Fast and Accurate 3D Reproduction of a Remote Collaboration Environment

ABM Tariqul Islam[1], Christian Scheel[1], Ali Shariq Imran[2], and Oliver Staadt[1]

[1] Visual Computing Lab, Dept. of Comp. Science, University of Rostock, Germany
[2] Gjøvik University College, Norway
{tariqul.islam,christian.scheel,oliver.staadt}@uni-rostock.de,
ali.imran@hig.no

Abstract. We present an approach for high quality rendering of the 3D representation of a remote collaboration scene, along with real-time rendering speed, by expanding the unstructured lumigraph rendering (ULR) method. ULR uses a 3D proxy which is in the simplest case a 2D plane. We develop *dynamic proxy* for ULR, to get a better and more detailed 3D proxy in real-time; which leads to the rendering of high-quality and accurate 3D scenes with motion parallax support. The novel contribution of this work is the development of a *dynamic proxy* in real-time. The *dynamic proxy* is generated based on depth images instead of color images as in the Lumigraph approach.

Keywords: 3D reproduction, remote collaboration, telepresence, unstructured lumigraph rendering, motion parallax.

1 Introduction

Recent advancements in display technology have made a tremendous influence on the research related to displaying realistic and interactive representation of a scene. Moreover, because of availability of low-cost color and depth cameras, systems capable of displaying such 3D representations have become cost effective, and many applications, such as elearning and remote collaboration, are including large camera arrays to support 3D visualization of a scene [1][2]. Nowadays, we see camera arrays consisting of different camera types, on either side of a remote collaboration setup, for capturing the whole scene [1]. In order to provide an immersive meeting experience, a remote collaboration system should provide the users with a real-time reproduction of the collaborating environment and also a way to interact properly with the distant participants [3].

A number of research projects have been carried out to provide such an interactive 3D representation; our focus is on remote collaboration or telepresence system, as depicted in Fig. 1. We can see in Fig. 1 that image acquisition is taking place on the local site of the telepresence system and then transmitted to the remote site where 3D scene is visualized on the display. Although, the area of 3D representation of a scene has been explored during the last few years [4][5][6], high-quality reconstruction and immersive interaction method is still an

R. Shumaker and S. Lackey (Eds.): VAMR 2014, Part I, LNCS 8525, pp. 351–362, 2014.

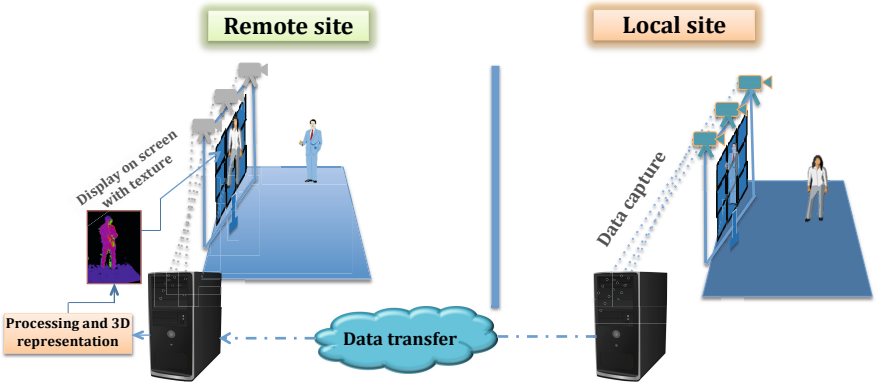


Fig. 1. Illustration of a telepresence system with 3D representation of the scene

open problem. Most of the existing approaches either suffer from slower reconstruction speed or from visible artifacts on the represented scene. Moreover, the huge dynamic data, generated from camera arrays, adds additional problem of data transmission [7].

