5. The best fitting pose from the garment database is selected by comparing the extracted features.

6. The selected pose of the desired garment and the captured user are rendered from the same viewpoint using image-based rendering.

7. Small pose mismatches are compensated for by rigid and non-rigid registration.

This process results in a composite mirror image, showing the user wearing a different item or items of clothing. A pipeline that enables such an approach has several key stages: first, the garment database needs to be created such that it can be accessed very efficiently at runtime (Section 8.3). Second, at runtime the garment and the user need to be aligned roughly before rendering (Section 8.4.1). Third, quick image-based rendering is required (Section 8.4.2). Finally, remaining mismatches between garment and user need to be resolved on a per-pixel level (Section 8.4.3).

8.3 Offline: garment database construction

A user puts on one or multiple garments which should be transferred to other users. The other clothing should be selected to allow for easy segmentation. We call this user the *model-user* to emphasize the difference to an end-user. We assume that the model-user can be instructed to perform all the desired motions, whereas the end-user just consumes the product as he or she wishes.

The model-user enters the dressing room and performs a series of different poses while being recorded. Recordings consist of synchronized videos from the ten cameras. Each garment database contains a single recorded sequence and therefore a single piece of clothing. When multiple garments or different sizes should be provided, each item needs its own recording and database.

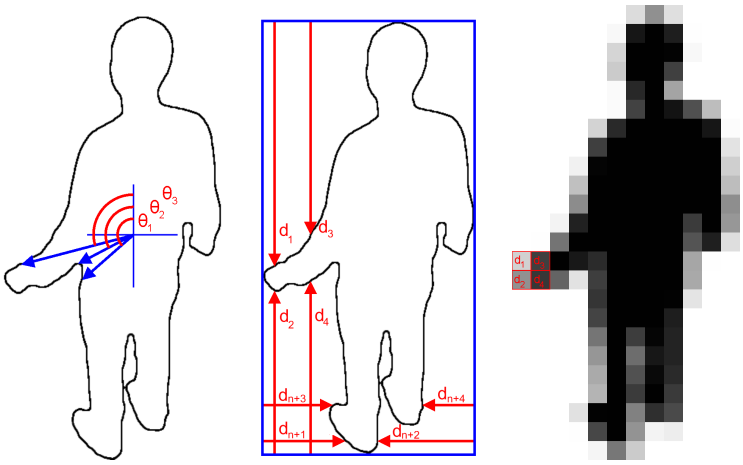
8.3.1 Preprocessing

This stage takes the camera videos as input. Videos are recorded at 15 Hz, so usually the user's relative motions between two successive frames are small. Similar looking frames are of limited use for the later stages because minor pose offsets are compensated for by registration. To reduce the database size, many frames can therefore be skipped. Currently, we only pick every second frame to be included in the garment database. Moreover, before insertion every new frame is checked for silhouette pixel overlap against the frames in the database to skip frames that are too similar.

Next, each camera image that is not skipped is segmented separately. First, a previously recorded static background image is subtracted to remove the background. Then, the garment is segmented: we use color keying and a graph cut tool. Defining the color key or a graph-cut seed is the only manual interaction that is needed to create the garment database from an image sequence. At runtime after the registration process, the segmented pixels are culled to remove the unwanted parts from the output.

8.3.2 Extracting features

At runtime, the frame which contains the model-user's pose that is most similar to the current user's pose has to be found. This can be achieved by matching colors, silhouettes



(a) (b) (c)

Figure 8.2: Silhouette features comparison. (a) the distance between center of mass and silhouette edge at regularly spaced angles may provide insufficient sampling of the arms. (b) axis aligned sampling of the bounding box alleviates the problem. (c) area-based sampling computes fill-rates of grid cells and therefore also represents the silhouette's interior.

or higher level features like motion capture positions. We did not find color information to be particularly useful because garments of the model-user and the current user are likely to be entirely different. Motion capture may fail for cases where limbs are close to the body or form loops. We therefore focus on silhouettes, which are well defined as long as the user stays inside the viewing frustums of the cameras. To find the best fitting pose based on silhouettes, a metric to calculate silhouette similarity is required. When silhouette similarity or difference can be measured, standard algorithms can be used to search the resulting space.

Our approach extends the work of Ehara et al. [43]. We work on more than one silhouette image. This additional input data is required to obtain a descriptive feature vector even when main features of the user, like his or her arms and legs, are located in front of the body and thus not visible in one or more silhouette images. During our evaluation we found that four cameras spaced around the user were usually sufficient to see the user's limbs for all body orientations. Figure 8.3 shows the spatial arrangement of the four cameras in our setup.

Required invariances

It is beneficial for a matching algorithm to be invariant to at least the scale and translation of the user's pose. For example, it is very likely that the model-user has a different body size than the end-user. Moreover, it is common that users do not stand on exactly the same spot in the room. Rotational invariance is not desired for our system because we use

a non-uniform distribution of cameras in the cabin. We therefore do not want to match poses that are oriented differently to avoid undersampling of important body parts.

To extract features, we compared three different silhouette sampling approaches. The first approach uses a radial pattern.

Radial line sampling

To extract features, the center of mass of each silhouette image is computed [92]. From the center of mass, 360 regularly spaced directions are sampled to find the closest and second closest silhouette exit edges (see Figure 8.2(a)). The closest edges usually describe the central body shape. The second closest edges describe the location of arms and legs. The distance between center of mass and edge is stored. When there is no second closest edge its distance is set to the distance of the closest edge. All distances are normalized per image to be independent of scale. Invariance to translation is given by relating all distances to the center of mass.

Axis-aligned line sampling

The second approach uses an axis-aligned sampling pattern. We chose this sampling pattern over a radial pattern because silhouettes of humans frequently have the property that arms and legs are aligned parallel to the radial sampling lines. Figure 8.2(a) illustrates this problem. Such an alignment causes an underrepresentation of arms and legs in the feature space, which makes it hard to match these body parts correctly.

First, the minimum bounding rectangle of each silhouette image is computed. Along the x and y intervals of the rectangle, regularly spaced lines are sampled (see Figure 8.2(b)). Along each line, the minimum distances from the ends of the line to the silhouette are stored as a vector. The concatenated description vectors of all four silhouette images form the silhouette matching space. Similar to previous approaches [201], all distances are normalized to unit length to allow matching across different scales. Invariance to translation is given because all measurements are taken at positions relative to the bounding box.

Area-based sampling

The third approach samples the silhouettes densely. The minimum bounding rectangle of each silhouette is split into grid cells, which are sampled to obtain the fill rate of each cell (see Figure 8.2(c)). By aligning the sampling grid with the bounding box, this method is invariant to translation and scale. This method is more robust against a noisy segmentation than line sampling because fill rates vary rather smoothly at silhouette edges and also the interior of the silhouette is explicitly contained in the feature vector. However because the feature vector's components only contain overlap information instead of location information, the feature vectors are less descriptive than their line sampled counterparts. As a consequence, the search space can not be reduced as effectively as with line sampling methods (Figure 8.10(b)).

Section 8.6 evaluates all three sampling strategies with the result that area-based and axis-aligned sampling performed better than radial in terms of matching precision.

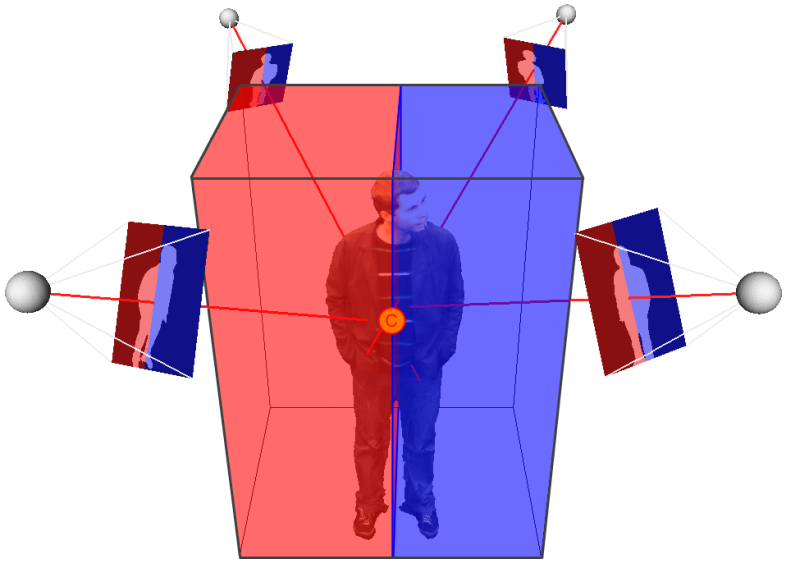


Figure 8.3: The pose space is split at the X coordinate of the projected center of mass in each silhouette image. This approximates a 3D splitting plane that would be slower to compute.

However, area-based sampling requires more memory access operations and a larger search space.

8.3.3 Reducing the dimensionality

Depending on the sampling resolution and pattern, the resulting feature space can have several thousand dimensions, which describe the pose of the model-user at one moment in time. All frames of the recorded videos are processed in that manner, resulting in a large data matrix of pose descriptions. Using principle component analysis (PCA), we were able to reduce this to around 50 dimensions for line sampling, or 200 dimensions for area sampling without losing descriptiveness. Both vector spaces are small enough to query the database at runtime by a simple search. Other silhouette matching approaches [201] create data structures to improve search performance. In our system, however, exhaustive searching is sufficiently fast and only of minor importance to the overall performance (see Figure 8.15).

8.3.4 Extending the pose space

The garment database may not always contain a satisfying pose. This is the case when the model-user forgets to cover certain poses, or when the temporal resolution of the recording is not sufficient. To alleviate this problem, it is possible to extend the pose space at runtime by combining several poses into a single output. See Figures 8.3 and 8.4 for an illustration of the process.

To achieve this, we propose the following procedure. First, the best fitting pose is queried as described above. Then the matching error is examined. If it does not exceed

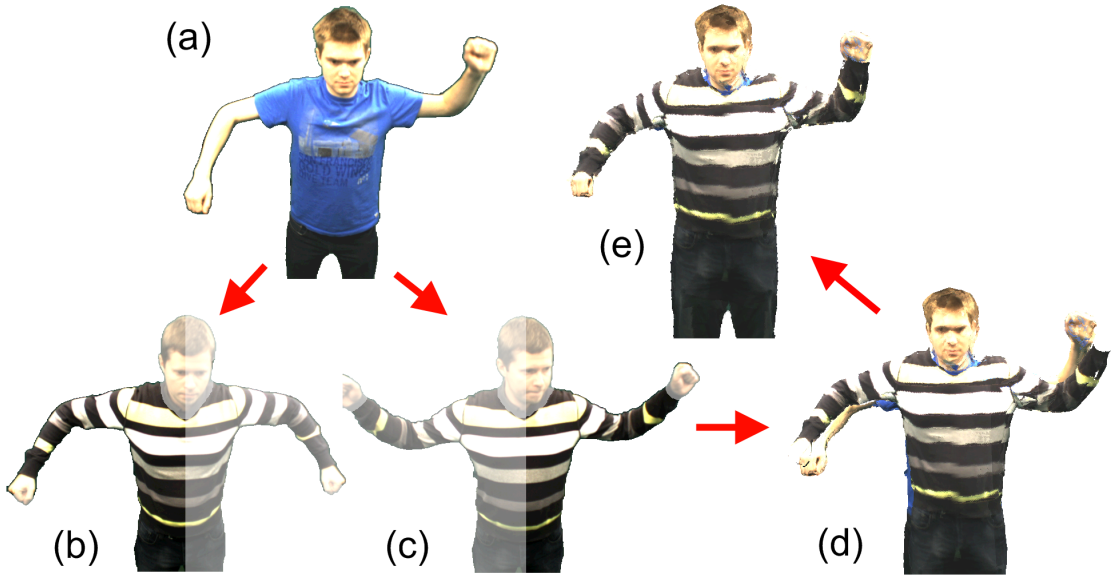


Figure 8.4: Extending the pose space by combining two poses into a new pose. (a) shows the current user's pose. (b) and (c) show the best fitting poses for each image half. The result is a rendered image of the two halves without (d) and with non-rigid registration (e).

a certain threshold then the match is considered good enough and only a single output is generated. However, if the threshold is exceeded, we split the silhouette images that are used for matching vertically at the X screen coordinate of the center of the user. Each half is then transformed to its PCA subspace and matched against the database separately. Therefore, during the offline phase the principle components of the garment database need to be computed for each half separately too. This method is an approximation of splitting the space in 3D, which to compute precisely would require more computations, in particular the creation of depth maps for each silhouette image. Images can also be split horizontally to allow for different poses in the upper/lower body half. This approximation assumes that the body segments are clearly distinguishable in the camera images that are used for matching. In practice this means that cameras mounted at the sides or straight above the user can not be used for pose matching.

Split segments are joined when they contain the same best match in order to decrease rendering time when possible. Finally, a blending kernel is launched that stitches the segments and employs a linear blending filter to cross-fade between them. To allow for free viewpoint motion, the blending operation can not be performed in image space because when viewing from the sides the occlusion needs to be resolved correctly. Instead, the visual hull points of all split segments are clipped against the axis-aligned planes that are assumed by the matching. Figure 8.4 shows an extended pose space rendering result. This method can improve the output quality considerably, which can be measured. See Section 8.6 for evaluation data.

The output of this stage is the garment database. Each entry consists of features extracted for later matching, background-subtracted color images and segmentation information.

8.4 At runtime: clothes augmentation

Once one or more garment databases have been created, users can enter the dressing room and watch themselves wearing different clothes on the display in front of them.

First, the same features as in the offline phase are extracted from the current camera images. These features are transformed to PCA space by applying the precomputed transformation matrix from the offline stage. The result is used to find the garment database entry where the model-user's pose is closest to the pose of the current user. We use a simple Euclidean distance measure to find the closest feature vector during the search. The camera images associated with this entry show the best match for the desired garment. Most likely the found images show the model-user standing in a slightly different position than the current user. To compensate for this offset, a rigid registration is performed.

8.4.1 Rigid registration

The goal of this stage is to find a translation vector and a scale factor that align the rendering output of the user and the garment model. At this stage no rendering is performed yet, which means that no 3D data is available to determine the transformation by standard methods. Instead, we approximate the translation and scale by utilizing the silhouette images and the projection matrices of the corresponding cameras.

To achieve this, we need two or more cameras that see the user entirely. While our system utilizes 10 cameras in total, four of them are mounted in the upper corners of the cabin (see Figure 8.5). Due to their distance to the user, the frustums of these cameras is sufficiently large to see the whole user and can be used for registration.

First, the 2D center of mass is extracted from each of the silhouette images of the cameras. By transforming these points with their corresponding inverted projection matrices we obtain a 3D ray for each camera. Such a ray emanates from a camera center and runs through the 2D center of mass on the corresponding image plane. These rays do not necessarily intersect, so we compute the point with the least squared distance to the rays. The result is an approximation of the 3D center of mass, which otherwise would require a 3D reconstruction to determine. To compute the translation, we subtract the 3D center of mass points of the garment model and the current user's model.

The 3D center of mass does not necessarily lie on the rays that are cast through the 2D center of masses due to the perspective projection of the cameras. However, garment and user are subject to a similar error and as a result the rigid registration offsets were satisfying during our experiments.

