Hasselt University
Transnationale Universiteit Limburg

Real-time, Adaptive Plane Sweeping for Free Viewpoint Navigation in Soccer Scenes

---

Dissertation submitted for the degree of

Doctor of Philosophy in Computer Science

at Hasselt University

defended by

Patrik Goorts

on June 16th, 2014 (TBD)

Promotor: Prof. Dr. ir. Gauthier Lafruit

Co-promotor: Prof. Dr. Philippe Bekaert

2010 – 2014

This is edition 1.2 of 2014-09-18
Changes can be found in chapter B.9.



This dissertation is supported by a bursary of the Flemish Organization IWT (Het agentschap voor Innovatie door Wetenschap en Technologie).

Abstract

In this dissertation, we present a system to generate a novel viewpoint using a virtual camera, specifically for soccer scenes. We demonstrate the applicability for following players, freezing the scene, generating 3D images, et cetera. The method is demonstrated and investigated for 2 camera arrangements, i.e. a curved and a linear setup, where the distance between the cameras can be up to 10 meters. The virtual camera should be located on a position between the real camera positions. The method is designed to be automatic and has high quality results using high performance rendering.

We presented an image-based method to generate the novel viewpoints based on the well-known plane sweep approach. The method consists of a preparation phase and a rendering phase.

In the preparation phase, geometric calibration is performed. Here, we presented a calibration system for large setups using the images of the recordings itself. No specific objects must be placed in the scene, but this is nevertheless possible. We applied feature detection on the input streams and match features between pairs of cameras. We present a method based on graphs that select multicamera feature matches using a voting mechanism. Furthermore, the matches are filtered based on the general direction in which the features appear to move across the different cameras, which is a robust outlier detection. These filtered multicamera feature matches are then used to generate the calibration data. The results demonstrate the quality of the calibration, which is sufficiently high for our method. Due to the automatic nature of the calibration method, we have achieved a convenient and practical solution for multicamera calibration in large scenes.

Once the calibration is known, we can start rendering. We demonstrate that normal plane sweeping is not sufficient for soccer scenes due to the high number of artifacts, such as ghost legs, ghost players, and halo effects. Therefore, we propose a depth-aware plane sweep approach. We have shown that the depth values of the artifacts differ from the depth values of the players. This can be used to filter out the artifacts. We determine the initial depth using a plane sweep approach. Next, we filter the depth map using a median-based or histogram-based approach, where each group of pixels is processes independently. The depth is furthermore compared to the depth of the background, eliminating ghost player artifacts.

The results show that the artifacts are effectively eliminated in most cases. We employed modern and traditional GPGPU technologies for the complete processing pipeline to develop a scalable and fast solution. The performance is higher than a few frames per second for a single GPU and HD resolution, which makes it practical and affordable to scale up to a real-time solution. The results are visually compared to existing systems, which demonstrates that our method can eliminate many artifacts visible in other systems.

Furthermore, a novel plane distribution method is developed to assign more processing power to the depths where there actually are objects and to reduce wasted processing power on empty space. The quality is checked qualitative and it is demonstrated that the difference between a high number of planes and a redistributed low number of planes is negligible, and the difference between a uniformly distributed low number of planes and a redistributed low number of planes is significant. This shows the usefulness of the optimization by reducing the required processing power, while keeping quality levels comparable.

Acknowledgments

Writing a doctoral dissertation is an individual project. Nevertheless, this would have been impossible without the help and support of numerous people. Here, I would like to thank everybody who helped me, directly or indirectly, with achieving this milestone in my life.

First of all, I would like to thank my promotor Prof. Dr. ir. Gauthier Lafruit, who guided me in concluding my work, and helped me improve the results and this text with extensive and valuable feedback. I greatly appreciate his effort in everything he did for me.

Next, I would like to express my gratitude to Prof. Dr. Philippe Bekaert, who gave me the opportunity to start my PhD. His effort and guidance helped me greatly, and his insightful remarks, suggestions, and intellectual inspiration were invaluable for this work. Furthermore, his effort to arrange all financial and material matters in our research group was a great support.

I would like to thank Dr. Ing. Sammy Rogmans, who guided my internship before my PhD and guided me into research. He was a great aid in acquiring my bursary, and in improving any publication made during my PhD. Without him, I wouldn't be where I am now.

Many people helped me with getting the results that are presented. Special thanks to Maarten Dumont, Steven Maesen, and Tom Haber, whose research was the foundation of my work, and therefore avoiding to start from nothing. Furthermore, I would like to thank all of my colleagues, those who are still around and those who already left, for all the big and small things that they did to support this work.

I am grateful for the people who provided helpful feedback on my text. Christophe De Vleeschouwer, for his suggestions on content, and Reinout Buysse for bringing the linguistical aspects of my text to the next level.

Furthermore, I would like to express my gratitude to my complete PhD jury for their feedback, support, and effort in evaluating my work: Prof. Dr. ir. Gauthier Lafruit, Prof. Dr. Philippe Bekaert, Prof. Dr. Wim Lamotte, Prof. Dr. Frank Van Reeth, Prof. Dr. Oliver Grau, Prof. Dr. Christophe De Vleeschouwer, Dr. Ing. Sammy Rogmans, and Prof. Dr. Marc Gyssens.

A special thanks is in order for the people who worked with me to acquire the input data. It is not trivial to set up a working camera network in a soccer stadium. Therefore, I would

like to thank the following people for carrying heavy material up and down stairs and ladders, for placing and securing cables and cameras, for shipping equipment, for securing access to the stadiums, and for being active and awake for many days in a row: the supervisor of the recordings Prof. Dr. Philippe Bekaert; my colleagues Sammy Rogmans, Steven Maesen, Maarten Dumont, Tom Haber, Johannes Taelman, Cosmin Ancuti, Tom Mertens, Bert de Decker, Cedric Vanaken, and Karel Robert; the team members of Media Pro Spain: Jordi Ramon Cipres, Jordi Alonso, and Sergi Sagas Rica; and the team members of Tracab Sweden: Eric Hayman and Gareth Loy.

The results presented here were integrated in the FINE project, which aimed to bring novel soccer broadcasting technologies to the living room. View interpolation is a piece of this puzzle. I would like to express my gratitude to the whole FINE consortium for the support and coordination of this project. Thanks to all of them, a successful project was completed, which allowed me to make my work a part of the big picture. My work is now not a small piece of research, but a meaningful part of a complete system. I would like to thank especially Michael Bastings, Olivier Barnich, and Philippe Latour from EVS Belgium for working with me on the integration and on creating powerful demonstrations.

Furthermore, I would like to thank my colleagues for the nice projects we did besides this work. A special thanks to Steven Maesen, Lode Vanacken, Sofie Notelaers, and Tom De Weyer for the fun projects of the 3DUI contests.

Doing a PhD is not just working. I would like to thank all my colleagues to make the road worthwhile by being there for all the movie nights, the barbecues, the game nights, the dinners, and the always pleasant lunch breaks.

I would like to express my gratitude to Ingrid Konings, Roger Claes, Luc Adriaens, Peter Quax, Tom De Weyer, Wim Lamotte, Edith Cloes, Frank Van Reeth, Peter Vandoren, and Eddy Flerackers for the technical, administrative, and managerial support provided during my time at the EDM. Their effort provided a stable working environment where we could focus on our research.

And last, but not least, I would like to thank my family and friends for their neverending support. A special thanks to my parents Gert Goorts and Ingrid Martens, and my brother Jeroen for giving me all the opportunities in my life, including this dissertation.

Thank you,

Patrik Goorts

Contents

Abstract i

Acknowledgments iii

Contents v

List of Figures ix

List of Tables xiii

1 Introduction 1
- 1.1 Use Cases . . . 4
  - 1.1.1 Producers . . . 6
  - 1.1.2 Consumers . . . 7
- 1.2 Design Considerations . . . 7
- 1.3 Contributions . . . 8
- 1.4 Dissertation Overview . . . 9

2 System Overview 11
- 2.1 System Overview . . . 13
- 2.2 Captation . . . 13
  - 2.2.1 Scene description . . . 13
  - 2.2.2 Camera Location and Orientation . . . 14
  - 2.2.3 Recording Requirements . . . 15
  - 2.2.4 Storage Requirements . . . 17
- 2.3 Setting up the system . . . 17
- 2.4 Choosing the Virtual Camera Position . . . 19
- 2.5 Rendering . . . 21
  - 2.5.1 Foreground rendering . . . 21
    - 2.5.1.1 Projective Geometry . . . 22

2.5.1.2 Plane Sweeping Concept . . . 24
2.5.1.3 Plane Sweeping with Depth Selection . . . 24
2.5.2 Background Rendering . . . 25
2.6 Conclusions . . . 25

3 State of the Art: Image Interpolation 31
3.1 General View Interpolation . . . 33
3.1.1 Explicit Geometry Information . . . 33
3.1.2 Depth Information . . . 34
3.1.3 No Geometry Information . . . 36
3.2 View Interpolation for Soccer Scenes and Other Sports . . . 37
3.3 Conclusions . . . 40

4 State of the Art: GPU technologies 41
4.1 Traditional GPU Technologies . . . 43
4.1.1 The Programmable Graphics Pipeline . . . 43
4.1.2 General Use of the Graphics Pipeline . . . 45
4.2 Modern GPU Technologies: CUDA . . . 45
4.2.1 Execution Model . . . 46
4.2.2 Architectural Model . . . 48
4.3 Conclusions . . . 50

5 Image Data Preparation 51
5.1 Calibration . . . 53
5.1.1 Representation of Camera Parameters . . . 54
5.1.2 Determination of Camera Parameters . . . 56
5.1.2.1 Determination of 2D Image Correspondences . . . 56
5.1.2.2 Angle-based 2D Image Correspondences Filtering . . . 59
5.1.2.3 Correspondences to Projection Matrices . . . 60
5.1.2.4 Decomposition of Projection Matrices . . . 61
5.2 Debayering . . . 62
5.2.1 FIR Filtering for Demosaicing . . . 65
5.2.2 Generic FIR Filtering using CUDA . . . 65
5.2.2.1 Conventional Method . . . 66
5.2.2.2 Filter Separation . . . 69
5.2.2.3 Fourier Transformation . . . 70
5.2.2.4 Experimental Results . . . 70
5.2.3 FIR Filtering for Demosaicing using CUDA . . . 72
5.2.3.1 Compute Capability 1.1 . . . 74
5.2.3.2 Compute Capability 2.0 . . . 75
5.2.3.3 Compute Capability 3.5 . . . 76

5.2.3.4 Conclusions . . . 76
5.3 Segmentation . . . 79
5.3.1 Background Generation . . . 79
5.3.2 Foreground/background Separation . . . 79
5.4 Conclusions . . . 80

6 View Interpolation **81**
6.1 Background Rendering . . . 85
6.2 Foreground Generation, phase 1: Plane Sweeping . . . 89
6.2.1 Applying Plane Sweeping to Soccer Scenes . . . 90
6.2.2 Soccer-specific Plane Sweep Considerations . . . 91
6.2.2.1 Aggregation Windows . . . 92
6.2.2.2 Number of Cameras Used . . . 93
6.3 Foreground Generation, phase 2: Depth Filtering . . . 94
6.3.1 Connected Components . . . 96
6.3.2 Median-based Depth Selection . . . 99
6.3.3 Histogram-based Depth Selection . . . 99
6.3.4 Background Depth Filtering . . . 101
6.4 Foreground Generation, phase 3: Depth-selective Plane Sweeping . . . 101
6.5 Merging Foreground and Background . . . 103
6.6 Conclusions . . . 103

7 Plane Distribution Optimization **105**
7.1 Adaptive Non-Uniform Plane Distribution . . . 107
7.2 Results . . . 110
7.3 Conclusions . . . 112

8 Results **121**
8.1 Final Result of the Method . . . 123
8.1.1 Comparison with Traditional Stereo Methods . . . 127
8.1.2 Location of the Virtual Camera . . . 127
8.2 Comparison with Existing Systems . . . 147
8.3 Performance of the Components . . . 159

9 Conclusions and Future Work **167**
9.1 Conclusions of this Dissertation . . . 169
9.2 Future Work . . . 171

A Recordings **173**
A.1 The Genk Dataset . . . 175
A.2 The Barcelona Dataset . . . 181

B Proof of Convolution using SVD Decomposition **187**
B.1 Notations . . . 189
B.2 Definitions . . . 189
B.3 Theorem . . . 190
B.4 Proof . . . 190
B.5 Separable Filters . . . 194

List of Symbols **195**

Nederlandse Samenvatting **199**

Scientific Contributions and Publications **201**
B.6 Related Scientific Publications . . . 203
B.7 Related Student Theses . . . 204
B.8 Other Public Dissemination . . . 204
B.9 Unrelated Scientific Publications . . . 207

Errata and Additions **209**
B.10 Edition 1.1 . . . 209
B.11 Edition 1.2 . . . 209

Bibliography **228**

List of Figures

1.1 The bullet time effect. 5
1.2 Capturing 3D soccer games using 2 broadcasting cameras. 6

2.1 Overview of our setup. 13
2.2 Possible camera arrangements 14
2.3 Example of the rolling shutter effect 16
2.4 Overview of the possible virtual camera paths for the curved arrangement. . 18
2.5 Overview of the possible virtual camera paths for the linear arrangement. . . 18
2.6 Overview of our method for the rendering. 22
2.7 Overview of our method for foreground interpolation. 23
2.8 Concept of plane sweeping 26
2.9 Concept of plane sweeping 27
2.10 Concept of plane sweeping 28
2.11 Concept of plane sweeping 29

4.1 The GPU Pipeline 44
4.2 Mapping an image to CUDA threads and blocks. 47
4.3 Abstracted CUDA hardware. 49

5.1 Overview of our method 53
5.2 The projective camera model 54
5.3 Intrinsic and extrinsic camera matrices explained 55
5.4 Example of feature detection using 3 cameras 58
5.5 The graph used in our calibration example. 58
5.6 Multicamera feature matches, considered as inliers. 60
5.7 Multicamera feature matches, considered as outliers. 60
5.8 The Bayer filter concept. 62
5.9 Comparison of debayering using bilinear interpolation and using FIR filtering. 63
5.10 Convolution filters for demosaicing. 64

5.11 Implementation strategies for conventional FIR filtering on parallel SIMT architectures. . . . 66
5.12 One block for filtering a part of the image, including the apron. . . . 67
5.13 Distribution of apron and image threads for a filter with radius 5, traditional and optimal case. . . . 68
5.14 Results of different FIR filtering approaches. . . . 72
5.15 Linearization of trade-offs in FIR filtering. . . . 73
5.16 Performance results for debayering using compute capability 1.1. . . . 74
5.17 Performance results for debayering using compute capability 2.0. . . . 75
5.18 Performance results for debayering using compute capability 3.5. . . . 76
5.19 Results of the foreground/background segmentation process. . . . 78

6.1 Overview of our method. . . . 83
6.2 Comparing results with and without shadow processing. . . . 84
6.3 Backprojection of camera backgrounds individually. . . . 86
6.4 Comparison of considering the goal as background or foreground. . . . 87
6.5 Principle of plane sweeping. . . . 88
6.6 Plane sweeping on soccer scenes using conventional plane sweeping without segmentation. . . . 90
6.7 Plane sweeping on soccer scenes using conventional plane sweeping with segmentation, detailed view. . . . 91
6.8 Plane sweeping on soccer scenes using conventional plane sweeping with segmentation. . . . 92
6.9 Comparison between using 2 or all cameras for the curved arrangement . . . 94
6.10 Comparison of using the segmentation as guideline . . . 95
6.11 Detection of connected pixels: stepwise overview . . . 96
6.12 Different iterations of the connected components algorithm, concrete example. 97
6.13 The unfiltered histogram for the histogram-based depth selection . . . 98
6.14 The filtered histogram for the histogram-based depth selection . . . 98
6.15 Comparison of the median-based and histogram-based method . . . 100
6.16 Artifact elimination based on background depth . . . 102
6.17 Detailed results of our method. . . . 103

7.1 Uniform plane distribution, with histogram. . . . 107
7.2 Depth histogram with the cumulative representation. . . . 108
7.3 Detail of the cumulative histogram with discrete values. . . . 109
7.4 Redistributed depth planes. . . . 110
7.5 Depth map with a uniform depth plane distribution (50 planes). . . . 113
7.6 Depth map with a non-uniform depth plane distribution (50 planes). . . . 114
7.7 Depth map with a uniform depth plane distribution (256 planes). . . . 115
7.8 Plane redistribution in an artificial setup, 1 person. . . . 116

7.9 Plane redistribution in an artificial setup, 2 persons. 117
7.10 Detailed quality comparison of a soccer scene. 118
7.11 Plane sweeping of a soccer scene with a uniform plane distribution (40 planes). 118
7.12 Plane sweeping of a soccer scene with an adaptive plane distribution (40 planes). 119
7.13 Plane sweeping of a soccer scene with a uniform plane distribution (5000 planes). 119

8.1 Input for the Barcelona dataset 128
8.2 Result for the Barcelona dataset, histogram-based depth filtering 129
8.3 Result for the Barcelona dataset, median-based depth filtering 130
8.4 Result for the Barcelona dataset, median-based depth filtering, varying parameters 131
8.5 Input for the Genk dataset, large field of view 132
8.6 Result for the Genk dataset, histogram-based depth filtering 133
8.7 Result for the Genk dataset, median-based depth filtering 134
8.8 Result for the Genk dataset, median-based depth filtering 135
8.9 Input for the Genk dataset, small field of view 136
8.10 Result for the Genk dataset, histogram-based depth filtering 137
8.11 Result for the Genk dataset, median-based depth filtering 138
8.12 Anaglyph demonstrating the application to create 3D images. 139
8.13 Result for the Barcelona dataset, where artifacts cannot be removed 140
8.14 Input for stereo matching on soccer scenes 141
8.15 Result for stereo matching on soccer scenes 142
8.16 Result for stereo matching on soccer scenes 143
8.17 Result for virtual camera positions outside the optimal region 144
8.18 Result for virtual camera positions outside the optimal region 145
8.19 The system of Liberovision 146
8.20 The system of Liberovision 148
8.21 The system of Liberovision 149
8.22 The system of Germann et al. [2010] 150
8.23 The system of Germann et al. [2010] 151
8.24 The system of Ohta et al. [2007] 152
8.25 The system of Ohta et al. [2007] 153
8.26 The system of Inamoto and Saito [2007b] 154
8.27 The Piero system [BBC, 2012] 155
8.28 The Piero system [BBC, 2011] 156
8.29 The Piero system [BBC, 2011] 157
8.30 The Piero system [BBC, 2011] 158
8.31 Timings per component, shown per number of planes, for the median-based approach. 161

8.32 Timings broken up in components, median-based approach, 512 planes. . . . 161
8.33 Timings broken up in components, median-based approach, 256 planes. . . . 162
8.34 Timings broken up in components, median-based approach, 128 planes. . . . 162
8.35 Timings broken up in components, median-based approach, 64 planes. . . . 163
8.36 Timings per component, shown per number of planes, for the histogram-based approach. 163
8.37 Timings broken up in components, histogram-based approach, 512 planes. . 164
8.38 Timings broken up in components, histogram-based approach, 256 planes. . . 164
8.39 Timings broken up in components, histogram-based approach, 128 planes. . . 165
8.40 Timings broken up in components, histogram-based approach, 64 planes. . . 165

A.1 Scheme of the Genk setup. 175
A.2 Photos of the Genk recording setup 176
A.3 Example of the Genk dataset, $12.5mm$ 177
A.4 Example of the Genk dataset, $12.5mm$ 178
A.5 Example of the Genk dataset, $25mm$ 179
A.6 Example of the Genk dataset, $25mm$ 180
A.7 Photos of the Barcelona recording setup 182
A.8 Example of the Barcelona dataset 183
A.9 Example of the Barcelona dataset 184
A.10 Example of the Barcelona dataset 185
A.11 Example of the Barcelona dataset 186

B.1 Public setup demonstrations. 205
B.2 Public setup demonstrations. 206

List of Tables

7.1 Error metrics for the difference between 5000 and 40 planes, and between 5000 and 40 planes with adaptive plane distribution. 112

8.1 Overview of the results of our method 124
8.2 Timings for the median-based depth filtering approach 160
8.3 Timings for the histogram-based depth filtering approach 160

Chapter 1

Introduction

1.1 Use Cases 4
1.1.1 Producers 6
1.1.2 Consumers 7
1.2 Design Considerations 7
1.3 Contributions 8
1.4 Dissertation Overview 9

In the current multimedia landscape, entertainment in the living room is more important than ever. With the advent of high definition and 3D television, fast content delivery networks, interactive gaming consoles, and impressive graphical effects, home entertainment is truly an upcoming aspect of everyday life. However, in spite of all the advancing technologies, sports broadcasting is still a traditional field. A game is played, and the recordings of cameras on site are broadcasted to the end user. Not so much interaction and photo-realistic special effects are used.

We can, however, introduce computer vision and rendering technologies in sports broadcasting. This dissertation will discuss one advancement in the broadcasting of soccer games. More precisely, we will present a system where a virtual, i.e. non-existing, camera can be used. This virtual camera is not limited by the locations of the real cameras used to record the scene, and will therefore decouple the viewpoint from the actual recording hardware. This technology is referred to as *free viewpoint rendering*.

No on-site operator is required. Instead, only the images of the static, real cameras are used to generate the photo-realistic image of an arbitrary placed virtual camera. Thanks to the use of recorded images, going back in time and freezing the action belong to the possibilities. The use of animation and modeling is avoided to provide a photo-realistic effect. The use of manual intervention is eliminated in the rendering phase to allow a fully automatic system. This makes our system usable in practice.

A virtual camera can be used by both professional and home users. The professional user, i.e. the broadcaster, can use a virtual camera to generate novel viewpoints and enhance the viewing experience; the choice of the broadcasted video stream is no longer limited to the real camera images. Alternatively, the home user can also choose his own camera viewpoint, and as a result create an interactive an enriched viewing experience.

The method we describe is one part of the final picture, and only a part of the use cases will be demonstrated. We will demonstrate a system for intermediate camera positions, i.e. virtual camera positions that are placed in between the real camera positions. This way, some assumptions can be made, which will make the system more performant and generates higher quality results. In chapter 3, we will discuss existing research on arbitrary virtual camera positions. By limiting the virtual camera position to intermediate camera positions, issues present in alternative systems are addressed and avoided.

To generate high quality results, i.e. have the same quality as the input images, we will use image-based rendering technologies. Image-based methods only use the images to create a novel viewpoint without 3D scenes. This way, we avoid a typically used 3D reconstruction step, which may introduce difficult to address artifacts and resolution loss. By using an image-based rendering approach, however, we cannot use a full 3D reconstructed scene. We do not create a full scene where we can place a virtual camera in. This will reduce the possible virtual camera positions. Because many systems described in chapter 3 use 3D reconstruction or basic image-based rendering, we will provide a novel way of performing view interpolation in soccer scenes, and therefore increase the number of available options. While our method is

based on previous research in image-based rendering, many properties of soccer scenes make the generic solution unusable. As a result, research was required for a fast and high quality solution. What these properties are is discussed in chapter 2.

1.1 Use Cases

In this section, we will discuss the scenarios where virtual cameras can be used, and by whom. In section 1.2, we will discuss which of these scenarios we will provide a solution for. Due to the expanded field of applications and the high engineering aspects of some scenarios, focus is mandatory. This explains why we will not discuss all possible scenarios.

Traditionally, soccer scenes are recorded and broadcasted using static pan-tilt cameras, or cameras with a limited range of movement. Many camera systems, such as cameras on rails, are avoided or limited in soccer stadiums due to safety aspects and obstructions by structures and people. This way, a lot of camera viewpoints or movements are not possible. Many good viewpoints of action scenes may be missed.

Broadcasting of sports entertainment, however, is an important aspect of the current society. This implies that the need for novel and improved technologies is desirable to attain the attention of the television spectators. Virtual camera positions can help to achieve a better sports broadcasting and viewing experience. Many novel, previously impossible scenarios can be considered. First, current television broadcasting methods using static cameras can be enhanced. When showing the scene from one angle, i.e. a camera viewpoint, and switching to another viewpoint, a virtual camera can move from the first camera to the second. This way, spatial context is preserved and no abrupt viewpoint changes are perceived. It looks as if the spectator himself is moving from one view to the next, enhancing the viewing experience. Furthermore, extra camera positions can be considered. Nowadays, most cameras are placed at the same side of the pitch. This holds especially for overview cameras. A camera at the other side, called a reverse-angle camera, is not commonly used [FIFA, 2004; Hilton et al., 2011; UEFA, 2013]. Using only one side of the pitch will help the spectator to retain spatial context as the playing teams will not switch sides abruptly. In fact, it would be very confusing if a team is moving from left to right, and the viewpoint changes to the other side of the field, resulting in the reversal of the team movement on screen. When using a virtual camera to move gradually from one side of the field to the other, no abrupt viewpoint changes occur, and confusion is avoided. When applying a virtual camera, more real camera positions can therefore be considered.

Second, extra previously impossible options can be considered. For example, the scene can be frozen and shown from different angles. A virtual camera can be moved from one camera to the other, while the scene is not moving. This will result in a detailed overview of the situation (for example, right before a goal kick). This technique is already well-known as a special effect in the movie industry, known as the bullet time effect, or the matrix effect (see Figure 1.1). However, controlled studio conditions, hundreds of cameras, and human



Figure 1.1: The bullet time effect, as seen in the movie The Matrix. By utilizing a virtual camera, these effects can now be transferred to sports broadcasting. [Wachowski and Wachowski, 1999]

intervention are available when creating movies. This is not the case in soccer broadcasting. In the latter case, fast results are required for live broadcasting. Therefore, human intervention and long running rendering in post-production are not desirable. A fast, high quality, and automatic system is required.

Last, more than conventional video streams can be created. By the upcoming of 3D television, 3D video is required in the form of stereo video pairs. Stereo vision creates an illusion of a 3D scene using flat displays, increasing the feeling of immersion in the provided content. This can be accomplished using glasses technology, where each eye sees a different image due to filtering or blocking by the glasses, or by using autostereoscopic displays, where parallax barriers are used to provide the eyes with different images. Regardless of the technology used to display 3D content, stereo images are required. The use of virtual cameras can help in the generation of stereo content. By placing a virtual camera right next to a real camera, stereo images can be created, together with their depth information. This information can be fed to a 3D display, creating a 3D effect on real images, without the use of 3D cameras (such as shown in Figure 1.2).

All these previously mentioned applications can be provided by a virtual camera where its position is limited in between the real cameras. However, many other applications of virtual cameras in soccer scenes can be considered. For example, the virtual camera can be placed on the shoulder of a player, or on the position of the ball. However, much research must be conducted to create these kind of effects.



Figure 1.2: Capturing 3D soccer games using 2 broadcasting cameras. These setups can be avoided by placing a virtual camera next to the real camera, and obtaining a second view for stereoscopic display devices. [Sony, 2013]

All of the above applications can be used by different users. In the following paragraphs, we will consider 2 kinds of users: producers and consumers.

1.1.1 Producers

The producer, or professional user, is the person (or persons) responsible for creating a video stream to be broadcasted. Traditionally, only the video streams from the real cameras are used to create a final stream to be broadcasted. When allowing the generation of the streams of virtual cameras, more tools become available to the broadcaster. Therefore, more impressive results can be achieved. Freezing the scene or avoiding hops between camera positions, as discussed before, are useful tools. Furthermore, a virtual camera can be used to follow a specific player from one side of the field to the other to keep him in the center of the image, while keeping the view angle constant. A virtual camera can also be used to provide the best angle for action shots in replays, removing the limitations of fixed locations.

When providing the stream of a live game, rendering should be efficient. Broadcasters cannot wait before a video fragment is ready to be broadcasted. Therefore, performant and scalable rendering is one of our design considerations. This is especially important when using the virtual camera to generate 3D content. Furthermore, the quality should be high to provide results worth broadcasting.

Even after the game has finished, the recorded images can be used to create virtual viewpoints. Typically, analysis of the game is provided to the spectators, or used by the soccer

team to prepare for future games. A virtual camera aids in the generation of viewpoints useful for these analyses. The game can be replayed and frozen, and it can be investigated thoroughly. Combined with traditional analysis tools, such as player and ball tracking, automatic statistics generation, animation of tactics, et cetera, free viewpoint video can be useful for more in-depth analysis. While these analyses are done after the game and have a less strict performance requirement, time considerations are still important here. Furthermore, quality is important to make analysis useful.

1.1.2 Consumers

The home user is the final consumer of the provided content. If the consumer is passive, all content creation is done by the professional user.

However, the home user can also play an active role in the content processing. For example, the user can choose his own viewpoint, virtual or not, predefined or not. The user can choose to freeze the scene and have a look around before continuing the game, or choose his own replays from its own chosen position. The user can choose to follow a player or the ball, in the way that a virtual camera will move automatically based on the movements of that player or ball.

This will move the spectator form a passive role to an active role, changing the paradigm of home sports entertainment.

These possibilities are not limited to television sets. A tablet or a laptop can be used during the game to provide a second screen experience or an interface to control the stream as seen on television.

Alternatively, the consumer can be located in the stadium itself. This way, viewpoints from the other side of the field can be streamed to a mobile phone, providing novel viewpoints of, for example, action at the other side of the field.

While the rendering of the image of a virtual camera is important for these scenarios, many considerations must be made regarding network capacity, processing power at home and storage possibility. These are not in the scope of this dissertation.

1.2 Design Considerations

Considering the previously discussed cases, some design considerations have to be made. We will only consider virtual camera positions between the real camera positions. This way, image-based rendering technologies, as described in chapter 3, can be used. Allowing only positions between real camera positions is still a valid constraint: frozen frames, following action, generating stereo video pairs, et cetera, are all possible. When deciding for trade-offs and technology directions, we use two focus points.

First, the system should be of high quality. More specifically, we aim to provide the same resolution, sharpness, and image quality in general as the input video streams and avoid any

animation technologies. This will allow the integration in the real video stream, without a noticeable effect for the viewer.

Second, the system should be performant and affordable at the same time. This is not a trivial consideration. To allow the generation of the image of a virtual camera, algorithms are used that are relatively slow due to the large data processing and complex computations. Indeed, multiple camera images are used to generate one, final image. Therefore, we employ modern commodity GPU processing units to generate the resulting images. This way, parallel image processing is possible, at a cost of a less flexible architecture. We chose commodity GPUs to make the method scalable, relatively cheap, and performant. To avoid the creation of a method that is later optimized for GPU, the method is designed from the start to be completely used on GPU. Therefore, GPU technology considerations, as discussed in chapter 4, are used throughout the complete chain, resulting in different design methods and considerations than a more classical CPU approach. As a result, GPU methodologies are used throughout each technical chapter, and no optimization chapter is considered.

It should be noted that a number of parameters can be used to create a trade-off between quality and performance. By cutting a few corners and reducing the amount of processing, quality can be traded for performance.

1.3 Contributions

In this section, we will give a short overview of the main contributions.

1. We develop a novel system for virtual camera rendering, tailored to soccer scenes. A virtual camera can be moved in between real, static cameras, and the image is generated for that virtual camera. The resulting image is of similar quality as the input camera images for most of the possible scenes, thanks to a depth-based artifact filtering approach. The system is fully automatic, scalable, and fast in processing time thanks to the use of GPU computing. Furthermore, the required setup for capturing the soccer scene is determined and analyzed.

2. We develop a novel approach for calibrating a large-scale and static camera network with a high number of cameras, where the cameras can be placed far apart from or close to each other. No calibration objects need to be placed in the scene. The method determines correspondences between pairs of images using existing feature detection algorithms. Next, multicamera matches are determined and filtered using a consensus-based voting mechanism and a similarity measurement. The result is a set of reliable multicamera matches that can be fed into existing camera calibration toolboxes.

3. We investigate finite impulse response filtering using GPUs and determined the most optimal approaches, based on filter sizes and targeted devices. We used the conclusions to perform debayering using the GPU in an efficient way.

4. We develop a novel approach to redistribute depth hypothesis testing in the well-known plane sweep approach. By eliminating some depths to be tested based on previously calculated depth maps, performance is increased and artifacts are reduced. The approach is not only applicable to soccer scenes, but can be used in general.

1.4 Dissertation Overview

In this section, we will give an overview of this dissertation.

The introduction (this chapter) described the main applications of our method and the possible use cases. Furthermore, we give an overview of the properties, the design considerations, and the method of operation. The main contributions of this dissertation are summarized at the end of the chapter.

Chapter 2 gives a technical overview of the system. We discuss the acquisition and storage of the input data and the location of the cameras. Next, we discuss the steps required to get the system up and running. Finally, we discuss the required input and the method of generating the novel viewpoints. To help in the understanding of the method, we provide an introduction to projective geometry. For a more mathematical discussion, we refer to chapter 5.

Chapter 3 gives an overview of existing systems for novel viewpoint generation, and we focus further on view interpolation for soccer scenes.

Chapter 4 discusses traditional and modern GPU technologies for generic parallel computing. We discuss the use of graphical computing shaders for general use, and the modern CUDA technology.

Chapter 5 gives an overview of all the steps that need to be done before it is possible to generate novel viewpoints. We discuss the method to calibrate the cameras geometrically based on feature correspondences. Next, we discuss debayering, which will convert the raw data provided by the cameras to RGB images. We use a FIR filtering approach, and we investigate the most optimal approach to perform FIR filtering on GPU. Finally, we discuss segmentation, which is based on previously generated backgrounds. Some of these steps must be done only once per soccer game, and some are done before every rendered frame.

Chapter 6 describes the actual view interpolation method, where the novel viewpoints are rendered. The foreground and background are rendered independently. We first describe the background rendering, and we continue with the discussion of the foreground rendering. The foreground rendering is done in 3 phases: an initial depth estimation, a depth filtering to reduce artifacts, and a final depth-aware image rendering.

Chapter 7 discusses an optimization of the plane sweep method, used in the foreground rendering. Conventional plane sweeping uses different depth hypotheses to find objects in the scene. It is, however, possible to eliminate some depth hypotheses if there were no objects in the previous temporal frame. We move computational power to the places where there are most likely objects, and therefore increase performance and quality.

Chapter 8 provides the results of our method. We first discuss the depth and color results for different scenes. Next, we compare our system to other, existing systems. Finally, we discuss the performance.

Chapter 9 gives the conclusions of our work, and provide some future steps that can be taken to advance the research.

Appendix A describes the recordings used for our results. Appendix B gives the mathematical proof of the general FIR filter decomposition, used in chapter 5.

We finish with a list of used symbols, a dutch summary, and an overview of my publications.

Chapter 2

System Overview

2.1 System Overview 13
2.2 Captation 13
2.2.1 Scene description 13
2.2.2 Camera Location and Orientation 14
2.2.3 Recording Requirements 15
2.2.4 Storage Requirements 17
2.3 Setting up the system 17
2.4 Choosing the Virtual Camera Position 19
2.5 Rendering 21
2.5.1 Foreground rendering 21
2.5.1.1 Projective Geometry 22
2.5.1.2 Plane Sweeping Concept 24
2.5.1.3 Plane Sweeping with Depth Selection 24
2.5.2 Background Rendering 25
2.6 Conclusions 25

In this chapter, we will provide an overview of the system. First, we will discuss the different components of the system. Next, we will elaborate on the components and discuss the captation, the generation of a viewpath, and the rendering.

2.1 System Overview

Our setup, depicted in Figure 2.1, consists of a number of cameras with a static location and orientation, aimed at the pitch. All images captured by the cameras are transferred to a storage server, where all the data is stored and synchronization is preserved. A render computer, the renderer, can there access all required images to generate a novel viewpoint.

The renderer takes in a viewpath, i.e. a spatiotemporal camera path, chosen by a human user. The renderer subsequently fetches the required images from the storage server and passes them to the rendering software. After the rendering, the results are pushed back to the storage server, where they can be fetched for display and/or broadcasting.

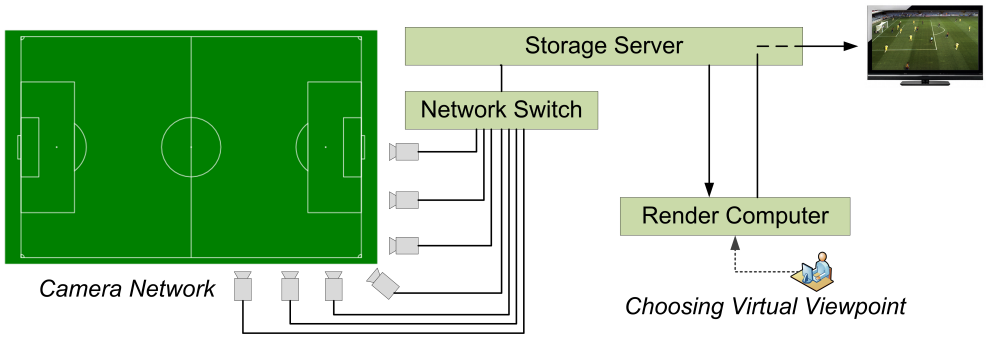


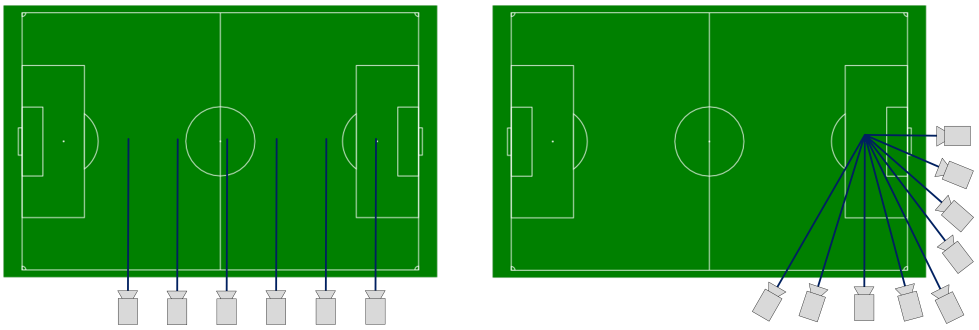
Figure 2.1: Overview of our setup. The setup consists of a camera network, connected to a storage server. The rendering module can fetch any image required to generate a novel viewpoint. The novel images are stored on the storage server for further distribution.

2.2 Captation

In this section, we will discuss the input requirements of the method. As stated before, the input of our method is an array of static cameras, placed around and aimed at the pitch. The technical details of the recordings used in this dissertation are provided in appendix A.

2.2.1 Scene description

We do not aim to solve general free viewpoint interpolation. We will, therefore, limit ourselves to specific scenes.



(a) Linear arrangement (b) Curved arrangement

Figure 2.2: Two possible camera arrangements for soccer scenes. Both arrangements have different applications and different properties. (a) In the linear arrangement, all cameras are placed on a line next to the long side of the pitch. All cameras have the same look-at angle. (b) In the curved arrangement, all cameras are placed around a corner of the pitch. All cameras point to a spot in the scene.

The scene considered to be recorded, and interpolated, is an outdoor soccer pitch. The pitch itself is typically around $50m$ by $100m$, and is more or less flat. However, it is not flat enough to consider the pitch as a calibration plane. The pitch as recorded in Barcelona, for example, (see appendix A) is about $30cm$ lower at the corners, compared to the middle of the pitch.

No presumptions are made about the color of the pitch, except that it differs significantly from the color of the players on it. The complete scene, including pitch and players, is considered Lambertian [Lambert and Anding, 1760], i.e. no mirroring surfaces, refraction, et cetera. The scene is well exposed by a distinct and possibly changing light source, being the sun or the stadium lights. This may result in very distinct shadows, which must be visible in the novel images.

The scene is considered very dynamic, with fast, uncontrollable, and unpredictable movements of the players.

2.2.2 Camera Location and Orientation

We considered 2 possible arrangements for the cameras: a linear arrangement and a curved arrangement. These are shown in Figure 2.2. In both arrangements, the cameras are placed around the pitch at a certain height to allow an overview of the scene. Examples of the recordings can be found in appendix A.

In the linear arrangement, as seen in Figure 2.2(a), the cameras are placed along one side of the pitch, and all cameras have the same look-at angle. This arrangement may be used to follow the players when they are moving along the field, or to move the virtual camera from

one broadcasting camera to the other. The field of view can be chosen, appropriate to the desired application. If a large field of view is used, an overview of the scene is available, and the virtual camera can be generally used. If a small field of view is used, detail of a portion of the pitch is available. A small field of view can only be used if the action is inside that portion. Nevertheless, the virtual camera can move left and right and can therefore keep the action in the center of the video. This is different from a camera that has a fixed location and turns towards the action.

In the curved arrangement, as shown in Figure 2.2(b), the cameras are placed in an arc and are all pointing to a single spot on the pitch. The main application of this arrangement is to focus and investigate some action on that spot. The scene can be frozen, and the virtual camera can move around the scene and show it from different angles. One example is right before a goal kick. A large field of view is preferred in this case to keep context and to not limit the possible positions of the action in the scene too much. A large overlap between the cameras is less of a requirement, because there is a lot of information about the 3D location of the players due to the different viewpoints on the scene. This is in contrast with the linear arrangement, where all the cameras look at the scene from the same side. Therefore, the cameras can be placed further apart than in the linear case.

Both the curved and linear arrangement use cameras with a fixed location and orientation. We use fixed cameras for multiple reasons. First, we want to record as much as possible to allow replay. If the camera is moving along, no replay from a different viewpoint is possible. Second, no on-site operator moving the camera is required, reducing the cost, and therefore allowing the use of more cameras. Furthermore, the cameras can be placed at a location not easily accessible by humans, such as the ceiling of the stadium floors. Third, calibration is easier; no ad-hoc calibration is required. The complete calibration can be done beforehand. Fourth, motion blur is avoided, a serious problem when using computer vision technologies. This is especially true when the scene must be frozen and motion blur is visible on a static scene. Fifth, background can be calculated beforehand, based on multiple frames. This is more reliable than determining the background in a single frame.

2.2.3 Recording Requirements

We opted for computer vision cameras, but any camera providing color images at a decent resolution (such as 720p) should suffice. Computer vision cameras are smaller and cheaper than the cameras used in broadcasting, and provide many options to control the capturing. Although computer vision cameras don't have the quality of professional broadcasting material (in noise, light sensitivity, dynamic range, and color gamut size), the price and flexibility makes it an interesting option for demonstrating computer vision methods. Cameras have a wide range of settings, and the right settings must be made for a specific method. We will now discuss the considerations made for our method.



Figure 2.3: Example of the rolling shutter effect. Each line in the image is taken at a different time. This will result in artifacts if part of the scene is moving at a high speed. [Vision, 2013]

Because we are working with soccer scenes, where the movements may be large, a higher framerate will provide better results. Furthermore, the exposure time should be as low as possible to avoid motion blur.

The final images should be progressive and the camera should use a global shutter technology. This will provide an image where every pixel is taken at the same moment, and not a fraction of time later. If this is not the case, image deformation may occur due to the moving scene. An extreme example of a rolling shutter, i.e. a non-global shutter, can be seen in Figure 2.3. These artifacts may seriously disturb or impede further processing, adding even more artifacts [Grau and Pansiot, 2012].

Another consideration is the shutter synchronization of all the cameras in the array. When recording a soccer game, all cameras should take an image at the same time. If this is not the case, extra challenges appear for further processing. For example, the speed of the ball can reach up to $30m/s$ [Bray and Kerwin, 2003]. If one camera takes an image only $1ms$ later than the other, the ball moved already $3cm$. If these inconsistencies occur in every camera, depth estimation and image interpolation is made more difficult. These issues are avoided by providing a synchronization at shutter level. There is a pulse generator, connected to every camera, that will transfer the pulse to every camera at the same time. This pulse determines the moment the camera will take an image. This is in contrast with the Genlock system, where the output of cameras and other video sources is synchronized, and not the capture moments [Easterbrook et al., 2010; Robison, 1992].

The final image quality is further determined by the quality of the lenses. We opted for lenses with low radial distortion to avoid extreme compensation further on in the processing [Hartley and Zisserman, 2003, page 189]. Any distortion that is present in the used data is compensated for by the calibration step, as discussed in section 5.1.

Color calibration between the cameras is not a requirement for the view interpolation method, because the method is robust against small color errors. Nevertheless, color calibra-

tion will improve the final result when the virtual camera is moving. Therefore, we applied color calibration to some extent.

Lens focusing was achieved with the aid of the squared gradients [Sun et al., 2004], which was demonstrated by Grognard et al. [2012] to be a good metric for conventional images. Here, a metric is used to aid in manually focusing the lenses to achieve sharp images:

$$M = \sum_x \sum_y (I(x+1,y) - I(x,y))^2$$

where $I(x,y)$ is the intensity of an image pixel. The higher the metric, the better. By using an objective measurement, the sharpness of the images is easier to obtain.

2.2.4 Storage Requirements

To store the recorded images, we employed a storage server with a RAID 0 disk configuration. The software communicates with the cameras using Gigabit Ethernet connections. It decodes the network packages from the cameras, and stores all raw data to disk, including raw image data, camera timestamps, frame number, size, et cetera. We dump all raw data, without compression or conversion, to eliminate heavy processing and to therefore eliminate the risk of framedrops. By using a RAM buffer of about 2 Gigabytes per camera, disk latency and fluctuations can be accounted for, eliminating frame drops due to variable writing performance.

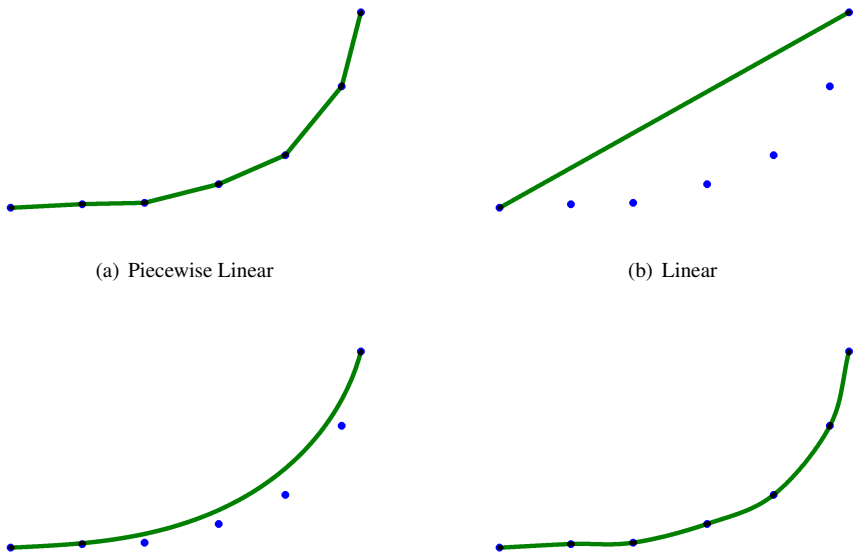
By using an external synchronization signal for the cameras, recording is started when the signal source is turned on, and all frames with the same sequence number are taken at the same time. This eliminates the requirement of a global timestamp clock.

2.3 Setting up the system

Once the cameras are up and running, some preprocessing should be done before the actual view interpolation can be started. This preprocessing stage consists of camera calibration and background determination.

The camera calibration uses image correspondences to determine the position, orientation, and intrinsic camera properties (for example, focal length), which are required in subsequent steps. This step is discussed in chapter 5.1.

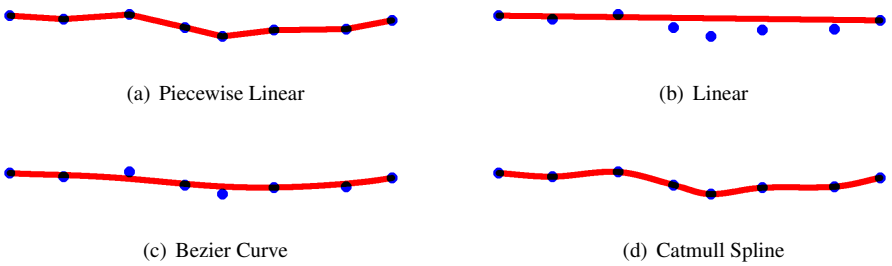
The background determination uses a small number of images (about 30) from a camera and calculates the median color value per pixel. This will result in a background image of that camera, and will be updated to include lighting changes. This background image is then used for foreground/background separation during the rendering. Detailed description of the background determination can be found in section 5.3.1 and the application is discussed in section 5.3.2.



(a) Piecewise Linear (b) Linear

(c) Bezier Curve (d) Catmull Spline

Figure 2.4: Overview of the possible virtual camera paths for the curved arrangement. The proposed paths are the piecewice (a) and global (b) linear path, the Bezier curve (c) and the catmull curve (d). The catmull curve is the most optimal path for this arrangement, because this path passes through the real camera positions and makes no sudden turns. By staying close to the real camera positions, we can increase the quality of the rendering, as explained further on.



(a) Piecewise Linear (b) Linear

(c) Bezier Curve (d) Catmull Spline

Figure 2.5: Overview of the possible virtual camera paths for the linear arrangement. The proposed paths are the piecewice (a) and global (b) linear path, the Bezier curve (c) and the catmull curve (d). The catmull curve is the most optimal path for this arrangement.

2.4 Choosing the Virtual Camera Position

Once the capture setup is running and the renderer is ready to accept a render request, the user of the system can choose a viewpath of the virtual camera. Because the rendering yields the highest quality if the virtual camera is in between the real cameras positions, we propose a system where a path can be chosen, instead of an arbitrary camera position. Therefore, the user can move left and right, which is represented by a fraction t, instead of having to choose a completely arbitrary position [Ware and Osborne, 1990], where $t = 0$ at the leftmost camera position, and $t = 1$ at the rightmost position. Any value for t, with $0 \leq t \leq 1$, represents a camera position. There are a number of options to convert t to a 3D camera position. We considered 4 of them.

To make these options work, the cameras must be sorted appropriately, from left to right. We can now number the cameras from C_1 to C_N, and we can choose 2 adjacent cameras to which we want to apply the path interpolation.

First, the cameras can be connected by straight lines, and the position of the virtual camera is interpolated on these lines. The position fraction t is converted to a local fraction t', valid on a line between 2 adjacent cameras C_n and C_{n+1}.

$$t' = mod(t(N-1),1) \tag{2.1}$$
$$C_n = floor(t(N-1)) \tag{2.2}$$

Now the position of the virtual camera between C_n and C_{n+1} can easily be determined by simple linear interpolation:

$$C_v(x) = (1-t')C_n(x) + t'C_{n+1}(x) \tag{2.3}$$
$$C_v(y) = (1-t')C_n(y) + t'C_{n+1}(y) \tag{2.4}$$
$$C_v(z) = (1-t')C_n(z) + t'C_{n+1}(z) \tag{2.5}$$

The result of the camera path is shown in Figure 2.4(a) and 2.5(a): the path is full of corners and sharp turns, i.e. there is no tangential continuity. This will result in an unpleasant effect.

Second, the outermost cameras C_1 and C_N can be connected by a straight line. Again, linear interpolation is used for the path.

$$C_v(x) = (1-t)C_0(x) + tC_N(x) \tag{2.6}$$
$$C_v(y) = (1-t)C_0(y) + tC_N(y) \tag{2.7}$$
$$C_v(z) = (1-t)C_0(z) + tC_N(z) \tag{2.8}$$

The result of the camera path is shown in Figure 2.4(b) and 2.5(b). For the curved case, the path diverges a lot from the real camera positions. For the straight arrangement, however, the path remains close to the real camera positions and gives a pleasant result without abrupt path changes and unnecessary turns. Therefore, this is the preferred method when the cameras are arranged in a line, even when the line is not perfect (as is the case in Figure 2.5).

