Efficient Virtual View Rendering by Merging Pre-rendered RGB-D Data from Multiple Cameras

Yusuke Sasaki
Iwate University
Graduate School of Engineering
Morioka, Japan
h24j063@eecs.iwate-u.ac.jp

Tadahiro Fujimoto
Iwate University
Graduate School of Engineering
Morioka, Japan
fujimoto@cis.iwate-u.ac.jp

***Abstract*—A virtual view, or a free-viewpoint video/image, gives us a video/image of an object seen from an arbitrary viewpoint in a 3D space. One typical approach uses multiview RGB videos captured by multiple RGB cameras surrounding the object. Then, a virtual view is obtained by estimating its 3D shape from the videos. This approach has difficulty in accuracy and efficiency for estimating a 3D geometry from 2D images. Recently, an RGB-D (RGB-Depth) camera is available to capture an RGB video with a depth, which is the distance from the camera to the surface of an object, per pixel. Using an RGB-D camera, the 3D shape of an object surface can be directly obtained without estimating 3D from 2D. However, a single RGB-D camera captures only the 3D shape of the surface part that the camera faces. In this research, we propose a method to efficiently render a virtual view using multiple RGB-D cameras. In our method, the 3D shapes of different surface parts captured by the respective cameras are efficiently merged according to a virtual viewpoint. Each camera is connected to a PC, and all PCs are connected to each other for parallel processing in a PC cluster network. RGB-D data captured by the cameras have to be transferred via the network to merge. Our method effectively reduces the size of RGB-D data to transfer by "view-dependent pre-rendering", in which "imperfect" virtual views are rendered using original RGB-D data captured by the respective cameras on their PCs in parallel. This pre-rendering greatly contributes to real-time rendering of a final virtual view.**

Keywords—virtual view; multiview videos; RGBD(RGB-Depth) camera; PC cluster; real-time rendering;

I. INTRODUCTION

Virtual view rendering is an active research topic in computer graphics and vision, and lots of methods have been proposed. One traditional approach obtains a virtual view of an object by estimating its 3D shape from *multiview RGB videos* captured by multiple RGB cameras surrounding it. This approach has difficulty in accuracy and efficiency for estimating a 3D geometry from 2D images. Instead, an RGB-D camera, such as a Kinect sensor, can directly capture the 3D shape of an object surface without 2D-to-3D estimation. Recently, some methods using *multiview RGB-D videos* have been proposed to render a virtual view by merging different surface parts captured by multiple RGB-D cameras. In such a method, generally, a large amount of RGB-D data captured by multiple cameras have to be transferred via a network, which is a serious bottleneck for real-time virtual view rendering. In this research, we propose a method to efficiently render a virtual view by reducing the size of RGB-D data to transfer via a PC cluster network. This data reduction relies on *view-dependent pre-rendering*, in which an "imperfect" virtual view is rendered to a virtual viewpoint using only an RGB-D data captured by each camera. The resulting pre-rendered virtual view has only object surface parts visible to the viewpoint by eliminating invisible parts. This greatly contributes to the data reduction. The pre-rendering is efficiently executed in parallel on multiple PCs connected to the respective cameras in the PC cluster.

II. RELATED WORK AND FEATURE OF PROPOSED METHOD

A. Related Work

There are some traditional approaches to render a virtual view using multiview RGB videos/images captured by multiple RGB cameras. Depth estimation approach estimates a depth per pixel on captured videos/images or a virtual view to obtain 3D points composing the surface of an object [1-3]. Light field approach constructs a light field from a large amount of reference images to render a virtual view by directly using pixel colors of the images without making a 3D model [4-5]. Object silhouette approach uses the silhouettes of an object on captured videos/images to reconstruct its 3D model [6-9].

