© 2009 Quang Huu Nguyen

THREE-DIMENSIONAL PROPAGATION ALGORITHM FOR DEPTH IMAGE-BASED RENDERING

BY

QUANG HUU NGUYEN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Adviser:

Associate Professor Minh N. Do

ABSTRACT

Depth image based rendering (DIBR) is the process of synthesizing new "virtual" views from a set of "real" views which include color and depth images. Because of its photorealism and less stringent computational requirement, DIBR has many applications such as 3D TV, remote reality, and video conferencing, and has become one of the hot research areas in visual computing in recent years. Since the general purpose graphics processing unit (GPGPU) is an ideal computing platform for image rendering, we consciously develop a novel and necessary image processing algorithm suitable for GPGPU by exploiting massive parallelism. The proposed 3D propagation algorithm for DIBR combines images from multiple color and depth cameras at arbitrary positions in 3D space and efficiently renders novel images at arbitrary virtual views by propagating all available depth information from depth cameras to color cameras, and then all available depth and color information from the color cameras to the virtual views. Furthermore, we consider the case when only low resolution depth images are obtained. A novel depth filling and enhancement technique for enhancing depth image quality using high resolution color images is proposed and significantly improves the rendering quality. Finally, the paper also describes the abundant but irregular parallelism of our algorithm and outlines a mapping onto massively parallel architectures such as GPGPUs.

To my family, for their love and support

ACKNOWLEDGMENTS

I would like to thank my adviser, Professor Minh N. Do, and my co-adviser, Professor Sanjay J. Patel, for their enthusiastic support, precious advice and guidance. I am grateful to my labmates and friends at UIUC for their enlightening discussions, assistance and friendship. Finally, I would like to dedicate this thesis to my family: my parents and sister for their unlimited support and encouragement, my wife, Nguyen, for her love and sacrifice, and my cute daughter, Thuy Minh, for giving me more strength.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IBR	Image-based rendering
DIBR	Depth image-based rendering
GPGPU	General purpose graphics processing unit

CHAPTER 1

INTRODUCTION

1.1 Depth image-based rendering

Image-based rendering (IBR) is the process of synthesizing novel views from pre-rendered or pre-acquired reference images of a 3D scene. Obviating the need to create a full geometric 3D model, IBR is relatively inexpensive compared to traditional rendering while still providing high photorealism. Because its rendering time is independent of the geometrical and physical complexity of the scene being rendered, IBR is extremely useful for efficient rendering of both real scenes and complex synthetic 3D scenes. Therefore, IBR has recently attracted a lot of research interest. Its applications can be found in many areas such as 3D TV, free-viewpoint TV, telepresence, video conferencing, and computer graphics [1, 2, 3].

Depth image-based rendering (DIBR) combines color images with per-pixel depth information of the scene to synthesize novel views. Depth information can be obtained by stereo match or depth estimation algorithms [4, 5, 6]. However, these algorithms are usually complicated, inaccurate and not applicable for real-time applications. Thanks to the recent developments of new range sensors (Figure 1.1)[7, 8, 9] which measure time delay between transmission of a light pulse and detection of the reflected signal on an entire frame at once, per-pixel depth information can be obtained in real time from depth cameras. This makes the DIBR problem less computationally intense and more robust than other techniques. Furthermore, it helps significantly reduce the number of necessary cameras.

Most DIBR techniques focus on cases of 1D or 2D arrays of cameras while arbitrary camera configurations in 3D are rarely considered. Moreover, they usually assume that depth images are available at the same location with color images [4, 10]. This assumption is true with depth estimation-based techniques, but is impractical for depth camera-based techniques because depth cameras generally do not provide color images. Another problem with depth camera-based IBR techniques is that the resolution of depth images from depth cameras is often quite low. Since color cameras are much cheaper than depth cameras, the need for using a mixture of several cheap, high resolution, color cameras and a few low cost, low resolution, depth cameras becomes significant.

Figure 1.1: An example of depth camera. PMD[vision] CamCube 2.0, with resolution of 204x204 pixels can measure the depth range from 0.3 to 7.0 m.

Furthermore, because the geometric information in DIBR is usually captured in real time from the real physical world instead of from the modeling and synthetic world (which also makes DIBR more photorealistic), the obtained data always suffer from noise and insufficient sampling effects [11, 12]. Therefore, the need for coupling image processing techniques with rendering techniques is a must. This combination significantly increases the computations and is infeasible for real-time applications without parallelism, which makes algorithm and architecture co-design critical. Besides, since rendering with full geometric information (color and depth) has been optimized for general purpose graphics processing units (GPGPUs, Figure 1.2), GPGPUs are considered the ideal

computing platform and a good choice for DIBR problems.

Figure 1.2: An example of GPGPU. Nvidia GTX 275, with 240 CUDA cores.

1.2 Related work

Many different DIBR techniques have been proposed in recent years. A generalized framework of DIBR for 3D TV applications was proposed in [13, 14]. Recently, an approach using a layered representation [4] that provides excellent rendering quality has attracted much interest. However, this approach requires intensive computations and offline processing and, hence, is impractical for real-time applications.

McMillan [15] with his 3D warping method maps a point in an image to a corresponding point in another image at a different view as long as its depth value is known. However, the work considered only single views and did not take advantage of multiple views. Besides, warping is only the first step of the synthesis work. The most difficult problem is how to deal with exposed disocclusion areas in the warped image. Some approaches to handle this problem were proposed in [16, 17]. However, these approaches considered only the 1D case where the virtual camera is forced to be on the same line with real cameras and assumed that depth images are given in the same views with color images. This assumption may not be a good choice because not all depth cameras

provide color information. Furthermore, standard color cameras are much cheaper and provide much higher color resolution than depth cameras. So the combination of a few depth cameras and many color cameras is more feasible.

Another approach which focuses on signal processing techniques is the Propagation Algorithm [10]. Using depth information, surface points that correspond to pixels in the real images are reconstructed and reprojected onto the virtual view. Therefore, the real pixels are said to be propagated to the virtual image plane. Again, it is implied in [10] that color cameras and depth cameras must have the same location and only the 1D case is considered.

In practice, depth cameras (i.e., SwissRanger, CamCube, etc.) often provide the depth images with much lower resolution than that of the color images [7, 8, 9] (Figure 1.3). Therefore, the combination of several high quality color cameras with only a few depth cameras is a potential commercial solution for future low cost IBR systems. The key problem is how to exploit the redundant information of high quality color images to increase the resolution and enhance the quality of depth images. Several publications have addressed this issue. The Markov model approach was proposed in [18]. Iterative bilateral filtering coupling with sub-pixel estimation in [19] provides very good enhancement. However, these approaches are heavily computational and require several iterative loops. Therefore, they are not suitable for real-time rendering.

