Real-time Depth Image Refinement using Hierarchical Joint Bilateral Filter

Dong-Won Shin, Sang-Beom Lee, and Yo-Sung Ho
School of Information and Communications
Gwangju Institute of Science and Technology (GIST)
123 Cheomdan-gwagiro, Buk-gu, Gwangju 500-712 Korea
Email: {dongwonsin, sblee, hoyo}@gist.ac.kr

Abstract—In this paper, we propose a real-time depth image refinement method. In order to improve the quality of the depth map acquired from Kinect camera, we employ constant and texture memories which are suitable for the 2D image processing in graphics processing unit (GPU). In addition, we execute joint bilateral filter (JBF) in parallel to accelerate the overall execution speed. To enhance the quality of the depth image, we apply the JBF hierarchically using the compute unified device architecture (CUDA). Finally, we obtain the refined depth image. The experimental results show that the proposed refinement algorithm improves the quality of the depth image and the computational time is 200 frames per second.

Keywords—depth image, refinement, joint bilateral filter, CUDA, hierarchical approach

I. INTRODUCTION

Currently, the three dimensional (3D) video system has taken the center stage as one of the most important next-generation broadcasting systems along with ultra-high definition TV (UHDTV) and internet protocol TV (IPTV) systems. The 3D video system is composed of numerous major components such as 3D video acquisition, content production, encoding, view synthesis, and rendering for display. 3D video contents can be developed by various content production tools such as stereo cameras, multi-view cameras and depth cameras. These contents are encoded by 3D video encoding technique and transmitted to the receiver. At the receiver end, the 3D content can be rendered by the depth image-based rendering (DIBR) technique [1]. The broadcasting service using DIBR technique takes very few traffics because it uses single color and depth images which are more efficient for compressibility than color images. Moreover, it can maintain an inverse compatibility with the existing system because it provides the 3D scene using only single depth and color images, but not multiple color images. Therefore, it allows user devices to convert from 2D to 3D service or vice versa [2]. Recently, thanks to the release of Microsoft Kinect, we can access 3D video easily even in home environment. The DIBR technique reconstructs the 3D scene by using color and depth images. However, as we can see in Fig. 1, the depth image from structured depth camera like a Kinect depth camera includes substantial holes where depth values do not exist. These are due to the different positions of the transmitter and the receiver of the infrared laser. This error causes the low quality of the reconstructed 3D scene. To eliminate the depth error, many researchers have studied various methods. Among them, the joint bilateral filter (JBF) is considered as one of the most efficient methods [3]. This employee spatial and range filters which reflect differences between target pixel and neighbor pixels. The range filter eliminates boundary mismatches between two images and simultaneously refine depth errors.

![Fig. 1 Image from Kinect camera](a) Color image (b) Depth image

In this paper, we propose the real-time refinement method for depth images. The proposed method applies JBF to the depth image acquired from Kinect depth camera. Because this filter includes various factors and complex equations like Gaussian function, we use CUDA parallel programming for the real-time computation. Furthermore, in order to fill occlusion areas, we apply JBF hierarchically. We make layers depending on the resolution and employee JBF from the bottom to the top layers. Finally, we acquire the depth image that occlusion areas filled with respectable depth values.

II. JOINT BILATERAL FILTER USING CUDA

Although the procedural programming using the central processing unit (CPU) was widely used, parallel programming using the graphic processing unit (GPU) is currently getting popular. Especially, distributing CUDA based on C language from NVIDIA is highly contributing the computational time reduction [4].