We present here an approach for high-quality 3D rendering of a remote collaboration scene, with real-time rendering speed, by means of expanding the Unstructured Lumigraph Rendering (ULR) method [8]. In the ULR approach, the authors use a 3D proxy, which is in the simplest case a 2D plane. A more complex proxy would be, for example, an arbitrary triangle mesh. In our work, we develop a *dynamic proxy* for getting a better and more detailed 3D proxy in real-time; which leads to a high-quality 3D scene representation. The proxy in ULR doesn't get updated when the scene is changed in real-time; our novel idea of generating dynamic proxies help to remove this obstacle and get updated in real-time. Like any other proxy, which supports motion parallax, our developed system with *dynamic proxy* also supports motion parallax; thus, proper interaction for the represented scene is obtained. Dynamic proxies also help to avoid generating huge dynamic data by reducing the number of required cameras. GPU implementation of our idea leads to achieving real-time rendering of the represented scene. We map the texture from color images on top of the depth images which are obtained from a set of Microsoft kinects. We also adapt a recently developed depth camera calibration method [9] for our system. The rest of the paper is organized as follows: in section 2, we present some related work for 3D scene representation; in section 3, we present a brief description of ULR method; section 4 describes details of our system; in section 5, we describe our system setup and implementation details; in section 6, we present our results and some existing limitations and finally, in section 7, we conclude our paper.

2 Related Work

Maimone and Fuchs, in [10], present a telepresence system in which the users can view a 3D representation of the scene captured by a set of kinects placed

on the remote site. On the basis of per pixel quality assessment, they merge overlapping depth contributions. Although, the system generates dynamic scene from arbitrary position, it suffers from lower rendering speed issues when more enhancement parameters, for removing interference issues, are activated [10]. Researchers also generated such 3D representations by combining textures from 2D video stream on top of the depth information obtained from depth cameras [1][4]. They obtain 3D representation by projecting the foreground silhouettes backwards onto 3D plane and refine it by view-dependent depth estimation [5]. Beck et al. presents an immersive telepresence system [11] where a set of kinects is used to capture the scene. They apply similar 3D reconstruction method as in [10]. Although, they provide an immersive interaction experience through a complex setup, the visual appearance is not quite satisfactory. Hansung et al. present a dynamic 3D scene representation approach [5] for outdoor environments.

For producing a 3D representation of a scene, researchers very often use image-based rendering (IBR) approach rather than traditional three-dimensional graphics [8]. Gortler et al. present an IBR method, named as *Lumigraph* [12], which is capable of reconstructing an accurate ray from a limited number of acquisition cameras. Shum et al. propose an alternate approach [13], which use low precission depth correction for lower dimensional Lumigraph method. Buehler et al., in [8], have presented an IBR method, called ULR. We discus briefly about ULR in the next section.

3 Unstructured Lumigraph Rendering

The ULR algorithm [8] generalizes two IBR approaches – *Lumigraph* [12] and *View-Dependent Texture Mapping* [14]. ULR has three input parameters – a polygon mesh that represents the scene geometry, an unstructured image set and camera pose information for each image. The authors refer the polygon mesh as *geometric proxy* which is in principle similar to the *proxy* stated in [15].

In the ULR, a *camera blending* field determines how each input camera is weighted to reconstruct a pixel. Each pixel of the reconstructed image is calculated from the weighted average of the corresponding pixels of the input images [8]. The blending weight function w_{ang} and normalization of blending weights $\widetilde{w}_{ang}(i)$ are calculated as in Eq. 1 and Eq. 2.

$$w_{ang}(i) = 1 - \frac{penalty_{ang}(i)}{thres_{ang}} \tag{1}$$

$$\widetilde{w}_{ang}(i) = \frac{w_{ang}(i)}{\sum_{j=1}^{k} w_{ang}(j)} \tag{2}$$

In Eq. 1, $penalty_{ang}(i)$ is the angular difference between a ray from a virtual camera (the new viewpoint) and the ray of i-th input camera to the point where the ray from the virtual camera intersects the proxy. The $thres_{ang}$ angle is the largest $penalty_{ang}$ of the k-nearest cameras.

4 System Description

Most of the existing telepresence systems, such as the *extended window metaphor* (EWM) [7] and [11], suffer from generating huge dynamic data because of using large number of cameras. We develop a *dynamic proxy* which combines the depth images from the depth cameras in addition to the basic idea of the ULR approach. Since, scene geometry information is used, ULR can reproduce a new view using few number of input camera images.

