Virtual View Generation with A Hybrid Camera Array

Engin Tola * Cha Zhang† Qin Cai† Zhengyou Zhang†

Technical Report EPFL/CVLAB2009

Abstract

Virtual view synthesis from an array of cameras has been an essential element of three-dimensional video broadcasting/conferencing. In this paper, we propose a scheme based on a hybrid camera array consisting of four regular video cameras and one time-of-flight depth camera. During rendering, we use the depth image from the depth camera as initialization, and compute a view-dependent scene geometry using constrained plane sweeping from the regular cameras. View-dependent texture mapping is then deployed to render the scene at the desired virtual viewpoint. Experimental results show that the addition of the time-of-flight depth camera greatly improves the rendering quality compared with an array of regular cameras with similar sparsity. In the application of 3D video boardcasting/conferencing, our hybrid camera system demonstrates great potential in reducing the amount of data for compression/streaming while maintaining high rendering quality.

1 Introduction

Three-dimensional video broadcasting/conferencing has attracted a lot of interest recently due to the rapid advances in computation power, network bandwidth, camera array and immersive display technologies. This opens a wide variety of research opportunities including high performance imaging, multi-view video compression and transmission, virtual view synthesis, etc. Take the application of free viewpoint TV [17, 6] (FTV) as an example. To offer an interactive three-dimensional depth impression, the system needs to capture live videos from tens

*École Polytechnique Fédérale de Lausanne, Switzerland - engin.tola@epfl.ch
†Microsoft Research Redmond, USA - {chazhang, qincai, zhang}@microsoft.com

or hundreds of cameras, compress and stream them to the remote users, and synthesize novel views based on the users' selection of viewpoint and direction. Such systems are often very complex to build, and involve huge amount of data to be processed in real-time.

The number of cameras used in an FTV system is usually a tradeoff between data amount and rendering quality. The FTV system built by Tanimoto [17] consisted of 100 synchronized high resolution video cameras to capture the scene at 30 frames per second (fps). The main advantage of using so many cameras is that the rendering becomes relatively easy. Recent advances in image-based rendering [25] showed that with a dense set of images, one can accurately synthesize novel views without much knowledge of the 3D geometry of the captured scene. Nevertheless, it is very challenging to build a system with so many cameras, and it is nearly impossible to stream all the captured videos to remote users with today's Internet bandwidth. Alternatively, in [10, 15, 14], no more than 10 cameras were used to capture the scene, and the rendering is conducted by reconstructing the scene geometry on-the-fly. Unfortunately, with such a sparse set of cameras, the reconstructed geometry tends to be error-prone, and the rendered virtual views are often of low quality. Carranza et al. [6] proposed to render novel views from 8 cameras with an analysis-by-synthesis scheme. Although the pre-captured human 3D model helped improve the rendering results dramatically, such an approach is limited when applied to scenes with generic objects.

One fundamental challenge in geometry reconstruction from traditional camera arrays is the lack of accuracy in low-texture or repeated pattern regions, e.g., human faces or clothes. Recently, depth sensing cameras are becoming available from Canesta [5], Swiss- Ranger [16] and 3DV Systems [1] at commodity prices. These cameras derive the scene depth based on the time-of-flight principle and are often robust to such problems. In this paper, we explore the idea of synthesizing virtual views from a hybrid camera array, namely, a sparse set of video cameras with a depth sensing camera, as shown in Fig. 1. The adopted ZCam depth camera from 3DV Systems [1] is capable of returning depth images with 320×240 pixels resolution at 30 fps. With such a hybrid camera array, we expect to maintain the low data rate of sparse camera arrays, and be able to render high quality virtual views with the help of the depth image captured by the depth camera.

Our view synthesis algorithm reconstructs a view-dependent geometry given the user's viewpoint and look direction. Because the depth image returned by the depth camera is often very noisy, we transform the depth image to the current virtual viewpoint to obtain an initial estimate of the view-dependent geometry. The sparse regular cameras are then utilized to refine the geometry through constrained plane sweeping. The proposed algorithm with the hybrid camera array generates rendering results much superior to those produced by a regular sparse camera array, and it has very low computational complexity.

Figure 1: **Left:** Our hybrid camera system is placed on top of a $22''$ monitor with 15 degree bent linear extensions. At the center we have the depth camera and the intensity cameras are spread on the grid to cover a large area of interest for a teleconferencing application. **Right:** On the top, one of the PGR Flea cameras used to capture intensity and on the bottom the infrared based depth camera.

The rest of the paper is organized as follows: After summarizing the related work in Section 2 we outline our hybrid system and its calibration in Section 3. Section 4 presents our algorithm to generate virtual views and finally in Section 5 we present the results of our algorithm.

2 Related Work

Camera array is an effective scheme to capture dynamic scenes for virtual view synthesis. There have been many camera arrays built in the literature. For instance, Matusik et al. [10] used 4 cameras for rendering using image-based visual hull (IBVH). Yang et al. [23] built a 5-camera system for real-time rendering with the help of modern graphics hardware; Schirmacher et al. [14] built a 6-camera system for on-the-fly processing of generalized Lumigraphs; Naemura et al. [11] constructed a system of 16 cameras for real-time rendering; Chan et al. [8] built an 8-camera system that involves all the necessary components for 3D video broadcasting/conferencing including capturing, compression, streaming and rendering. Several large arrays consisting of tens of cameras have also been built, such as the Stanford multi-camera array (128 cameras) [20], the MIT distributed light field camera (64 cameras) [22], the CMU 3D room (49 cameras) [9] and self-

reconfigurable camera array (48 cameras) [24], and the Nagoya University FTV system (100 cameras) [17].