A very similar operation is performed with the topmost 2D point of each silhouette image. We assume that the 3D intersection of the corresponding rays is a good approximation of the top of the head of the user. The Z-coordinate of this 3D point describes the height above the floor. To determine scale, we simply compare the body heights of

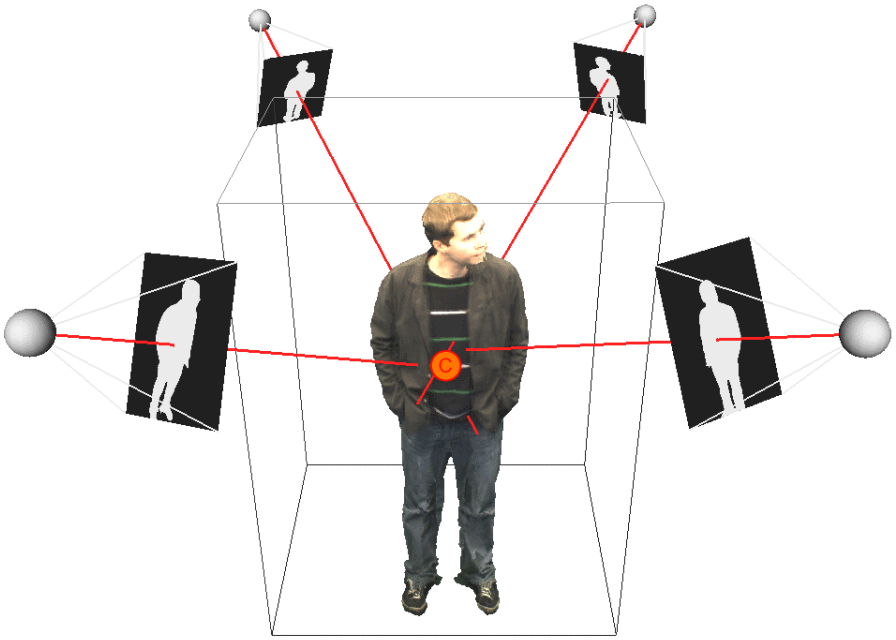


Figure 8.5: This illustration shows how the 3D center of mass point C is estimated. The 2D centers of mass from the silhouette images are extruded and a 3D intersection is approximated.

the user and the model-user. This of course only compensates for different body heights. We leave the other dimensions to be compensated by the non-rigid registration, which can also handle different body shapes more precisely.

During evaluation, these approximations proved to be sufficiently stable. With the computed translation and scale factors, the current viewing matrix is updated. This way, the subsequent garment rendering pass produces its output at the desired location in 3D space.

We do not compensate for rotation because the matching phase is only invariant to translation and scale. This means that the matching phase already found a correctly rotated pose at this stage. This behavior is particularly useful because it is hard to determine the rotational registration from silhouette images alone.

8.4.2 Image-based rendering

In this phase, the matched garment dataset and the current user are rendered. Both rendering tasks need to quickly generate an output image from a set of camera images with silhouette information.

We suggest the IBVH algorithm (Chapter 3) to compute depth maps of novel viewpoints directly from the segmented camera images. It bypasses the computation of an explicit representation, such as a voxel grid or a mesh. Extracting such a representation and rendering it are two separate tasks which contain some redundancy. Therefore, it is

(a) (b)

Figure 8.6: (a) illustrates how the non-rigid registration adapts the garment to the user's shape and pose. (b) shows the influence of the depth data term on the non-rigid registration: the garment adapts to the user's hand.

beneficial to the overall performance to directly derive an output image from the camera images.

After the depth map is extracted, it has to be textured using the camera images. We use a view-dependent texture mapping [39] scheme that utilizes the viewing direction of the cameras and surface normals to find suitable color samples for every output pixel.

The garment and the user are rendered to different buffers from the same viewpoint. These intermediate buffers differ from the desired final output: parts of the model-user are still visible. For example, the head, hands and all items of clothing that should not be augmented. To remove these parts later on, the segmentation labels that were determined in the offline phase are used during rendering to write the alpha channel. After non-rigid registration the alpha values are used to remove any remaining parts of the model-user from the output.

8.4.3 Non-rigid registration of the output images

The last remaining difference to the desired output image is usually a small pose and shape inconsistency. Large offsets and differences in scale have already been compensated by the rigid registration. But the garment database does not cover all possible poses at an infinitely dense resolution: it only contains discrete samples of the pose space. Moreover, the body shapes of the current user and the model-user do not necessarily match. The non-rigid registration is used to compensate for this minor lack of overlap. See Figure 8.6 for an illustration.

We identified two main methods to achieve a better overlap: by interpolating adjacent poses, or by moving the pixels of one adjacent pose. Interpolating is rather difficult because silhouette-based pose spaces are high-dimensional and therefore many adjacent datasets

have to be found and interpolated. Additionally, a high quality interpolation requires corresponding features in color or silhouette space.

We therefore apply an optimization procedure that translates the pixels of the rendered garment to account for small deviations of pose. In addition to reducing pose inconsistencies, the optimization allows the garment shape to adapt to the current user's body shape.

The domain of the problem can be formulated as an energy function that should be minimized. We use pixel locations as the data elements to allow the silhouette to shrink or grow and to retain the spatial arrangement of pixels. Previous approaches align garment models in 3D [128], which is accurate but slow due to the larger problem domain. For producing a correct appearance, however, it is sufficient to align the rendered images in 2D. This reduces the dimensionality of the optimization problem considerably. Non-rigid deformations are often formulated as sets of linear equations [14]. However, linear approaches require corresponding features to guide the deformation. For aligning a garment with the user, however, it may be hard to find such correspondences because both shape and appearance can be different. We therefore formulate the problem as a non-linear energy function, which allows us to directly specify the desired objective: overlap between garment and user.

The energy function E_{total} that is minimized is the sum of all per-pixel energies $E(x, y)$. Every pixel (x_{ij}, y_{ij}) of the garment silhouette is a particle and can move freely.

$$E_{total} = \sum_i \sum_j E(x_{ij}, y_{ij}) \tag{8.1}$$

The optimization procedure is initialized with the garment pixel's locations:

$$\begin{pmatrix} x_{ij} \\ y_{ij} \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix} \tag{8.2}$$

Energies consist of a data term and a neighbor-term for regularization.

$$E(x_{ij}, y_{ij}) = D(x_{ij}, y_{ij}) + \alpha \cdot N(x_{ij}, y_{ij}) \tag{8.3}$$

$$\begin{aligned} D(x_{ij}, y_{ij}) = (I_{garment}(i, j) - I_{user}(x_{ij}, y_{ij}))^2 + \\ \gamma \cdot (M_{garment}(i, j) - M_{user}(x_{ij}, y_{ij}))^2 \end{aligned} \tag{8.4}$$

$$I_{garment}(i, j) = \begin{cases} 1, & \text{if } (i, j) \text{ belongs to garment} \\ 0, & \text{otherwise} \end{cases} \tag{8.5}$$

$$I_{user}(x, y) = \begin{cases} 1, & \text{if } (x, y) \text{ belongs to user} \\ 0, & \text{otherwise} \end{cases} \tag{8.6}$$

The data term D of Equation 8.3 describes the overlap between garment and user by computing the L2-norm between silhouette and depth value differences. The I-terms are 1 for foreground pixels and 0 for background and thus describe a silhouette. For

performance reasons the buffers holding these terms are convolved with a Gauss kernel before optimization. The M-terms denote the depth maps of the garment and the rendered user. During rendering the depth map values are normed to unit range according to the near- and far clipping planes. These plane settings should be equal when rendering the user and the garment to avoid unwanted distortions.

During optimization, the I-terms push garment pixels towards the user's body and thus align the shape outlines. In addition, the M-terms move pixels within the shapes to align regions with similar depth patterns. For example, misplaced sleeves are moved towards the user's arms even when the user holds his arms in front of his body (see Figure 8.6 for an illustration).

The regularization term N of Equation 8.3 tries to retain the spatial arrangement of the cloth. We use the direct neighbors of each garment pixel as well as neighbors that are further away, but using a smaller weight. One such neighbor is N_{uv}.

$$N_{uv}(x_{ij}, y_{ij}) = \delta \cdot C_{uv}(x_{ij}, y_{ij}) + \cdot S_{uv}(x_{ij}, y_{ij}) \tag{8.7}$$

$$C_{uv}(x_{ij}, y_{ij}) = (min(\sqrt{(x_{ij} - x_{uv})^2 + (y_{ij} - y_{uv})^2}, d) - d)^2 \tag{8.8}$$

$$\text{with } d = \sqrt{(u - i)^2 + (v - j)^2}$$

$$S_{uv}(x_{ij}, y_{ij}) = (u' - x_{uv})^2 + (v' - y_{uv})^2 \tag{8.9}$$

$$\text{with } u' = x_{ij} + (u - i) \text{ and } v' = y_{ij} + (v - j)$$

x_{uv} and y_{uv} are the new x and y coordinates of the neighbor, and u' and v' are its initial coordinates relative to x_{ij} and y_{ij}. The compression term C_{uv} tries to keep the neighbor at least at its initial distance d, while the stretch term S_{uv} tries to keep it at its relative position. This regularization is important to retain features of the garment, like stripes and printed logos.

The factor α weighs the regularization versus the data term. It therefore can be seen as the *stiffness* of the garment. Moreover, it indirectly controls the convergence speed of the optimization: stiffer garments converge faster because the influence of the data terms propagate quicker. However, when the stiffness is set too high, the performance decreases again. For such cases it is more desirable to use a larger neighborhood for regularization.

γ weighs the depth data- against the silhouette term. To achieve a registration effect within the silhouette like in Figure 8.6(b), the depth term needs to be emphasized.

δ scales the compression- against the stretch penalty. Both are good for regularization, but there are slight differences that justify using both at the same time. The compression term is particularly useful for reducing pixel collisions when the optimization result is converted back to discrete pixel locations. The maximum penalty for compression is bounded, so we weighed it higher than the stretch term. The stretch term on the other hand is more suitable for retaining the spatial arrangement of a neighborhood because it also penalizes rotations.

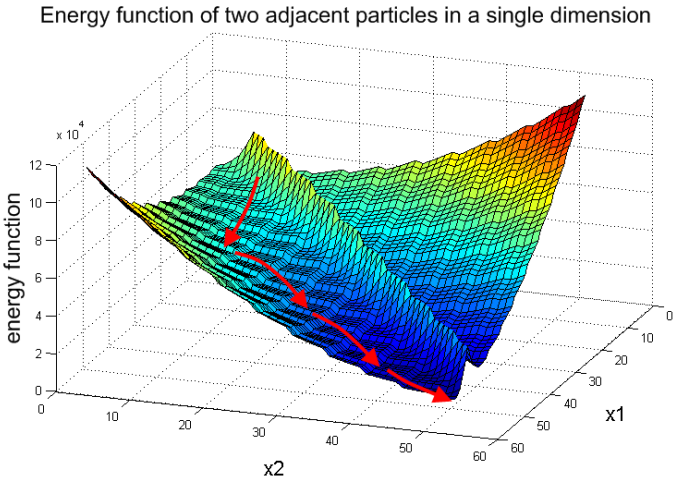


Figure 8.7: Plot of the energy function E_{total}. For illustration purposes, the problem is reduced to two particles that can move along a single dimension in space. The shallow valley that is marked with red arrows indicates a quick path to the minimum that a conjugate gradient method can take.

The solver

We assume that the pose differences between garment and user are small, which makes the structure of the problem not prone to local minima. The energy function is minimized hierarchically, but not in a globally optimal way. We use the nonlinear conjugate gradient (NCG) method for optimization. We observed that the proposed energy function contains many narrow valleys formed by opposing gradients of the data term and the regularization term. Figure 8.7 illustrates the error function in a space with reduced dimensionality: only two particles are considered, and the image space is reduced to a single dimension. The path to the minimum is formed by a diagonal ridge that is caused by the $C_{uv}(x, y)$ term, the quadratic slopes (red) caused by the $S_{uv}(x, y)$ term and the data term that decreases in positive x-direction. For such problems, NCG converges faster than gradient descent because it can follow the valleys. Our solver computes the β factor according to Polak−Ribière and uses an automatic reset mechanism $\beta = max(0, \beta^{PR})$ [171]. The output of Figure 8.6, for example, can be computed by 50 iterations of gradient descent, or 15 iterations of the nonlinear conjugate gradient method with the same fixed step length. Just like gradient descent, it is well parallelizable and has a low memory footprint. The additional cost of computing the β factor every iteration is negligible compared to this speed up. We assume the derivative of the *min*-function in Equation 8.8 to be 0 where it is not differentiable.

The registration starts by converting the IBVH-rendered color image of the current user to a binary image that only shows the user's silhouette. Then, low resolution copies of the silhouette and the IBVH-rendered depth map are computed and both are smoothed by convolution with a Gauss kernel. The low resolution level usually has only $\frac{1}{16}$th of the output resolution. For both the low- and the high resolution levels, the x and y gradients are precomputed because these terms are static during the optimization. The optimization

procedure itself consists of a repeated execution of three steps: computing the derivatives at each garment pixel location in x and y direction, computing β^{PR} and the conjugate gradient and finally updating the pixel locations. We use a fixed step length instead of a line search, which we found empirically to reduce execution time. First, 70 iterations are executed on the low resolution buffers. Then, the optimization state is magnified to full resolution by linear interpolation. Now it is possible to compute several iterations at the full resolution, but we observed that the difference is usually not perceivable.

The vector between the new, optimized pixel locations and their initial positions yields an offset buffer that describes how garment pixels need to move in order to maximize overlap with the user. This buffer is morphologically closed and smoothed by a 20x20 box filter to suppress noise. Finally, the garment pixels are moved according to the offset buffer. All of these steps are implemented as CUDA kernels, which allows such a high number of operations to be performed for every frame. Section 8.6 evaluates the positive impact on the output quality of the non-rigid registration.

Initialization and temporal coherence

The described registration process is very sensitive to the current user's output shape, especially when rather large step lengths are chosen for fast convergence. This can lead to a perceivable swinging and jumping of the garment image over time even when the same database frame is used. To remove this behavior, temporal coherence can be enforced for successive output images that are computed from the same database frame. An effective way to achieve this goal is to initialize the optimization procedure with the last frame's state.

This way, only the user's relative motion between the last frame and the current frame needs to be compensated for by the registration. As a result, pixels of body parts that have not moved are already in a converged state and therefore stay at a coherent location over time. Moreover, the last frame's state is usually closer to the desired result than a newly initialized state and therefore the optimization converges faster.

However, the last frame's state can not be reused by simply copying because between the frames the viewpoint might have changed. Therefore, the state needs to be transformed between the different coordinate systems. This is called image warping [184]. To reuse the last frame's state, the initial state defined in Equation 8.2 is now translated:

$$\begin{pmatrix} x_{ij} \\ y_{ij} \end{pmatrix} = \begin{pmatrix} i + d_i \\ j + d_j \end{pmatrix} \tag{8.10}$$

The translation offsets d_i, d_j are transferred from the last frame $t-1$:

$$\begin{pmatrix} d_i \\ d_j \end{pmatrix} = \begin{pmatrix} x_{i'_{t-1}j'_{t-1}} - i'_{t-1} \\ y_{i'_{t-1}j'_{t-1}} - j'_{t-1} \end{pmatrix} \tag{8.11}$$

$$\text{with } i'_{t-1} = i_{t-1}/h_{t-1} \text{ and } j'_{t-1} = j_{t-1}/h_{t-1}$$

The location in the last frame from which the offsets are computed is:

Figure 8.8: This figure illustrates the weight map that is used for cross-fading previous outputs while arms and legs of the user moved. Red indicates motion.

$$\begin{pmatrix} i_{t-1} \\ j_{t-1} \\ . \\ h_{t-1} \end{pmatrix} = MVP_t^{-1} \cdot MVP_{t-1} \cdot \begin{pmatrix} i \\ j \\ M_{garment}(i,j) \\ 1 \end{pmatrix} \tag{8.12}$$

with MVP_t^{-1} being the inverse of the current modelview- and projection (MVP) matrix, and MVP_{t-1} being the last frame's MVP matrix.