However, ULR has its own limitation as well; the *proxy* in ULR doesn't update when the scene is changed in real-time. To resolve this, we develop *dynamic proxy* for representing the scene in 3D from a new view-point; thus, it supports depth images for creating a new depth image out of existing depth images which can be used as a proxy. *Dynamic proxies* also help to reduce the processing time significantly, because they generate arbitrary view-points only for the part of the scene which is visible from a particular view rather than for the whole scene.

4.1 Dynamic Proxy Generation

The data of the depth cameras should be merged to generate the *dynamic proxy*; the *dynamic proxy* is then used to render the scene from a new point of view. To generate a *dynamic proxy*, depth information is first obtained with the Microsoft kinects; then, the depth values are passed through a median filter for noise reduction. We choose the median filter in order to preserve the edges of the objects inside a scene. Then the filtered points of each kinects are transferred to a common coordinate system. The camera extrinsics, via calibration, must be known for transferring the 3D points to a common coordinate system.

For each camera, a triangle mesh is generated from the 3D points of that camera that are transferred to a common coordinate system. For the creation of the triangles, the neighbor relationships of the 3D points are used for each camera. For each 3D point, a check is performed whether the immediate-right, lower-right and bottom neighbors are still present in each case after filtering. If a 3D point does not have at least two neighbors, no triangle is created. When there are two neighbors, one triangle is created and when there are three neighbors, two triangles of the 3D point are generated. To remove unwanted triangles between object edges and object background or foreground object, triangles are removed, in which the distance between two points is above a threshold.

For each pixel, a ray R_i is cast from the position of the virtual camera V into the scene. For each ray, the intersection points S_{ik} with the triangle meshes T_k are calculated, see Fig. 2(a). When a ray has multiple intersections with the triangle mesh of a camera, only the closest one to the virtual camera is taken. When a ray has intersection points with the triangle meshes of several cameras, the depth values of this intersection points are blended through a weighting. Instead as in the *Lumigraph* [12], where best color information is searched, here we look for the best depth information.

The weighting is performed by comparing the angle between the rays of the cameras to their 3D points and the rays from the virtual camera to these points.

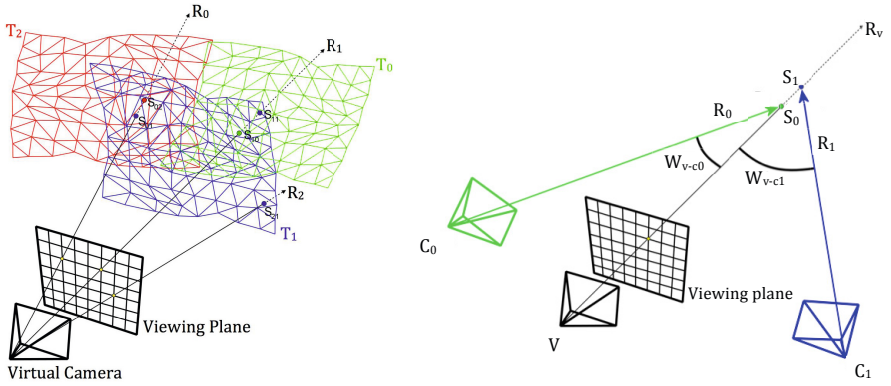


Fig. 2. (a) Rays R_i cast from the virtual camera V; S_{ik} intersection points, T_k triangle meshes. (b) Weighting the angles W_{V-C_i} between the rays; C_i kinects.

Smaller angle between the ray of a camera and the ray of the virtual camera indicates a higher weighting for the point from the camera.

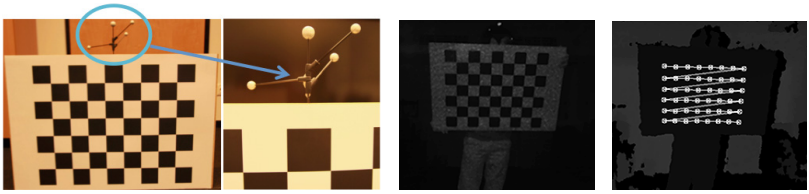
The weighting also takes into account that a intersection point of a ray, from the virtual camera, that lies far behind other intersection points from the ray, is not used. Because, it is assumed that this point then belongs to another object or to another side of the same object. The total weight ω is defined as follows:

