AN END-TO-END SYSTEM FOR FREE VIEWPOINT VIDEO FOR SMOOTH CAMERA TRANSITIONS

Patrik Goorts, Maarten Dumont, Sammy Rogmans, Philippe Bekaert

Hasselt University - tUL - iMinds
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium

ABSTRACT

In this paper, we present an end-to-end system for free viewpoint video for smooth camera transitions in sport scenes. Our system consists of a network of static computer vision cameras, a storage infrastructure and an interpolation rendering module, connected with a 10 Gigabit Ethernet network. The user of the system requests a viewpath for the virtual camera and the rendering module then generates the images using a depth-aware plane sweep approach. First, the foreground and background are separated and rendered independently. The foreground is rendered using a plane-sweep approach and the obtained depth map is split up in groups of players. Each group is assigned a global depth, which is used in a second plane sweep to restrict the depth. This will reduce artifacts, such as extra limbs and ghost players. The algorithm is demonstrated on actual soccer recordings. The system is fully automatic and can work in near real-time, thus providing virtual images of high quality in a fast manner.

Index Terms— Free Viewpoint Video, Interpolation, Plane Sweep, CUDA, GPU, Soccer, Broadcasting

1. INTRODUCTION

Nowadays, broadcasting of entertainment events, such as soccer games, require high quality video processing in real-time. Streaming of raw camera images is no longer sufficient. Different stages of processing are required to provide high quality broadcasting that is pleasant and entertaining for the final viewer. To provide a part of the high quality real-time pipeline, we present an end-to-end system for view interpolation to acquire smooth camera transitions in soccer games. The user of the system, which can be the end user or a camera operator, can choose a view path for a virtual camera. This way, transition from one camera position to another can be done without sudden hops, which can confuse the spectator. Using view interpolation, a virtual camera can move between physical camera positions, which allows the spectators to keep track of their global viewing position in the scene. Alternatively, the action can be freezed and the virtual camera can be moved to allow novel ways of viewing the soccer game. To achieve this setup, a network of cameras with fixed locations is placed around the pitch, capturing the game from different angles. These captured image sequences are subsequently streamed to a storage infrastructure, where they can be requested by the rendering module. This experimental storage server is designed by EVS [4] to capture and retrieve multimedia data. The rendering module can request any stream of any camera at any time from the storage infrastructure to use in the interpolation. The rendering module acquires a view path for a virtual camera from the user for a certain time and generates a video stream for the virtual camera. This generated video stream will be pushed to the storage infrastructure, where it can be retrieved for broadcasting. The rendering itself is near real-time and fully automatic, eliminating the intervention of manual input. To provide flexible and reliable communication between cameras, storage servers and rendering modules, standard 10 Gigabit Ethernet connections are used.

The view interpolation is based on Image-Based Rendering (IBR), where no geometry is required or generated; all required information is extracted from the camera images. In our system, the foreground is first extracted and rendered independently from the background. The depth of the dynamic objects, i.e. the foreground, is first determined by a plane sweep approach. Next, we determine the general depth of every connected group of pixels, which is used in a second plane sweep to limit the used depth values. This way, we can effectively eliminate ghosting artifacts, such as third legs, while respecting local depth values. Indeed, these artifacts tend to have very different depth values, which can be eliminated by limiting the used depth values.

All rendering phases are implemented using a combination of NVIDIA's Cg shader language and CUDA, allowing the exploitation of the advantages of both frameworks, while impeding the weaknesses.

We demonstrated our system by making recordings of an actual soccer game and storing them on the storage infrastructure. Generated results proved to be of high quality and provided clear camera transitions from one viewpoint to the other.