As a result, the garment does not swing or jump between successive frames as long as the same database frame is used. Moreover, it allows the non-rigid registration to compensate for larger pose mismatches with the same number of iterations, or, to require less iterations for the same distance because the optimization converges across frames.

Cross-fading during frame transitions

For the other case, where two successive output images are created from different garment database frames, there may be a perceivable *jump* between the frames. This is mostly due to garment deformations that are too small for the registration algorithms to capture. To alleviate this problem, we cross-fade the output image with the previous output, thus creating an infinite impulse response (IIR) filter. The lack of coherence is particularly visible in image regions that should stay static from one image to the next. Such regions usually show body parts of the user that do not move. Other image regions that show moving body parts can not be cross-faded without producing ghosting artifacts. Therefore, the IIR filter needs to be weighted by a map that describes the user's relative motion between two successive output frames. We compute each pixel's value of this weight map as the normed average length of the previous and current non-rigid registration offsets (see Figure 8.8). This effectively suppresses the filter response at image regions that are subject to strong user motion. To reuse the previous frame's output color and registration offsets

correctly even during viewpoint motion, image warping needs to be applied as shown in Equation 8.12. As a result, garment frame transitions triggered by user motion are less obvious.

Detecting convergence

The process as described above uses a fixed number of iterations. However, a satisfying state may be reached earlier. Usually convergence is detected by checking the gradient length or evaluating the error function every n-th iteration. However, in our scenario the error function is rather noisy and flat due to the strong neighbor terms. Therefore, detecting convergence is more robust by only evaluating the data term. The algorithm stops iterating when the summed data term values do not improve for a certain number of iterations.

Detecting convergence is particularly useful in combination with reusing the last frame's state for initialization: on top of frame coherence, the system's performance automatically increases when pose mismatches are small while still maintaining the ability to cover larger mismatches when necessary. In combination with the coherent initialization, we observed that the execution time of the optimization process can drop to 60% when the previous garment frame is reused.

Loose clothing

Loosely fitting garments break the assumption of silhouette similarity between the garment worn by the model-user and the current user. The non-rigid registration may have problems with loose clothing: the algorithm tries to move clothes pixels towards the target silhouette. We addressed this issue by increasing the stiffness term of the energy function. This effectively preserves the cloth structure even for dresses or coats, see Chapter 9 for examples. However, the increased stiffness causes unstable optimization results for larger step lengths, so these need to be reduced. As a consequence, to cover the same amount of pose mismatch, the number of iterations needs to be increased. This approach is not a physically correct simulation of clothes, but succeeds for most cases.

The garment database lookup also assumes silhouette similarity between garment and user. For example, a long skirt results in large silhouette-matching errors in the leg regions. During our experiments with a long dress and a lab coat this did not break the matching algorithm because all frames of the database suffered from the same error. Thus, the relative matching order between frames stayed valid. However, we can not generally guarantee that silhouette-based matching works for all existing garments.

8.4.4 Composing and display

Parts of the model-user are still visible in the garment buffer, but with alpha values that indicate which pixels have to be removed. These regions were required as a good optimization target during the non-rigid registration procedure. After optimization the unwanted pixels are removed.

In a final step, the garment buffer and the current user's buffer are composed. Depth buffering can resolve the overlap. However, this can lead to unwanted effects when the

Figure 8.9: A screenshot that shows how multiple overlapping garments can be combined.

user's body shape is wider than the garment. In such cases, the user's body is visible where it should not be. We allow for clothes to float on top of the user to avoid such wrong or noisy occlusions. As a result, the virtual garment is visible except in regions where the model user was removed. In these regions, the current user's body replaces the model user. This occlusion effect is correct because the model-user occluded the same regions.

When multiple garments are augmented, the items are consecutively composed with the previous result. Figure 8.9 illustrates how a shirt and an open jacket can be combined despite the overlap. In this case, the composing starts by merging the user and the shirt. The result is used as the new image of the user and is composed with the jacket. The graphics API is used to display the result and enables a combination with conventionally rendered objects.

8.5 Hardware and implementation

The image-based augmentation pipeline uses the free viewpoint rendering hardware described in the chapters above.

All stages are wrapped into Coin3D scene graph nodes that allow the components to be easily combined. The runtime stages are implemented as Nvidia CUDA kernels, which allows us to formulate the computations without any graphics pipeline overhead. Results are written to an OpenGL frame buffer, which allows our results to be augmented by conventionally rendered objects and finally be displayed on the screen.

Storage of the garment database

For every database entry, all ten camera images are written to the hard disk using the PNG format with alpha channel for segmentation. The extracted features are also stored on the hard disk. Before the runtime phase starts, the images are copied onto a RAM-Disk, which maps a file handle to chunks of main memory. This method speeds up the file access and therefore allows us to efficiently access the garment database every frame in a convenient way. Moreover, our system is capable of caching parts of the garment database on the GPU for further speedup.

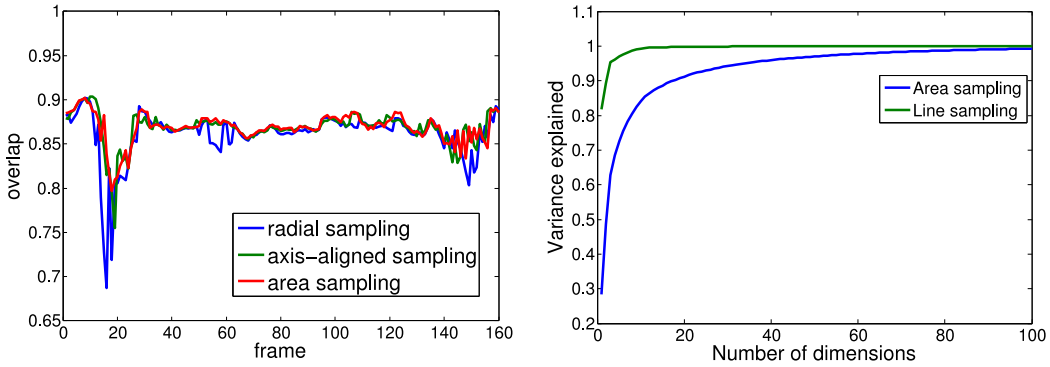
8.6 Results

We evaluated our methods both in terms of visual quality and performance. First, the silhouette matching algorithm's precision is tested to find the best sampling pattern. Next, the visual output of a garment transfer is compared to recorded ground truth. This test compares silhouette matching to its strongest competitor: motion capture. Moreover, it shows the relative quality improvement of the non-rigid registration. Finally, performance data is shown. The output resolution was fixed to 1000×800 pixels for all tests and the viewpoint set such that the rendered user is maximized on the screen. The chosen resolution is higher than the camera resolutions because some cameras are closer to the user than the virtual viewpoint.

8.6.1 Sampling patterns for silhouette matching

In our first test, we evaluated different silhouette features in terms of the resulting matching quality. Features are extracted using radial, axis-aligned and area-based sampling. For a recorded sweater sequence, we measured the pixel overlap between the current user and the garment in each output image from a frontal viewpoint. The pixel overlap factor is computed as the fraction of user pixels that are covered with garment pixels. Figure 8.10(a) shows the results. Area-based sampling provides a 0.8% better average overlap than radial sampling. This rather small percentage corresponds to an approximate average of 900 pixels per frame, which is a clearly visible difference. However, while quality is better with area-based sampling, the feature vectors are less descriptive.

The dimensionality reducing effect of the PCA is therefore less effective for area-based sampling, which requires the subspace in which silhouettes are matched to have more dimensions than with the line sampling (radial or axis-aligned) approaches. Figure 8.10(b) shows that area-based sampling requires 100 search dimensions to become equally descriptive. During our experiments, we found that the number of dimensions that are used for searching must account for almost all (at least 99.9%) of the variance in the data to achieve a satisfying matching quality. To guarantee this we use 50 dimensions for the two line sampling approaches and 200 dimensions for area sampling. Due to its lower requirements, we favor the axis-aligned line sampling approach. It performs almost as well as area-based sampling (less than 0.1% average difference in this evaluation) at lower computational cost and therefore is a noticable improvement.



(a) (b)

Figure 8.10: Different feature sampling schemes for silhouette-based matching. In (a) quality is evaluated as the factor of overlapping pixels in the output images of a recorded sequence. (b) compares line- and area-based sampling strategies in terms of how much variance in the data can be explained with a certain number of dimensions.

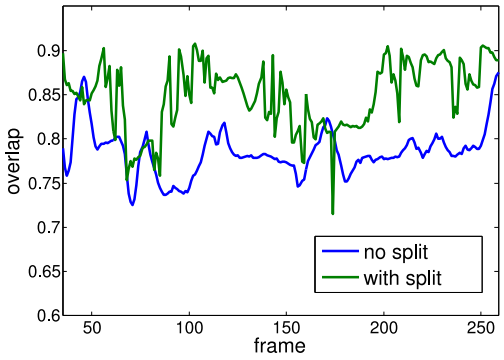
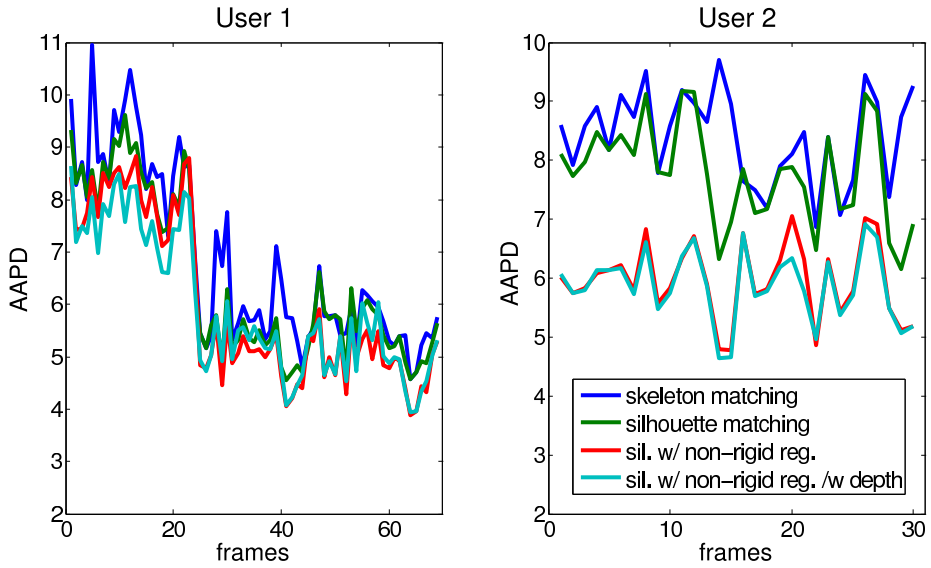


Figure 8.11: This figure shows the positive effect of extending the pose space on the pixel overlap factor for the dataset of Figure 8.4.

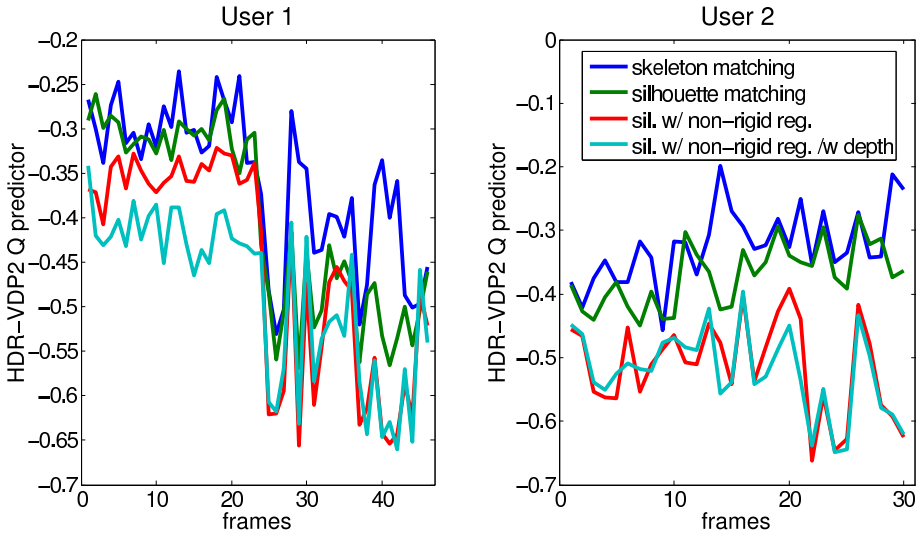
8.6.2 Pose space extension

To evaluate the usefulness of extending the pose space by splitting the silhouette image domains, we applied the same overlap factor metric as in the first test. For recorded garment data that has a sufficient collection of poses, this method of course does not improve the result. However, for the dataset of Figure 8.4, where poses are missing, we measured an average overlap improvement of 8.25%. This corresponds to an approximate average of 9000 pixels per frame, which is a considerable improvement. Figure 8.11 shows the overlap factor for the whole sequence. This proves the viability of the pose space extension method. However, there is an impact on the performance that amounts to an additional garment (see Figure 8.15).



(a) (b)

Figure 8.12: Quality evaluation of our approach for two users. A set of frames created by our image-based augmentation pipeline are compared to ground truth by means of average absolute pixel differences (AAPD). Small values are better.



(a) (b)

Figure 8.13: Same quality evaluation as in Figure 8.12 but using the HDR-VDP2 quality metric. Small values are better.

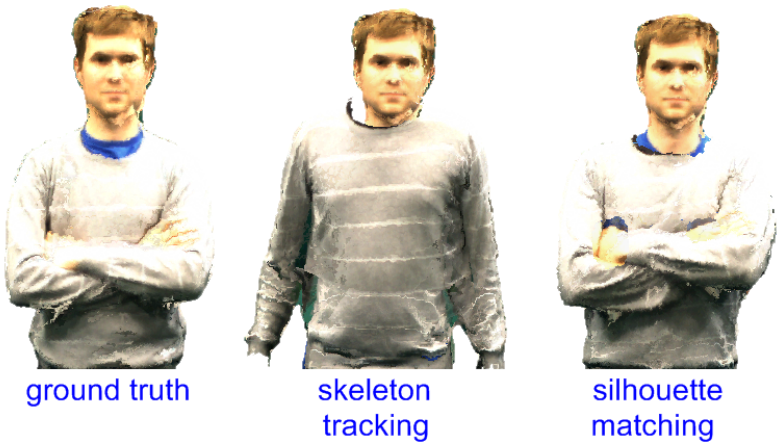


Figure 8.14: This figure shows a frame from the evaluation data set of user 1 of Figure 8.12, where skeleton- and silhouette-based matching strongly differ.

8.6.3 Overall quality and non-rigid registration

To assess the quality of the overall clothes augmentation system, we first created two ground truth datasets. We recorded two users with different body sizes: one male, 1.86 meters tall (user 1), and one female, 1.62 meters tall (user 2). Both performed the same body poses, once with and once without the garment (a striped sweater) that is later subject to image-based clothes transfer. We manually labeled a set of frames from these videos, where body poses match exactly. By using these correspondences, we have reference images for a set of frames, which are used during the evaluation. We also recorded the model-user who was 1.80 meters tall and wore the same sweater to generate the garment database.