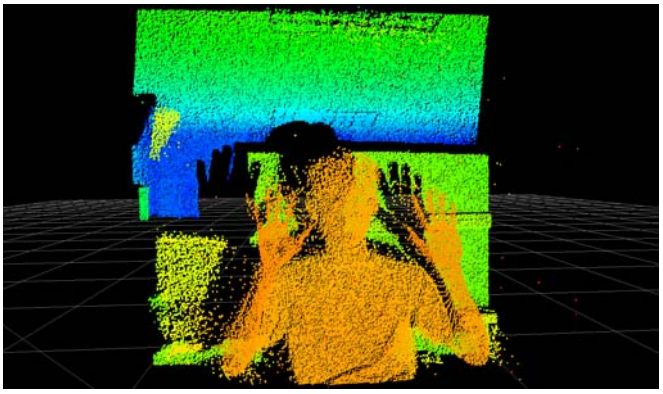
1.3 Goal of this work

In the scope of this work, we propose a new generalized 3D propagation algorithm. Our algorithm has three main advantages compared to other algorithms:

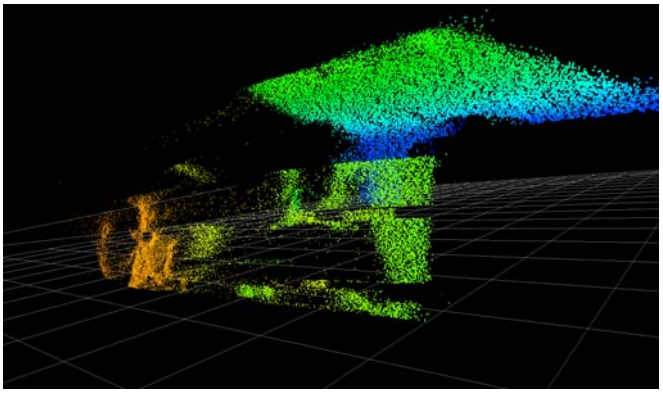
- It considers general configurations of color and depth cameras in 3D space.
- It adapts well with any combination of many high-quality color cameras and a few low-resolution depth cameras, which allows it to perform with

(a) Depth image

(b) Intensity image



(c) Disocclusion areas appear at virtual viewpoint



(d) Depth edges are not sharp enough

Figure 1.3: Depth and intensity images captured by PMD[vision] CamCube 2.0.

low cost DIBR systems.

- It is consciously developed with image processing techniques that have a high degree of locality (e.g., bilateral filtering) and massive parallelism (e.g., process pixel independently) whenever possible. Therefore, it can be easily mapped onto massively parallel architectures such as GPGPUs in order to facilitate real-time operation.

CHAPTER 2

3D PROPAGATION ALGORITHM

2.1 Problem statement

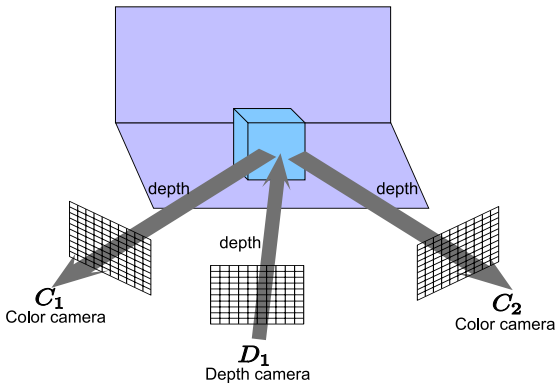
The main goal of our proposed 3D propagation algorithm is to render a new image at an arbitrary virtual view based on images collected from multiple color and depth cameras. Assume that there are N color cameras and M depth cameras capturing a scene in 3D space. The inputs for our algorithm are a set of color images $\{I_i(\vec{x})\}_{i=1}^{N}$, depth images $\{d_j(\vec{x})\}_{j=1}^{M}$, and parameters of depth and color cameras $\{C_i(\vec{x}), f_i, \vec{w}_i\}_{i=1}^{N+M}$, where C_i is camera position of the i^{th} camera, f_i is its focal length, and $\vec{w}_i$ is its normalized viewing direction vector which points from C_i to the image plane center. The output is a rendered image at virtual view $I_v(\vec{x})$. Two assumptions need to be followed: (i) *Calibrated cameras*: the positions, focal lengths, and viewing direction of all cameras are known; (ii) *Lambertian surfaces*: the color of a point on a surface is constant regardless of the angle from which it is viewed.

The proposed algorithm is divided into three main steps (Figure 2.1):

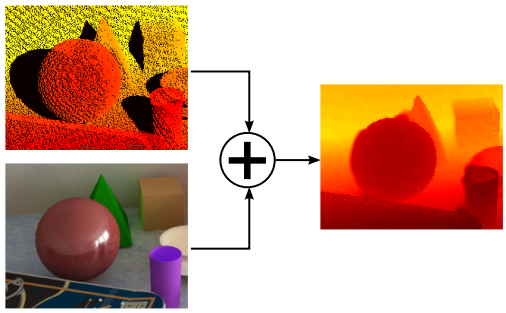
1. *Depth propagation*: Depth information from each depth camera is propagated to every color camera's image plane.

2. *Color-based depth filling and enhancement*: signal processing techniques are applied to obtain an enhanced complete depth image at each color camera.

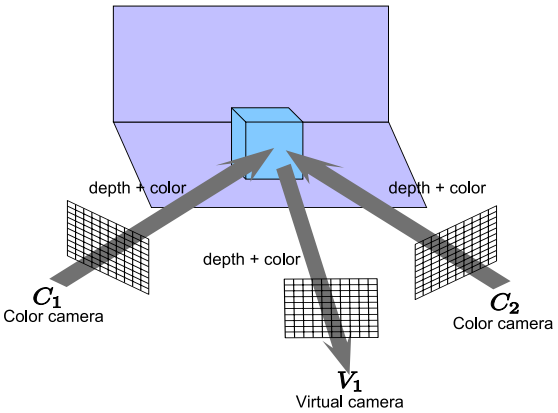
3. *Rendering*: Depth and color information from each color camera is propagated, then merged and filtered to produce the rendered image at a



(a) Depth Propagation step: Depth information is propagated from depth cameras to color cameras.



(b) Color-based Depth Filling and Enhancement step performed at each color camera.



(c) Rendering step: Depth and color information are propagated from the color cameras to the virtual view.

Figure 2.1: Three main steps in the 3D propagation algorithm.

virtual view.

2.2 Depth propagation

In this section, we describe how depth information from a range camera can be propagated to a color camera. The range camera is considered as the reference view and the color camera is considered as the desired view. The warping technique proposed in [15] allows us to map a point in a reference image to a corresponding point in a desired image at a different view as long as we know the depth value of that point. Consider a reference camera $\{C_r, f_r, \vec{w}_r\}$ and a desired camera $\{C_d, f_d, \vec{w}_d\}$ in a 3D Euclidian space with basis vectors $(\vec{i}, \vec{j}, \vec{k})$.