In this paper, we refine a depth image acquired from Kinect camera by using hierarchical joint bilateral filter as a real-time. As a first, we explain a single joint bilateral filter in this chapter. Figure 2 shows that a flowchart of joint bilateral filter using CUDA.
First of all, we load depth and color images from Kinect to the texture memory in CUDA which is properly designed for 2D image processing. Next, we operate the joint bilateral filter on the images in parallel and accelerate the computational speed by operating worker threads as many as the number of pixels in the image. Moreover, in order to reduce the operation time, we calculate some weighting factors before the kernel function is operated. After that, we use the calculated values when we need. These weighting factors are stored in the constant memory; it is relatively faster than any other memory in GPU. When we calculate weighting factors, the operation amount dramatically increases depending on the kernel size. Fortunately, this kind of 2D Gaussian function can be divided as a multiplication of 1D Gaussian function. So we implement a fast filter by using this [5]. Equation (1) shows the detail of joint bilateral filter.

$$D_u(x,y) = \frac{\sum_{u \in U_y, v \in V_y} W(u,v)D_v(u,v)}{\sum_{u \in \mathbb{U}_y, v \in \mathbb{V}_y} W(u,v)}$$

(1)

In this equation, $D_u$ means an input depth image and $W$ is the weighting factor. We calculate this equation on each pixel and get $D_u$ through the normalization which divide as a total sum of the weighting factor. The symbol $u$ and $v$ are elements of set $\mathbb{U}_P$ and $\mathbb{V}_P$ representing the specific position in filter kernel. Therefore, we can represent like (2) and (3).

$$\mathbb{U}_P = \{x - r, \cdots , x + r\}$$

(2)

$$\mathbb{V}_P = \{y - r, \cdots , y + r\}$$

(3)

Figure 3 shows symbols in (2) and (3). The black grid pattern represents pixels in the image and the rectangular with the thick red line represents the filter kernel. In the filter kernel, $(u, v)$ means a position of the neighbor pixels and $(x, y)$ means a position of the center pixel. Symbol $r$ represents the radius of the filter kernel. Vector $\mathbf{u}_P$ and $\mathbf{v}_P$ means the horizontal and vertical position.

![Joint Bilateral Filter Diagram](image)

**Fig. 3 Description on a symbol**

The weighting factor is defined as (4).

$$W(u,v) = \begin{cases} 0 & \text{if } D(u,v) = 0 \\ g(u,v) \cdot f(u,v) & \text{otherwise} \end{cases}$$

(4)

Function $f(u,v)$ represents the spatial filter depending on the distance between the center pixel and the neighbor pixel. Function $g(u,v)$ represents the range filter depending on color differences. Both two functions are defined by Gaussian function like (5) and (6).

$$g(u,v) = \exp\left(-\frac{|I(x,y)-I(u,v)|^2}{2\sigma_R^2}\right)$$

(5)

$$f(u,v) = \exp\left(-\frac{(x-u)^2 + (y-v)^2}{2\sigma_D^2}\right)$$

(6)

In this equation, $\sigma_R$ and $\sigma_D$ means the standard deviation for Gaussian function. Depending on this value, the width of Gaussian distribution is determined.
III. HIERARCHICAL JOINT BILATERAL FILTER

We cannot refine all holes in the original depth image from Kinect as we operate just once. There are some areas which its boundary is not clear or just filled with the same depth value although those are different areas. In order to fill occlusion areas having no depth values, we employ JBF hierarchically [4, 5]. Figure 4 shows the diagram of hierarchical JBF.

Layer 1 has original color and depth images. If the number becomes low as going down to the lower layer. The image size becomes small as going down to the lower layer, however the size of JBF kernel is same in all layers. In our experiment, we limited the level of lower layer up to five. Figure 5 shows the process of the image refinement in detail.

First of all, we make color and depth images in each layer and perform JBF by using color and depth images in the lowest layer. We copy these depth values to the occlusion area of the upper layer’s depth image that has no depth values. Because the image size of upper layer is two times larger than that of the lower layer, the occlusion area in the depth image of the upper layer is filled with a pixel value of lower layer every three pixels. By using the updated depth and the related color images, we perform the JBF again. Finally, as repeating this iteration, we can get the final depth image which most of its occlusion areas are filled with proper depth values.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In order to evaluate the performance of the proposed method, we used GeForce GTX Titan graphics card and Intel Xeon CPU 2.53GHz. Filter radius and Gaussian delta are five and Euclidean delta is three. First, we compared the visual quality of the original depth and the refined depth images. Figure 6 (a) shows the original depth image and Fig. 6 (b) shows the refined depth image by using JBF just one time. It shows a clear distinction between original depth image and refined depth image so we can find it with the naked eye.