Third, we used a Bezier curve [Prautzsch et al., 2002, page 9] as the camera path and used the real camera positions as control points. The path is constructed as follows:

$$C_v(x) = \sum_{i=1}^{N} \left(\begin{array}{c} N-1 \\ i-1 \end{array} \right) (1-t)^{n-i} t^{i-1} C_i \tag{2.9}$$

Analogous for the y and z coordinates.

The result of the camera path is shown in Figure 2.4(c) and 2.5(c). There are no direction jumps in the path, as is the case in the first linear interpolation method. The path stays close to the camera positions if the linear arrangement is used. However, the second linear interpolation approach yields better results and is preferred to Bezier curves. In the curved case, the path is closer to the camera positions than the second linear interpolation method, but still a distance from it. We have opted for the path going through the camera positions for the curved arrangement.

Fourth, we used Catmull-Rom splines [Parent, 2002, page 458] as a solution for the curved arrangement. This curve goes through the camera positions, while avoiding direction jumps. This combines the advantages of the first linear method and the Bezier curves. The path is constructed as follows, using t':

$$C_v(x) = UMB \tag{2.10}$$

$$U = \begin{bmatrix} t'^3 \\ t'^2 \\ t' \\ 1 \end{bmatrix} \tag{2.11}$$

$$M = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \tag{2.12}$$

$$B = \begin{bmatrix} C_{n-1}(x) \\ C_n(x) \\ C_{n+1}(x) \\ C_{n+2}(x) \end{bmatrix} \tag{2.13}$$

Analogous for the y and z coordinates. If C_{n-1} or C_{n+2} do not exist, it is created by mirroring C_{n+1} around C_n or C_n around C_{n+1}.

Figure 2.4(d) and 2.5(d) reveal the results of the camera path. The results for the curved setup are pleasant and provide a path without direction jumps through the real camera positions, i.e. there is tangential continuity. This is a desired results, and outperforms the 3 previously discussed methods. For the linear arrangement, however, the path makes unnecessary turns, and a straight path is preferred.

Once a path method is determined, the user can choose a value t and get the virtual camera position belonging to that t. The value of t can also be determined by tracking a specific player [Iwase and Saito, 2002]. By selecting an appropriate value for t, the tracked player can be kept automatically in the center of the virtual image. This effect is the most useful in the linear arrangement, where a player can be followed along the field.

The rotation of the virtual camera is determined by applying the slerp interpolation method on the quaternion representations of the rotations of the first and last cameras [Dam et al., 1998, page 42]. The first and last cameras were chosen to avoid excessive rotation if the rotations of the real cameras are not aligned with each other. If the slerp interpolation was applied to the 2 closest cameras, angular motion of the virtual camera could be inverted when another camera becomes the closest. For example, if camera 1 and 3 look a little bit to the left, and camera 2 a little bit to the right, the virtual camera will turn from left to right between the first pair of cameras, and from right to left between the second pair of cameras. This is avoided by only considering 2 cameras for rotational interpolation.

2.5 Rendering

Given a set of time synchronized raw images and the corresponding chosen virtual camera position, the renderer generates an image, as seen from the virtual camera position. The process is shown in Figure 2.6. The method is developed using traditional and modern GPU approaches to allow a fast and scalable solution.

First, the images are debayered using a FIR filtering approach to acquire RGB images, as discussed in section 5.2. By employing GPU technologies, fast processing is achieved. Next, foreground/background separation is applied, as discussed in section 5.3.2.

After these preprocessing steps of the rendering phase, foreground and background are processed independently. The different steps are depicted in Figure 2.6, and the foreground processing is shown in detail in Figure 2.7.

2.5.1 Foreground rendering

The foreground rendering uses a plane-sweep approach, based on the method of Yang et al. [2002], to generate a novel viewpoint, where the reprojection consistency for different depths is maximized, hence generating a novel image and depth map simultaneously. Plane sweeping uses the concept of projective geometry. We will introduce this concept here, and formal-

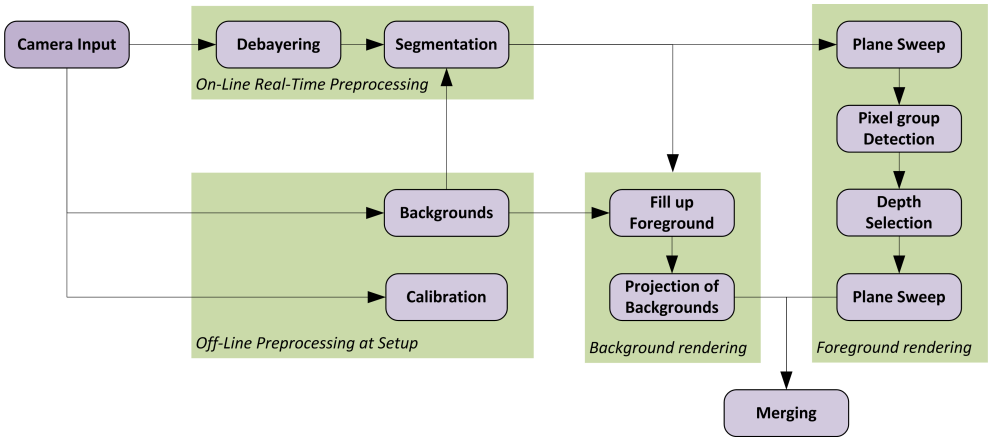


Figure 2.6: Overview of our method for the rendering. Both the preprocessing and rendering phase are shown.

ize it mathematically in section 5.1. Plane sweeping using projective geometry is described formally in section 6.2.

2.5.1.1 Projective Geometry

Projective geometry is the branch of mathematics that deals with projection of 3D points onto a plane [Hartley and Zisserman, 2003]. We will focus on projections where a 3D point is projected onto the camera plane by a single connecting line through the camera center. This can be represented as matrix and vector multiplications, which we will discuss in section 5.1.

This is a conceptual representation of a camera. In a real-world scene, there will be a real camera, which projects 3D scene points onto its image plane. If we want to create a virtual camera placed relative to real cameras, we will need to know where these real cameras are located, and how they perform the projection exactly. Determining these properties of the real cameras is called calibration.

Once the properties of a real camera is known, we can do a backprojection of the image points. We do this by connecting a line between the camera center and the image point on the plane. Every point on this line will have the same projection, i.e. the previously used image point. There is no unique 3D point that is a backprojection of the image point. As a consequence, we cannot determine the corresponding 3D point of an image point, even if the camera is fully calibrated. Reconstruction of a 3D scene using 1 image with only projective properties is hence not possible.

Therefore, multiple cameras are used to do a 3D reconstruction. We will now discuss a multicamera method called plane sweeping, which uses the backprojection and projection concepts described above.

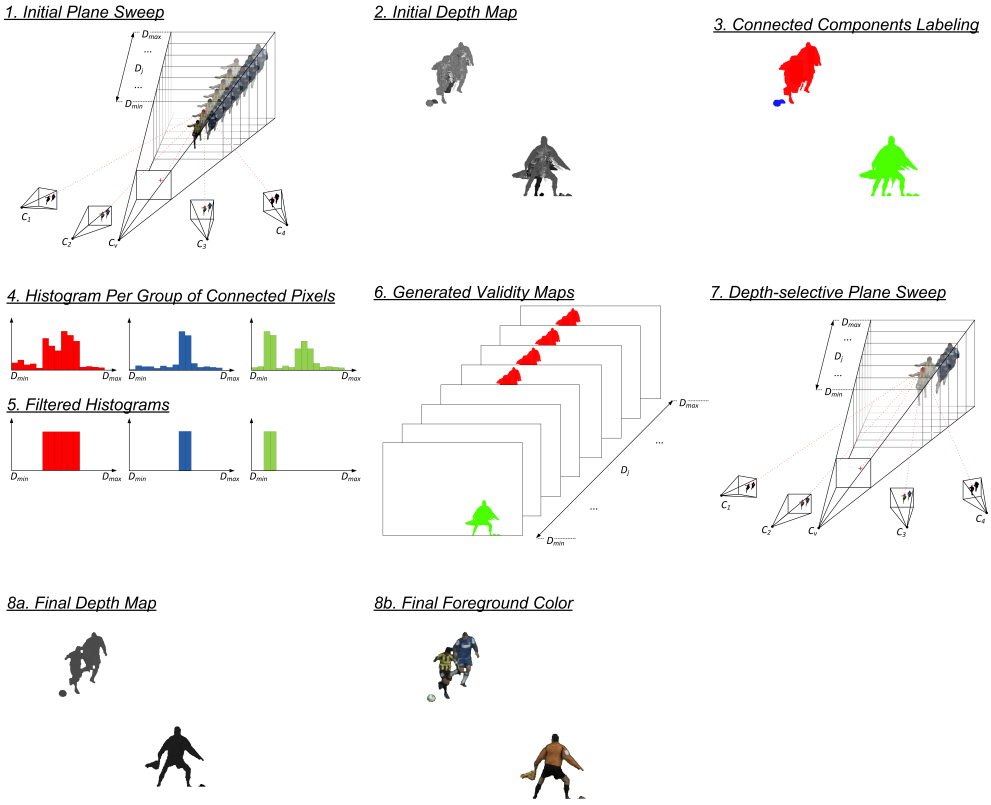


Figure 2.7: Overview of our method for foreground interpolation. 1: An initial depth map is acquired by a plane sweep approach. Only the two closest cameras are used for color consistency; the other are used for segmentation consistency. 2: The initial depth map. Serious artifacts, such as a third leg, can be seen. 3: The depth map is segmented using a parallel connected components approach. 4: For every group of connected pixels, a depth histogram is acquired. 5: The histogram is filtered using the depth of the background and depth assumptions. 6: The resulting validity map. Colored pixels denote a valid depth; white pixels are non-valid depths. There is a validity map per depth. 7: A second, depth-selective plane sweep is used, where some depths for a group of pixels are omitted, based on the corresponding validity map. 8a: The final depth map, and 8b: the corresponding color result.

2.5.1.2 Plane Sweeping Concept

Plane sweeping deprojects all the input images onto a plane in front of the virtual camera, and projects these onto the virtual image plane. The color consistency is determined per pixel. This process is repeated for different depths of the planes (relative to the image plane) and the best color consistency per pixel is saved. This way, color and depth for the virtual image is determined concurrently.

This concept is shown in Figure 2.8, where the images of $D_1 \dots D_4$ are shown in Figures 2.9, 2.10, and 2.11. Each image of the input cameras C_1, C_2, and C_3 is projected to a depth plane ($D_1 \dots D_4$), and reprojected onto the virtual image plane of camera C_v. The virtual images are a color blend of 7 input cameras; Figure 2.8 only shows 3 input cameras to reduce clutter.

Take for example the projection of the images in Figure 2.9(a), where the input images are projected onto D_1. In this figure, no player projections coincide. If a player in 3D space would coincide with D_1, all backprojections will be on the same position, resulting in a sharp image of the player in the virtual image.

For depth plane D_2, shown on Figure 2.9(b), a single player position coincides with the depth of D_2, resulting in a backprojection of that player on the same location on D_2 for all input cameras.

This is further investigated in Figure 2.10(a). A player is deprojected onto the same location, annotated as a red cross in Figure 2.10(b). At the same time, another player is not projected onto the same location, shown as blue circles. The head of the player under the blue circles belongs however to the same player. The situation is reversed in Figure 2.11(a).

We can now compare the projected colors (per pixel in the virtual image) to determine which depth plane yields the best color consistency. This is determined per pixel in the virtual image. For example, if for a specific depth, the colors red and green are projected on a virtual pixel, the color consistency is low; if the colors are all green, the color consistency is high. We check per pixel in the virtual plane the color consistency for each considered depth plane and store the depth and average projected color of the depth plane where the color consistency is the highest.

In Figure 2.10(b), the projected pixels under the red cross have a high color consistency, as these projected colors come from the same player. The color consistency of the pixels around all the blue circles is low, because there is a projection of the background of one camera and a player of another camera onto the same virtual pixel.

2.5.1.3 Plane Sweeping with Depth Selection

Normal plane sweeping does not yield high quality results for soccer scenes using a wide baseline setup (Figure 2.7(1-2)). Serious artifacts, such as ghost players, can be perceived. These artifacts are further discussed in section 6.2.1. We observed that these artifacts have different depth values than the valid results. To cope with these artifacts, we therefore employ

a depth selection method (after the initial plane sweep) where the acceptable depths of groups of pixels are determined (2.7(3-6)) and used in a second, depth-selective plane sweep (2.7(7)).

2.5.2 Background Rendering

Once the foreground is determined, we require the virtual background. We use the backgrounds of the input images, which contain the shadows. The holes in these images, where the foreground was, is filled up by the backgrounds generated in the preprocessing step (section 2.3). This way, complete backgrounds for the cameras are generated, where the shadows and lighting effects are preserved, as present in the frames for that time. Next, all these frame-specific backgrounds are deprojected to the pitch plane, as known from the calibration information, and reprojected to the virtual camera image. Overlapping parts are blended together using specific weights to cope with color differences in the different cameras. This process is detailed in section 6.1.

After the foreground and background are generated, the results are merged together to the final result.

2.6 Conclusions

In this chapter, we presented an overview of our method. We discussed the requirements of the captation, the steps required to set up the system, and the choice of the virtual viewpoint. These considerations are not the main part of the research, but they are important for the final quality and care should therefore be taken.

Next, we discussed a high level overview of the rendering itself. These steps are discussed in detail further on.

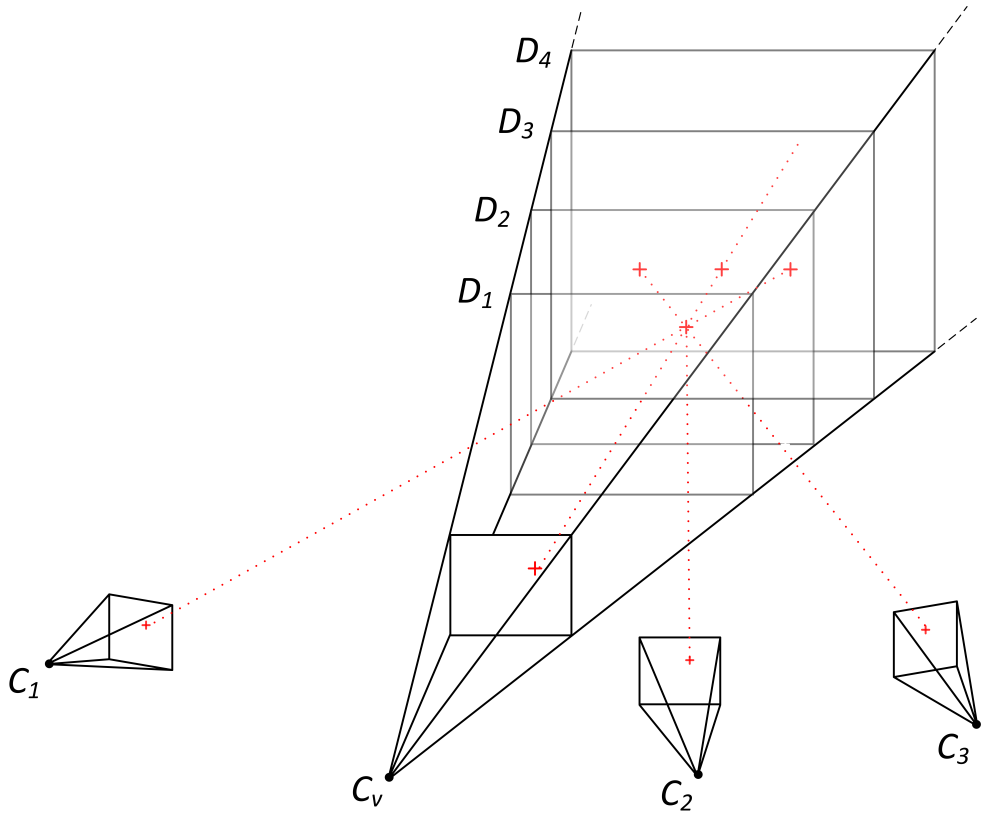
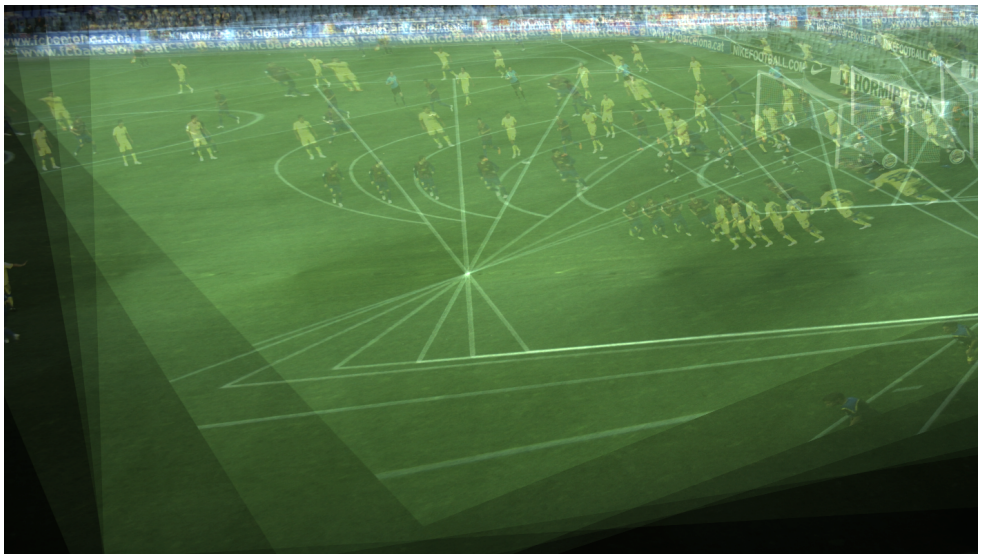


Figure 2.8: Concept of plane sweeping. The real cameras C_1, C_2, and C_3 are projected to the depth planes $D_1 \ldots D_4$ and reprojected to the virtual image of camera C_v. The results are shown in Figures 2.9, 2.10, and 2.11. For each pixel, for each depth plane, the color consistency of the reprojected colors is determined and the depth (and corresponding average color value) of the depth plane with the best color consistency is saved as result.



(a) Image of D1



(b) Image of D2

Figure 2.9: Projection of the input images on D_1 and D_2. Each image is the projection of the real cameras on a different depth plane, as seen from the virtual camera. Every pixel represents a projection of different cameras, and multiple colors are projected on each pixel and blended together. A player is found on a specific depth if the projected colors are consistent for that player for each input image.



(a) Image of D3



(b) Zoomed in part

Figure 2.10: Projection of the input images on D_3. (a) This image is the projection of the real cameras on a depth plane D_3, as seen from the virtual camera. Every pixel represents a projection of different cameras, and multiple colors are projected on each pixel and blended together. A player is found on a specific depth if the projected colors are consistent for that player for each input image. (b) Zoomed in part of the image. The red cross shows where a player is projected onto the same location for every input image. This is not the case for the blue circles.



(a) Image of D4



(b) Zoomed in part

Figure 2.11: Projection of the input images on D_4. (a) This image is the projection of the real cameras on a depth plane D_4, as seen from the virtual camera. Every pixel represents a projection of different cameras, and multiple colors are projected on each pixel and blended together. (b) Zoomed in part of the image. The blue circle shows where a player is projected onto the same location for every input image. This is not the case for the red crosses.

Chapter 3

State of the Art: Image Interpolation

3.1 General View Interpolation **33**
3.1.1 Explicit Geometry Information **33**
3.1.2 Depth Information **34**
3.1.3 No Geometry Information **36**
3.2 View Interpolation for Soccer Scenes and Other Sports **37**
3.3 Conclusions **40**

In this chapter, we will give an overview of existing view interpolation systems, followed by a discussion of the systems specifically designed for sport scenes.

3.1 General View Interpolation

Generating a novel viewpoint of a real-life scene is a well-known research field, with different approaches. We will classify these approaches based on the required geometry of the scene, i.e. the geometry that is known beforehand, as was done similarly by Shum and Kang [2000].

All methods will have input information, such as a 3D model or a collection of images, and a virtual image plane with camera center. The problem of image interpolation is defined as determining the color pixels on the virtual image plane.

3.1.1 Explicit Geometry Information

The most straightforward method of free viewpoint rendering relies on a complete geometric model with color information (or texture information) of the scene. The model is rendered using the classical rendering pipeline, briefly explained in section 4.1.1.

The 3D model can be acquired using different methods. The first method uses manual modeling. A person copies the scene to a 3D representation using modeling software (such as AutoCAD or Blender). This method is tedious, error-prone, and not very useful for dynamic scenes. The result is typically animation-like, and not so realistic.

Second, 3D modeling and capturing can be automated using application-specific methods. For example, human movement can be captured using skeleton-based tracking [Carranza et al., 2003; Starck and Hilton, 2003]. Static environments and objects can be modeled using camera-based SLAM (Simultaneous Localization and Mapping) [Montemerlo et al., 2002], structure from motion [Weng et al., 2012], manual modeling assistance [Debevec et al., 1996], or object scanning [Izadi et al., 2011; Newcombe et al., 2011] techniques. More specialized methods use lasers to probe the surface of objects to reconstruct color and geometry, where the result is a set of 3D points. This has been demonstrated by Levoy et al. [2000] by modeling ancient statues. This point cloud can then be converted to a 3D model [Lin et al., 2004].

A well-known generic, multicamera method is the visual hull method by Laurentini [1994] and Matusik et al. [2000]. Similar approaches are demonstrated by Slabaugh et al. [2002], called the photo hull, and by Bogomjakov et al. [2006], called the depth hull. The silhouettes of the object of interest is determined in every input image by segmentation methods. A 3D cone is determined by the camera location and the silhouette on the image plane. Every line in this cone going through the camera center is a projection ray of the object. By intersecting the cones of all the viewpoints, a 3D object is reconstructed that contains the object of interest. The object can be represented as a collection of voxels [Cheung, 2003], planes, lines [Baumgart, 1974], etc. These representations are compared by Geebelen et al. [2013]. The color information can be projected on this object, and this object can then be

rendered from different viewpoints. The method works best if the object is convex and the cameras are placed at wide baseline locations.

Colors can be applied to 3D models using textures recovered from color cameras. Efficient texture mapping can be achieved using relief textures, as discussed by Oliveira et al. [2000], where the input color textures are prewarped to allow the use of the texture mapping capabilities of the GPU. Due to errors in the reconstructed model or due to specular effects, recovered color values can be view-dependent. To cope with this, view-dependent texture mapping combines different images from different viewpoints to create a better result [Debevec et al., 1998, 1996; Eisemann et al., 2008].

Video-based interpolation systems have been demonstrated using 3D reconstruction. Kanade et al. [1997] demonstrate a system in studio conditions with a large number of cameras (about 50 cameras) placed in a dome to reconstruct a 3D model of the scene within. Moezzi et al. [1997] and Ladikos et al. [2008] present a visual hull based system, using a lower number of cameras.

While the rendering of 3D models will result in high quality images, independent of virtual camera position and output resolution, quality is directly depending on the quality of the model. By creating the model, automatic or manual, lots of detail and other information may get lost, which will result in artifacts in the rendering phase. While the rendering is of high quality, modeling may introduce errors.

3.1.2 Depth Information

If not the geometry, but the depth values from a specific viewpoint (or viewpoints) are known, other methods become possible. This is represented as an image (or multiple images), where each pixel is augmented with a depth value.

Depth can be acquired using active stereo cameras, such as the Microsoft kinect [Zhang, 2012], or time-based structured light methods, where a changing light pattern is used [Scharstein and Szeliski, 2003; Zhang et al., 2002, 2014].

Passive, image-based depth recovery methods use only the input images to reconstruct the depth information. Stereo vision methods, where 2 rectified images are used to calculate disparity values based on the color of the images, are well known and discussed by Scharstein and Szeliski [2002], Rogmans et al. [2009b], and Seitz et al. [2006]. Many approaches have been used to extract disparity values, and therefore depth values, from 2 rectified images. These include window-based scanline matching using edge-sensitive weights [Richardt et al., 2010] or adaptable window sizes [Lu et al., 2007; Zhang et al., 2009a,b], where different disparity possibilities are tested and an error value is assigned to each. The disparities with the minimal error value is selected for each pixel independently. Alternatively, global optimization methods based on graph cuts (and similar) [Papadakis and Caselles, 2010; Wang et al., 2006; Yang et al., 2006], segmented patches [Zitnick and Kang, 2007], and spatiotemporal consistency [Davis et al., 2003] are used.

Once the depth is known, we can warp the depth and color values to generate a novel viewpoint where the virtual camera is on the axis between the 2 cameras or, alternatively, the color values can be deprojected in the scene space, and projected on the virtual image plane [McMillan, 1997; Seitz and Dyer, 1997]. The reverse is also possible, where the virtual pixels are mapped onto the input images. This is referred to as backward warping [Kang et al., 2006]. There are, however, some possible issues with rendering based on color pixels with depth values. First, resolution changes can result in the degradation of the novel viewpoint. If the virtual image is further away, multiple pixels of the input image get projected to a single pixel in the virtual image. This will result in blended colors, and reduces the resolution of the projected image. Second, information might be unavailable due to occlusions in the input images. Some methods have been proposed to cope with these holes caused by occlusions by inpainting or extrapolation [Chen et al., 2005; Po et al., 2011; Sjostrom et al., 2011; Wang et al., 2008] or by including temporal information [Camplani and Salgado, 2012].

An alternative to the hole filling problem involves layered depth maps, as presented by Shade et al. [1998] and extended by Chang et al. [1999], and by Zitnick et al. [2004]. A layered depth image represents the scene as seen from an input camera, but has multiple pixels along the outgoing rays, each with its depth value and color value. Rendering novel viewpoints uses warping techniques [McMillan, 1997] where the multiple depth and color values per pixel are used to fill up the resulting holes. The layered depth images can be acquired from a large collection of images using motion estimation algorithms [Shade et al., 1998] or traditional disparity calculation [Yoon et al., 2005].

Another method of rendering a novel viewpoint using geometric information deduced from the input images is by using billboards [Hayashi and Saito, 2006; Waschbüsch et al., 2007]. The billboard method segments objects in the input images, determines their depth and places a geometric proxy in the scene, such as a plane. This geometric proxy is referred to as a billboard. The color information is projected on the correct proxy and a novel viewpoint is acquired by rendering these planes, textured with their color information. The proxies can be placed in an artificial scene or projected on the warped backgrounds of the input images. Some extensions are used, such as depth values onto the proxies [Shade et al., 1998], or using the human skeleton to improve the image on the billboard [Germann et al., 2010]. The method works best if the scene consists of sparse objects and the virtual camera does not differ too much from the input cameras. If the novel viewpoint is too far away from the input viewpoints, projective distortions may be visible, resulting in low quality renderings.

If the depth is not known, we can generate a novel viewpoint without first calculating the depth values from the input images. Depth information will be acquired as part of the rendering process, but is not used for the rendering itself. One method is the plane sweep method [Gallup et al., 2007; Geys et al., 2004; Nozick et al., 2006; Yang et al., 2002], as already discussed in section 2.5. Different depth hypotheses, i.e. depth planes, before the virtual camera are tested by projecting all the input cameras on that plane and calculating the color consistency of each pixel. The pixels with the best color consistency are kept, and the

average color of the projected pixels is used as the final result. By storing the depth values of the depth plane with the best color consistency, a depth map is achieved. The depth map, however, is only a byproduct of the method, and is not used for the actual rendering. A number of optimizations can be used to increase the quality of the results, including segmentation of the input images [Dumont et al., 2009], depth plane elimination or redistribution based on temporal criteria [Goorts et al., 2013b; Rogmans et al., 2009a]. Since we use the plane sweep principles for our method, some of these optimizations are discussed in chapter 6.

3.1.3 No Geometry Information

When no geometry is available, reconstructed, or available after rendering, many images can be used to generate a novel viewpoint. Most of these methods define their problem as resampling the plenoptic function.

The plenoptic function is defined by Adelson and Bergen [1991] as a 7D function:

$$p = P(\Phi, \Theta, \lambda, x, y, z, T)$$

The function P defines the strength of an incoming light ray at a location (x, y, z) and a lookat direction defined by 2 angles Φ and Θ with a wavelength λ at a time T. This representation is independent of the camera model or parameters, making it a very versatile representation. If the complete plenoptic function is known for all valid arguments, the complete scene is known. Rendering an arbitrary viewpoint is then trivial.

The complete plenoptic function is, however, seldomly known. The plenoptic function for a specific set of arguments is therefore determined based on a number of known samples of the function. Images can be considered as samples of the plenoptic function; there is a camera at a location at a specific time in the scene, capturing light rays of 3 wavelengths. Each pixel corresponds to a specific light ray.

Image morphing, presented by Chen and Williams [1993], is a method where the pixel movements between 2 images are calculated and are used to calculate the images of intermediate positions. The pixel movements are represented as flow fields. The method has good results if the cameras are not too far apart from each other, and if the flow fields are correct. Flow field calculation, however, is still an open problem [Baker et al., 2011].

McMillan and Bishop [1995] uses cylindrical images and flow fields for the novel viewpoint rendering. The method is referred to as plenoptic modeling.

Another approach is light fields, presented by Levoy and Hanrahan [1996], and lumigraphs, presented by Gortler et al. [1996]. If the camera stays out of the bounding box of the static objects of interest and the space between the camera and the object is empty, the plenoptic function can be simplified to a 4D function:

$$p = P(u, v, s, t)$$

where (u,v) and (s,t) represent coordinates on 2 parallel planes. These parallel planes are the samples of the plenoptic function, and are generated by placing a camera at a location (u,v) and taking an image with the focal plane at the same location as the (s,t) plane. Specialized cameras have been developed that capture the light field directly and can be used to refocus specific parts of an image in post-processing [Marwah et al., 2013; Ng et al., 2005]. Creating a novel viewpoint consists of determining the (u,v,s,t) parameters of each pixel of the virtual image, and resampling the radiance information from the (u,v) and (s,t) planes. The method, however, uses a large number of samples and requires a large storage. The light field representation is developed to allow fast resampling and to efficiently compress and decompress the input samples.

The unstructured lumigraph method by Buehler et al. [2001] uses the same principles, but uses a geometric proxy to improve rendering performance. Furthermore, there are less restrictions on the sample acquisition, making the method more versatile. The camera positions do not have to be located on the same plane, and the image planes do not have to be coplanar. However, the cameras need to be geometrically calibrated. When rendering the novel image, the input colors are blended according to the angle of the view rays, and, if available, visibility information.

3.2 View Interpolation for Soccer Scenes and Other Sports

The previously discussed methods perform very well under controlled studio conditions. Using these methods, however, in outdoor, large-scale environments is typically not trivial. Some specialized systems have been developed to allow free viewpoint video in outdoor sport scenes, as discussed below. We will make a comparison of these systems in chapter 8, after we discussed our system.

The earliest systems used a large number of cameras. The Super Bowl of 2001 was recorded using 30 motorized cameras, called the Eye Vision system [Kanade, 2001], where a rendering was created by jumping from one camera to another. Visible camera jumps could be perceived.

Fehn et al. [2001] and Fehn and Kauff [2002] describe a complete system where a virtual camera can be used. They mainly focus on the system and do not provide a view interpolation used on real life sports recordings. The camera calibration is performed using a feature matching approach, but is limited to 3 cameras.

Liberovision [Liberovision, 2013a] (now part of Vizrt [Vizrt, 2013]) demonstrated a semi-automatic system for free viewpoint soccer interpolation using the lines to determine the pitch plane, and by projecting the players on billboards. The billboard textures are created by projecting the closest camera onto the billboards. Blending, image jumping, and disappearing players can be seen if the closest camera changes [Liberovision, 2013b]. The method, however, is demonstrated for fast moving cameras, such that these artifacts cannot be seen easily,

and the method only uses existing broadcasting cameras. No specialized setups are required, making it a versatile solution.

Germann et al. [2010] propose a system for free viewpoint rendering in soccer scenes using a billboard approach. The players are subdivided in regions and the pose of the players is estimated using a pose database and manual intervention [Germann et al., 2011]. This pose is used to create a collection of billboards, where the color information is projected on. These billboards can then be rendered from any viewpoint. The method is demonstrated to work much better than normal billboard methods, but requires heavy processing due to the pose estimation (about 6 minutes per frame using GPU processing) and is heavily dependent on input resolution and pose database quality. Furthermore, the system is not fully automatic, requiring human intervention per frame.

Germann et al. [2012] reconstruct simple geometric proxies and use view-dependent texture projection and blending. The method is fully automatic, but the results show some missing limbs and noise in the colors, even for sparse placed players.

Koyama et al. [2003], Kameda et al. [2004], and Ohta et al. [2007] propose a method using billboards, where the players are subtracted from the background in each input camera and projected on a plane in the virtual space. Overlapping textures are determined and solved using a stereo-based approach. The players are tracked by an overhead camera, and the tracking is used to set the plane in the virtual space. No perspective correction is applied, and all foreground objects are assumed to be standing on the ground. Calibration is performed using laser measuring instruments and calibration objects in the scene. The background is manually modeled using CAD software. It is possible to generate virtual camera positions everywhere around the pitch by using only 9 cameras.

Inamoto and Saito [2002a,b, 2003a, 2007b] describe a system that uses a homography and linear interpolation to generate novel viewpoints using 2 cameras. The input streams are segmented in a dynamic region, a field region, and a background region using the static backgrounds of the scene and the shapes in the background. The players are matched in each viewpoint using the homography based on the field, and the 3D location is determined similarly. This assumes that the players are standing on the ground. The pixels of the players are then matched to each other using epipolar geometry and the position of the pixels are linearly interpolated, based on the virtual camera position. This way, a warped image is acquired. The field region is considered as a plane and moved based on the calibration, and the background is considered a plane at infinity. Calibration is done manually. The method limits the virtual camera in between the real camera positions. This method is used in an augmented reality system using head-mounted displays [Inamoto and Saito, 2003b, 2004a,b, 2005, 2007a; Saito et al., 2004]. The position of the head-mounted display is calculated to determine the position of the virtual camera by extracting lines on a model of the pitch, and by matching these on the input data. Furthermore, Kimura and Saito [2005a,b] applied the method on tennis games.

Hayashi and Saito [2006] present a system that extends the previous systems of Inamoto and Saito [2002b] by using billboards instead of linear interpolation of pixel positions. The players are tracked manually and placed in a virtual environment. In contrast to the system of Inamoto and Saito [2002b], the virtual camera can go outside the region between the real cameras.

The Piero system of the BBC [BBC, 2010], commercialized by Red Bee [Red Bee Corporation, 2001], uses manually generated models or billboards of the players, that are subsequently placed in a virtual scene. The intersection of the players with the pitch plane determines the location of the players. The virtual camera is limited around a real camera position. If the virtual camera is moved too far away from the real camera, visible artifacts occur [Grau et al., 2007a].

The Iview project by the BBC [BBC, 2008], discussed by Grau et al. [2005] [Grau and Vinayagamoorthy, 2010; Grau et al., 2007a,b; Hilton et al., 2010, 2011; Kilner et al., 2007], had the objective to provide a virtual viewpoint system for soccer and rugby with replay at interactive framerate. The system can work with a minimum number of 4 synchonized cameras, and is based on 3D reconstruction and view-dependent texture mapping. Geometric calibration is performed by detecting the lines on the pitch [Thomas, 2007]. This method was demonstrated to work well, except when there were no lines in the images due to the zoom level. Earlier tests used calibration objects placed in the scene [Grau et al., 2005]. The background is modeled manually and consists of a simplified geometric model.

Kilner et al. [2006, 2009] performed a comparative study between visual-hull based 3D reconstruction and billboard techniques to generate novel viewpoints of the foreground. The comparisons use real soccer data. The shape quality, distortion, temporal stability, etc. were considered. It is concluded that all tested methods can remove players if the calibration is inaccurate. Furthermore, segmentation errors can introduce errors in all tested methods. If the reconstruction of the player's location is incorrect, double images appear, resulting in significant artifacts. The billboard method works best if a high number of cameras is available, but fails considerably if there are only a few cameras. Occlusions pose a problem for the billboard methods, as parts of another player are projected on the billboards, resulting in double images and ghosting effects. The visual hull approach will create ghosting volumes when occlusion happens. These artifacts tend to be unstable over time, and appear and disappear in the video seqences. The view-dependent visual hull technique was considered the method with the best overall results.

Based on the previous study, Grau et al. [2005] used the visual hull approach, with an octree-based representation. The silhouettes are created using image segmentation based on chroma keying, where the background is considered green [Grau et al., 2005], based on user-selected color seeds to determine appropriate background colors [Grau and Vinayagamoorthy, 2010; Hilton et al., 2011], or based on motion compensated pixel-based differences between the input and a predetermined background [Grau and Vinayagamoorthy, 2010; Hilton et al., 2011]. The visual hull representation, together with the segmentation, are refined us-

ing a deformable model approach, where an error measurement based on the input images is minimized [Hilton et al., 2011; Kilner et al., 2007].

After the visual hull reconstruction, the player textures are projected on them using a view-dependent texture mapping method. The textures of up to three cameras are blended together, where the blending weights and chosen cameras are determined by the viewing direction.

The results show that some artifacts, such as ghosting legs, ghosting players, and flickering remain. These are mainly devoted to segmentation and calibration errors. Generally, the results are of high quality. The 3D models were not created in real-time, but replay is demonstrated to work at interactive rates.

Rodriguez et al. [2005] use a triangle-based disparity map beween 2 views to represent matches and location. It is created using a seed growing method. The result is an image for each camera, divided in triangles, and each triangle in one image has a match in the other image. Novel viewpoints are created by shape interpolation and texture morphing. The method is limited to virtual camera positions in between the real camera positions.

Furuya et al. [2007] applied the billboard technique to wrestling. Because only 1 billboard was used, the method is not applicable to soccer as such.

Hulth and Melakari [2005] applied the billboard technique to soccer, together with tracking software. No tests on real soccer data were performed.

3.3 Conclusions

In this chapter, we gave an overview of existing free viewpoint rendering methods. We used the classification based on available geometric information to provide a natural way of representing these methods.

Many methods were presented that are specifically tailored to soccer scenes. Most methods use a billboarding technique or a 3D reconstruction. The few methods based on image-based rendering are limited in applicability. Therefore, our method fills up a void in the methods used for free viewpoint video in soccer scenes. We present a fast and high quality, image-based method that solves many problems of the previously discussed methods, including ghosting and long processing time. A comparison of our method with other methods, and a discussion of resulting quality, is presented in section 8.2.

Chapter 4

State of the Art: GPU technologies

4.1 Traditional GPU Technologies **43**
4.1.1 The Programmable Graphics Pipeline **43**
4.1.2 General Use of the Graphics Pipeline **45**
4.2 Modern GPU Technologies: CUDA **45**
4.2.1 Execution Model **46**
4.2.2 Architectural Model **48**
4.3 Conclusions **50**

The Graphics Processing Unit of a computer, a GPU, is the device responsible for transforming 3D computer models into a 2D representation, suitable for display devices. However, the GPU can also be used as a general purpose parallel computing device, which is especially interesting for its high performance and scalability. First, a GPU offers more performance than a CPU for off-the-shelf devices. For example, a NVIDIA GeForce GTX Titan offers a theoretical processing power of 4500 GFLOPS per second [NVIDIA, 2014b], compared with the Intel Core i7 XE, obtaining a performance of 109 GFLOPS [Williams, 2010]. These large differences demonstrate the potential power of GPUs using only consumer graded hardware, and therefore reduce the price per GFLOP. The main reason behind this difference is the relative amount of transistors assigned to processing (instead of caching and flow control), which is much higher on GPUs. Second, scalability is an important consideration when dealing with image and video processing. The resolution of video data steadily increases with time, and therefore the processing power must follow to provide the same quality. GPUs offer an easy way to scale up the processing system. Extra GPUs can easily be used thanks to their parallel nature, and GPUs become more powerful with time. However, high performance and scalability entail extra cost and effort. The GPU is not a general purpose processor and there are considerations and restrictions to use a GPU as computing device, restricting the applicability in general.

Nowadays, there are two ways to use the GPU as a general purpose computing device: the traditional way and the modern way. In the traditional way, the programmable rendering capabilities of the GPU is misused to perform general calculations. In comparison, the modern way can use the GPU directly, and developing parallel algorithms is less tedious. However, both methods have their strengths and weaknesses, and both are used in combination for our method. Therefore, we will discuss both approaches as this will contribute to the understanding of subsequent chapters.

4.1 Traditional GPU Technologies

When the GPU renders a 3D scene, a classical approach is used [Shreiner, 2009]. This approach consists of a pipeline of graphics data where each stage in the pipeline accepts the result of the previous stage. The pipeline stages are fixed and operate in parallel. We will now discuss the general principles of this pipeline. Details not relevant for further considerations will be omitted. Then, we will discuss the general use of the graphics pipeline.

4.1.1 The Programmable Graphics Pipeline

As shown in Figure 4.1, the pipeline consists of, at least, a vertex transformation, a rasterization, a fragment transformation and raster operations. The application sends vertices to the GPU, where they are processed during the vertex transformation stage. Each vertex typically consists of a position, color, and texture coordinate sets, and is accompanied by assembly

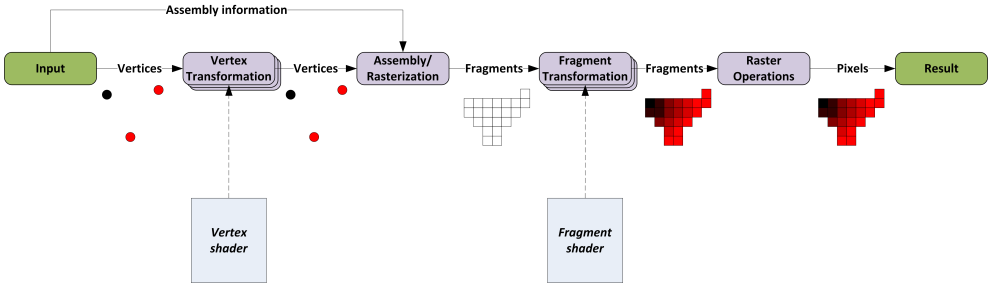


Figure 4.1: The GPU pipeline. The steps consists of the vertex transformation, the assembly and rasterization, the fragment transformation, and the raster operations. The final result is a color and depth buffer. The vertex and fragment transformations are programmable, allowing a flexible use of the GPU pipeline.

information to allow the combination of vertices into geometric primitives. The vertex transformation considers each vertex individually and applies operations, such as transforming it to a screen position and preparing it for rasterization.

The assembly and rasterization stage combines vertices into geometric primitives, as defined by the application. The result is a collection of triangles, points, lines, or other geometric primitives. The GPU now selects the primitives for rendering. The leftover primitives are converted to a set of fragments. Fragments are pixels, accompanied by other information, such as depth, interpolated color values (based on vertex color values), texture coordinates, etc.

These fragments are passed to the fragment transformation. Here, the color of each fragment is determined, based on its interpolated color values, texture coordinates, and other relevant properties. During the fragment transformation stage, some fragments may be discarded, if needed.

During the final stage, the GPU will perform some testing and will update the final color and depth buffer, if necessary. For example, the depth of each fragment will be tested with the depth of other fragments at the same location, and only the closest fragment will be retained. The color of the leftover fragments will be blended in the color buffer at the appropriate location, and will result in the final image.

To allow a more flexible rendering pipeline, the vertex transformation stage and the fragment transformation stage are programmable on modern GPUs [Mark et al., 2003]. This is done by defining a program for one vertex or one fragment. These will be executed independently from each other for each vertex or fragment, and can be executed in parallel or sequentially, depending on the GPU capabilities. It is the programmability of these two stages that allow the creative use of the GPU capabilities.

4.1.2 General Use of the Graphics Pipeline

Typically, the GPU pipeline processes 3D scenes, to be displayed on a 2D display device. However, the GPU can solve problems other than rendering by using the programmable vertex and fragment stages. If the problem is easily parallelizable, the problem can be split up in small parts, where each part will undergo the same processing, independent from any other part. We will illustrate this concept in the context of image processing, because this is our main application. Nevertheless, it has been demonstrated that using the GPU pipeline is also possible for searching, sorting, matrix operations and other algebraic problems, physics simulation, and many more [Fernando, 2004].

To perform image processing using the GPU pipeline, we can define a quad to render an image. We create a texture, and populate it with the desired image to be processed. We then render the quad to a buffer and assign the texture coordinates to its vertices. The result is that the texture will be rendered to a buffer, and the fragment shader will process each pixel of the image. This way, we can define a shader program for each pixel – which are independent from each other – and exploit the parallel capabilities of the GPU, and therefore increase the performance. As an extra performance improvement, the graphics capabilities are available in the shader programs, such as depth testing and projective texturing, and therefore allow the offloading of parts of the processing to specialized hardware.

There are many libraries and frameworks that ease the use of the GPU for general processing. The most notable include Sh [McCool and Du Toit, 2004; McCool et al., 2002], Scout [McCormick et al., 2007], RapidMind [McCool, 2006], Microsoft Accelerator [Tarditi et al., 2005], Brook/Brook+/BrookGPU [Buck et al., 2004], CGis [Fritz et al., 2004], Glift [Lefohn et al., 2006], and CTM [ATI, 2009]. However, this dissertation will not consider these libraries, and will not discuss them further. Instead, we will only use the Cg framework [Mark et al., 2003].

4.2 Modern GPU Technologies: CUDA

Because the previously discussed methods to leverage the computing capabilities of GPUs for general computing misuse the graphical interface, novel technologies have been developed to ease the use of general, parallel computing on GPUs. One of these technologies is CUDA [NVIDIA, 2014b], and is at the moment considered as the modern way to use the GPU as a parallel computing device. NVIDIA started developing CUDA in 2006. CUDA exposes the GPU as a collection of SIMD (single instruction, multiple data) multiprocessors, together with a more flexible programming model. The GPU is no longer a purely graphical computing engine; generic parallel computing is now possible with ease.

CUDA is only available for NVIDIA devices. While there are other technologies available, such as OpenCL [Stone et al., 2010] and DirectCompute [Yang and Howes, 2010], that provide the same functionality on a more extended group of devices, we opted for CUDA

for the rapid development of the technology, and the increased performance due to a more tailored approach.

CUDA capable devices all have a number of features, parameters, and capabilities. These are divided in distinct groups, called compute capabilities, to ease the comparability of different devices. Different devices with the same compute capability can have, for example, a different number of processors and a different amount of memory, but their features, block limits, etc., are all the same. Many approaches for optimizing a method depend on the used compute capability. We will therefore discuss a number of them if the results are significantly dependent on the compute capability.

The CUDA technology can be divided in two separate models: an execution model and an architectural model. The execution model determines how the algorithm should be written to be able to run on a GPU (or other devices), without the knowledge of how this eventually will be executed, including the actual level of parallelism. In this model, possible parallelization will be defined and some execution guarantees will be determined. The designer determines how its algorithm will be parallelized and which synchronization is required for correct execution. Some limitations may apply, and are determined by the targeted compute capability.

The architectural model defines how this execution model will map on a specific device with a specified compute capability and a specified number of computing units and memory. In essence, only the execution model is required to design a parallel algorithm; CUDA will map this model to GPU and execute it. However, the architectural model is useful to optimally use the device, without loss of processing power, time, and other resources [Goorts et al., 2010; NVIDIA, 2014a].

We will now discuss both models in depth.

4.2.1 Execution Model

Using the execution model, the programmer divides the algorithm in a number of independent execution threads, where each thread executes the same instructions, i.e. the kernel. The set of threads is divided into groups, called blocks, which are executed independently from each other by the GPU. All blocks have the same size, and the maximum size is determined by the compute capability. Each thread is aware of its position in a block and its current block number, to allow the use of different input data. 1D, 2D, or 3D coordinates can be used to determine the thread and block positions.

For example, when processing images, a typical thread division is performed by defining a thread per pixel of the image. Two coordinate numbers are used per thread and per block, to easily map these numbers on a specific pixel in the image. If the block size (S_x, S_y), the block number (B_x, B_y), and the thread number (T_x, T_y) are determined, one can easily identify the pixel coordinate to be used to read the required data: $x = B_x \times S_x + T_x$ and $y = B_y \times S_y + T_y$. This is shown in Figure 4.2.

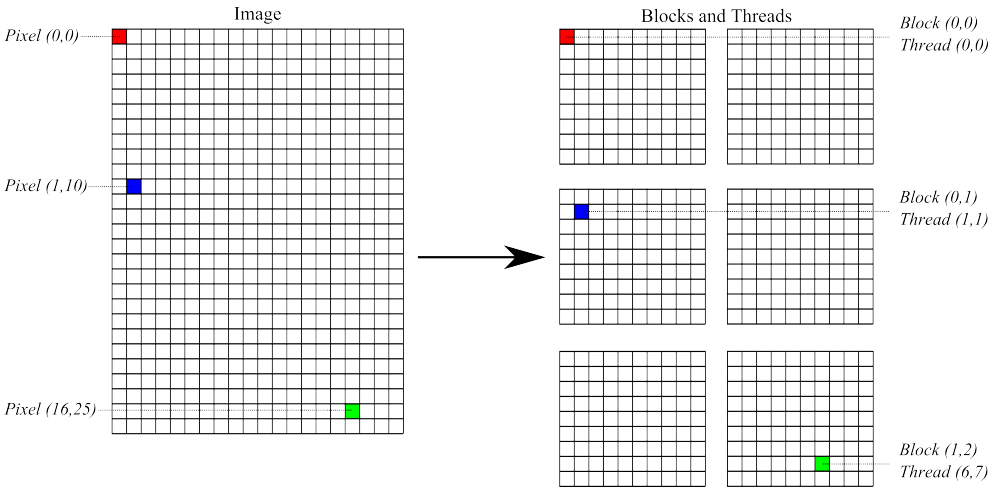


Figure 4.2: Mapping an image to CUDA threads and blocks. One thread is assigned per pixel.

The GPU assigns a block of memory to each thread; no other threads can access this memory. The access time of this memory is fast; the size of the memory is defined by the compute capability, and the memory is used for local calculations.

Additionally, each block has a block of memory that each thread in that block can access, and this block of memory is therefore called the shared memory. All blocks have a distinct, separate shared memory. This shared memory is larger than the thread memory and has the same access speed; its size is also determined by the compute capability. It can be used to reduce memory operations by caching data required by multiple threads in the same block, or for interthread communication. To facilitate easy interthread communication or consistent memory usage, thread synchronization is provided by CUDA, where each thread will wait at a determined point until all threads of that block will reach that point. No interblock synchronization is available.

Furthermore, there is a global memory available, randomly accessible by all threads and the hosting CPU. This memory is very large, but relatively slow. As a result, loading and storing should be avoided as much as possible. How much slower is device-dependent. Additionally, a texture memory is available to allow interoperability between OpenGL and CUDA. CUDA kernels can access OpenGL textures for reading and writing and can be considered as global memory accesses.

The effect of slow memory, compared to fast processing speed, is called the memory wall [Wulf and McKee, 1995], where the memory operations are a significant portion of the processing time, compared to actual calculations. The problem of the memory wall is expected to be more prominent in the future [Asanovic et al., 2006]. Special care should be taken to avoid stalling of the calculations due to the slow memory accesses. This memory wall

will have an impact on the final design of the algorithm, and is an important consideration for the performance.

