Real-Time Multi-View Depth Map Generation Based on Guided Image and Parallel Processing

Sunho Kim and Yo-Sung Ho
School of Electrical Engineering and Computer Science
Gwangju Institute of Science and Technology (GIST)
Gwangju, Republic of Korea
{sunhokim, hoyo}@gist.ac.kr

***Abstract*--In this paper, we propose a real-time multi-view depth generation method using a fast upsampling technique and parallel programming. To upsample input depth maps, we use guided image filter to improve computational speed and make a similar quality of depth map compared to conventional methods. To transfer the memory between CPU and GPU, we apply the OpenMP and CUDA for the parallel programming using both CPU and GPU. Experimental results show that our method can improve computational speed, and remain the quality of high-definition depth map.**

***Index Terms*—Depth upsampling, Parallel programming, Guided image filter, CUDA, OpenMP**

I. Introduction

Research on image processing for creating realistic 3D contents is still going on today. Moreover, as the fourth industrial revolution came into being, the technological advancement in the field of information and communication has been demanded, and the interest in more realistic video contents such as augmented reality and virtual reality and the technology to support it has been amplified. Accordingly, research has been continuing from technologies capable of rapidly generating 3D contents using high-quality images to technologies of producing high-quality 3D objects.

In this paper, we propose a real-time depth map generation method using high-definition quality multi-view color images. In order to get high-resolution depth maps, depth values from the low-resolution depth map captured by time-of-flight (ToF) camera are warped to another color spaces corresponding to the viewpoints of the respective color images first, then we refine the displaced images into smooth depth maps. To do this, we applied the guided image filtering method to the depth upsampling process.

But it takes a lot of computational time to process a high-resolution image using a single processor of the CPU. So we made use of CUDA library and OpenMP API to take advantage of parallel processing both CPUs and GPUs. From the experimental results, we show that our proposed method can generate a multi-view high-definition depth map faster than conventional methods.

This work was supported by the 'Civil-Military Technology Cooperation Program' grant funded by the Korea government

II. Real-Time Multi-View Depth Map Generation

The overall process of depth map generation is shown in figure 1. In the pre-processing, camera intrinsic and extrinsic parameters are acquired through camera calibration, and these parameters are used for depth upsampling process.

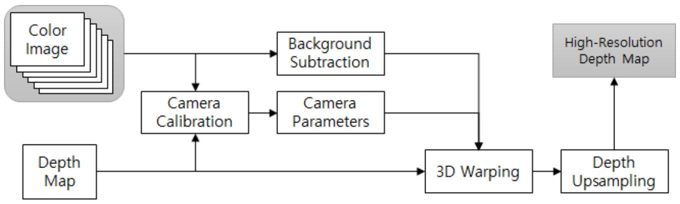


Fig. 1. Pipeline of multi-view depth map generation

A. 3D Warping

To make use of values of the low-resolution depth map in high-resolution color images captured by multiple view cameras, we should upsample the depth map based on the camera's intrinsic and extrinsic parameters. 3D warping is the first procedure of the depth map upsampling. In the proposed method, we changed the color space from the RGB color space to the HSV color space. Then we removed the background of the color image to get the object mask of the color image and warped depth values from the depth map.

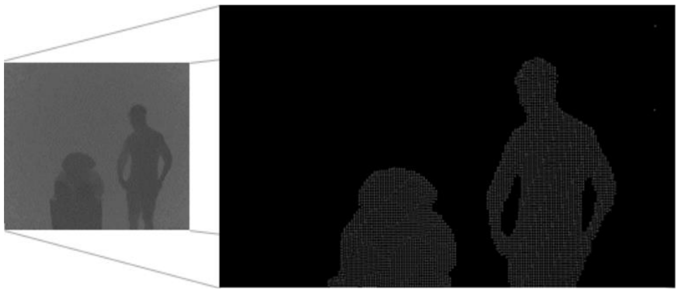


Fig. 2. High-resolution depth map acquired from the 3D warping

In 3D warping, the information of the depth map corresponds to positions corresponding to the viewpoints of the respective color images. Figure 2 shows the result of 3D warping. But there are a lot of holes inside each object because this result is expanded image from the low-resolution to the high-resolution. These holes are filled by using interpolation methods or specific filters.