Large camera arrays often cover a wide viewpoint range. The cameras are dense (relatively speaking), hence the rendering algorithm can be simple. On the other hand, these systems are often constrained by the huge amount of data to be processed. The Stanford system focused on grabbing synchronized video sequences onto hard drives. It was demonstrated to be useful for approximating a conventional camera with various resolution, dynamic range, frame rate, aperture, etc [21]. The CMU 3D room was able to generate good-quality novel views both spatially and temporarily [19]. It utilized the scene geometry reconstructed from a scene flow algorithm that took several minutes to run. The MIT system did render live views at a high frame rate. Their method assumed constant depth of the scene, however, and suffered from severe ghosting artifacts due to the lack of scene geometry.

When the number of captured images for a scene is limited, adding geometric information can significantly improve the rendering quality [7]. In practice, an accurate geometric model is often difficult to attain. Recently, there has been increasing interest in on-the-fly geometry reconstruction for virtual view synthesis. For instance, Schirmacher et al. [14] built a 6-camera system which was composed of 3 stereo pairs and reconstructed the depth from stereo algorithms. Each stereo pair needed a dedicated computer for the depth reconstruction, which is expensive to scale when the number of input cameras increases. Naemura et al. [11] constructed a camera array system consisting of 16 cameras. A single depth map was reconstructed from 9 of the 16 images using a stereo matching PCI board. Such a depth map is computed with respect to a fixed viewpoint, thus the synthesized view is sensitive to geometry reconstruction errors. Matusik et al. [10] proposed image-based visual hull (IBVH), which rendered dynamic scenes in real-time from 4 cameras. The computational cost of IBVH is low thanks to an efficient pixel traversing scheme, which can be implemented with software only. To overcome geometry errors in concave regions, Slabagh et al. [15] later extended IBVH to image-based photo hull (IBPH). IBPH utilizes the color information of the images to identify scene geometry, which results in more accurately reconstructed geometry.

Most of the algorithms described above reconstruct the scene geometry from a fixed viewpoint. Since the geometry is very error-prone, when the virtual viewpoint changes, the rendering results may suffer. A few researchers have explored the idea of view-dependent geometry [13, 24, 23], which seems to provide better rendering results in general. In [23], Yang et al. proposed a real-time consensus-based scene reconstruction method using commodity graphics hardware. Their algorithm utilized the Register Combiner for color consistency verification (CCV) with a sum-of-square-difference (SSD) measure, and obtained a per-pixel depth

map in real-time. In [24], the view-dependent geometry is reconstructed in software by a coarse-to-fine subdivided mesh data structure. Their system with 48 cameras is capable of rendering at 4-10 fps with a single computer handing data collection, decompression, camera calibration and rendering. Our virtual view synthesis algorithm resembles the method in [24]. However, it makes use of the additional depth camera information, and integrates a few recent advances in stereo matching for improving the accuracy of the depth sweeping algorithm, such as using the DAISY descriptor [18] for feature matching.

As mentioned earlier, geometry reconstruction from traditional camera arrays lacks accuracy in low-texture or repeated pattern regions. In recent publications [2, 3, 27], researchers have studied the problem of obtaining the scene geometry with the help of depth sensing cameras. In [2], a low-resolution depth sensing camera is combined with a high resolution video camera to boost the resolution of the final depth map. In [3], Beder et al. conducted an interesting comparison between the accuracy of depth maps obtained from depth sensing cameras and stereo rigs. Their conclusion is that the time-of-flight system outperformed the stereo system in terms of achievable accuracy for distance measurements, while the estimation of normal direction is comparable for both systems. The most interesting study is probably by Zhu et al. [27], which combined a depth sensing camera with a stereo camera, and demonstrated quantitatively that by fusing their depth maps together, one can achieve much better accuracy than the result from either source. They showed that the two systems are in many sense complimentary – time-of-flight systems perform better in low texture regions, while stereo systems perform better in high texture regions. Nevertheless, their scheme is relatively slow and applicable to only static scenes from a fixed viewpoint. In this paper, we combine a depth sensing camera with 4 regular cameras (Fig. 1), and explore algorithms to reconstruct view-dependent geometry from both types of sensors for dynamic scenes.

3 The Hybrid-Camera Array

3.1 System Hardware

Our hybrid camera array shown in Fig. 1 is composed of a single 3D depth camera in the center and 4 regular intensity cameras in a planar grid around the range of interest. The depth camera is a USB connected infrared pulse based camera from 3DV Systems[1]. It is capable of returning 320×240 pixels resolution depth map at 30 fps or 160×120 pixels resolution at 60 fps. The camera emits an infrared pulse using the infrared emitters around the camera sensor and by shuttering the reflected pulse returns a depth estimate. Intensity cameras are standard Point Grey