In a typical application, the host CPU will upload the input data to global memory and will start the computing. The threads will load their required data in shared or thread memory, calculate the results, and store the results back in global memory. The host CPU will download the results back to CPU memory, and the calculation is done. For example, when processing a pixel-wise operation on an image, the CPU will upload the image, issue a calculation command with a thread per pixel. Each thread will load one pixel in its local memory, calculate the result, and store its resulting pixel back at the correct position in global memory. Because each thread is aware of its coordinate, it can load and store a different pixel. Therefore, this can be considered as a logical SIMD architecture.

Nevertheless, CUDA is more flexible than traditional SIMD architectures. Divergent branching, i.e. conditional execution paths, are allowed, and therefore increase the flexibility of the technology. There is no need to consider multiple threads at once when designing the parallel algorithm; only a single thread has to be considered. The opposite is true for a typical SIMD architecture. Therefore, the CUDA architecture is typically referenced to as a single instruction, multiple threads (SIMT) technology, instead of SIMD.

4.2.2 Architectural Model

This execution model can now be executed on a specific device, with specific features and parameters. The combination of these parameters and the actual parallelization in the execution model determine the performance of execution. Therefore, knowledge of the targeted device and architectural details is important to achieve an as high as possible performance.

Current devices consist of a collection of independent multiprocessors and a global, off-chip memory block. Each multiprocessor consists, in essence, of a thread scheduler, a number of stream processors, one program counter (valid for each stream processor), and a block of memory. This is shown in Figure 4.3. Each multiprocessor will process one or more execution blocks concurrently, but only one block will be active per multiprocessor at a specific point in time. A block can be suspended, for example, during memory operations. If a block finishes its calculations, the next block can be loaded. This implies that many more blocks than multiprocessors can be created; the number of multiprocessors only determines the level of actual parallelism.

When executing a block, the threads per block are divided into groups of 32 threads, called warps. The threads of these warps are actually executed in parallel, not just conceptually. All warps have their own program counter, allowing independent execution, where the synchronization between warps is defined in the execution model. Warps can be suspended and another warp can be loaded, where the warp scheduler of the multiprocessor takes care of the selection of the correct information (e.g. the correct program counter and registers for that warp). When a warp encounters a block synchronization instruction, the warp is sus-

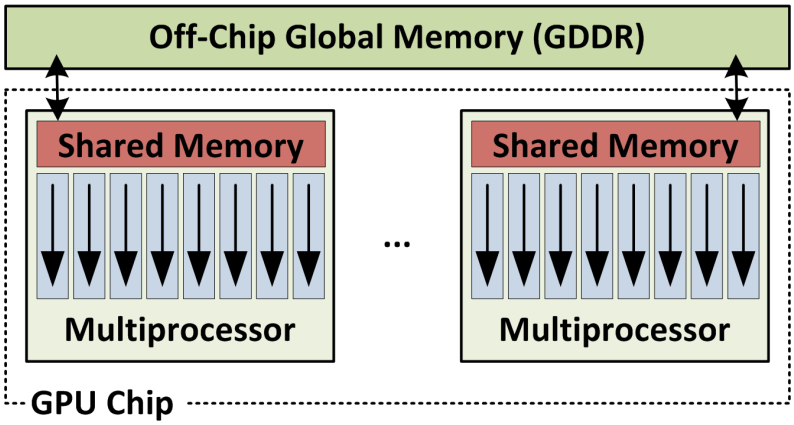


Figure 4.3: CUDA hardware can be abstracted as a number of SIMD multiprocessors, each containing 8 (scalar) processors and a dedicated on-chip shared memory to allow user-managed caching and inter-thread communication. However, communicating between multiprocessors is only possible through the global video memory.

pended and not woken up again until all warps of that block have reached the synchronization instruction.

When the execution encounters a conditional instruction, that will result in two different execution paths, both paths will be executed for all threads in that block, but no results will be stored when execution is on the wrong path for that thread. This must, of course, be avoided, because these calculations do not yield useful results and are a lost processing power resource. However, when all threads in a warp follow the same execution path, other paths are skipped. This implies that divergent branching will not result in lost processing power, as long as all threads in a warp follow the same execution path. By knowing the warp size and designing the execution paths in the execution model such that no divergent branching occurs in the same warp, performance loss can be reduced.

When the execution encounters a memory operation on the global memory, the calculations will stall. The global memory has a high access time of about a few hundred clock cycles. To avoid that the multiprocessor is just waiting, warp suspension is provided by CUDA. After a memory instruction is issued, thanks to the context switching abilities of the multiprocessor, the complete warp is suspended and a new warp can now use the multiprocessor. Even warps from different blocks can be used to fill up the available processing time. When the memory operations of all the threads of the first warp are complete, the first warp has its required data available, and the processing can continue. This process of suspending warps is called latency hiding.

To allow latency hiding to work, it is required that enough warps are available. If the block size is too large, too many threads per multiprocessor are assigned or too much memory is

used, then only a single block is assigned per multiprocessor. If that is the case, then not enough warps may be available to fill up processing time. If all warps of a block are waiting for their data from global memory, the multiprocessor will be doing nothing. This is a loss of processing power and can be avoided by considering the size of blocks, the synchronization places, the order of calculations, the amount of data to be loaded, etc. To quantify the ability of warp suspension, NVIDIA introduces the concept of occupancy. Occupancy is defined by NVIDIA as the ratio between actual warps on a multiprocessor and the maximum possible warps on a multiprocessor. Many factors have an influence on the number of warps on a multiprocessor (and therefore the occupancy), including registry and shared memory usage, block sizes, etc.

To increase performance by speeding up memory operations, CUDA provides memory coalescing per half-warp. Memory coalescing combines multiple memory read and write instructions into one. If all the threads from the same half-warp read a memory location in the same block of 64 bytes, only one read instruction for this block of 64 bytes is issued. This is possible thanks to the wide memory bus inside the GPU itself, where the transfer of the 64 bytes between the multiprocessor and the off-chip memory can occur in parallel. If, however, the memory operations are applied on random locations in the global memory, each thread may require a different block of 64 bytes, and multiple read instructions will be issued. Because the global memory operations are relatively slow, coalescing is an important consideration that will significantly speed up the processing. The coalescing conditions depend on the compute capability, where each new compute capability relaxes the requirements for coalescing to happen. However, there are still requirements, even in the modern devices, and a "free lunch" coalescing cannot be achieved [Goorts et al., 2009]. Therefore, compute capability is important to consider when designing a method to allow coalescing.

4.3 Conclusions

We have discussed two approaches to acquire general parallel computing using the GPU. By using the GPU, efficient and scalable processing is possible, while still allowing a practical and usable programming interface. Both approaches use a slightly different paradigm, and provide a different functionality. We therefore use both approaches together to leverage their strengths and hide their weaknesses.

Chapter 5

Image Data Preparation

5.1 Calibration ... 53
5.1.1 Representation of Camera Parameters ... 54
5.1.2 Determination of Camera Parameters ... 56
5.1.2.1 Determination of 2D Image Correspondences ... 56
5.1.2.2 Angle-based 2D Image Correspondences Filtering ... 59
5.1.2.3 Correspondences to Projection Matrices ... 60
5.1.2.4 Decomposition of Projection Matrices ... 61
5.2 Debayering ... 62
5.2.1 FIR Filtering for Demosaicing ... 65
5.2.2 Generic FIR Filtering using CUDA ... 65
5.2.2.1 Conventional Method ... 66
5.2.2.2 Filter Separation ... 69
5.2.2.3 Fourier Transformation ... 70
5.2.2.4 Experimental Results ... 70
5.2.3 FIR Filtering for Demosaicing using CUDA ... 72
5.2.3.1 Compute Capability 1.1 ... 74
5.2.3.2 Compute Capability 2.0 ... 75
5.2.3.3 Compute Capability 3.5 ... 76
5.2.3.4 Conclusions ... 76
5.3 Segmentation ... 79
5.3.1 Background Generation ... 79
5.3.2 Foreground/background Separation ... 79
5.4 Conclusions ... 80

This chapter describes the steps performed before the actual interpolation can be done, including prerendering steps and frame preparation steps. The steps before the rendering stages are camera calibration and background determination. These should be executed once for a specific recording, hence a CPU implementation suffices. The frame preparation steps, referred to as the preprocessing steps of the rendering phase, consist of debayering and foreground/background segmentation. These steps are developed using GPU technologies to allow fast processing. Figure 5.1 shown an overview of the system.

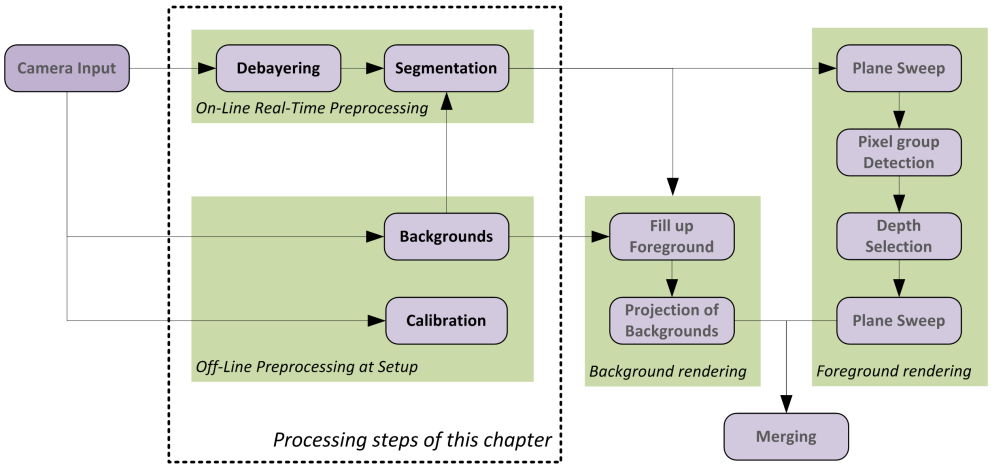


Figure 5.1: Overview of our method for the rendering. Both the preprocessing and rendering phase are shown. This chapter discusses the non real-time and real-time preprocessing.

5.1 Calibration

Before we can perform projective operations on the images, the corresponding cameras must be calibrated, i.e. we need to know where the cameras are and what their properties are. These parameters are represented by camera matrices, and calibration is the determination of these matrices.

There are a number of existing camera calibration methods available for outdoor sports scenes. Most of the methods use the lines of the soccer area to determine camera locations [Farin et al., 2003, 2005; Hayet et al., 2005; Li and Luo, 2004; Szenberg et al., 2001; Thomas, 2007; Yu et al., 2009]. These methods are therefore only applicable if the scene is a soccer pitch, and where the lines are visible and placed in a plane. This is, however, not always the case. The pitch is seldom a plane and cameras with a small field of view do not always have lines in their image stream. We propose a solution without the assumption of lines on a plane, which makes our large-scale calibration solution more robust and more widely

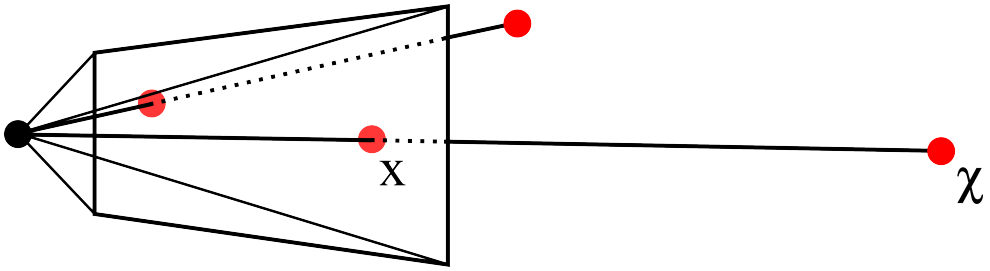


Figure 5.2: The projective camera model. A camera center and an image plane is defined. The image is formed by connecting a line between the camera center and the 3D point. The intersection between this line and the image plane defines the position of the projection for that 3D point.

applicable. The drawback, however, is that our calibration system is not real-time. This is not an issue for our method, as we require that the input cameras are static.

5.1.1 Representation of Camera Parameters

A camera is a model or construction that maps a 3D scene point to a 2D image point. In this dissertation, we will assume that the cameras follow the projective camera model, as defined by Hartley and Zisserman [2003, page 6]. This model represents the camera as a 3D point somewhere in the scene (the center of projection), and defines an image plane. A line connecting a 3D point in the scene and the camera point can be constructed. The intersection of this line and the image plane defines the position of the image projection point of the 3D point. This is shown in Figure 5.2.

This projective process can be mathematically represented in matrix notation as follows. Consider a 3D point χ, represented in homogeneous coordinates. In essence, homogeneous coordinates represent a point $\chi = [X,Y,Z]^T$, using four coordinates $\chi = [WX,WY,WZ,W]^T$ with $W \neq 0$ or $\chi = [X,Y,Z,1]^T$. A projective camera now transforms this 3D point χ in a homogeneous 2D point $x = [x,y,1]^T$ using a projection matrix P:

$$x = P\chi \Leftrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

We model our camera as a pinhole camera [Hartley and Zisserman, 2003, page 153], as a more specific model of what was described above. Here, the projection matrix can be split up in two sets of components: intrinsic and extrinsic parameters, represented by the intrinsic matrix K and the extrinsic matrix M, with $P = KM$.

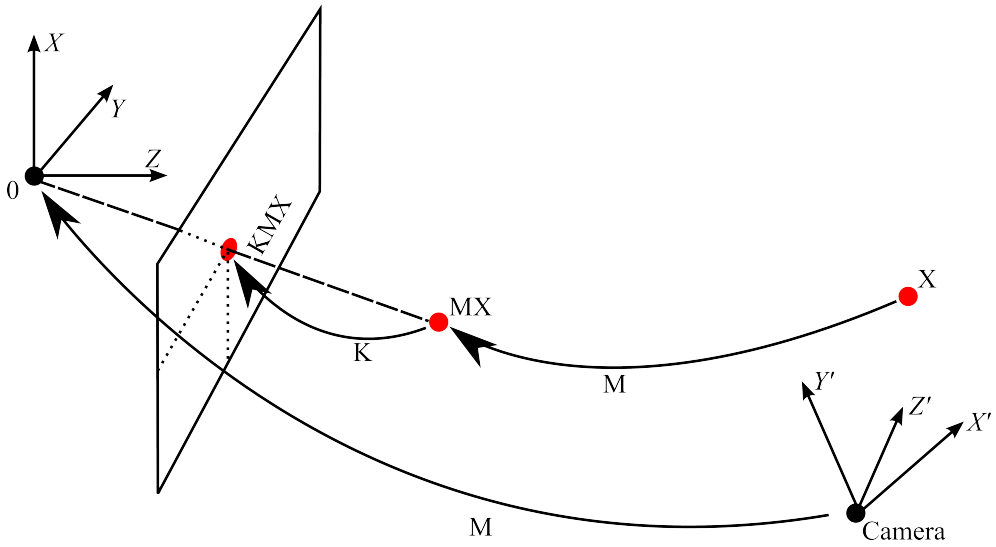


Figure 5.3: Intrinsic and extrinsic camera matrices explained. The point X and the camera center C are defined in an arbitrary coordinate system. Multiplying by M will transfer C to the origin of the coordinate system, and X will have the same relative position. Multiplying by K will project X to the image plane.

The intrinsic camera parameters represent the relation between a 2D pixel location and its corresponding 3D ray, presuming the camera is placed at the origin, and the image plane is parallel to the XY plane at $Z = f$, where f is called the focal distance. The line perpendicular to the image plane and passing through the center of projection is called the principal axis, while the point where the principal axis intersects the image plane is called the principal point. The principal point can be represented as a 2D point (p_x, p_y) on the image plane, assuming an origin is defined. We crop the image plane to a plane of finite size and place the origin of the image plane at a corner. This way, we can create the matrix K based on f and (p_x, p_y):

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Using $x = K\chi$ for a camera placed at the origin with an image plane $Z = f$, x is the image coordinate on the image plane with principal point (p_x, p_y).

However, the camera is seldomly placed at the origin, especially if multiple cameras are involved. Therefore, the extrinsic matrix M is used, which transforms 3D points to a new location and orientation, so that the intrinsic matrix is applicable [Hartley and Zisserman,

2003, page 155]. The matrix M consists of a rotation and a translation, as shown in Figure 5.3. The matrix has the following form:

$$M = \begin{bmatrix} R & -R\tilde{C} \\ [0\,0\,0] & 1 \end{bmatrix}$$

where R is a 3×3 rotation matrix and $\tilde{C}$ is the camera location in inhomogeneous coordinates.

In essence, M will translate and rotate the world such that the camera is placed at the world origin, and K will project the 3D points to the image plane, resulting in the final, projected image.

5.1.2 Determination of Camera Parameters

Using this camera model, we need to determine the projection matrices, when only the images of the cameras are given. To do this, we acquire image correspondences using feature matching, use these correspondences to generate projection matrices, and finally split the projection matrices in their intrinsic and extrinsic components. We only use the input images, and use a scene where players are moving. This way, each frame is different, yielding different features per frame, and therefore increasing the robustness of the method.

5.1.2.1 Determination of 2D Image Correspondences

We will determine image point correspondences by using feature detection. We run a feature detector on all the images and find the matches between the features of different images. To increase robustness, feature matching between a pair of images is done in two directions, i.e. find the matches from image 1 to image 2, and cross-check with the matches from image 2 to image 1. A number of feature detectors were tested, including the well-known SIFT [Lowe, 2004] and SURF [Bay et al., 2006], where SIFT proved to provide the most reliable matches for our dataset.

The configuration of the cameras determine the exact approach for finding matches. If the cameras are far away from each other, only the nearest camera to the left and right is considered to find matches. This is to avoid extreme outliers due to matches in images that contain a different part of the scene. If the cameras are placed in an arc, one camera can have a view angle perpendicular to the view angle of another camera. This will make matching of features on players unreliable, and is therefore avoided by using only three cameras.

An example is shown in Figure 5.4. Here, 318 matches were found over 3 cameras. There were 2661, 3168, and 3011 matches in the three images resulting in 1171, 951, and 1088 matching features between pairs of images. Due to cross checking between three images, only 318 matches were retained, therefore yielding a higher robustness.

Algorithm 1 Overview of the multicamera feature matching and selection algorithm.

Create empty list of multicamera matches L_m (list of lists of features)
for all Cameras C_p **do**
  for all Cameras $C_s, C_p \neq C_s$ **do**
    for all Feature $F_p \leftrightarrow F_s$ of $C_p \leftrightarrow C_s$ **do**
      Construct matrix M
      for all Cameras C_f **do**
        for all Cameras C_t **do**
          if $c_f = c_t$ **then**
            $M[C_f][C_t] = unset$
          else if $c_f = c_p$ **then**
            Select match $F_p \leftrightarrow F_2$ from $C_p \leftrightarrow C_t$
            $M[C_f][C_t] = F_2$
          else
            Select match $F_p \leftrightarrow F_2$ from $C_p \leftrightarrow C_f$
            Select match $F_2 \leftrightarrow F_3$ from $C_f \leftrightarrow C_t$
            $M[C_f][C_t] = F_3$
          end if
        end for
      end for
      Create empty list of features L_l
      for all Rows in M **do**
        Select the most occurring feature F_m in the row
        if Occurrence of $F_m \geq N/3 * 2$ **then**
          Add F_m to L_l
        end if
      end for
      if 3 or more features in L_l and L_l not in L_m **then**
        Add L_l to L_m
      end if
    end for
  end for
end for

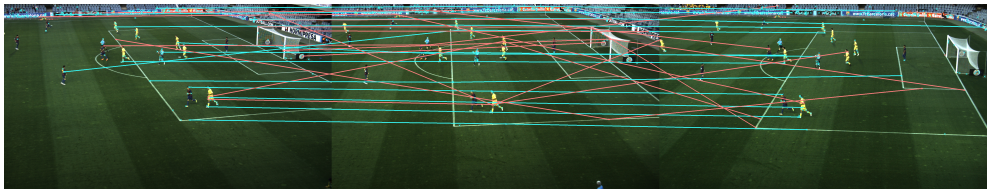


Figure 5.4: Example of multicamera feature matching using 3 cameras. All pairwise features are connected with each other using lines. Only a subset of the multicamera matches are shown, and the red mismatches will be removed later on in section 5.1.2.2.

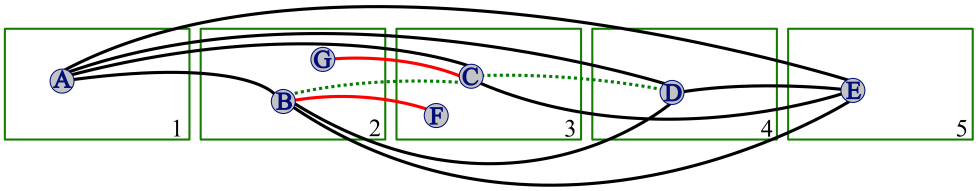


Figure 5.5: The graph used in our example. Nodes A - G are features detected in a set of images, each of which is taken by a different camera at the same moment. The red edges show mismatches between features, the black correct matches between features. The green, dashed lines show the feature matches that should have been found, but were not. After running our algorithm, A, B, D, E are considered as accepted in the multicamera feature match; C, F, G are rejected.

If the cameras are close to one another, matches between all pairs of images are searched for. We will select matches between all cameras based on a consensus based searching approach. An overview of this algorithm is given in Algorithm 1.

The algorithm can better be explained using the example of Figure 5.5. Here, a graph is shown, where each node is a feature, belonging to a specific camera image, and each edge represents matching features in two directions. An edge between node A and B, corresponds to a match between feature A and feature B, and vice versa.

We consider every pair of images and decide which feature pair will be kept, and which will be discarded. We decide which other features in other images belong to this match, therefore creating a multicamera match. For example, we consider camera 1 and camera 2. One of the cameras is the primary camera C_p, the other is the subordinate camera C_s. We choose camera 1 as C_p. Next, we construct a feature cross check matrix for each feature F_p that is a part of a match between C_p and C_s. In our case, we consider feature A. The matrix consists of N rows and N columns (where N is the number of cameras) and each row and column corresponds to a camera image.

We now complete every element of the matrix. For each element there is a "from" camera C_f and a "to" camera C_t. First, we select the match from C_p to C_f, that is $C_p \leftrightarrow C_f$, and use this feature to find the match to C_t ($C_f \leftrightarrow C_t$). For $C_p = 1$ with feature A, $C_f = 4$ and $C_t = 5$,

this would result in $A \leftrightarrow D$ and $D \leftrightarrow E$. The result is the final feature from the second match, and is placed in the matrix on row C_t and column C_f. If there is no match, or if $C_f = C_t$, the position in the matrix is left empty. For our example in Figure 5.5, this results in the following matrix:

		From Camera				
		1	2	3	4	5
To camera	1	-	A	A	A	A
	2	B	-	G	B	B
	3	C	F	-	-	C
	4	D	D	-	-	D
	5	E	E	E	E	-

There are several important elements worth noticing. First, as shown in Figure 5.5, there is no match between C and D, while there is a match $A \leftrightarrow C$ and $E \leftrightarrow C$, and a match $A \leftrightarrow D$, $B \leftrightarrow D$, and $E \leftrightarrow D$. Therefore, we can conclude that the match between C and D should exist (as indicated by the dashed green edge) and is just not found by the matching algorithm. Second, B matches to F, but both A and E match to both B and C. Furthermore, D matches to B. Therefore, we conclude that the match from B to F is a mismatch and should be eliminated. The same is true for $G \leftrightarrow C$. These two cases are handled by selecting the most occurring feature in each row.

To address the mismatches in Figure 5.5, we select the most occurring feature in each row and keep this feature only if it occurs more than two thirds of the time (including the empty places). For row 2, we see three times B and one time G. We can therefore consider B as part of the complete match, and ignore G. All rows in our example have a feature that is occurring two thirds of the time, except for the third row. Therefore, we will remove C (and F) from our multicamera feature set. Since C is only supported by two cameras, it is too weak to be considered as a reliable inlier, and is hence removed from the list.

By this method, we create a set of features for the feature from C_p, where we have calculated that they all presumably belong together. We will add this set of features to the global list of multicamera matches, after checking for duplicates. This process is repeated for every combination of C_p and C_s, and for every feature pair between these cameras.

5.1.2.2 Angle-based 2D Image Correspondences Filtering

Once the multicamera matches are determined, we perform an angle-based filtering which further enhances the correctness of the final result of the calibration by eliminating possible mismatches. The basis of this approach lies on the observation that correctly matched features in adjacent images have similar vertical displacement across images because our cameras are not rotated around the optical axis. More confidence is given to features that are more vertically "consistent" in adjacent images as large discrepancy in features' vertical position is a good indication of mismatch.



Figure 5.6: Multicamera feature matches, considered as inliers. Most outliers are removed using the angle-based filtering. Only a subset of the multicamera matches are shown.



Figure 5.7: Multicamera feature matches, considered as outliers. These matches were rejected using the angle-based filtering method. There are no false outliers in this example. Only a subset of the multicamera matches are shown.

To perform a filtering based on this vertical displacement, we place a pair of images next to each other and connect all matches between these images. Next, we determine the angles between the horizontal and the lines connecting the features. Of these angles, we erase the top and bottom 5% and calculate the average of the remaining angle values. We will now discard any match of which the angle differs more than 3 degrees from the average. This parameter is determined empirically and can be adjusted if required. This is an effective outlier removal method, as demonstrated in Figure 5.6 and 5.7. Figure 5.6 shows the matches that passed the angle test. There are 288 matches, compared to the previous 318 matches. Most outliers are effectively removed, and no valid multicamera matches are erroneously removed. Figure 5.7 shows the matches where the angle test failed. All these matches are outliers, and are therefore removed from the succeeding calibration process.

This process is only applicable if the cameras are not too much rotated relative to each other, especially around the optical axis. If that were the case, the assumption that lines connecting matching features are more or less parallel would not be correct. For the linear camera arrangement, all cameras are set up such that they are upright relative to each other. For the curved camera arrangement, only 3 cameras are considered at a time, so that this angle-based selection remains effective.

5.1.2.3 Correspondences to Projection Matrices

Once the 2D correspondences are determined and filtered, we feed them to the calibration toolbox of Svoboda et al. [2005]. Here, the projection matrices P are determined based on

the correspondences using a bundle adjustment approach [Triggs et al., 2000][Hartley and Zisserman, 2003, page 434]. RANSAC [Fischler and Bolles, 1981][Hartley and Zisserman, 2003, page 117] is used to remove outliers. Furthermore, radial distortion is determined [Hartley and Zisserman, 2003, page 189] and will be removed before any other processing.

The projection matrices are furthermore extended with an extra row $[0\,0\,0\,1]$ to make it possible to invert the matrices. The projection then becomes:

$$x = P\chi \Leftrightarrow \begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

5.1.2.4 Decomposition of Projection Matrices

Once the projection matrices P are known, we can decompose them in the intrinsic and extrinsic matrices using QR decomposition [Hartley and Zisserman, 2003, page 579]. This is not required for the plane sweep approach, as we only use P and P^{-1}, but the separation is used for camera position determination and visualization [Hartley and Zisserman, 2003, page 163].

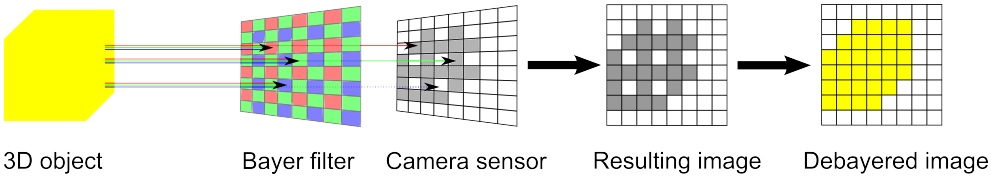


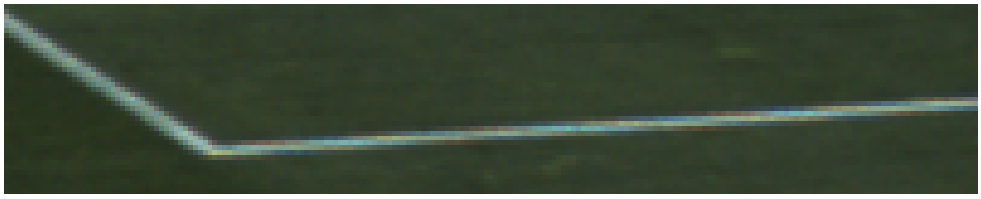
Figure 5.8: The concept of debayering. A 3D object is captured using a camera consisting of a bayer filter and a camera sensor. The light rays (in this case only red and green) are filtered by the Bayer patters, such that only one color component passes per pixel. Therefore, the resulting image consists of grayscale values, where these values represent different color components per pixel. In the example above, only the pixels that capture red and green values have nonzero values. The process of debayering reconstructs the final image using the bayered image as input. In the example above, the yellow pixels are reconstructed successfully.

5.2 Debayering

The images provided by the cameras are provided in raw format, i.e. no RGB images are available at this stage. This raw format is the data directly made available by the camera sensor. The majority of the cameras nowadays use a CCD array of sensors where every pixel sensor on the array can capture only one light intensity. Therefore, a color filter with different colors for every pixel is placed in front of the sensor array to capture red, green or blue values of the color spectrum. The colors are placed in a specific pattern i.e. the Bayer pattern [Bayer, 1976]. Typically, there are more green values than red and blue values, because of the spectral sensitivity of the human eye. Therefore, it is desirable to have more color information in the green channel, hence more green pixels in the Bayer pattern. The effect of such a color filter is that every pixel of the captured image only has a specific value for one color channel and the other color channels should hence be computed from the surrounding pixels. The calculation of the missing color channels is frequently called debayering or demosaicing. The process of capturing and debayering is depicted in Figure 5.8.

Even though most cameras perform demosaicing at the device level, it is useful to perform this processing later on. First, the raw data is only a third of the debayered image; the raw image only has one channel, instead of three. The raw image can be considered as a greyscale image, and only a third of the bandwidth is required, compared to RGB images. This will speed-up the communication between the camera and the processing device, which is a significant part of the image processing pipeline, and therefore increase the overall performance. Moreover, demosaicing on devices with more processing power may result in higher quality images. More complex algorithms can be applied and less processing restrictions apply.

The most straightforward method of demosaicing is bilinear interpolation of the surrounding pixels. To calculate the value of a missing color channel, the values of the surrounding pixels of that color channel are averaged. This method is fast, but does not yield sharp results and ignores borders and details, resulting in severe artifacts, e.g. color bleeding. This is



(a) Bilinear interpolation



(b) Malvar FIR filtering

Figure 5.9: (a) Debayering using bilinear interpolation. Artifacts, such as red and blue spots on the lines can be seen. (b) Debayering using FIR filtering. Here, the artifacts of bilinear interpolation are reduced, but not completely eliminated.

especially visible in images of soccer scenes, where white lines are placed in a green field, yielding blue and red artifacts. This is shown in Figure 5.9(a). Furthermore, blurring around fine details can be noticed. All these artifacts can regress the interpolation quality.

To generate better results, Laroche and Prescott [1994] and Malvar et al. [2004] propose methods that incorporate the gradient of the values per color channel. Interpolating along an object edge is better than across an edge, as to reduce color artifacts from selecting the color of the wrong objects in the scene. For example, at the border of a white line, pixels of the green field can be used. These have low values for the blue and red filtered pixels, which are used to generate the colors of the white line. This inevitably results in erroneous values. When using the gradient between the red values, a large gradient can be perceived, indicating a border in the image, thus the contribution of the red color values in that direction is reduced. However, if all color values are similar (i.e. the gradient is low), pixels probably come from the same object and will result in a higher contribution. Figure 5.9(b) shows this use of the gradient. In this Figure, the color artifacts are reduced. The method still produces artifacts, but we chose a more performant method over a more qualitative one.

More advanced methods exist. Hirakawa and Parks [2005] present an adaptive homogeneity-directed demosaicing algorithm that cancels aliasing and selects the interpolation direction with the least color artifacts. The interpolation direction is chosen such that a homogeneity metric is maximized. These kind of methods, however, are typically less performant when using GPU implementations, and we therefore avoid them. Results later on show that the debayering is of sufficient quality to produce good results.

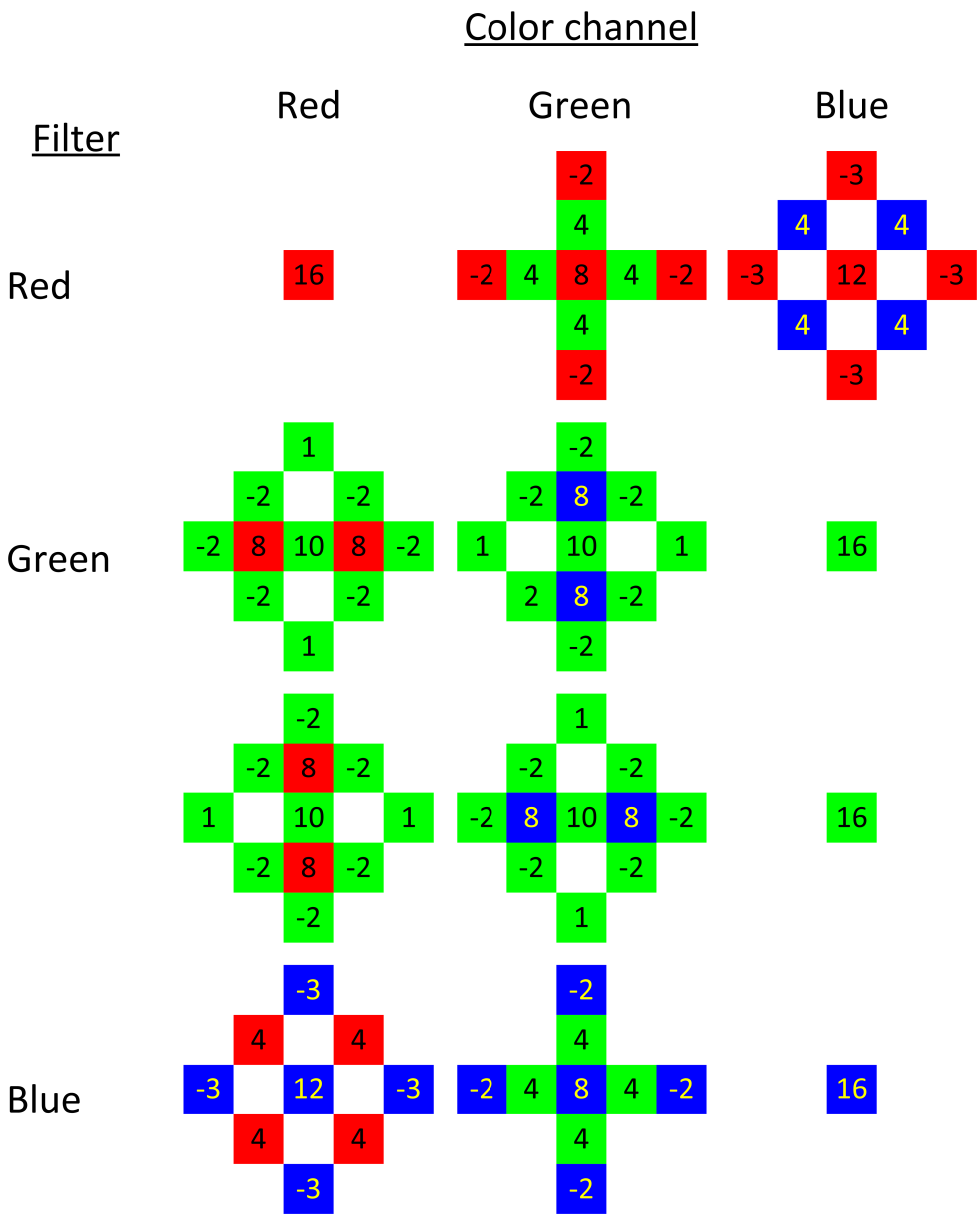


Figure 5.10: Convolution filters for demosaicing. The choice of filter is based on the desired color channel for that pixel (column) and the filter used for that pixel (row).

We will now investigate the real-time aspect of the demosaicing problem and apply the method directly to soccer data. More specifically, we will discuss the method of Malvar et al. implemented on CUDA. We choose this method because it uses linear finite impulse response (FIR) filtering to produce high-quality results. FIR filtering is known to map very well on CUDA [Goorts et al., 2009], which will maximize the performance, while preserving the quality. Although this method was implemented earlier on by McGuire [2008] using traditional GPGPU paradigms, their optimization principles do not map on CUDA. By leveraging modern GPU technologies, i.e. CUDA, even more performant processing can be achieved.

We will briefly describe the algorithm below. Next, we investigate generic FIR filtering using CUDA. Finally, we will apply these conclusions to the method of Malvar et al. [2004] to acquire the most optimal approach.

5.2.1 FIR Filtering for Demosaicing

Malvar et al. propose a non-directional demosaicing method implementable by a FIR convolution filter. Typical demosaicing algorithms, like bilinear interpolation, only use green filtered pixels for the green channel, red filtered pixels for the red channel, etcetera. The method of Malvar et al., on the other hand, also incorporates pixels where the filtered color differs from the current channel. This way, edge information is incorporated in the debayering process, resulting in increased overall quality around the edges. The edge information is acquired by using the gradient between adjacent pixels of the same filtered color.

This is obtained by applying a pattern to the pixel and its neighbors. The different patterns and their respective pixel weights are shown in Figure 5.10. The pattern used is dependent on the filtered color of the pixel and the desired color channel. For every pixel, three patterns are applied (one for every color channel), where one pattern is trivial.

These patterns are designed to improve the results around edges by incorporating the gradient of the luminance values. When we calculate the green value on a red filtered pixel, for example, we do not discard the red value. The red value is used to calculate the luminance change (using adjacent red values) and this is incorporated when calculating the green value. Doing so, we calculate the bilinear interpolation of the green pixels around the red filtered pixel and use the red filtered pixels to correct this interpolation for edges. The same principle holds for different color channels.

These patterns can easily be used as FIR filters and implemented as such. We applied the method of Malvar et al. [2004] using CUDA to implement the debayering FIR filtering. Because this method is a specialization of generic FIR filtering, we will discuss this first.

5.2.2 Generic FIR Filtering using CUDA

FIR filters or convolutions serve many purposes besides debayering, and are the primary driver behind various practical applications. More specifically, they can be used for simple image blurring to more complex low-pass noise reduction filters with edge preservation

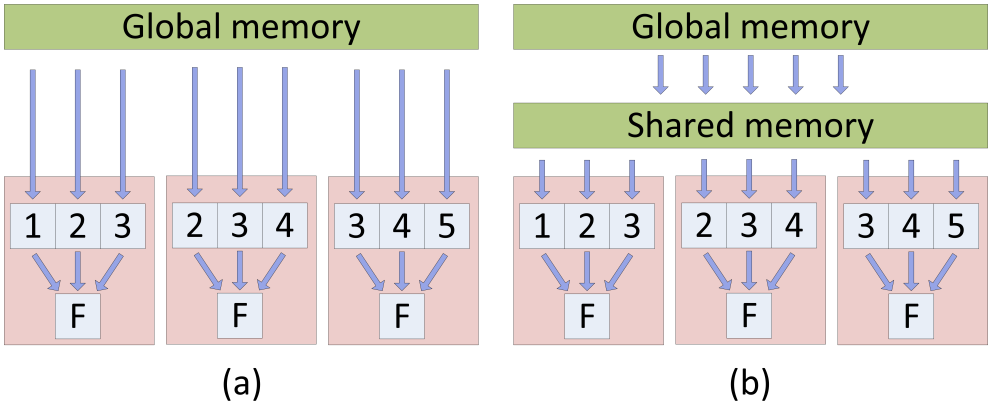


Figure 5.11: Implementation strategies for conventional FIR filtering on parallel SIMT architectures. (a) Naive strategy. Every thread fetches all required data, resulting in multiple slow memory accesses per thread. (b) Optimized strategy. Every thread fetches only one data element and stores this in a shared memory. Because other threads fetch the other data elements, data reuse is possible, resulting in less slow memory accesses.

[Bakker et al., 1999], edge detection [Luo and Duraiswami, 2008], and even as the core mechanism for real-time parallax determination [Gong et al., 2007; Rogmans et al., 2009b] and 3D reconstruction [Gallup et al., 2007]. Since they are both computationally and communication-wise very intensive, they often tend to progressively increase the algorithm complexity, resulting in inevitable application bottlenecks. Due to their inherent heavy data reuse and memory communication, these bottlenecks are most often caused by the 'memory wall' [Asanovic et al., 2006], i.e. the increasing discrepancy of contemporary computer architectures between the cost of data communication and regular computations, as discussed in section 4.2.

We investigate three conventional methods to perform FIR filtering, and determine the performance on modern GPU devices using CUDA, as described in section 4.2. Using these results for the different approaches, conclusions can be drawn for the debayering application further on.

5.2.2.1 Conventional Method

We can use the parallel direct GPU computing architecture, as described in section 4.2 to implement linear FIR filtering with the aid of a user-managed cache i.e. the shared memory [Goorts et al., 2009]. When implementing, for example, a 3×3 filter without optimizations, we allocate one thread for every pixel of the image. This thread will access 9 pixels of the image in global memory to calculate the final result for the allocated pixel (see Figure 5.11 (a)). However, it is possible to reduce the memory accesses by reusing the information of nearby threads, i.e. each thread only loads its allocated pixel in shared memory and can

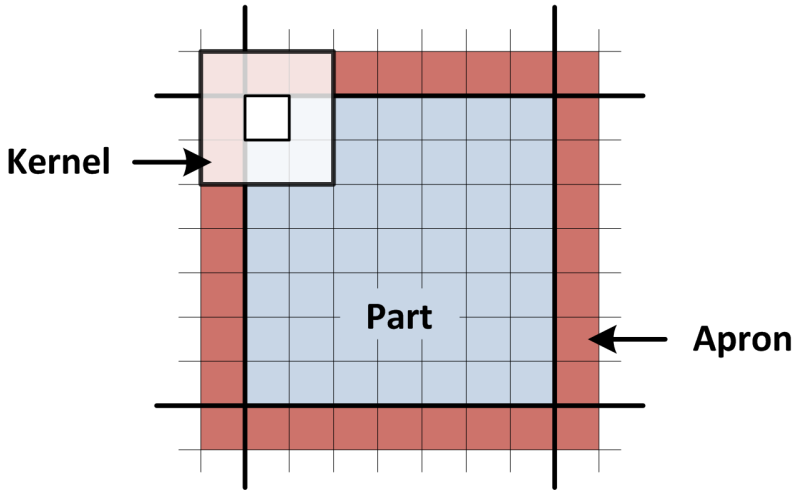
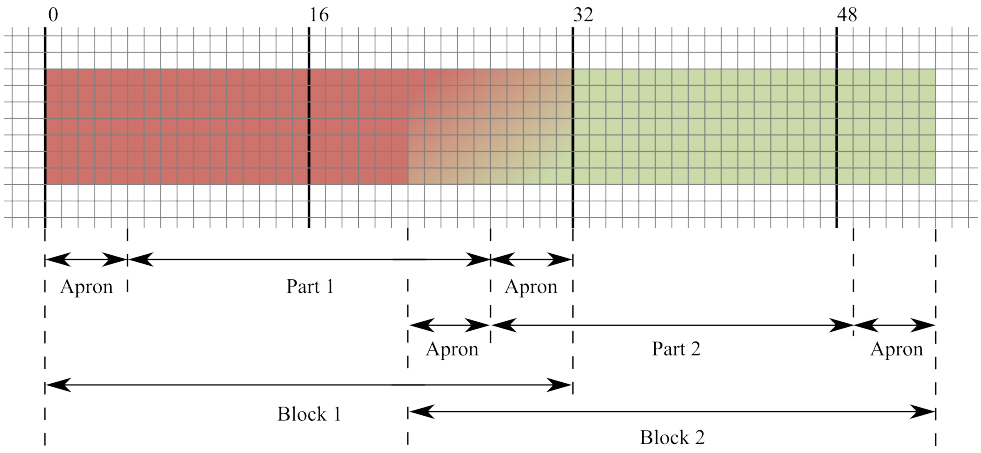


Figure 5.12: One block for filtering a part of the image. The threads at the border (red) are inside the apron and do not calculate new values for their pixels. They only load data for use by the internal threads (blue).

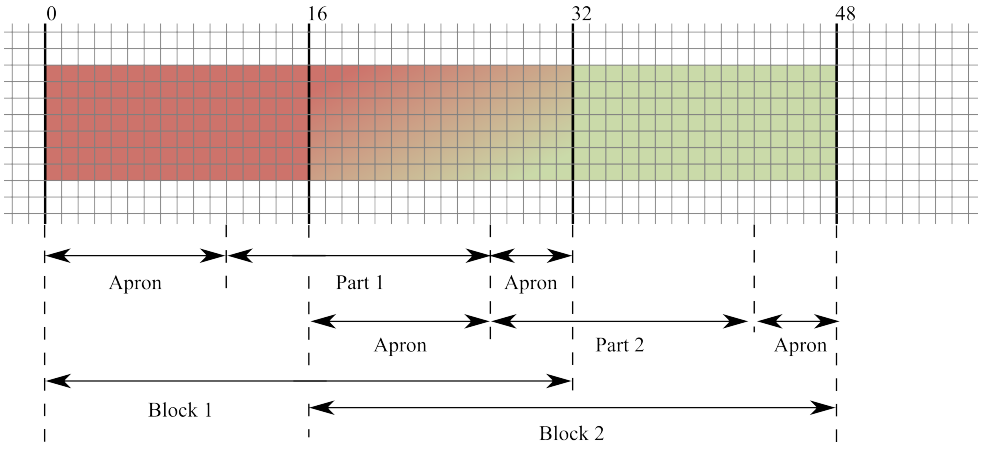
then use the values of nearby pixels which are loaded in shared memory by other threads. Therefore, the amount of global accesses per thread is reduced to one, which is shown in Figure 5.11 (b).

Since the size of the blocks is limited and some threads at the borders of the blocks do not have enough data available to calculate the filtered value, we must create extra threads at the borders which only read pixel information and hence will not calculate a new value. This way, these threads do not need the value of neighboring threads. The set of these specific threads is called the apron (see Figure 5.12). Therefore, the conventional method cannot be optimized further when the convolution kernel size, and therefore the apron size, becomes significant compared to the image part and as a result, the blocks are fully contaminated by the apron.

Extra threads for the apron are created to avoid warp divergence. This differs from creating threads for the image parts, and leaving the loading of the apron to these threads. This way, a simplified execution path is achieved, and less branching is necessary. Two possible methods for defining the apron threads are possible: a trivial partitioning and an aligned partitioning. The trivial partitioning, as shown in Figure 5.13(a), uses as many apron threads as the apron is wide. As can be seen, alignment is not achieved, starting from the second (green) block. In this example, the filter radius is 5 wide, and the second block will therefore start at an image position not divisible by 16. This will reduce memory coalescing: the red block needs two read operations, while the green one needs three (or even more using compute capability 1). This extra load operation can be avoided by using an optimized apron size, as



(a) Conventional distribution



(b) Optimized distribution

Figure 5.13: Distribution of apron and image threads for a filter with radius 5. The first block is red (on the left, darker in black-and-white print), and the second green (on the right, lighter in black-and-white print). Overlapping parts are represented by a gradient color. (a) Trivial data distribution. The first block is aligned to a multiple of 16, allowing coalesced memory reads. However, the second block is not aligned, resulting in three memory operations, and therefore a lower memory loading and storing performance (the block starts at 22 = 5 (apron block 1) + 22 (image part = 32 - 5 $\times$ 2) - 5 (apron block 2)). (b) Setting both the total apron size and the block size as a multiple of 16. The blocks are always aligned to multiples of 16, allowing coalesced reads (the block starts at 16 = 11 (apron block 1) + 16 (image part = 32 - 16) - 11 (apron block 2)).

shown in Figure 5.13(b). In this figure, the total apron size is always a multiple of 16, and the total block size is also a multiple of 16. This will result in an alignment of all the blocks at pixel locations that are a multiple of 16. The data loading will always be coalesced as much as possible.

5.2.2.2 Filter Separation

To avoid the optimization constraints that occur when the apron size becomes rather large compared to the image part, we propose to separate the convolution kernel in horizontal and vertical filters using singular value decomposition (SVD). By decomposing the kernel in a set of horizontal and vertical single dimensional kernels, we loosen the kernel size constraint without reducing the possibility to optimize the implementation. For example, a 3×3 kernel is separated into 3 horizontal and 3 vertical filter kernels. The previously discussed apron considerations are still applicable, but the filter is reduced to a 1D filtering, and therefore the apron in one direction is eliminated. The total apron size is therefore reduced, allowing more threads to be allocated to actual result calculations.

The convolution can consequently be performed by the following steps:

1. Calculate the SVD of the original kernel K, resulting in three matrices U, D and V, where $K = U \times D \times V^T$, and D is a diagonal matrix with elements $d_1 \dots d_n$.
2. For every column u of matrix U, iterate over the following consecutive kernel convolutions:
   (a) Convolve the image with column u of U as an individual single dimensional filter.
   (b) Convolve the result with row v of V^T, multiply with d_u, and save the intermediate result as S_u.
3. Calculate the sum of every S_u, and the identical result is achieved when compared to convolving the image with kernel K, using the conventional 2D method.

The validity of this procedure is proven in appendix B

As an alternative for the standard vertical filtering, the image and convolution kernel are transposed to be able to fully exploit memory coalescing, by performing only consecutive reads.

The reduction in individual kernel size has two major advantages; exponentially releasing the constraints of the size of an optimizable filter kernel, and greatly reducing the joint memory footprint (and therefore increasing the possibility to achieve maximum processor occupancy). Since single dimensional vertical kernels do not exhibit consecutive memory reads in the linear global video memory, it is impossible to efficiently coalesce their required data communication. To tackle this problem, we transpose both the image and kernel, enabling the kernel to be convolved horizontally. Analogous to switching to SVD from the conventional

method, this technique also introduces a significant overhead, which can only be justified in the proper situation, for example when the filters are both known and can be separated in advance.

5.2.2.3 Fourier Transformation

As a final alternative convolution technique, we perform the FIR filtering using a versatile kernel as the component-wise product of the fast Fourier transformed (FFT) image and convolution kernel. After all, by transforming both to the frequency domain, the convolution between the image and kernel is transformed into a simple entry-wise product or multiplication. The result is then inversely transformed back to the spatial domain. This method does not induce any apron whatsoever, but nevertheless generates a significant amount of overhead to perform the Fourier transformations back and forth.

To be able to compute the convolution as the entry-wise product of the transformed image and kernel, proper padding needs to be preceded, to adjust both data structures to the same dimensions. The steps that need to be performed are as follows:

1. Calculate the new dimensions w and h of the image and kernel according to $w = K_w + I_w - 1$ and $h = K_h + I_h - 1$, where K_w, K_h and I_w, I_h are the kernel, respectively image width and height.

2. Pad the image and the original convolution filter to w and h with zeroes, while cyclically moving the center of the convolution kernel to the topleft position. The convolution kernel has now non-zero values at its corners.

3. Calculate the fast Fourier transform of the padded image and filter, perform the entry-wise product, and calculate the inverse transform of the result.

Thanks to the possibility of computing the convolution by the entry-wise product, apron threads have become obsolete, and warp divergence can be completely avoided. All threads in the block can therefore be used for the actual convolution. Nonetheless, the FFT requires a significant overhead that cannot be neglected. We use the optimized FFT implementation that is publicly available in the NVIDIA CUFFT library [Podlozhnyuk, 2007a], which further pads w and h to the first consecutive power of a prime number to speed up the transformation [NVIDIA, 2014c, page 5]. The advantage is that various kernel sizes therefore show an identical processing time, stepwisely increasing when w or h exceeds their first following power of a prime number.