B. Fast Guided Image Upsampling

To remove noises or upsample the image, various types of filters were utilized. gaussian filter, joint bilateral filter, and a noise-aware filter for depth upsampling (NAFDU) use one or more kernels based on gaussian distribution to fill the noises such as holes. In particular, the joint bilateral filter can preserve the edge and has a simple algorithm. But there is a slight distortion in near the gradient values and computational speed is slow although the algorithm is simple.

To overcome these problems, guided image filter [1] is proposed. Guided image filter can preserve the edge region, solve the distortion near the gradient value, and improve the computational speed. In [1], they said that the time complexity of the guided image filter is O(1). In this method, the guide image is used to refine the input image. Here, the guide image may be the input image itself or another image may be used according to the purpose.

$$a = \frac{\mathrm{cov}(I, p)}{\mathrm{var}(I) + \varepsilon} \tag{1}$$

$$b = E[p] - aE[I] \tag{2}$$

$$q = IE[a] + E[b] \tag{3}$$

Equation (1) to (3) shows all the formulas used in guided image filtering, where I is guide image, p is input image, and q is final output. ε is user-defined. Unlike joint bilateral filter and NAFDU, which use a time-consuming calculation based on the gaussian distribution in each kernel, guided image filter only required mean, variance, and covariance value in each kernel, each step. The size of the kernel is user-defined.

First, we compute the covariance of the guide image and input image and the variance of the guide image. Then, add the small ε value for the normalization. Based on this normalized variance, we divide the covariance value to get the value *a* like (1). Second, we compute the mean of guide image and input image respectively. Based on two mean values and value *a*, we can get the value *b* using (2). The mean value of the input image is subtracted from the multiplied value between *a* and the mean of the guide image. Finally, we compute the value q by adding the multiplied value between the mean of the value *a* and the guide image to the mean of the value *b*.

Fast guided image filter [2] improves the speed compared by previous guided image filter while maintaining the performance as possible. A feature of the fast guided image filter is that it resizes the input and result values before and after the filtering process. First, we downsample the guide image and the input image, then compute the value *a* from (1) and *b* from (2). After that, we compute the mean of *a* and *b* and upsample these two mean values. Finally, we can get the value *q* from (3) using the upsampled two mean values. This method has an advantage that reduces the amount of computation. Nevertheless, we can get the similar upsampled result to the result of guided image filter. Figure 3 (c) is the result of guided image filter and (d) is the result of fast guided image filter. In figure 3 (d), we subsampled input image and guide image to 1/2 times. The computation time can reduce and the result of the upsampled depth map is similar to the (c). Of course, when we subsample the input image and guide image under 1/4 times, the result has some noise and blurred regions.

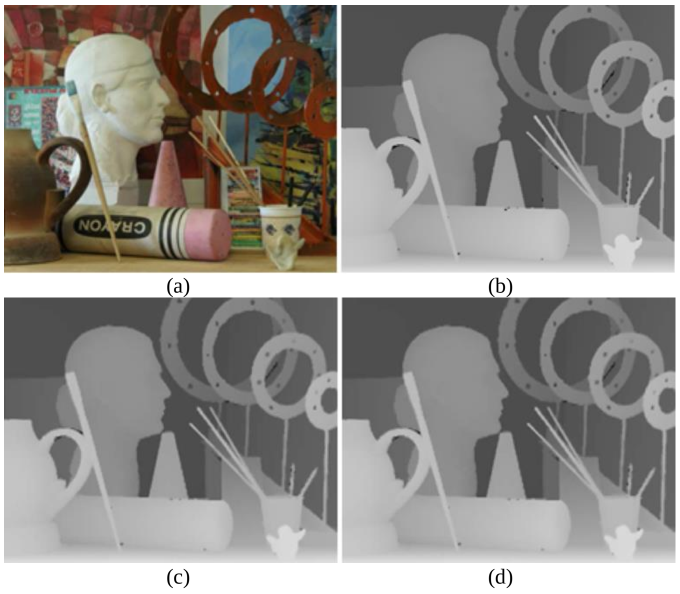


Fig. 3. Upsampling results using two filters (a) high-resolution input color image (b) ground truth of high-resolution depth map (c) depth map upsampling result using guided image filter (d) result using fast guided filter, which uses subsampled input to 1/2 times

C. CUDA GPU Programming

Recently, GPU-based processing has continued to be demanded in many types of research requiring a high computational cost such as deep learning. Compute unified device architecture (CUDA) [3] enables parallel processing based on one or more GPUs. CUDA, based on C programming language, helps us to access the GPU and perform faster programming without any knowledge of GPU API. Basically, CUDA can be applied C or C++ language. However, some libraries have come to be able to apply it to other languages recently such as Java and Python.