Each point of an image in 2D space can be mapped one-to-one with a ray in 3D space that goes through the camera position. Given a 2D image plane with basis vectors $(\vec{s}, \vec{t})$ and a 3D space $(\vec{i}, \vec{j}, \vec{k})$, the 2D point to 3D ray mapping relation is:

$$\vec{r} = \begin{bmatrix} r_i \\ r_j \\ r_k \end{bmatrix} = \begin{bmatrix} \vec{s}_{ijk} & \vec{t}_{ijk} & f * \vec{w}_{ijk} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.1}$$

where (u, v) is the 2D coordinate of the point in the image plane; $\vec{r}$ represents the corresponding ray's direction; $\vec{s}_{ijk}$, $\vec{t}_{ijk}$ and $\vec{w}_{ijk}$ are representations of $\vec{s}$, $\vec{t}$, and viewing direction $\vec{w}$ in $\{\vec{i}, \vec{j}, \vec{k}\}$. Matrix P is called the *mapping matrix*.

Consider a point X in 3D space $\{\vec{i}, \vec{j}, \vec{k}\}$. Let $\vec{x}_r$ and $\vec{x}_d$ be homogeneous coordinates of X in the reference image plane and the desired image plane (Figure 2.2). Let P_r and P_d be mapping matrices of the reference camera and the desired camera. It was proven in [15] that the warping equation between $\vec{x}_r$ and $\vec{x}_d$ is:

$$\vec{x}_d = P_d^{-1} \left(\frac{|P_r \vec{x}_r|}{d(\vec{x}_r)} (\vec{C}_r - \vec{C}_d) + P_r \vec{x}_r \right) \tag{2.2}$$

where $d(\vec{x}_r)$ is the depth value of point $\vec{x}_r$. In many-to-one mapping cases where

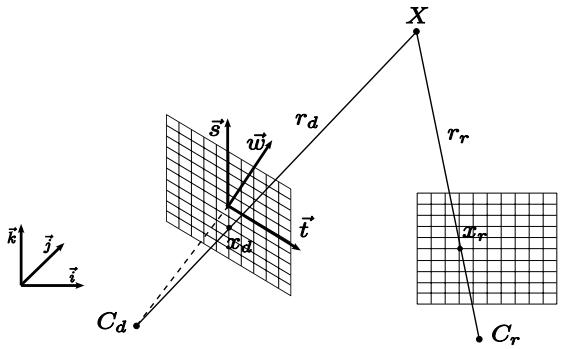


Figure 2.2: A point X can be warped from the reference image plane to the desired image plane.

more than one point in the reference image are mapped to the same point in the desired image, depth values of these points are compared and the one with smallest depth is kept. Figure 3.4 (a) and Figure 3.5 (a) on pages 26 and 27 show the incomplete propagated depth images D_{warped} after warping. Note that all figures of depth images in this thesis are color-coded for better visualization.

2.3 Color-based depth filling and enhancement

This step fills all missing depth pixels in the propagated depth image and performs depth image enhancement to prepare for the rendering step. Its block diagram is shown in Figure 2.3.

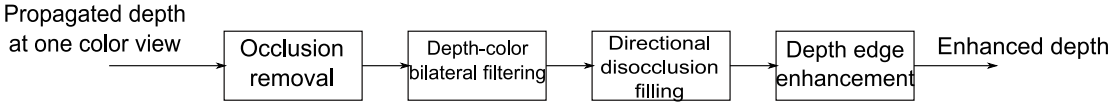


Figure 2.3: Block diagram of the color-based depth filling and enhancement technique.

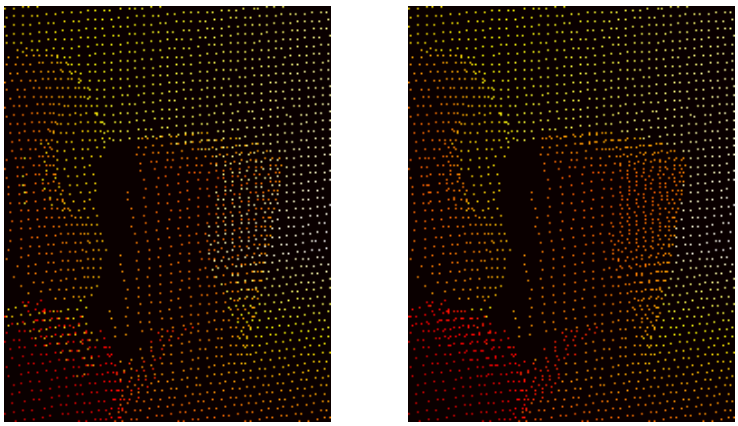
Unknown depth patches or holes, represented by black areas in Figure 3.4 (a) and Figure 3.5 (a), happen for two reasons. First, uniform sampling in the reference image becomes non-uniform sampling in the desired image. This means that in some certain patches, there are fewer sample points than in some other patches. Those holes can be filled by the interpolation step using

depth-color bilateral filtering (Section 2.3.2). Second, holes are also created when occluded areas in the reference image are revealed in the desired image, which is a disocclusion problem. It seems to be impossible to fill these holes. However, for our problem setting, we have full color information at the desired view. Therefore, these holes can be interpolated based on color image at the desired view (Section 2.3.3).

In addition, as shown in Figure 2.4 (a), some background sample points (in brighter color) visible in the reference depth image should be occluded by the foreground (pixels with darker color) in the desired depth image but are still visible. That significantly degrades the interpolation quality. Therefore, an occlusion removal step is necessary to correct those points before performing further processing steps.

2.3.1 Occlusion removal

The key idea of this occlusion removal method is based on the smoothness of surfaces. If a point A in D_{prop} is locally surrounded by neighboring points whose depth values are σ smaller than the depth of A, then A is decided to be occluded by the surface composed of those neighbors.



(a) Before occlusion removal (b) After occlusion removal

Figure 2.4: A patch in the propagated depth image before and after the occlusion removal step.

In our implementation, for every point A, a $(2w+1) \times (2w+1)$ window whose center is A is divided into four partitions as shown in Figure 2.5. If there are at least three of those partitions each of which has at least one point B such that $depth(A) - depth(B) > \sigma$, then point A is decided to be occluded and its depth is replaced by a new interpolated value. Figure 2.4 shows a patch of D_{prop} before and after occlusion removal step respectively. In our experiment, we choose $w = 3$.