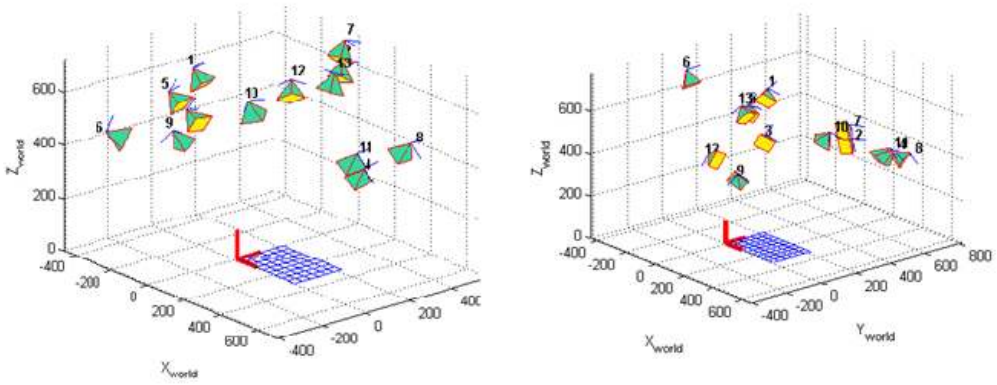

Figure 2: Poses of 13 frames with respect to the calibration pattern for two different cameras. By computing the relative poses of the individual frames in different cameras, it is possible to estimate the relative poses of the cameras.

Flea [12] cameras with 640×480 resolution at 30fps.

The cameras are placed on a roughly arc grid. The depth camera is in the center and the intensity cameras are placed regularly in a not-so-small baseline fashion (around $7''$ between the two cameras on the same side) on top of a $22''$ monitor. The intensity cameras are synchronized in between themselves by connecting them to a single firewire bus. The depth camera has a USB connection and no external synchronization input. Therefore it is not possible to synchronize the system at the hardware level. The overall system is semi-synchronized during capture by buffering the outputs of the cameras and dumping them once the buffer is full. Our system is able to capture depth (320×240) and intensity (640×480) maps at 20 fps.

3.2 System Calibration

The depth camera returns an intensity image together with the depth map, and these two images are internally aligned. In order to calibrate the depth camera with the four intensity cameras, we use the depth camera's intensity image. The camera calibration toolbox of [4], which implements Zhang's method [26], is used for obtaining the internal parameters of the cameras. The pose estimates of the planar pattern computed during internal calibration are then used to perform external calibration. As shown in Fig. 2, the relative poses of each of the captured frames during calibration is provided by the toolbox. We compute the pose of each frame

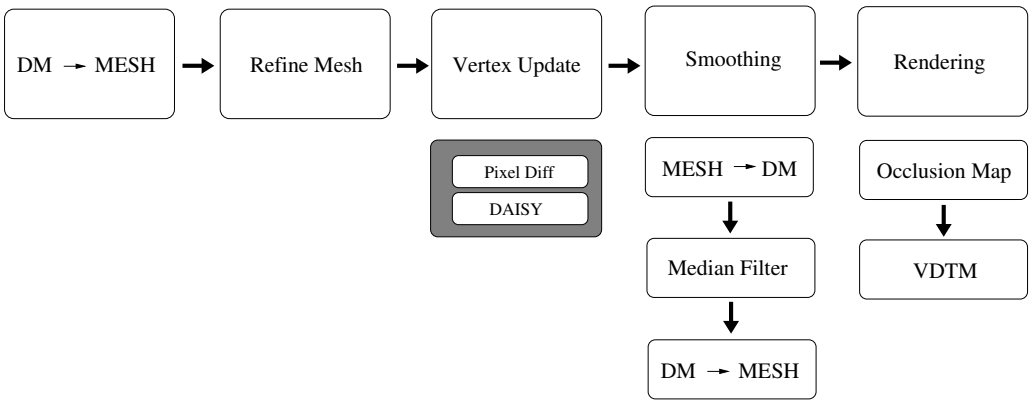

Figure 3: **Mesh based view generation.** Block diagram of our overall algorithm: We first convert the depth estimate of the depth camera into a mesh and the refine this mesh. Then it is updated through constrained plane sweeping using the texture cameras and the finally new view is rendered from the smoothed depth map in a view dependent fashion.

with respect to the reference camera by using

$$\begin{array}{rcl} \overline{R}_i^k & = & R_i^k (R_f^k)^T \\ \overline{t}_i^k & = & t_i^k - \overline{R}_i^k t_f^k \end{array} \tag{1}$$

where R is the rotation matrix, t is the translation vector, lower subscript i is the camera index, upper subscript k is the frame index and overline represents the resulting relative quantity with respect to the reference camera f. The final pose of the camera is found by averaging the relative poses over all frames. Averaging of the rotation is done by first converting the rotation matrix to its Rodrigues form in order to retain its rotation properties.

$$\begin{array}{rcl} \overline{r}_i & = & \frac{1}{K} \sum_{k=1}^{K} \overline{r}_i^k, \;\; \overline{r}_i^k = Rodrigues(\overline{R}_i^k) \\ \overline{t}_i & = & \frac{1}{K} \sum_{k=1}^{K} t_i^k \end{array} \tag{2}$$

This multi-camera calibration gives us a reasonable calibration quality and requires minimal user interaction.

4 Virtual View Synthesis

In this section, we first briefly outline our overall algorithm to fuse the output of a depth camera and a multiplicity of intensity cameras for virtual view generation. Detailed information about each step is given in the following sub-sections.