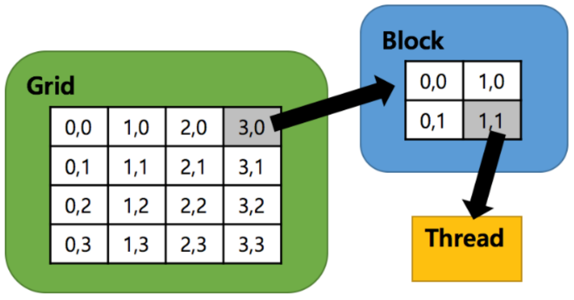


Fig. 4. Basic unit of operation of GPU programming.

Figure 4 represents the unit of operation of CUDA GPU programming. There are a lot of threads in single GPU. And block consists of several threads, grid consists of several blocks.

Generally, CUDA uses this grid as a basic unit of processing. The number of threads in GPU is more than in CPU. So, in image processing, when we proceed the program in parallel, it is more efficient to compute in GPU, because there are so many pixel values in high-resolution images.

In order to compute the program through the GPU, we have to transfer the memories from the CPU. As shown in figure 5 (a), however, conventional memory transfer process runs synchronously. Therefore, the memory should wait until the previous memory was returned to CPU right after the operation in GPU was finished. Our approach uses asynchronous memory transfer system like figure 5 (b) to reduce the transmission time. Once a memory is transferred from CPU to GPU, depth map upsampling process is performed in GPU. In the proposed method, unlike the synchronous memory transfer system, the next memory is transferred from CPU to GPU simultaneously, although the upsampling process is still performing. Transferred another memory is allocated to another empty thread, performs the depth map upsampling process, and the next memory is also transferred sequentially.

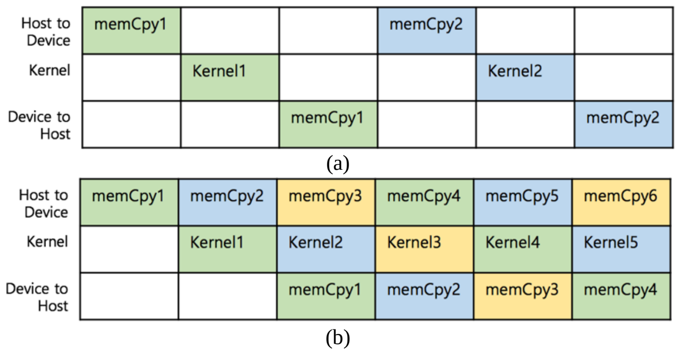


Fig. 5. Memory transfer process (a) synchronous memory transfer system (b) asynchronous memory transfer system.

D. OpenMP API for Parallel Programming

OpenMP [4], designed for multi-processing in shared memory, is an API that can be used for parallel programming in programming languages such as C, C++, and Fortran. In OpenMP, we can share the task among the threads and perform synchronization and communication through the compiler directive command. Figure 6 shows the fork-join model for multi-threading in OpenMP.

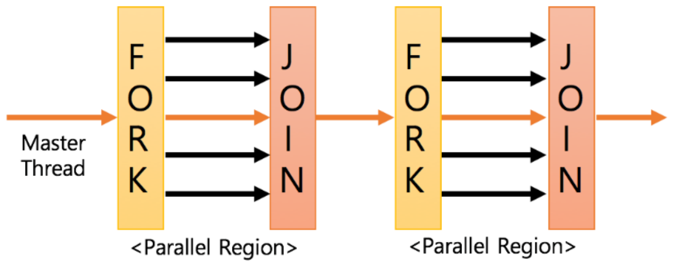
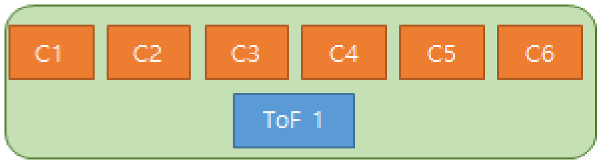


Fig. 6. Fork-join model for multi-thread.