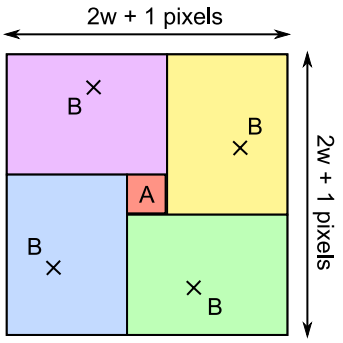


Figure 2.5: Window partitions in the occlusion removal.

2.3.2 Depth color bilateral filtering

Bilateral filtering [20] is a simple, non-iterative scheme for edge-preserving smoothing. It is a combination of a spatial filter, whose weights depend on Euclidean distance between samples, and a range filter, whose weights depend on differences between values of samples. Bilateral filtering is usually applied only for color images and provides excellent enhancement quality [21, 22]. In this paper, by integrating known depth and color information, the proposed DCBF effectively interpolates unknown depth pixels in D_{prop} caused by non-uniform resampling while keeping sharp depth edges. The DCBF is defined as follows:

$$d_A = \frac{1}{W_A} \sum_{B \in S_A} G_{\sigma_s^2}(|\vec{x}_A - \vec{x}_B|) \cdot G_{\sigma_r^2}(|I_A - I_B|) \cdot d_B \tag{2.3}$$

$$W_A = \sum_{B \in S_A} G_{\sigma_s^2}(|\vec{x}_A - \vec{x}_B|) \cdot G_{\sigma_r^2}(|I_A - I_B|) \tag{2.4}$$

where

$$
\begin{aligned}
d_A &: \textit{depth value of point A.} \\
I_A &: \textit{color value of point A.} \\
\vec{x}_A = [u_A, v_A] &: \textit{2D coordinate of point A.} \\
S_A &: \textit{set of A neighboring points.} \\
G_\sigma(|\vec{x}|) = exp\left(\frac{-|\vec{x}|^2}{2\sigma^2}\right) &: \textit{Gaussian kernel.} \\
W_A &: \textit{normalizing term.}
\end{aligned}
$$

The idea of using color differences as a range filter to interpolate depth value is based on the observation that whenever a depth edge appears, there is almost always a corresponding color edge due to color differences between objects or between foreground and background. The DCBF also works well with textured surfaces since it counts only pixels on that surface which have similar color to the interpolated pixel. If surfaces have the same color, color does not give any new information and the DCBF works as a simple interpolation scheme such as bilinear or bicubic. Figure 3.4 (b) and Figure 3.5 (b) show the depth image after the DCBF step. The black areas caused by disocclusion are handled in Section 2.3.3.

2.3.3 Directional disocclusion filling

In order to fill holes caused by disocclusion, the DCBF can also be used, but it needs to follow a specific direction. Otherwise, filtering is performed from all directions, and incorrect depth values may be obtained. As described in Figure 2.6, filling from right to left causes incorrect interpolating depths because the holes should be filled with the background's depth, whereas all neighboring known depth values belong to the foreground.

It is shown in [15] that, in the 2D image plane, the disocclusion holes always appear toward the projection of the reference camera position onto the desired

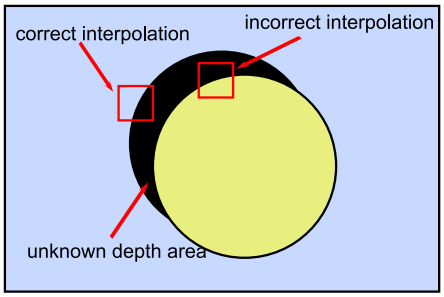


Figure 2.6: Problem with non-directional disocclusion filling.

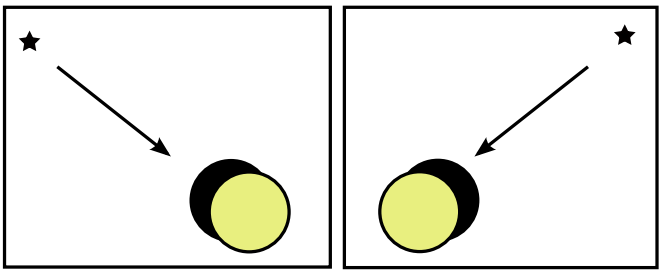


Figure 2.7: Relationship between the disocclusion holes and the camera position.

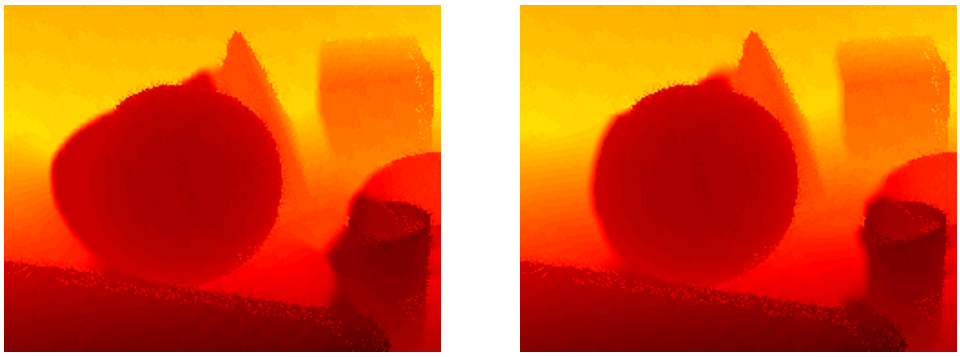
image plane (epipole point, see Figure 2.7). This fact is reasonable since the disocclusion holes are caused by the change of the camera position moving from one view to another. So there must be a relationship between the disocclusion holes and the camera position. The filling direction can be decided correctly based on this relation. The epipole $\vec{e}$ can be computed as follows:

$$\begin{bmatrix} e_x & e_y & e_z \end{bmatrix}^T = P_d^{-1}(C_r - C_d) \tag{2.5}$$

$$\vec{e} = (e_x/e_z, e_y/e_z) \tag{2.6}$$

where C_r and C_d are positions of the reference and desired views, and P_d is the mapping matrix of the desired view. Then the filling direction is a vector pointing from the epipole to the center of the desired depth image. For example, if the epipole lies in the top left quadrant of the image, the filling should start from the top left corner. Figure 2.8 shows that the proposed directional bilateral filling gives more precise depths compared to the non directional bilateral filling.

Figure 3.4 (c) and Figure 3.5 (c) show the complete propagated depth image after filling disocclusion areas.



(a) Non directional bilateral filling (b) Directional bilateral filling

Figure 2.8: The directional bilateral filling gives more precise depths.

2.3.4 Depth edge enhancement

In DIBR problems, the sharpness of depth edges is extremely important because in the rendering step, a slightly blurred edge may blow up to a significant and visually annoying smearing artifact in the rendered image. Performing interpolation before depth propagation has two drawbacks: (i) it increases smoothing of depth edges, (ii) it does not use edge clues from available high resolution color images. When using lower resolution depth images, the number of unknown depth pixels increases significantly. Even though the above filling techniques preserve edges, blurring of edges in the low resolution depth case causes too many visual artifacts in the end result. Therefore, at the end of step 2, a new color-based depth edge enhancement stage is added before the rendering step. The purpose of this stage is to correct and sharpen edges in the propagated depth images at color views.