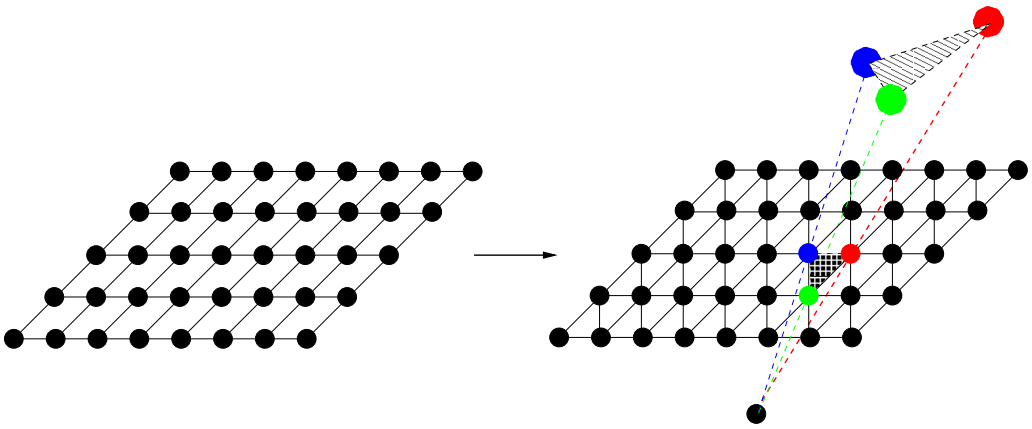

Figure 4: **Computing 3D mesh from a depthmap.** Every 2×2 pixel square on the left is transformed into two triangles as in the right figure. After 2D triangulation, corresponding 3D points for each pixel location is computed by using the depth map and the 3D triangulated mesh is computed.

The diagram of our system is shown in Fig. 3. The depth map from the depth camera is first transformed into a 3D mesh. This enables us to work independent of the resolution of the depth estimate and that of the desired virtual view. Depending on the position and internal parameters of the virtual view, new vertices and facets are introduced to the mesh in order to avoid facets to be projected onto a non-planar region in the virtual view. Afterwards we update the positions of the vertices using the intensity cameras through constrained plane sweeping. The vertices are updated independently from each other to reduce computational requirements. To reduce the noise caused by such independence, we smooth the mesh and then render the virtual view with view-dependent texture mapping.

4.1 Compute The Initial Mesh

The depth estimate is first transformed into a 3D mesh in order to render disregarding the depth camera resolution. This is a straightforward process. As shown in Fig. 4, the 2D grid of the depth map is first triangulated by defining 2 triangles for every 2×2 square pixels and then the 3D position of each pixel is computed using the depth estimate. When necessary, mesh simplification can be conducted to reduce the number of vertices and facets. The final result is a 3D mesh representation that is independent of the view-point.

Note the depth estimate may contain holes where no infrared light is reflected from the scene (e.g., some hair region). We apply a smoothing filter to fill such holes before the above mesh conversion.

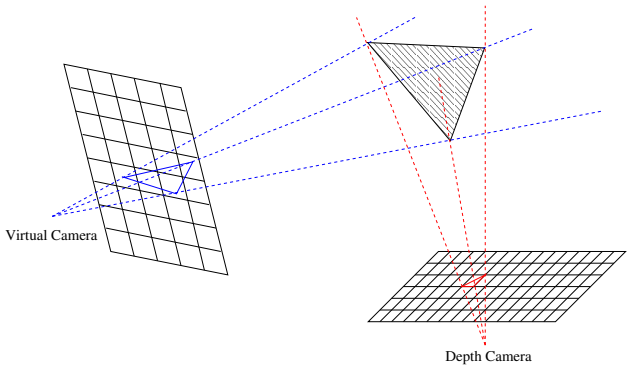

Figure 5: **Resolution.** Depending on the relative position of the virtual camera and the depth camera, the image of a facet on the mesh can occupy different areas in these images.

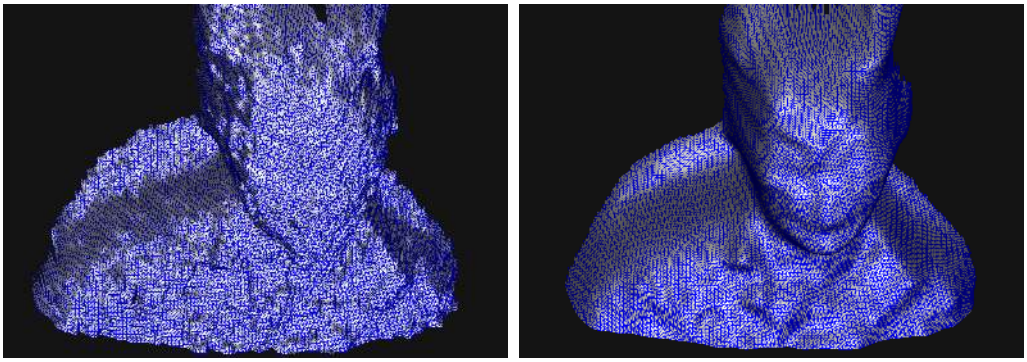

Figure 6: **Mesh Refinement.** *Left:* The original mesh is noisy, has holes and of lower resolution. *Right:* We smooth the mesh and introduce new vertices in order to reduce the noise level and close the holes.