In the original depth image, there are some areas which has no depth values around the object boundary. Moreover, there are some hole areas on the table because it is flat. However, we can find more clear depth values than original one in the refined depth image.
Fig. 6 Visual comparison of depth images

Next, we show the refined depth images by using Intel Xeon CPU (2.53GHz) and GeForce GTX Titan GPU respectively in Fig. 7. In this experiment, we consider performing JBF once as a criteria and deduce result by using CPU and GPU.

Fig. 7 Depth map computed by using GPU

Each image on the right side is an enlarged image. There were no significant difference between the depth image by CPU and the depth image by GPU. However, in the aspect of the time complexity, there was a significant difference between the two conditions. We can find that using GPU programming is remarkably faster than using CPU. Table 1 shows the running time for various image sequences. I use image sequences from middlebury website [6]. In the case of GPU, it shortened the running time up to 99% in comparison with using CPU.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>CPU(ms)</th>
<th>GPU(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsukuba</td>
<td>1016</td>
<td>4.12</td>
</tr>
<tr>
<td>(384x288)</td>
<td>1007</td>
<td>4.25</td>
</tr>
<tr>
<td></td>
<td>1016</td>
<td>4.22</td>
</tr>
<tr>
<td>venus</td>
<td>1423</td>
<td>5.76</td>
</tr>
<tr>
<td>(434x383)</td>
<td>1521</td>
<td>5.99</td>
</tr>
<tr>
<td></td>
<td>1575</td>
<td>5.99</td>
</tr>
<tr>
<td>cones</td>
<td>1685</td>
<td>6.13</td>
</tr>
<tr>
<td>(450x375)</td>
<td>1877</td>
<td>5.73</td>
</tr>
<tr>
<td></td>
<td>1692</td>
<td>6.41</td>
</tr>
<tr>
<td>teddy</td>
<td>1531</td>
<td>6.35</td>
</tr>
</tbody>
</table>

Lastly, I compared the bad pixel rate of each depth image. Table 2 shows the result of bad pixel rate.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Original depth (%)</th>
<th>JBF once (%)</th>
<th>Proposed method (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsukuba</td>
<td>8.90</td>
<td>7.50</td>
<td>7.42</td>
</tr>
<tr>
<td>venus</td>
<td>12.02</td>
<td>10.60</td>
<td>10.55</td>
</tr>
<tr>
<td>cones</td>
<td>16.39</td>
<td>16.87</td>
<td>16.88</td>
</tr>
<tr>
<td>teddy</td>
<td>8.00</td>
<td>6.29</td>
<td>6.13</td>
</tr>
</tbody>
</table>

In Table 2, bad pixel rate of the proposed depth image is lower than that of the original depth image. However, in case of cones sequence, it shows a worse result. Therefore, the proposed method shows better depth quality in the overall cases.

V. CONCLUSION

In this paper, we proposed the depth refinement method using JBF via GPU programming. Also, in order to improve the accuracy, we employed hierarchical joint bilateral filter depending on the image size. In the aspect of time complexity, we can find that GPU programming implementation is remarkably faster than CPU based implementation. Depending on the number of layers, it shows better quality in the bad pixel rate. The proposed method achieved about 200 frames per second on average.

ACKNOWLEDGMENT

This research was supported by the ‘Cross-Ministry Giga KOREA Project’ of the Ministry of Science, ICT and Future Planning, Republic of Korea (ROK). [GK13C0100, Development of Interactive and Realistic Massive Giga-Content Technology]

REFERENCES