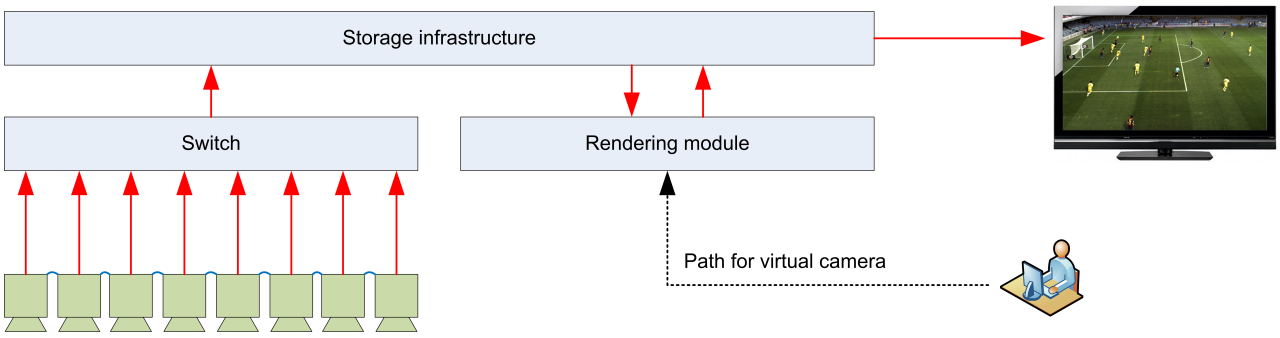

Fig. 1. Overview of the system. The setup consists of a network of cameras, a storage infrastructure and a rendering module, connected with 10 Gigabit Ethernet. The cameras store the retrieved data on the storage device, where they can be retrieved by the rendering module. The user can input a view path in the rendering module and the final rendered images are also stored on the storage infrastructure.

2. RELATED WORK

There are typically two approaches to generate novel viewpoints starting from existing viewpoints, Image-Based Rendering (IBR) and geometry-based rendering.

For geometry-based rendering, a collection of 3D models is required. These can be acquired by using, for example, visual hull [17, 18], photo hull [14] or space carving [22].

Image-based rendering, on the other hand, does not use geometry in its rendering process. Instead, only the information from the input images is used to generate an interpolated image. Stereo matching is a common method of image-based rendering [21, 26]. Another method, where our method is based on, is plane sweeping, where the space before the virtual camera is subdivided in depth planes. Plane sweeping finds the depth with the best color matching for every pixel in the virtual camera [24, 5]; depth and color information are generated simultaneously. Due to numerous possible artifacts using the standard algorithm, different improvements have been proposed, such as segmentation and depth selection [20].

Some attempts have been made to create systems to interpolate outdoor sport scenes. One of the early systems is the Eye vision system [2], described by [13], where multiple motorized cameras are placed around the scene. No real interpolation is done; the cameras switch between the images and hops can be perceived. The results are still plausible, because the cameras are close to each other. Yet, the system is expensive and requires a dense camera setup. The iView system [9] from Red Bee Media [1] generates billboards or visual hull using a narrow baseline. This method reports some artifacts, such as missing limbs, ghosting, etc. The method of Hilton et al.[11] reconstructs a layered 3D proxy representation to define the geometry of players, followed by a view-dependent refinement step using multiple views. Hayashi and Saito [10, 12] and Ohta et al.[19] propose a method that uses billboards to represent dynamic objects. A more advanced method is used by Germann et al. [6] that estimate the 2D pose using a database of silhouettes. Using the poses, billboards can be refined to generate more pleasant results. However, manual intervention is still required.

Fig. 2. Example of the camera line setup. Cameras are placed approximately one meter apart from each other. The distance to the field is about 20 meters.

3. SYSTEM OVERVIEW

The system consists of a computer vision camera network, a storage infrastructure and a rendering module. An overview can be seen in Figure 1.