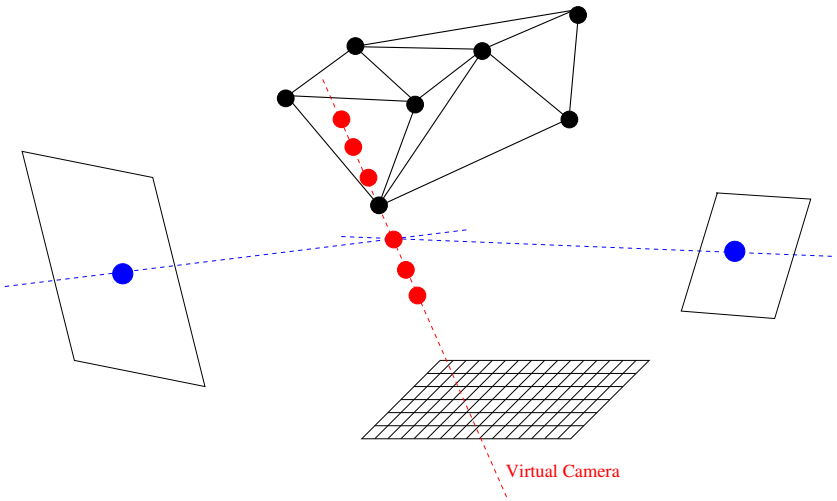

Figure 7: **Vertex Update.** Each vertex position is updated by checking the dissimilarity of the candidate vertex position (red circles) on visible intensity images. Candidate vertex positions are computed on the ray that combines the virtual camera center with the updated vertex within the search space for that vertex.

In addition, due to perspective projection, some facets of the mesh may project to an overly large region in the virtual view, as shown in Fig. 5, which may not be planar. Such facets are subdivided in order to maintain a relatively constant resolution of the mesh with respect to the virtual view point. In practice, a threshold is set, and facets with projected area larger than the given threshold will be subdivided. Fig. 6 demonstrates the effectiveness of smoothing and mesh subdivision for a typical scene.

4.2 Vertex Update

Up to this point only the information coming from the depth camera is used. In this sub-section, the position of the vertices of the mesh are updated through constrained plane sweeping using a dissimilarity measure. As shown in Fig. 7, for a given mesh vertex, we introduce candidate vertices along the ray joining the camera center of the virtual point and the vertex position. The search space is determined by the uncertainty associated with each vertex. Each candidate position is tested using a dissimilarity score and the best one is selected as the new vertex position. In our current implementation, we first project the candidate vertex onto the intensity image. The dissimilarity of the projected positions are measured through pixel differencing and DAISY [18]. We noted that the DAISY descriptor performs better in this task and used it in all of our experiments. A comparison between pixel differencing and DAISY as the dissimilarity measure will be given

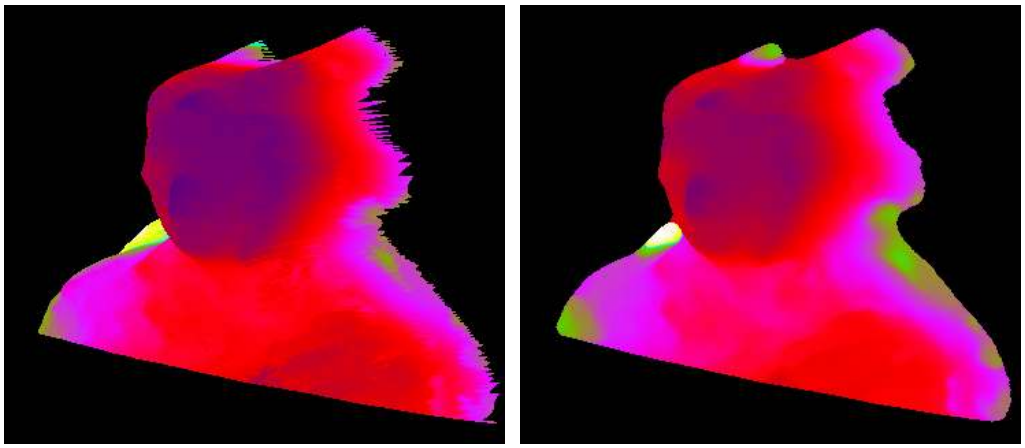

Figure 8: **Smoothing the depth map.** On the **left**, the depth map after the vertex update stage is shown. Since the vertices are updated independently, the resulting depth map is noisy. On the **right**, the final depth map after the smoothing is shown. In both of the images, for visual clarity, we increased the contrast of the depth maps and colorized them.

in Section 5.

Since the update of the vertices are done independently from each other, the resulting mesh is very noisy, which can cause artifacts during rendering. In the current implementation, we project the mesh onto the virtual viewpoint, and subsequently smooth the resultant 2D depth image using median filtering, as shown in Fig. 8. Note such smoothing can also be done on the 3D mesh directly.

4.3 View Synthesis

Given the refined view-dependent scene geometry, rendering is performed with view-dependent texture mapping. In this process, we first compute occlusion maps for each intensity camera, and then render the virtual view through the visible pixels via alpha blending.