5.2.2.4 Experimental Results

We performed experiments to determine the performance of each method. For the ease of reporting, we used square convolution kernel shapes without loosing generality, as the thread blocks can be formed in a rectangular shape – similar to the rectangular kernel shape – to

match the same ratio of apron threads versus image part threads. The various implementation techniques were examined on an image resolution of 2048×2048, using small (size 3–40) and relatively large (size 50–750) convolution kernels. The kernel separation with SVD was also tested using both a random kernel K_R and Gaussian kernel K_G. As the Gaussian is circular isotropic, i.e. uniform in all orientations, the kernel separation will result in a special (best) case scenario, generating only a single horizontal and vertical one dimensional convolution kernel. This is demonstrated by the following example. Here, a Gaussian and a random kernel are shown, together with their decompositions. As can be seen, the diagonal of the decomposition of the Gaussian kernel K_G is zero everywhere, except for the first element. When applying the separation method, this will result in zeros, after applying its accompanying horizontal and vertical filter. Because the result is zero anyway and is used in a summation, the filtering can be skipped when the value on the diagonal is zero, resulting in a more efficient processing. However, this is not the case for arbitrary filters. This can be seen in the decomposition of K_R. Here, none of the elements on the diagonal are zero. In other words, all of the filtering steps must be performed.

$$\mathrm{SVD}(K_G) = \mathrm{SVD}\left(\begin{bmatrix} 0.0232 & 0.0338 & 0.0383 & 0.0338 & 0.0232 \\ 0.0338 & 0.0492 & 0.0558 & 0.0492 & 0.0338 \\ 0.0383 & 0.0558 & 0.0632 & 0.0558 & 0.0383 \\ 0.0338 & 0.0492 & 0.0558 & 0.0492 & 0.0338 \\ 0.0232 & 0.0338 & 0.0383 & 0.0338 & 0.0232 \end{bmatrix}\right) = U \times D \times V =$$

$$\begin{bmatrix} -0.3342 & 0.9367 & 0.0373 & -0.0978 & 0 \\ -0.4863 & -0.1837 & 0.4722 & 0.0827 & -0.7071 \\ -0.5510 & -0.2344 & -0.5581 & -0.5744 & 0 \\ -0.4863 & -0.1837 & 0.4722 & 0.0827 & 0.7071 \\ -0.3342 & -0.0157 & -0.4912 & 0.8042 & 0 \end{bmatrix} \begin{bmatrix} 0.2081 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.3342 & 0.5322 & 0.7288 & -0.2718 & 0 \\ -0.4863 & 0.3010 & -0.3988 & 0.1179 & -0.7071 \\ -0.5510 & -0.6499 & 0.0269 & -0.5228 & 0 \\ -0.4863 & 0.3010 & -0.3988 & 0.1179 & 0.7071 \\ -0.3342 & -0.3366 & 0.3873 & 0.7906 & -0.0000 \end{bmatrix}$$

$$\mathrm{SVD}(K_R) = \mathrm{SVD}\left(\begin{bmatrix} 0.0509 & 0.0061 & 0.0099 & 0.0089 & 0.0410 \\ 0.0566 & 0.0174 & 0.0607 & 0.0264 & 0.0022 \\ 0.0079 & 0.0342 & 0.0598 & 0.0573 & 0.0531 \\ 0.0571 & 0.0599 & 0.0303 & 0.0495 & 0.0584 \\ 0.0395 & 0.0603 & 0.0500 & 0.0600 & 0.0424 \end{bmatrix}\right) = U \times D \times V =$$

$$\begin{bmatrix} -0.2475 & -0.5600 & 0.4131 & 0.5759 & 0.3504 \\ -0.3542 & -0.5207 & -0.7577 & -0.0111 & -0.1707 \\ -0.4641 & 0.6013 & -0.1679 & 0.6063 & -0.1652 \\ -0.5475 & -0.1183 & 0.4755 & -0.3314 & -0.5919 \\ -0.5460 & 0.1992 & -0.0298 & -0.4369 & 0.6859 \end{bmatrix} \begin{bmatrix} 0.207 & 0 & 0 & 0 & 0 \\ 0 & 0.059 & 0 & 0 & 0 \\ 0 & 0 & 0.052 & 0 & 0 \\ 0 & 0 & 0 & 0.03 & 0 \\ 0 & 0 & 0 & 0 & 0.001 \end{bmatrix} \begin{bmatrix} -0.4307 & -0.8839 & 0.0530 & -0.0884 & 0.1503 \\ -0.4309 & 0.2207 & 0.1961 & -0.7322 & -0.4370 \\ -0.4617 & 0.0890 & -0.7467 & 0.3098 & -0.3539 \\ -0.4730 & 0.3701 & -0.0798 & -0.1023 & 0.7890 \\ -0.4380 & 0.1585 & 0.6283 & 0.5913 & -0.1968 \end{bmatrix}$$

Looking at Figure 5.14(a), the conventional method clearly outperforms all other methods considering small kernels (size 3–15). These results are actually strengthened by the findings of Podlozhnyuk in [Podlozhnyuk, 2007b], which indicate that small single dimensional kernels are further accelerated by resorting to (automatic) texture caching instead of using memory coalescing. Furthermore, we notice that using the filter separation technique is only justified when kernels exhibit circular isotropic behavior, which nicely follows the common intuition. However, in case of a versatile kernel, the intersection point where SVD becomes faster than the conventional method is still slower than the Fourier transformation. The constant timing of the Fourier technique is caused by padding both the image and kernel size to the first following power of a prime number of their sum. Since the image size is rather large compared to the filter kernel, the size of the latter becomes less important. Looking at Figure 5.14(b), a counter-intuitive result can be noticed for circular isotropic convolution kernels. Although one would intuitively use separability, in case of large kernels, the versatile

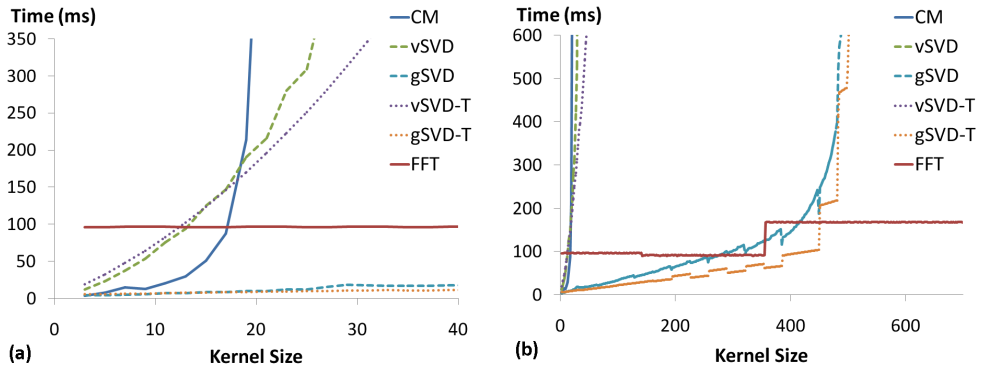


Figure 5.14: Performing the conventional method (CM), versatile SVD (vSVD), Gaussian SVD (gSVD), their transposed versions (vSVD-T, gSVD-T), and the FFT using (a) small and (b) large kernels.

approach with Fourier becomes significantly faster. This is explained by the fact that the GPU platform constraints limit the amount of threads in a single block and less threads can be allocated to perform the actual convolution when the apron size – and joint threads – become too significant. Although the transposition of the vertical filtering does not really increase the speed for small kernels, the speedup does become noticeable for larger kernel sizes, and is therefore recommendable.

Considering the design trade-off between redundant data reads and spatiotemporal utilization or processor occupancy, we performed the data distribution experiments on a 10×10 and 256×256 image resolutions. When comparing Figure 5.15(a) and (b), it can be noticed that the trade-off is linearized away when the image resolution grows. Basically, the reason is due to the limitation of 512 threads in a single block, always resulting in a sufficient number of blocks for the maximization of the occupancy, without neglecting the required memory footprint limitations of the block. However, the presented trade-off can be scaled into a future architecture, by correlating the number of blocks to the number of stream processors, and correlating the number of possible threads per block to the image resolution.

5.2.3 FIR Filtering for Demosaicing using CUDA

We will now apply the demosaicing principles of Malvar et al. [2004] on the GPU using CUDA. The algorithm is in essence a 2D linear FIR filtering using the filters from Figure 5.10. We investigated FIR filtering using CUDA in the previous section. Because the kernels are small, separating the kernels in 1D filters or using the Fourier transforms will not result in a speedup. Therefore, we will only use direct, straightforward filtering.

However, the effect of the number of threads per block and the trade-off between the total size of the apron and the occupancy must be investigated because of the specific nature of

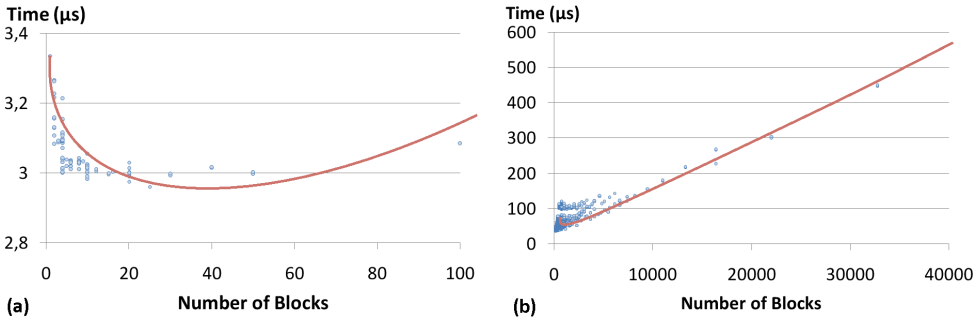


Figure 5.15: The (a) design trade-off in a small image of 10 × 10, or equivalently, massive amount of processors, and (b) its contemporary linearization (in an 256 × 256 image) due to platform constraints.

the filters. When we have a small amount of threads per block, the overall amount of threads in the aprons is large, and a lot of accesses to global memory must be made. However, as the blocks are small, this allows for multiple blocks per multiprocessor and enough blocks to utilize every available multiprocessor. When the number of threads per block is large, the overall number of threads in the aprons is smaller, but less blocks can be defined and some multiprocessors can become idle or no memory latency hiding can be employed. Therefore, we will investigate what the optimal number of threads per block is to maximize performance.

To avoid divergent branching, we defined a thread per square of four pixels. This way, every thread will process the same kind of data and no selection of the filter is needed; every thread uses all 12 filters of Figure 5.10. This differs from conventional FIR filtering and can require an adaption of the applied parameters.

We performed our experiments on 3 devices using CUDA version 5.0. The first device is an NVIDIA GeForce 8800 GT with compute capability 1.1 and 112 streaming processors at 600 MHz; the second device is an NVIDIA GTX 580 with compute capability 2.0 and 512 streaming processors at 772 MHz; the third device is an NVIDIA GTX Titan with compute capability 3.5 and 2688 streaming processors at 876 MHz. The input images have a resolution of 1600×1200. To reduce the effects of low-level process management by the operating system, we executed every configuration 5000 times and computed the average running time for a single execution.

We will present different results for all compute capabilities to stress the effect of changes in the architecture when it evolves to more advanced massive parallelism.

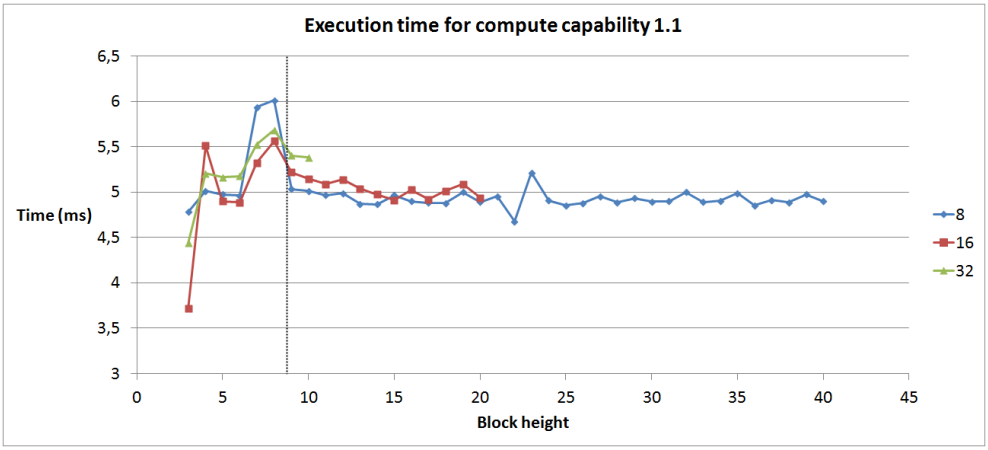


Figure 5.16: Results for compute capability 1.1. Every graph represents the width of the block, only considering coalesced configurations. The height is varied on the horizontal axis.

5.2.3.1 Compute Capability 1.1

The measured execution times of the different configurations are shown in Figure 5.16. In this graph, we only consider block widths that allow data coalescing; other widths will result in uncoalesced reads and decrease the performance severely.

Two effects are revealed: first, the general performance for very small blocks is high. This is caused by the occupancy of the multiprocessors; there are enough blocks to fill every multiprocessor and the memory footprint is small enough to allow multiple blocks per multiprocessor. Multiple blocks per multiprocessor allow for effective hiding of the memory latencies caused by starting fetches from global memory for other parallel threads. The performance is higher compared to larger blocks, which is counter-intuitive as smaller blocks result in an increased number of apron threads and therefore more memory fetches.

The second effect is the almost constant execution time after a certain block size (shown as the dotted line on Figure 5.16). Increasing the block size will prevent the allocation of multiple blocks to one multiprocessor, and will hence decrease the performance. Nevertheless, when the blocks become larger, the overall number of threads in the aprons will decrease, simultaneously reducing the amount of global memory fetches. Ergo, after the performance drop due to the reduced occupancy, the performance will increase again, but remains lower than with small block sizes.

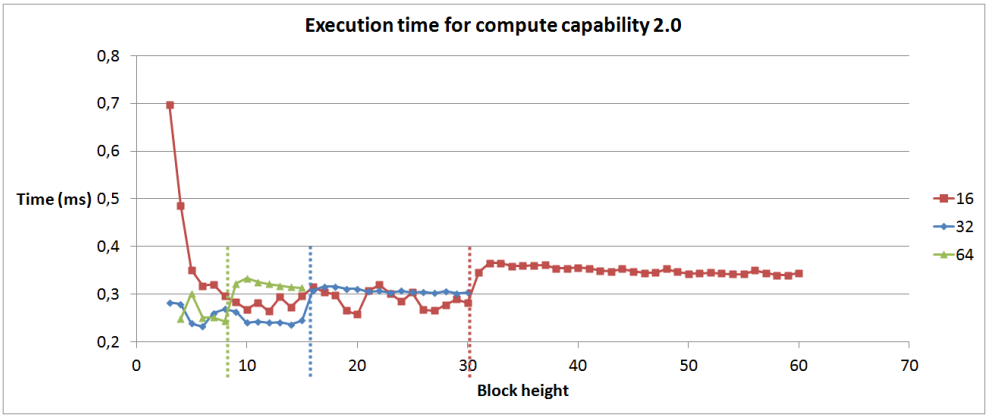


Figure 5.17: Results for compute capability 2.0. Every graph represents the width of the block, only considering coalesced configurations. The height is varied on the horizontal axis.

5.2.3.2 Compute Capability 2.0

The measured execution times of the different configurations are shown in Figure 5.17. The results are interestingly different compared to compute capability 1.1; the most performant configuration is no longer the smallest block size. The total number of simultaneous threads raises because of the increased warp size. Therefore, the total number of simultaneous global memory reads increases. The latency becomes too high to effectively hide it with more threads. Therefore, it is better to reduce the total number of memory fetches while keeping the occupancy as high as possible by making sure that multiple blocks per multiprocessor can be executed independently in succession.

Slightly increasing the block size does not decrease occupancy immediately. The specifications of compute capability 2.0 provide more flexibility, therefore allowing larger block sizes. Therefore, we see a more distinct effect on the occupancy, which is clearly visible in Figure 5.17. This phenomenon is apparent for block sizes of 64x9 (576 threads), 32x16 (512 threads) and 16x31 (496 threads), where the performance suddenly drops significantly (shown as the dotted lines on Figure 5.17). The reason for this is that less blocks are allocated per multiprocessor, therefore impeding on the advantage of memory latency hiding.

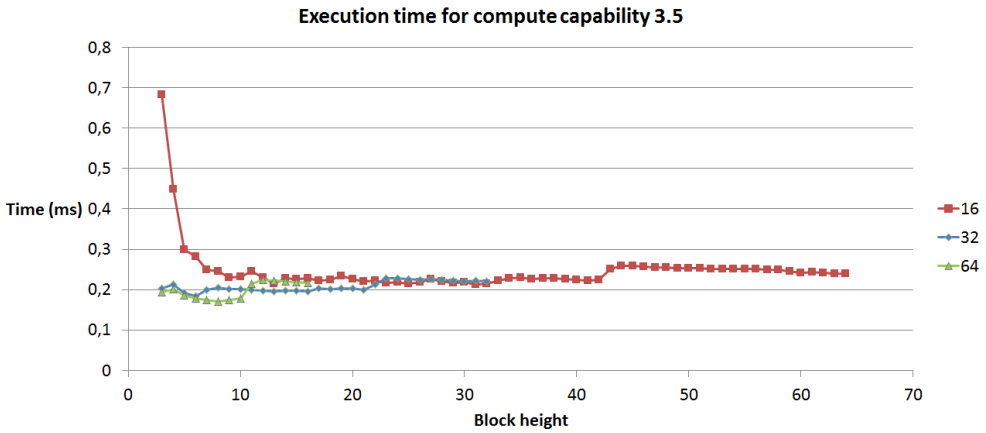


Figure 5.18: Results for compute capability 3.5. Every graph represents the width of the block, only considering coalesced configurations. The height is varied on the horizontal axis.

5.2.3.3 Compute Capability 3.5

The measured execution times of the different configurations are shown in Figure 5.18. The results are similar to the results of compute capability 2.0, and the same effects can be seen. The difference between compute capability 2.0 and 3.5 does not result in different conclusions. In fact, only the location of the effects change. In this case, the performance drop due to the occupancy decrease is noticeable for block sizes of 64x11 (704 threads), 32x22 (704 threads) and 16x43 (688 threads).

5.2.3.4 Conclusions

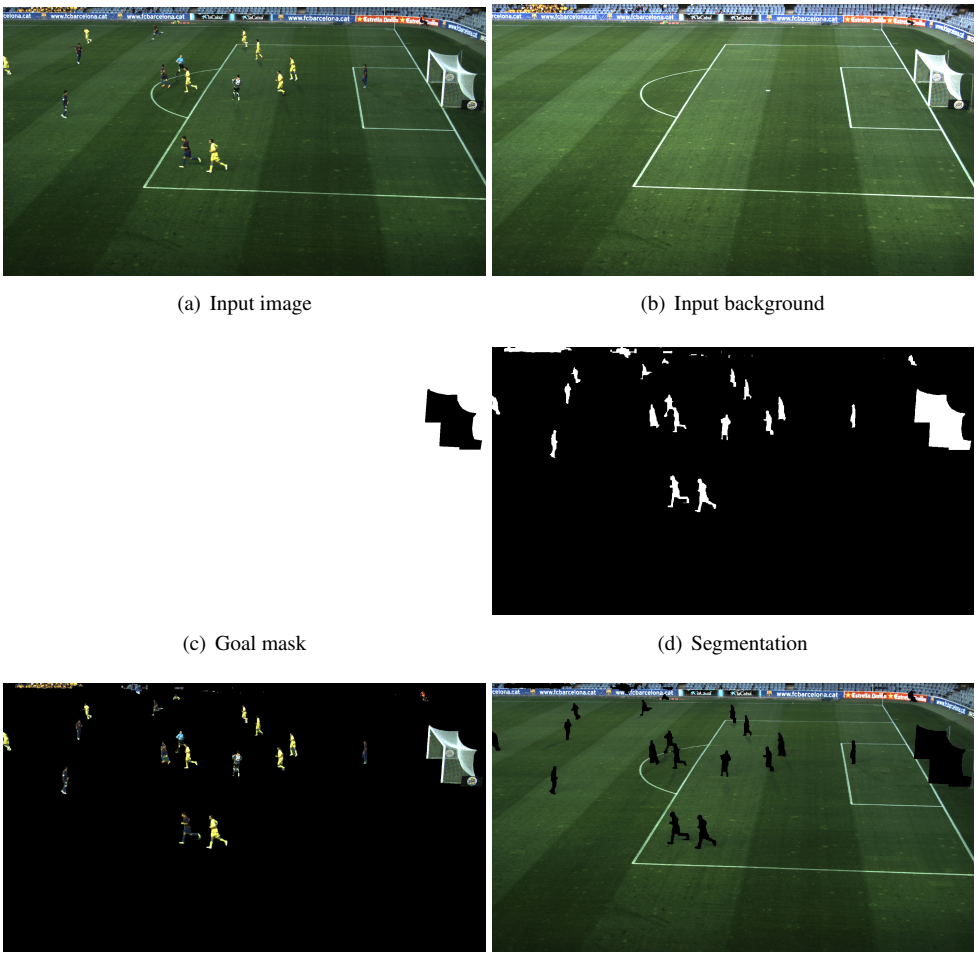
In view of the conclusions of section 5.2.2.4, we used the conventional FIR filtering for debayering on CUDA. Because we use 12 small filters in one thread, instead of 1 filter per thread, optimal division of threads will differ from the generic FIR filtering approach. A technique of using 12 small filters in one kernel leads to a result that is similar to using just 1 filter due to the similar memory pattern and the high cost of memory accesses compared to calculations. The block size, however, is specific to this application and to the compute capability. We found that there is a clear difference between compute capability 1.1, 2.0, and 3.5 when choosing the block size.

Compute capability 1.1 has strict rules for coalescing and actual achievable occupancy. Therefore, it is more performant to hide memory latency and raise the performance by using small block sizes, even if by doing so the memory accesses increase significantly.

Compute capability 2.0 and 3.5 allows more simultaneous threads and has more flexibility. More threads and flexibility automatically increases the occupancy by allowing more blocks per multiprocessor. However, the throughput of the threads is too high to effectively

hide all memory latency, with the result of lower occupancy for very small block sizes. Therefore, the number of memory fetches can be decreased without affecting the occupancy, which increases the performance. This is valid until a specific threshold is reached. Crossing this threshold, the number of blocks per multiprocessor decreases, and the performance drops significantly.

These different results prove that the memory wall for systems with slow memory and fast processors, as stated by [Asanovic et al., 2006], still holds and that the effect becomes more distinct when the individual processor capabilities increase, and the number of processors increases faster than the speed of the memory. The trade-offs between processing and memory accesses are important and must always be properly investigated to reach maximum performance. We investigated this effect experimentally for debayering, and results may differ for each application if memory patterns change.



(a) Input image (b) Input background

(c) Goal mask (d) Segmentation

(e) Foreground (f) Background

Figure 5.19: Results of the foreground/background segmentation process. (a) The input frame I_i of the process. (b) The input background B_i of the process. (c) The mask used to determine the goal position. (d) Result S_i of the segmentation. (e) The foreground pixels. (f) The background pixels. The shadows are still present in the background, while the other foreground pixels are effectively segmented from the background. Shadows are important in the final result, because they provide visual clues of the location of the players; when no shadows are present, players seem to float in the air. We keep the shadows in the background to avoid an interpolation step in the foreground rendering. This is valid, because shadows are merely darker background.

5.3 Segmentation

To allow a separate foreground and background processing, segmentation must be applied. We use a threshold-based approach to separate foreground and background, based on automatically generated backgrounds of each view.

5.3.1 Background Generation

To allow foreground/background separation, a background B_i of each camera view is required. We acquire the background by using a per pixel median filtering approach, i.e. calculating the median per pixel per color channel for 30 images, spaced 2 seconds apart from each other. Due to the movement of the scene, background becomes visible in most of the images, for every pixel. Median filtering will therefore effectively select the backgrounds for that sequence of images. A result of a single view is shown in Figure 5.19(b).

Next, a mask is created for the goal, as can be seen in Figure 5.19(c). The goal will be considered foreground to avoid projective distortions, as discussed in Section 6.1. The pixels marked as goal will never be used to generate the background, but are used to separate foreground and background. Because both the goal and the players are considered as (the same) foreground, no special treatment is applied.

When the lighting changes gradually, the foreground and background subtraction may introduce errors. Too much will be detected as foreground, resulting in low quality foreground interpolation. This subtraction error is caused by a too large difference between the previously determined background and the current, changed background. To reduce the effects of gradual lighting changes, the background used for subtraction is continuously updated with the current detected background pixels. By only updating the background, the foreground pixels are ignored. These will be updated when the foreground has moved. This methodology can only work for gradually changing backgrounds, which is a valid assumption due to the nature of changing lighting conditions in outdoor scenes, such as twilight.

5.3.2 Foreground/background Separation

Given the backgrounds and the debayered input frames, a foreground/background segmentation can be performed. The debayered images I_i are segmented in foreground and background pixels, represented by the segmentation images S_i. These segmentation images are based on the backgrounds B_i, as obtained in Section 5.3.1. Segmentation is performed on a per-pixel basis using the differences between the color values, compared against three thresholds τ_f, τ_b and τ_a, with $\tau_f > \tau_b$:

$$s_i = \begin{cases} 1: & \tau_f < & \|c_i - b_i\| \\ 1: & \tau_f \geq & \|c_i - b_i\| \geq \tau_b \text{ and } cos(\widehat{c_i b_i}) \leq \tau_a \\ 0: & & \|c_i - b_i\| < \tau_b \\ 0: & \tau_f \geq & \|c_i - b_i\| \geq \tau_b \text{ and } cos(\widehat{c_i b_i}) > \tau_a \end{cases} \tag{5.1}$$

where $s_i = S_i(x,y)$, $c_i = C_i(x,y)$ and $b_i = B_i(x,y)$, for all pixels (x,y). $\widehat{c_i b_i}$ is the angle between the foreground and background color vectors. This method allows fast segmentation in high quality. τ_f and τ_b allow the determination for very large or very small differences, while τ_a considers more subtle color differences. Furthermore, we use the mask of the goal to set all of the pixels of the goal to foreground. We want to actually interpolate the goal to cope with moving parts and perspective differences in the images. This effect is further discussed in section 6.1 and shown in Figure 6.4.

Finally, the segmentation is enhanced by an erosion and dilation step to reduce errors caused by input noise [Yang and Welch, 2002]. The result is shown in Figure 5.19(d). Any segmentation errors left in the result are not necessarily a problem. If they do not occur in every view, they will be ignored by the interpolation step further on.

The thresholds should be chosen such that the foreground objects, like players and the ball, are considered foreground, but the shadows are considered background (see Figure 5.19(f)). The shadows are in essence a darker background, while foregrounds do not correlate with the background, thus allowing the use of thresholds. We do not consider shadow as foreground to reduce interpolation artifacts caused by mismatches in shadow regions. Shadows are located on the background, thus eliminating the need for separate interpolation; the shadows are interpolated together with the background.

To reduce artifacts and increase performance, spectators are always considered as background. This is achieved using a segmentation mask, similar to the one used for the goal.

Due to the highly local nature of the thresholding method, efficient implementation in Cg is possible. This way, use of the texture cache can be maximized. Therefore, we optimally use the GPU capabilities.

5.4 Conclusions

In this chapter, we presented the preprocessing steps required for our free viewpoint video method. We require geometric calibration and we therefore present a robust method specifically for our method. Every frame that is processed in our method also requires a debayering step and a segmentation step. We used a FIR filtering approach to acquire debayered images, and investigated the optimal approach for GPU processing. These FIR filtering conclusions can be used for other applications as well.

Chapter 6

View Interpolation

6.1 Background Rendering 85
6.2 Foreground Generation, phase 1: Plane Sweeping 89
6.2.1 Applying Plane Sweeping to Soccer Scenes 90
6.2.2 Soccer-specific Plane Sweep Considerations 91
6.2.2.1 Aggregation Windows 92
6.2.2.2 Number of Cameras Used 93
6.3 Foreground Generation, phase 2: Depth Filtering 94
6.3.1 Connected Components 96
6.3.2 Median-based Depth Selection 99
6.3.3 Histogram-based Depth Selection 99
6.3.4 Background Depth Filtering 101
6.4 Foreground Generation, phase 3: Depth-selective Plane Sweeping 101
6.5 Merging Foreground and Background 103
6.6 Conclusions 103

Once the preprocessing of the images is done, novel viewpoint generation can be performed. The input is a virtual viewpoint location with image I_v and the segmented and debayered camera input images I_i. The previously generated backgrounds B_i are used for the virtual background rendering. The complete rendering pipeline is shown in Figure 6.1.

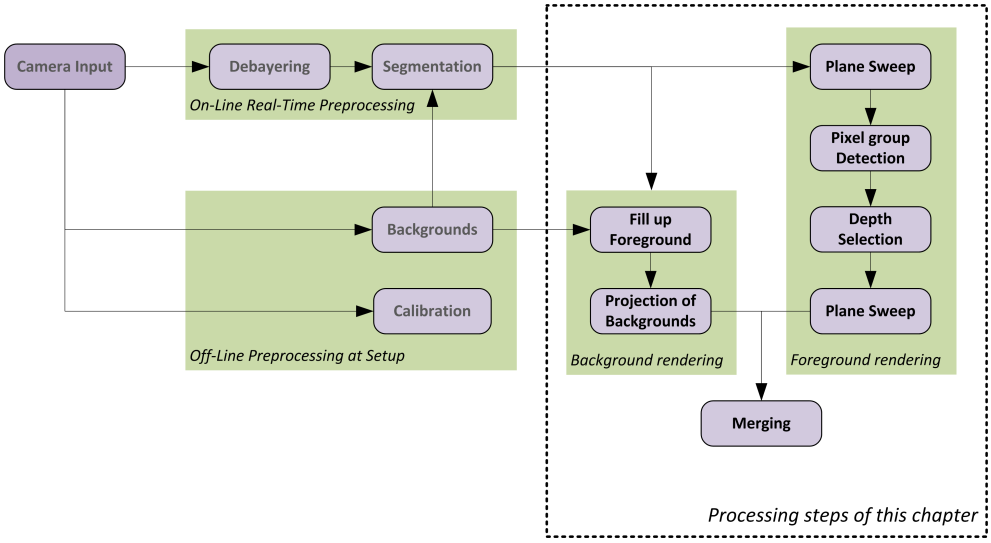


Figure 6.1: Overview of our method for the rendering. Both the preprocessing and rendering phase are shown. This chapter discusses the foreground and background rendering.

We generate the novel virtual image I_v by rendering the foreground F_v and the background B_v separately. The foreground F_v is rendered in three phases. First, we use a plane sweep approach to generate a crude depth map of the scene (section 6.2). Second, we employ a depth selection method to determine which depth values for which pixels are acceptable (section 6.3). These results are used in the last phase, where we use a second, depth-selective, plane sweep to generate the final foreground image F_v (section 6.4). These three steps will effectively handle the artifacts that are common for soccer interpolation, as discussed in section 6.2.1.

The background B_v is rendered by projecting the backgrounds onto the pitch plane. Shadows are also considered in this step. As a final step, foreground and background are merged to the final result I_v. This is discussed in section 6.1.



(a) No shadows



(b) With shadows

Figure 6.2: (a) Using backgrounds directly to render the virtual background. No shadows are preserved, resulting in unpleasant results. (b) Replacing background pixels with the color of the current frame. Shadows and lighting effects are preserved.

6.1 Background Rendering

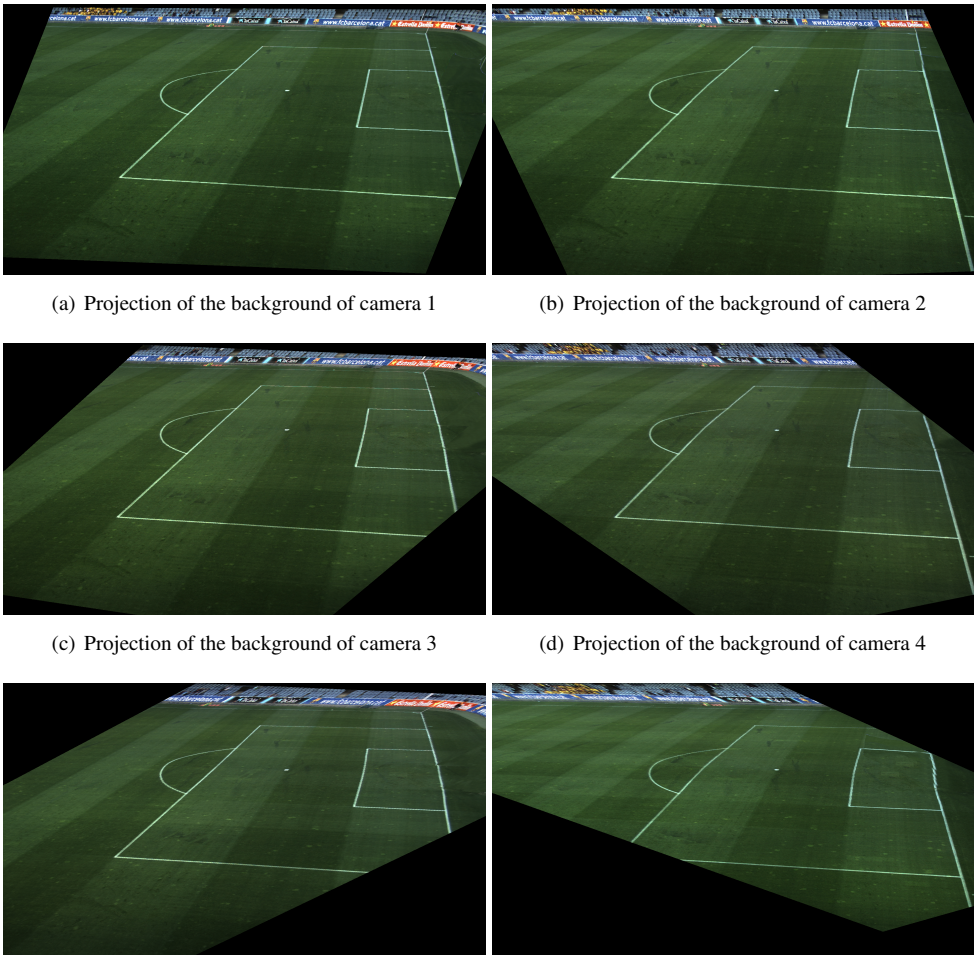
To generate the background B_v of the virtual image, we generate the backgrounds of every input image. The foreground pixels of the input images I_i are replaced by the corresponding pixels of the backgrounds B_i. This way, we have the backgrounds of every input stream, where the shadows and lighting effects are still present. Because the shadows are considered background during the segmentation step of section 5.3.2, generated backgrounds cannot be used directly. Without reasonably correct shadows, objects seem to float, resulting in an unpleasant result, as shown in Figure 6.2.

To solve this problem, backgrounds are deprojected on a virtual plane, coplanar to the pitch, and reprojected on the virtual camera image. We know the location of the pitch thanks to the calibration from Section 5.1. We start with the camera closest to the virtual camera to fill the virtual image as much as possible. The other cameras complete the parts of the image that are not covered by the closest background. To provide a pleasant looking result and to compensate for color differences between cameras, smoothing is applied to the borders of the reprojected backgrounds. This way, changes from one background to the other are not visible. The different steps are shown in Figure 6.3. This figure clearly shows the problem of generating the background. None of the projected backgrounds completely fill the virtual image; blending of different backgrounds is therefore required.

The goal is considered to be foreground. If the goal was to be background, it would be projected to a plane, resulting in serious projective distortions, as can be seen in Figure 6.4. By considering the goal as foreground, the height of the goal is kept (as is the case for the players), resulting in higher quality results.

Furthermore, the depth of the virtual background is stored for further use in the foreground rendering.

Even though the background has a decent quality, some artifacts can be perceived in the bleachers. Other methods, such as the method of Li and Flierl [2012] will increase the quality, but decrease performance. Here, the goal, pitch, and other background objects are modeled and can be reconstructed for a novel viewpoint. We opted for a more performant method, instead of a costly reconstruction.



(a) Projection of the background of camera 1

(b) Projection of the background of camera 2

(c) Projection of the background of camera 3

(d) Projection of the background of camera 4

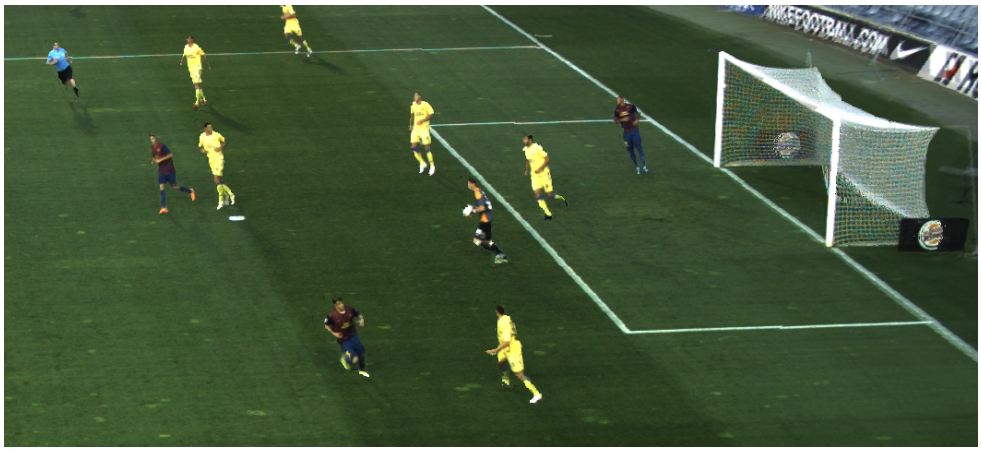
(e) Projection of the background of camera 5

(f) Projection of the background of camera 6

Figure 6.3: Backprojection of the backgrounds of different camera positions. A single camera cannot provide a complete virtual image; blending of the backgrounds is required.



(a) Goal as background



(b) Goal as foreground

Figure 6.4: Comparison of considering the goal as background or foreground. In (a), the goal is considered background. Serious projective distortions are perceived. In (b), no projective distortion is perceived. The goal is interpolated using the plane sweep method, described below.

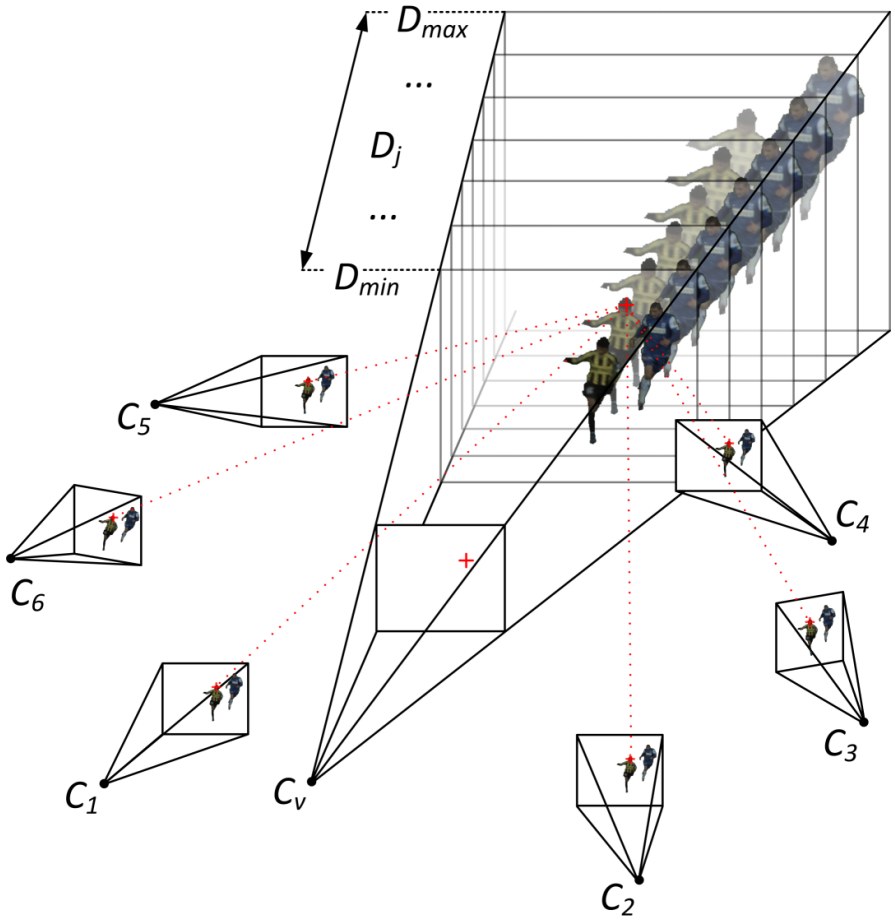


Figure 6.5: Principle of plane sweeping. The space before the virtual camera C_v is discretized in planes. For every depth D_j, every pixel of the virtual view is deprojected on this plane and projected back onto every input camera $C_1 - C_6$. Using these color values, a cost error ε can be calculated, from which the optimal depth for that virtual pixel can be determined.

6.2 Foreground Generation, phase 1: Plane Sweeping

The first phase of the foreground rendering is performed using a global plane sweep approach to generate a crude depth map and segmentation of the virtual viewpoint. Our plane sweep is a modified version of the well-known method from Yang et al. [2002]. The world before the virtual camera is divided into M planes of depths $D_p \in [D_{min}, D_{max}]$, parallel to the virtual image plane, as shown in Figure 6.5. The distribution of the planes in the range $[D_{min}, D_{max}]$ will be discussed in chapter 7. For every plane, every pixel $I_v(x,y) = [x_v, y_v, 1]$ of the virtual camera image is deprojected on this plane, reprojected on the U selected input images and the error ε is calculated per pixel and per depth plane using the sum of squared differences (SSD):

$$[x_i, y_i, 1, 1]^T = P_i P_v^{-1} [x_v * D_p, y_v * D_p, D_p, 1]^T \tag{6.1}$$

$$\varepsilon(x_v, y_v, D_p) = \sum_{i=1}^{U} \frac{\|\gamma - I_i(x_i, y_i)\|^2}{3U} \text{ with } \gamma = \sum_{i=1}^{U} \frac{I_i(x_i, y_i)}{U} \tag{6.2}$$

where P_i is the 4×4 projection matrix of input camera i, P_v the 4×4 projection matrix of the virtual camera, γ is the average of the reprojected pixels and I_i is the ith input image of the total of U input images. When a reprojected pixel falls outside the segmentation mask, the error ε for that depth is set to infinity. This will guarantee consistency with the segmentation. The resulting depth d_l for a virtual pixel is the depth plane on depth D_l with the lowest error ε for that virtual pixel, and the color is the average γ of the corresponding pixels in the input images. When every ε has a value of infinity, the pixel in the virtual image is considered background. The calculation of the depth map and the resulting color image can be efficiently implemented using Cg on graphics hardware by exploiting the projective texturing capabilities, resulting in real-time processing [Dumont et al., 2009]. The usage of texture memory in Cg shaders results in efficient device-managed cache usage, presuming memory accesses are relatively local. Furthermore, depth testing is used to preserve the minimal value of ε and its corresponding color value γ, therefore exploiting the built-in capabilities of the graphical engine. While plane sweeping is possible in CUDA, the performance is worse than using traditional GPGPU technologies due to the missing projective texturing and depth testing.

We compared the performance of plane sweeping in CUDA and in Cg shaders. Only pure plane sweeping is tested, without the filtering discussed in subsequent sections. The CUDA implementation used global memory to store the input images, the intermediate depth and color values, and the final result. The depth buffer is not writable from CUDA, and is therefore not used.

We used 6 cameras and 521 planes, using images of 1600x1200. The Cg implementation had a running time of 79*ms*, while the CUDA implementation took 754*ms*. The memory



Figure 6.6: Plane sweeping on soccer scenes using conventional plane sweeping without segmentation. Many artifacts can be perceived.

accesses of the CUDA implementation are highly scattered, because the required image information is not located on scanlines. This makes coalescing the memory accesses problematic, resulting in slow processing. The Cg implementation allows the use of the built-in caching facilities, which can handle the scattered load much better.

6.2.1 Applying Plane Sweeping to Soccer Scenes

When applying conventional plane sweeping to soccer scenes, some artifacts can be perceived.

First, when applying plane sweeping on the complete input images, i.e. without segmentation, the quality is unacceptable. The foreground, i.e. the players, are mixed with the background, while the background seems blurred and full of speckle noise. This can be seen in Figure 6.6. The main reason is the matching between the green pixels in the background. These can be matched anywhere on the background with a low SSD error ε. When ε happens to be lower than the error values found while matching foreground pixels, background pix-



(a) Intermediate position (b) Intermediate position, depth map

Figure 6.7: (a) Plane sweeping on soccer scenes using conventional plane sweeping with segmentation. Ghost limbs can be perceived. (b) The depth map of the color image.

els will overrule the foreground and artifacts occur. This happens when color values do not match perfectly between camera images. Therefore, segmentation is applied.

When using segmented images, other artifacts may occur. These include extra limbs on interpolated players. When the left leg in one image is matched to the right leg in another image, for example, matching with low SSD values is possible and ghosting occurs. This is shown in Figure 6.7. However, the depth values of these extra limbs is different, which allows filtering of these kind of artifacts. This is discussed in section 6.3.

A third, important artifact occurs when one player matches with another. In this case, a ghost player can be perceived. This is depicted in Figure 6.8. These artifacts are different from ghost limbs, as these ghost layers are not connected to the correct players. The processing method is therefore different for both these artifacts.

6.2.2 Soccer-specific Plane Sweep Considerations

Due to the specific nature of soccer scenes, more performant and higher quality results can be achieved by using several specialized approaches. The following subsections subsequently discuss the use of aggregation windows and the impact of the number of used cameras.



Figure 6.8: Plane sweeping on soccer scenes using conventional plane sweeping with segmentation. Ghost players can be perceived. All 8 cameras were used.

6.2.2.1 Aggregation Windows

Typically, when using plane sweeping or other depth estimation algorithms, an aggregation window is used [Scharstein and Szeliski, 2002]. Here, an error value σ for a pixel is determined by aggregating the error values ε in a window around the pixel using a weighting function w with coordinates (u,v):

$$\sigma(x,y) = \sum_{u,v} w(u,v)\varepsilon(x+u,y+v) \tag{6.3}$$

The weighting function can be, for example, a Gaussian kernel [Scharstein, 1994] or a non-linear selection kernel [Zhang et al., 2009a]. While these methods proved to yield good results, these are not required for our intended application. The depths of the players are relatively uniform, therefore relaxing the requirement for a dense depth estimation. Depth errors due to occlusion at the borders are relatively small. Instead, other artifacts, such as ghost players, are more prominent. These artifacts are not handled by using aggregation windows, and other strategies must be used.

Small depth errors, limited to a small number of pixels, tend to be eliminated by using aggregation windows. Even though this can be the case in our application, other depth filter-

ing approaches, used further on, tend to remove these errors, therefore reducing the need for aggregation windows.

Furthermore, our intended result is not a depth map of the scene, but a novel viewpoint. Lastly, using aggregation windows requires more processing power, which is in direct conflict with the requirement of fast processing.

6.2.2.2 Number of Cameras Used

The number of used cameras, denoted by U, is dependent on the setup used. When using a linear setup, where all cameras are relatively close to each other and all have parallel view directions, many cameras can be used. In our setup, all 8 cameras are used ($U = N$). This is possible due to the limited occlusion and uniform look angle.

When using a curved setup, only two cameras ($U = 2$) are used for the color operations, i.e. calculating γ and ε. In this setup, the cameras are placed further away from each other, therefore increasing the possibility of occlusions. Furthermore, due to the placement in an arc, all cameras observe a different part of the scene, e.g. the leftmost camera can see the front of a player, while the rightmost will see its side. Calculating the SSD using all camera images will not yield correct results, due to the different portions of the scene perceived (see Figure 6.9). However, the segmentation images S_i of all camera images are used to perform a foreground/background check. All projected pixels of the virtual image are also reprojected on the segmentation masks of the unused cameras. If the pixel is projected to the background, ε is set to infinity.

While different in method and final result, some comparison can be made with the visual hull approach [Matusik et al., 2000]. The visual hull approach uses a foreground/background separation, where the foreground is called the silhouette. In 3D space, a cone, called the silhouette cone, is formed by the camera position and the silhouette on the image plane. The borders of the cone are in essence the projection rays of the foreground objects on the image plane through the camera center. By intersecting the silhouette cones, a reconstruction can be made. The scene object must be contained in the intersection.

We use the same principles to eliminate artifacts in the curved setup. We do not use the visual hull approach to generate a 3D reconstruction. Instead, we merely use the principles to aid in the image-based method.

This reprojection on segmentation approach is effective for removing artifacts in the curved setup. This can be seen in Figure 6.10. Artifacts that are not projected on foreground pixels in the unused cameras are effectively removed. This is less effective in a linear setup, where all cameras have the same view angle on the scene, and therefore provide less added value for artifact filtering based on depth. However, more cameras for usable color information are available in the linear setup, while this is limited to two for the curved setup. A trade-off can then be considered between the number of the cameras used for color information and the number of the cameras used for depth information based on foreground/background segmentation.



(a) Leftmost image

(b) Rightmost image

(c) Intermediate position, all cameras used

(d) Intermediate position, 2 cameras used

Figure 6.9: Comparison between using 2 or all cameras for the curved arrangement. (a) The leftmost input image. (b) The rightmost image. As can be seen, a different portion of the scene is visible, making the use of color information more difficult or impossible. (c) Using view interpolation with all cameras. The result is noisy and contains many artifacts. The wrong colors are used to generate the novel viewpoint, resulting in erroneous results. (d) Using view interpolation with 2 cameras, including the segmentation information of the other cameras. The result is sharper and many artifacts (especially around the borders) are eliminated.

6.3 Foreground Generation, phase 2: Depth Filtering

At this point, we have a normalized depth map and a foreground rendering of both the foreground and the background. However, the foreground contains many artifacts. Therefore, we employ a depth selection method to effectively eliminate these artifacts. We notice that the depth values of the artifacts are very different from the correct depth values. We can use this information to determine which depth values are valid and which are not. Furthermore, we notice that most artifacts are connected to the pixels of the correct results, for example a third leg of a player. Therefore, we employ a strategy that will consider the depths of



(a) Intermediate position, no segmentation check



(b) Intermediate position, depth map, no segmentation check



(c) Intermediate position, with segmentation check



(d) Intermediate position, depth map, with segmentation check

Figure 6.10: Comparison of using the segmentation as a guideline. (a) Using 2 cameras, without using the segmentation of the other cameras. Background depth filtering is turned off. There are some ghost players visible. (b) The depth map of subfigure (a). As can be seen, the depth of the ghost players is different from the depth map of the correct players. This indicates that the ghost players are located on a different place than the other players. (c) Using 2 cameras, with the segmentation of the other cameras. Background depth filtering is turned off. As can be seen, the ghost players are effectively removed. (d) The depth map of subfigure (c).

groups of pixels simultaneously. Furthermore, we will consider the normalized depth of the background, determined in section 6.1, to remove excessive foreground depth values, such as players floating high in the air or buried in the ground. The result is a list of valid depth values per pixel, which will be used in a second, depth-selective plane sweep.