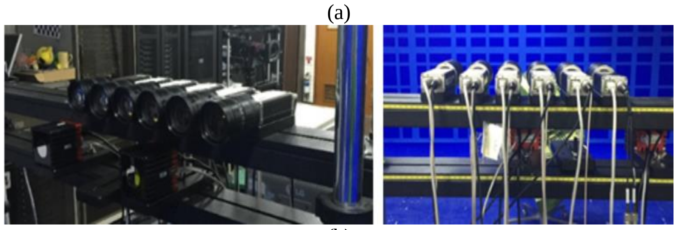
OpenMP can apply parallelism in source code explicitly. When we insert a compiler directive command at a desired position in the source code, it is possible to perform parallel processing of the multi-thread at the corresponding position. Generally, it is often used in a loop that requires a large amount of computation, such as for loop. In OpenMP, the operation of each loop is distributed to multiple threads, which reduces the computation time. We also applied this API to transfer multi-view color images and depth maps in parallel. This allowed us to reduce the memory transmission time.

III. Experimental Results

For the experiment of the proposed method, we set the capturing environment as shown in figure 7. Using 6 color cameras, we captured multiple view high-resolution color images, and using the ToF camera, we captured a low-resolution depth map. After that, we perform the camera calibration process to compute intrinsic and extrinsic parameters of each camera, remove the background information to remain the object information only, perform the 3D warping process in low-resolution depth map based on camera parameters acquired from camera calibration, and apply our depth map upsampling method.



(a)



(b)

Fig. 7. Multi-view color and depth image capturing environment. (a) the layout of camera alignment, (b) capturing environment of studio.

Figure 8 shows the final result of depth map upsampling. The result can be acquired from each viewpoint of the color camera. Each image, we captured one person, two people, and two people with doll respectively. In order to show that the proposed method can successfully perform not only for one viewpoint but also for various viewpoints, we included the results for two viewpoints in this paper.

TABLE I
Comparison of Real-Time Multi-View Depth Map Generation Speed

	One person	Two people	Two people with doll
Circular Queue-based [5]	30.293 fps	27.882 fps	22.035 fps
CUDA + Gaussian Filter [6]	34.881 fps	31.228 fps	27.232 fps
CUDA + Gaussian Filter + down-sampled object [6]	36.339 fps	33.001 fps	31.533 fps
Proposed method	48.012 fps	45.212 fps	43.355 fps

To compare the performance, we use three conventional methods. First method [5] uses a circular queue to manage the memory, second method [6] uses CUDA library and gaussian filter to upsample the depth map. And the third method [6] added downsampling process in the second method to resize the input color image because they want to reduce the amount of data transferred from the CPU to the GPU. After the transmission, the color image is up-scaled to the original size. Table 1 shows the comparison results of computation speed. Conventional three methods use the gaussian filter to upsample the depth map and second and third methods use CUDA library only for parallel processing. As shown in table I, our method is at least 12 fps faster than conventional methods, because our method reduces both memory transmission time and upsampling time.