The proposed depth edge enhancement stage includes two parts. First, depth edge gradients are detected with the Sobel operator in vertical and horizontal directions. Then pixels with significant edge gradients are marked as undetermined depth pixels and their depth values need to be recalculated. Next,

for each undetermined depth pixel, a block-based search is used to find the best determined pixel that matches in the color domain. Once the best candidate is chosen, its depth value is assigned for that of the undetermined depth pixel. The second part may be iterated a few times so that most of the undetermined depth pixels are adjusted.

Given a pixel $x = [x_u, x_v]$ in an image plane $\mathcal{X}$, $I(x)$ and $D(x)$ are color and depth value of pixel x. We define a σ-neighborhood of x $\mathcal{N}_\sigma(x)$ as:

$$\mathcal{N}_\sigma(x) = \{y \in \mathcal{X} : |y_u - x_u| < \sigma \ \ \& \ \ |y_v - x_v| < \sigma\}$$

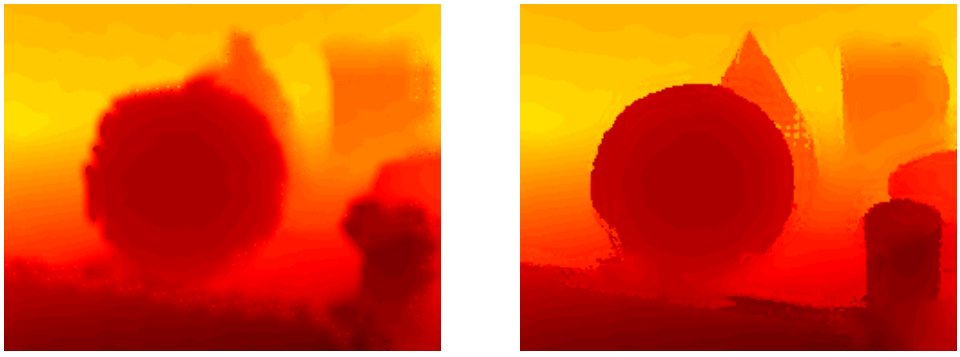
Let $\mathcal{G}_u$ and $\mathcal{G}_v$ be the horizontal and vertical Sobel kernel. Generally, $\mathcal{G}_u$ and $\mathcal{G}_v$ can be the size of 3x3, 5x5, or 7x7. In our experiments, we use the 3x3 Sobel kernel. Let $\mathcal{X}_u$ be the set of undetermined depth pixels and $\mathcal{P}_{\omega,\alpha}(x) = \{y \in \mathcal{X} : y \notin \mathcal{X}_u, y \in \mathcal{N}_{\alpha+\omega}(x), y \notin \mathcal{N}_\omega(x)\}$ be the search range of pixel x. Then the pseudo-code of the proposed depth edge enhancement stage is described in Figure 2.9.

Our experimental results show that the method significantly enhances the propagated depth image at the color viewpoint and, hence, improves rendering quality (Figure 2.10). The method works based on the following observation: There is no texture in a depth image, the significant depth gradients only appear around the true depth edges and at the same location with color gradients (the reverse is not true). Hence, information about color edges in the high quality color images can be used to detect, correct and sharpen the true depth edges. Since the depth values of pixels in a small vicinity of the same object are almost the same, it is appropriate to directly copy a depth value from one neighboring pixel of the same object. This can significantly reduced the computations while still preserving the enhancement quality.

Because the proposed technique is consciously developed using image processing tools that have high degree of locality (e.g., Sobel operator, block-based search), it is purely parallel and can be done extremely fast on

% Apply the Sobel operator to detect depth gradient
`For each` $x \in \mathcal{X}$
  $S_\sigma(x) = Sobel\left(\mathcal{N}_\sigma(x)\right)$
  $= \left(\mathcal{G}_u * \mathcal{N}_\sigma(x)\right)^2 + \left(\mathcal{G}_v * \mathcal{N}_\sigma(x)\right)^2$
  `If` $S_\sigma(x) > T_s$
    $\mathcal{X}_u = \mathcal{X}_u + \{x\}$
  `Endif`
`Endfor`
% Perform block-based search in color domain
`For each` $x \in \mathcal{X}_u$
  `For each` $y \in \mathcal{P}_{\omega,\alpha}(x)$
    `If` $\exists y_0 \in \mathcal{P}_{\omega,\alpha}(x) : \forall y \in \mathcal{P}_{\omega,\alpha}(x),$
$\|I\left(\mathcal{N}_\beta(y)\right) - I\left(\mathcal{N}_\beta(x)\right)\|_2 \geq \|I\left(\mathcal{N}_\beta(y_0)\right) - I\left(\mathcal{N}_\beta(x)\right)\|_2 \geq T_p$
      $D(x) = D(y_0)$
      $\mathcal{X}_u = \mathcal{X}_u - \{x\}$
    `Endif`
  `Endfor`
`Endfor`

Figure 2.9: Pseudocode of the depth edge enhancement stage. T_s and T_p are thresholds; ω and α determine the search window size; β determines the comparison block size.



(a) Before (b) After

Figure 2.10: The depth edge enhancement corrects and sharpens the edges in low resolution depth case.

GPGPU platforms. Therefore, it is easily coupled with the other steps of the 3D propagation algorithm for the goal of real-time rendering.

2.4 Rendering

Now each color camera has both depth and color information. The last step is propagating this information to the virtual camera. The color cameras become the reference views and the virtual camera becomes the desired view. This process is quite similar to the first two parts of the algorithm. First, all depth and color information of each color view is propagated into the virtual view using the same technique in Section 2.2. Then the occlusion removal technique in Section 2.3.1 is performed at the virtual view. Finally, the rendered image is filled and denoised with a 3x3 median filter. Note that most of the unknown color pixels in this step are caused by non-uniform resampling since the color cameras are intentionally installed in a way to capture the whole scene from different views and, therefore, reduce as much as possible the holes caused by disocclusion. The complete rendered image is shown in Figure 3.7.

2.5 Mapping to GPGPU architecture

A major advantage of the proposed algorithm is that it can be easily mapped onto data parallel architectures such as modern graphics processing units (GPUs). In this section, we briefly describe the parallelism of each processing step of our algorithm, and the high level mapping onto the Nvidia CUDA architecture for GPU-based computing.