We will now discuss the depth selection strategy. First, the depth map is labeled in groups of connected pixels, where each group has a unique label. This is done by using a parallel connected components strategy. Then, for every group, i.e. for every label, valid depth values are chosen. This is done by using a median-based or histogram-based approach. Lastly, the background depth is incorporated to eliminate excessive depth values.

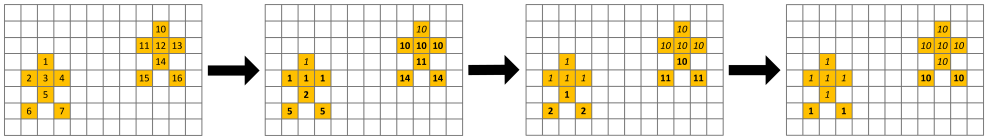


Figure 6.11: Detection of connected pixels: a thread is assigned to every pixel. For every iteration, the threads compare the label of its neighboring pixels with its own and stores the lowest label. All connected pixels have the same label if no more changes are made.

6.3.1 Connected Components

To uniquely label every group of connected pixels, we applied a connected components method using CUDA. Typically, a CPU scanline implementation is employed [Di Stefano and Bulgarelli, 1999], but it is unfeasible to transfer image data back and forth to and from the GPU to employ a CPU implementation. Therefore, we propose a GPU region growing approach to acquire high performance processing.

Initially, we apply a unique label δ to every pixel and set background pixels to zero. Next, we compare every pair of neighboring pixels. If one of the two labels is zero, nothing happens. If the labels are greater than zero and different, both labels are set to the smallest of the two. By repeating this process until no changes are possible anymore, all connected pixels will have the same label. This is shown in Figure 6.11.

This method can be mapped to the CUDA model. Every pixel is assigned a thread on the GPU, so that these are processed in parallel. Because only neighboring pixels are considered, fast memory access strategies can be employed by only using localized memory accesses. This allows the use of fast, thread-shared memory and reduces the amount of slow copies from global memory [Goorts et al., 2009]. Swapping is done in shared memory instead of global memory to allow fast reading and writing of results. After no swaps can be done anymore in a thread block, results are written to global memory. Then, another iteration is done with different block sizes to allow propagation of labels across thread blocks. By using a combination of iterations in shared memory and iterations in global memory, fast processing can be achieved by reducing expensive memory operations.

The 8 least significant bits of δ are reserved for the normalized depth value of the pixel, to be used in a reduction step later on. We chose the initial δ values such that the least significant bits are unused and can be used for the depth values. When comparing and assigning labels, depth values are ignored:

$$d = \delta' \,\&\, 0\text{xff} \tag{6.4}$$

$$\delta = \delta' \,\&\, \sim 0\text{xff} \tag{6.5}$$

where &, $|$, and $\sim$ are the bitwise and, or, and not operations, respectively.

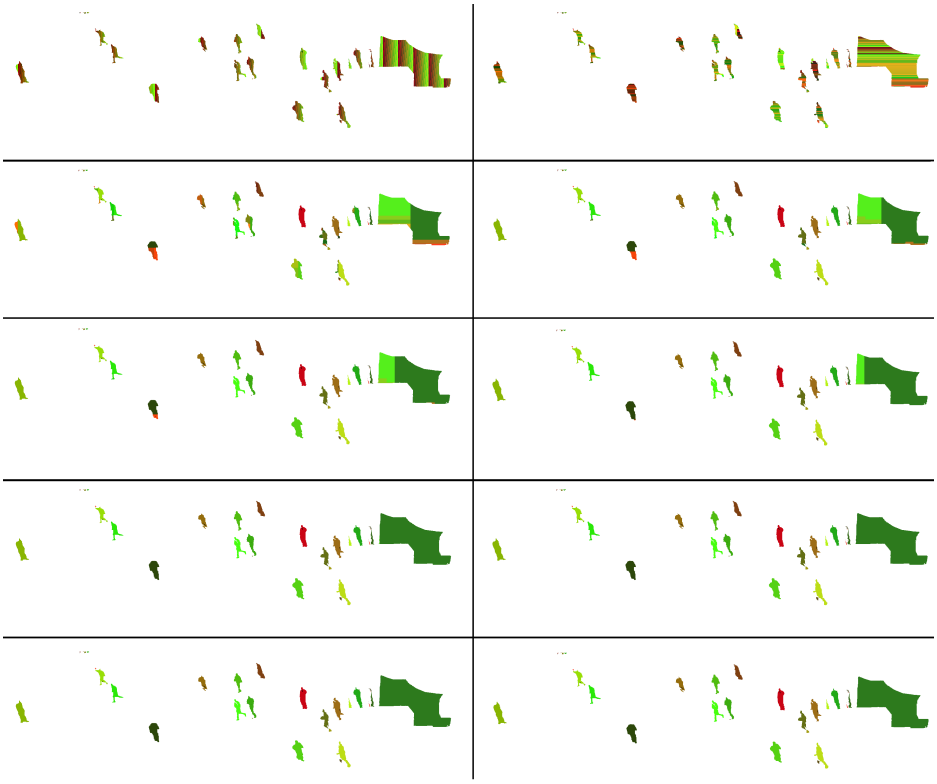


Figure 6.12: Different iterations of the connected components algorithm using CUDA. Images go from left to right, then from top to bottom. Every distinct color is a different label. As can be seen, label propagation is achieved in a group of connected pixels. Apparent borders inside a group are caused by the thread blocks.

When comparing labels, δ is used. When storing the labels, the depth is appended again:

$$\delta' = \delta \mid d \tag{6.6}$$

The final result is that every foreground pixel has a label, where connected pixels all have the same label, and all non-connected pixels have a different label. The different steps are depicted in Figure 6.12. In this figure, the different iterations are shown, where each color is a different label. At the final stage, each connected group of pixels has the same color, and therefore the same label.

Next, we sort the list of labels, appended by their depth values, and apply a reduction by key, where the key array is the array of labels δ', and the value array is 1 everywhere. This way, all unique label values, appended by depth values, are counted. In essence, this is a histogram of depth values per label.

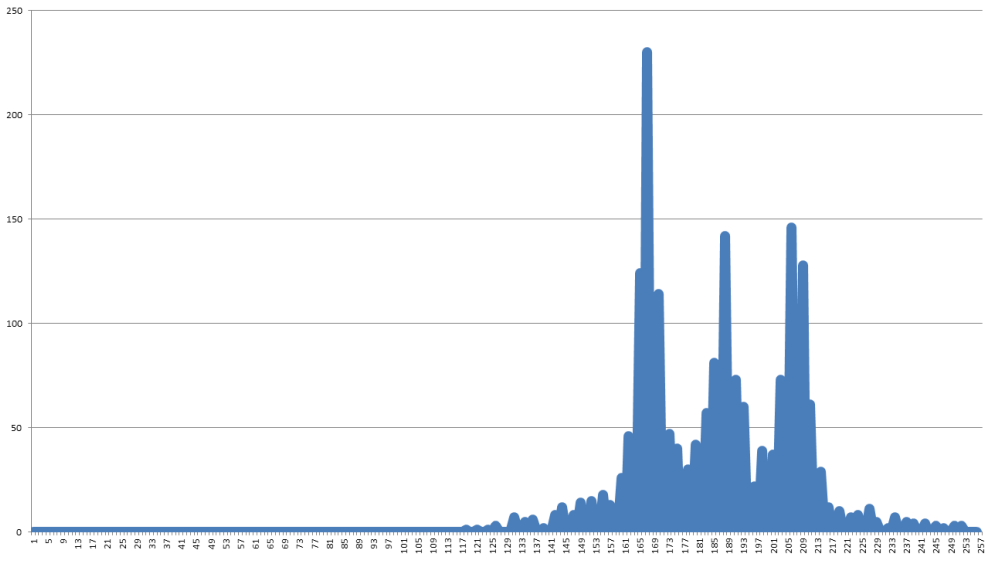


Figure 6.13: The unfiltered histogram. Three peaks can be seen, representing the depth values of three players. Also, a lot of noise can be seen.

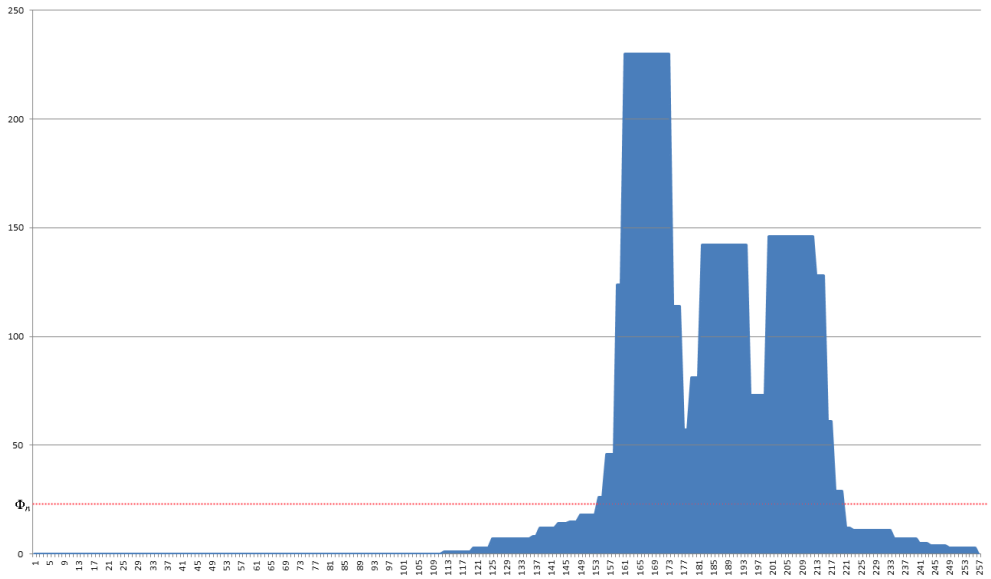


Figure 6.14: The filtered histogram. Three peaks can still be seen, representing the depth values of three players. The noise, however, is reduced. Only depths with values above Φ_n are considered for the next, depth-aware, plane sweep.

As a first filtering step, all histograms with a total count value lower than a threshold of Φ_h are removed. In essence, all pixels with the corresponding label are now considered background, because no depth values will be considered valid later on.

Lastly, these histograms per label are used to calculate the allowed depth values for that label δ. Two strategies are possible: a median-based and a histogram-based. Both strategies create one or more validity maps V, which contains, per pixel, a depth value or a zero, which is used in the depth-aware plane sweep phase.

6.3.2 Median-based Depth Selection

In the median-based approach, we create a single validity map V_m, where each pixel receives a depth value or a zero. For each label, we calculate the median value and set the corresponding pixels in V_m that have that label to the median value. Background pixels are set to zero. We will only allow normalized depth values that differ less than a threshold Φ_m from the median normalized depth value for that pixel.

6.3.3 Histogram-based Depth Selection

In the median-based method, depths were filtered, so that only a range of depths was allowed in the interpolation. This depth is different for every group of connected pixels and is determined by the median depth value. This proved to work well in the provided data, even if different foreground objects overlap in the virtual viewpoint, and thus form one group of pixels. However, as the players are in reality a large distance away from each other, but still overlap, the players with the least pixels visible in the virtual viewpoint can disappear. Indeed, the difference between the median of the group of pixels and the median depth of the player is larger than the threshold. This will effectively filter out the player and therefore remove it completely. The effect is mainly seen in image sets where the camera distance is large, where there are a lot of players simultaneously on the field and where the field of view is large.

To cope with this aspect of the algorithm, we can alternatively consider a number of depths. Instead of calculating the median per group of pixels, we consider the histogram per group of pixels and select the peaks herein that are considerable relative to the amount of pixels. See for example Figure 6.13. This is the depth histogram for three players, merged in one group of pixels. If we select only one depth, some peaks will disappear, including the corresponding players.

To allow multiple valid depths per group of pixels, we apply an envelope operation on the previously calculated histograms to reduce noise and to merge close-by peaks together. We do this by considering a range Φ_e around a value in the histogram and saving the maximum of all the values in the range. This can be seen in Figure 6.14.

We now have a set of filtered histograms, one for each group of pixels. Then, we generate a validity map V_d for every processed depth, encoding if a pixel is a valid value for this depth



(a) Median-based



(b) Histogram-based

Figure 6.15: Comparison of the median-based and histogram-based method. (a) Interpolation using median-based depth filtering. When compared with Subfigure (b), a player has disappeared (the closest player of the group of two players at the right side). This is caused by the currently invalid assumption that every group of pixel only has one valid depth. (b) Interpolation using histogram-based depth filtering. There are no disappearing players.

(encoded as 1) or not (encoded as 0). To reduce memory usage, the map is generated on-the-fly when executing the depth-aware plane sweep, thus only one image is available, and used, at one time instance. To generate the validity map for a given depth, we consider all the foreground pixels of the previous depth map, and compare the current processed depth with the histogram value for the corresponding group of pixels. All local maximums are considered valid, except when the value is lower than a noise threshold Φ_n.

If there is only one player in the group of pixels, there will be only one peak in the histogram that is of considerable height, and thus only depth values around this value will be considered.

Because of the extra processing to increase the quality of the final result, extra processing power is required. A trade-off can therefore be made between quality and speed. When speed is more important, some disappearing players can be perceived. For more correct results, multiple depths must be considered.

The difference is shown in Figure 6.15. In Figure 6.15(a), the median-based approach is used. In this case, however, a player has disappeared, which is not a desired result. The players are connected in the virtual image, but are placed a distance apart from each other in the scene. Figure 6.15(b) shows the histogram-based result, where no disappearing players are perceived. This demonstrates the requirement of the histogram-based method, justifying the increased processing requirements.

6.3.4 Background Depth Filtering

Lastly, all validity maps, being the single map V_m for the median-based approach or the collection of maps V_d for the histogram-based approach, are filtered to remove obvious error values, based on the background depth. The background depth is known from section 6.1. When a depth value that was considered valid differs more than a threshold Φ_b from the background depth, this value is removed. This will remove artifacts that are caused by matching a player in one camera image to another player in another camera image. This will result in a ghost player, as seen in Figure 6.16. These ghost players have depths that differ a lot from the background, allowing successful filtering.

6.4 Foreground Generation, phase 3: Depth-selective Plane Sweeping

The second, depth-selective plane sweep is equivalent to the first plane sweep, but the error value ε is set to infinity if the depth of the plane is not valid, according to the validity maps from the previous filtering phase. When using the median-based validity map V_m, a valid range is determined by the threshold α:

$$|D_p - d_l| < \alpha(D_{max} - D_{min}) \tag{6.7}$$



(a) Intermediate position, no background depth filtering



(b) Intermediate position, no background depth filtering, depth map



(c) Intermediate position, with background depth filtering



(d) Intermediate position, with background depth filtering, depth map

Figure 6.16: Artifact elimination based on background depth. (a) Interpolation without background depth filtering. Ghost players can be seen. (b) The depth of subfigure (a). As can be seen, the depth of the ghost player is very different from the background depth. (c) Interpolation with background depth filtering. The ghost players are effectively removed. (d) Depth of subfigure (c).

where d_l is the denormalized value of the validity map on the corresponding pixel. Invalid values of d_l are always considered invalid, independent of the difference.

When using the histogram-based approach, no range is used. Instead, the validity is directly read from the validity map.

In both cases, depths that are not accepted are explicitly assigned an error ε of infinity. If every error is infinite, the virtual pixel will be considered background, removing artifacts caused by mismatching.

This ensures a detailed depth map of the player, while respecting its global depth. Extra limbs and other ghosting artifacts are filtered out using this method. Indeed, these arise from errors in the matching process of the first plane sweep. For example, the left leg of one player is matched to his right leg, resulting in a third leg in the virtual view (this can be seen in Figure 6.17 (a)). However, the depth of this ghost leg is distinct from the depth of the

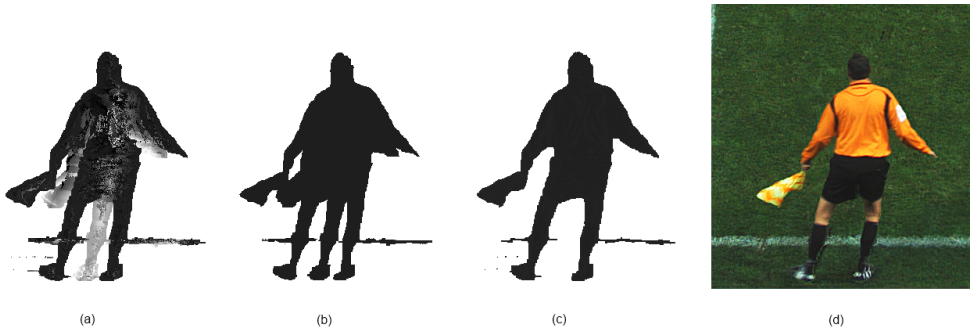


Figure 6.17: Detail of the method: (a) Depth map of standard plane sweep. (b) Filtered depth. (c) Depth map of the depth-selective plane sweep. The ghost leg and other artifacts are effectively removed. (d) Final merged result.

correct pixels of the player. Using our method, this depth is filtered out and the ghost leg is considered background. Small artifacts due to errors in the reprojection are reduced by limiting the depth range per player, thus obtaining a high quality depth map without large errors in the individual depth values.

It is possible that all error values ε for a pixel are infinity, thus resulting in background and the effective removal of artifacts. To increase performance, the number of valid pixels in the validity map is counted. If the value is zero, the plane is skipped. The final color result consists of the average color values γ where ε is minimal.

6.5 Merging Foreground and Background

The rendering of the foreground F_v and background B_v are then merged using the segmentation obtained from the second plane sweep. To generate a pleasant-looking result, the borders of the foreground are slightly feathered and blended with the background. The final results of our method are discussed in chapter 8.

6.6 Conclusions

We discussed the real-time rendering step of our method, including background and foreground rendering. By separating the foreground and background, we can use depth filtering based on the depths of the players. The discussed method for the foreground rendering is effective for removing the artifacts that we considered, including ghost limbs and ghost players. We will present more results in chapter 8

Chapter 7

Plane Distribution Optimization

7.1 Adaptive Non-Uniform Plane Distribution **107**
7.2 Results **110**
7.3 Conclusions **112**

Typically, the planes for the depth hypotheses in the plane sweeping method are distributed evenly in the scene space, thus allocating uniform computational power to all depth hypotheses. Because the scene typically does not have a uniform distribution of objects, wasted performance may be perceived by considering depth values where no objects are present. Therefore, we present an optimization where the distribution of the planes is adapted to the scene.

A histogram is calculated of the resulting depth map. This histogram will guide the plane distribution for the next temporal frame. This will redistribute computational power to the more dense regions of the scene, and consequently increase the quality of the interpolation by reducing mismatches and noise. Using the histogram of the previous temporal frame is a valid assumption due to the relatively smooth movements of the players; there is no sudden change in depth of the players. While the method is developed for soccer scenes specifically, it can be applied to any scene where the objects move smoothly or are static.

The next section will describe the method in detail.

7.1 Adaptive Non-Uniform Plane Distribution

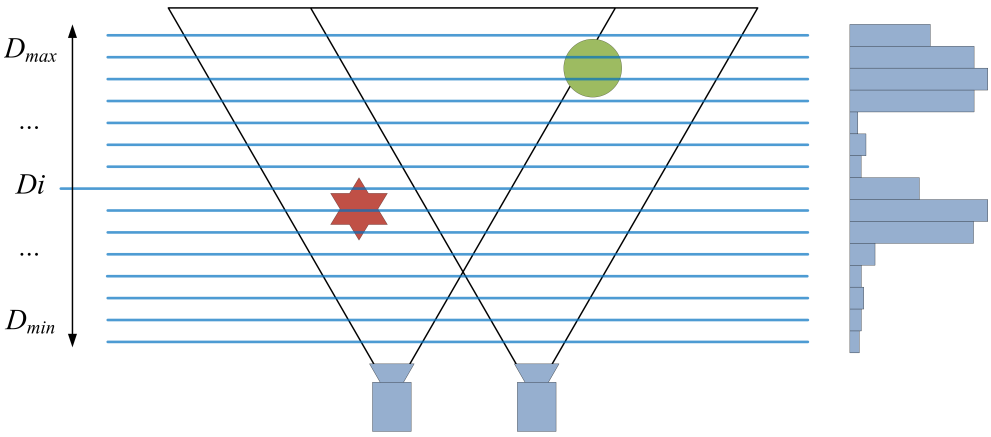


Figure 7.1: Uniform plane distribution, with the histogram of the depth values at the right.

When the scene consists of a limited range of depths between D_{min} and D_{max}, some processing resources are allocated to depth planes where no objects are present. This is demonstrated in Figure 7.1. In this figure, a lot of planes are placed in the scene where no objects are positioned. This will waste resources and introduce more noise due to mismatches between the cameras. Therefore, we rearrange the distribution of the depth planes to provide less planes in depth ranges with less objects, and more, dense planes in scene regions with more objects. We determine the interest of a depth by analyzing the previous frame in a tem-

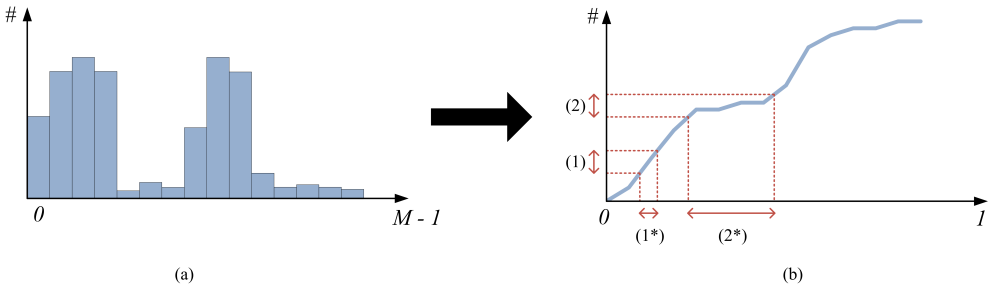


Figure 7.2: (a) Resulting histogram (b) Corresponding cumulative histogram $H(x)$.

poral sequence. The method works best when the movement of the scene is limited, such as moving people or scenes with many static objects.

After the interpolation step, we generate the histogram of the depth map using the well-known occlusion querying method [Green, 2005] on GPU, allowing fast processing. The histogram can be seen in Figure 7.1 at the right. The occurrence of every depth value, as determined by the depth of the depth planes, in the depth map is counted. The histogram will have discrete depth values between D_{min} and D_{max}, represented by the depth plane numbers, because there is a limited number of planes. Scene depths of high interest will contain more depth values than depths of low interest. If there are depths in the scene where no objects are present, few of these depth values will be available in the depth map and this will be reflected in the histogram. In the next frame, we want to provide more planes in depth ranges where a lot of depth values can be found, hence where there are large values in the depth histogram. The depth planes are not necessarily uniformly distributed, thus the histogram uses the depth plane number as the bin value, instead of the depth directly.

To use the depth distribution information, we convert the histogram to its cumulative version, as shown in Figure 7.2. Here, we do not count the number of occurrences per depth value, but we rather include the number of occurrences lower than this depth. Furthermore, we rescale the depth values from $[D_{min}, D_{max}]$, as represented by the depth plane numbers, to $[0,1]$. This will transform the non-uniform distribution of the depth planes to actual normalized depth values between 0 and 1. This transformation will generate an increasing function $H(x) = y$, where $x \in [0,1]$ is a normalized depth value and y is the number of values in the rescaled depth map smaller or equal to x. For values of x where there are a lot of corresponding values in the depth map, $H(x)$ will be steep. For values of x with a low number of occurrences, $H(x)$ will be flat. Because of the non-uniform depth plane distribution as input, $H(x)$ will be constant at some points where there were no depth planes for the corresponding normalized depth value.

We will use the cumulative histogram to determine a mapping of a plane number m with $0 \leq m < M$ to a depth value D_m with $D_{min} \leq D_m \leq D_{max}$. For a uniform distribution, this would be:

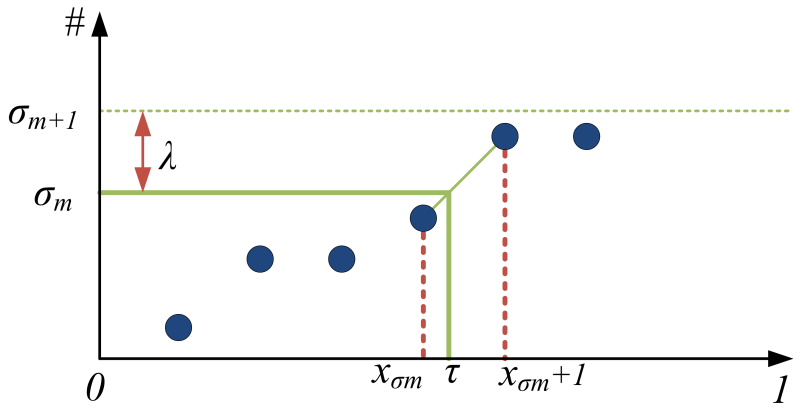


Figure 7.3: Detail of the cumulative histogram with discrete values. τ is calculated by determining $x_{\sigma m}$ and $x_{\sigma m}+1$, such that $H(x_{\sigma m}) \leq \sigma_m$ and $H(x_{\sigma m}+1) > \sigma_m$, where σ_m represents a depth plane number.

$$D_m = D_{min} + \frac{m}{M}(D_{max} - D_{min}) \tag{7.1}$$

We will adapt this uniform distribution method. When using the cumulative histogram to determine the distribution, we calculate a fraction $\tau_m \in [0,1]$ based on the plane number m, applied as follows:

$$D_m = D_{min} + \tau_m(D_{max} - D_{min}) \tag{7.2}$$

The fraction τ_m is determined by the cumulative histogram. The Y axis is divided in M cross sections, with a distance λ from each other, where $\lambda = max(H)/M$. Each cross section represents a depth plane m. The actual depth fraction τ_m for each cross section σ_m , i.e. a depth plane, is calculated by first determining the depth value $x_{\sigma m}$ where $H(x_{\sigma m}) \leq \sigma_m$ and $H(x_{\sigma m}+1) > \sigma_m$. This is demonstrated in Figure 7.3. Because the depth values x in the cumulative histogram are discrete, finding a value $x_{\sigma m}$ where $H(x_{\sigma m}) = \sigma_m$ is unlikely, and not desirable when generating planes that are dense, i.e. closer together, than the depth values provided in the cumulative histogram.

Once $x_{\sigma m}$ is determined, τ_m is calculated as follows:

$$\xi = \frac{m\lambda - H(x_{\sigma m})}{H(x_{\sigma m}+1) - H(x_{\sigma m})}$$

$$\tau_m = \xi(x_{\sigma m}+1) + (1-\xi)(x_{\sigma m}) \tag{7.3}$$

Figure 7.2(b) shows the transformation from a uniform depth plane distribution to a non-uniform distribution based on the cumulative histogram. In point (1), where the cumulative

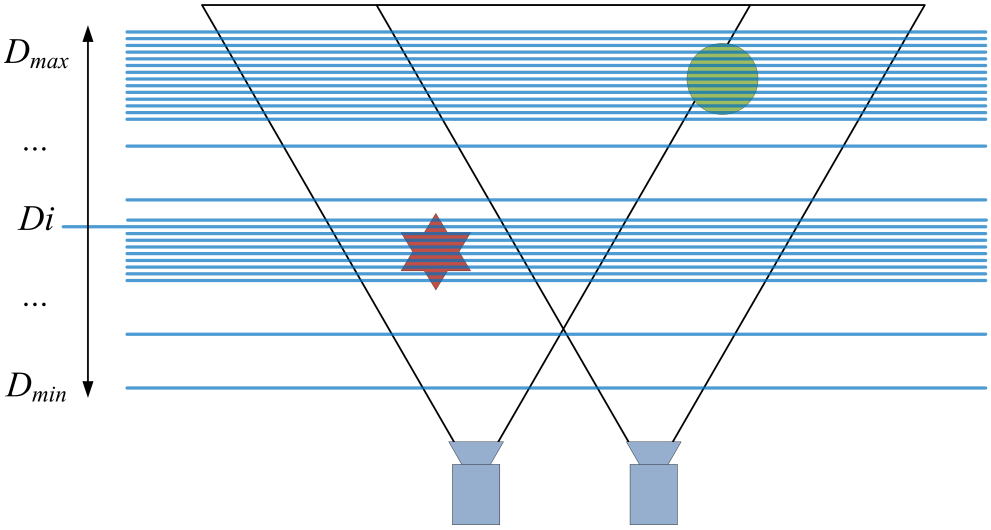


Figure 7.4: Redistributed depth planes.

histogram is steep, there will be a dense plane distribution, as can be seen at (1*). When the cumulative histogram is flat, a sparse plane distribution is acquired, as can be seen at (2*).

Using τ_m, an actual depth for every plane m ($0 \leq m < M$) is determined and used in the plane sweeping step:

$$D_m = D_{min} + \tau_m(D_{max} - D_{min}) \tag{7.4}$$

This is depicted in Figure 7.4. Here, the planes are redistributed using the cumulative histogram of Figure 7.2(b). As can be seen, more planes are available for determining the depth of the objects, and less planes are available in empty space. It is desirable to include some planes in the empty spaces between objects to allow the appearance of objects in dynamic scenes. To allow this, we increase all of the values in the histogram by a fixed number, based on the number of pixels. This way, the cumulative histogram will be less flat in less interesting regions, allowing some planes here.

7.2 Results

We tested the proposed method on different scenes and compared image quality and required planes. Both scenes under controlled conditions and outdoor scenes are considered. We will present in this section the result of the adaptive plane sweep approach. In chapter 8, these results are used in the complete system, and the results of the whole system is presented.

The first experiment shows the quality increase when a low number of planes is available. To increase the overall quality in both methods, foreground and background segmentation is

used. Figure 7.5 shows the result for a uniform depth plane distribution. Artifacts caused by the sparse plane distribution can be clearly seen; the depth map shows clear outliers. The depth map when using a non-uniform plane distribution, based on the histogram of the first depth map, can be seen in Figure 7.6. Less noise and outliers in the depth values can be perceived. Furthermore, the silhouette is more distinct and the features of the persons are clearer. Using the non-uniform plane distribution increases the quality of the depth map using a low number of planes, therefore increasing overall performance.

Figure 7.7 shows the result for a high number of planes. Here, some noise and unclear edges can be perceived. These artifacts are effectively filtered out using the non-uniform plane distribution. The depth planes generating vague edges and noise are not used and cannot contribute to the depth map, and therefore to the noise and artifacts.

To demonstrate the effect of the cumulative histograms, Figure 7.8 and 7.9 show an input image of a video sequence (a), the corresponding cumulative histogram of the depth map of the preceding frame (b) and the corresponding fraction τ from equation 7.3 (c). When only one dominant depth can be perceived, such as in Figure 7.8, one steep section in the cumulative histogram is visible. This part will be transformed to a flat value of τ, thus increasing the density of the planes in the corresponding region in the sweeping space. Flat sections of the cumulative histogram will correspond to steep values in the graph of τ, resulting in a sparse plane distribution.

When multiple dominant depths are available in the scene, the cumulative histogram will show multiple steep sections (see Figure 7.9). This will result in multiple dense regions in the plane distribution, as reflected by the values of τ in Figure 7.9(c).

The second experiment shows the results for the interpolation for soccer games. The results can be seen in Figure 7.12. The quality is increased compared to the uniform plane distribution, as seen in Figure 7.11. Details of the quality difference can be seen in Figure 7.10. In the uniform plane distribution in 7.10(a), missing heads and limbs can be perceived, caused by the low number of planes used to determine the interpolated view of the players. By redistributing the depth planes to the position of the players, as can be seen in Figure 7.12 and Figure 7.10(b), artifacts are seriously reduced. The non-uniform plane distribution method is especially applicable to soccer scenes due to the sparse location of players on the field and the multiple open spaces in the scene. Redistributing the depth planes will thus increase performance by reducing the number of wasted planes. The quality is not reduced by the movement of the scene due to the inclusion of depth planes in empty space. By including a few depth planes in empty space, players moving in these spaces are detected and the plane distribution is adapted accordingly.

To demonstrate the high quality of our results, we increased the number of depth planes to 5000. The quality of the result is high, as shown in Figure 7.13, but real-time processing is no longer possible due to the high computational requirements. Comparing Figure 7.12 and Figure 7.13 visually, we see little difference, proving the effectiveness of our method. We can compare the difference numerically to demonstrate the effect. Applying the PSNR metric

(peak signal-to-noise ratio) and the RMSE metric (root-mean-square error) yields the values in Table 7.1. As can be seen, both metrics show a lower difference between the reference image using 5000 depth planes and the adapted plane distribution using 40 planes, compared to the case without plane redistribution. This demonstrates that the adapted plane distribution yields better results with the same number of planes than the conventional distribution.

Metric	Non-adaptive	Adaptive
PSNR (higher is better)	38.64	46.63
RMSE (lower is better)	766.094	305.252

Table 7.1: Error metrics for the difference between 5000 and 40 planes, and between 5000 and 40 planes with adaptive plane distribution.

7.3 Conclusions

This chapter presents a method to reduce the computational requirements of plane sweeping by reducing the required number of frames by redistributing the planes to the places with a high object density. We tested the method on different kind of scenes, including a scene under controlled conditions and a scene of an outdoor soccer game. The results, both qualitatively and visual, show that the method yields the same or similar results compared to using many evenly distributed planes. This demonstrates the effectiveness of our method to increase performance.

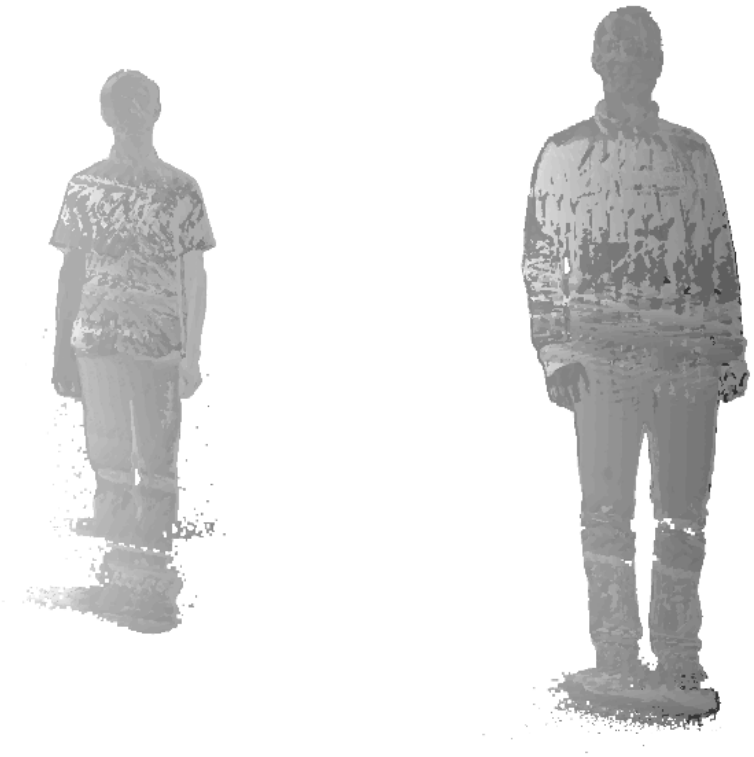


Figure 7.5: Depth map with a uniform depth plane distribution. A low number of planes (50) is used.

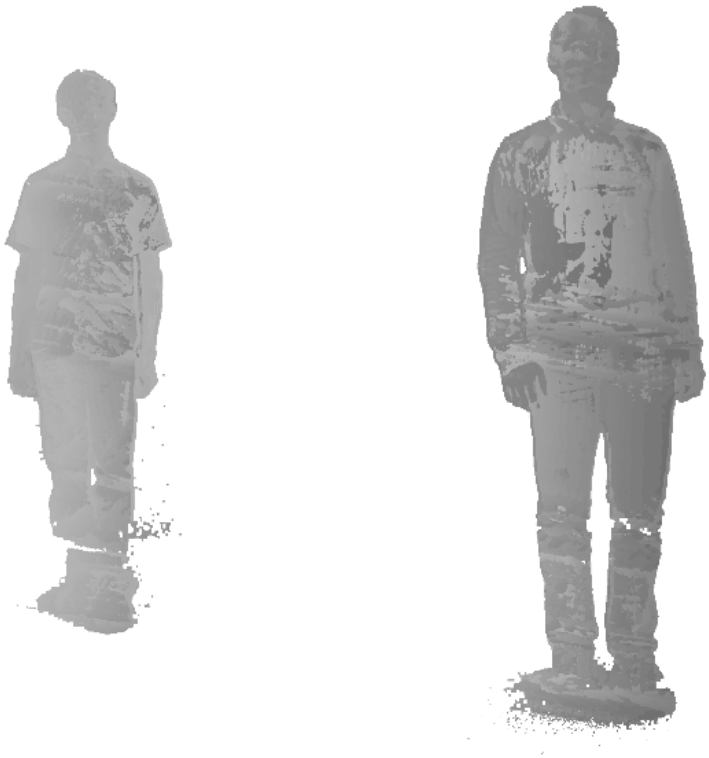


Figure 7.6: Depth map with a non-uniform depth plane distribution. A low number of planes (50) is used.

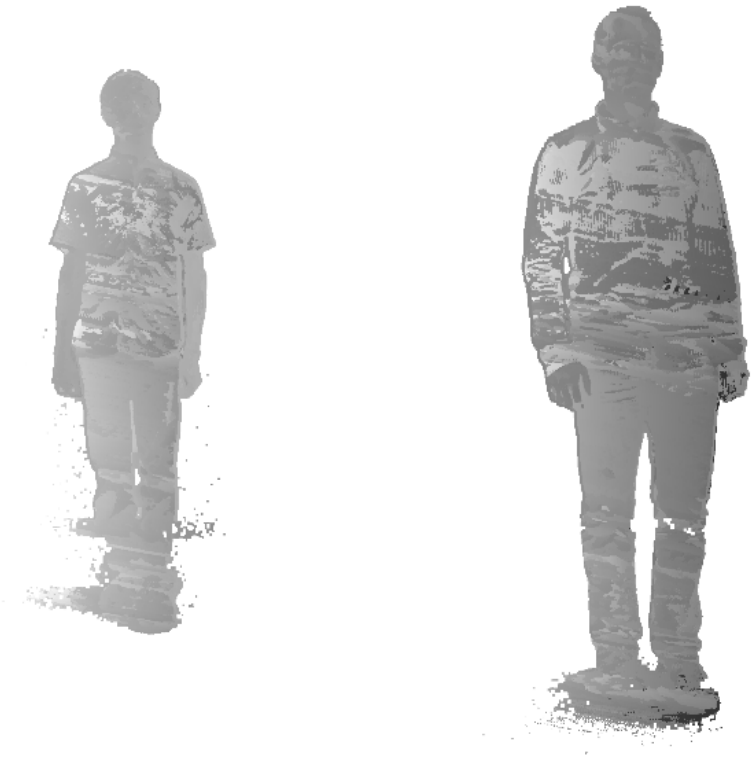


Figure 7.7: Depth map with a uniform depth plane distribution. A high number of planes (256) is used.

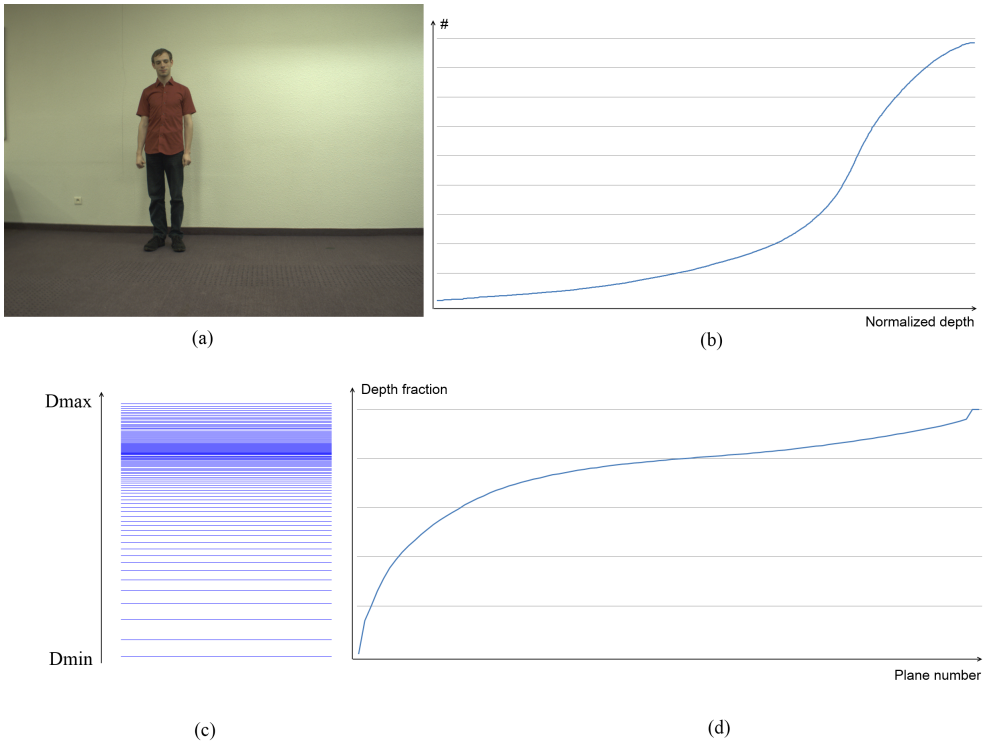


Figure 7.8: (a) Input image with one person. (b) Cumulative histogram of the depth map. (c) New depth plane distribution. (d) Corresponding fraction τ for a given plane number.

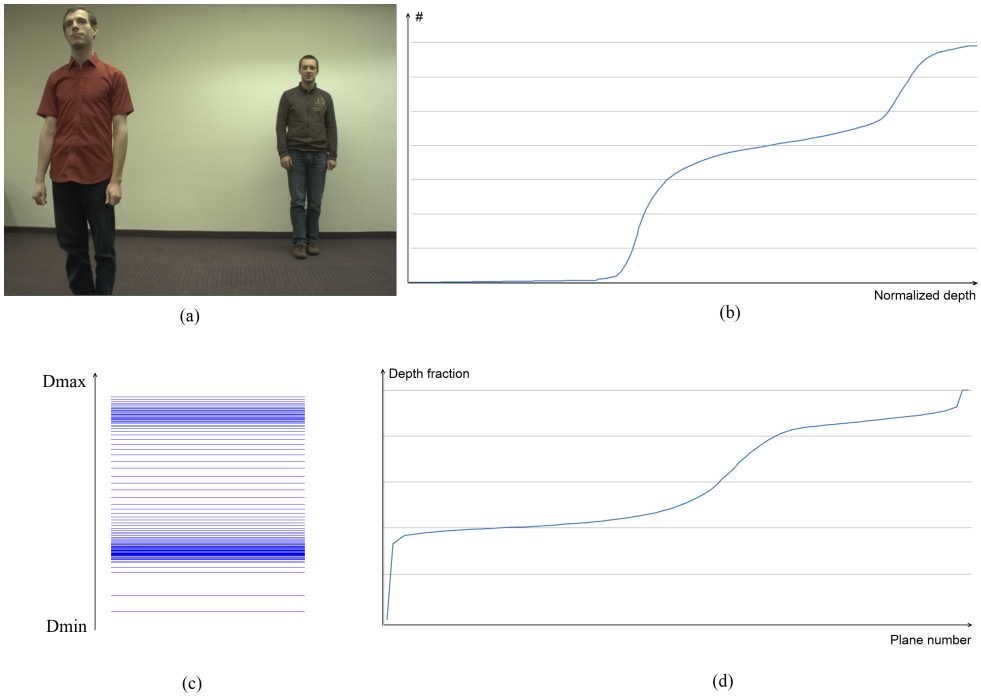


Figure 7.9: (a) Input image with two persons on different depths. (b) Cumulative histogram of the depth map. (c) New depth plane distribution. (d) Corresponding fraction τ for a given plane number.



(a) Uniform distribution (b) Adaptive distribution

Figure 7.10: Details of the quality differences between (a) Figure 7.11 and (b) Figure 7.12 (our method).



Figure 7.11: Plane sweeping of a soccer scene with a low number of depth planes (40) and a uniform plane distribution. Many artifacts and missing people can be perceived.



Figure 7.12: Plane sweeping of a soccer scene with a low number of depth planes (40) and an adaptive plane distribution. The quality is greatly increased in comparison with Figure 7.11.



Figure 7.13: Plane sweeping of a soccer scene with a high number of depth planes (5000) and a uniform plane distribution. The quality is comparable with Figure 7.12, which proves the effectiveness of the method.

Chapter 8

Results

8.1 Final Result of the Method **123**
8.1.1 Comparison with Traditional Stereo Methods **127**
8.1.2 Location of the Virtual Camera **127**
8.2 Comparison with Existing Systems **147**
8.3 Performance of the Components **159**

In this chapter, we will present the results for our systems. First, we will present the final product of our method. Second, we will compare them to other systems available. Third, we will discuss the performance of our system. We give an overview of the results of our method in Table 8.1, including the already discussed results.

8.1 Final Result of the Method

Results of the complete method are discussed in this section. The effects of different components of the system are not shown, as they have already been discussed in their corresponding sections in previous chapters. For all results, we opted for the most difficult position, i.e. the virtual position in the middle of 2 real cameras. This will demonstrate the applicability of our method. The same conclusions can be drawn from the other camera positions.

Figure 8.2 and Figure 8.3 show the result for the Barcelona dataset, where a curved arrangement is used, with 10 meters between each camera. We placed a virtual camera in the middle of 2 real cameras, which is considered the most difficult location. The position is shown in Figure 8.1(c), and the input images in Figure 8.1(a) and 8.1(a). We used 512 depth planes, and 2 cameras for the color consistency check.

Figure 8.2 shows the result and depth map of the histogram-based method. The most notable artifacts can be found in the background of the scene, due to the assumption that the background is flat. This is, however, not the case for the building behind the pitch, hence the artifacts.

The foreground interpolation shows some artifacts for the goal area. Here, the complete goal is white, resulting in mismatching color values. There is, however, no projective distortion.

The color result and the depth values of the players are good. There are no missing or extra limbs or players, there are no projective distortions, et cetera. This demonstrates the effect of our method, compared to, for example, Figure 6.8. The method generated a pleasant-looking, realistic, and geometrically correct result.

Figure 8.3 shows the result and depth map of the median-based method. Compared to the histogram-based approach, some artifacts can be perceived (best visible as the black noise in the depth map). The median-based approach is strongly dependent on the threshold Φ_m, which determines the allowed depth value range per player. If this value is too large, no filtering will occur. If the filtering is too small, players (and other objects) can disappear. This is shown in Figure 8.4, where Φ_m is too small. The goal and players disappear, resulting in unpleasant effects.

Figure 8.6 and Figure 8.7 show the result for the Genk dataset, where a linear arrangement is used, with 1 meter between each camera. Here, a wide field of view is used. We placed a virtual camera in the middle of 2 real cameras, shown in Figure 8.5(c). The most left and right input images are shown in Figure 8.5(a) and 8.5(a). We used 512 depth planes, and 8 cameras as input for the color consistency check.

Table 8.1: Overview of the results of our method. The results of the previous chapters and the results of this chapter are included.

Figure	Dataset	Demonstrated Concept
6.2	Genk, 12.5 mm	Comparison between results with and without shadows. Shadows are required for a realistic result.
6.4	Barcelona	Comparison between results with the goal as foreground or as background. The goal as foreground avoids projective distortion.
6.6	Genk, 25 mm	Conventional plane sweeping. Artifacts can be observed.
6.7	Genk, 25 mm	Conventional plane sweeping with segmentation. Artifacts can be observed.
6.8	Barcelona	Conventional plane sweeping with segmentation. Artifacts can be observed.
6.9	Barcelona	Comparison between using 2 or all cameras for the curved arrangement.
6.10	Barcelona	Comparison of using the segmentation as guideline. Many artifacts are removed.
6.15	Barcelona	Comparison of the median-based and histogram-based method. The histogram-based method does not make players disappear.
6.16	Barcelona	Comparison of using the background depth check. Many artifacts are removed.
6.17	Genk, 25 mm	Details of the depth filtering.
7.10	Genk, 12.5 mm	Detailed comparison of uniform and adaptive plane distribution.
7.11	Genk, 12.5 mm	Histogram-based interpolation method with uniform plane distribution (40 planes). Artifacts can be observed.
7.12	Genk, 12.5 mm	Histogram-based interpolation method with adaptive plane distribution (40 planes). The result is better than the uniform distribution with the same number of planes.
7.13	Genk, 12.5 mm	Histogram-based interpolation method with uniform plane distribution (5000 planes).

Continued on next page

Table 8.1 – continued from previous page

Figure	Dataset	Demonstrated Concept
8.2	Barcelona	Histogram-based interpolation method.
8.3	Barcelona	Median-based interpolation method.
8.4	Barcelona	Median-based interpolation method with wrong parameters. Objects are cut off.
8.6	Genk, 12.5 mm	Histogram-based interpolation method.
8.7	Genk, 12.5 mm	Median-based interpolation method.
8.8	Genk, 12.5 mm	Madian-based interpolation method with wrong parameters. Players have disappeared.
8.10	Genk, 25 mm	Histogram-based interpolation method.
8.11	Genk, 25 mm	Median-based interpolation method.
8.12	Barcelona	Stereo pair generation, represented as anaglyph.
8.13	Barcelona	Scene where the method fails. The scene is complex and not all artifacts are eliminated.
8.15	Genk, 12.5 mm	Stereo matching using the method of Zhang et al. [2009b]. No valid reconstruction is acquired.
8.16	Genk, 12.5 mm	Stereo matching using the MPEG reference software. No valid reconstruction is acquired.
8.17	Barcelona	Histogram-based interpolation method with the virtual camera in a non-optimal location. There are artifacts at the image borders due to missing input information.
8.18	Barcelona	Histogram-based interpolation method with the virtual camera in a non-optimal location. There are artifacts at the image borders due to missing input information.

Figure 8.6 shows the result and the depth map for the histogram-based filtering method. No artifacts can be perceived in the players. There are some errors in the background, including the advertising boards and the top of the bleachers.

Figure 8.7 demonstrates the median-based depth filtering. There are slight errors on the colors of the players, but they are barely noticeable. There is, however, still the issue of choosing the right value for Φ_m. When Φ_m is too small, players disappear, as shown in Figure 8.8.

Figure 8.10 and Figure 8.11 show the result for the Genk dataset with a smaller field of view. The camera positions are the same as the previous results, with the most left and right images shown in Figure 8.9.

Figure 8.10 shows the histogram-based depth filtering method. Because the field of view is smaller, players are larger in the image, and a more detailed depth map can be observed. The depth can be distinguished per player, which demonstrates that the depth information per player is important. This is an often ignored aspect by other methods.

Figure 8.11 shows the median-based depth filtering method. Some detailed artifacts, such as noise, can be observed on the players. This is also visible in the depth map. The allowed depth range is a bit larger, resulting in more opportunity for mismatching in the depth-selective plane sweep.

Figure 8.17 shows the application to generate 3D images using virtual camera rendering. A virtual camera is placed next to a real camera, and the image is rendered. Figure 8.17 demonstrates this using an anaglyphic image [Beiser, 1981] to allow a printable result, but any 3D format is applicable for use in, for example, 3D displays.

These results demonstrate that our method yields good results for the used camera setups. The number of artifacts in the foreground is low, compared to the more traditional image-based methods. The results are pleasant-looking, realistic, and give an impression of a real camera image.