Fig. 9 demonstrates the basic process of occlusion map computation. If a pixel is occluded for camera C_i, then its depth with respect to C_i computed from the 3D point X_v using the virtual camera's depth value will be different from the value in C_i's depth map D_i. Formulating this observation shapes our occlusion map estimation algorithm:

$$o_i(y,x) = \begin{cases} 1 & \text{if } \frac{|d_i - d_f|}{d_f} > 0.01 \\ 0 & \text{else} \end{cases}$$

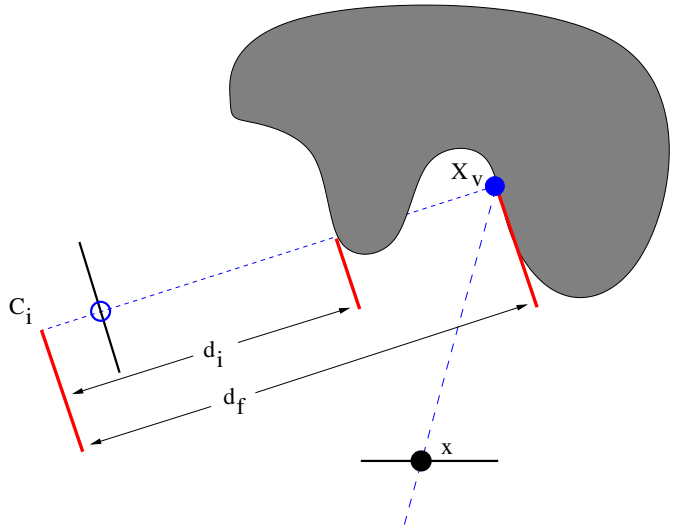

Figure 9: **Occlusion Map Computation.** In order to find if the black point, x, is occluded or not with respect to the camera, C_i, the point is back-projected onto the 3D model and then the found 3D point position, X_v, is projected onto that camera. If the distance, d_f, between the 3D point and the camera center is different from the value, d_i, of the depthmap from the point of view of camera C_i, then we say that the point x is occluded with respect to camera C_i

.

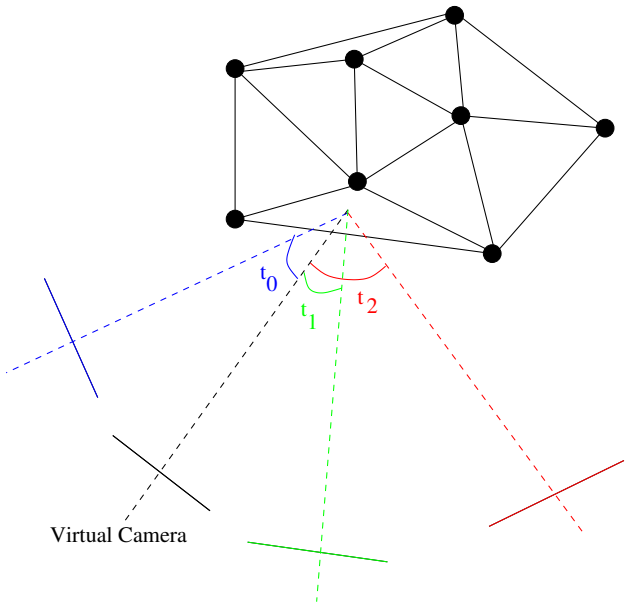

Figure 10: **Rendering.** The intensity of the new image is found by weighting the intensities of all the visible images inversely proportional to the angle between the ray coming from the virtual image and the ray of the respective cameras. For instance in the above example, t_2 is the biggest angle and therefore the intensity in the rightmost image will have the lowest weight.

We allow for the small error(1%) in order to account for numerical errors and the fact that the depth map, D_i, will have pixel resolution whereas the projection of X_v will be a sub-pixel location.

Once the occlusion maps are found for each input image, we render the virtual view by weighting the intensities of the visible images inversely proportional to their incidence angles on the surface of the mesh (Fig. 10). The image intensities are weighted according to

$$w_i = \begin{cases} exp\left(-\frac{t_i^2}{2\sigma^2} \right) & \text{if } t_i < 3\sigma \\ 0 & \text{else} \end{cases}$$

where σ is the maximum allowable ray angle and t_i is the angle that the ray coming from image i makes with the virtual camera's ray on the surface of the mesh. The weights are normalized to sum to 1 and the intensity of the rendered view is computed using alpha blending. The virtual view is rendered by weighting the Laplacian pyramids of the warped images by their respective weight Gaussian

Figure 11: **Frozen Time.** In this example, we turn the virtual viewpoint around the subject at a frozen instant. The input to the system is shown in the first row and the remaining images are example renderings at different camera locations

pyramids:

$$V = LaplacianPyr^{-1}\left(\sum_i LaplacianPyr(T(I_i, M)).*GaussianPyr(W_i)\right) \tag{3}$$

where I_i is the i^{th} image with weight map W_i, and $T(., M)$ is the warp function with respect to a mesh, M.

5 Results

In this section we present results of our algorithm with different sequences. The first example, Fig. 11, is the rendering of a subject in a frozen time where we render the scene at different positions and look directions of the virtual viewpoint. For each view a separate depth map is computed (thus view-dependent geometry) as outlined in the previous sections. The background of the scene are removed during rending using the depth map given by the depth camera. It can be seen that the rendering quality is in general very good.

In Fig. 12, we present the results of our algorithm for a number of challenging

sequences. In this figure, the first four 2×2 frames are the input intensity images. The rendering results are presented in the middle column. The last column shows the depth maps we computed in the process. Note even for scenes with complex geometry such as the first 3 rows, the rendering quality is still very good. This is very encouraging considering the sparsity of our camera array compared with previous approaches.