The occlusion removal, and DCBF steps are purely parallel as each pixel in the desired view can be computed independently. Copying the depth values in the reference view to appropriate pixels in the desired view is more complex from a parallelism perspective since, at some pixels, this is not a one-to-one mapping. This operation requires some form of synchronization to prevent

concurrent writes to the same pixel, and can be accomplished with the use of atomic memory operations, or alternatively, with the use of Z-buffering hardware available on modern GPUs.

The disocclusion filling step in Section 2.3.3 also has a sequential component since calculating unknown depth information is dependent on previously interpolated values. However, this dependence exists only on 1D lines emanating from the epipole point, and thus the problem can be expressed as a parallel set of 1D filters. First, find the epipole point position and categorize into one of eight following subsets: top, bottom, left, right, top left, top right, bottom left, or bottom right, corresponding to eight sets of parallel lines for every 45 degree angle. The parallel lines in each set need to pass through all pixels in the depth image. For each set of parallel lines, all pixel coordinates of each line can be pre-computed and stored in a lookup table. The 1D DCBF is performed with each line proceeding in parallel, which can be easily mapped onto the GPU architecture.

The depth edge enhancement step described in Section 2.3.4 is simply a series of independent window-based operators (Sobel operators and window-based searches for the best match) and, hence, is naturally parallel. The final rendering step is quite similar to the first and second part of the algorithm except for the inclusion of a median filter. However, the median filter is another window-based operator and, hence, is data parallel in nature.

In order to check the efficiency of the parallelism, we parallelized two main bottlenecks in the algorithm. Then we compare the CPU-based mode and the GPU-based mode. The experiment was run on the platform of Intel Core2 Duo E8400 3.0 GHz and a Nvidia GeForce 9800GT with 112 processing cores.

The first bottleneck for GPU-based implementation is the depth propagation step. Even though there is not too much computation, this copying causes two problems. First, the algorithm is not trivially parallel because several depth pixels can map to the same pixel in the color view, which must be detected and resolved by selecting the closest pixel. Second, it tends to produce irregular

memory accesses (both reading and writing). This is a typical problem whenever there is a 3D-to-2D projection in the video synthesis, which shows that not all video processing techniques are naturally amenable to parallelism. Our implementation uses atomic hardware primitives, which guarantee the read-modify-write sequence of operations is performed without interference from other threads. Since atomic primitives temporarily lock a part of the memory, the GPU-based implementation suffers a significant slowdown. Nonetheless, it is still slightly faster than the CPU-based implementation (1.6 times). We are investigating a prefix-sum based parallel algorithm for a higher speedup.

The second bottleneck is the DCBF step due to its heavy computational requirement. The computational complextity of the DCBF is approximately $O(N^2K^2)$ for an $N \times N$ propagated depth map and a $K \times K$ kernel size. The filter is similar to the 2D convolution with variant impulse responses. First, the depth map and the corresponding color image are tiled with the overlapped boundary, whose width equals the half of the kernel size, and loaded into the shared memory. Loading the center of the tile is obviously coalesced. By loading the boundary horizontally, even though very few threads are used, we obtain a faster and uncoalescing-free load. To avoid branching inside the CUDA kernel, the whole depth map and color image are boundary padded. The Gaussian kernel weights for both spatial and range filters can be precomputed and stored in the constant memory as a lookup table because they are used many times repeatedly. In each thread block, each thread sticks with one pixel. Whether the depth of the pixel is known or not, the filter is always performed based on the neighboring known depth pixels for filling and denoising purposes. Each task calculates a depth value for one pixel based on neighboring pixels. Due to the constraint of the limited number of registers, full occupancy of threads cannot be achieved for such a big kernel. The shared memory availability is also another constraint when the kernel size grows bigger. In our experiment, the image resolution is 800x600 and the kernel size is 11x11.

Table 2.1 shows the comparison result, with a speedup of 74.4 times compared

Table 2.1: Timing comparison (in milliseconds) of sequential CPU-based and GPU-based implementations for two main bottlenecks: depth propagation and depth-color bilateral filtering (DCBF). The image resolution is 800×600 and the filter kernel size is 11×11.

	Hardware	Depth Prop.	DCBF
CPU	Intel Core 2 Duo E8400 3.0GHz	38	1041
GPU	NVIDIA GeForce 9800 GT	24	14
Speedup		1.6x	74.4x

to the sequential implementation on CPU.

Regarding the parallel scalability of our algorithm: our experiments show that there is ample data parallelism to take advantage of the heavily threaded 128-core modern GPU architecture. Our technique scales further with image size, and higher resolution images will create additional parallel work for future data parallel architectures that support still higher degrees of parallelism. Furthermore, with the use of additional cameras, the data parallel computational load increases still further, creating additional work that can be gainfully accelerated on future data parallel architectures. In the case of additional cameras, two approaches can be used: (1) the problem can be simplified into the case of using two nearest color cameras on the left and right with one nearest depth camera, and ignore other cameras (this approach is appropriate if the computational load is too high for real-time operation), or (2) data from every camera can be used to improve rendering quality. Note that processing each color view (depth propagation, filling and enhancement) is independent and suitable for parallelism.

CHAPTER 3

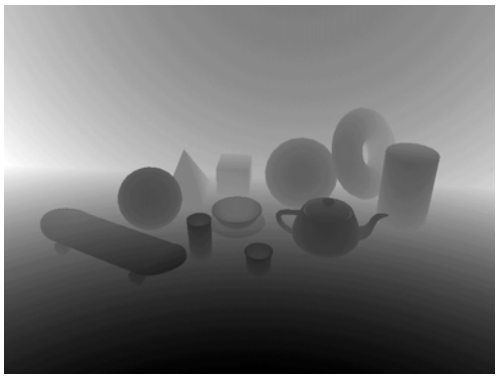
EXPERIMENT RESULTS

For the experiments, we adopted a synthesis configuration with two color cameras and one depth camera. The color input images are shown in Figures 3.1 (a) and (b). For the high resolution depth case, the resolution is 800x600 (Figures 3.1 (c)). The window size for the occlusion removal step in Section 2.2 is 7x7. Parameters for the DCBF in Section 2.3.2 are set as follows: $\sigma_s^2 = 3$, $\sigma_r^2 = 0.01$, kernel window size = 11x11. The depth enhancement and rendering results are shown in the following figures. The black area at the bottom right of the rendered image is due to the fact that there is no information at all (neither depth nor color) from the input images corresponding to those areas.