There are, however, a few cases where the method fails. This is shown in Figure 8.13. Here, a very complex scene can be perceived. Almost all of the players are focused in a single spot on the field, and are considered a single group of players in our method. First, the result reveals that many peaks will be visible in the histogram of depth values, but the noise will also be considered to be valid. There is just too much noise. Second, there is a large amount of valid depth values after the depth filtering. This will stimulate mismatches that cannot be filtered out by the depth filtering. Third, segmentation checks in the other cameras will fail, as the artifacts will be considered foreground in all the cameras. There are so many foreground objects, such that ghost objects are actually projected on other, valid, foreground objects. This class of scenes should be considered in future depth filtering approaches to allow high quality view interpolation in all possible cases.

8.1.1 Comparison with Traditional Stereo Methods

To compare our method with traditional stereo methods, we rectified 2 images from nearby cameras (as shown in Figure 8.14), and used these for 2 stereo methods.

First, we applied the real-time method of Zhang et al. [2009b], which uses a local matching cost with adaptive aggregation windows. Zhang et al. [2009b] report high quality results for standard datasets. For the soccer scenes, however, no valid disparity map is acquired, as shown in Figure 8.15(a). This is caused by the ambiguity in the matching process, where the green colors of the background can be matched to any place on the background. If we use these disparity maps to warp the input images, many artifacts can be observed, such as ghost players, misplaced players, and a distorted background.

Second, we applied the currently developed MPEG reference software for depth estimation (DERS) [Stankiewicz et al., 2013] and view synthesis (VSRS) [Wegner et al., 2013]. DERS uses stereo matching based on aggregation blocks and a global refinement step based on graph cuts using the approach of Boykov and Kolmogorov [2004]. The input color images are then warped and blended, based on the previously calculated depth maps [Tanimoto et al., 2009]. Holes are filled using the closest pixel values where data is available, or texture patches are used when the holes are relatively big [Koppel et al., 2012; Ndjiki-Nya et al., 2011]. In the depth map for the soccer scene (Figure 8.16(a)), the players are distinguishable, but not very correct. Furthermore, the background depth is refined too much due to the uniform green background. When applying the view synthesis step, artifacts can be observed due to these depth map errors, as shown in Figure 8.16(b). Ghost players can be observed, as well as ghosting artifacts on the background. Furthermore, the processing required 1 hour and 23 minutes, making the method unusable for real-time performance.

8.1.2 Location of the Virtual Camera

The location of the virtual camera has a large effect on the resulting quality.

If the camera is placed too much to the front (Figure 8.17(a)), the plane sweeping phase will use color values of the input camera images for multiple virtual pixels, resulting in the increased effect of errors. Furthermore, the reduced input resolution is visible in the result.

Figure 8.17 shows the result if the camera is placed too close to the front or too far to the back. If the camera is placed too much to the back (Figure 8.17(b)), not enough background information is available to fill up the virtual image, resulting in empty places in the result. Cropping the result will only have a similar effect as moving the virtual camera forward.

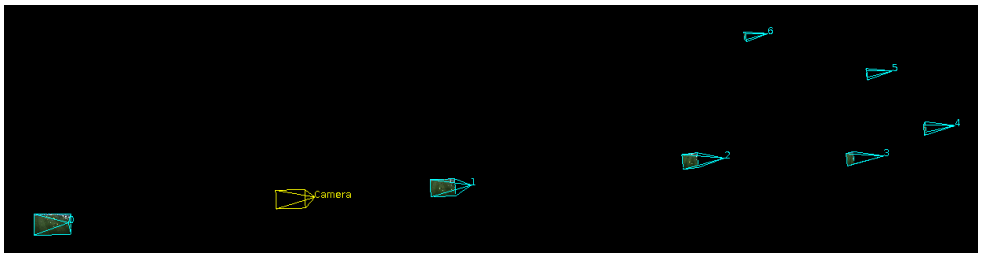
If the virtual camera is positioned outside the region between the cameras, similar effects occur. Figure 8.18 shows the effect when the virtual camera is placed on the left of the leftmost real camera. A large portion of the background is missing. Furthermore, no players can be interpolated in that region, as this information is not available, resulting in a half result.



(a) Left image



(b) Right image



(c) Virtual and real camera positions

Figure 8.1: Input for the Barcelona dataset. (a) and (b) show the left and right images for the 2 cameras closest to the virtual camera position. These 2 cameras are used to determine the color values in the virtual image. (c) shows the camera positions of the real cameras (in blue) and the virtual camera (in yellow).



(a) Intermediate position



(b) Intermediate position, depth map

Figure 8.2: Result and depth map for the Barcelona dataset, histogram-based depth filtering. The color result and the depth values of the players are good. Some artifacts can be observed in the background and the goal area.

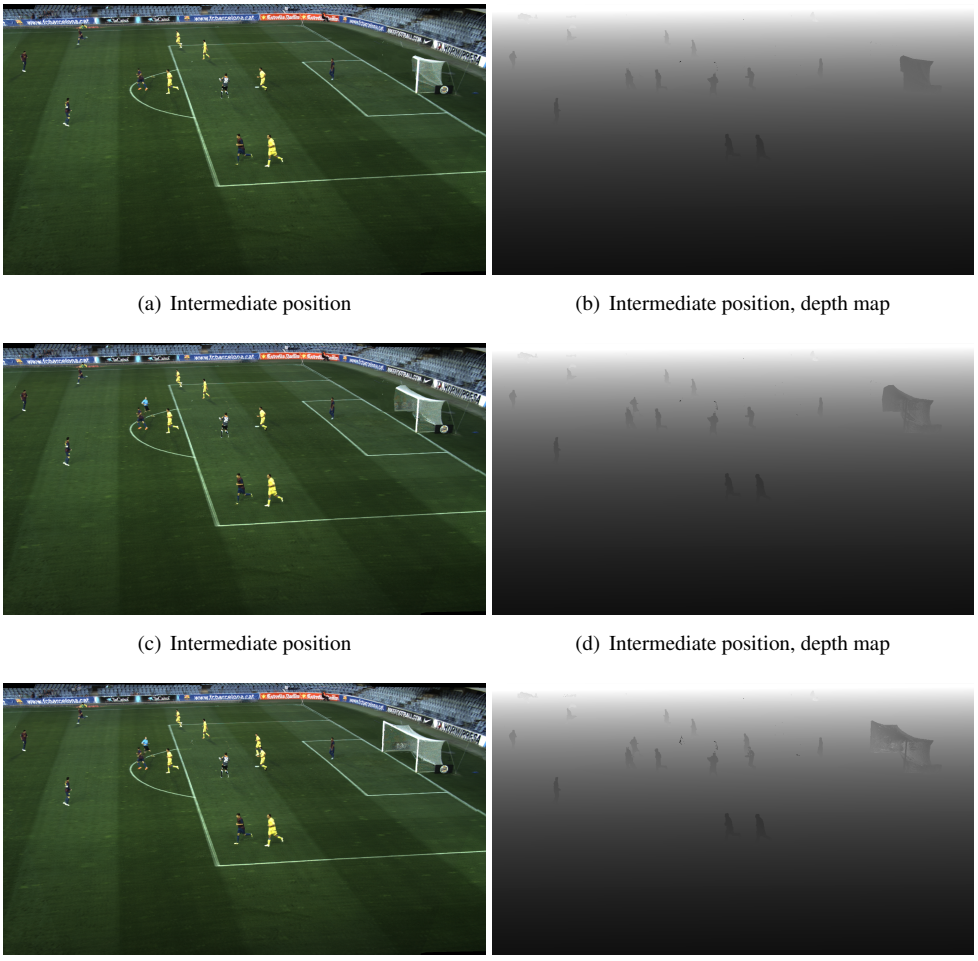


(a) Intermediate position



(b) Intermediate position, depth map

Figure 8.3: Result and depth map for the Barcelona dataset, median-based depth filtering. The color result and the depth values of the players are good, but there are some artifacts not present in the histogram-based method. Some artifacts can be observed in the background and the goal area.



(a) Intermediate position (b) Intermediate position, depth map

(c) Intermediate position (d) Intermediate position, depth map

(e) Intermediate position (f) Intermediate position, depth map

Figure 8.4: Result and depth map for the Barcelona dataset, median-based depth filtering. The parameter controlling the allowed depth range around the median depth value per pixel group is varied. Wrongly-chosen parameters result in serious artifacts, such as disappearing players.



(a) Left image



(b) Right image

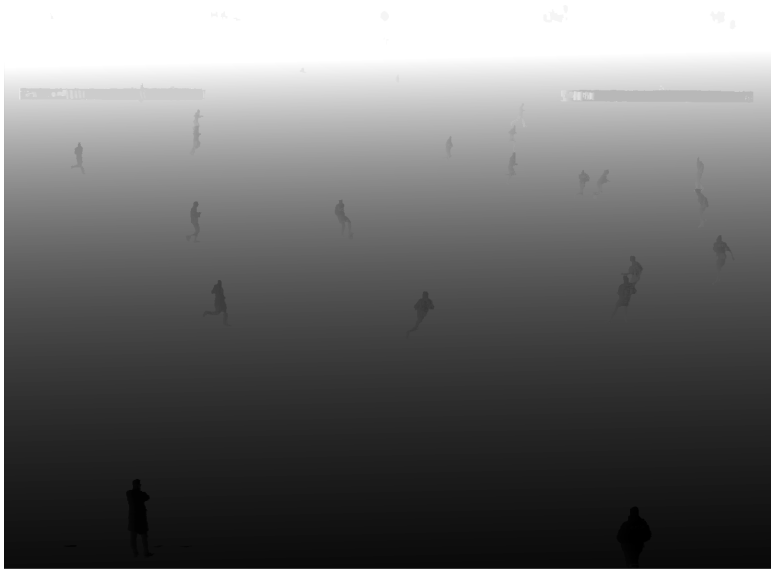


(c) Virtual and real camera positions

Figure 8.5: Input for the Genk dataset with a large field of view. (a) and (b) show the left and right images for the 2 cameras closest to the virtual camera position. These 2 cameras are used to determine the color values in the virtual image. (c) shows the camera positions of the real cameras (in blue) and the virtual camera (in yellow).



(a) Intermediate position



(b) Intermediate position, depth map

Figure 8.6: Result for the Genk dataset, histogram-based depth filtering. No artifacts can be perceived in the players. There are some errors in the background, including the advertising boards and the top of the bleachers.



(a) Intermediate position

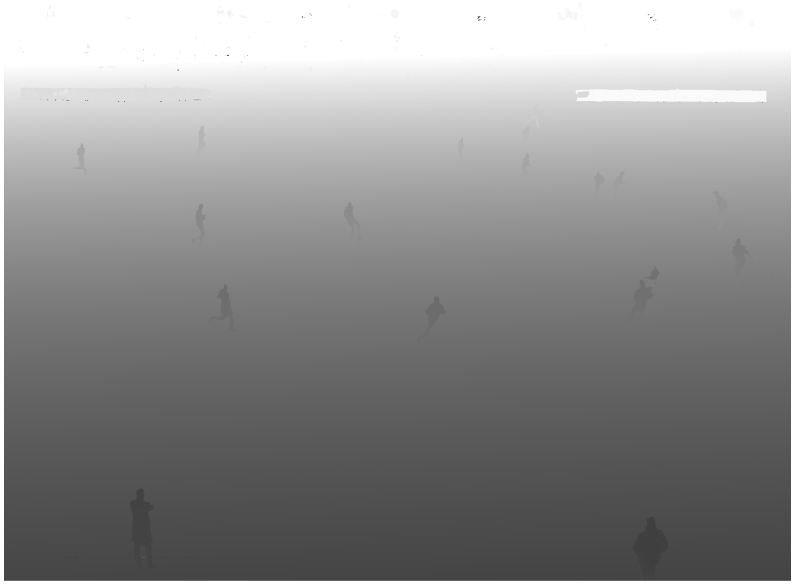


(b) Intermediate position, depth map

Figure 8.7: Result for the Genk dataset, median-based depth filtering. There are slight errors on the colors of the players, but they are barely noticeable.

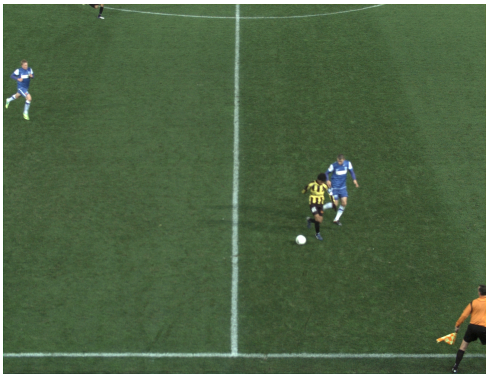


(a) Intermediate position



(b) Intermediate position, depth map

Figure 8.8: Result for the Genk dataset, median-based depth filtering, using a wrong value for Φ_m. Disappearing players can be perceived.



(a) Left image



(b) Right image



(c) Virtual and real camera positions

Figure 8.9: Input for the Genk dataset with a small field of view. (a) and (b) show the left and right images for the 2 cameras closest to the virtual camera position. These 2 cameras are used to determine the color values in the virtual image. (c) shows the camera positions of the real cameras (in blue) and the virtual camera (in yellow).



(a) Intermediate position



(b) Intermediate position, depth map

Figure 8.10: Result for the Genk dataset, histogram-based depth filtering. More pixels are available per player, resulting in a more detailed depth map. This demonstrates that a single depth per player is not sufficient.



(a) Intermediate position



(b) Intermediate position, depth map

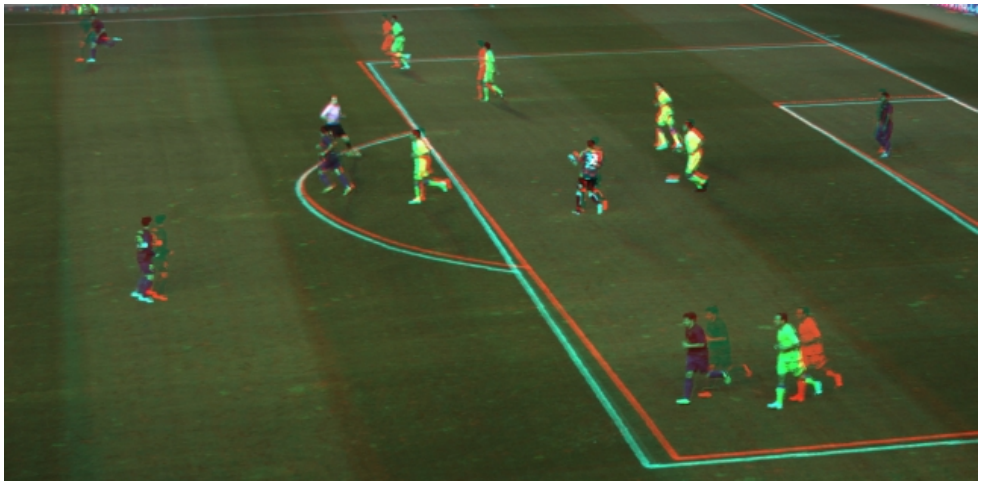
Figure 8.11: Result for the Genk dataset, median-based depth filtering. The result and depth map show more noise than the median-based approach.



(a) Intermediate position, left

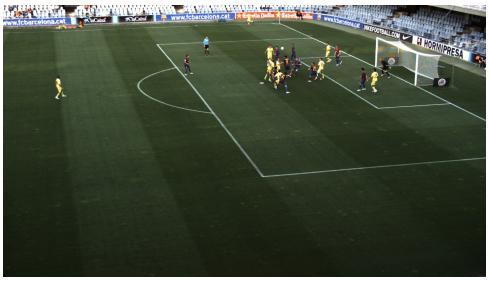


(b) Intermediate position, right



(c) Anaglyph image

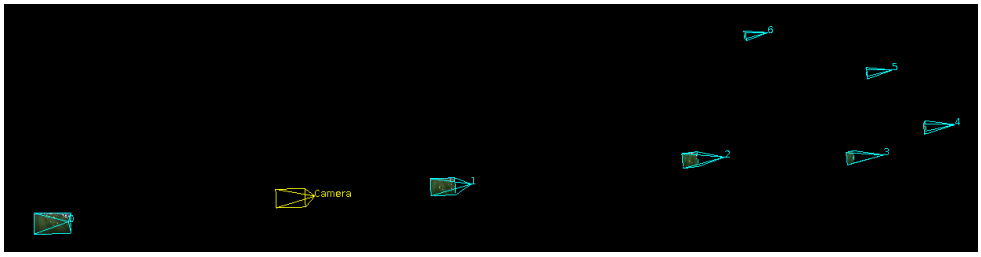
Figure 8.12: Anaglyph demonstrating the application to create 3D images using a virtual camera. Both the left image (a) and the right image (b) are rendered using our method. The anaglyph (c) is merely a 3D visualization, usable for paper. Other methods are also possible and more applicable for 3D displays.



(a) Left image



(b) Right image



(c) Virtual and real camera positions



(d) Intermediate position



(e) Intermediate position, depth map

Figure 8.13: Result for the Barcelona dataset, where artifacts cannot be removed. There are too many foreground objects and too many valid depth values to allow effective depth filtering. Mismatching is therefore still possible, resulting in artifacts.

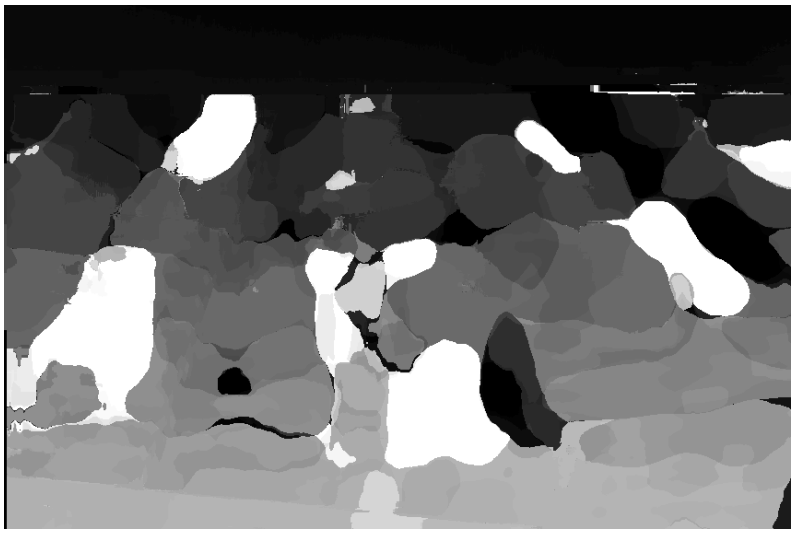


(a) Left rectified image



(b) Right rectified image

Figure 8.14: Result for stereo matching on soccer scenes. We rectified 2 successive cameras, placed 1 meter apart from each other, and applied the stereo matching method of Zhang et al. [2009b] and the MPEG stereo reference software [Stankiewicz et al., 2013; Wegner et al., 2013], as shown in Figure 8.15 and Figure 8.16.

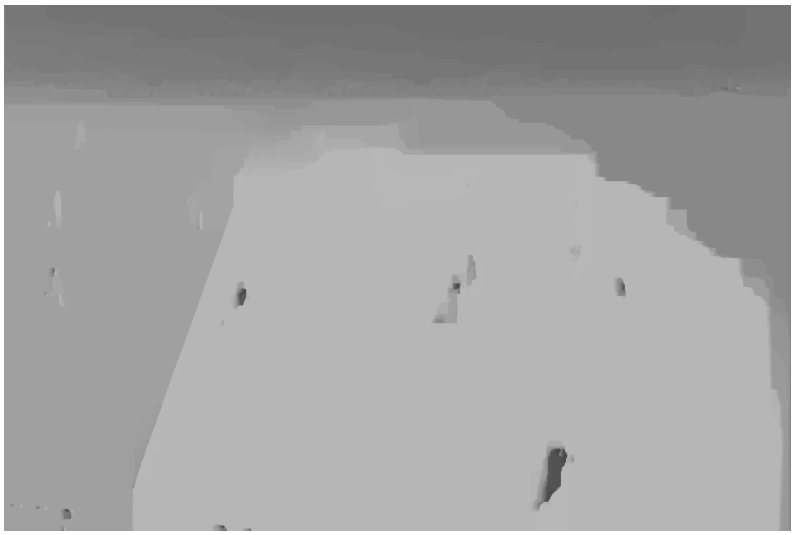


(a) Disparity map



(b) Recolored disparity map

Figure 8.15: Result for stereo matching on soccer scenes. We rectified 2 successive cameras, placed 1 meter apart from each other, and applied the stereo matching method of Zhang et al. [2009b]. As can be seen, the disparity map (a) is not valid, resulting in erroneous values, such as ghost players, when the image (b) is reconstructed based on the disparity map. This is caused by the large number of green values in the images, stimulating mismatching.



(a) Depth map

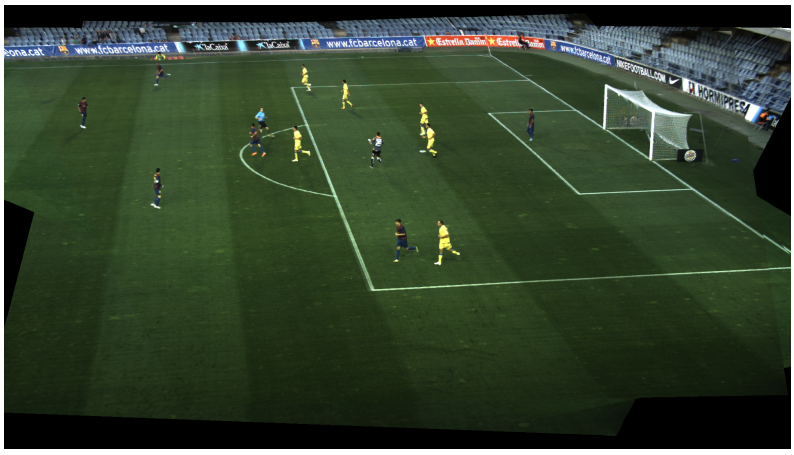


(b) Recolored depth map

Figure 8.16: Result for stereo matching on soccer scenes. We rectified 2 successive cameras, placed 1 meter apart from each other, and applied the MPEG stereo reference software [Stankiewicz et al., 2013; Wegner et al., 2013]. As can be seen, the depth map (a) has more distinct players, but is still erroneous, especially the background. This is also reflected in the warped result (b), where ghost players and ghost lines can be observed.



(a) Intermediate position, too near

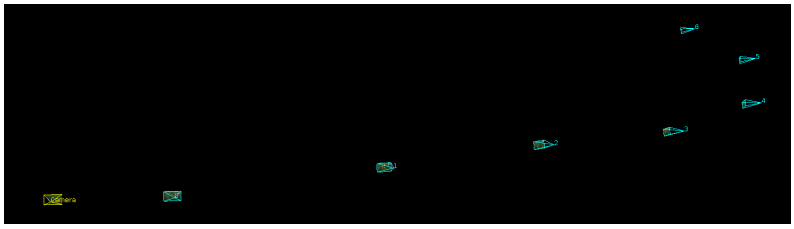


(b) Intermediate position, too far

Figure 8.17: Result for virtual camera positions outside the optimal region. (a) The camera is placed too much to the front, resulting in a reduced output resolution. (b) The camera is placed too much to the back, resulting in image parts without information. This information is not available and cannot be created.



(a) Intermediate position



(b) Virtual and real camera positions

Figure 8.18: Result for virtual camera positions outside the optimal region. The effect is the same as placing the camera too much backwards; there are image regions without color information, as these are not available in the input images.



(a) Intermediate position



(b) Intermediate position

Figure 8.19: The system of Liberovision [Liberovision, 2013b]. Ghost players and ghost limbs can be observed. The method blends the images from one camera to the other, resulting in these ghosting artifacts.

8.2 Comparison with Existing Systems

In this section, we will present a comparison with existing systems with public results. Because no datasets are provided, we will discuss and compare the provided visual results.

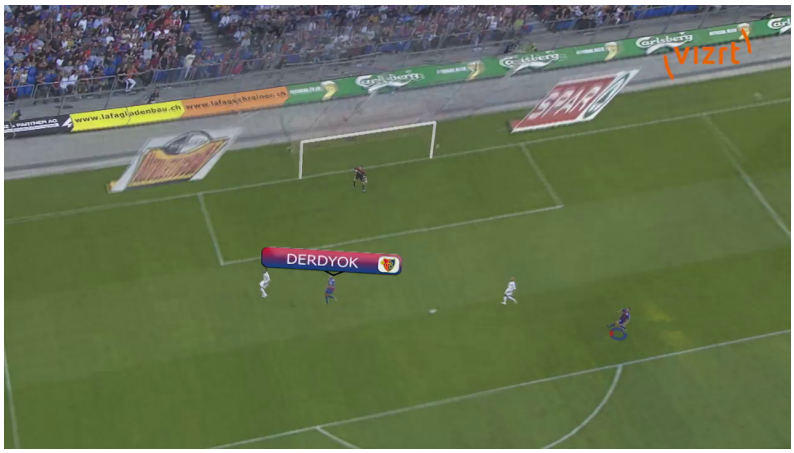
Liberovision [Liberovision, 2013a] demonstrated a semi-automatic system using billboards, based on existing camera positions. Some frames are shown in Figures 8.19, 8.20, and 8.21. The results show clearly that billboards are used. The players in Figure 8.19(a) and Figure 8.19(b) (at the right side of the image) are actually flat and the billboards are shown from the side. Blending is used to generate intermediate positions. This, however, results in extra limbs and blending artifacts, as shown in Figure 8.19(a) and 8.19(b). Furthermore, the background and the goal are modeled. Slanting of the goal can be seen in Figure 8.20(a). The system, however, can handle a larger distance between the cameras and is less restricted. If the method is used in fast moving camera transitions, artifacts can hardly be perceived due to the speed [Liberovision, 2013b].

The method of Germann et al. [2010][Germann et al., 2012] demonstrate 2 approaches, the conventional billboarding technique [Germann et al., 2012], and the articulated billboard technique [Germann et al., 2010]. The conventional technique (Figure 8.22) shows artifacts in the interpolation, seriously reducing the quality. These artifacts are not present in our method. The articulated technique (Figure 8.23) provides results that are close to the ground truth, demonstrating a good method for player interpolation. The method, however, is not fully automatic and takes up to 6 minutes per frame.

The system of Ohta et al. [2007] suffers from the same artifacts as other billboard techniques. The players look flat and no real background is used, as shown in Figures 8.24 and 8.25. Furthermore, the calibration is more strict and an overhead camera is required, making the system less flexible. The virtual viewpoints, however, are less restricted than in our system.

The method of Inamoto and Saito [2007b] describe a system that is also limited to the viewpoints between real cameras. The interpolation method, however, uses linear interpolation between color values. The results show a reasonable quality (Figure 8.26), but no depth is used in the final results, which may yield in perspective artifacts if the cameras are far apart. Furthermore, overlapping players in the input images pose a problem for the system, which we handle by the incorporation of depth values.

The Piero system [BBC, 2010] appears to use both a model-based system and a photorealistic system similar to Liberovision [Liberovision, 2013a]. The model-based approach is shown in Figures 8.27 and 8.30. The result is a clearly animated model of the scene and has no photorealistic effect. The photorealistic system shows the same artifacts as the liberovision system, as shown in Figure 8.28. Players are fading in and out, and ghosting artifacts occur on the players themselves. They demonstrated, however, results where these artifacts are not present, as shown in Figure 8.29. Here, no ghosting occurs, but the virtual camera movement is small.



(a) Intermediate position



(b) Intermediate position

Figure 8.20: The system of Liberovision [Liberovision, 2013b]. (a) The goal is slanted, and seems modeled. (b) Complex situations result in noise, just as our method.

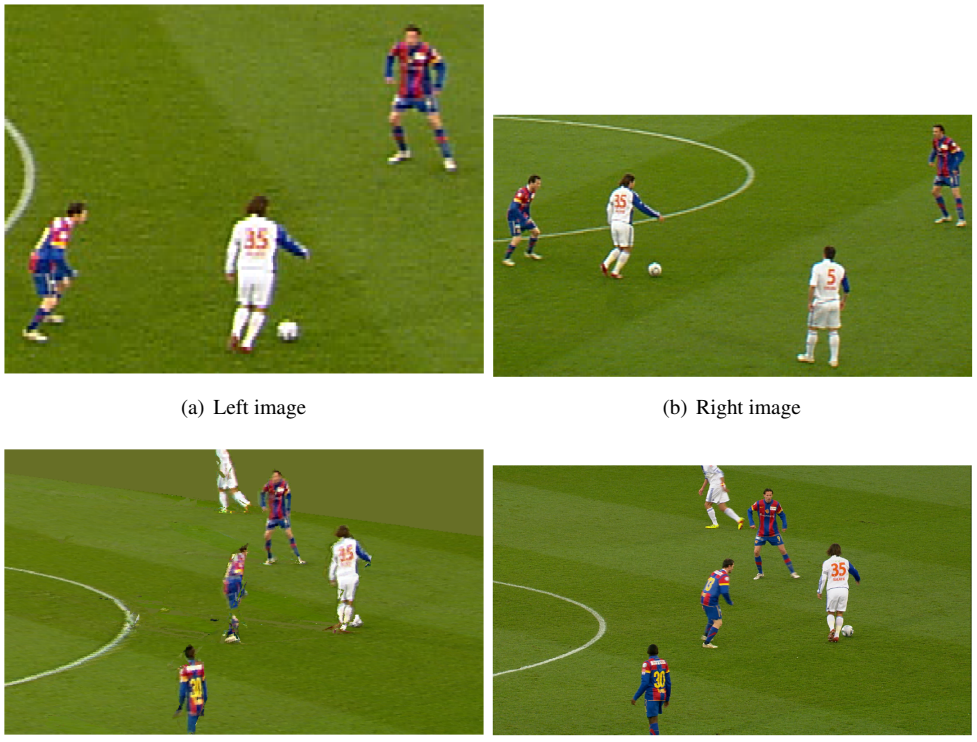


(a) Intermediate position



(b) Intermediate position

Figure 8.21: The system of Liberovision [Liberovision, 2013b]. The players are clearly modeled as planes.



(a) Left image

(b) Right image

(c) Intermediate position

(d) Ground truth

Figure 8.22: The system of Germann et al. [2010], using conventional billboards. (a) The left input image. (b) The right input image. (c) The interpolated image. The result is noisy, compared to the ground truth (d).



(a) Left image (b) Right image

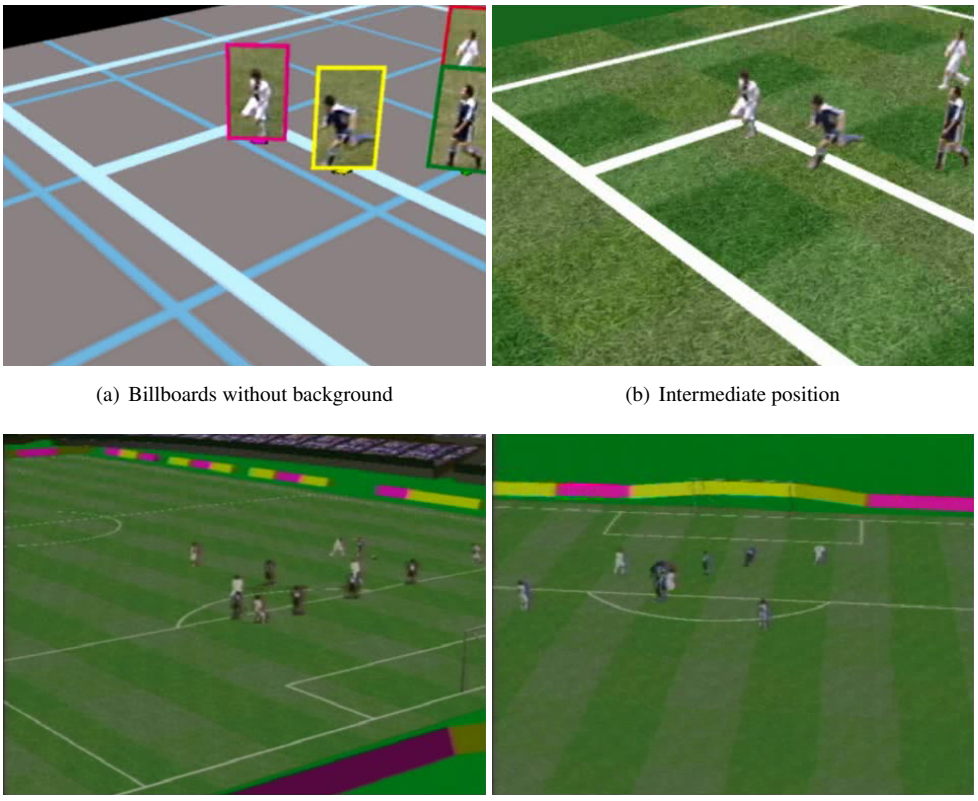


(c) Intermediate position



(d) Intermediate position (e) Ground truth

Figure 8.23: The system of Germann et al. [2010], using articulated billboards. (a) The left input image. (b) The right input image. (c) Comparison between the conventional and the articulated billboards. The results are good. (d) Results of a complete interpolation. The result is close to the ground truth (e).



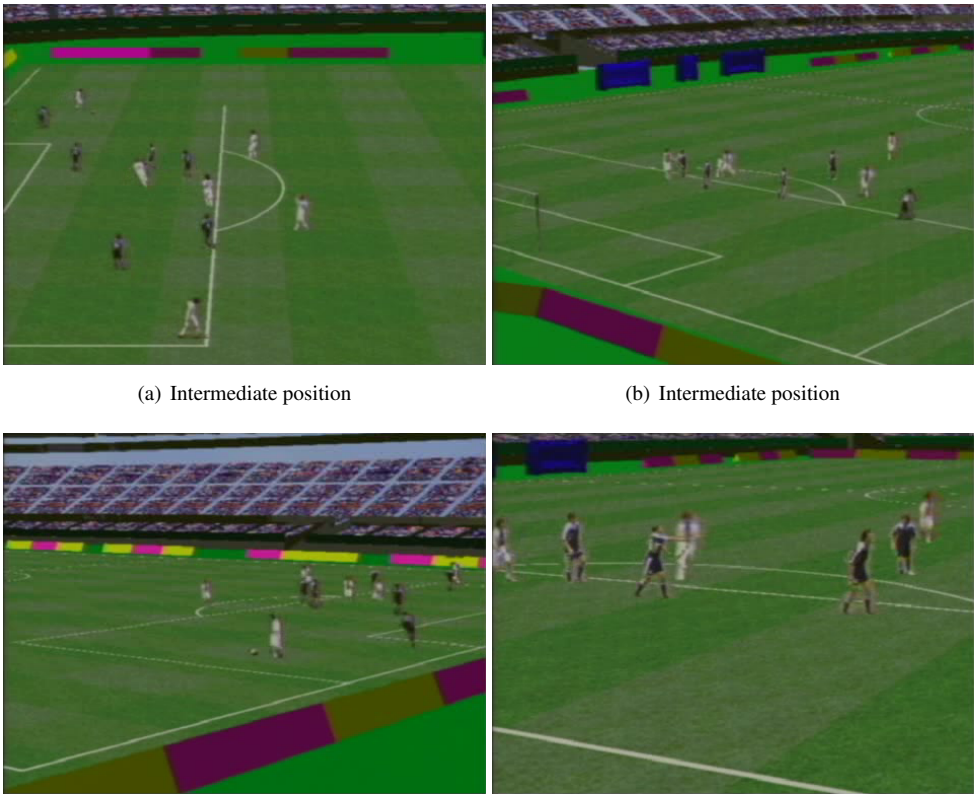
(a) Billboards without background

(b) Intermediate position

(c) Intermediate position

(d) Intermediate position

Figure 8.24: The system of Ohta et al. [2007]. The method uses billboards, placed in a model of the pitch. No perspective correction is applied, resulting in a flat looking result.



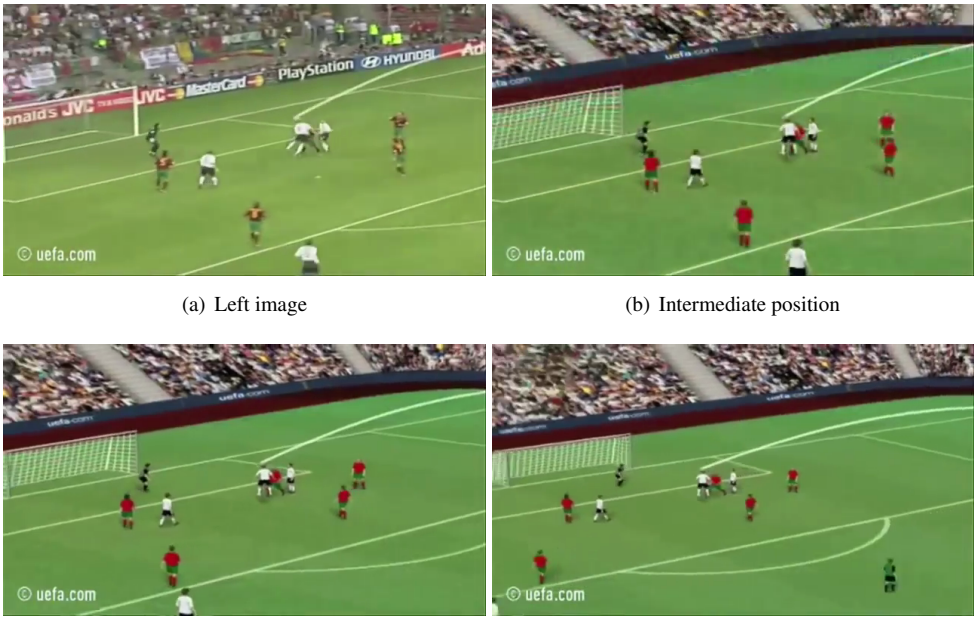
(a) Intermediate position (b) Intermediate position

(c) Intermediate position (d) Intermediate position

Figure 8.25: The system of Ohta et al. [2007]. The method uses billboards, placed in a model of the pitch. No perspective correction is applied, resulting in a flat looking result.



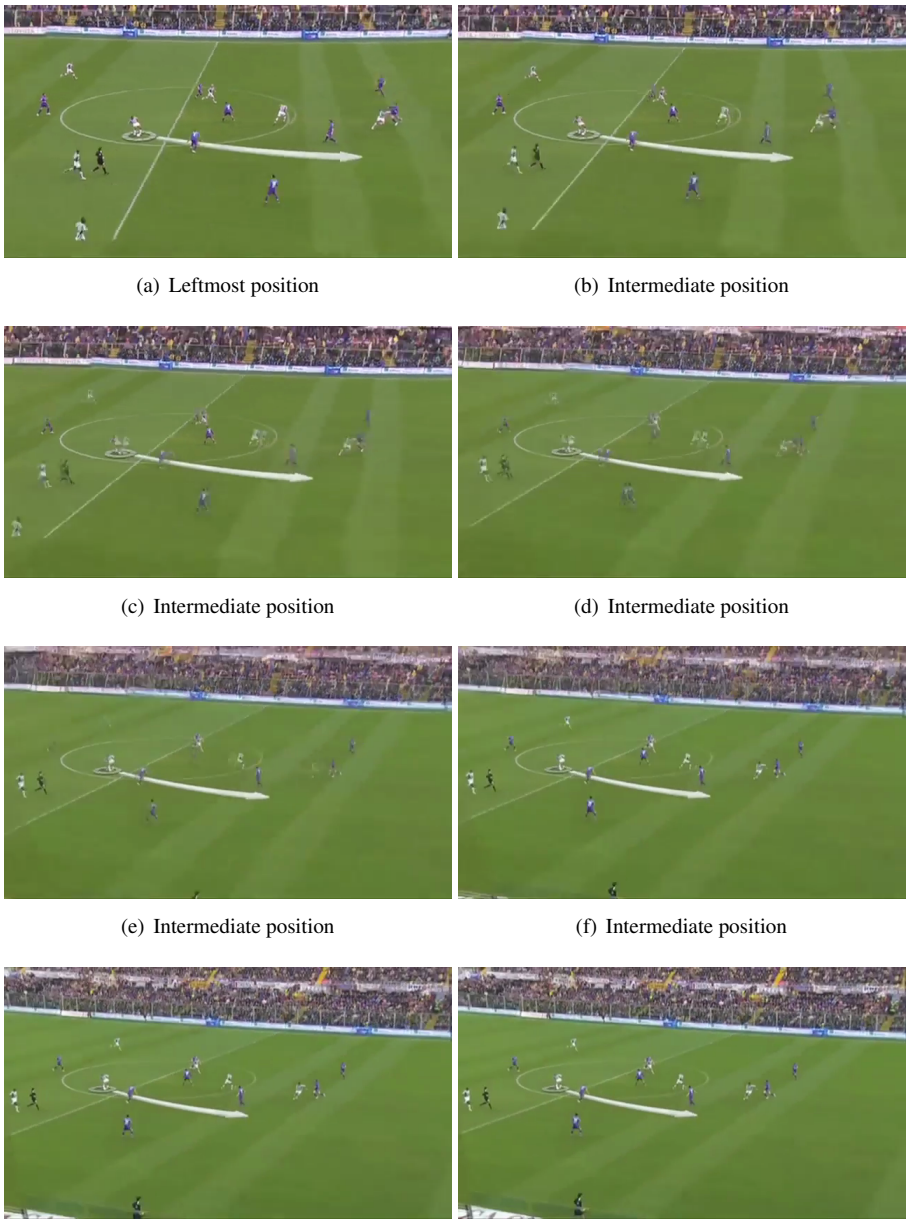
Figure 8.26: The system of Inamoto and Saito [2007b]. The interpolation method uses linear interpolation between color values. The results show a reasonable quality, but depth values are not considered, which may pose a problem for complex scenes.



(a) Left image (b) Intermediate position

(c) Intermediate position (d) Intermediate position

Figure 8.27: The Piero system [BBC, 2012]. The piero system, using computer models of the scene. The result has an animated look and is not photorealistic.



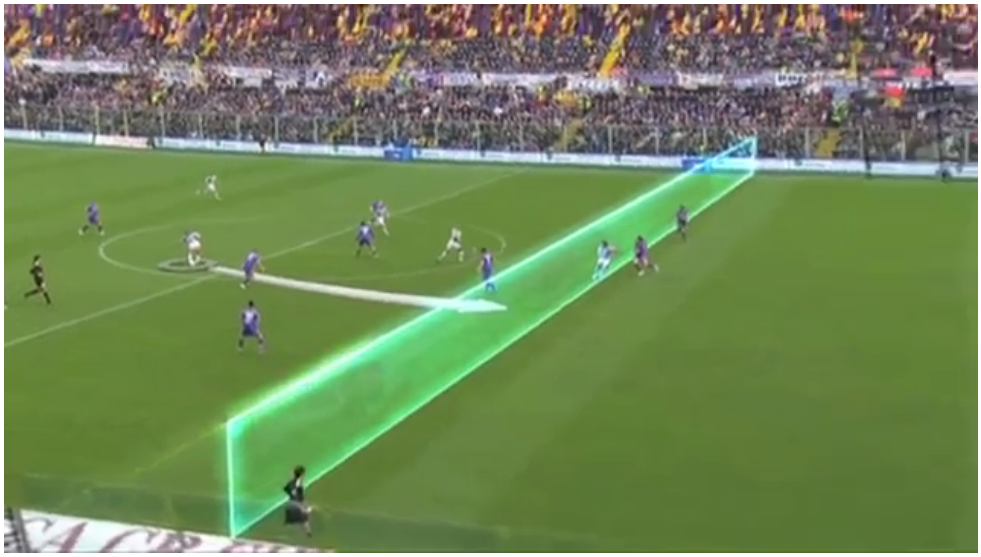
(a) Leftmost position (b) Intermediate position

(c) Intermediate position (d) Intermediate position

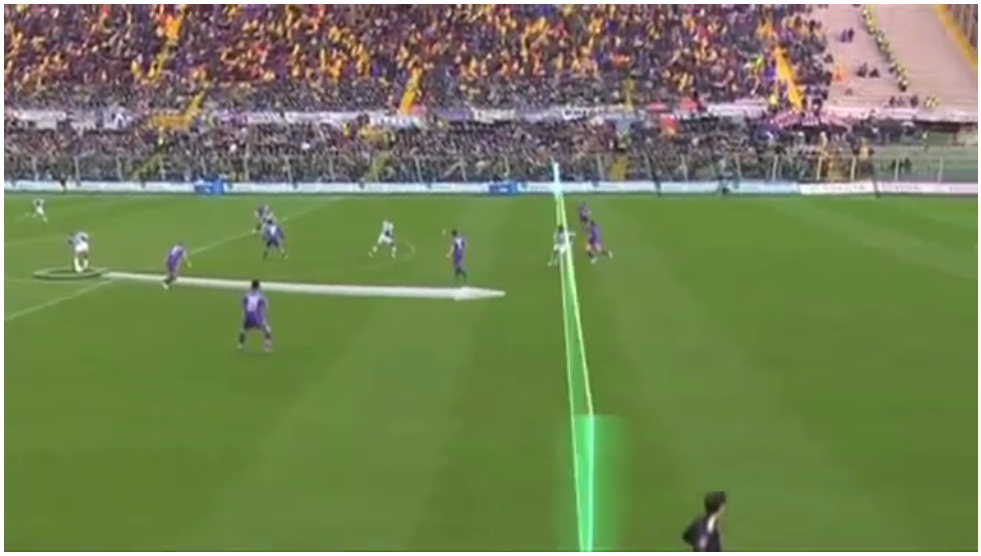
(e) Intermediate position (f) Intermediate position

(g) Intermediate position (h) Rightmost position

Figure 8.28: The Piero system [BBC, 2011]. Interpolation between cameras far apart from each other. Players fade in and out, and ghosting occurs. The arrows are part of the result video and are used for annotation.



(a) Intermediate position



(b) Intermediate position

Figure 8.29: The Piero system [BBC, 2011]. Small virtual camera movements show no artifacts. The wall is part of the annotation system.



(a) Input image (b) Modeled image

(c) Input image (d) Modeled image

Figure 8.30: The Piero system [BBC, 2011]. The piero system, using computer models of the scene. The result has an animated look and is not photorealistic.

8.3 Performance of the Components

To analyze the performance, we split up the algorithm and measure the components independently, with varying number of planes. Figures 8.31, 8.32, 8.33, 8.34, and 8.35 show the results for the median-based method. Figures 8.36, 8.37, 8.38, 8.39, and 8.40 show the results for the histogram-based method. The raw results are given in Table 8.2 for the median-based depth filtering approach, and in Table 8.3 for the histogram-based depth filtering approach. All times were measured using an NVIDIA GTX Titan with compute capability 3.5 and 2688 streaming processors at 876 MHz. While the rendering framerate is not as fast as the capturing framerate, the performance is good enough to allow the practical use of multiple GPUs to acquire real-time results. Because the rendering only uses the information of a single frame, GPU parallelization is possible.

The timings are given for a different number of planes, because this parameter can be reduced without affecting quality when using the plane sweep optimization as described in chapter 7. More planes does not necessarily mean that the quality is higher. 512 planes with a uniform plane distribution and 128 planes with an adaptive plane distribution, as described in chapter 7, are typically used in our results, with comparable quality. When the number of players is low, 64 planes can be used, together with adaptive plane distribution, without loss of quality.

Figure 8.32 show that the main part of the method is the plane sweeping steps. Both the initial plane sweep and the depth-aware plane sweep take up a large portion of the method. This is the same for the histogram-based method, as shown in Figure 8.37, where the portion of the depth-aware plane sweep is even larger. This is caused by the increased data requirements, compared to the median-based approach. The median-based approach only used a single map, containing the median values. The histogram-based approach uses a different validity map per depth hypothesis. This validity map must be calculated every time before the plane sweep can actually process a depth hypothesis, hence the increased processing requirements.

The plane sweep steps are strongly dependent of the number of depth hypotheses used. When looking at Figure 8.31 and Figure 8.36, we notice that only the plane sweep steps are influenced by the number of depth hypotheses. This is trivial, as the other steps do not use the possible depth values directly, but only the depth maps itself. This will, however, change the processing distribution from plane sweeping to depth map filtering. This is best shown in Figure 8.35, where a low number of depth hypotheses is used. We can see that the depth map filtering already takes 34% of the processing power, which demonstrates that the depth filtering should not be neglected when analyzing the performance of the system.

The timings of the components may change when the number of players is varied. The histogram calculation and filtering is dependent on the number of histograms, which is dependent on the number of players. The less foreground objects, the faster these steps. The plane sweep steps, however, are not influenced by the number of players.

Component	512 planes	256 planes	128 planes	64 planes
Background Plane Generation	15.6122	9.17679	13.3264	4.53509
Background Color Generation	2.55498	2.57462	2.6001	2.56956
Initial Plane Sweep	79.0522	40.3802	20.8254	10.8444
Label Determination	3.99372	3.95211	3.60307	3.37141
Depth Map Filtering	17.8132	18.407	20.7038	16.841
Depth-aware Plane Sweep	80.23	40.8671	21.5556	11.1356
Merge Foreground/Background	0.22935	0.22	0.228	0.226
Total	149.709	116.186	99.358	83.029

Table 8.2: Timings for the median-based depth filtering approach, divided by their components and number of planes. The times are in milliseconds. The times of the components do not add up to the total due to the non-overlapping processing steps (such as host-device copying and calculations) when measuring the performance of the components.

Component	512 planes	256 planes	128 planes	64 planes
Background Plane Generation	16.3573	10.2752	13.6434	5.05724
Background Color Generation	2.56434	2.56292	2.89968	2.89492
Initial Plane Sweep	77.0628	39.0316	23.9255	12.1731
Label Determination	3.909	3.86994	4.08283	3.85293
Histogram Calculation	3.27499	3.29332	3.6703	3.63784
Histogram Filtering	3.913	3.90459	3.89656	3.91725
Depth-aware Plane Sweep	139.967	114.116	58.1498	26.5797
Merge Foreground/Background	0.22	0.22	0.23	0.23
Total	199.645	182.806	116.297	99.4668

Table 8.3: Timings for the histogram-based depth filtering approach, divided by their components and number of planes. The times are in milliseconds. The times of the components do not add up to the total due to the non-overlapping processing steps (such as host-device copying and calculations) when measuring the performance of the components.

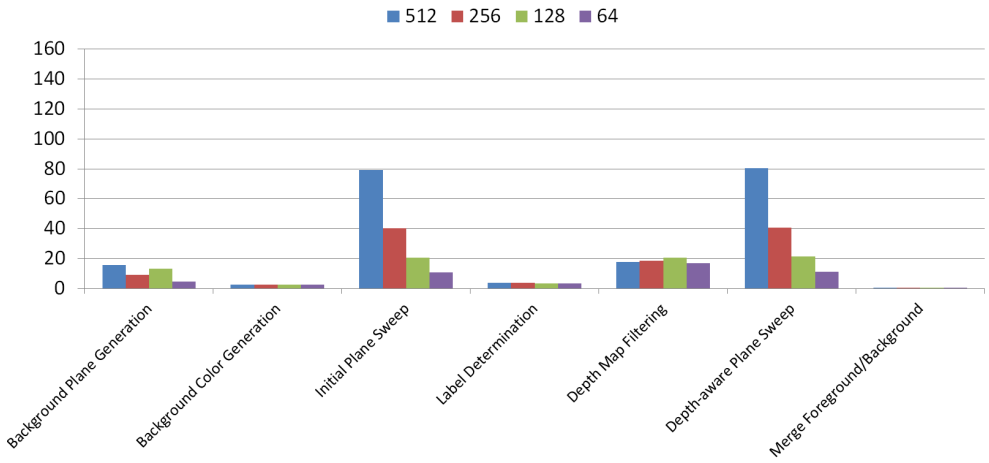


Figure 8.31: Timings per component, shown per number of planes, for the median-based approach.