Our system can handle two kinds of camera setups: linear and all around (see Figure 2). When using a linear camera setup, cameras are places along one side of the scene and the focus is forward. This way, interpolation from one side of the field to the other side is possible, for example to follow the action. The all around setup places the cameras in a circle around the scene, where the focus is at one spot on the pitch. Here, when there is action in the neighborhood of this spot, interpolation of different angles of that action can be shown. Nowadays, only one side of the pitch is covered by cameras to avoid confusion of the spectator. When using an all around setup, interpolation from one side to the other is possible, thus allowing more rich broadcasting opportunities

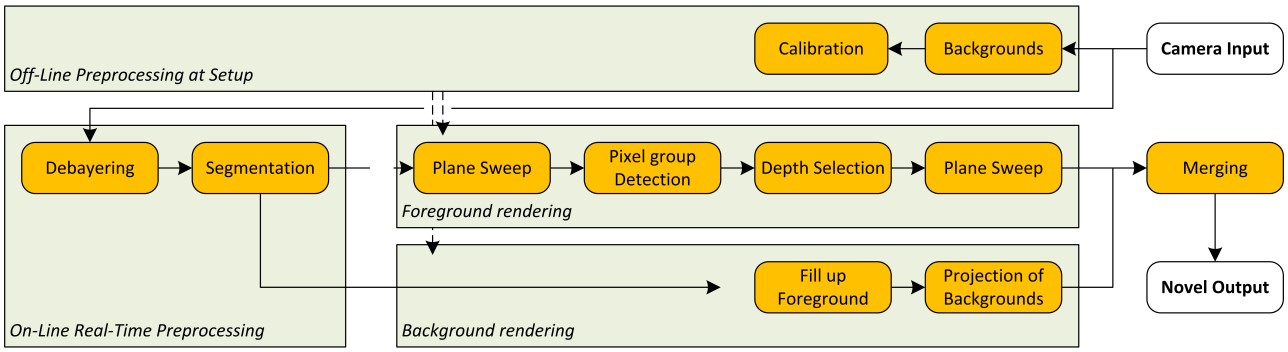

Fig. 3. Overview of the rendering. The rendering consists of a one time preprocessing step and a rendering step. The rendering step, furthermore, is divided in a background and foreground rendering to allow high quality results.

without confusing the spectator; the global viewing position stays clear. In both setups, we use fixed cameras to reduce the need of an operator and to ease the calibration process. In our method, we employ a wide baseline setup. This way, the rendering is less trivial, but reduces the hardware cost and eases the process of preparing the setup. To allow a global shutter synchronization, all cameras are attached to a global clock which determines the exact time a frame is taken.

All captured images are transferred to an experimental prototype of multimedia server, developed and provided by EVS [4], using standard 10 Gigabit Ethernet connections using a network switch. The cameras itself are attached to the switch using a 1 Gigabit Ethernet connection. To reduce the storage and network load, all images are transferred in raw format, i.e. before demosaicing. All frames are stored together with a global timestamp to allow easy retrieval of all the frames of a specific time. The infrastructure also allows the storage of the final rendered images to be retrieved later on.

Finally, a rendering module is attached to the storage infrastructure using standard 10 Gigabit Ethernet connections. The rendering module is equipped with an NVIDIA GeForce GTX 680 with 1536 cores running at 1006 MHz. In our setup, the user input is integrated with the rendering module. Due to the central storage of the image data, multiple rendering modules can be attached to the network.

The rendering consists of a preprocessing step and a rendering step. An overview is given in Figure 3.

4. PREPROCESSING STAGE

In the preprocessing stage, all cameras are calibrated to compensate for lens distortions and to determine the positions relative to the pitch. Firstly, pixel correspondences are generated using feature detection. In every camera image C_i, features are detected using SIFT [15], and matched between the different camera pairs using the k-d tree algorithm. These matching pairs are then traced over every camera pair, thus resulting in a multicamera group of matching features. This way, robust correspondences between camera images are generated, which can be fed into the well-known calibration toolbox of Svoboda et al. [23] using a bundle adjustment approach with RANSAC. Here, the correspondences are used to determine intrinsic and extrinsic calibration matrices, which fully define the lens distortion and the position of the cameras.