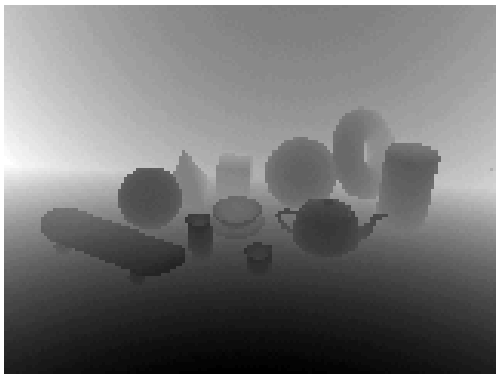
For the low resolution depth case, the depth image is down-sampled to the resolution of 160x120 (keep only 4% of the depth information in the high resolution case) (Figure 3.1 (d)). Some intermediate results are shown in Figure 3.2 and Figure 3.3. From the experimental results in Figures 3.4-3.8 , we see that the proposed color-based depth filling and enhancement technique not only fills and preserves the depth edges but also corrects the depth edges. The enhanced propagated depth of the left color view and the complete rendered image are almost the same as those in the high resolution depth case. Note that the information contained in a depth image is not as much as in a color image. So if a sufficiently sparse set of depth samples and true depth edges are given, the high resolution version of the depth image can be restored completely.

(a) Input left color view (resolution: 800x600)

(b) Input right color view (resolution: 800x600)

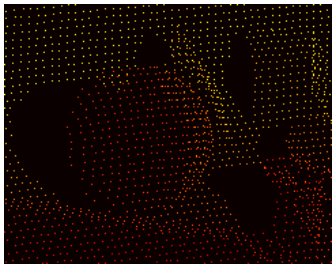


(c) High resolution input depth image at the middle view (resolution: 800x600)

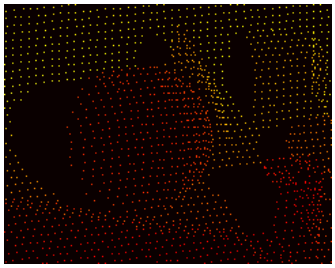


(d) Low resolution input depth image at the middle view (resolution: 160x120)

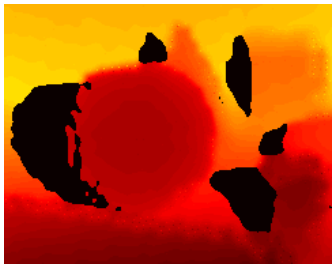
Figure 3.1: Input images for both low and high resolution depth cases.



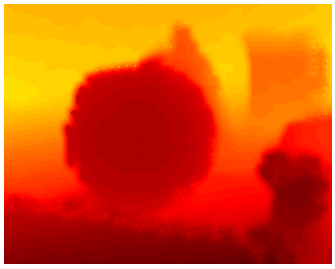
(a) Propagated depth



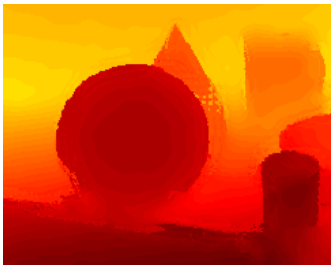
(b) After occlusion removal



(c) After depth-color bilateral filtering

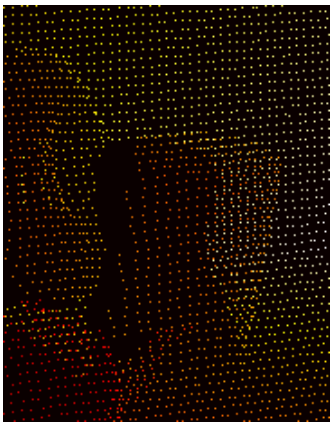


(d) After disocclusion filling

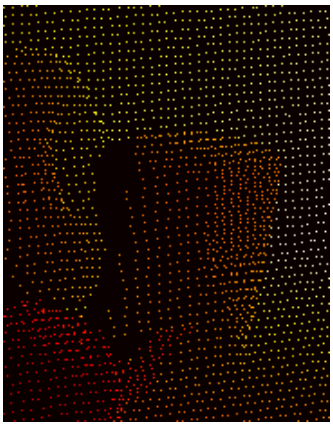


(e) After depth edge enhancement

Figure 3.2: Intermediate results for the low resolution depth case.



(a) Propagated depth



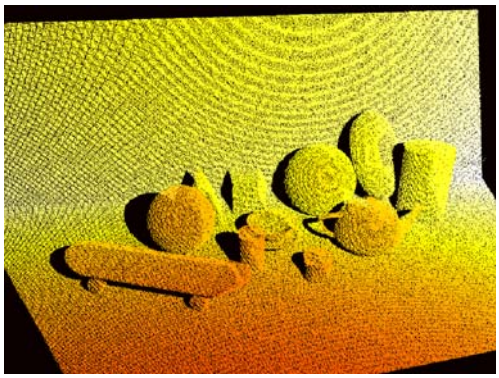
(b) After occlusion removal

(c) After depth-color bilateral filtering

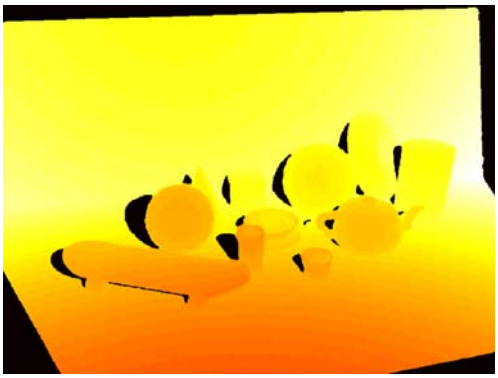
(d) After disocclusion filling

(e) After depth edge enhancement

Figure 3.3: Intermediate results for the low resolution depth case.



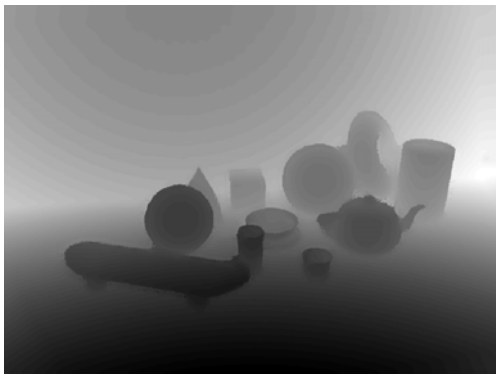
(a) Propagated depth before filling and enhancement



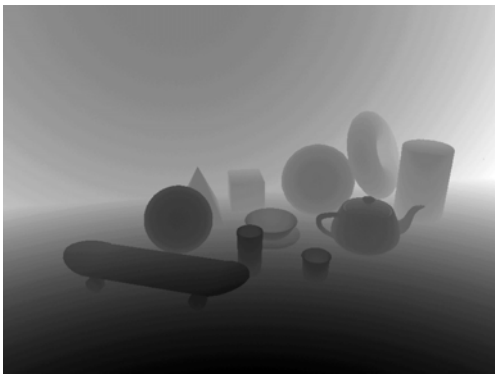
(b) After depth-color bilateral filtering