We also compared our results with the results of using the depth camera's depth estimate directly in Fig. 13 in order to show that using only the depth camera is not enough most of the time and that we also need texture cameras to correct the noise, misalignment and registration errors of the depth camera. The first column of Fig. 13 shows the results of our algorithm where the second column is the results when we use the depth camera's depth estimate directly. Our algorithm clearly renders much better results. In addition, we also compare the performance of pixel differencing (third column) and DAISY [18] (first column) as the dissimilarity measure for geometry refinement. In a sparse camera arrangement like ours, we notice that although pixel differencing sometimes works as good as DAISY, in many cases it can cause more artifacts due to incidental intensity matching.

In Fig. 14, we show two failure cases to demonstrate the limitation of our algorithm. One problem we need to address in the future work is the heavy dependence on the depth map provided by the depth camera. For example, in cases where the depth camera fails to return a depth estimate due to the lack of infrared light reflection, our current algorithm does not have a good way to correct it. One potential solution is to increase the depth search range during vertex update, which could be risky if no constraints of the resultant depth is provided. Also, shiny objects tend to cause halo's in the depth estimate and this translates to artifacts in our rendering results.

6 Conclusion

Virtual view synthesis is a hard problem and even more so when it is applied to 3D video broadcasting/conferencing where the purpose of the rendering is generally a human face which is sensitive to distortions or artifacts. In this paper, we propose a new view synthesis algorithm which uses a hybrid system, a depth camera and four regular cameras, for this hard problem. To our best knowledge, our hybrid system is the first of its kind and it shows promising results in merging the depth cameras which became recently available with regular camera arrays. This has the advantage of reducing the total number of necessary cameras and still achieving high rendering quality, which will reduce the system cost and bandwidth requirements needed for boardcasting/conferencing.

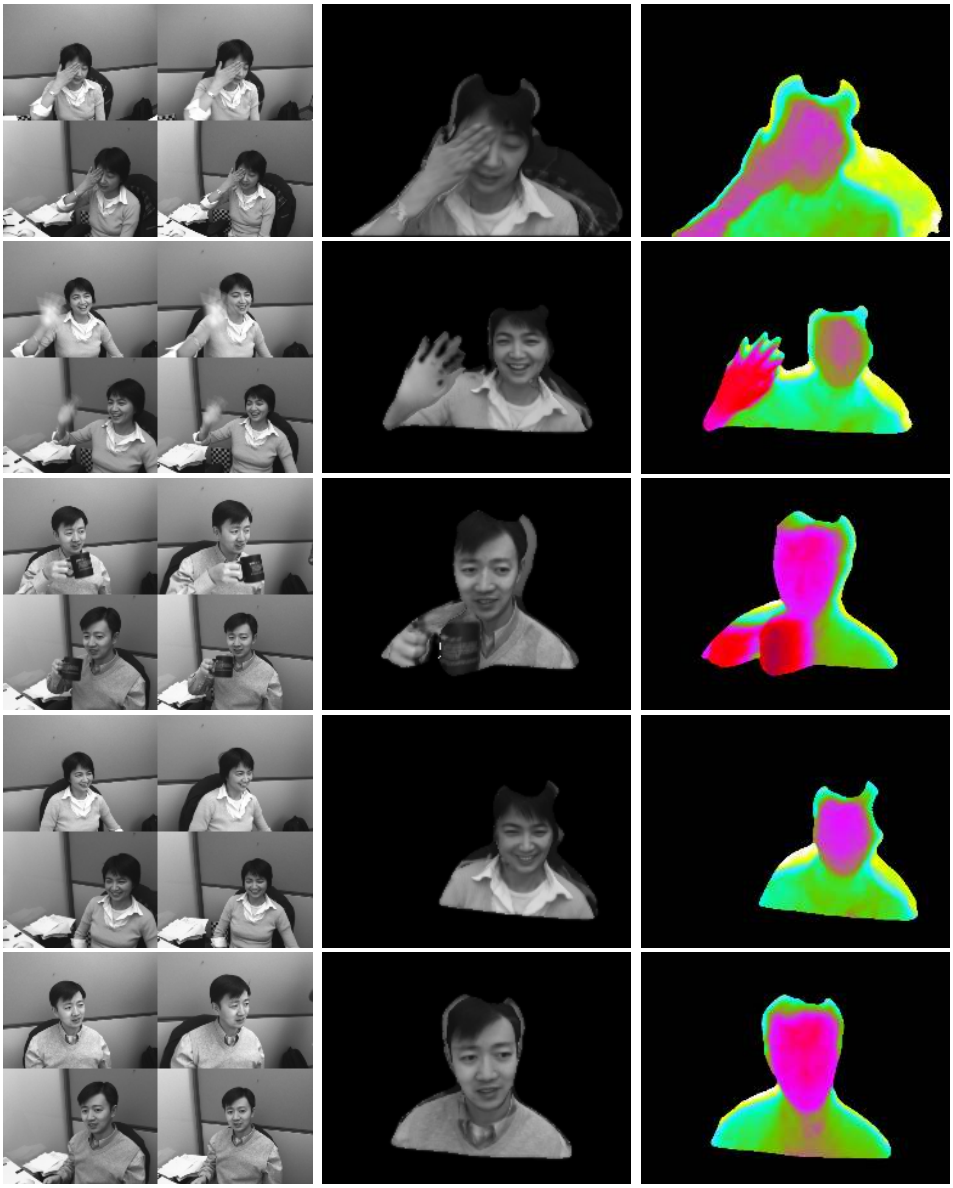

Figure 12: **Some Renderings.** Here we show the results of our algorithm for some example frames. Within the rows, first 2×2 image shows the input of the cameras, middle column is our result and last column is the computed depth map. We see that even for some challenging cases, like the first 3 rows, our algorithm produces decent results.