Furthermore, background images are determined for every input camera. About 30 images are stored over a time period of a few minutes. These images are used to calculate the median value of every pixel for every color channel, which are used as the background color values.

5. RENDERING

In the rendering phase, the images for the virtual camera are generated. The user of the system determines a path for the virtual camera and a specific time in the video sequence. Using this information, the rendering module generates the images of the virtual camera for every position on the virtual viewpath using the images of the input cameras on the corresponding time. For every frame, the required images are retrieved from the storage facility. After retrieval, the images are transferred to the GPU, where highly parallel processing can be performed using traditional vertex and fragment shaders, and the more generic parallel processing architecture CUDA.

To reduce network and storage load, images are stored and retrieved as raw data, i.e. before demosaicing. Therefore, images need to be demosaiced. To perform real-time demosaicing using GPUs, we use the method of Malver et al. [16, 7], where a FIR filtering method is used to perform edge-aware demosaicing. Because FIR filtering is highly parallel [8], the algorithm is easily mapped on GPU.

Next, the foreground-background segmentation is performed to divide the players from the background. Foreground and background are distinguished using a per pixel threshold-based filtering approach using the backgrounds from the preprocess-

ing stage. To provide high-quality segmentation, we use three thresholds, τ_f, τ_b, τ_a, with $\tau_f > \tau_b$:

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

with $s_i = S_i(x, y)$, $c_i = C_i(x, y)$ 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. First, we calculate the difference between the foreground and background pixels and compare them with τ_f and τ_b. When the difference is between τ_f and τ_b, we calculate the angle between the color values and compare with τ_a. We enhance the segmentation using erosion and dilation to reduce errors caused by noise in the input images [25].

The thresholds are chosen such that shadows are considered as background to reduce matching errors and occlusion in the foreground interpolation. The shadows are blended in the background, because they are in essence darker background and small location errors are not noticeable. The foreground objects, on the other hand, are not related to the background and cannot be blended accordingly.

The foreground and background are then processed independently.

6. BACKGROUND RENDERING

After the segmentation, the foreground pixels in the backgrounds are filled up with the color values of the backgrounds of the preprocessing stage. This way, we have an actual background for every input image, where the shadows of the players are present and the foreground pixels are removed. Next, we project the backgrounds on the pitch, blend the color values and reproject the blended values to the virtual camera. Because the camera locations are calibrated relatively to the pitch, we can determine the projected location of the backgrounds. The backgrounds are blended together to acquire good approximations of the shadows of the players as seen from the virtual viewpoint. This way, we avoid the interpolation of shadows, which is sensitive to occlusion and mismatching.

7. FOREGROUND RENDERING

The foreground, i.e. the players, are rendered using a plane sweep approach, where depth and color values are generated simultaneously. When a standard plane sweep approach is used, different ghosting artifacts can occur, such as third legs. Because players in a soccer setting are similar, mismatches are frequent, resulting in low quality depth estimation and rendering. Therefore, we use a depth-aware plane sweep approach. This approach uses three phases. First, depth and color for every output pixel is determined using standard plane sweeping. Next, foreground objects in the output image are detected and a uniform depth for every object is determined. Finally, these uniform depths are used to rerender the foreground without artifacts using another plane sweep approach.