$$\omega = \rho \cdot \theta \tag{3}$$

$$\rho = \upsilon_\tau \cdot d_\tau \tag{4}$$

In Eq. 3 and Eq. 4, θ, ρ, υ_τ and d_τ refer to the angle, penalty, position threshold and distance threshold respectively. $\upsilon_\tau = 0$, if a point lies outside a given scene, otherwise 1 and $d_\tau = 0$, when an intersection point from the view of the virtual camera is more than a threshold value behind a different intersection from the same ray. If one of these two variables is 0, the total weight ω is set to 0. θ is a weight for the angle between a ray of the virtual camera and a ray of a kinect to a point in the scene (see Fig. 2(b)). θ is calculated and normalized (see, *blending shader* in section 5.1) like in Eq. 5. Acceleration techniques must be applied to calculate the intersection of a ray with a triangle mesh, since it is computationally very expensive to test the ray with each triangle.

To accelerate the *dynamic proxy* generation, we use GPU acceleration technique (see section 5.1). The generated *dynamic proxy* can then be used for the ULR. The important point for using the *dynamic proxy* with the ULR is to interlace the processing of the color cameras in the processing of the depth cameras; this is required especially for a GPU implementation to get real-time performance. Moreover, it is also necessary to calibrate the color cameras with respect to the depth cameras, or to calibrate all cameras to a specific world coordinate system. It is also possible to generate a *dynamic proxy* from the perspective of



(a) (b) (c)

Fig. 3. (a) Checkerboard with the rigid body, (b) blurred IR image [9] of kinect, (c) detected corners that are mapped from the IR image to the depth image [9]

each color camera and then obtain depth information for each color value. This depth information is very useful when combining multiple color images to a new image, for example, when occlusions occur within the scene.

4.2 Camera Calibration

There exist a number of works for depth camera calibration; for our proposed 3D scene rendering system, we adapt the work [9], by Avetisyan et al. It uses a 3-D lookup table to support per-pixel and per-distance mapping at every pixel in the depth image. For finding the *intrinsic* and *extrinsic* parameters [9], we use the standard checkerboard based approach. The calibration method uses a rigid body attached to the standard planar checkerboard as shown in Fig. 3(a). During the calibration, the rigid body is being traced by well calibrated 12 DOF (degrees of freedom) OptiTrack [16] system.

After obtaining the mapping between infrared (IR) and depth images of the kinects, we transfer the corner coordinates from the IR image into the depth image [9]. Illustration of the resulting depth images with the detected corners is shown in Fig. 3(b) and 3(c). This approach doesn't need any hard mechanical setup or distance measuring tools. The ground truth values are recorded by the real-time tracking system and are used for calculating real distances of the points that are extracted from the checkerboard pattern. Moreover, this approach supports simultaneous depth correction for multiple kinects.

5 System Setup and Implementation

We use two kinects as depicted in Fig. 4 for capturing the scene. For displaying the rendered 3D scene, we use a display wall which consists of 24 DELL 2709W displays; the color cameras are integrated on the bezels of the displays. The display wall has a combined resolution of 55 Mio pixels. For tracking purpose, we use the 12 DOF OptiTrack [16] system. We develop the system with C++ and use libfreenect from OpenKinect as camera driver libraries. For this work, we use a single Macbook Pro, with 2.3 GHz intel i7 processor and NVIDIA GeForce GT 650M 1024 MB graphics card.