Recently, in relation to the above research, *sensor network* is an active research field. In *Wireless Multimedia Sensor Network* (*WMSN*), various kinds of sensors are placed in a wide area to transfer multimedia data in a wireless network. *Visual Sensor Network* (*VSN*) [10] uses visual sensors, such as cameras, to treat visual data, such as images and videos. A technique in this research field needs to efficiently transfer data obtained by sensors via a network. In order to achieve efficient data transfer, the reduction of the size of data is important. In VSN, various methods to compress videos/images have been proposed. For example, some methods reduce the size of data by eliminating redundancy, such as common parts in a scene, displayed in multiple videos/images. Besides, techniques of *Multiview Video Coding* (*MVC*) [11] to efficiently compress multiview videos have been proposed and standardized.

As an example of compression method for multiview RGB videos/images, Chia et al. proposed a compression method to

978-1-5386-2615-3/18/$31.00

eliminate the redundancy in images of adjacent cameras by stitching the images [12]. Bai et al. proposed a method to efficiently determine a common part displayed in different videos by extracting and transferring feature points instead of the videos [13]. The empirical model proposed by Colonnese et al. quantifies the relation between compression ratio of MVC and common parts in different videos [14]. As an example of a compression method of a single depth or RGB-D video/image, Pece et al. tried to apply some standard RGB image compression methods to a depth image to efficiently transfer via a network [15]. The method of Nenci et al. efficiently compress and transfer a depth image using H.264 [16]. The Patent of [17] proposed a method to compress a depth image by Z-lossless compression. Besides, some methods for multiview RGB-D videos/images were proposed. Almazan et al. proposed a surveillance system to track people by multiple RGB-D cameras [18]. The method proposed by Liu et al. compresses and uncompresses multiview depth videos in real time by adjusting compression ratio, image quality, and processing time [19]. The method proposed by Wang et al. efficiently eliminates the redundancy of common parts displayed in two RGB-D videos by using the camera coordinate transformation [20]. Su et al. proposed a real-time 3D telepresence system between local and remote environments by using the network of multiple RGB-D cameras [21].

B. Feature of Proposed Method

When a virtual view is rendered by most existing methods that use multiview RGB or RGB-D videos, basically, the RGB or RGB-D data of the videos captured by all the cameras are transferred via a network to a PC that renders the virtual view. In this case, useless data of object surface parts invisible to the virtual viewpoint are transferred in addition to necessary data of visible parts, even if the redundancy of common parts displayed in different videos is eliminated. For efficiency, our method reduces the size of data to transfer by eliminating the useless data of invisible parts according to the viewpoint. This greatly contributes to real-time rendering of a virtual view.

III. Proposed Method

A. System Overview

Our method efficiently renders a virtual view of an object seen from an arbitrary virtual viewpoint by merging different surface parts of the object captured by multiple RGB-D cameras. In our system, the cameras are placed so as to surround the object. Each camera is connected to a PC, called *camera-PC*. A virtual view is finally rendered on another PC, called *view-PC*. All the PCs are connected to each other in a network to construct a PC cluster. All the cameras are calibrated in advance by directly matching corresponding points on respective camera images. In our experiments, four Microsoft Xbox One Kinect sensors were used as RGB-D cameras.

B. Local RGB-D data

An RGB-D camera captures an RGB-D video having R, G, B colors and a depth, which is the distance from the camera to the surface of an object, per pixel in real time. The camera coordinates (x, y, z) is calculated from pixel coordinates (x_p, y_p) and depth d. Thus, a pixel of pixel coordinates (x_p, y_p) with RGB-D data (r, g, b, d) is equivalent to a 3D point of camera coordinates (x, y, z) with a color (r, g, b). This means that the RGB-D data of each pixel is related to a 3D colored point. Thus, an RGB-D data of a video frame image is a set of 3D colored points composing an object surface visible to the camera. An RGB-D data captured by each camera is called *local RGB-D data*.