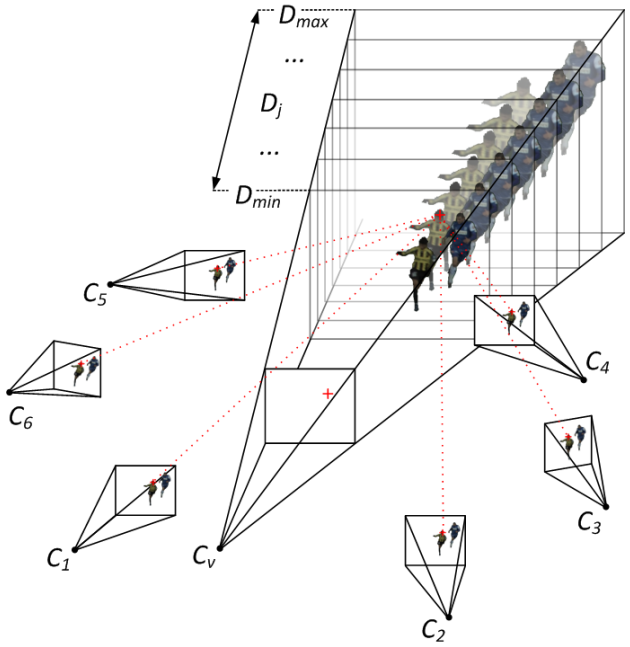

Fig. 4. Principle of plane sweeping. The space before the virtual camera is divided in planes. For each plane, all camera images are projected on it and reprojected on the virtual camera image. The reprojected pixels with the best color consistencies are kept, resulting in a depth and color result.

7.1. Initial Depth Calculation

In the first phase, a global depth map is acquired using plane sweeping. We use a segmentation-aware adaptation of the well-known method of Yang et al. [24]. The space before the virtual camera is divided in planes, each with a depth D. Then, for every depth D_i, all input images are projected on this plane, and reprojected on the virtual camera image (See Figure 4). For every pixel, a metric is used to calculate the difference between the reprojected color values, resulting in an error value ϵ.

$$\epsilon = \sum_{i=1}^{N} \frac{\|\gamma - C_i\|^2}{3N} \text{ with } \gamma = \sum_{i=1}^{N} \frac{C_i}{N} \quad (2)$$

where γ is the average of the reprojected pixels and C_i is the i^{th} input image of total N. When the reprojected color value belongs to the background in the input images, the error value is infinite. Finally, we select for every pixel in the virtual camera image the depth with the lowest error value.

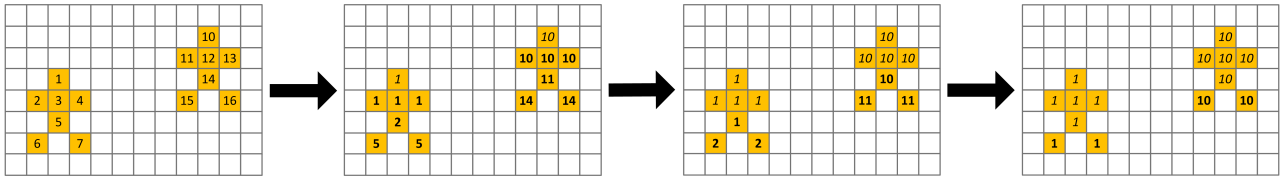

Fig. 5. Detection of connected pixels: a thread is assigned to every pixel. Every iteration, the threads compare the label of its neighboring pixels with its own and stores the lowest label. All connected pixels have the same label if no more changes are made.

When all error values are infinite, the pixel is considered as background. This approach can be efficiently implemented using Cg on graphics hardware, using its projective texturing capabilities.

7.2. Depth Selection

In the second phase, all groups of foreground pixels are assigned one depth value. First, groups of unconnected pixels are determined, which represent the dynamic foreground objects, i.e. (a group of) players. We use a region growing algorithm using CUDA (see Figure 5). Initially, all foreground pixels are assigned a unique label. Then we iteratively process pairs of neighboring foreground pixels, where we assign the lowest label of the two pixels to the pixel with the highest label. Background pixels don't have a label, therefore no assignment will be done when one of the pixels of the considered pair is background. We continue until no more assignments can be done. Eventually, all neighboring foreground pixels will have the same label and every group of foreground pixels will have a unique label. Because the highly local and parallel nature of the algorithm, realization using CUDA is straightforward. Using the interoperability capabilities of both frameworks, no penalty for the switch between Cg and CUDA can be perceived.