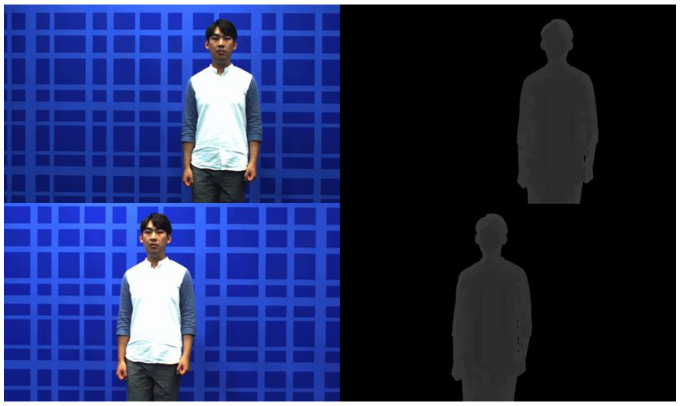
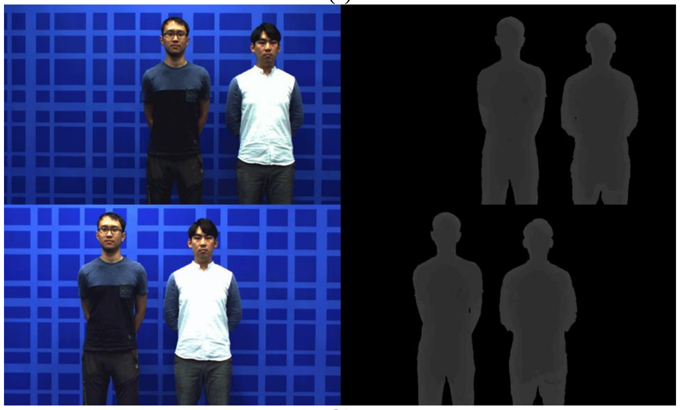
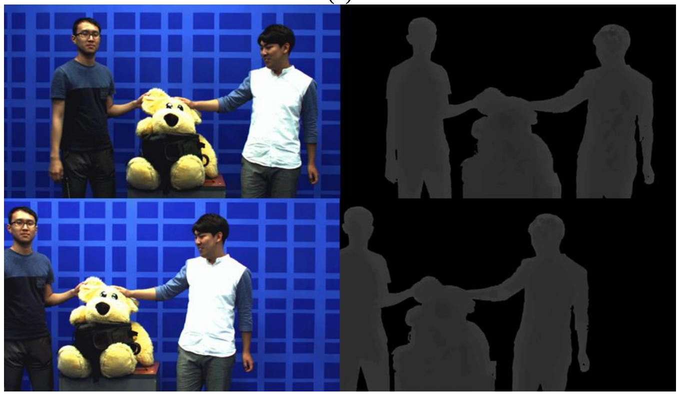

(a)

(b)

(c)

Fig. 8. Results of real-time multi-view depth map generation. Results shows upsampled depth maps for two viewpoints. Each color images capture (a) one person, (b) two people, and (c) two people with doll.

IV. Conclusion

In this paper, we proposed real-time multi-view depth generation method., Our upsampling algorithm is based on the guided image filtering, faster than other filters based on gaussian distribution. And we used CUDA library for the GPU parallel processing and OpenMP API for the CPU parallel processing. Through this, we could get high-resolution depth map in real-time. In experimental results, our method can get high-resolution depth maps faster than other conventional methods. Our method can generate more depth maps 12-18 frames per seconds than conventional methods. Based on these results, we prove that the proposed parallel programming method and upsampling algorithm have fast computation speed.

References

[1] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Trans. Pattern Analysis and Machine Intelligence. 35(6),* pp. 1397-1409, June 2013.

[2] K. He and J. Sun, "Fast guided filter," *arXiv preprint arXiv:1505.00996*, May 2015.

[3] R. Farber, "CUDA application design and development," Elsevier, pp. 8-10, Oct. 2011.

[4] http://www.openmp.org

[5] E. Ko, Y. Song, and Y. S. Ho, "Real-time multi-view depth generation using CUDA multi-GPU," International Conference on Embedded Systems and Intelligent Technology, pp. 114-116, Sep. 2014.

[6] S. Kim and Y. S. Ho, "Real-time multi-view depth map generation using high-definition image downsampling," Spring conference 2017 of KISM & SEBS, pp. 145-148, Apr. 2017.

Sunho Kim received his B.S. degree in computer engineering from Hanbat National University, Korea in 2015. He is currently working towards his M.S. degree in the school of electrical engineering and computer science at Gwangju Institute of Science and Technology (GIST), Korea. His research interests include 3D computer vision, computer graphics, 3D scene reconstruction, simultaneous localization and mapping (SLAM), and augmented reality (AR).

Yo-Sung Ho received the B.S. and M.S. degrees in electronic engineering from Seoul National University, Seoul, Korea, in 1981 and 1983, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, Santa Barbara, in 1990. He joined the Electronics and Tele-communications Research Institute (ETRI), Daejeon, Korea, in 1983. From 1990 to 1993, he was with Philips Laboratories, Briarcliff Manor, NY, where he was involved in development of the advanced digital high-definition television system. In 1993, he rejoined the Technical Staff of ETRI and was involved in development of the Korea direct broadcast satellite digital television and high-definition television systems. Since 1995, he has been with the Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, where he is currently a Professor in the School of Electrical Engineering and Computer Science. Since August 2003, he has been Director of Realistic Broadcasting Research Center at GIST in Korea. His research interests include digital image and video coding, image analysis and image restoration, advanced coding techniques, digital video and audio broadcasting, 3- D television, and realistic broadcasting.