C. Pre-rendered RGB-D data

In order to render a virtual view of an arbitrary viewpoint, the local RGB-D data captured by the respective cameras have to be transferred via the PC cluster network from their camera-PCs to the view-PC. One simple approach transfers the local RGB-D data of all the cameras. However, such a large amount of data hinder real-time rendering. For efficient data transfer, we utilize the fact that all the 3D points related to those RGB-D data are not always visible to a viewpoint. The surface part composed by some points is invisible when it is occluded behind the part composed by other points nearer to the viewpoint. Thus, our method reduces the data to transfer by new RGB-D data, called *pre-rendered RGB-D data*, related to 3D points that have possibility to be visible to a viewpoint without occlusion. This data is obtained as follows.

First, a local RGB-D data captured by each camera is used to render its own virtual view to a viewpoint in the same resolution as a final virtual view. This rendering using a single local RGB-D data is called *pre-rendering*, and it is executed on each camera-PC. The resulting virtual view is called *pre-rendered virtual view*. Each pre-rendered virtual view is not perfect, because the surface part that is not captured by the camera is not displayed even if the part is visible to the viewpoint. This means that the pre-rendered virtual view depends on the camera position. Then, a pre-rendered RGB-D data for the virtual view is obtained by reading back R, G, B colors and a depth per pixel from the frame buffer and the depth buffer of the camera-PC. Fig. 1 shows an example of pre-rendering. Among all the 3D points related to an original local RGB-D data, the 3D point nearest to a viewpoint is displayed for each pixel of a pre-rendered virtual view. Conversely, the 3D points occluded by other points are not displayed. This means that a pre-rendered RGB-D data has only the 3D points that have "possibility" to be visible to the viewpoint by eliminating invisible 3D points. Actually, 3D points in a pre-rendered RGB-D data are not always visible to a viewpoint because those points can be occluded by points in other pre-rendered RGB-D data.

One simple way of pre-rendering is point rendering, that is, point splatting that projects 3D points related to a local RGB-D data onto the screen of a pre-rendered virtual view. In this case, it is difficult to determine the optimal size of each point to project. If the size is too large, the resulting virtual view is blurred. If the size is too small, the virtual view is colored sparsely with lots of gaps. To make matters worse, 3D points that are actually invisible to a viewpoint by the occlusion of front points are displayed through the gaps between the front points. This problem does not complete the elimination of invisible 3D points, which are useless in a pre-rendered RGB-D data to transfer. In order to solve this problem, we use polygon rendering by making a triangle mesh from 3D points in a local RGB-D data. It is

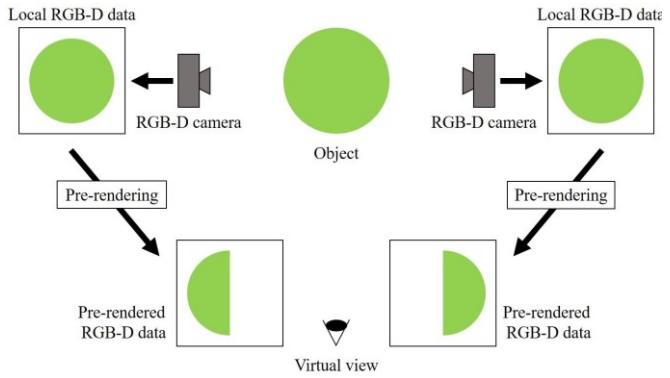

Fig. 1. Pre-rendering.

easy to make triangles from adjacent 3D points because all the 3D points are arranged regularly in the pixel coordinates of the local RGB-D data. A part with a large difference of depth, such as a boundary between two objects, should not be given triangles. Thus, triangles are not made for adjacent 3D points apart from each other by a pre-defined threshold of depth difference. In point rendering, 3D points composing a surface part whose back face faces to a viewpoint can be displayed in a pre-rendered virtual view. This yields useless 3D points in a pre-rendered RGB-D data. In polygon rendering, the back face culling technique displays only the front faces of triangles by preventing back faces from being displayed. This results in removing the useless 3D points.