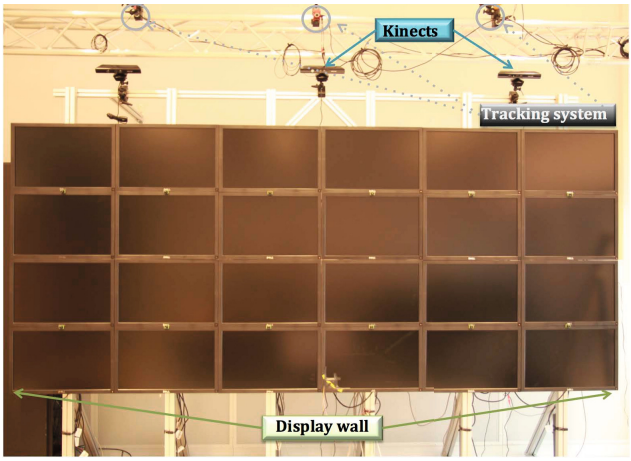


Fig. 4. Lab setup with kinects, tracking system and display wall

5.1 GPU Implementation

To accelerate the generation of the *dynamic proxy*, we approach for a GPU implementation of our method. The implementation is sub-divided into three shader programs in OpenGL – a filter shader, a rendering shader and a blending shader. OpenGL version 2.1 and GLSL version 1.2 is used for the implementation. The illustration of the workflow is depicted in Fig. 5. We can see in Fig. 5 that, after the scene is captured by the camera set, each camera data goes, at first, to filter shader, then to render shader and then, they are blended at the blender shader and finally, the *dynamic proxy* for the scene is generated. A brief description of the functionalities of the shaders is given below.

Filter Shader. The filter shader is used for noise reduction in the depth values. It filters the depth values of the kinects by a 5x5 median filter.

Render Shader. In the render shader, the real point values from the depth values of the kinects are calculated and transferred to a common coordinate system. Also the weighting of each point is performed here. The weight ω and penalty ρ are calculated as in Eq. 3 and Eq. 4; to calculate the angle θ in Eq. 3, we use the following formula:

$$\theta = \cos^{-1}\left(\frac{R_k \cdot R_v}{|R_k| \cdot |R_v|}\right) \cdot \frac{\theta_\tau}{\frac{\pi}{2}} \tag{5}$$

In Eq. 5, θ_τ is the threshold value for angle θ; R_k and R_v are the rays from the kinects and the virtual camera V respectively. θ_τ is 0 if θ is bigger than $\frac{\pi}{2}$, otherwise 1.

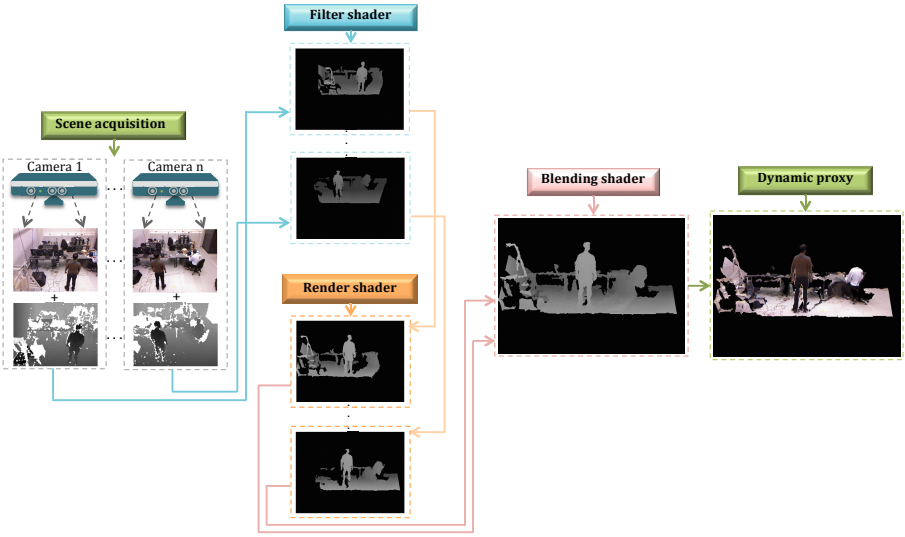


Fig. 5. Illustration of workflow for GPU implementation

Blending Shader. In the blending shader, the depth values in the textures from each camera are blended together based on their weights. For each pixel the weight ω_i, calculated like in Eq. 7, for camera i of n cameras is multiplied with a normalization factor f; f is calculated as in Eq. 6. The depth values Z_i of the n kinects are blended on the basis of the weights ω_i to get final depth Z, as in Eq. 8. The blended depth values are stored in a final resulting texture which represents the *dynamic proxy*.

$$f = \frac{1}{\sum_{i=0}^{n-1} \omega_i} \tag{6}$$

$$\omega_i = \omega_i \cdot f \tag{7}$$

$$Z = \sum_{i=0}^{n-1} Z_i \omega_i \tag{8}$$

6 Results

We are able to achieve real-time rendering speed (29.88 fps) by implementing our idea with the GPU implementation. Table 1 shows the time required for different sectors of the processing pipeline. From table 1, we can see that, each image takes 33 ms to process, in which the rendering takes only 4 ms and most of the time is consumed to transfer camera data from the camera to the system for processing. Fig. 6 shows the 3D rendering of an object (a bag on a chair)

from various view-points inside the test scene. Fig. 7 shows the 3D rendering, via *dynamic proxy*, of a scene with a person and objects in the test environment. Fig. 8 shows the 3D rendering of the test environment; on the top row, it shows a person in different positions, on bottom row, it shows different view-points of a person sitting behind a desk.