Figure 13: **Comparison.** In this figure we compare our results with two different approaches. **Left** column shows our results, in the **middle** column we display the results when the depth camera's depth estimate is directly used to render the virtual view and in the **right** column, we show the results of our algorithm but with pixel differencing as a dissimilarity measure. Direct rendering results in artifacts caused by registration errors and misalignments. Using pixel differencing, on the other hand, sometimes results in good renderings (third row) whereas in other cases it causes artifacts (first and second rows).

Figure 14: **Failed Renderings:** Some frames where our rendering algorithm failed. Left figure shows that at the places where the depth camera failed to return a depth estimate, we could not render the scene. Also, the right figure shows an example where we fail to correctly estimate the sharp boundary of the shiny cup and thereby causing a distortion of the texture around the cup in the rendering.

References

[1] 3DV Systems. http://www.3dvsystems.com.

[2] T. D. A.Prasad, K.Hartmann, W.Weihs, S. E. Ghobadi, and A.Sluiter. First steps in enhancing 3d vision technique using 2d/3d sensors. In *Computer Vision Winter Workshop*, pages 82–86, 2006.

[3] C. Beder, B. Bartczak, and R. Koch. A comparison of pmd cameras and stereo-vision for the task of surface reconstruction using patchlets. In *Proc. CVPR*, 2007.

[4] J.-Y. Bouguet. Camera calibration toolbox for matlab, http://www.vision.caltech.edu/bouguetj/calib_doc/, 1999.

[5] Canesta Inc. http://www.canesta.com/.

[6] J. Carranza, C. Theobalt, M. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. In *Proc. SIGGRAPH*, pages 569–577, 2003.

[7] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong. Plenoptic sampling. In *Proc. SIGGRAPH*, pages 307–318, 2000.

[8] Shing-Chow Chan, King-To Ng, Zhi-Feng Gan, Kin-Lok Chan, and Heung-Yeung Shum. The plenoptic video. *IEEE Trans. CSVT*, 15(12):1650–1659, 2005.

[9] T. Kanade, H. saito, and S. Vedula. The 3d room: Digitizing time-varying 3d events by synchronized multiple video streams. Technical Report CMU-RITR-98-34, Department of Computer Sciences, Carnegie Mellon University, 1998.

[10] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *Proc. SIGGRAPH*, pages 369–374, 2000.

[11] T. Naemura, J. Tago, and H. Harashima. Real-time video-based modeling and rendering of 3d scenes. *IEEE Computer Graphics and Applications*, 22(2):66–73, 2002.

[12] Point Grey Flea. http://www.ptgrey.com/products/flea/index.asp.

[13] P. Rademacher. View-dependent geometry. In *Proc. SIGGRAPH*, pages 439–446, 1999.

[14] H. Schirmacher, M. Li, and H.-P. Seidel. On-the-fly processing of generalized lumigraphs. In *Proc. EUROGRAPHICS*, 2001.

[15] G. G. Slabaugh, R. W. Schafer, and M. C. Hans. Image-based photo hulls. Technical Report HPL-2002-28, HP Labs, 2002.

[16] Swissranger Inc. http://www.csem.ch/fs/imaging.htm.

[17] M. Tanimoto. Free viewpoint television. *J. Three Dimensional Images*, 15(3):17–22, 2001.

[18] E. Tola, V. Lepetit, and P. Fua. A Fast Local Descriptor for Dense Matching. In *Proc. CVPR*, Anchorage, AK, June 2008.

[19] S. Vedula, S. Baker, and T. Kanade. Spatio-temporal view interpolation. In *13th ACM Eurographics Workshop on Rendering*, 2002.

[20] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. Horowitz. The light field video camera. In *Proc. of Media Processors*, 2002.

[21] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville (Eddy) Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Marc Levoy, and Mark Horowitz. High performance imaging using large camera arrays. In *Proc. SIGGRAPH*, pages 765–776, 2005.

[22] Jason C. Yang, Matthew Everett, Chris Buehler, and Leonard McMillan. A real-time distributed light field camera. In *Eurographics Workshop on Rendering*, 2002.

[23] R. Yang, G. Welch, and G. Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proc. Pacific Graphics*, 2002.

[24] C. Zhang and T. Chen. A self-reconfigurable camera array. In *Eurographics Symposium on Rendering*, 2004.

[25] C. Zhang and T. Chen. A survey on image-based rendering - representation, sampling and compression. *EURASIP Signal Processing: Image Communication*, 19(1):1–28, 2004.

[26] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. PAMI*, 22(11):1330–1334, 2000.

[27] Jiejie Zhu, Liang Wang, Ruigang Yang, and James Davis. Fusion of time-of-flight depth and stereo for high accuracy depth maps. In *Proc. CVPR*, 2008.