Pre-rendered RGB-D data of respective cameras are generated on their camera-PCs in parallel using parallel processing by the PC cluster. Thus, all the pre-rendered RGB-D data are efficiently obtained in the execution time of one rendering.

D. Compression of Pre-rendered RGB-D data

The resolution of a pre-rendered virtual view is the same as that of a final virtual view. If a pre-rendered RGB-D data is given RGB-D values (r, g, b, d) of all the pixels of a pre-rendered virtual view as they are, the size of the RGB-D data depends on the resolution of the virtual view. The same applies to an RGB-D video captured by a camera and its local RGB-D data. Then, for example, when an RGB-D video and a final virtual view have the same resolution, a local RGB-D data and a pre-rendered RGB-D data have the same size. In this case, the size of data to transfer is not reduced by pre-rendering. Thus, our method compresses a pre-rendered RGB-D data such that the size of the pre-rendered RGB-D data becomes less than that of its original local RGB-D data after compression.

As described in section II, there are lots of useful compression methods. Currently, we use the following simple method using Run Length Encoding (RLE). This method separates pixels of objects from those of others, that is, background or nothing in a pre-rendered RGB-D data by using a pre-defined threshold of depth in scanline order. When an object pixel is scanned, its RGB-D values (r, g, b, d) are stored. When a non-object pixel is scanned, the number of successive non-object pixels after it is stored. This compression is so effective for an image with a large area of successive non-object pixels.

The above compression method can be applied to not only a pre-rendered RGB-D data but also its original local RGB-D data. A pre-rendered RGB-D data has only front surface parts visible to a viewpoint while its local RGB-D data generally has invisible surface parts, including back surface parts, too. As a result, when cameras are arranged so as to surround an object, the total number of object pixels in pre-rendered RGB-D data of all the cameras is much less than that in their local RGB-D data, as shown in Fig. 1. This means that the effect of compression for a pre-rendered RGB-D data is greater than its local RGB-D data. By compressing pre-rendered RGB-D data, the size of data to transfer is greatly reduced.

E. Rendering of final virtual view

All the compressed pre-rendered RGB-D data generated on the respective camera-PCs are transferred to the view-PC via the PC cluster network and are uncompressed on the view-PC. Each pre-rendered RGB-D data has the same resolution as a final virtual view to render on the view-PC. Thus, for each pixel p in pixel coordinates (x_p, y_p) of the virtual view, the depths of the pixels in (x_p, y_p) of all the pre-rendered RGB-D data are compared. Then, the color of the pixel with the smallest depth is given to the pixel p to complete the virtual view.

By parallel pre-rendering on the multiple camera-PCs, our method reduces the amount of computation to render a virtual view on the view-PC, which includes camera-to-world coordinate transformation and matrix calculation. Our method achieves efficiency by the reduction of not only transferred data but also computation time.

IV. Experimental Result

In our experiments, four Microsoft Xbox One Kinect sensors were used as RGB-D cameras. The cameras were connected to four camera-PCs respectively, and one of these PCs was also used as a view-PC. The spec of each PC is as follows: OS: Windows 10 Home 64 bit, CPU: Intel(R) Core i7-6700 (3.4-4.0GHz), Mem: 8GB, GPU: NVIDIA(R) GeForce GTX 970 4GB. The resolutions of each RGB-D video and a virtual view were the same: 512×424 pixels. The next four options were used. Option 1 did not use our method and the local RGB-D data of all the cameras were transferred. Options 2, 3, and 4 used our method. Option 2 pre-rendered the local RGB-D data of each camera by point rendering. Option 3 made a triangle mesh and pre-rendered it by polygon rendering for both of the front and back faces of triangles. Option 4 used the back face culling technique to pre-render only the front faces of triangles.