Figures 8.12 and 8.13 show the result of the evaluation for both users. We compared the rendered garment frames to the corresponding ground truth frames for two viewpoints: a frontal view and a side view. Two quality metrics are used: the average absolute pixel difference and the HDR-VDP2 quality predictor [149] that also accounts for the human visual system and perception. We first compared silhouette-based matching to its major alternative: motion capture. For motion capture, we used the OpenNI skeleton tracking API, which can be used to track poses from depth sensors. To make the approaches comparable, we compute suitable depth maps from the visual hull. The motion capture results in the form of joint positions were normalized and matched using Euclidean distances just like the silhouette feature vectors. Results differ depending on the user's body pose: when arms and legs can be identified easily, the motion capture delivers similar results as the silhouette matching. However, for poses with arms fully touching the body, the tracking fails. See Figure 8.14 for an example. This proves the suitability of our approach. In addition, the plots show how the non-rigid registration improves the output quality. By adding the depth data term, we were able to improve it further.

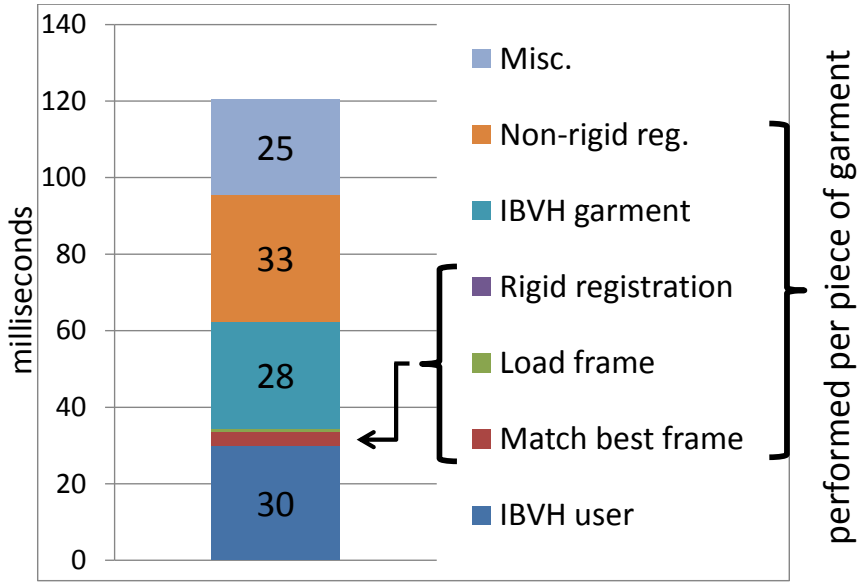


Figure 8.15: Performance of the runtime processes. The execution times for generating a typical augmented output image is given in milliseconds.

8.6.4 Performance

We measured the execution times of the runtime processes. One garment (a sweater) is transferred from one user to another. The garment database contains 275 frames, which in turn consist of 10 camera images. The evaluation system was equipped with a GeForce GTX 480 from Nvidia. Other PC components only have minor impact on the performance. The timings in Figure 8.15 are picked from a representative frame that required a reload of all camera images of the current user, and a reload of a frame from the garment database. The garment database was fully cached in GPU memory. Our implementation is not perfectly optimized, but runs at interactive rates for a single garment. We assume that the next hardware generation will be suitable for handling more than one item of clothing interactively.

8.7 Limitations and discussion

The suggested clothes augmentation pipeline transfers the appearance of garments from one user to another. It uses captured images for rendering, which inherently contain features like realistic light-cloth interaction and wrinkles. It is suitable for a wide range of garments, like shirts, sweaters, trousers, dresses and accessories like scarfs, sunglasses etc. Chapter 9 shows examples. Moreover, recording the garment data is quick and cheap compared to manual 3D modeling. However, the proposed pipeline stages also have limitations that can reduce the visual quality that the system achieves. In this section we want to discuss these limitations and their impact on the result.

Figure 8.16: This figure shows the visual impact of a missing pose in the garment database: the current user's left arm can not be covered by the transferred sweater.

We propose IBVH rendering for image synthesis from silhouette images. It is an approximative surface reconstruction method that can not capture all concavities and requires a relatively high number of cameras to produce a satisfying output quality. As a result, artifacts might appear, especially when the user's arms cover other body parts. At the same time, IBVH rendering is a very quick method even at high geometry/camera resolutions, which makes it particularly suitable for interactive reconstruction systems. Moreover, the visual hull is a good geometric proxy for projective texturing because it consists of large, smooth surface patches that can be textured with little distortions.

Both IBVH rendering and garment augmentation rely on image segmentation methods. Errors in the segmentation process lead to missing image regions or artifacts. However, due to the controlled background in our setup and control over what the model-user wears, this was not a problem during our experiments.

Matching algorithms that are based on silhouette shapes assume a certain shape similarity between images of the model- and the end user. This is not always given. For example, when the desired garment is a dress and the current user is wearing trousers, then the shapes are not similar in the leg region. While silhouette matching is quite reliable for small dissimilarities, it loses precision with, for example, longer dresses. In practice, we achieved good matching results even for dresses and coats (see Figure 9.2(a)), but we can not generally guarantee that it works for arbitrary garment and user shapes.

Visual artifacts can originate from missing poses in the garment database, see Figure 8.16 for an example. However, such problems are alleviated by our method for extending the pose space, and can be further reduced by carefully instructing the model-user. In practice it takes only a few minutes to create a sufficient pose space record.

The suggested composing method allows the garment to float on top, while the current user is only visible in image regions where the model-user was removed. It is not robust to bad matching or registration quality. In such cases, parts of the user might be covered by

garment where they should not be. In future work, a more robust method for composing could be used. For example, a graph-cut method could consider segmentation, color and depth information to find smooth borders between garment and user. However, the execution time of such algorithms may be too high for interactive systems.

The scene lighting in the virtual try-on room should be constant to allow the recorded garments' appearances to fit to the current user. But even under static lighting, there might be subtle coherence issues. Figure 8.16 shows an example: the rendered sweater looks rather masculine, which is due to the fact that the model-user was male. The non-rigid registration morphs the shape to match the user, but does not modify the colors. Shape information that is conveyed through shading is therefore still present in the augmentation. Future work should preprocess the garment database with a deshading algorithm, and perform shading computations during augmentation to suppress unwanted shape cues. In addition, such an extension would alleviate the limitation to a constant scene lighting.

The suggested image-based approach is not suitable for simulating physical effects of user motion. For example, dresses can not swing realistically. On the other hand we do not require any physical simulation to produce nicely fitting clothes with realistic wrinkles and lighting, and avoid potentially unstable computations.

8.8 Summary

In this chapter we suggested a pipeline that allows users to see themselves wearing different clothes from arbitrary viewpoints while being able to move freely. The process is interactive and can be used for a virtual dressing room application. As a result, the suggested approach reduces most of the shortcomings of previous approaches. We contributed efficient methods for silhouette matching, rigid- and non-rigid registration. We proposed to extend the pose space by combining several garment frames to improve the output quality. Moreover, loose clothing and virtual garments can be handled. An efficient solver for the optimization problem of non-rigid registration was described and evaluated.

In contrast to prior work, we use low-level image processing algorithms. This allows us to circumvent critical parts, such as motion capture and 3D reconstruction. Our approach is suitable for a wide range of clothing and even combinations of garments. Moreover, our system inherently offers realism by using image-based rendering, which makes it a good alternative to manually modeled garments. Manual interaction is only required once per garment to define its segmentation mask, which is a particularly fast task given modern segmentation algorithms. Recording garments is also faster and cheaper than modeling, which makes us confident that in the future such a system can be deployed in a retail environment. The augmentation quality and speed of the implemented system support our image-based augmentation hypothesis.

Chapter 9

Applications

Contents

9.1 Free viewpoint mirror

Image-based visual hull rendering in combination with view-dependent texture mapping allows for a free viewpoint mirror application. As shown in Figure 9.1, users can see themselves from all sides. Marker-less motion capture [153, 200] is used to track the user's body pose. First, a low resolution voxel grid is carved from the silhouette images. Then, a skeletal graph is extracted from the voxel grid to detect gestures of the user. Stretching out one arm triggers a viewpoint motion in the according direction. Stretching out both arms resets the viewpoint to a frontal view. Rendering is performed by the IBVH algorithm to allow for a high image resolution.

This virtual mirror system was demonstrated at the *Lange Nacht der Forschung* (long night of research) event in 2012. The green room was open at one side to allow for spectators. The viewpoint was configured to slowly rotate around the center of the room to demonstrate the smooth transitions between camera images. We observed that the rendering quality was satisfying for up to three simultaneous users when standing closely together. See Figure 9.1(c) for a picture. This mirror system is a prerequisite for our mixed reality virtual try-on application.

9.2 Virtual try-on

Virtual try-on applications allow users to watch themselves wearing different clothes without the effort of changing them. They help users to save time and make quick buying

(a)

(b)

(c)

Figure 9.1: A virtual mirror application using our image-based rendering architecture during usage. The viewpoint can be rotated seamlessly. A small number of users can be displayed when standing closely. Figure (c): Copyright - TU Graz / Lunghammer

decisions. In web shops virtual try-on components can be used to aid the user with selecting garments and determining the fit. As a consequence, retailers are able to improve the sales efficiency of their shop or web shop [41].

Previous solutions usually involve motion capture, 3D reconstruction or modeling, which are time consuming and error-prone. Our method avoids these steps by combining image-based renderings of the user and previously recorded garments [92, 94]. It transfers

(a) (b)

(c) (d)

Figure 9.2: Results of our image-based augmentation method: a user is dressed with the clothing of a previously recorded user. Image of the user and the garment before augmentation are shown for illustration.

the appearance of a garment recorded from one user to another by matching input and recorded frames, image-based visual hull rendering and online registration methods. Using images of real garments enables a realistic rendering quality with high performance. It is suitable for a wide range of clothes and complex appearances and allows arbitrary viewing angles. Figure 9.2 shows some results of our virtual try-on application. The process

requires only little manual input: when creating the garment database, the user needs to specify the segmentation mask in form of a color key or seed region to identify the desired garment. Our system is particularly useful for virtual try-on applications as well as interactive games.

9.3 Mixed reality systems

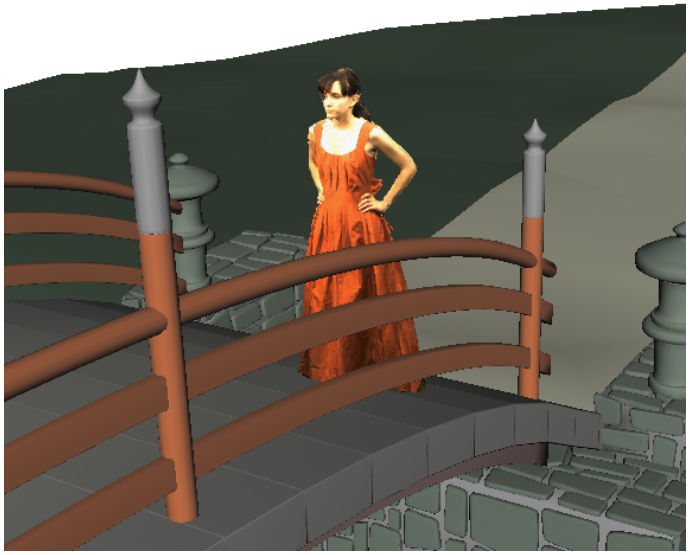
The image-based rendering methods described in this thesis operate in 3D. Therefore, the results can be embedded into a conventionally rendered visualization with correct overlap. To allow our rendering and augmentation components to be easily combined with an extensible graphics framework, all rendering stages are wrapped into Coin3D scene graph nodes. The current viewing and transformation matrix from the scene graph framework are queried and used by our algorithms. As a result, meshes and animations can be added quickly. User interface components like draggers and buttons can be used to place virtual objects relative to the image-based augmentations and trigger events. The runtime stages are implemented as Nvidia CUDA kernels, which allows us to formulate the computations without any graphics pipeline overhead. Results are written to the OpenGL frame buffer to integrate with the scene graph rendering process.

Figure 9.3 shows such a mixed reality scenario, where a user can watch himself in a virtual environment. This can help interactive entertainment applications, like games, to increase the sense of immersion. Moreover, remote collaboration systems become possible where multiple users interact with shared virtual objects or view and design products together.

9.4 Teleconferencing and telepresence

Modern teleconferencing or telepresence systems allow remote users to communicate and collaborate in an intuitive way [12]. Users are not required to switch between reading data on a screen and keeping eye contact while talking with each other. Instead, users, virtual objects and data share a synthesized space [75]. Other applications use 3D viewing for shopping with remote sales clerks and digitized products [119].

Our free viewpoint rendering methods create an output image directly from a set of camera images. When these images are transmitted over a network or internet connection, a three-dimensional remote viewing system can be realized. Such a system can be used by a telepresence or teleconferencing application with free viewpoint rendering and stereo viewing. Figure 9.4 illustrates a possible setup. It can provide an improved sense of immersion and presence to the participants.



(a)

(b)

Figure 9.3: Mixed reality demonstration applications: a user is augmented with a dress and rendered on a virtual bridge (a). Another user is displayed next to a virtual couch (b).

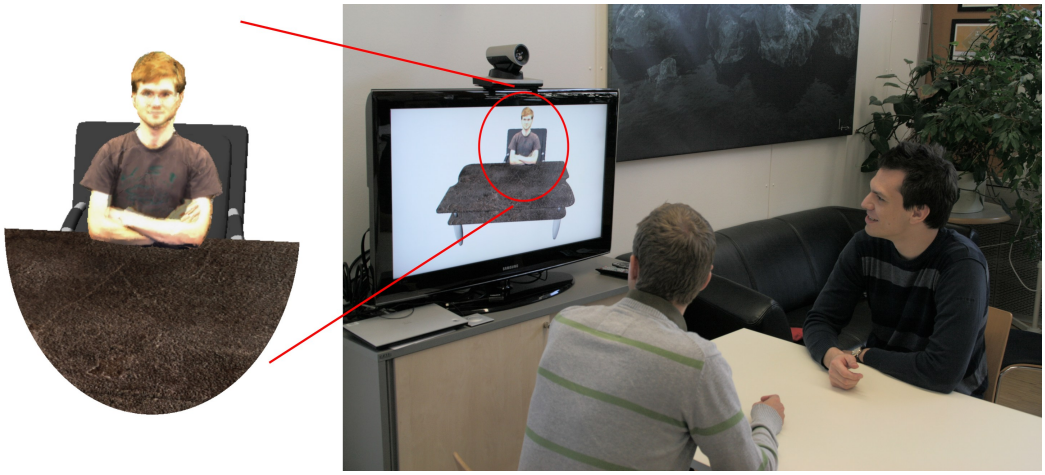


Figure 9.4: An illustration of a teleconferencing application with 3D rendering of a remote participant.

Chapter 10

Conclusions

Contents

In this thesis we have shown how image-based rendering methods can be used for the interactive free viewpoint rendering of humans captured by a multi-camera setup. We started with a state-of-the-art GPU implementation of the IBVH algorithm and analyzed its performance with respect to different resolutions. From this evaluation, we derived a range of novel algorithms that improve the execution performance and robustness of the rendering process.