Next, we calculate the median depth value of all the foreground pixels with the same label and assign this depth value to all these pixels. This way, we acquire a depth map of the foreground where every group of pixels, i.e. players, have the same filtered depth value. This depth value can be seen as the global depth of a player, or a group of players.

7.3. Final Depth-aware Interpolation

We use these filtered depth values in the next phase, where we perform a second plane sweep rendering, but the depth range is limited on a per pixel base. Only depth values between $D_f - \alpha$ and $D_f + \alpha$ are considered, where D_f is the filtered depth value for a specific pixel and α is a parameter controlling the tolerance. Depths that are not considered automatically get an error value ϵ of infinity. If every error ϵ is infinite, the pixel is considered as background. The final depth of a pixel in the virtual camera image can thus be different than the depth value calculated in the previous phase. This way, excessive errors in the generated depth values are eliminated and ghosting artifacts, such as third legs, are effectively removed. Indeed, these arise due to mismatching in the plane sweep approach. For example, the left leg and the right leg of a player are similar, resulting in a third leg in the virtual camera image. This leg, however, will have a depth that is very different from the rest of the player, and can thus be eliminated using depth-selective plane sweeping. Our method will consider the third leg as background and provide thus higher quality interpolation results.

While many artifacts can be removed using our method, some artifacts still remain. Mismatching can also occur between different players, resulting in ghost players. Therefore, we also consider the depth of the background to determine the correct depth values of the players. If the global depth of a player, determined in the second phase, is very different from the depth of the background, the whole player is considered as an artifact and eliminated. That is, only consider the depths between $D_f - \beta$ and $D_f + \beta$, where β is a parameter controlling the tolerance. Because dynamic objects are typically not flat on the background, β should be relatively high. This kind of mismatches rarely happen due to segmentation constraints, but are very obnoxious. Therefore, filtering of extreme foreground artifacts is required.

7.4. Merging

After the foreground and the background are calculated independently, we merge them using the mask obtained from the foreground interpolation. To generate more pleasant-looking results, we blend the foreground and background colors slightly at the borders of the segments.

8. RESULTS

To test the quality of the interpolation results, we recorded a real soccer game using a linear camera setup. We used 8 Basler avA1600-50gc cameras [3] with 25mm lenses, capturing at a resolution of 1600x1200 at 30 Hz. The parameters were fixed at 0.01 for α and 0.2 for β.

Figure 6 shows the details of the interpolation result. Figure 6a shows the result without depth-aware plane sweeping. Third legs, multiple balls and noise can be perceived. This can