Fig. 2 shows resulting images rendered by option 4. The left four images are pre-rendered virtual views of the respective cameras. The right image is a final virtual view rendered from the four pre-rendered RGB-D data. As shown in Fig.2, each pre-rendered virtual view has only surface parts visible to the virtual viewpoint. The visible parts of all the pre-rendered virtual views are unified to complete the final virtual view.

The effect of pre-rendered RGB-D data was demonstrated by comparing the results of options 1 and 2, as shown in Table I, in which the data size and the transfer time per frame are shown. When RGB-D data were not compressed, the sizes of

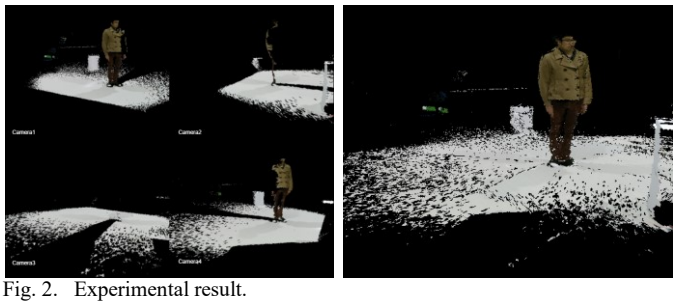
Fig. 2. Experimental result.

data to transfer by both options were the same. This was caused by the reason that the resolutions of each RGB-D video and a final virtual view were the same. As a result, the transfer time and the frame rate to render a final virtual view by both options were also the same. On the other hand, when RGB-D data were compressed, the transfer time by option 2 was greatly reduced in comparison to that by option 1. This was caused by the reason that the data compression for pre-rendered RGB-D data was much more effective than that for local RGB-D data, as described in section III D. This resulted in greatly improving the frame rate.

The effects of different pre-rendering options were demonstrated by comparing the results of options 2, 3 and 4, as shown in Table II. As described in section III C, it is difficult to determine the optimal point size for projection in option 2. In this experiment, a small point size was used. As a result, the virtual view was colored sparsely with lots of gaps and surface parts invisible to a virtual viewpoint were also displayed, although the transfer time and frame rate were better than options 3 and 4. In the virtual views rendered by options 3 and 4, only surface parts visible to a virtual viewpoint were displayed without such gaps. The back face culling technique improved the performance of option 4 in comparison to that of option 3.

V. Conclusion

We proposed a method to efficiently render a virtual view using multiple RGB-D cameras by view-dependent pre-rendering. Some experimental results demonstrated the good performance of our method. As future work, we are planning to develop a more efficient data compression method and extend our system to wireless network.

TABLE I. Comparison between options 1 and 2

opt.	compression	data size (Mbyte)	time (ms)	frame rate (fps)
1	not used	13.9	118 - 119	6 - 7
	used		69 - 73	9 - 11
2	not used	13.9	118 - 119	6 - 7
	used		20 - 22	29 - 30

TABLE II. Comparison of options 2, 3, and 4

opt.	data size (Mbyte)	time (ms)	frame rate (fps)
2	2.1 - 2.5	20 - 25	20 - 22
3	4.7 - 5.0	45 - 50	12 - 13
4	3.9 - 4.0	35 - 40	14 - 15

References

[1] S. E. Chen, L. Williams, View Interpolation for Image Synthesis, Proc. of SIGGRAPH 1993, pp.279-288, 1993.

[2] L. McMillan, G. Bishop, Plenoptic Modeling: An Image-Based Rendering System, Proc. of SIGGRAPH 1995, pp.39-46, 1995.

[3] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, R. Szeliski, High-quality Video View Interpolation Using a Layered Representation, Proc. of SIGGRAPH 2004, pp.600-608, 2004.

[4] M. Levoy, P. Hanrahan, Light Field Rendering, Proc. of SIGGRAPH 1996, pp.31-42, 1996.

[5] S. J. Gortler, R. Grzeszczuk, R. Szeliski, M. F. Cohen, The Lumigraph, Proc. of SIGGRAPH 1996, pp.43-54, 1996.