We introduced a novel angular caching formulation that is free of singularities. We parallelized the algorithm across multiple GPUs by using different workload distribution schemes. We integrated depth sensors into our IBVH rendering pipeline to achieve higher rendering quality for complex objects or multiple simultaneous users, leading to the OmniKinect capturing room.

The suggested image-based rendering and augmentation pipeline can produce interactive frame rates for a resolution of up to one megapixel. To improve performance at higher resolutions on current GPU computing hardware, we introduced methods to exploit the frame-to-frame coherence. This temporal coherence is present in most multi-camera systems that capture humans because they typically move smoothly and slowly compared to the camera's frame rate. Therefore, new output images do not need to be computed from scratch every time.

Reusing previously computed data is a challenging problem in a free viewpoint system that renders arbitrarily moving and deforming surfaces. We achieved this task by detecting visual hull surface patches that remain static between consecutive frames. These static patches can be reprojected from the previous frame, and only the remaining image parts need to be rendered by the IBVH algorithm. Depending on the amount of coherence, this method reduces the execution time considerably. We showed how the motion detector that is used to detect static visual hull patches can also be used to guarantee a desired frame rate.

For scenes with weak coherence, such as when the user moves his or her entire body by walking, we had to find intermediate computation results that remain valid between consecutive frames even when the output depth map changes. We found that the camera association of a surface patch is more stable over time than the surface itself. Moreover, the user's body motions can be constrained by assuming a maximum motion distance. We showed how these insights can be exploited to increase performance. Furthermore, we showed how a dynamic camera order can help the IBVH algorithm to terminate quicker.

The suggested image-based methods can be used to render both humans and clothes. We used this idea to build an image-based augmentation pipeline, which uses previously captured sequences to augment users with new clothes or accessories. This virtual try-on application works with many types of garments. Capturing garments requires relatively little manual work: the garment must be worn while performing various motions, and after it is captured, the garment must be segmented by specifying a color key or region in the first set of camera images. The process is finished within minutes, while the manual modeling of garments in a CAD program is very labor intensive and requires a skilled artist.

The main focus of our work was to provide methods that allow for immediate 3D image synthesis from camera images with as little latency as possible. We developed image-based rendering and augmentation methods that are light-weight and provide a desirable rendering quality and resolution. The results that we achieved in terms of performance and visual quality allowed us to verify our hypotheses and enable a number of applications, including magic mirrors, virtual try-on systems, mixed reality (entertainment) applications and teleconferencing systems.

10.1 Lessons learned

Realizing a long image-based rendering and augmentation pipeline was a challenging task, particularly in terms of the achieved output quality. Many stages employ computer vision methods, such as camera calibration, visual hull reconstruction, segmentation and image registration. These methods are not entirely precise and suffer from noise, ambiguous data, poor illumination conditions, among other issues. In the field of computer vision, some of these problems can be alleviated or even solved with robust algorithms, such as optimization procedures, outlier removal methods or machine learning. However, our system is focused on immediate rendering, and thus, many of these advanced algorithms can not be used for performance reasons.

In addition to these conceptual difficulties, we came across many practical issues that needed to be considered for interactive image-based rendering of humans.

The room size that is used to host the cameras and the user requires special attention. The cameras must be a certain distance from the user to capture the entire body or at least significant parts of it. Deploying more cameras to capture a larger volume is not a good solution because it requires more processing power. Making the room too large moves the cameras too far away from the user and thus decreases the resolution of the

reconstructed surface. We observed that these principles are true for both color cameras and active depth sensors.

The number, type and arrangement of sensors are equally important. There is a practical limit to the number of active depth sensors depending on the method for coping with the interference between sensors. The number of color cameras is also limited but not as strictly. More color cameras cause more bus traffic and computation time, but they do not suffer from crosstalk. We experimented with different combinations of Microsoft Kinect devices and color cameras and suggest observing the quality of the reconstructed geometry and texturing separately. A good surface can be achieved by a relatively low number of active depth sensors, while good texturing requires a relatively high number of color cameras. We found that two active depth sensors along with a higher number of color cameras, at least eight, yield good results for both reconstruction and texturing. Moreover, this arrangement allows for interactive frame rates when GPUs are used for processing.

GPUs are very useful devices for image-based rendering and image processing tasks, particularly when programmed without using a standard graphics pipeline, such as OpenGL or DirectX. They allow for an execution throughput that is far beyond current CPUs as long as simple and independent operations are performed. The availability of these devices influenced our design decisions towards algorithms that facilitate a parallel execution.

10.2 Directions for future work

While the achieved quality and performance of our interactive IBR system are satisfying, there is still room for future improvements. For deploying the combined hard- and software setup at a client, in particular, the output quality must be improved. The first step towards an improved system would be to use higher camera resolutions and new bus types that provide sufficient bandwidth. We believe that future USB 3.0 or Ethernet cameras can achieve this bandwidth requirement. Future graphics processing units are likely to consist of even more parallel processing cores. Some of the additional computing power will be consumed by the higher camera resolutions, but we believe that sufficient resources will be available to improve other components. For example, surface reconstruction methods based on optimization between surface smoothness and data fit could help to reduce visual hull artifacts. Advanced texturing methods that align the textures of different cameras by a non-rigid deformation could help to decrease ghosting artifacts and increase the sharpness of the output image.

Given the promising results of this thesis and the improvements that future sensing and computing hardware will bring, we are confident that virtual try-on systems will be more common in fashion stores and that other interactive mixed reality systems will become marketable products.

Bibliography

[1] J. Allard, J. Franco, C. Menier, E. Boyer, and B. Raffin. The grimage platform: A mixed reality environment for interactions. In *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, p. 46, 2006. doi:10.1109/ICVS.2006.59. Cited on page 14.

[2] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph.*, vol. 22, no. 3, pp. 587–594, 2003. doi:10.1145/882262.882311. Cited on page 11.

[3] B. Allen, B. Curless, Z. Popović, and A. Hertzmann. Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pp. 147–156. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006. ISBN 3-905673-34-7. Cited on page 11.

[4] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. Scape: shape completion and animation of people. *ACM Trans. Graph.*, vol. 24, no. 3, pp. 408–416, 2005. doi:http://doi.acm.org/10.1145/1073204.1073207. Cited on pages 10 and 11.

[5] A. O. Balan, L. Sigal, M. J. Black, J. E. Davis, and H. W. Haussecker. Detailed human shape and pose from images. *Proc. IEEE CVPR*, vol. 0, pp. 1–8, 2007. doi:http://doi.ieeecomputersociety.org/10.1109/CVPR.2007.383340. Cited on page 10.

[6] L. Ballan, G. J. Brostow, J. Puwein, and M. Pollefeys. Unstructured video-based rendering: interactive exploration of casually captured videos. *ACM Trans. Graph.*, vol. 29, pp. 87:1–87:11, 2010. doi:http://doi.acm.org/10.1145/1778765.1778824. Cited on page 17.

[7] L. Ballan and G. M. Cortelazzo. Marker-less motion capture of skinned models in a four camera set-up using optical flow and silhouettes. In *3DPVT*. Atlanta, GA, USA, 2008. Cited on page 10.

[8] B. G. Baumgart. *Geometric modeling for computer vision*. Ph.D. thesis, Stanford, CA, USA, 1974. AAI7506806. Cited on page 10.

[9] K. Berger, K. Ruhl, C. Brümmer, Y. Schröder, A. Scholz, and M. Magnor. Markerless motion capture using multiple color-depth sensors. In *Proc. VMV 2011*, pp. 317–324, 2011. Cited on page 15.

[10] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popović, and S. M. Seitz. Estimating cloth simulation parameters from video. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pp. 37–51. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003. ISBN 1-58113-659-5. Cited on page 21.

[11] A. Bigdelou, A. Ladikos, and N. Navab. Incremental visual hull reconstruction. In *BMVC'09*, pp. –1–1, 2009. Cited on pages 20 and 78.

[12] M. Billinghurst, A. Cheok, S. Prince, and H. Kato. Real world teleconferencing. *IEEE Comput. Graph. Appl.*, vol. 22, no. 6, pp. 11–13, 2002. doi:http://dx.doi.org/10.1109/MCG.2002.1046623. Cited on pages 15 and 112.

[13] A. Bogomjakov, C. Gotsmann, and M. Magnor. Free-viewpoint video from depth cameras. In *Proc. Vision, Modeling and Visualization (VMV) 2006*, pp. 89–96, 2006. Cited on page 15.

[14] M. Botsch and O. Sorkine. On linear variational surface deformation methods. *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 1, pp. 213 –230, 2008. doi:10.1109/TVCG.2007.1054. Cited on page 94.

[15] H. Bowles, K. Mitchell, R. W. Sumner, J. Moore, and M. Gross. Iterative image warping. *Computer Graphics Forum*, vol. 31, no. 2pt1, pp. 237–246, 2012. doi:10.1111/j.1467-8659.2012.03002.x. Cited on page 19.

[16] E. Boyer. A hybrid approach for computing visual hulls of complex objects. In *In Computer Vision and Pattern Recognition*, pp. 695–701, 2003. Cited on pages 13 and 31.

[17] E. Boyer. On Using Silhouettes for Camera Calibration. In P. J. Narayanan, S. K. Nayar, and H.-Y. Shum (Editors), *7th Asian Conference on Computer Vision (ACCV '06)*, vol. 3851/2006 of *Lecture Notes in Computer Science (LNCS)*, pp. 1–10. Springer-Verlag, Hyderabad, India, 2006. doi:10.1007/11612032_1. Cited on page 12.

[18] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, 2001. doi:10.1109/34.969114. Cited on page 12.

[19] D. Bradley, T. Popa, A. Sheffer, W. Heidrich, and T. Boubekeur. Markerless garment capture. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pp. 99:1–99:9. ACM, New York, NY, USA, 2008. ISBN 978-1-4503-0112-1. doi:http://doi.acm.org/10.1145/1399504.1360698. Cited on page 14.

[20] D. Bradley, G. Roth, and P. Bose. Augmented reality on cloth with realistic illumination. *Machine Vision and Applications*, vol. 20, pp. 85–92, 2009. 10.1007/s00138-007-0108-9. Cited on page 17.

[21] G. J. Brostow, C. Hernandez, G. Vogiatzis, B. Stenger, and R. Cipolla. Video normals from colored lights. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 10, pp. 2104–2114, 2011. doi:10.1109/TPAMI.2011.37. Cited on page 10.

[22] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *proceedings of SIGGRAPH '01*, pp. 425–432. ACM, New York, NY, USA, 2001. ISBN 1-58113-374-X. doi:http://doi.acm.org/10.1145/383259.383309. Cited on pages 16 and 19.

[23] C. Buehler, W. Matusik, L. McMillan, and S. J. Gortler. Creating and rendering image-based visual hulls. Tech. rep., MIT LCS, 1999. Cited on page 18.

[24] A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim. Shake'n'sense: reducing interference for overlapping structured light depth cameras. In *Proc. CHI '12*, pp. 1933–1936, 2012. Cited on pages 15 and 52.

[25] J. Carranza, C. Theobalt, M. A. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, vol. 22, no. 3, pp. 569–577, 2003. doi:http://doi.acm.org/10.1145/882262.882309. Cited on page 10.

[26] E. E. Catmull. *A subdivision algorithm for computer display of curved surfaces.* Ph.D. thesis, 1974. AAI7504786. Cited on page 18.

[27] G. Chen, H. Su, J. Jiang, and W. Wu. Safe polyhedral visual hulls. In S. Boll, Q. Tian, L. Zhang, Z. Zhang, and Y.-P. Chen (Editors), *Advances in Multimedia Modeling*, vol. 5916 of *Lecture Notes in Computer Science*, pp. 35–44. Springer Berlin / Heidelberg, 2010. Cited on page 13.

[28] G. Cheung, S. Baker, J. Hodgins, and T. Kanade. Markerless human motion transfer. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pp. 31–. ACM, New York, NY, USA, 2004. ISBN 1-58113-896-2. doi:http://doi.acm.org/10.1145/1186223.1186262. Cited on page 11.

[29] K. M. G. Cheung. *Visual hull construction, alignment and refinement for human kinematic modeling, motion tracking and rendering.* Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2003. AAI3114280. Cited on page 11.

[30] K.-M. G. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time part i: Theory and algorithms. *Int. J. Comput. Vision*, vol. 62, no. 3, pp. 221–247, 2005. doi:10.1007/s11263-005-4881-5. Cited on page 13.

[31] K.-M. G. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time part ii: Applications to human modeling and markerless motion tracking. *Int. J. Comput. Vision*, vol. 63, no. 3, pp. 225–245, 2005. doi:10.1007/s11263-005-6879-4. Cited on page 13.

[32] F. Cordier and N. Magnenat-Thalmann. Real-time animation of dressed virtual humans. *Computer Graphics Forum*, vol. 21, no. 3, pp. 327–336, 2002. Cited on page 21.

[33] N. D'Apuzzo. Surface measurement and tracking of human body parts from multi-image video sequences. *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 56, pp. 360 – 375, 2002. doi:10.1016/S0924-2716(02)00069-2. ¡ce:title¿Theme Issue: Medical Imaging and Photogrammetry¡/ce:title¿. Cited on page 10.

[34] N. D'Apuzzo. Recent advances in 3d full body scanning with applications to fashion and apparel. In *Optical 3-D Measurement Techniques IX*. Vienna, Austria, 2009. Cited on page 9.

[35] A. Dayal, C. Woolley, B. Watson, and D. Luebke. Adaptive frameless rendering. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05. ACM, New York, NY, USA, 2005. doi:10.1145/1198555.1198763. Cited on page 20.

[36] E. de Aguiar, L. Sigal, A. Treuille, and J. K. Hodgins. Stable spaces for real-time clothing. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pp. 106:1–106:9. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0210-4. doi:http://doi.acm.org/10.1145/1833349.1778843. Cited on page 21.

[37] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. In *Proc. SIGGRAPH '08*, pp. 1–10. ACM, New York, NY, USA, 2008. ISBN 978-1-4503-0112-1. doi:http://doi.acm.org/10.1145/1399504.1360697. Cited on page 10.

[38] E. de Aguiar, C. Theobalt, C. Stoll, and H.-P. Seidel. Marker-less deformable mesh tracking for human shape and motion capture. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. XX–XX. IEEE, IEEE, Minneapolis, USA, 2007. Cited on page 10.

[39] P. Debevec, Y. Yu, and G. Boshokov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. Tech. rep., University of California at Berkeley, Berkeley, CA, USA, 1998. Cited on page 93.

[40] P. E. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In G. Drettakis and N. L. Max (Editors), *Rendering Techniques*, pp. 105–116. Springer, 1998. ISBN 3-211-83213-0. Cited on page 19.

[41] A. e. a. Divivier. Virtual try-on: Topics in realistic, individualized dressing in virtual reality. In *Proc. of Virtual and Augmented Reality Status Conference*. Leipzig (D), 2004. Cited on pages 21, 84, and 110.

[42] A. Djelouah, J.-S. Franco, E. Boyer, F. Leclerc, and P. Pérez. N-Tuple Color Segmentation for Multi-View Silhouette Extraction. In *12th European Conference on Computer Vision (ECCV'12)*. University of Florence, Firenze, Italy, 2012. Cited on page 12.