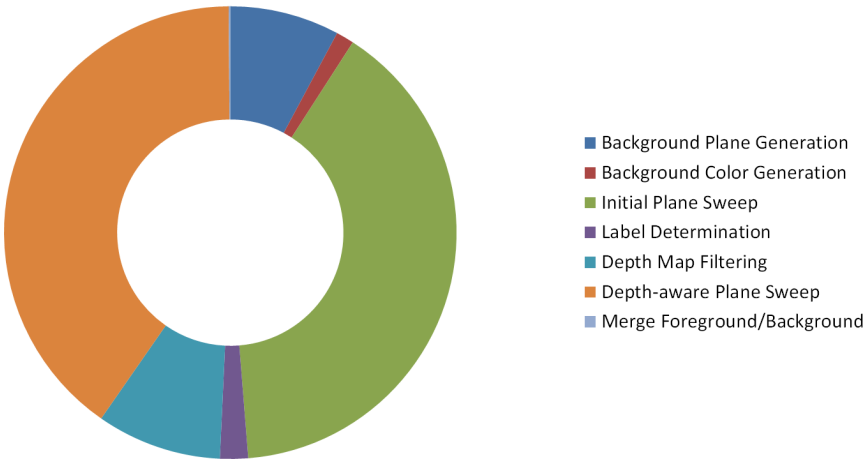


Figure 8.32: Timings broken up in components, median-based approach, 512 planes.

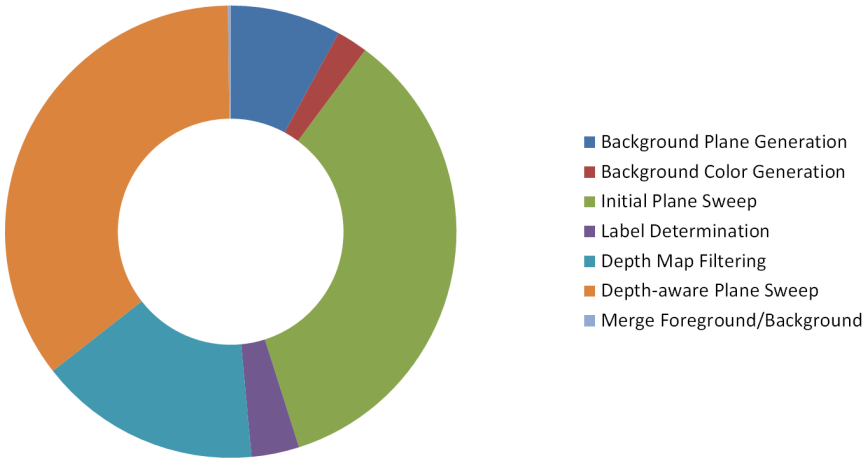


Figure 8.33: Timings broken up in components, median-based approach, 256 planes.

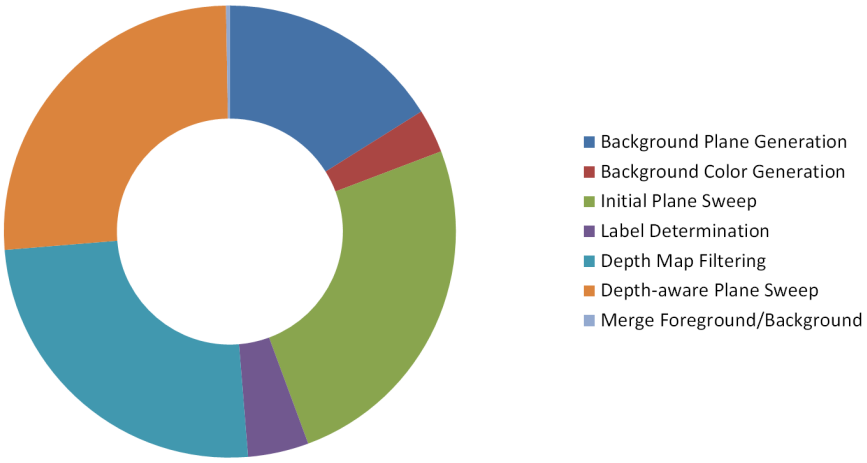


Figure 8.34: Timings broken up in components, median-based approach, 128 planes.

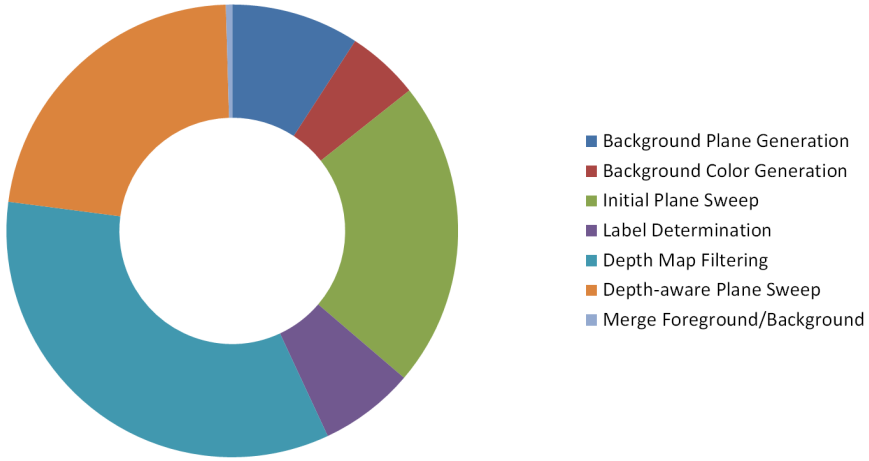


Figure 8.35: Timings broken up in components, median-based approach, 64 planes.

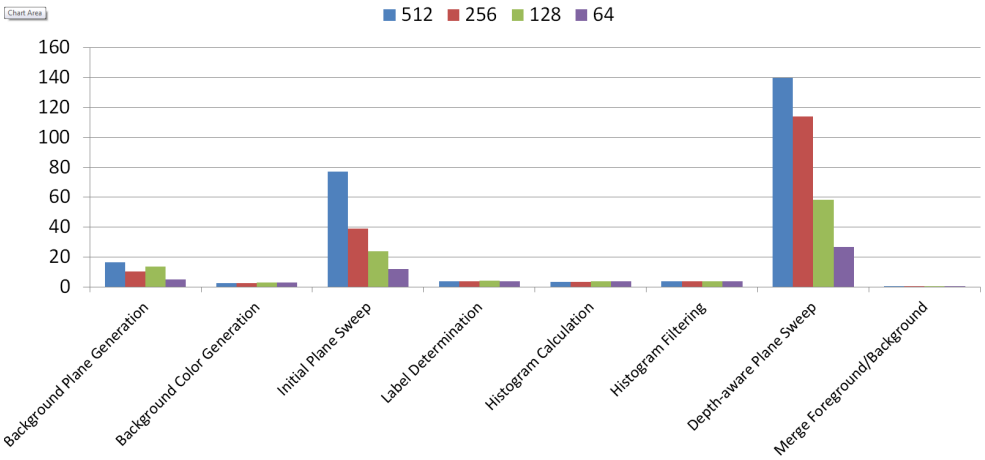


Figure 8.36: Timings per component, shown per number of planes, for the histogram-based approach.

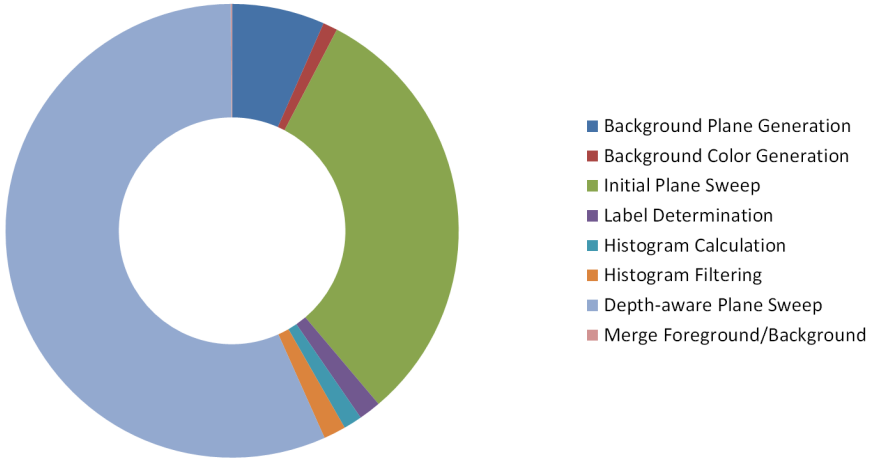


Figure 8.37: Timings broken up in components, histogram-based approach, 512 planes.

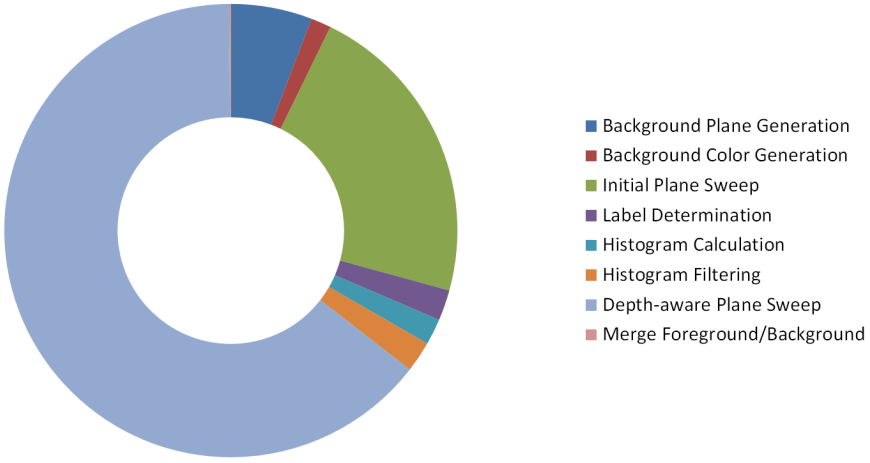


Figure 8.38: Timings broken up in components, histogram-based approach, 256 planes.

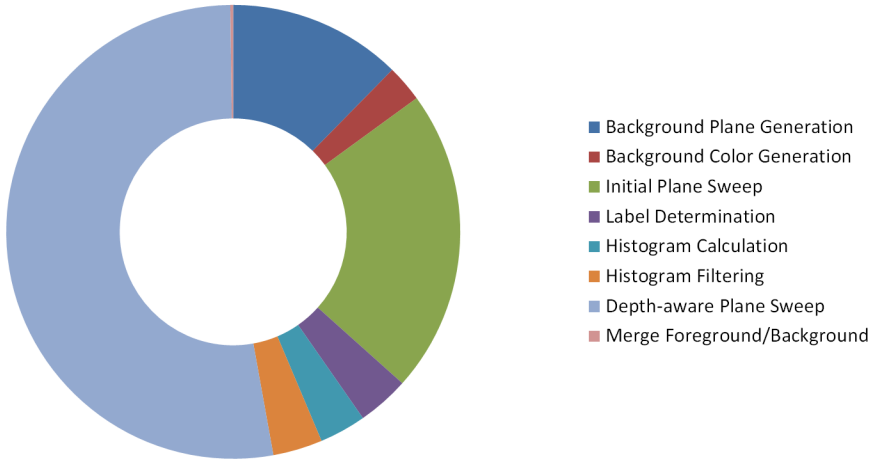


Figure 8.39: Timings broken up in components, histogram-based approach, 128 planes.

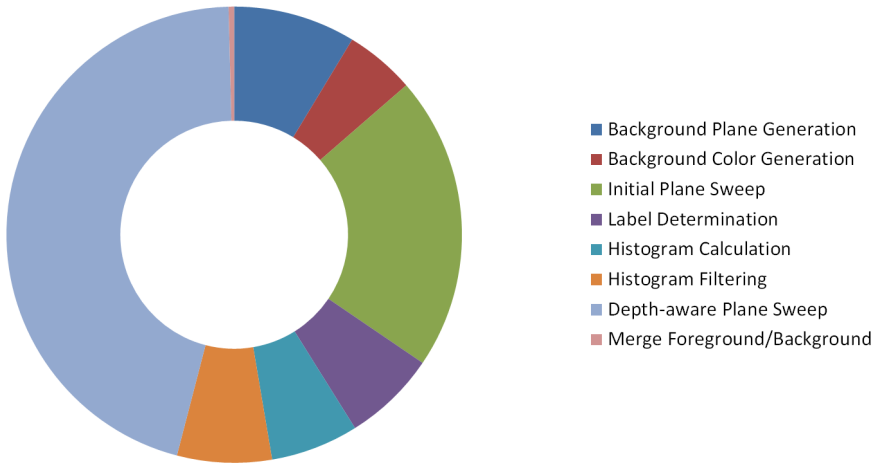


Figure 8.40: Timings broken up in components, histogram-based approach, 64 planes.

Chapter 9

Conclusions and Future Work

9.1 Conclusions of this Dissertation **169**
9.2 Future Work **171**

We presented the results and conclusions per chapter and section. The discussion of the results itself can be found in chapter 8. Here, we will present the global conclusions of our method, and provide some suggestions for future work to improve the method.

9.1 Conclusions of this Dissertation

In this dissertation, we presented a system to render the images of a virtual camera in soccer scenes. A number of static cameras are placed around the pitch. These images are used to generate the image of a virtual camera, placed in between the real cameras. This virtual camera can be used to generate virtual rail cameras, to look around a frozen frame, to provide pleasant camera transitions, et cetera. We designed the method to be automatic, to have similar quality as the input images, and to be scalable to more cameras and more processing units. To allow performant and scalable processing, we used both traditional and modern GPU technologies.

To calibrate the cameras geometrically, we determine correspondences between image pairs using SIFT feature detection and feature matching. These image pair matches are then processed by a consensus-based voting system to generate multicamera feature matches, that determines how much different image pair matches agree to each other for a specific multicamera match. The multicamera matches are filtered using an angle-based approach, where matches that differ too much from the other matches are eliminated. The resulting matches are fed to existing camera calibration toolboxes. The method proved to be practical and robust for our intended camera setups.

Once the calibration was performed, we can start the generation of the virtual image itself. The user of the system determines a virtual viewpoint – or multiple viewpoints on a view path – represented as a position and orientation, placed on a straight line between the cameras or on a Catmull spline. We demonstrated that this method of determining a virtual camera position is intuitive and easy. The virtual viewpoint, together with a time in the video sequences, is provided to the rendering software.

For each frame, there is a frame preparation stage and a rendering stage. In the frame preprocessing stage, the input images are first debayered using a FIR filtering approach using CUDA. By determining the optimal approach to FIR filtering for these specific filters, we were able to maximize the performance of the debayering. Next, we perform a foreground/background segmentation based on the previously determined backgrounds. We use 3 thresholds to allow a high quality segmentation, which we demonstrated to be of sufficient quality. Shadows are considered as background, and the goal is considered as foreground. Once these steps are done, we process the foreground and background independently.

We render the background by projecting the background pixels of the input cameras onto the pitch plane. The location of the pitch plane is known thanks to the geometric calibration. Because we don't use the previously calculated backgrounds, shadows are retained in the rendering.

The foreground is rendered in 3 phases: an initial plane sweep, a depth filtering, and a depth-aware plane sweep. These steps did seriously reduce common artifacts, such as ghost legs and players, halo effects around players, and noise on the final image. The plane sweep phase creates a plane on a specific depth before the virtual camera. All input images are projected on this plane, back projected on the virtual camera image, and the color consistency of the foreground is calculated per pixel. This is repeated for a number of depths, where the plane sweeps through the space before the virtual camera. For every pixel on the virtual image, the depth with the best color consistency is stored, together with its average color value. This method will generate a virtual viewpoint, but will have some artifacts. We detected that these artifacts, such as ghosting and noise, have a distinct depth value, different from the correct depth values.

Therefore, we proposed a depth filtering phase. In this phase, the players and other foreground objects are determined in the virtual image. We use a pixel-based seed growing method on the GPU to determine connected groups of pixels. These groups of pixels represent a player or multiple players. Next, we determine the allowed depth value per group. One way to do this is by determining the median value of the depth values per group of pixels, and only allowing the depth values in a range around this median value. Another way is by calculating the histogram of the depth values per group of pixels and selecting a range around the peaks as the allowed depth values. Each group of pixels will generally have a different range of allowed depth values.

Once these allowed depth values are known, we perform another plane sweep, where we will skip all the depth values that are outside the allowed range. Because each pixel will generally have a different allowed depth range, the complete plane sweep must be repeated. Furthermore, depth values are checked against the depth of the background, to eliminate ghost players that appear to be under the ground or high in the air.

The results show that most of the artifacts are effectively eliminated. In complex scenes, where there are multiple players in one group of pixels, histogram-based filtering is required to avoid the filtering of correct players. The processing time is in the order of a few frames per second, thanks to the use of GPU computing for the complete rendering pipeline. Our system is visually comparable with other existing and commercial systems. We were able to eliminate many artifacts that were still present in these other systems. Some other systems, however, are more flexible and can be used on existing video streams without specialized setups.

To increase performance, we proposed a system to redistribute the computational power of the GPU during the plane sweep phases. To do that, we use the cumulative histogram of the depth values of the previous (temporal) frame. When the cumulative histogram is steep, many corresponding depth values are present in the depth map, and vice versa. We convert the steepness of the cumulative histogram to a plane distribution, such that the depth plane distribution is more dense for the more occurring depth values. This way, there will be more planes at the depths where there actually are objects, and less planes at the depths without

objects. The results show that this redistribution can be used to reduce the number of planes, without reducing the quality, in soccer scenes. The method can also be applied for any kind of scene, as demonstrated in the results, but works best for scenes with sparse objects.

9.2 Future Work

After showing the method and the results, we now propose directions for improvement.

The background quality can be improved using, for example, a model-based rendering, where the goal and the pitch are modeled in advance [Li and Flierl, 2012]. By using a model, interpolation errors may be seriously reduced in the background.

Overall quality may be improved by using prosumer cameras. These are now not considered due to the significantly higher hardware cost. The recordings, however, will have a noticeably higher quality. The interpolation method described in this dissertation has to be able to handle these video input, but tests must be conducted to be certain.

There are some artifacts left in special cases, such as the case where all the players are together at one spot. These cases must be considered, and other depth filtering methods may handle them.

Temporal information can be incorporated in the overall method to allow consistency and additional artifact reduction. We opted for a method where each frame can be rendered independently to allow parallel and distributed rendering methods. Temporal information, however, may increase quality and is worth considering.

In a future system, moving cameras must be considered. Our system using static cameras has a number of benefits, as discussed in section 2.2.2, including replay possibilities, no operator requirement, and easier calibration and background subtraction. There are, however, some disadvantages, including a higher number of required cameras, the use of a specialized setup, et cetera. If moving cameras are considered, real-time calibration and foreground/background segmentation based on a single frame is required, and not enough background might be available for filling up the missing virtual background. Furthermore, zoom level and the point of interest must be synchronized over all cameras.

If the system must work together with other sports-related systems, such as player tracking, advertisement placement, et ceterea, the camera calibration must be performed in the coordinate system of the real world. Now, the coordinate system is arbitrary. This is sufficient, because virtual camera positions are relative from the real camera positions. This is, however, not sufficient if other systems have to be incorporated.

Appendix A

Recordings

A.1 The Genk Dataset **175**
A.2 The Barcelona Dataset **181**

A number of recordings were made to provide real soccer data to demonstrate the view interpolation. These recordings are described below. Datasets to perform initial algorithm or capture tests are not described as not relevant to the results.

A.1 The Genk Dataset

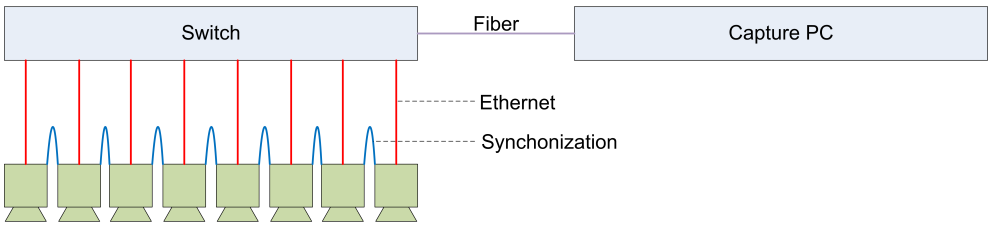


Figure A.1: Scheme of the Genk setup.

The second relevant dataset was recorded in the Fenix stadium in Genk, Belgium. We used 8 Basler avA1600-50gc cameras, placed on a line approximately 1 meter apart from each other. This will provide the effect of a camera on rails without moving elements. In the first half, we used 25*mm* lenses, and in the second half 12.5*mm* lenses. The recordings were done at 60 Hz using a resolution of 1600x1200. The height of the players range from 50 to 160 pixels for the 12.5*mm* lenses, and from 170 to 270 pixels for the 25*mm* lenses.

To transfer the captured data, Gigabit Ethernet connections of 10 meters were used to a 10 Gigabit Ethernet switch, which was connected to a storage device using a 100 meters 10 Gigabit fiber Ethernet connection. Ethernet connections proved to be more reliable than previously used FireWire connections. Furthermore, capturing using a single computer becomes more centralized, allowing more control over synchronization of the different input streams and reducing the need for managing multiple devices simultaneously. The system is conceptually depicted in Figure A.1, and photos of the setup are shown in Figure A.2.

Hardware synchronization between different cameras was done by connecting the cameras in a daisy chain setup. The first camera generates a signal indicating the capturing of a frame. The next camera uses this signal to capture a frame and generate a signal for the next camera, and so forth. The input and output signals are camera dependent; in this case a 5V DC pulse signal. The cameras are connected using 10 meters cables with RCA connectors, but even 100 meter cable segments have been reliably used in other setups we deployed. Thanks to the confined stand-alone setup and the already strict synchronization, no timestamps were recorded along the images. Sound was not recorded, as we did not find it relevant for the intended application.

A single frame of the recordings can be found in Figures A.3 and A.4 for the 12.5*mm* lenses, and in Figures A.5 and A.6 for the 25*mm* lenses.



Figure A.2: Photos of the Genk recording setup.



Figure A.3: Example of the Genk dataset, 12.5*mm*.



Figure A.4: Example of the Genk dataset, 12.5*mm*

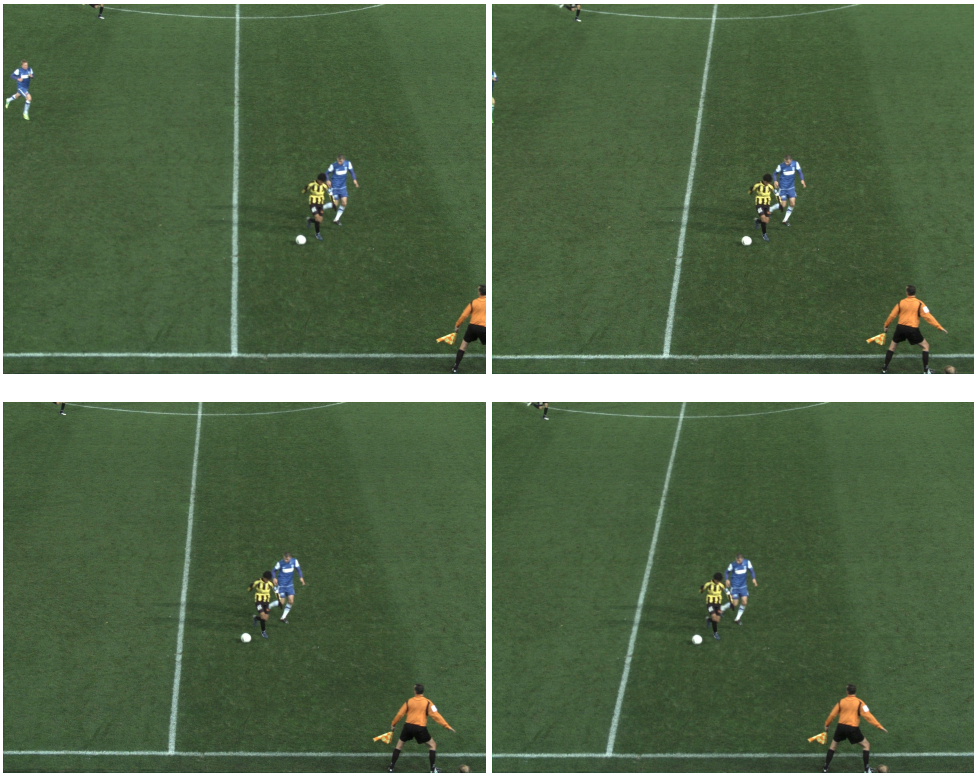


Figure A.5: Example of the Genk dataset, $25mm$**.**

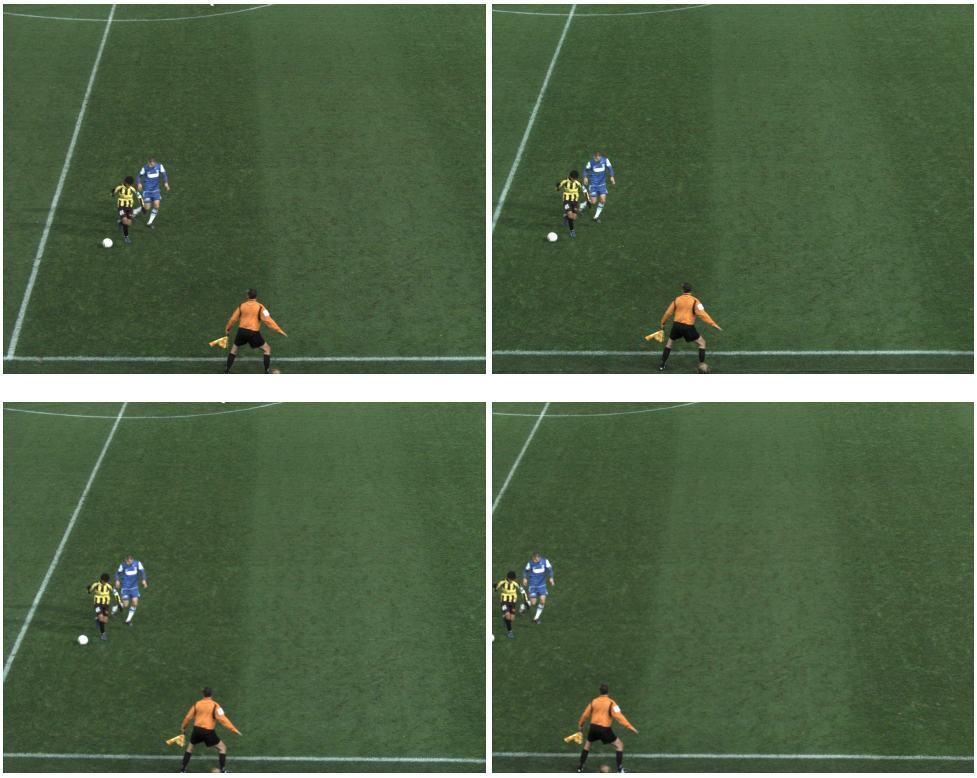


Figure A.6: Example of the Genk dataset, $25mm$**.**

A.2 The Barcelona Dataset

The first relevant dataset was recorded in the mini stadium in Barcelona, Spain, in cooperation with Media Pro and Barcelona Media. Here, 16 Prosilica GC cameras were used, 8 of UHasselt and 8 of Barcelona Media. The cameras were placed around one half of the field, focused on the penalty mark. This way, all-round interpolation is possible, instead of following action across the field (such as the Genk recordings). The cameras were placed 10 meters apart from each other to acquire a broad angular coverage. We used 16*mm* lenses to have a general overview of the scene. The recordings were done at 25 Hz using a resolution of 1920x1080. The height of the players range from 70 to 180 pixels.

The recorded data was stored using a distributed setup, where 3 computers were used to process the data streams. The image data was transmitted in raw format using standard 1 Gigabit Ethernet connections attached to a pair of switches and from there further transmitted to the capture machines using 10 Gigabit fiber. Synchronization was performed using a daisy-chaining approach using one 25 Hz clock as source. The synchronization signal was transmitted from camera to camera using 2×10 meters cables with RCA connectors. Photos of the setup are shown in Figure A.7.

A single frame of the recordings can be found in Figures A.3 and A.4. Because lens flares were visible in some of the images, we opted to only use 7 camera views.

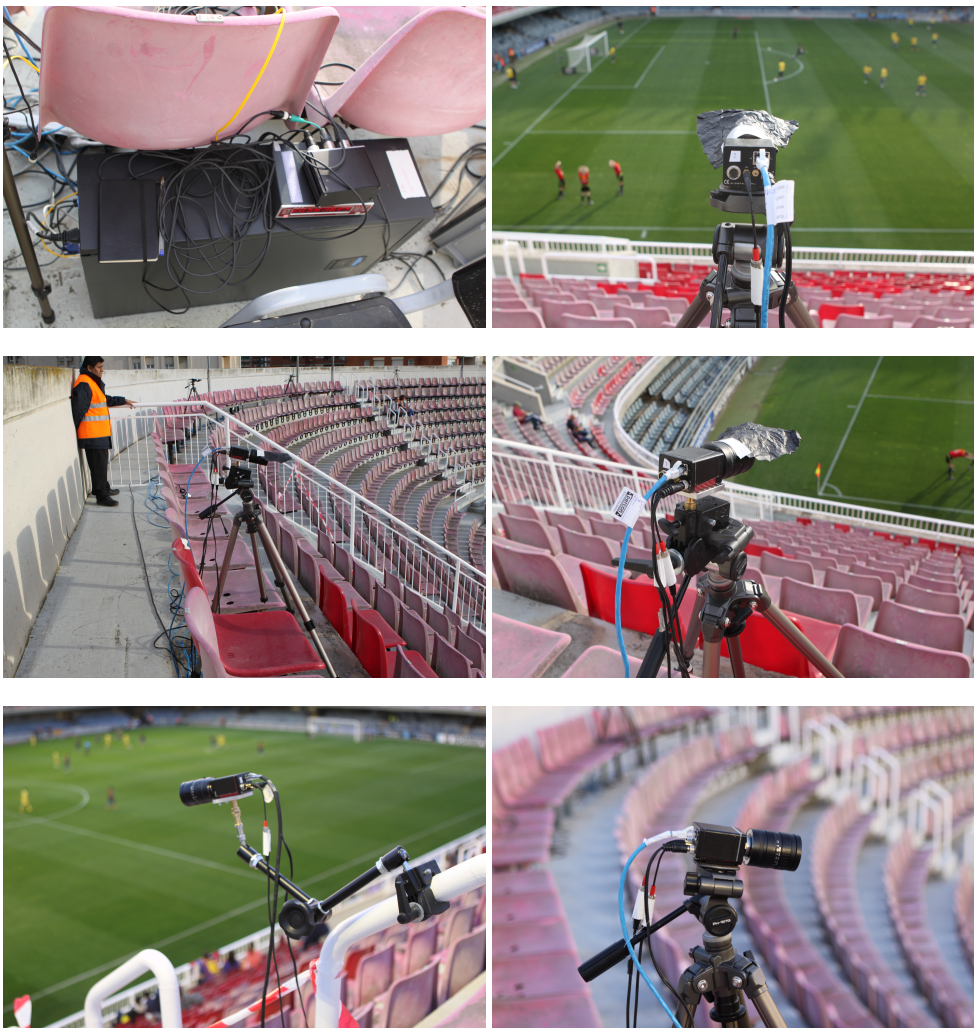


Figure A.7: Photos of the Barcelona recording setup.



Figure A.8: Example of the Barcelona dataset.



Figure A.9: Example of the Barcelona dataset.



Figure A.10: Example of the Barcelona dataset.



Figure A.11: Example of the Barcelona dataset.

Appendix B

Proof of Convolution using SVD Decomposition

B.1 Notations **189**
B.2 Definitions **189**
B.3 Theorem **190**
B.4 Proof **190**
B.5 Separable Filters **194**

In this chapter, we will proof that every FIR filter can be separated in a number of horizontal and vertical FIR filters, allowing optimizations, as discussed in section 5.2.2.2.
More specifically, we will proof that the following procedure is the same as convolving the image I_u with kernel K:

1. Calculate the SVD of the original kernel K, resulting in three matrices U, D and V, where $K = U \times D \times V^T$, and D is a diagonal matrix with elements $d_1 \dots d_n$.
2. For every column u of matrix U, iterate over the following consecutive kernel convolutions:
   (a) Convolve the image with column u of U as an individual single dimensional filter.
   (b) Convolve the result with row v of V^T, multiply with d_u, and save the intermediate result as S_u.
3. Calculate the sum of every S_u. This is the final result.

B.1 Notations

- $I_u[x][y]$: Pixel x,y of input image I_u
- $I_r[x][y]$: Pixel x,y of resulting image I_r
- $K[x][y]$: Element x,y of the filter K. K has dimensions WxH. W and H are odd, and not necessarily equal. The point $K[0][0]$ lies in the center.

B.2 Definitions

- Normal, conventional FIR filtering of the image I_u using filter K:

$$C_n(I_u,K) = I_r : \forall x,\forall y \in I_u : I_r[x][y] = \sum_{i=-(W-1)/2}^{i=(W-1)/2} \sum_{j=-(H-1)/2}^{j=(H-1)/2} K[i][j] \times I_u[x+i][y+j]$$

- Singular value decomposition of a matrix M:

$$M = U \times D \times V^T = \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix} \times \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{pmatrix} \times \begin{pmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \\ v_{13} & v_{23} & v_{33} \end{pmatrix}^T$$

- Element e of a matrix:

  e.g Row 2, column 3: $e = M[3][2]$

- Column of a matrix:

  e.g. Column 1 of $M = M[1][:] = M[1:]$

B.3 Theorem

Defining the previously given procedure in symbols:

$$\begin{aligned}(U,D,V) &= \text{SVD}(K)\\ C_s(I_u,K) &= \sum_{i=1}^{i=W} C(C(I_u, d_i \times U[i:]), V^T[:i])\end{aligned} \tag{B.1}$$

When K has an unequal width and height, it should be padded with rows or columns, filled with zeros, to make it square, yielding $W = H$.
Then we state:

$$C_s(I_u,K) = C_n(I_u,K)$$

B.4 Proof

We will use a filter of 3*x*3, without loss of generality. The proof can be extended to larger filter sizes.
The FIR filtering using the SVD decomposition is then:

$$\begin{aligned}C_s(I_u,K) &= \sum_{i=1}^{i=W} C(C(I_u, d_i \times U[i:]), V^T[i:])\\ &= \sum_{i=1}^{i=3} C(C(I_u, d_i \times U[i:]), V^T[i:])\\ &= \quad C(C(I_u, d_1 \times U[1:]), V^T[1:])\\ &\quad + C(C(I_u, d_2 \times U[2:]), V^T[2:])\\ &\quad + C(C(I_u, d_3 \times U[3:]), V^T[3:])\end{aligned} \tag{B.2}$$

Now we calculate what $C(I_u, d_i \times U[i:])$ is for each element of I_u

$$\begin{aligned}
&\forall x, \forall y : C(I_u[x][y], d_i \times U[i:]) \\
&\quad = d_i \times I_u[x][y-1] \times U[i][1] + d_i \times I_u[x][y] \times U[i][2] + d_i \times I_u[x][y+1] \times U[i][3] \\
&\quad = Q_i[x][y]
\end{aligned}$$

We apply this to equation B.2:

$$\begin{aligned}
C_s(I_u, K) = & C(C(I_u, d_i \times U[1:]), V^T[1:]) \\
& + C(C(I_u, d_i \times U[2:]), V^T[2:]) \\
& + C(C(I_u, d_i \times U[3:]), V^T[3:]) \\
= & C(Q_1, V[1:]) + C(Q_2, V[2:]) + C(Q_3, V[3:]) \\
= & R_1 + R_2 + R_3
\end{aligned}$$

$$\begin{aligned}
&\forall x, \forall y : \\
&\quad R_1 = Q_1[x-1][y] \times V[1][1] + Q_1[x][y] \times V[1][2] + Q_1[x+1][y] \times V[1][3] \\
&\qquad = (d_1 \times I_u[x-1][y-1] \times U[1][1] \\
&\qquad\quad + d_1 \times I_u[x-1][y] \times U[1][2] \\
&\qquad\quad + d_1 \times I_u[x-1][y+1] \times U[1][3] \\
&\qquad\quad) \times V[1][1] \\
&\qquad + (d_1 \times I_u[x][y-1] \times U[1][1] \\
&\qquad\quad + d_1 \times I_u[x][y] \times U[1][2] \\
&\qquad\quad + d_1 \times I_u[x][y+1] \times U[1][3] \\
&\qquad\quad) \times V[1][2] \\
&\qquad + (d_1 \times I_u[x+1][y-1] \times U[1][1] \\
&\qquad\quad + d_1 \times I_u[x+1][y] \times U[1][2] \\
&\qquad\quad + d_1 \times I_u[x+1][y+1] \times U[1][3] \\
&\qquad\quad) \times V[1][3]
\end{aligned}$$

$$
\begin{aligned}
&\forall x, \forall y : \\
&\quad R_2 = Q_2[x-1][y] \times V[2][1] + Q_2[x][y] \times V[2][2] + Q_2[x+1][y] \times V[2][3] \\
&\qquad = (d_2 \times I_u[x-1][y-1] \times U[2][1] \\
&\qquad\quad + d_2 \times I_u[x-1][y] \times U[2][2] \\
&\qquad\quad + d_2 \times I_u[x-1][y+1] \times U[2][3] \\
&\qquad\quad) \times V[2][1] \\
&\qquad + (d_2 \times I_u[x][y-1] \times U[2][1] \\
&\qquad\quad + d_2 \times I_u[x][y] \times U[2][2] \\
&\qquad\quad + d_2 \times I_u[x][y+1] \times U[2][3] \\
&\qquad\quad) \times V[2][2] \\
&\qquad + (d_2 \times I_u[x+1][y-1] \times U[2][1] \\
&\qquad\quad + d_2 \times I_u[x+1][y] \times U[2][2] \\
&\qquad\quad + d_2 \times I_u[x+1][y+1] \times U[2][3] \\
&\qquad\quad) \times V[2][3]
\end{aligned}
$$

$$
\begin{aligned}
&\forall x, \forall y : \\
&\quad R_3 = Q_3[x-1][y] \times V[3][1] + Q_3[x][y] \times V[3][2] + Q_3[x+1][y] \times V[3][3] \\
&\qquad = (d_3 \times I_u[x-1][y-1] \times U[3][1] \\
&\qquad\quad + d_3 \times I_u[x-1][y] \times U[3][2] \\
&\qquad\quad + d_3 \times I_u[x-1][y+1] \times U[3][3] \\
&\qquad\quad) \times V[3][1] \\
&\qquad + (d_3 \times I_u[x][y-1] \times U[3][1] \\
&\qquad\quad + d_3 \times I_u[x][y] \times U[3][2] \\
&\qquad\quad + d_3 \times I_u[x][y+1] \times U[3][3] \\
&\qquad\quad) \times V[3][2] \\
&\qquad + (d_3 \times I_u[x+1][y-1] \times U[3][1] \\
&\qquad\quad + d_3 \times I_u[x+1][y] \times U[3][2] \\
&\qquad\quad + d_3 \times I_u[x+1][y+1] \times U[3][3] \\
&\qquad\quad) \times V[3][3]
\end{aligned}
$$

The result is equal to $R_1 + R_2 + R_3$. We will calculate this by combining all components of I_u. We will calculate C, such that:

$$C = \begin{pmatrix} c_{11} & c_{21} & c_{31} \\ c_{12} & c_{22} & c_{32} \\ c_{13} & c_{23} & c_{33} \end{pmatrix} \tag{B.3}$$

and

$$\begin{aligned} I_r[x][y] = {} & c_{11} \times I_u[x-1][y-1] \\ & + c_{12} \times I_u[x-1][y] \\ & + c_{13} \times I_u[x-1][y+1] \\ & + c_{21} \times I_u[x][y-1] \\ & + c_{22} \times I_u[x][y] \\ & + c_{23} \times I_u[x][y+1] \\ & + c_{31} \times I_u[x+1][y-1] \\ & + c_{32} \times I_u[x+1][y] \\ & + c_{33} \times I_u[x+1][y+1] \end{aligned}$$

$$\begin{aligned} c_{11} &= d_1 \times V[1][1] \times U[1][1] + d_2 \times V[2][1] \times U[2][1] + d_3 \times V[3][1] \times U[3][1] \\ c_{12} &= d_1 \times V[1][1] \times U[1][2] + d_2 \times V[2][1] \times U[2][2] + d_3 \times V[3][1] \times U[3][2] \\ c_{13} &= d_1 \times V[1][1] \times U[1][3] + d_2 \times V[2][1] \times U[2][3] + d_3 \times V[3][1] \times U[3][3] \\ c_{21} &= d_1 \times V[1][2] \times U[1][1] + d_2 \times V[2][2] \times U[2][1] + d_3 \times V[3][2] \times U[3][1] \\ c_{22} &= d_1 \times V[1][2] \times U[1][2] + d_2 \times V[2][2] \times U[2][2] + d_3 \times V[3][2] \times U[3][2] \\ c_{23} &= d_1 \times V[1][2] \times U[1][3] + d_2 \times V[2][2] \times U[2][3] + d_3 \times V[3][2] \times U[3][3] \\ c_{31} &= d_1 \times V[1][3] \times U[1][1] + d_2 \times V[2][3] \times U[2][1] + d_3 \times V[3][3] \times U[3][1] \\ c_{32} &= d_1 \times V[1][3] \times U[1][2] + d_2 \times V[2][3] \times U[2][2] + d_3 \times V[3][3] \times U[3][2] \\ c_{33} &= d_1 \times V[1][3] \times U[1][3] + d_2 \times V[2][3] \times U[2][3] + d_3 \times V[3][3] \times U[3][3] \end{aligned}$$

Now, we will calculate the SVD itself:

$$\begin{aligned} M = U \times D \times V^T &= \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix} \times \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{pmatrix} \times \begin{pmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \\ v_{13} & v_{23} & v_{33} \end{pmatrix}^T \\ &= \begin{pmatrix} d_1 \times u_{11} & d_2 \times u_{21} & d_3 \times u_{31} \\ d_1 \times u_{12} & d_2 \times u_{22} & d_3 \times u_{32} \\ d_1 \times u_{13} & d_2 \times u_{23} & d_3 \times u_{33} \end{pmatrix} \times \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix} \end{aligned}$$

Column 1:

$$\begin{pmatrix} d_1 \times u_{11} \times v_{11} + d_2 \times u_{21} \times v_{21} + d_3 \times u_{31} \times v_{31} \\ d_1 \times u_{12} \times v_{11} + d_2 \times u_{22} \times v_{21} + d_3 \times u_{32} \times v_{31} \\ d_1 \times u_{13} \times v_{11} + d_2 \times u_{23} \times v_{21} + d_3 \times u_{33} \times v_{31} \end{pmatrix}$$

Column 2:

$$\begin{pmatrix} d_1 \times u_{11} \times v_{12} + d_2 \times u_{21} \times v_{22} + d_3 \times u_{31} \times v_{32} \\ d_1 \times u_{12} \times v_{12} + d_2 \times u_{22} \times v_{22} + d_3 \times u_{32} \times v_{32} \\ d_1 \times u_{13} \times v_{12} + d_2 \times u_{23} \times v_{22} + d_3 \times u_{33} \times v_{32} \end{pmatrix}$$

Column 3:

$$\begin{pmatrix} d_1 \times u_{11} \times v_{13} + d_2 \times u_{21} \times v_{23} + d_3 \times u_{31} \times v_{33} \\ d_1 \times u_{12} \times v_{13} + d_2 \times u_{22} \times v_{23} + d_3 \times u_{32} \times v_{33} \\ d_1 \times u_{13} \times v_{13} + d_2 \times u_{23} \times v_{23} + d_3 \times u_{33} \times v_{33} \end{pmatrix} \tag{B.4}$$

We notice that the components of equation B.3 are equal to the ones in equation B.4. The result of the FIR filtering using K and the result of the combined FIR filterings using B.1 is the same. QED.

B.5 Separable Filters

Typically, a filter is called separable if it can be decomposed in one row and one column FIR filter. This proof generalizes this idea, where the conventional separable filter results in zero values in all but the first rows and columns of U and V^T, respectively, or zeros on the diagonal of D in all but the first element, therefore resulting in one row and one column with non-zero elements. For example, the decomposition of the Gaussian convolution kernel will result in zero values on the diagonal, and therefore only one row and one column filter:

$$\mathrm{SVD}(K_G) = \mathrm{SVD}(\begin{bmatrix} 0.0232 & 0.0338 & 0.0383 & 0.0338 & 0.0232 \\ 0.0338 & 0.0492 & 0.0558 & 0.0492 & 0.0338 \\ 0.0383 & 0.0558 & 0.0632 & 0.0558 & 0.0383 \\ 0.0338 & 0.0492 & 0.0558 & 0.0492 & 0.0338 \\ 0.0232 & 0.0338 & 0.0383 & 0.0338 & 0.0232 \end{bmatrix}) = U \times D \times V =$$

$$\begin{bmatrix} -0.3342 & 0.9367 & 0.0373 & -0.0978 & 0 \\ -0.4863 & -0.1837 & 0.4722 & 0.0827 & -0.7071 \\ -0.5510 & -0.2344 & -0.5581 & -0.5744 & 0 \\ -0.4863 & -0.1837 & 0.4722 & 0.0827 & 0.7071 \\ -0.3342 & -0.0157 & -0.4912 & 0.8042 & 0 \end{bmatrix} \begin{bmatrix} 0.2081 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.3342 & 0.5322 & 0.7288 & -0.2718 & 0 \\ -0.4863 & 0.3010 & -0.3988 & 0.1179 & -0.7071 \\ -0.5510 & -0.6499 & 0.0269 & -0.5228 & 0 \\ -0.4863 & 0.3010 & -0.3988 & 0.1179 & 0.7071 \\ -0.3342 & -0.3366 & 0.3873 & 0.7906 & -0.0000 \end{bmatrix}$$

List of Symbols

Camera Parameters, Camera Images, and Calibration

(p_x, p_y)	Principal point
χ	3D point in homogeneous coordinates $[WX, WY, WZ, W]^T$
$\tilde{C}$	Camera location
$b_i(x,y), i \in [1,N]$	Pixel (x,y) of the background of camera i
$B_i, i \in [1,N]$	Background for camera i
B_v	Background for the virtual camera
$C_i, i \in [1,N]$	Camera
C_v	Virtual camera
f	Focal distance
F_v	Foreground for the virtual camera
$I_i^r, i \in [1,N]$	Camera image, raw format
$I_i, i \in [1,N]$	Camera image
I_v	Virtual camera image
K	Intrinsic matrix of a camera
M	Extrinsic matrix of a camera
N	Number of cameras
$P = KM$	Projection matrix of a camera
P^{-1}	Inverse projection matrix of a camera

R	Rotation matrix
$s_i(x,y), i \in [1,N]$	Pixel (x,y) of the segmentation mask of camera image i
$S_i, i \in [1,N]$	Segmentation mask of camera image i
$S_i, i \in [1,N]$	Segmentation mask of the virtual image
x	2D point in homogeneous coordinates $[wx, wy, w]^T$

Feature Detection and Match Determination

$A \leftrightarrow B$	A match between feature A and feature B
$C_f \leftrightarrow C_t$	A match between camera C_f and camera C_t
C_f	"From" camera
C_p	Primary camera
C_s	Subordinate camera
C_t	"To" camera

FIR Filtering

K	Convolution filter

Plane sweeping Parameters

α	Valid range for the median-based validity map
δ	Label of a group of pixels
δ'	Label of a group of pixels, appended with the depth value of the corresponding pixel
ε	Error value
γ	Average color value
Φ_b	Threshold for maximum distance between normalized foreground and background depth
Φ_e	Size of the neighborhood when filtering histograms per pixel group
Φ_h	Threshold for total count of histogram values

Φ_m	Threshold for maximum distance to the normalized median depth value
Φ_n	Threshold for a single histogram value
σ	Error value for an aggregation window
$d_l \in [D_{min}, D_{max}]$	Depth for a pixel where ε is minimal
$D_p \in [D_{min}, D_{max}]$	Depth of a plane
D_{max}	Maximum depth of a plane
D_{min}	Minimum depth of a plane
M	Number of planes
$U, 2 \leq U \leq N$	Number of selected cameras
V_d	Validity map for depth d, containing one for valid and zero for invalid
V_m	Validity map, containing depth values for foreground and zero for background
w	Weighting function for an aggregation window

Plane Redistribution Parameters

λ	Distance between uniform planes on cumulative histogram
σ_m	Value of cumulative histogram for plane m
τ_m	Depth fraction of plane m
ξ	Fraction for linear interpolation between integer values of H
$D_m, D_{min} \leq D_m \leq D_{max}$	Depth of plane m
$H(x), x \in [0,1]$	Cumulative histogram of normalized depth map
$m, 0 \leq m < M$	Plane number
x_{σ_m}	Corresponding integer bin of σ_m

Segmentation Parameters

τ_a	Angle threshold for foreground/background separation
τ_b	Background threshold for foreground/background separation
τ_f	Foreground threshold for foreground/background separation

Nederlandse Samenvatting

In dit doctoraat wordt een systeem beschreven om een virtuele camera te creëren in voetbalscenes. Een virtuele camera laat toe om een beeld van de scene te genereren, zonder dat er een echte, fysieke camera aanwezig is op die plaats. Een virtuele camera kan gebruikt worden om de scene te bevriezen en hierin rond te kijken, om stereobeelden te genereren, om vloeiende overgangen te maken tussen verschillende standpunten, enzovoort.

Het creëren van het beeld van de virtuele camera maakt gebruik van een collectie van echte, statische camera's. We beperken onze virtuele camera tot posities tussen deze echte camera's. De camera's kunnen op een lijn worden geplaatst, of in een cirkel rond het veld. Het systeem is ontworpen om dezelfde kwaliteit te genereren als de invoerbeelden, zowel op vlak van resolutie,scherpte, kleurkwaliteit, enzovoort. Verder is het systeem ontworpen om te worden uitgevoerd op parallelle rekenapparaten, meer specifiek grafische kaarten, gekend onder de engelse naam van GPU en hun GPGPU programmeerparadigma. Dit maakt een snelle berekening mogelijk, in de orde van een 10-tal beelden per seconde. Omdat alle beelden in de videosequentie onafhankelijk van elkaar verwerkt worden, is het mogelijk een klein aantal grafische kaarten te gebruiken om dezelfde uitvoersnelheid te bekomen als de invoersnelheid. Deze 2 aspecten maken het systeem bruikbaar voor televisieuitzendingen.

Er zijn 2 fasen in het systeem: een opzetfase en een berekeningsfase. In de opzetfase worden de echte camera's gekalibreerd en worden de achtergrondafbeeldingen bepaald. In de berekeningsfase worden de effectieve virtuele beelden berekend.

De camerakalibratie is vereist om de locatie, oriëntatie en eigenschappen van de echte camera's te kennen. We doen dit door correspondenties te berekenen tussen de camerabeelden. Deze correspondenties worden vervolgens gebruikt om de kalibratie te berekenen, voorgesteld in conventionele projectieve geometrie.