Table 1. Time measurement for 1100 images (1024×768 res)

Image per sec.	29.88
Time per image	33 ms
Get camera data to computer	28 ms
Filtering	1 ms
Rendering	4 ms

As a limitation of our system, we observe presence of holes on the represented 3D scene at the intersection of two kinects; this is caused due to the interference problem of multiple kinect projectors. We plan to solve this issue by capturing the image stream for kinects on different time domains; by switching the stream from one kinect to the another very fast. We can also solve this by software based solutions as described in [10]. Since our focus in this paper is on a faster and accurate 3D representation of a scene, we overlook the interference issue and keep it as a future work.

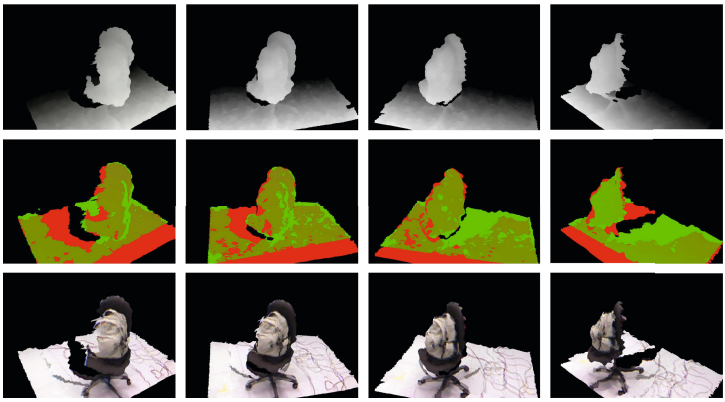


Fig. 6. Top row: depth image (*dynamic proxy*), middle row: camera weighting (for 2 kinects), bottom row: final output with texture

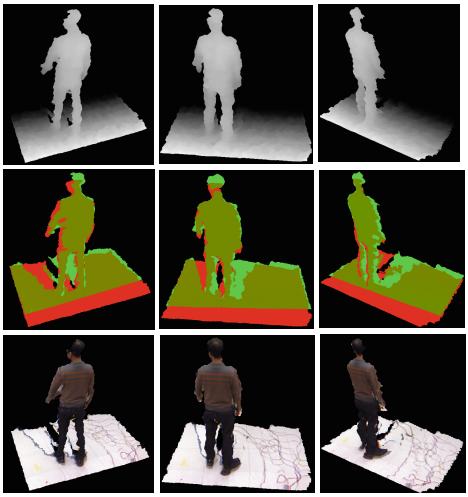


Fig. 7. Top row: *dynamic proxy* (depth image), middle row: camera weighting (for 2 kinects), bottom row: *dynamic proxy* with texture

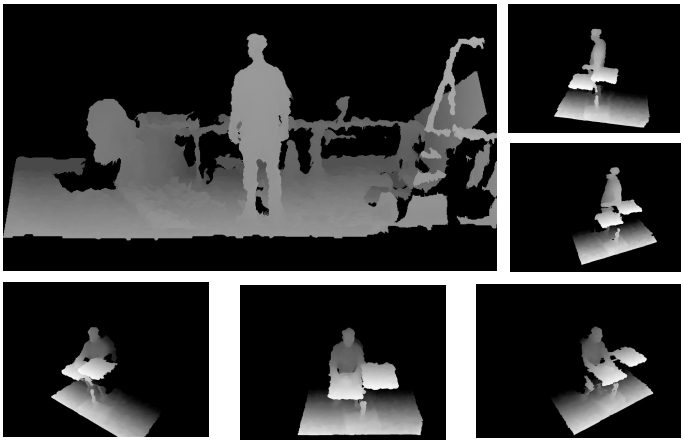


Fig. 8. Rendered depth image (*dynamic proxy*) of the test environment

7 Conclusion

Modern display technology along with availability of cheaper cameras have influenced remote collaboration systems to incorporate large camera array to capture and transmit the recorded data, to be represented as 3D scene on display, to other side of the collaborating environment. Until today, a method providing both real-time reproduction speed and high visual quality is far from achieving. We present an approach for high quality rendering of the 3D representation of a remote collaboration scene along with real-time rendering speed by expanding