[43] J. Ehara and H. Saito. Texture overlay for virtual clothing based on pca of silhouettes. In *Proceedings of the 5th International Symposium on Mixed and Augmented Reality*, ISMAR '06, pp. 139–142. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 1-4244-0650-1. doi:http://dx.doi.org/10.1109/ISMAR.2006.297805. Cited on pages 18 and 87.

[44] M. Eisemann, B. D. Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating Textures. *Computer Graphics Forum (Proc. Eurographics EG'08)*, vol. 27, no. 2, pp. 409–418, 2008. Cited on page 19.

[45] M. Eisemann and M. A. Magnor. Filtered blending: A new, minimal reconstruction filter for ghosting-free projective texturing with multiple images. In H. P. A. Lensch, B. Rosenhahn, H.-P. Seidel, P. Slusallek, and J. Weickert (Editors), *Proceedings of the Vision, Modeling, and Visualization Conference 2007, VMV 2007, Saarbrücken, Germany, November 7-9, 2007*, pp. 119–126. Aka GmbH, 2007. ISBN 978-3-89838-085-0. Cited on page 19.

[46] M. Eisemann, T. Stich, and M. Magnor. 3-d cinematography with approximate and no geometry. *Image and Geometry Processing for 3-D Cinematography*, 2010. Cited on page 16.

[47] P. Eisert, P. Fechteler, and J. Rurainsky. 3-d tracking of shoes for virtual mirror applications. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1 –6, 2008. doi:10.1109/CVPR.2008.4587566. Cited on page 18.

[48] A. M. Elgammal, D. Harwood, and L. S. Davis. Non-parametric model for background subtraction. In *Proceedings of the 6th European Conference on Computer Vision-Part II*, ECCV '00, pp. 751–767. Springer-Verlag, London, UK, UK, 2000. ISBN 3-540-67686-4. Cited on page 12.

[49] P. Favaro and S. Soatto. A geometric approach to shape from defocus. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 3, pp. 406 –417, 2005. doi:10.1109/TPAMI.2005.43. Cited on page 10.

[50] J.-S. Franco and E. Boyer. Exact polyhedral visual hulls. In *British Machine Vision Conference (BMVC'03)*, vol. 1, pp. 329–338. Norwich, Royaume-Uni, 2003. Cited on page 13.

[51] J.-S. Franco and E. Boyer. Fusion of multiview silhouette cues using a space occupancy grid. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2, pp. 1747 –1753 Vol. 2, 2005. doi:10.1109/ICCV.2005.105. Cited on page 12.

[52] J.-S. Franco and E. Boyer. Efficient polyhedral modeling from silhouettes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 3, pp. 414 –427, 2009. doi:10.1109/TPAMI.2008.104. Cited on page 13.

[53] J.-S. Franco, M. Lapierre, and E. Boyer. Visual Shapes of Silhouette Sets. In *Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization*

and Transmission, pp. 1–8. United States, 2006. doi:10.1109/3DPVT.2006.148. Cited on page 13.

[54] J.-S. Franco, C. Menier, E. Boyer, and B. Raffin. A distributed approach for real time 3d modeling. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 3 - Volume 03*, pp. 31–. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2158-4. Cited on page 13.

[55] A. Fuhrmann, C. Gro?, V. Luckas, and A. Weber. Interaction-free dressing of virtual humans. *Computers & Graphics*, vol. 27, no. 1, pp. 71 – 82, 2003. doi:DOI: 10.1016/S0097-8493(02)00245-5. Cited on page 22.

[56] T. Furukawa, J. Gu, W. Lee, and N. Magnenat-Thalmann. 3d clothes modeling from photo cloned human body. In *Proceedings of the Second International Conference on Virtual Worlds*, VW '00, pp. 159–170. Springer-Verlag, London, UK, UK, 2000. ISBN 3-540-67707-0. Cited on page 14.

[57] Y. Furukawa and J. Ponce. Carved visual hulls for image-based modeling. *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 53–67, 2009. doi:10.1007/s11263-008-0134-8. Cited on page 13.

[58] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 8, pp. 1362 –1376, 2010. doi:10.1109/TPAMI.2009.161. Cited on page 10.

[59] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *Proc. IEEE CVPR*, 2009. Cited on page 10.

[60] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun. Real time motion capture using a single time-of-flight camera. In *CVPR*, 2010. Cited on page 11.

[61] V. Gay-Bellile, A. Bartoli, and P. Sayd. Deformable surface augmentation in spite of self-occlusions. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 235 –238, 2007. doi:10.1109/ISMAR.2007.4538853. Cited on page 17.

[62] M. Germann, A. Hornung, R. Keiser, R. Ziegler, S. Würmlin, and M. Gross. Articulated billboards for video-based rendering. *Computer Graphics Forum*, vol. 29, no. 2, pp. 585–594, 2010. doi:10.1111/j.1467-8659.2009.01628.x. Cited on page 17.

[63] I. Geys and L. V. Gool. View synthesis by the parallel use of gpu and cpu. *Image Vision Comput.*, vol. 25, no. 7, pp. 1154–1164, 2007. doi:http://dx.doi.org/10.1016/j.imavis.2006.07.023. Cited on page 14.

[64] B. Goldlücke. *Multi-Camera Reconstruction and Rendering for Free-Viewpoint Video*. Ph.D. thesis, Max-Planck-Institut für Informatik, 2006. Cited on page 17.

[65] B. Goldlücke and M. Magnor. Space-time isosurface evolution for temporally coherent 3d reconstruction. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'04),* Washington, USA, vol. I, pp. 350–355, 2004. Cited on page 20.

[66] B. Goldluecke and D. Cremers. Superresolution texture maps for multiview reconstruction. In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1677 –1684, 2009. doi:10.1109/ICCV.2009.5459378. Cited on page 17.

[67] B. Goldluecke and M. Magnor. Joint 3D reconstruction and background separation in multiple views using graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. I, pp. 683–694, 2003. Cited on page 12.

[68] B. Goldluecke and M. Magnor. Real-time, free-viewpoint video rendering from volumetric geometry. In T. Ebrahimi and T. Sikora (Editors), *Visual Communications and Image Processing 2003*, vol. 5150 of *SPIE proceedings*, pp. 1152–1158. The International Society for Optical Engineering (SPIE), SPIE, Lugano, Switzerland, 2003. ISBN 0-8194-5023-5. Cited on page 12.

[69] B. Goldluecke and M. Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. In *Proceedings of the IEEE International Conference on Image Processing*, vol. 3, pp. 713–716, 2003. Cited on page 15.

[70] B. Goldluecke and M. Magnor. Spacetime-continous geometry meshes from multiview video sequences. In *Proceedings of the IEEE International Conference on Image Processing*, pp. 516–522, 2005. Cited on page 20.

[71] M. Gong, J. M. Selzer, C. Lei, and Y.-H. Yang. Real-time backward disparity-based rendering for dynamic scenes using programmable graphics hardware. In *GI '07: Proceedings of Graphics Interface 2007*, pp. 241–248. ACM, New York, NY, USA, 2007. ISBN 978-1-56881-337-0. doi:http://doi.acm.org/10.1145/1268517.1268557. Cited on page 19.

[72] H. Graf, L. Hazke, S. Kahn, and C. Malerczyk. Accelerated real-time reconstruction of 3d deformable objects from multi-view video channels. In V. Duffy (Editor), *Digital Human Modeling*, vol. 6777 of *Lecture Notes in Computer Science*, pp. 282–291. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-21798-2. Cited on page 18.

[73] H. Graf, L. Hazke, S. Kahn, and C. Malerczyk. Accelerated real-time reconstruction of 3d deformable objects from multi-view video channels. In *HCI (17)'11*, pp. 282–291, 2011. Cited on page 18.

[74] C. Groß, A. Fuhrmann, and V. Luckas. Automatic pre-positioning of virtual clothing. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pp. 99–108. ACM, New York, NY, USA, 2003. ISBN 1-58113-861-X. doi:http://doi.acm.org/10.1145/984952.984970. Cited on page 22.

[75] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V. Moere, and O. Staadt. blue-c:

a spatially immersive display and 3d video portal for telepresence. In *proceedings of SIGGRAPH '03*. ACM, New York, USA, 2003. ISBN 1-58113-709-5. doi:http://doi.acm.org/10.1145/1201775.882350. Cited on pages 14, 84, and 112.

[76] L. Guan, J.-S. Franco, and M. Pollefeys. 3D Object Reconstruction with Heterogeneous Sensor Data. In *Proc. 3DPVT*, 2008. Cited on page 15.

[77] L. Guan, J. S. Franco, and M. Pollefeys. Multi-object shape estimation and tracking from silhouette cues. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008. Cited on page 12.

[78] L. Guan, J.-S. Franco, and M. Pollefeys. Multi-view occlusion reasoning for probabilistic silhouette-based dynamic scene reconstruction. *International Journal of Computer Vision*, vol. 90, pp. 283–303, 2010. 10.1007/s11263-010-0341-y. Cited on page 12.

[79] L. Guan, S. Sinha, J.-S. Franco, and M. Pollefeys. Visual hull construction in the presence of partial occlusion. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, 3DPVT '06, pp. 413–420. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2825-2. doi:10.1109/3DPVT.2006.147. Cited on page 12.

[80] F. Guggeri, R. Scateni, and R. Pajarola. Shape Reconstruction from Raw Point Clouds using Depth Carving. In *EG 2012 - Short Papers*, pp. 33–36. Eurographics Association, Cagliari, Sardinia, Italy, 2012. doi:10.2312/conf/EG2012/short/033-036. Cited on page 15.

[81] S. Guha, S. Krishnan, K. Munagala, and S. Venkatasubramanian. Application of the two-sided depth test to csg rendering. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, I3D '03, pp. 177–180. ACM, New York, NY, USA, 2003. ISBN 1-58113-645-5. doi:10.1145/641480.641513. Cited on page 13.

[82] I. Guskov, S. Klibanov, and B. Bryant. Trackable surfaces. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pp. 251–257. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003. ISBN 1-58113-659-5. Cited on page 14.

[83] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, second edn., 2003. ISBN 0521540518. Cited on page 10.

[84] J.-M. Hasenfratz, M. Lapierre, J.-D. Gascuel, and E. Boyer. Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors. In *Vision, Video and Graphics*, pp. 49–56. Eurographics, Elsevier, Bath, Royaume-Uni, 2003. Cited on page 15.

[85] J.-M. Hasenfratz, M. Lapierre, and F. X. Sillion. A Real-Time System for Full Body Interaction with Virtual Worlds. *Eurographics Symposium on Virtual Environments*, pp. 147–156, 2004. Cited on page 15.

[86] J. Hasselgren and T. Akenine-Möller. Efficient depth buffer compression. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '06, pp. 103–110. ACM, New York, NY, USA, 2006. ISBN 3-905673-37-1. doi:10.1145/1283900.1283917. Cited on page 42.

[87] S. Hauswiesner, P. Grasmug, D. Kalkofen, and D. Schmalstieg. Frame cache management for multi-frame rate systems. In *Proceedings of the 8th International Symposium on Visual Computing (ISVC)*. Crete, Greece, 2012. Cited on page 80.

[88] S. Hauswiesner, D. Kalkofen, and D. Schmalstieg. Multi-frame rate volume rendering. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. Norrköping, Sweden, 2010. Cited on pages 6, 40, 45, and 62.

[89] S. Hauswiesner, R. Khlebnikov, M. Steinberger, M. Straka, and G. Reitmayr. Multi-gpu image-based visual hull rendering. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. Sardinia, Italy, 2012. Cited on pages 5, 26, 34, and 40.

[90] S. Hauswiesner, M. Straka, and G. Reitmayr. Coherent image-based rendering of real-world objects. In *Symposium on Interactive 3D Graphics and Games*, pp. 183–190. ACM, New York, USA, 2011. ISBN 978-1-4503-0565-5. doi:http://doi.acm.org/10.1145/1944745.1944776. Cited on pages 5, 23, 26, 27, 39, 44, 65, and 76.

[91] S. Hauswiesner, M. Straka, and G. Reitmayr. Free viewpoint virtual try-on with commodity depth cameras. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '11, pp. 23–30. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-1060-4. doi:10.1145/2087756.2087759. Cited on page 7.

[92] S. Hauswiesner, M. Straka, and G. Reitmayr. Image-based clothes transfer. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Basel, SWITZERLAND, 2011. Cited on pages 5, 84, 88, and 110.

[93] S. Hauswiesner, M. Straka, and G. Reitmayr. Temporal coherence in image-based visual hull rendering. *Under major revisions at: IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2012. Cited on pages 5 and 34.

[94] S. Hauswiesner, M. Straka, and G. Reitmayr. Virtual try-on through image-based rendering. *Under major revisions at: IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2012. Cited on pages 6, 84, and 110.

[95] T. Hayashi, F. de Sorbier, and H. Saito. Texture overlay onto non-rigid surface using commodity depth camera. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, pp. 66–71. Rome, Italy, 2012. Cited on page 15.

[96] P. S. Heckbert. Survey of texture mapping. *IEEE Comput. Graph. Appl.*, vol. 6, no. 11, pp. 56–67, 1986. doi:10.1109/MCG.1986.276672. Cited on page 18.

[97] C. Hernandez, F. Schmitt, and R. Cipolla. Silhouette coherence for camera calibration under circular motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, pp. 343–349, 2007. Cited on page 12.

[98] A. Hilsmann and P. Eisert. Tracking and retexturing cloth for real-time virtual clothing applications. In *Proceedings of the 4th International Conference on Computer Vision/Computer Graphics CollaborationTechniques*, MIRAGE '09, pp. 94–105. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-01810-7. doi:10.1007/978-3-642-01811-4_9. Cited on page 17.

[99] A. Hilsmann and P. Eisert. Image-based Animation of Clothes. In *EG 2012 - Short Papers*. Eurographics Association, Cagliari, Sardinia, Italy, 2012. Cited on page 14.

[100] A. Hilsmann, D. C. Schneider, and P. Eisert. Realistic cloth augmentation in single view video under occlusions. *Comput. Graph.*, vol. 34, no. 5, pp. 567–574, 2010. doi:10.1016/j.cag.2010.05.015. Cited on page 17.

[101] A. Hilton, D. Beresford, T. Gentils, R. Smith, and W. Sun. Virtual people: Capturing human models to populate virtual worlds. In *Proceedings of the Computer Animation*, CA '99, pp. 174–. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0167-2. Cited on page 10.

[102] M. Hofmann and D. M. Gavrila. 3d human model adaptation by frame selection and shape-texture optimization. *Comput. Vis. Image Underst.*, vol. 115, no. 11, pp. 1559–1570, 2011. doi:10.1016/j.cviu.2011.08.002. Cited on page 10.

[103] B. Horn. Obtaining shape from shading information. In P. H. Winston (Editor), *The Psychology of Computer Vision*, McGraw-Hill Computer Science Series. McGraw-Hill, New York, 1975. Cited on page 10.

[104] A. Hornung and L. Kobbelt. Robust and efficient photo-consistency estimation for volumetric 3d reconstruction. In *Proceedings of the 9th European conference on Computer Vision - Volume Part II*, ECCV'06, pp. 179–190. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 3-540-33834-9, 978-3-540-33834-5. doi:10.1007/11744047_14. Cited on page 13.

[105] A. Hornung and L. Kobbelt. Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling. *Computer Graphics Forum*, vol. 28, no. 8, pp. 2090–2103, 2009. doi:10.1111/j.1467-8659.2009.01416.x. Cited on page 17.