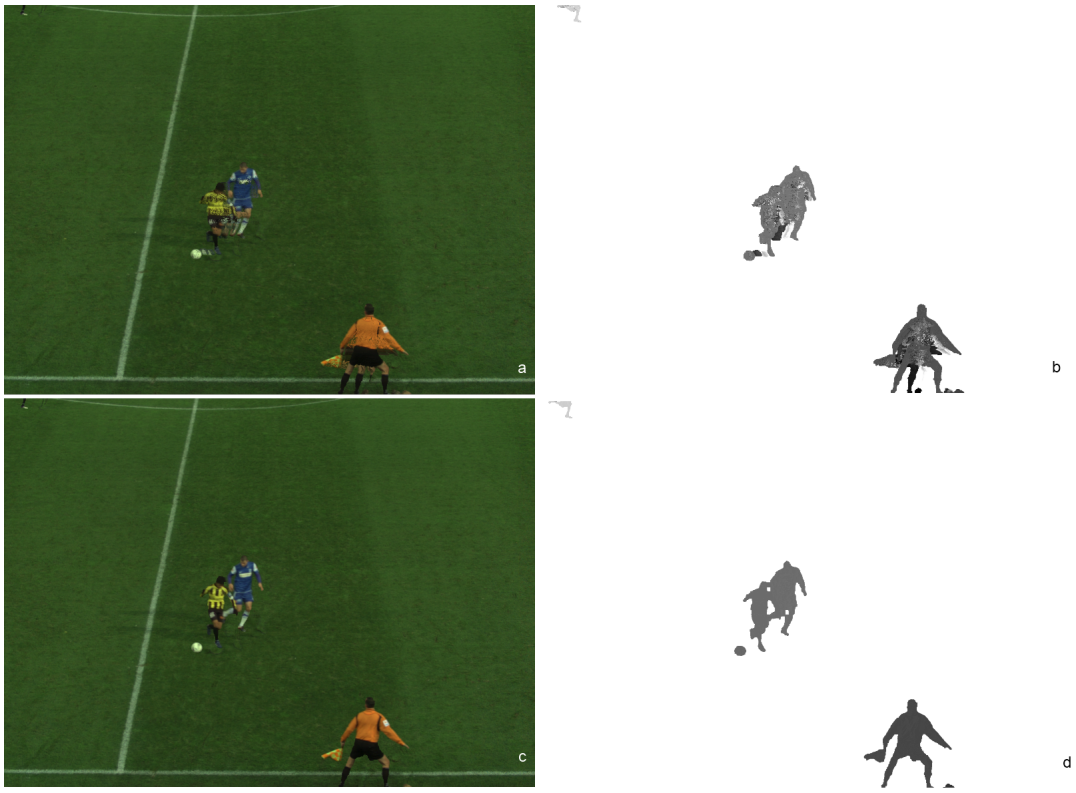

Fig. 6. Results with and without depth-selective plane sweeping. (a) and (b) show the results without depth-selective plane sweeping. Many artifacts can be seen, such as extra limbs. (c) and (d) show the results with depth-selective plane sweeping, where the artifacts are effectively removed.

also be seen in the depth map (Figure 6b), where the depth is noisy. Furthermore, large depth differences between the actual global depth and the artifacts can be noticed.

These artifacts are effectively removed by using depth-aware plane sweeping, as can be seen in Figure 6c and Figure 6d. The extra limb artifacts are removed by only considering a specific range based on the global depth of the group of players. By using a second interpolation step, instead of using the filtered depth map directly, we are able to preserve the subtle depth variations on a dynamic object. The ghosting artifacts of the ball are disconnected from the real ball or the players and thus cannot be removed by this method. However, by also considering the depth of the background and removing depth values that vary too much, these artifacts are also eliminated successfully.

Retrieving and rendering the images is near real-time, running at 15 Hz. The rendering itself runs at 30 Hz.

9. CONCLUSION

We presented an end-to-end system for virtual viewpoint generation for smooth camera transitions. Our system consists of separated capturing, storage and rendering parts, easing the extension and upgrade of individual parts. We demonstrated our setup using a linear camera setup, recording a real soccer game. This data is stored on the storage facility, where retrieving of the input images and storing of the rendered data are demonstrated. For the rendering, we demonstrated an approach with different foreground and background interpolation. The foreground interpolation uses a depth-aware method to reduce artifacts due to mismatches. The rendering proved to be pleasant-looking with significantly less artifacts than more generic approaches. The whole rendering pipeline in implemented using graphics hardware, resulting in high rendering speed. The system is near real-time, running at 15 frames per second, including network transportation and rendering. The rendering itself runs real-time.

Our system demonstrated to be a good method for free viewpoint video for smooth camera transition.

10. ACKNOWLEDGMENTS

Patrik Goorts would like to thank the IWT for its PhD specialization bursary.

We would like to thank EVS, Michael Bastings and Olivier Barnich for developing and providing the experimental prototype for the storage infrastructure.

11. REFERENCES

[1] 3d graphics systems by red bee media, 2001. http://redbeemedia.com/html/live-graphic.html.