We berekenen de achtergronden door enkele beelden uit de invoerbeelden te nemen en de mediaan per pixel te berekenen. We doen dit per invoercamera. Omdat de spelers in de scene bewegen, zal dit een geldige achtergrond opleveren. De achtergrondafbeeldingen worden later bijgewerkt om veranderingen in belichting op te vangen.

Eens deze gegevens bekend zijn, kunnen we virtuele beelden genereren. Eerst bepalen we een positie voor de virtuele camera. Omdat we alleen posities tussen de echte camera's toelaten, kan de gebruiker een positie op een curve kiezen.

Eens deze positie bekend is, zullen we het virtuele beeld van deze camera berekenen. Eerst scheiden we de voorgrond van de achtergrond in de invoerbeelden door gebruik te maken van de eerder berekende achtergronden. De achtergrond van de virtuele camera wordt apart van de voorgrond berekend. Dit gebeurt door de achtergronden uit de invoerbeelden op het 3D vlak van het veld te projecteren en bij elkaar te mengen.

De voorgrond berekenen we door dieptehypotheses te testen. We kiezen een vlak voor de virtuele camera op een bepaalde diepte en projecteren de invoerbeelden hierop. In essentie beschouwen we de invoercamera's als projectors en het vlak als een projectiescherm. Verschillende camera's zullen nu een overlappend beeld projecteren, dat informatie over de diepte oplevert. Inderdaad, als de objecten perfect over elkaar geprojecteerd worden, dan gaan we ervan uit dat het object in de scene op dezelfde locatie staat als het vlak, of met andere woorden de diepte van het vlak heeft. Door dit de testen voor verschillende vlakken op verschillende dieptes, kunnen we de meest optimale diepte per object, of per pixel, bepalen. We bepalen het nieuwe, initiële, virtuele beeld door de kleuren van de beste projectie te bewaren per pixel.

Deze methode geeft fouten, zoals een derde been of extra spelers. We wissen deze door de voorgrondobjecten uit het virtuele beeld te extraheren en de diepte per object te filteren op fouten. Dit object kan een speler of een groep van spelers zijn. We gaan ervan uit dat de diepte per speler relatief uniform is, en we merken op dat de dieptes van de fouten zeer verschillend zijn van de correcte diepte. Omdat de fouten relatief klein zijn in de voorgrondobjecten, kunnen we het histogram van de dieptewaarden gebruiken. We bewaren alleen de pieken in het histogram als geldige dieptewaarden voor dat object. We gebruiken nu deze informatie in een tweede dieptehypothesetest, waarbij we alleen de dieptes (per voorgrondobject) beschouwen die in de vorige stap als geldig zijn getest.

We versnellen de berekeningen door geen vlakken te plaatsen op de locaties waar geen spelers zijn. We weten op welke dieptes de voorgrondobjecten waren in het vorige beeld en kunnen deze informatie gebruiken om de dieptehypotheses te concentreren op deze gebieden. Omdat de spelers in een voetbalscene niet verschijnen en verdwijnen, zullen er geen spelers gemist worden.

Onze resultaten geven een goed virtueel beeld voor de meeste scenes en overtreft bestaande systemen. De fouten zijn effectief verwijderd en de de verwerking gebeurt in reële tijd.

Scientific Contributions and Publications

B.6 Related Scientific Publications **203**
B.7 Related Student Theses **204**
B.8 Other Public Dissemination **204**
B.9 Unrelated Scientific Publications **207**

This chapter presents a list of publications and other dissemination activities, related and unrelated to this dissertation.

B.6 Related Scientific Publications

This is a list of scientific publications, related to this PhD dissertation.

[Goorts et al., 2009] Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. Optimal Data Distribution for Versatile Finite Impulse Response Filtering on Next-Generation Graphics Hardware Using CUDA. In *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*, pages 300–307, Shenzhen, China, December 2009

[Goorts et al., 2010] Patrik Goorts, Sammy Rogmans, Steven Vanden Eynde, and Philippe Bekaert. Practical Examples of GPU Computing Optimization Principles. In *Proceedings of the 2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, pages 46–49, Athens, Greece, 2010. IEEE

[Goorts et al., 2012a] Patrik Goorts, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. An End-to-end System for Free Viewpoint Video for Smooth Camera Transitions. In *Proceedings of the Second International Conference on 3D Imaging (IC3D 2012)*, Liege, Belgium, 2012a. 3D Stereo Media
• Won the *Best Paper* award.

[Goorts et al., 2012b] Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. Raw Camera Image Demosaicing using Finite Impulse Response Filtering on Commodity GPU Hardware using CUDA. In *Proceedings of the Tenth International Conference on Signal Processing and Multimedia Applications (SIGMAP 2012)*, Rome, Italy, 2012b. INSTICC

[Goorts et al., 2013a] Patrik Goorts, Cosmin Ancuti, Maarten Dumont, and Philippe Bekaert. Real-time Video-Based View Interpolation of Soccer Events using Depth-Selective Plane Sweeping. In *Proceedings of the Eight International Conference on Computer Vision Theory and Applications (VISAPP 2013)*, pages 131–137, Barcelona, Spain, 2013a. INSTICC

[Goorts et al., 2013b] Patrik Goorts, Steven Maesen, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. Optimization of Free Viewpoint Interpolation by Applying Adaptive Depth Plane Distributions in Plane Sweeping. In *Proceedings of the Tenth International Conference on Signal Processing and Multimedia Applications (SIGMAP 2013)*, Reykjavik, Iceland, 2013b. INSTICC
• Won the *Best Student Paper* award.

[Goorts et al., 2013d] Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. GPU-based View Interpolation for Smooth Camera Transitions in Soccer. In *GPU technology Conference (GTC 2013)*, San Jose, CA, USA, 2013d

[Goorts et al., 2014b] Patrik Goorts, Steven Maesen, Yunjun Liu, Maarten Dumont, Philippe Bekaert, and Gauthier Lafruit. Self-Calibration of Large Scale Camera Networks. In *Proceedings of the 11th International Conference on Signal Processing and Multimedia Applications (SIGMAP 2014)*, pages 1–10, Vienna, Austria, 2014b

[Goorts et al., 2014a] Patrik Goorts, Steven Maesen, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. Free Viewpoint Video for Soccer using Histogram-Based Validity Maps in Plane Sweeping. In *Proceedings of the Ninth International Conference on Computer Vision Theory and Applications (VISAPP)*, Lisbon, Portugal, 2014a. INSTICC

B.7 Related Student Theses

[Grognard et al., 2012] Sander Grognard, Patrik Goorts, Johannes Taelman, and Philippe Bekaert. Autofocus van Cameralenzen. Bachelor's thesis, Hasselt University, Hasselt, Belgium, 2012

[Geebelen et al., 2013] Gert Geebelen, Patrik Goorts, Maarten Dumont, and Philippe Bekaert. Beeldinterpolatie van Voetbalscenes. Master's thesis, Hasselt University, Hasselt, Belgium, 2013

B.8 Other Public Dissemination

- Public demonstration at the International Broadcasting Convention (IBC) in 2012. The interpolation method using the median-based approach, connected to a storage server of EVS for image retrieval, was demonstrated. This is shown in Figure B.1(a).
  • Won the *What Caught My Eye* award.
- Demonstration for the FINE consortium in 2013. The interpolation method using the median-based approach was demonstrated. This is shown in Figure B.1(b).
- Demonstration for the FINE consortium in 2013. The interpolation method using the histogram-based approach, connected to the EVS viewpath selector and storage server, was demonstrated. This is shown in Figure B.2.
- Presented GPU principles for image processing at the HPC information symposium in Leuven, 2010.



(a) Demonstration at IBC 2012



(b) Demonstration for the FINE project, 2013

Figure B.1: Public setup demonstrations of our system, working together with the video storage system of EVS, Belgium.

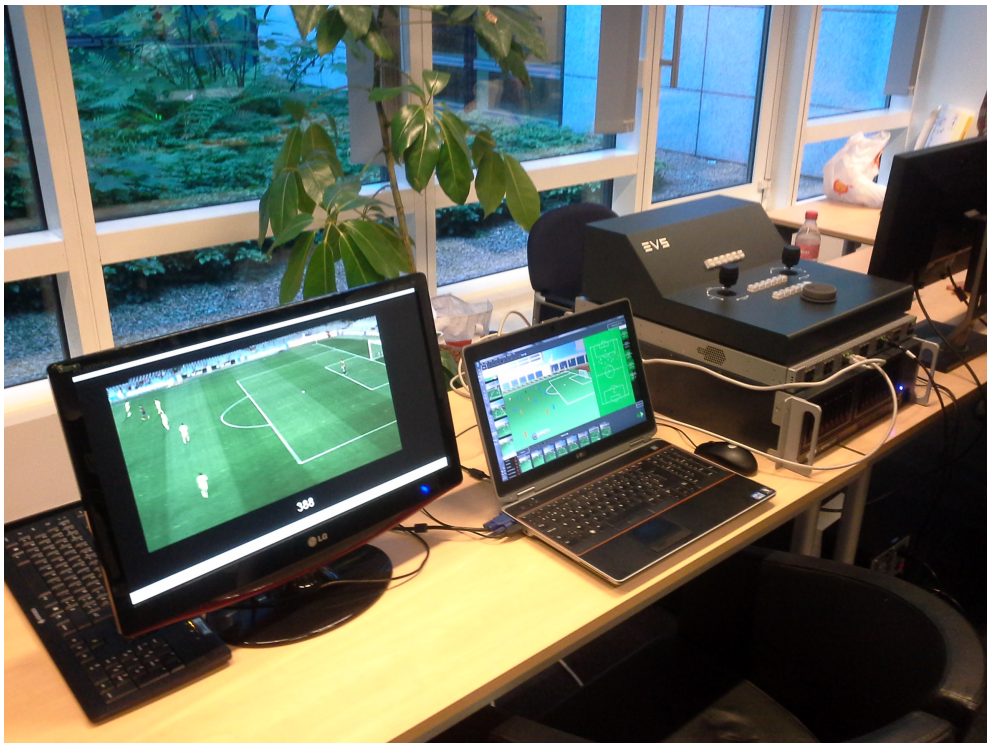


Figure B.2: Public setup demonstrations of our system, working together with the video storage system of EVS, Belgium. Demonstration for the FINE project, 2013.

B.9 Unrelated Scientific Publications

This is a list of other scientific publications, unrelated to this PhD dissertation.

[Maesen et al., 2011] Steven Maesen, Patrik Goorts, Lode Vanacken, Sofie Notelaers, and Tom De Weyer. Look Mother, Virtual Puzzling without Buttons! In *2011 IEEE Symposium on 3D User Interfaces (3DUI 2011)*, pages 139–140, Singapore, March 2011. IEEE. ISBN 978-1-4577-0037-8
• Won second place in the 3DUI contest.

[Goorts and Bekaert, 2012] Patrik Goorts and Philippe Bekaert. ARDO: Automatic Removal of Dynamic Objects. In *Proceedings of the Seventh International Conference on Computer Vision Theory and Applications (VISAPP 2012)*, pages 192–196, Rome, Italy, 2012. INSTICC

[Notelaers et al., 2012] Sofie Notelaers, Tom De Weyer, Patrik Goorts, Steven Maesen, Lode Vanacken, Karin Coninx, and Philippe Bekaert. HeatMeUp: a 3DUI Serious Game to Explore Collaborative Wayfinding. In *2012 IEEE Symposium on 3D User Interfaces (3DUI 2012)*, pages 177–178, Orange County, California, March 2012. IEEE. ISBN 978-1-4673-1204-2
• Won third place in the 3DUI contest.

[Goorts et al., 2013c] Patrik Goorts, Steven Maesen, Dimitri Scarlino, and Philippe Bekaert. Bringing 3D Vision to the Web: Acquiring Motion Parallax using Commodity Cameras and WebGL. In *Proceedings of the International Conference on 3D Imaging (IC3D 2013)*, pages 1–6, Liege, Belgium, 2013c. IEEE

[Maesen et al., 2013] Steven Maesen, Patrik Goorts, and Philippe Bekaert. Scalable Optical Tracking for Navigating Large Virtual Environments using Spatially Encoded Markers. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 101–110, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2379-6

[Dumont et al., 2014a] Maarten Dumont, Patrik Goorts, and Philippe Lafruit. Plane sweeping in eye-gaze corrected, tele-immersive 3d video conferencing. In Branislav Kisaèanin and Margrit Gelautz, editors, *Advances in Embedded Computer Vision*. Springer, 2014a

[Dumont et al., 2014b] Maarten Dumont, Patrik Goorts, Steven Maesen, Philippe Bekaert, and Gauthier Lafruit. Real-time Local Stereo Matching using Edge Sensitive Adaptive Windows. In *Proceedings of the 11th International Conference on Signal Processing and Multimedia Applications (SIGMAP 2014)*, pages 1–10, Vienna, Austria, 2014b

[Jorissen et al., 2014] Lode Jorissen, Patrik Goorts, Bram Bex, Nick Michiels, Sammy Rogmans, Philippe Bekaert, and Gauthier Lafruit. A qualitative comparison of mpeg view synthesis and light field rendering. In *Proceedings of the Converence on 3D TV (3DTV 2014)*, pages 1–4, 2014

Errata and Additions

B.10 Edition 1.1 ..... **209**
B.11 Edition 1.2 ..... **209**

This chapter lists all the changes since the defence of this thesis. The cover and title page is different for the printed editions.

B.10 Edition 1.1

29/06/2014 Added the errata chapter.

29/06/2014 Changed title of section 4.2 from *Traditional GPU Technologies: CUDA* to *Modern GPU Technologies: CUDA*.

29/06/2014 Table 8.1: projecticve → projective.

B.11 Edition 1.2

18/09/2014 Changed references from unpublished to published: [Goorts et al., 2014b], [Dumont et al., 2014b], [Dumont et al., 2014a], [Jorissen et al., 2014].

Bibliography

Edward H Adelson and James R Bergen. The Plenoptic Function and the Elements of Early Vision. *Computational models of visual processing*, 1(2):3–20, 1991.

Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html`.

ATI. *Close-to-metal Guide*, 2009. `http://ati.amd.com/companyinfo/researcher/documents/ATI_CTM_Guide.pdf`.

Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

Peter Bakker, Lucas J. Van Vliet, and Piet W. Verbeek. Edge Preserving Orientation Adaptive Filtering. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 535–540, Fort Collins, CO, USA, June 1999. IEEE.

Bruce Guenther Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, Stanford, CA, USA, 1974.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In *Proceedings of the ninth European Conference on Computer Vision (ECCV)*, pages 404–417, Graz, Austria, May 2006.

Bryce E. Bayer. Color Imaging Array, July 1976. US Patent 3971065.

BBC. Iview: Free-viewpoint Video. Virtual Cameras for Exploring Live Scenes, 2008. `http://www.bbc.co.uk/rd/projects/iview`.

BBC. Piero Sports Graphics System: Putting a New Perspective on Sport, 2010. http://www.bbc.co.uk/rd/projects/piero.

BBC. De Piero-voetbalanalysesoftware van Eredivisie Live, 2011. https://www.youtube.com/watch?v=nNJr1BUlzmw.

BBC. Virtual Reel (Piero and Epsis), 2012. https://www.youtube.com/watch?v=e5VB8rnCgU4.

Leo Beiser. Anaglyph Stereoscopy, June 1981. US Patent 4290675 A.

Alexander Bogomjakov, Craig Gotsman, and Marcus Magnor. Free-viewpoint Video from Depth Cameras. In *Procoeedings of the Vision, Modeling, and Visualization Conference (VMV)*, pages 89–96, Aachen, 2006.

Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-cut/max-flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

Ken Bray and David Kerwin. Modelling the Flight of a Soccer Ball in a Direct Free Kick. *Journal of Sports Sciences*, 21(2):75–85, 2003.

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 777–786, New York, NY, USA, 2004. ACM.

Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 425–432, Los Angeles, California , USA, 2001. ACM.

Massimo Camplani and Luis Salgado. Efficient Spatio-temporal Hole Filling Strategy for Kinect Depth Maps. In *Proceedings of the Three-Dimensional Image Processing (3DIP) and Applications II Conference*, Burlingame, California, USA, 2012.

Joel Carranza, Christian Theobalt, Marcus A Magnor, and Hans-Peter Seidel. Free-viewpoint Video of Human Actors. *ACM Transactions on Graphics (TOG)*, 22(3):569–577, 2003.

Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. LDI Tree: A Hierarchical Representation for Image-based Rendering. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 291–298, Los Angeles, California , USA, 1999. ACM.

Shenchang Eric Chen and Lance Williams. View Interpolation for Image Synthesis. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 279–288, Anaheim, CA, USA, 1993. ACM.

Wan-Yu Chen, Yu-Lin Chang, Shyh-Feng Lin, Li-Fu Ding, and Liang-Gee Chen. Efficient Depth Image-based Rendering with Edge Dependent Depth Filter and Interpolation. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 1314–1317, Amsterdam, The Netherlands, 2005. IEEE.

Kong Man Cheung. *Visual Hull Construction, Alignment and Refinement for Human Kinematic Modeling, Motion Tracking and Rendering*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, 2003.

Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, Interpolation and Animation*. Datalogisk Institut, Københavns Universitet, 1998.

James Davis, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Spacetime stereo: A Unifying Framework for Depth from Triangulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):296–302, 2003.

Paul Debevec, Yizhou Yu, and George Boshokov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *Proceedings of the 9th Eurographics Rendering Workshop*, pages 105–116, Vienna, Austria, 1998.

Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry and Image-based Approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 11–20, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4.

Luigi Di Stefano and Andrea Bulgarelli. A Simple and Efficient Connected Components Labeling Algorithm. In *Proceedings of the International Conference on Image Analysis and Processing*, pages 322–327, Venice, Italy, 1999. IEEE.

Maarten Dumont, Sammy Rogmans, Steven Maesen, and Philippe Bekaert. Optimized Two-party Video Chat with Restored Eye Contact using Graphics Hardware. *Springer Communications in Computer and Information Science*, 48(11):358–372, November 2009.

Maarten Dumont, Patrik Goorts, and Philippe Lafruit. Plane sweeping in eye-gaze corrected, tele-immersive 3d video conferencing. In Branislav Kisaèanin and Margrit Gelautz, editors, *Advances in Embedded Computer Vision*. Springer, 2014a.

Maarten Dumont, Patrik Goorts, Steven Maesen, Philippe Bekaert, and Gauthier Lafruit. Real-time Local Stereo Matching using Edge Sensitive Adaptive Windows. In *Proceedings of the 11th International Conference on Signal Processing and Multimedia Applications (SIGMAP 2014)*, pages 1–10, Vienna, Austria, 2014b.

Jim Easterbrook, Oliver Grau, and Peter Schubel. A System for Distributed Multi-camera Capture and Processing. In *Proceedings of the Conference on Visual Media Production (CVMP)*, pages 107–113, London, UK, 2010. IEEE.

Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson De Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating Textures. *Computer Graphics Forum*, 27(2):409–418, 2008.

Dirk Farin, Susanne Krabbe, Peter H.N. de With, and Wolfgang Effelsberg. Robust Camera Calibration for Sport Videos using Court Models. In *Proceedings of SPIE 5307, Storage and Retrieval Methods and Applications for Multimedia*, pages 80–91, San Jose, CA, 2003.

Dirk Farin, Jungong Han, and Peter H.N. de With. Fast Camera Calibration for the Analysis of Sport Sequences. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–4, Amsterdam, The Netherlands, 2005. IEEE.

Christoph Fehn and Peter Kauff. Interactive Virtual View Video (IVVV)-The Bridge Between Immersive TV and 3D-TV. In *Proceedings of SPIE Three-Dimensional TV, Video and Display I*, pages 14–25, Boston, MA, USA, 2002.

Christoph Fehn, Peter Kauff, Oliver Schreer, and Ralf Schäfer. Interactive Virtual View Video for Immersive TV Applications. In *Proceedings of International Broadcast Conference*, pages 53–61, Amsterdam, the Netherlands, 2001.

Randima Fernando. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-time Graphics*. Addison-Wesley Professional, 2004.

FIFA. Football stadiums: Technical recommendations and requirements. Technical report, FIFA, 2004. `http://www.secopa.ba.gov.br/`.

Martin A Fischler and Robert C Bolles. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Nicolas Fritz, Philipp Lucas, and Philipp Slusallek. CGiS, a New Language for Data-parallel GPU Programming. In *Proceedings of the 9th International Workshop Vision, Modeling, and Visualization(VMV)*, pages 241–248, Stanford, CA, USA, 2004.

Hitoshi Furuya, Itaru Kitahara, Yoshinari Kameda, and Yuichi Ohta. Viewpoint-Dependent Quality Control on Microfacet Billboarding Model for Sports Video. In *Poceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 1199–1202, Beijing, China, 2007. IEEE.

David Gallup, Jan-Michael Frahm, Philippos Mordohai, Qingxiong Yang, and Marc Pollefeys. Real-Time Plane-Sweeping Stereo with Multiple Sweeping Directions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Minneapolis, Minnesota, USA, June 2007. IEEE.

Gert Geebelen, Patrik Goorts, Maarten Dumont, and Philippe Bekaert. Beeldinterpolatie van Voetbalscenes. Master's thesis, Hasselt University, Hasselt, Belgium, 2013.

Marcel Germann, Alexander Hornung, Richard Keiser, Remo Ziegler, Stephan Würmlin, and Markus Gross. Articulated Billboards for Video-based Rendering. *Computer Graphics Forum*, 29(2):585–594, 2010.

Marcel Germann, Tiberiu Popa, Remo Ziegler, Richard Keiser, and Markus Gross. Space-time Body Pose Estimation in Uncontrolled Environments. In *Proceedings of the International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pages 244–251, Hangzhou, China, 2011.

Marcel Germann, Tiberiu Popa, Richard Keiser, Remo Ziegler, and Markus Gross. Novel-View Synthesis of Outdoor Sport Events Using an Adaptive View-Dependent Geometry. *Computer Graphics Forum*, 31(2):325–333, 2012.

Indra Geys, Thomas P Koninckx, and Luc Van Gool. Fast Interpolated Cameras by Combining a GPU based Plane Sweep with a Max-Flow Regularisation Algorithm. In *Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 534–541, Thessaloniki, Greece, 2004. IEEE.

Minglun Gong, Ruigang Yang, Liang Wang, and Mingwei Gong. A Performance Study on Different Cost Aggregation Approaches used in Real-Time Stereo Matching. *International Journal of Computer Vision*, 75(2):283–296, November 2007.

Patrik Goorts and Philippe Bekaert. ARDO: Automatic Removal of Dynamic Objects. In *Proceedings of the Seventh International Conference on Computer Vision Theory and Applications (VISAPP 2012)*, pages 192–196, Rome, Italy, 2012. INSTICC.

Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. Optimal Data Distribution for Versatile Finite Impulse Response Filtering on Next-Generation Graphics Hardware Using CUDA. In *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*, pages 300–307, Shenzhen, China, December 2009.

Patrik Goorts, Sammy Rogmans, Steven Vanden Eynde, and Philippe Bekaert. Practical Examples of GPU Computing Optimization Principles. In *Proceedings of the 2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, pages 46–49, Athens, Greece, 2010. IEEE.

Patrik Goorts, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. An End-to-end System for Free Viewpoint Video for Smooth Camera Transitions. In *Proceedings of the Second International Conference on 3D Imaging (IC3D 2012)*, Liege, Belgium, 2012a. 3D Stereo Media.

Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. Raw Camera Image Demosaicing using Finite Impulse Response Filtering on Commodity GPU Hardware using CUDA. In *Proceedings of the Tenth International Conference on Signal Processing and Multimedia Applications (SIGMAP 2012)*, Rome, Italy, 2012b. INSTICC.

Patrik Goorts, Cosmin Ancuti, Maarten Dumont, and Philippe Bekaert. Real-time Video-Based View Interpolation of Soccer Events using Depth-Selective Plane Sweeping. In *Proceedings of the Eight International Conference on Computer Vision Theory and Applications (VISAPP 2013)*, pages 131–137, Barcelona, Spain, 2013a. INSTICC.

Patrik Goorts, Steven Maesen, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. Optimization of Free Viewpoint Interpolation by Applying Adaptive Depth Plane Distributions in Plane Sweeping. In *Proceedings of the Tenth International Conference on Signal Processing and Multimedia Applications (SIGMAP 2013)*, Reykjavik, Iceland, 2013b. INSTICC.

Patrik Goorts, Steven Maesen, Dimitri Scarlino, and Philippe Bekaert. Bringing 3D Vision to the Web: Acquiring Motion Parallax using Commodity Cameras and WebGL. In *Proceedings of the International Conference on 3D Imaging (IC3D 2013)*, pages 1–6, Liege, Belgium, 2013c. IEEE.

Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. GPU-based View Interpolation for Smooth Camera Transitions in Soccer. In *GPU technology Conference (GTC 2013)*, San Jose, CA, USA, 2013d.

Patrik Goorts, Steven Maesen, Maarten Dumont, Sammy Rogmans, and Philippe Bekaert. Free Viewpoint Video for Soccer using Histogram-Based Validity Maps in Plane Sweeping. In *Proceedings of the Ninth International Conference on Computer Vision Theory and Applications (VISAPP)*, Lisbon, Portugal, 2014a. INSTICC.

Patrik Goorts, Steven Maesen, Yunjun Liu, Maarten Dumont, Philippe Bekaert, and Gauthier Lafruit. Self-Calibration of Large Scale Camera Networks. In *Proceedings of the 11th International Conference on Signal Processing and Multimedia Applications (SIGMAP 2014)*, pages 1–10, Vienna, Austria, 2014b.

Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The Lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–54. ACM, 1996.

Oliver Grau and Julien Pansiot. Motion and Velocity Estimation of Rolling Shutter Cameras. In *Proceedings of the 9th European Conference on Visual Media Production (CVMP)*, pages 94–98, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1311-7.

Oliver Grau and Vinoba Vinayagamoorthy. Stereoscopic 3D Sports Content Without Stereo Rigs. *SMPTE Motion Imaging Journal*, 119(7):51–55, 2010.

Oliver Grau, Michael Prior-Jones, and Graham Thomas. 3d Modelling and Rendering of Studio and Sport Scenes for TV Applications. In *Proceedings of the 6th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, Montreux, Switzerland, 2005.

Oliver Grau, Adrian Hilton, Joe Kilner, Gregor Miller, Tim Sargeant, and Jonathan Starck. A free-viewpoint Video System for Visualization of Sport Scenes. *SMPTE motion imaging journal*, 116(5-6):213–219, 2007a.

Oliver Grau, Graham A Thomas, Adrian Hilton, Joe Kilner, and Jonathan Starck. A Robust Free-viewpoint Video System for Sport Scenes. In *Proceedings of the 3DTV Conference*, pages 1–4, Kos Island, Greece, 2007b. IEEE.

Simon Green. Image Processing Tricks in OpenGL. Presentation at GDC, 2005.

Sander Grognard, Patrik Goorts, Johannes Taelman, and Philippe Bekaert. Autofocus van Cameralenzen. Bachelor's thesis, Hasselt University, Hasselt, Belgium, 2012.

Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2003.

Kunihiko Hayashi and Hideo Saito. Synthesizing Free-Viewpoing Images from Multiple View Videos in Soccer Stadium. In *Proceedings of the IEEE International Conference on Computer Graphics, Imaging and Visualisation*, pages 220–225, Sydney, Australia, 2006.

Jean-Bernard Hayet, Justus H Piater, and Jacques G Verly. Fast 2D Model-to-image Registration using Vanishing Points for Sports Video Analysis. In *Proceedings of the IEEE International Conference on Image Processing(ICIP)*, pages 417–420, Genoa, Italy, 2005.

Adrian Hilton, Jean-Yves Guillemaut, Joe Kilner, Oliver Grau, and Graham Thomas. Free-Viewpoint Video for TV Sport Production. *Image and Geometry Processing for 3-D Cinematography*, 5:77–106, 2010.

Adrian Hilton, Jean-Yves Guillemaut, Joe Kilner, Oliver Grau, and Graham Thomas. 3D-TV Production From Conventional Cameras for Sports Broadcast. *IEEE Transactions on Broadcasting*, 57(2):462–476, 2011.

Keigo Hirakawa and Thomas W. Parks. Adaptive Homogeneity-directed Demosaicing Algorithm. *IEEE Transactions on Image Processing*, 14(3):360–369, 2005.

Anna-Karin Hulth and Erik Melakari. IBR Camera System for Live TV Sport Productions. Master's thesis, Linkoping University, 2005.

Naho Inamoto and Hideo Saito. Fly Through View Video Generation of Soccer Scene. *Entertainment Computing*, 112:109–116, 2002a.

Naho Inamoto and Hideo Saito. Intermediate View Generation of Soccer Scene from Multiple Videos. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 2, pages 713–716, Quebec, Canada, 2002b. IEEE.

Naho Inamoto and Hideo Saito. Fly-through Viewpoint Video System for Multi-view Soccer Movie using Viewpoint Interpolation. In *Proceedings of the IEEE International Conference on Visual Communications and Image Processing*, pages 1143–1151, Benalmadena, Spain, 2003a.

Naho Inamoto and Hideo Saito. Immersive Observation of Virtualized Soccer Match at Real Stadium Model. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, page 188, Tokyo, Japan, 2003b. IEEE.

Naho Inamoto and Hideo Saito. Arbitrary Viewpoint Observation for Soccer Match Video. In *Proceedings of the European Conference on Visual Media Production (CVMP)*, pages 21–30, London, UK, 2004a. Citeseer.

Naho Inamoto and Hideo Saito. Free Viewpoint Video Synthesis and Presentation of Sporting Events for Mixed Reality Entertainment. In *Proceedings of the ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 42–50, Singapore, 2004b. ACM.

Naho Inamoto and Hideo Saito. Free Viewpoint Video Synthesis and Presentation from Multiple Sporting Videos. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 42–50, Amsterdam, The Netherlands, 2005. IEEE.

Naho Inamoto and Hideo Saito. Free Viewpoint Video Synthesis and Presentation from Multiple Sporting Videos. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 90(2):40–49, 2007a.

Naho Inamoto and Hideo Saito. Virtual Viewpoint Replay for a Soccer Match by View Interpolation from Multiple Cameras. *IEEE Transactions on Multimedia*, 9(6):1155–1166, 2007b.

Sachiko Iwase and Hideo Saito. Tracking Soccer Player Using Multiple Views. In *IAPR Workshop on Machine Vision Applications (MVA)*, pages 102–105, Nara, Japan, 2002.

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. KinectFusion: Real-time 3D Reconstruction and Interaction using a Moving Depth Camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, Santa Barbara, CA, USA, 2011. ACM.

Lode Jorissen, Patrik Goorts, Bram Bex, Nick Michiels, Sammy Rogmans, Philippe Bekaert, and Gauthier Lafruit. A qualitative comparison of mpeg view synthesis and light field rendering. In *Proceedings of the Converence on 3D TV (3DTV 2014)*, pages 1–4, 2014.

Yoshinari Kameda, Takayoshi Koyama, Yasuhiro Mukaigawa, Fumito Yoshikawa, and Yuichi Ohta. Free Viewpoint Browsing of Live Soccer Games. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 747–750, Taipei, Taiwan, 2004. IEEE.

Takeo Kanade. Eye Vision at the Super Bowl, 2001. `http://www.ri.cmu.edu/events/sb35/tksuperbowl.html`.

Takeo Kanade, Peter Rander, and P J Narayanan. Virtualized Reality: Constructing Virtual Worlds from Real Scenes. *IEEE Multimedia, Immersive Telepresence*, 4(1):34–47, January 1997.

Sing Bing Kang, Yin Li, Xin Tong, and Heung-Yeung Shum. Image-based Rendering. *Foundations and Trends in Computer Graphics and Vision*, 2(3):173–258, 2006.

Joe Kilner, Jonathan Starck, and Adrian Hilton. A Comparative Study of Free Viewpoint Video Techniques for Sports Events. In *Proceedings of the European Conference on Visual Media Production (CVMP)*, pages 87–96, London, UK, 2006. IET.

Joe Kilner, Jonathan Starck, Adrian Hilton, and Oliver Grau. Dual-mode Deformable Models for Free-viewpoint Video of Sports Events. In *Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM)*, pages 177–184, Montreal, Quebec, Canada, 2007. IEEE.

Joe Kilner, Jonathan Starck, Jean-Yves Guillemaut, and Adrian Hilton. Objective Quality Assessment in Free-viewpoint Video Production. *Signal Processing: Image Communication*, 24(1):3–16, 2009.

Kenji Kimura and Hideo Saito. Player Viewpoint Video Synthesis using Multiple Cameras. In *Proceedings of the The 2nd IEE European Conference on Visual Media Production (CVMP)*, pages 112–121, London, UK, 2005a.

Kenji Kimura and Hideo Saito. Video Synthesis at Tennis Player Viewpoint from Multiple View Videos. In *Proceedings of the IEEE International Conference on Virtual Reality (VR)*, pages 281–282, Bonn, Germany, 2005b. IEEE.

Martin Koppel, Xi Wang, Dimitar Doshkov, Thomas Wiegand, and Patrick Ndjiki-Nya. Depth Image-based Rendering with Spatio-temporally Consistent Texture Synthesis for 3-D video With Global Motion. In *Proceedings of the 19th IEEE International Conference on Image Processing (ICIP)*, pages 2713–2716, Orlando, FL, USA, 2012. IEEE.

Takayoshi Koyama, Itaru Kitahara, and Yuichi Ohta. Live Mixed-reality 3d Video in Soccer Stadium. In *Proceedings of The Second IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 178–186, Tokyo, Japan, 2003. IEEE.

Alexander Ladikos, Selim Benhimane, and Nassir Navab. Efficient Visual Hull Computation for Real-time 3D Reconstruction using CUDA. In *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, Anchorage, AK, USA, 2008.

Johann Heinrich Lambert and Ernst Anding. *Lamberts Photometrie : Photometria, sive De mensura et gradibus luminus, colorum et umbrae*. W. Engelmann, Leipzig, 1760.

C.A. Laroche and M.A. Prescott. Apparatus and Method for Adaptively Interpolating a Full Color Image Utilizing Chrominance Gradients, December 1994. US Patent 5373322.

Aldo Laurentini. The Visual Hull Concept for Silhouette-based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

Aaron E. Lefohn, Joe Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Glift: Generic, Efficient, Random-access GPU Data Structures. *ACM Transactions on Graphics (TOG)*, 25(1):60–99, 2006.

Marc Levoy and Pat Hanrahan. Light Field Rendering. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 31–42, New Orleans, Louisiana, USA, 1996. ACM.

Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, et al. The Digital Michelangelo Project: 3D Scanning of Large Statues. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 131–144, New Orleans, Louisiana, USA, 2000.

Haopeng Li and Markus Flierl. SIFT-based Modeling and Coding of Background Scenes for Multiview Soccer Video. In *Proceedings of the 19th IEEE International Conference on Image Processing (ICIP)*, pages 1221–1224, Orlando, FL, USA, 2012. IEEE.

Qihe Li and Yupin Luo. Automatic Camera Calibration for Images of Soccer Match. In *Proceedings of the International Conference on Computational Intelligence*, pages 482–485, Istanbul, Turkey, 2004.

Liberovision. Liberovision, 2013a. www.liberovision.com.

Liberovision. Liberovision, Results of the View Interpolation, 2013b. http://player.vimeo.com/video/58623456.

Hong-Wei Lin, Chiew-Lan Tai, and Guo-Jin Wang. A Mesh Reconstruction Algorithm Driven by an Intrinsic Property of a Point Cloud. *Computer-Aided Design*, 36(1):1–9, 2004.

David Lowe. Distinctive Image Features from Scale-invariant Keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Jiangbo Lu, Sammy Rogmans, Gauthier Lafruit, and Francky Catthoor. High-speed Dense Stereo via Directional Center-biased Support Windows on Programmable Graphics Hardware. In *Proceedings of 3DTV-CON: The True Vision Capture, Transmission and Display of 3D Video*, Kos, Greece, May 2007.

Yuancheng Luo and Ramani Duraiswami. Canny Edge Detection on NVIDIA CUDA. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–8, Anchorage, AK, June 2008.

Steven Maesen, Patrik Goorts, Lode Vanacken, Sofie Notelaers, and Tom De Weyer. Look Mother, Virtual Puzzling without Buttons! In *2011 IEEE Symposium on 3D User Interfaces (3DUI 2011)*, pages 139–140, Singapore, March 2011. IEEE. ISBN 978-1-4577-0037-8.

Steven Maesen, Patrik Goorts, and Philippe Bekaert. Scalable Optical Tracking for Navigating Large Virtual Environments using Spatially Encoded Markers. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 101–110, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2379-6.

Henrique S. Malvar, Li-wei He, and Ross Cutler. High-quality Linear Interpolation for Demosaicing of Bayer-patterned Color Images. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 485–488, Montreal, Canada, 2004. IEEE.

William R Mark, R Steven Glanville, Kurt Akeley, and Mark J Kilgard. Cg: a System for Programming Graphics Hardware in a C-like Language. *ACM Transactions on Graphics (TOG)*, 22(3):896–907, 2003.

Kshitij Marwah, Gordon Wetzstein, Yosuke Bando, and Ramesh Raskar. Compressive Light Field Photography using Overcomplete Dictionaries and Optimized Projections. *ACM Transactions on Graphics*, 32(4):1–11, 2013.

Wojciech Matusik, Chris Buehler, Ramesh Raskar, Leonard McMillan, and Steven Gortler. Image-based Visual Hulls. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 369–374, New Orleans, Louisiana, USA, 2000. ACM.

Michael D. McCool. Data-parallel Programming on the Cell BE and the GPU using the RapidMind Development Platform. In *Proceedings of the GSPx Multicore Applications Conference*, 2006.

Michael D. McCool and Stefanus Du Toit. *Metaprogramming GPUs with Sh*. AK Peters, 2004.

Michael D. McCool, Zheng Qin, and Tiberiu S. Popa. Shader Metaprogramming. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 57–68, Saarbrucken, Germany, 2002. Eurographics Association.

Patrick McCormick, Jeff Inman, James Ahrens, Jamaludin Mohd-Yusof, Greg Roth, and Sharen Cummins. Scout: a Data-parallel Programming Language for Graphics Processors. *Parallel Computing*, 33(10):647–662, 2007.

Morgan McGuire. Efficient, High-quality Bayer Demosaic Filtering on GPUs. *Journal of Graphics, GPU, and Game Tools*, 13(4):1–16, 2008.

Leonard McMillan. *An Image-based Approach to Three-dimensional Computer Graphics*. PhD thesis, University of North Carolina, 1997.

Leonard McMillan and Gary Bishop. Plenoptic Modeling: An Image-based Rendering System. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 39–46, Los Angeles, CA, USA, 1995. ACM.

Saied Moezzi, Li-Cheng Tai, and Philippe Gerard. Virtual View Generation for 3d Digital Video. *IEEE multimedia*, 4(1):18–26, 1997.

Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, Edmonton, Alberta, Canada, 2002.

Patrick Ndjiki-Nya, Martin Koppel, Dimitar Doshkov, Haricharan Lakshman, Philipp Merkle, Karsten Muller, and Thomas Wiegand. Depth Image-based Rendering with Advanced Texture Synthesis for 3-D Video. *IEEE Transactions on Multimedia*, 13(3):453–465, 2011.

Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon.

KinectFusion: Real-time Dense Surface Mapping and Tracking. In *Proceedings of the 10th IEEE international symposium on Mixed and augmented reality (ISMAR)*, pages 127–136, Basel, Switserland, 2011. IEEE.

Ren Ng, Marc Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz, and Pat Hanrahan. Light Field Photography with a Hand-held Plenoptic Camera. Computer science technical report cstr, Stanford University, 2005.

Sofie Notelaers, Tom De Weyer, Patrik Goorts, Steven Maesen, Lode Vanacken, Karin Coninx, and Philippe Bekaert. HeatMeUp: a 3DUI Serious Game to Explore Collaborative Wayfinding. In *2012 IEEE Symposium on 3D User Interfaces (3DUI 2012)*, pages 177–178, Orange County, California, March 2012. IEEE. ISBN 978-1-4673-1204-2.

Vincent Nozick, Sylvain Michelin, and Didier Arquès. Real-time Plane-Sweep with Local Strategy. In *Journal of WSCG*, volume 14, pages 121–128, Bory, Czech Republic, jan 2006.

NVIDIA. *CUDA Best Practices Guide*, 2014a. `http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html`.

NVIDIA. *CUDA Programming Manual*, 2014b. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

NVIDIA. *CUFFT Programming Manual*, 2014c. `http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf`.

Yuichi Ohta, Itaru Kitahara, Yoshinari Kameda, Hiroyuki Ishikawa, and Takayoshi Koyama. Live 3D Video in Soccer Stadium. *International Journal of Computer Vision*, 75(1):173–187, 2007.

Manuel M Oliveira, Gary Bishop, and David McAllister. Relief Texture Mapping. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 359–368, New Orleans, Louisiana, USA, 2000. ACM.

Nicolas Papadakis and Vicent Caselles. Multi-label Depth Estimation for Graph Cuts Stereo Problems. *Journal of Mathematical Imaging and Vision*, 38(1):70–82, 2010.

Rick Parent. *Computer Animation, Algorithms and Techniques*. Morgan Kaufmann, 2002.

Lai-Man Po, Shihang Zhang, Xuyuan Xu, and Yuesheng Zhu. A New Multidirectional Extrapolation Hole-filling Method for Depth-image-based Rendering. In *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP)*, pages 2589–2592, Brussels, Belgium, 2011. IEEE.

Victor Podlozhnyuk. FFT-based 2D Convolution. Technical report, NVIDIA, June 2007a.

Victor Podlozhnyuk. Image Convolution with CUDA. Technical report, NVIDIA, June 2007b.

Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline Techniques*. Springer, 2002.

Red Bee Corporation. 3D Graphics Systems, 2001. `http://www.redbeemedia.com/piero/piero`.

Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A Dodgson. Real-time Spatiotemporal Stereo Matching using the Dual-cross-bilateral Grid. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 510–523, Crete, Greece, 2010. Springer.

Jerel D Robison. Genlock Frequency Generator, October 1992. US Patent 5155595.

Tomas Rodriguez, Ian Reid, Radu Horaud, Navneet Dalal, and Marcelo Goetz. Image Interpolation for Virtual Sports Scenarios. *Machine Vision and Applications*, 16(4):236–245, 2005.

Sammy Rogmans, Maarten Dumont, Tom Cuypers, Gauthier Lafruit, and Philippe Bekaert. Complexity Reduction of Real-Time Depth Scanning on Graphics Hardware. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 547–550, Lisbon, Portugal, February 2009a.

Sammy Rogmans, Jiangbo Lu, Philippe Bekaert, and Gauthier Lafruit. Real-Time Stereo-Based View Synthesis Algorithms: A Unified Framework and Evaluation on Commodity GPUs. *Signal Processing: Image Communication*, 24(1-2):49–64, January 2009b. Special issue on advances in three-dimensional television and video.

Hideo Saito, Naho Inamoto, and Sachiko Iwase. Sports Scene Analysis and Visualization from Multiple-view Video. In *Proceedings of the IEEE International Conference on Multimedia and Expo(ICME)*, pages 1395–1398, Taipei, Taiwan, 2004. IEEE.

Daniel Scharstein. Matching Images by Comparing their Gradient Fields. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition A: computer Vision and Image Processing*, volume 1, pages 572–575, Jerusalem, Israel, 1994. IEEE.

Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-frame Stereo Correspondence Algorithms. *International journal of computer vision*, 47(1):7–42, 2002.

Daniel Scharstein and Richard Szeliski. High-accuracy Stereo Depth Maps using Structured Light. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 195–202, Madison, Wisconsin, USA, 2003. IEEE.

Steven M Seitz and Charles R Dyer. View Morphing: Uniquely Predicting Scene Appearance from Basis Images. In *Proceedings of the Image Understanding Workshop*, pages 881–887, New Orleans, LA, USA, 1997.

Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A Comparison and Evaluation of Multi-view Stereo Reconstruction Algorithms. In *Proceedings of the IEEE Computer Society Conference on Computer vision and pattern recognition (CVPR)*, volume 1, pages 519–528, New York, NY, USA, 2006. IEEE.

Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered Depth Images. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 231–242, New York, USA, 1998. ACM.

Dave Shreiner. *OpenGL Programming Guide: the Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, 2009.

Harry Shum and Sing B Kang. Review of Image-based Rendering Techniques. In *Proceedings of SPIE. Visual Communications and Image Processing*, pages 2–13, Perth, Australia, 2000. International Society for Optics and Photonics.

Marten Sjostrom, Peter Hardling, Linda S Karlsson, and Roger Olsson. Improved Depth-image-based Rendering Algorithm. In *Proceedings of the 3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, Antalya, Turkey, 2011. IEEE.

Greg Slabaugh, Ron Schafer, and Mat Hans. Image-based Photo Hulls. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission*, pages 704–709, Chapel Hill, NC, USA, 2002. IEEE.

Sony. HDC-1500: 3G Ready HD Portable Camera, 2013. URL `http://www.sony.co.uk/pro/product/broadcast-products-system-cameras-hd-system-cameras/hdc-1500/overview`.

Olgierd Stankiewicz, Krzysztof Wegner, Masayuki Tanimoto, and Marek Domanski. Enhanced Depth Estimation Reference Software (DERS) for Free-viewpoint Television, ISO/IEC JTC1/SC29/WG11 M31518. Technical report, MPEG, Geneva, October 2013.

Jonathan Starck and Adrian Hilton. Model-based multiple view reconstruction of people. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, pages 915–922, Nice, France, 2003. IEEE.

John E Stone, David Gohara, and Guochun Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in science and engineering*, 12(3): 66–73, 2010.

Yu Sun, Stefan Duthaler, and Bradley J Nelson. Autofocusing in Computer Microscopy: Selecting the Optimal Focus Algorithm. *Microscopy research and technique*, 65(3):139–149, 2004.

Tomáš Svoboda, Daniel Martinec, and Tomáš Pajdla. A Convenient Multicamera Self-calibration for Virtual Environments. *PRESENCE: teleoperators and virtual environments*, 14(4):407–422, 2005.

Flávio Szenberg, Paulo Cezar Pinto Carvalho, and Marcelo Gattass. Automatic Camera Calibration for Image Sequences of a Football Match. In *Advances in Pattern Recognition (ICAPR)*, pages 303–312. Springer, Rio de Janeiro, Brazil, 2001.

Masayuki Tanimoto, Toshiaki Fujii, and Kazuyoshi Suzuki. View Synthesis Algorithm in View Synthesis Reference Software 2.0 (VSRS 2.0), ISO/IEC JTC1/SC29/WG11 M16090. Technical report, MPEG, Lausanne, Switzerland, February 2009.

David Tarditi, Sidd Puri, and Jose Oglesby. Accelerator: Simplified Programming of Graphics Processing Units for General-purpose Uses via Data-parallelism. Technical report, Microsoft Research, 2005.

Graham Thomas. Real-time Camera Tracking using Sports Pitch Markings. *Journal of Real-Time Image Processing*, 2(2-3):117–132, 2007.

Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle Adjustment - a Modern Synthesis. In *Vision algorithms: theory and practice*, volume 1883, pages 298–372. Springer, 2000.

UEFA. Commercial Regulations for the European Qualifying Matches for UEFA EURO 2016 and the 2018 FIFA World Cup. Technical report, UEFA, 2013. `http://www.uefa.org`.

Scorpion Vision. CMOS Sensors: What's the Difference between Global and Rolling Shutters?, 2013. URL `http://scorpionvision.co.uk/`.

Vizrt. Vizrt, 2013. URL `http://www.vizrt.com/products/viz_libero/`.

Andy Wachowski and Lana Wachowski. The matrix, 1999. Warner Brothers Pictures, United States.

Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nister. High-quality Real-time Stereo using Adaptive Cost Aggregation and Dynamic Programming. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 798–805, Chapel Hill, North Carolina, USA, 2006. IEEE.

Liang Wang, Hailin Jin, Ruigang Yang, and Minglun Gong. Stereoscopic Inpainting: Joint Color and Depth Completion from Stereo Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Anchorage, Alaska, USA, 2008. IEEE.

Colin Ware and Steven Osborne. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, pages 175–183, New York, NY, USA, 1990. ACM.

Michael Waschbüsch, Stephan Würmlin, and Markus Gross. 3D Video Billboard Clouds. *Computer Graphics Forum*, 26(3):561–569, 2007.

Krzysztof Wegner, Olgierd Stankiewicz, Masayuki Tanimoto, and Marek Domanski. Enhanced View Synthesis Reference Software (VSRS) for Free-viewpoint Television, ISO/IEC JTC1/SC29/WG11 M31520. Technical report, MPEG, Geneva, October 2013.

Juyang Weng, Thomas S. Huang, and Narendra Ahuja. *Motion and Structure from Image Sequences*. Springer Publishing Company, Incorporated, 1 edition, 2012.

Rob Williams. Intel's core i7-980x extreme edition – ready for sick scores?, 2010. URL `http://techgage.com/article/intels_core_i7-980x_extreme_edition_-_ready_for_sick_scores/8`.

Bill Wulf and Sally McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH computer architecture news*, 23(1):20–24, 1995.

Jason Yang and Lee Howes. Advanced DirectX 11 technology: DirectCompute by Example. In *Game Developers Conference Europe*, Cologne, Germany, 2010.

Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao, and David Nister. Real-time Global Stereo Matching Using Hierarchical Belief Propagation. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 6, pages 989–998, Edinburgh, UK, 2006.

Ruigang Yang and Greg Welch. Fast Image Segmentation and Smoothing using Commodity Graphics Hardware. *Journal of graphics tools*, 7(4):91–100, 2002.

Ruigang Yang, Greg Welch, and Gary Bishop. Real-Time Consensus-Based Scene Reconstruction Using Commodity Graphics Hardware. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, pages 225–234, Beijing, China, 2002. Wiley Online Library.

Seung-Uk Yoon, Eun-Kyung Lee, Sung-Yeol Kim, and Yo-Sung Ho. A Framework for Multi-view Video Coding using Layered Depth Images. In *Advances in Multimedia Information Processing (PCM)*, pages 431–442. Springer, 2005.

Xinguo Yu, Nianjuan Jiang, Loong-Fah Cheong, Hon Wai Leong, and Xin Yan. Automatic Camera Calibration of Broadcast Tennis Video with Applications to 3D Virtual Content Insertion and Ball Detection and Tracking. *Computer Vision and Image Understanding*, 113(5):643–652, 2009.

Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based Local Stereo Matching using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009a.

Ke Zhang, Jiangbo Lu, Gauthier Lafruit, Rudy Lauwereins, and Luc Van Gool. Real-time Accurate Stereo with Bitwise Fast Voting on CUDA. In *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 794–800, Kyoto, Japan, 2009b. IEEE.

Li Zhang, Brian Curless, and Steven M Seitz. Rapid Shape Acquisition using Color Structured Light and Multi-pass Dynamic Programming. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission*, pages 24–36, Chapel Hill, NC, USA, 2002. IEEE.

Yueyi Zhang, Zhiwei Xiong, Zhe Yang, and Feng Wu. Real-time Scalable Depth Sensing with Hybrid Structured Light Illumination. *IEEE Transactions on Image Processing*, 23 (1), 2014.

Zhengyou Zhang. Microsoft Kinect Sensor and its Effect. *IEEE Multimedia*, 19(2):4–10, 2012.

C Lawrence Zitnick and Sing Bing Kang. Stereo for Image-based Rendering using Image Over-segmentation. *International Journal of Computer Vision*, 75(1):49–65, 2007.

C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality Video View Interpolation using a Layered Representation. *ACM Transactions on Graphics (TOG)*, 23(3):600–608, 2004.