[106] A. Jain, T. Thormählen, H.-P. Seidel, and C. Theobalt. Moviereshape: Tracking and reshaping of humans in videos. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2010)*, vol. 29, no. 5, 2010. Cited on page 11.

[107] Z. Janko and J.-P. Pons. Spatio-temporal image-based texture atlases for dynamic 3-d models. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 1646 –1653, 2009. doi:10.1109/ICCVW.2009.5457481. Cited on page 20.

[108] B. Kainz, S. Hauswiesner, G. Reitmayr, M. Steinberger, R. Grasset, L. Gruber, E. Veas, D. Kalkofen, H. Seichter, and D. Schmalstieg. Omnikinect: Real-time dense volumetric data acquisition and applications. In *Symposium on Virtual Reality Software and Technology (VRST)*, 2012. Cited on pages 6, 8, and 49.

[109] M. Kampel, S. Tosovic, and R. Sablatnig. Octree-based fusion of shape from silhouette and shape from structured light. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pp. 754 – 757, 2002. doi:10.1109/TDPVT.2002.1024154. Cited on page 15.

[110] T. Kanade, A. Gruss, and L. R. Carley. A very fast VLSI rangefinder. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1322–1329, 1991. doi:10.1109/ROBOT.1991.131796. Cited on page 15.

[111] Y.-M. Kang and H.-G. Cho. Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *CA '02: Proceedings of the Computer Animation*, p. 203. IEEE Computer Society, Washington, DC, USA, 2002. ISBN 0-7695-1594-0. Cited on page 21.

[112] M. Keckeisen, M. Feurer, and M. Wacker. Tailor tools for interactive design of clothing in virtual environments. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 182–185. ACM, New York, NY, USA, 2004. ISBN 1-58113-907-1. doi:http://doi.acm.org/10.1145/1077534.1077572. Cited on page 21.

[113] S. Kim, H.-D. Kim, W.-J. Kim, and S.-D. Kim. Fast computation of a visual hull. In *Proceedings of the 10th Asian conference on Computer vision - Volume Part IV*, ACCV'10, pp. 1–10. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-19281-4. Cited on page 18.

[114] Y. M. Kim, D. Chan, C. Theobalt, and S. Thrun. Design and calibration of a multi-view TOF sensor fusion system. In *Proc. IEEE CVPR Workshops*, pp. 1 –7, 2008. doi:10.1109/CVPRW.2008.4563160. Cited on page 15.

[115] D. Knoblauch and F. Kuester. Region-of-interest volumetric visual hull refinement. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, VRST '10, pp. 143–150. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0441-2. Cited on page 13.

[116] C. Kuster, T. Popa, C. Zach, C. Gotsman, and M. Gross. Freecam: A hybrid camera system for interactive free-viewpoint video. In *Proc. of the International Workshop on Vision, Modeling and Visualization (VMV)*, 2011. Cited on page 15.

[117] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, vol. 38, pp. 199–218, 2000. 10.1023/A:1008191222954. Cited on page 12.

[118] A. Ladikos, S. Benhimane, and N. Navab. Efficient visual hull computation for real-time 3d reconstruction using cuda. *Computer Vision and Pattern Recognition*

Workshop, vol. 0, pp. 1–8, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/CVPRW.2008.4563098. Cited on page 12.

[119] S. Lang, M. Naef, M. Gross, and L. Hovestadt. In:shop - using telepresence and immersive vr for a new shopping experience. In *In Proceedings of the 8th International Fall Workshop on Vision, Modelling and Visualization 2003. IEEE*, 2003. Cited on pages 14, 84, and 112.

[120] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 2, pp. 150–162, 1994. doi:10.1109/34.273735. Cited on page 11.

[121] J.-M. Lavest, G. Rives, and M. Dhome. Three-dimensional reconstruction by zooming. *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 2, pp. 196 –207, 1993. doi:10.1109/70.238283. Cited on page 10.

[122] S. Lazebnik, Y. Furukawa, and J. Ponce. Projective visual hulls. *International Journal of Computer Vision*, vol. 74, pp. 137–165, 2007. 10.1007/s11263-006-0008-x. Cited on page 13.

[123] C. Lee, J. Cho, and K. Oh. Hardware-accelerated jaggy-free visual hulls with silhouette maps. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 87–90. ACM, New York, NY, USA, 2006. ISBN 1-59593-321-2. doi:http://doi.acm.org/10.1145/1180495.1180513. Cited on page 13.

[124] W. Lee, W. Woo, and E. Boyer. Identifying foreground from multiple images. In *Proceedings of the 8th Asian conference on Computer vision - Volume Part II*, ACCV'07, pp. 580–589. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 3-540-76389-9, 978-3-540-76389-5. Cited on page 12.

[125] W. Lee, W. Woo, and E. Boyer. Silhouette segmentation in multiple views. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 7, pp. 1429 –1441, 2011. doi:10.1109/TPAMI.2010.196. Cited on page 12.

[126] J. Lengyel. The convergence of graphics and vision. *Computer*, vol. 31, no. 7, pp. 46 –53, 1998. doi:10.1109/2.689676. Cited on page 16.

[127] H. Li, L. Luo, D. Vlasic, P. Peers, J. Popović, M. Pauly, and S. Rusinkiewicz. Temporally coherent completion of dynamic shapes. *ACM Transactions on Graphics*, vol. 31, no. 1, 2011. Cited on page 20.

[128] J. Li and G. Lu. Customizing 3d garments based on volumetric deformation. *Computers in Industry*, vol. 62, no. 7, pp. 693 – 707, 2011. doi:10.1016/j.compind.2011.04.002. Cited on pages 14 and 94.

[129] J. Li, J. Ye, Y. Wang, L. Bai, and G. Lu. Technical section: Fitting 3d garment models onto individual human models. *Comput. Graph.*, vol. 34, pp. 742–755, 2010. doi:http://dx.doi.org/10.1016/j.cag.2010.07.008. Cited on page 14.

[130] M. Li. *Towards Real-Time Novel View Synthesis Using Visual Hulls*. Ph.D. thesis, Universität des Saarlandes, 2004. Cited on page 13.

[131] M. Li, M. Magnor, and H. peter Seidel. Hardware-accelerated visual hull reconstruction and rendering. In *In Graphics Interface 2003*, pp. 65–71, 2003. Cited on page 13.

[132] M. Li, M. Magnor, and H. peter Seidel. Improved hardware-accelerated visual hull rendering. In *Proc. Vision, Modeling, and Visualization (VMV-2003)*. Munich, Germany, 2003. Cited on page 13.

[133] M. Li, M. Magnor, and H.-P. Seidel. Hardware-accelerated rendering of photo hulls. *Computer Graphics Forum (Eurographics'04)*, vol. 23, no. 3, pp. 635–642, 2004. Cited on page 14.

[134] M. Li, M. Magnor, and H.-P. Seidel. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *GI '04: Proceedings of Graphics Interface 2004*, pp. 41–48. Canadian Human-Computer Communications Society, 2004. ISBN 1-56881-227-2. Cited on page 13.

[135] M. Li, H. Schirmacher, M. Magnor, and H.-P. Siedel. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Multimedia Signal Processing, 2002 IEEE Workshop on*, pp. 9 – 12, 2002. doi:10.1109/MMSP.2002.1203235. Cited on page 18.

[136] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor. Virtual video camera: Image-based viewpoint navigation through space and time. *Computer Graphics Forum*, vol. 29, no. 8, pp. 2555–2568, 2010. Cited on page 16.

[137] Y. Liu, G. Chen, N. Max, C. Hofsetz, and P. McGuinness. Visual hull rendering with multi-view stereo refinement. In *WSCG*, pp. 261–268, 2004. Cited on page 14.

[138] Y. Liu, G. Chen, N. Max, C. Hofsetz, and P. Mcguinness. Visual hull rendering with multi-view stereo refinement. *Journal of WSCG*, 2004. Cited on page 14.

[139] Y. Liu, Q. Dai, and W. Xu. A point-cloud-based multiview stereo algorithm for free-viewpoint video. *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 3, pp. 407 –418, 2010. doi:10.1109/TVCG.2009.88. Cited on page 10.

[140] T. Luginbühl, L. Delattre, and A. Gagalowicz. Towards the automatic generation of 3d photo-realistic avatars using 3d scanned data. In *Proceedings of the 5th international conference on Computer vision/computer graphics collaboration techniques*, MIRAGE'11, pp. 192–203. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-24135-2. Cited on page 11.

[141] L. Ma, J. Hu, and G. Baciu. Generating seams and wrinkles for virtual clothing. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pp. 205–211. ACM, New York, NY, USA, 2006. ISBN 1-59593-324-7. doi:http://doi.acm.org/10.1145/1128923.1128957. Cited on page 21.

[142] N. Magnenat-Thalmann, C. Luible, P. Volino, and E. Lyard. From measured fabric to the simulation of cloth. In *12th IEEE Inter. Conference on Emerging Technologies and Factory Automation*, 2007. Cited on page 21.

[143] N. Magnenat-Thalmann, H. Seo, and F. Cordier. Automatic modeling of animatable virtual humans - a survey. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pp. 2 – 10, 2003. doi:10.1109/IM.2003.1240226. Cited on page 11.

[144] M. A. Magnor. *Video-Based Rendering*. AK Peters, first edn., 2005. ISBN 1568812442, 978-1568812441. Cited on page 16.

[145] A. Maimone and H. Fuchs. Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras. In *Proc. ISMAR '11*, pp. 137–146, 2011. ISBN 978-1-4577-2183-0. Cited on page 15.

[146] A. Maimone and H. Fuchs. Real-time volumetric 3d capture of room-sized scenes for telepresence. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*. Zurich, Switzerland, 2012. Cited on page 15.

[147] A. Maimone and H. Fuchs. Reducing interference between multiple structured light depth sensors using motion. In *Proc. IEEE VR*, pp. 51–54, 2012. ISBN 978-1-4673-1247-9. Cited on pages 9, 15, and 52.

[148] Y. Makihara and Y. Yagi. Silhouette extraction based on iterative spatio-temporal local color transformation and graph-cut segmentation. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1 –4, 2008. doi:10.1109/ICPR.2008.4761121. Cited on page 12.

[149] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich. Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pp. 40:1–40:14. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0943-1. doi:http://doi.acm.org/10.1145/1964921.1964935. Cited on page 104.

[150] M. L. Marcus, M. Li, M. Magnor, and H. peter Seidel. Improved hardware-accelerated visual hull rendering, 2003. Cited on page 13.

[151] W. R. Mark, L. Mcmillan, and G. Bishop. Post-rendering 3d warping. In *In 1997 Symposium on Interactive 3D Graphics*, 1997. Cited on page 19.

[152] W. N. Martin and J. K. Aggarwal. Volumetric descriptions of objects from multiple views. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-5, no. 2, pp. 150 –158, 1983. doi:10.1109/TPAMI.1983.4767367. Cited on page 12.

[153] M. R. Matthias Straka, Stefan Hauswiesner and H. Bischof. Skeletal graph based human pose estimation in real-time. In *Proceedings of the British Machine Vision Conference*, pp. 69.1–69.12. BMVA Press, 2011. ISBN 1-901725-43-X. Http://dx.doi.org/10.5244/C.25.69. Cited on page 109.

[154] W. Matusik, C. Buehler, and L. McMillan. Efficient view-dependent sampling of visual hulls, 2001. Cited on page 18.

[155] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pp. 115–126. Springer-Verlag, London, UK, 2001. ISBN 3-211-83709-4. Cited on page 13.

[156] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 369–374. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. ISBN 1-58113-208-5. doi:http://doi.acm.org/10.1145/344779.344951. Cited on pages 18 and 32.

[157] W. Matusik and H. Pfister. 3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *Proc. SIGGRAPH '04*, pp. 814–824, 2004. doi:10.1145/1186562.1015805. Cited on page 16.

[158] W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. *ACM Trans. Graph.*, vol. 21, no. 3, pp. 427–437, 2002. doi:10.1145/566654.566599. Cited on page 18.

[159] G. Miller and A. Hilton. Safe hulls. In *Visual Media Production, 2007. IETCVMP. 4th European Conference on*, pp. 1 –8, 2007. Cited on page 13.

[160] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, pp. 35:1–35:11. ACM, New York, NY, USA, 2008. doi:http://doi.acm.org/10.1145/1508044.1508079. Cited on pages 41 and 42.

[161] S. K. Nayar and Y. Nakagawa. Shape from focus. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 8, pp. 824–831, 1994. Cited on page 10.

[162] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '07, pp. 25–35. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2007. ISBN 978-1-59593-625-7. Cited on page 20.

[163] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc IEEE ISMAR '11*, pp. 127–136, 2011. ISBN 978-1-4577-2183-0. doi:10.1109/ISMAR.2011.6092378. Cited on page 15.

[164] W. Niem and H. Broszio. Mapping texture from multiple camera views onto 3d-object models for computer animation. In *in Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, pp. 99–105, 1995. Cited on page 19.

[165] C. Nitschke, A. Nakazawa, and H. Takemura. Real-time space carving using graphics hardware. *IEICE - Trans. Inf. Syst.*, vol. E90-D, pp. 1175–1184, 2007. doi:10.1093/ietisy/e90-d.8.1175. Cited on page 12.

[166] S. Pabst, S. Krzywinski, A. Schenk, and B. Thomaszewski. Seams and bending in cloth simulation. In *Proceedings of the Fifth Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS*, pp. 31–38. Eurographics Association, Grenoble, France, 2008. Cited on page 21.

[167] I. Park, M. Germann, M. Breitenstein, and H. Pfister. Fast and automatic object pose estimation for range images on the gpu. *Machine Vision and Applications*, vol. 21, pp. 749–766, 2010. 10.1007/s00138-009-0209-8. Cited on page 11.

[168] B. Petit, J.-D. Lesage, C. Ménier, J. Allard, J.-S. Franco, B. Raffin, E. Boyer, and F. Faure. Multi-camera real-time 3d modeling for telepresence and remote collaboration. *International Journal of Digital Multimedia Broadcasting*, 2010. Cited on page 14.

[169] J. Pilet. *Augmented reality for non-rigid surfaces*. Ph.D. thesis, IC, Lausanne, 2008. doi:10.5075/epfl-thesis-4192. Cited on page 17.

[170] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Comput. Vision Graph. Image Process.*, vol. 40, no. 1, pp. 1–29, 1987. doi:10.1016/0734-189X(87)90053-3. Cited on page 12.

[171] M. J. D. Powell. Convergence properties of algorithms for nonlinear optimization. *SIAM Review*, vol. 28, no. 4, pp. pp. 487–500, 1986. Cited on page 96.

[172] S. Prince, A. D. Cheok, F. Farbiz, T. Williamson, N. Johnson, M. Billinghurst, and H. Kato. 3d live: Real time captured content for mixed reality. In *proceedings of ISMAR '02*, p. 7. IEEE Computer Society, Washington, DC, USA, 2002. ISBN 0-7695-1781-1. Cited on page 14.

[173] D. Pritchard and W. Heidrich. Cloth motion capture. In *Eurographics*, pp. 263–271, 2003. Cited on page 14.

[174] L. Rogge, T. Neumann, M. Wacker, and M. Magnor. Monocular pose reconstruction for an augmented reality clothing system. In *Proc. Vision, Modeling and Visualization (VMV) 2011*, pp. 339–346, 2011. Cited on page 10.

[175] D. Rohmer, T. Popa, M.-P. Cani, S. Hahmann, and A. Sheffer. Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Trans. Graph.*, vol. 29, no. 6, pp. 157:1–157:8, 2010. doi:10.1145/1882261.1866183. Cited on page 21.