[2] Eye vision at the super bowl, 2001. http://www.ri.cmu.edu/events/sb35/tksuperbowl.html.

[3] Basler ava1600-50gc cameras, 2012. http://www.baslerweb.com/products/aviator.html?model=204.

[4] Evs belgium: video equipment and systems, 2012. http://www.evs.tv.

[5] M. Dumont, S. Rogmans, S. Maesen, and P. Bekaert. Optimized two-party video chat with restored eye contact using graphics hardware. *e-Business and Telecommunications*, pages 358–372, 2009.

[6] M. Germann, A. Hornung, R. Keiser, R. Ziegler, S. Würmlin, and M. Gross. Articulated billboards for video-based rendering, 2010.

[7] P. Goorts, S. Rogmans, and P. 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)*. INSTICC, 2012.

[8] Patrik Goorts, Sammy Rogmans, and Philippe Bekaert. Optimal Data Distribution for Versatile Finite Impulse Response Filtering on Next-Generation Graphics Hardware using CUDA . In *Proc. of The Fifteenth Intl. Conference on Parallel and Distributed Systems*, pages 300–307, 2009.

[9] O. Grau, A. Hilton, J. Kilner, G. Miller, T. Sargeant, and J. Starck. A Free-Viewpoint Video System for Visualization of Sport Scenes. *SMPTE motion imaging journal*, 116(5/6):213, 2007.

[10] K. Hayashi and H. Saito. Synthesizing Free-Viewpoing Images from Multiple View Videos in Soccer Stadium. In *Intl. Conf. on Computer Graphics, Imaging and Visualisation*, pages 220–225, 2006.

[11] A. Hilton, J.Y. Guillemaut, J. Kilner, O. Grau, and G. Thomas. 3d-tv production from conventional cameras for sports broadcast. *IEEE Transactions on Broadcasting*, (99):1–1, 2011.

[12] N. Inamoto and H. Saito. Virtual viewpoint replay for a soccer match by view interpolation from multiple cameras. *IEEE Transactions on Multimedia*, 9(6):1155–1166, 2007.

[13] T. Kanade, P. Rander, and PJ Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *Multimedia, IEEE*, 4(1):34–47, 1997.

[14] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *Intl. Journal of Computer Vision*, 38(3):199–218, 2000.

[15] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. journal of computer vision*, 60(2):91–110, 2004.

[16] H.S. Malvar, L. He, and R. Cutler. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 3, pages 485–488. IEEE, 2004.

[17] W. Matusik, C. Buehler, R. Raskar, S.J. Gortler, and L. McMillan. Image-based visual hulls, 2000.

[18] G. Miller, A. Hilton, and J. Starck. Interactive free-viewpoint video, 2005.

[19] Y. Ohta, I. Kitahara, Y. Kameda, H. Ishikawa, and T. Koyama. Live 3D Video in Soccer Stadium. *Intl. Journal of Computer Vision*, 75(1):173–187, 2007.

[20] S. Rogmans, M. Dumont, T. Cuypers, G. Lafruit, and P. Bekaert. Complexity reduction of real-time depth scanning on graphics hardware, 2009.

[21] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms, 2006.

[22] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *Intl. Journal of Computer Vision*, 35(2):151–173, 1999.

[23] T. Svoboda, D. Martinec, and T. Pajdla. A convenient multicamera self-calibration for virtual environments. *Presence: Teleoperators & Virtual Environments*, 14(4):407–422, 2005.

[24] R. Yang, M. Pollefeys, H. Yang, and G. Welch. A unified approach to real-time, multi-resolution, multi-baseline 2d view synthesis and 3d depth estimation using commodity graphics hardware. *Intl. Journal of Image and Graphics*, 4(4):627–651, 2004.

[25] R. Yang and G. Welch. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of graphics tools*, 7(4):91–100, 2002.

[26] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation, 2004.