the ULR method. We develop a *dynamic proxy* for ULR, to get a better and more detailed 3D proxy in real-time; which leads to a better quality rendering of the 3D scene. Our system also supports motion parallax for the represented 3D scene. Although, we observe some holes at the interference point of two kinects, it can be solved with existing solutions. As a future work, we plan to extend our method for a networked camera system for generating arbitrary view-points for multiple users.

Acknowledgments. This work was supported by the EU FP7 Marie Curie ITN "DIVA" under REA Grant Agreement No. 290227 and by the German Research Foundation (DFG) within the research training group GRK 1424 MuSAMA. We thank Razmik Avetisyan for his valuable suggestions in camera calibration. We would also like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Maimone, A., Bidwell, J., Peng, K., Fuchs, H.: Enhanced personal autostereoscopic telepresence system using commodity depth cameras. Computers & Graphics 36(7), 791–807 (2012)
2. Marcus, R., Alan, R.: Developing Interaction 3D models for e-learning applications online, `http://shura.shu.ac.uk/5306/1/developing_3d_models_for_e-learning_applications.pdf` (last accessed October 30, 2013)
3. Edwards, J.: Telepresence: Virtual Reality in the Real World [Special Reports]. IEEE Signal Processing Magazine 28(6), 9–142 (2011)
4. Petit, B., Lesage, J.-D., Menier, C., Allard, J., Franco, J.-S., Raffin, B., Boyer, E., Faure, F.: Multicamera real-time 3d modeling for telepresence and remote collaboration. International Journal of Digital Multimedia Broadcasting 2010, 247108, 12 pages (2009)
5. Kim, H., Guillemaut, J.-Y., Takai, T., Sarim, M., Hilton, A.: Outdoor Dynamic 3-D Scene Reconstruction. IEEE Transactions on Circuits and Systems for Video Technology 22(11), 1611–1622 (2012)
6. Maimone, A., Fuchs, H.: A first look at a telepresence system with room-sized real-time 3d capture and life-sized tracked display wall. In: ICAT (November 2011)
7. Willert, M., Ohl, S., Staadt, O.G.: Reducing bandwidth consumption in parallel networked telepresence environments. In: VRCAI, pp. 247–254 (2012)
8. Buehler, C., Bosse, M., Mcmillan, L., Gortler, S., Cohen, M.: Unstructured lumigraph rendering. In: Proceedings of the 28th annual Conference on Computer Graphics and Interactive Techniques ACM, S. 425–432 (2001)
9. Avetisyan, R.: Calibration of depth camera arrays (Master thesis). Dept. of computer science, University of Rostock, Germany (2013)
10. Maimone, A., Fuchs, H.: Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras. In: 2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 137–146 (2011)
11. Beck, S., Kunert, A., Kulik, A., Froehlich, B.: Immersive Group-to-Group Telepresence. IEEE Transactions on Visualization and Computer Graphics 19(4), 616–625 (2013)

12. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. In: SIGGRAPH, pp. 43–54 (1996)
13. Shum, H.-Y., He, L.-W.: Rendering with concentric mosaics. In: SIGGRAPH, pp. 299–306 (1999)
14. Debevec, P., Taylor, C., Malik, J.: Modeling and rendering architecture from photographs. In: SIGGRAPH, pp. 11–20 (1996)
15. Eisemann, M., De Decker, B., Magnor, M., Bekaert, P., De Aguiar, E., Ahmed, N., Theobalt, C., Sellent, A.: Floating textures. In: Computer Graphics Forum Bd, vol. 27, S. 409–418. Wiley Online Library (2008)
16. OptiTrack, `http://www.naturalpoint.com/optitrack/` (last accessed January 25, 2014)