[176] M. Salzmann, J. Pilet, S. Ilic, and P. Fua. Surface deformation models for nonrigid 3d shape recovery. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pp. 1481–1487, 2007. doi:http://dx.doi.org/10.1109/TPAMI.2007.1080. Cited on page 14.

[177] D. Scherzer, L. Yang, O. Mattausch, D. Nehab, P. V. Sander, M. Wimmer, and E. Eisemann. A survey on temporal coherence methods in real-time rendering. In *EUROGRAPHICS 2011 State of the Art Reports*, pp. 101–126. Eurographics Association, 2011. Cited on page 20.

[178] V. Scholz and M. A. Magnor. Cloth motion from optical flow. In *VMV'04*, pp. 117–124, 2004. Cited on page 14.

[179] V. Scholz, T. Stich, M. Keckeisen, M. Wacker, and M. Magnor. Garment motion capture using color-coded patterns. *Computer Graphics Forum (Proc. Eurographics EG'05)*, vol. 24, no. 3, pp. 439–448, 2005. Cited on page 14.

[180] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pp. 249–252. ACM, New York, NY, USA, 1992. ISBN 0-89791-479-1. doi:10.1145/133994.134071. Cited on page 19.

[181] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 519–528. IEEE Computer Society, 2006. Cited on page 10.

[182] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pp. 1067–1073, 1997. Cited on pages 10, 11, and 12.

[183] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proc. IEEE CVPR'11*, 2011. Cited on page 11.

[184] H.-Y. Shum, S.-C. Chan, and S. B. Kang. *Image-Based Rendering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387211136. Cited on pages 16, 19, 43, and 97.

[185] M. Simmons and C. H. Séquin. Tapestry: A dynamic mesh-based display representation for interactive rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pp. 329–340. Springer-Verlag, London, UK, UK, 2000. ISBN 3-211-83535-0. Cited on page 20.

[186] S. Sinha and M. Pollefeys. Visual-hull reconstruction from uncalibrated and unsynchronized video streams. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pp. 349 – 356, 2004. doi:10.1109/TDPVT.2004.1335227. Cited on page 12.

[187] G. G. Slabaugh, G. G. Slabaugh, R. W. Schafer, R. W. Schafer, M. C. Hans, and M. C. Hans. Image-based photo hulls. In *In The Proceedings of the 1 st International Symposium on 3D Processing, Visualization, and Transmission*, pp. 704–708, 2002. Cited on page 18.

[188] F. A. Smit, R. van Liere, S. Beck, and B. Fröhlich. An image-warping architecture for vr: Low latency versus image quality. In *Proc. IEEE VR*, pp. 27–34, 2009. Cited on page 21.

[189] A. Smolic, K. Mueller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, and T. Wiegand. 3d video and free viewpoint video - technologies, applications and mpeg standards. In *Multimedia and Expo, 2006 IEEE International Conference on*, pp. 2161 –2164, 2006. doi:10.1109/ICME.2006.262683. Cited on page 16.

[190] J. P. Springer, S. Beck, F. Weiszig, D. Reiners, and B. Froehlich. Multi-frame rate rendering and display. In W. R. Sherman, M. Lin, and A. Steed (Editors), *VR*, pp. 195–202. IEEE Computer Society, 2007. ISBN 1-4244-0905-5. Cited on page 20.

[191] J. Starck and A. Hilton. Virtual view synthesis of people from multiple view video sequences. *Graph. Models*, vol. 67, no. 6, pp. 600–620, 2005. doi:http://dx.doi.org/10.1016/j.gmod.2005.01.008. Cited on page 13.

[192] J. Starck and A. Hilton. Surface capture for performance-based animation. *IEEE Comput. Graph. Appl.*, vol. 27, no. 3, pp. 21–31, 2007. doi:10.1109/MCG.2007.68. Cited on page 13.

[193] J. Starck and A. Hilton. Model-based human shape reconstruction from multiple views. *Computer Vision and Image Understanding*, vol. 111, no. 2, pp. 179 – 194, 2008. doi:10.1016/j.cviu.2007.10.001. Cited on page 10.

[194] J. Starck, A. Maki, S. Nobuhara, A. Hilton, and T. Matsuyama. The multiple-camera 3-d production studio. *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 6, pp. 856 –869, 2009. doi:10.1109/TCSVT.2009.2017406. Cited on page 11.

[195] J. Starck, G. Miller, and A. Hilton. Video-based character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pp. 49–58. ACM, New York, NY, USA, 2005. ISBN 1-59593-198-8. doi:10.1145/1073368.1073375. Cited on page 11.

[196] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2, pp. 2 vol. (xxiii+637+663), 1999. doi:10.1109/CVPR.1999.784637. Cited on page 12.

[197] M. Steinberger, B. Kainz, B. Kerbl, S. Hauswiesner, M. Kenzel, and D. Schmalstieg. Softshell: Dynamic scheduling on gpus. In *SIGGRAPH Asia '12: ACM SIGGRAPH Asia 2012 papers*. ACM, New York, NY, USA, 2012. Cited on pages 6, 8, and 66.

[198] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor. Perception-motivated interpolation of image sequences. In *Proc. ACM Applied Perception in Computer Graphics and Visualization (APGV) 2008*, pp. 97–106. ACM, ACM, Los Angeles, USA, 2008. Http://doi.acm.org/10.1145/1394281.1394299. Cited on page 16.

[199] C. Stoll, J. Gall, E. de Aguiar, S. Thrun, and C. Theobalt. Video-based reconstruction of animatable human characters. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pp. 139:1–139:10. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0439-9. doi:http://doi.acm.org/10.1145/1866158.1866161. Cited on page 14.

[200] M. Straka, S. Hauswiesner, M. Ruether, and H. Bischof. A free-viewpoint virtual mirror with marker-less user interaction. In *Proc. of the 17th Scandinavian Conference on Image Analysis (SCIA)*, 2011. Cited on pages 6, 24, and 109.

[201] H. Tanaka and H. Saito. Texture overlay onto flexible object with pca of silhouettes and k-means method for search into database. In *Proceedings of the IAPR Conference on Machine Vision Applications*. Yokohama, JAPAN, 2009. Cited on pages 18, 85, 88, and 89.

[202] A. Taneja, L. Ballan, and M. Pollefeys. Modeling dynamic scenes recorded with freely moving cameras. In *Proceedings of the 10th Asian conference on Computer vision - Volume Part III*, ACCV'10, pp. 613–626. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-19317-0. Cited on page 17.

[203] A. Tevs, A. Berner, M. Wand, I. Ihrke, M. Bokeloh, J. Kerber, and H.-P. Seidel. Animation cartography - intrinsic reconstruction of shape and motion. *ACM Trans. Graph.*, vol. 31, no. 2, pp. 12:1–12:15, 2012. doi:10.1145/2159516.2159517. Cited on page 20.

[204] C. Theobalt, N. Ahmed, H. Lensch, M. Magnor, and H.-P. Seidel. Seeing people in different light-joint shape, motion, and reflectance capture. *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 4, pp. 663–674, 2007. doi:10.1109/TVCG.2007.1006. Cited on page 10.

[205] C. Theobalt, J. Carranza, M. A. Magnor, and H.-P. Seidel. Combining 3d flow fields with silhouette-based human motion capture for immersive video. *Graphical Models*, vol. 66, no. 6, pp. 333 – 351, 2004. doi:10.1016/j.gmod.2004.06.009. ¡ce:title¿Special Issue on Pacific Graphics 2003¡/ce:title¿. Cited on page 11.

[206] C. Theobalt, M. Li, M. A. Magnor, and H. peter Seidel. A flexible and versatile studio for synchronized multi-view video recording. In *Vision, Video and Graphics, p.9-16, Bath, UK*, 2003. Cited on page 12.

[207] C. Theobalt, M. Magnor, P. Schüler, and H.-P. Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. *International Journal of Image and Graphics*, vol. 4, no. 4, pp. 563–584, 2004. Pacific Graphics 2002. Cited on page 10.

[208] J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan. Scanning 3d full human bodies using kinects. *IEEE TVCG*, pp. 643–650, 2012. Cited on page 15.

[209] L. Torresani, D. B. Yang, E. J. Alexander, and C. Bregler. Tracking and modeling non-rigid objects with rank constraints. In *CVPR (1)'01*, pp. 493–500, 2001. Cited on page 14.

[210] E. Turquin, J. Wither, L. Boissieux, M.-P. Cani, and J. F. Hughes. A sketch-based interface for clothing virtual characters. *IEEE Computer Graphics and Applications*, vol. 27, no. 1, pp. 72–81, 2007. doi:10.1109/MCG.2007.1. Special issue on sketch based modeling. Cited on page 21.

[211] K. Tzevanidis, X. Zabulis, T. Sarmis, P. Koutlemanis, N. Kyriazis, and A. Argyros. From multiple views to textured 3d meshes: a gpu-powered approach. In *ECCV'2010 Workshop on Computer Vision on GPUs (CVGPU2010)*, pp. 5–11, 2010. Cited on page 12.

[212] N. Ukita and T. Kanade. Reference consistent reconstruction of 3d cloth surface. *Computer Vision and Image Understanding*, vol. 116, no. 8, pp. 869 – 881, 2012. doi:10.1016/j.cviu.2012.04.001. Cited on page 14.

[213] S. Vedula, S. Baker, and T. Kanade. Image-based spatio-temporal modeling and view interpolation of dynamic events. *ACM Trans. Graph.*, vol. 24, no. 2, pp. 240–261, 2005. doi:10.1145/1061347.1061351. Cited on page 17.

[214] S. Vedula, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 3, pp. 475 –480, 2005. doi:10.1109/TPAMI.2005.63. Cited on page 17.

[215] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–9, 2008. doi:http://doi.acm.org/10.1145/1360612.1360696. Cited on page 10.

[216] D. Vlasic, P. Peers, I. Baran, P. Debevec, J. Popović, S. Rusinkiewicz, and W. Matusik. Dynamic shape capture using multi-view photometric stereo. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pp. 174:1–174:11. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-858-2. doi:http://doi.acm.org/10.1145/1661412.1618520. Cited on page 14.

[217] P. Volino, C. Luible, and N. Magnenat-Thalmann. Virtual clothing. In B. W. Wah (Editor), *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., 2008. Cited on page 21.

[218] P. Volino and N. Magnenat-Thalmann. Accurate garment prototyping and simulation. *Computer-Aided Design & Applications*, vol. 2, no. 5, pp. 645–654, 2005. Cited on page 21.

[219] M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, R. Sarlette, M. Sattler, and R. Klein. Simulation and visualisation of virtual textiles for virtual try-on. *Special Issue of Research Journal of Textile and Apparel: Virtual Clothing Technology and Applications*, vol. 9, no. 1, 2005. Cited on page 21.

[220] D. Wagner, T. Langlotz, and D. Schmalstieg. Robust und unobtrusive marker tracking on mobile phones. In *Proc. IEEE ISMAR'08*, pp. 121–124, 2008. Cited on page 25.

[221] W. Waizenegger, I. Feldmann, P. Eisert, and P. Kauff. Parallel high resolution real-time visual hull on gpu. In *ICIP'09: Proceedings of the 16th IEEE international conference on Image processing*, pp. 4245–4248. IEEE Press, Piscataway, NJ, USA, 2009. ISBN 978-1-4244-5653-6. Cited on pages 18, 27, 30, and 68.

[222] B. Walter, G. Drettakis, and D. P. Greenberg. Enhancing and optimizing the render cache. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pp. 37–42. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2002. ISBN 1-58113-534-3. Cited on page 20.

[223] C. C. L. Wang, K.-C. Hui, and K.-M. Tong. Volume parameterization for design automation of customized free-form products. *Automation Science and Engineering, IEEE Transactions on*, vol. 4, no. 1, pp. 11 –21, 2007. doi:10.1109/TASE.2006.872112. Cited on page 14.

[224] M. Waschbüsch, S. Würmlin, and M. Gross. 3d video billboard clouds. *Computer Graphics Forum*, vol. 26, no. 3, pp. 561–569, 2007. doi:10.1111/j.1467-8659.2007.01079.x. Cited on page 17.

[225] X. Wei and J. Chai. Videomocap: modeling physically realistic human motion from monocular video sequences. *ACM Trans. Graph.*, vol. 29, no. 4, pp. 42:1–42:10, 2010. doi:10.1145/1778765.1778779. Cited on page 11.

[226] R. White, K. Crane, and D. Forsyth. Capturing and animating occluded cloth. In *ACM Transactions on Graphics (SIGGRAPH)*, 2007. Cited on pages 10 and 14.

[227] R. White and D. Forsyth. Retexturing single views using texture and shading. In *Proceedings of the 9th European conference on Computer Vision - Volume Part IV*, ECCV'06, pp. 70–81. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 3-540-33838-1, 978-3-540-33838-3. doi:10.1007/11744085_6. Cited on page 17.

[228] A. D. Wilson and H. Benko. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proc. ACM UIST '10*, pp. 273–282, 2010. ISBN 978-1-4503-0271-5. doi:10.1145/1866029.1866073. Cited on page 15.

[229] H. Winnemöller, A. Orzan, L. Boissieux, and J. Thollot. Texture design and draping in 2d images. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2009)*, vol. 28, no. 4, pp. 1091–1099, 2009. Cited on page 17.

[230] X. Wu, O. Takizawa, and T. Matsuyama. Parallel pipeline volume intersection for real-time 3d shape reconstruction on a pc cluster. In *Proceedings of the Fourth IEEE International Conference on Computer Vision Systems*, pp. 4–. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2506-7. doi:10.1109/ICVS.2006.49. Cited on page 12.

[231] S. Würmlin, E. Lamboray, O. G. Staadt, and M. H. Gross. 3d video recorder: a system for recording and playing free-viewpoint video. *Computer Graphics Forum*, vol. 22, pp. 10–22, 2003. Cited on page 17.

[232] F. Xu, Y. Liu, C. Stoll, J. Tompkin, G. Bharaj, Q. Dai, H.-P. Seidel, J. Kautz, and C. Theobalt. Video-based characters: creating new human performances from a multi-view video database. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pp. 32:1–32:10. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0943-1. doi:10.1145/1964921.1964927. Cited on page 11.

[233] K. Yamauchi, H. Kameshima, H. Saito, and Y. Sato. 3d reconstruction of a human body from multiple viewpoints. In *Proceedings of the 2nd Pacific Rim conference on Advances in image and video technology*, PSIVT'07, pp. 439–448. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 3-540-77128-X, 978-3-540-77128-9. Cited on page 11.

[234] R. Yang, G. Welch, and G. Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*, pp. 225 – 234, 2002. doi:10.1109/PCCGA.2002.1167864. Cited on page 14.

[235] S. Yous, H. Laga, M. Kidode, and K. Chihara. Gpu-based shape from silhouettes. In *proceedings of GRAPHITE '07*, pp. 71–77. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-912-8. doi:http://doi.acm.org/10.1145/1321261.1321274. Cited on page 13.

[236] Z. Yue, L. Zhao, and R. Chellappa. View synthesis of articulating humans using visual hull. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pp. 489–492. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7803-7965-9. Cited on page 18.

[237] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, 2000. doi:10.1109/34.888718. Cited on page 25.

[238] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, vol. 23, no. 3, pp. 600–608, 2004. doi:http://doi.acm.org/10.1145/1015706.1015766. Cited on page 16.