[6] A. Laurentini, The Visual Hull Concept for Silhouette-based Image Understanding, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.16, No.2, pp.150-162, 1994.

[7] W. Matusik, C. Buehler, R. Raskar, S. Gortler, L. McMillan, Image Based Visual Hulls, Proc. of SIGGRAPH 2000, pp.369-374, 2000.

[8] W. Matusik, C. Buehler, L. McMillan, Polyhedral Visual Hulls for Real-Time Rendering, Proc. of Eurographics Workshop on Rendering 2001, pp.115-125, 2001.

[9] S. M. Seitz, C. R. Dyer, Photorealistic Scene Reconstruction by Voxel Coloring, Computer Vision and Pattern Recognition Conf., pp.1067-1073, 1997.

[10] A. Mammeri, B. Hadjou, A. Khoumsi, A Survey of Image Compression Algorithms for Visual Sensor Networks, International Scholarly Research Network (ISRN), Sensor Networks, vol. 2012, 760320, pp. 1-19, 2012.

[11] A. Vetro, T. Wiegand, G. J. Sullivan, Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard, Proc. of the IEEE, vol. 99, no. 4, pp. 626-642, 2011.

[12] W. C. Chia, L. Ang, K. P. Seng, Multiview Image Compression for Wireless Multimedia Sensor Network using Image Stitching and SPIHT Coding with EZW Tree Structure, Proc. of the International Conf. on Intelligent Human-Machine Systems and Cybernetics (IHMSC) 2009, vol. 2, pp. 298-301, 2009.

[13] Y. Bai, H. Qi, Feature-Based Image Comparison for Semantic Neighbor Selection in Resource-Constrained Visual Sensor Networks, EURASIP Journal on Image and Video Processing, vol. 2010, 469563, pp. 1-11, 2010.

[14] S. Colonnese, F. Cuomo, T. Melodia, An Empirical Model of Multiview Video Coding Efficiency for Wireless Multimedia Sensor Networks, IEEE Trans. on Multimedia, vol. 15, no. 8, pp. 1800-1814, 2013.

[15] F. Pece, J. Kautz, T. Weyrich, Adapting Standard Video Codecs for Depth Streaming, Proc. of the 17th Eurographics Conf. on Virtual Environments Joint Virtual Reality (EGVE-JVRC'11), pp. 59-66, 2011.

[16] F. Nenci, L. Spinello, C. Stachniss, Effective Compression of Range Data Streams for Remote Robot Operations using H.264, Proc. of the International Conf. on Intelligent Robots and Systems (IROS), 2014.

[17] Real-time Lossless Compression of Depth Streams, United States Patent Application 20170237996 A1, 2017.

[18] E. J. Almazan, G. A. Jones, Tracking People across Multiple Non-Overlapping RGB-D Sensors, Proc. of the 2013 IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW 2013), pp. 831-837, 2013.

[19] Y. Liu, S. Beck, R. Wang, J. Li, H. Xu, S. Yao, X. Tong, B. Froehlich, Hybrid Lossless-Lossy Compression for Real-Time Depth-Sensor Streams in 3D Telepresence Applications, 16th Pacific-Rim Conf. on Multimedia, Advances in Multimedia Information Processing, pp. 442-452, 2015.

[20] X. Wang, Y. A. Sekercioglu, T. Drummond, E. Natalizio, I. Fantoni, V. Fremont, Collaborative Multi-Sensor Image Transmission and Data Fusion in Mobile Visual Sensor Networks Equipped with RGB-D Cameras, IEEE International Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI) 2016, 2016.

[21] P. Su, J. Shen, M. U. Rafique, RGB-D Camera Network Calibration and Streaming for 3D Telepresence in Large Environment, IEEE Third International Conf. on Multimedia Big Data (BigMM), pp. 362-369, 2017.