(c) After directional disocclusion filling

(d) After depth edge enhancement



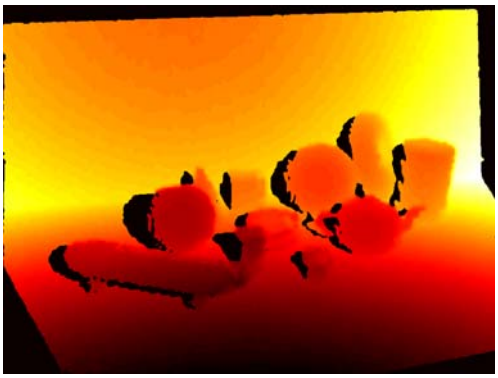
(e) Final grayscale depth image at the left color view



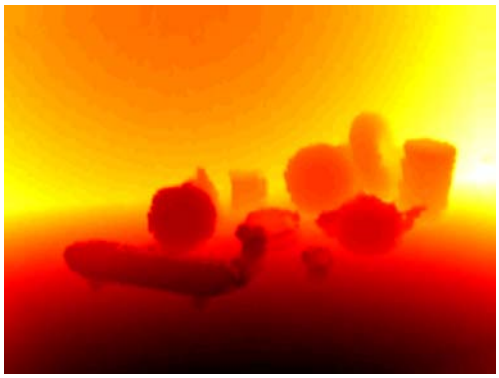
(f) Ground truth depth image at the left color view

Figure 3.4: Result of the depth filling and enhancement step at the left color view for the high resolution case.

(a) Propagated depth before filling and enhancement

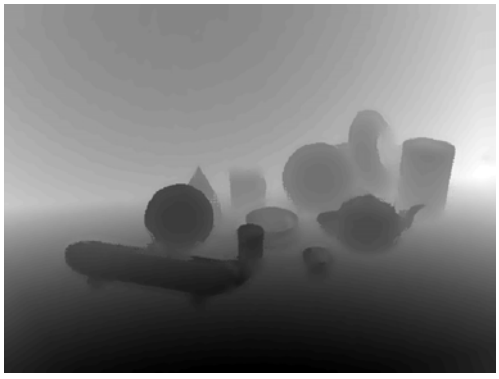


(b) After depth-color bilateral filtering

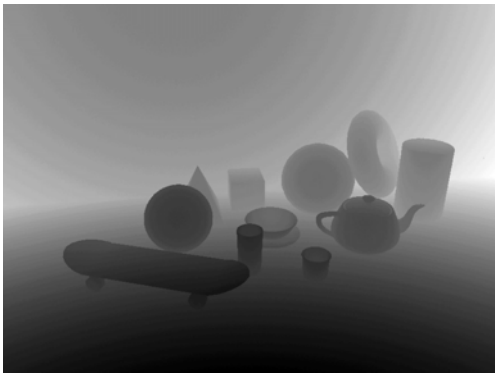


(c) After directional disocclusion filling

(d) After depth edge enhancement

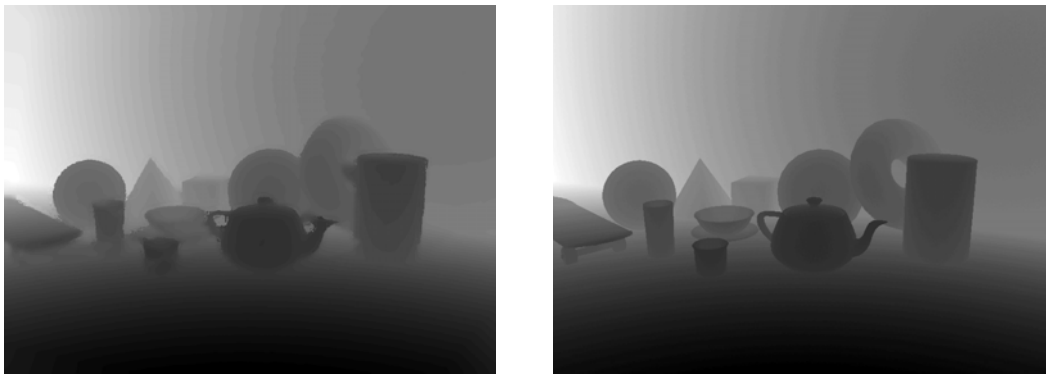


(e) Filled and enhanced depth image in grayscale



(f) Ground truth

Figure 3.5: Result of the depth filling and enhancement step at the left color view for the low resolution case.



(a) Filled and enhanced depth image (b) Ground truth

Figure 3.6: Filled and enhanced depth image at the right color view in the high resolution case compared to the ground truth.

(a) Rendered image with high resolution input depth (b) Rendered image with low resolution input depth

Figure 3.7: Rendered results for both low and high resolution input depth cases.

Figure 3.8: Rendered images at some other virtual views.

CHAPTER 4

CONCLUSION

Our proposed 3D propagation algorithm provides an effective framework to integrate and combine depth and color information from multiple cameras. The depth information from range cameras is first propagated to the color views, then both color and depth information are propagated to the virtual view. The algorithm provides an excellent improvement of rendering quality, and can be formulated as a series of mostly data parallel operations, and thus mapped onto a GPU architecture using the Nvidia CUDA programming system. Our experimental studies show that the mapping onto GPUs is quite efficient, and yields nearly real-time operation of our algorithm on a set of standard resolution images.

As for future work, the following directions could be pursued:

- First, we are intensively working on the complete GPU-based implementation. Then further optimization for improving performance will be focused on.

- Second, we will extend our work to real-time video rendering. A straightforward extension is to apply the current algorithm independently frame-by-frame for videos. However, we plan to exploit the temporal correlation in videos to reduce computations and memory access, and enhance the rendering quality.

- Finally, we will deploy and integrate our developed algorithm and computational platform in a test-bed application with real cameras and scenes. We will aim for a remote reality system that uses commodity

hardware and provides users with a real-time free-viewpoint 3D viewing experience.

REFERENCES

[1] S. C. Chan, H. Y. Shum, and K. T. Ng, "Image-based rendering and synthesis," *IEEE Signal Processing Magazine*, vol. 24, pp. 22–33, Nov 2007.

[2] A. Kubota, A. Smolic, M. Magnor, M. Tanimoto, T. Chen, and C. Zhang, "Multiview imaging and 3DTV," *IEEE Signal Processing Magazine*, vol. 24, pp. 10–21, Nov 2007.

[3] T. Kanade, P. W. Rander, and P. J. Narayanan, "Virtualized reality: Constructing virtual worlds from real scenes," *IEEE Multimedia Magazine, Immersive Telepresence*, vol. 4, pp. 34–47, Jan 1997.

[4] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Transactions on Graphics*, vol. 23, pp. 600–608, Aug 2004.

[5] A. Saxena, S. Chung, and A. Y. Ng, "3D depth reconstruction from a single still image," *International Journal of Computer Vision*, vol. 76, pp. 53–69, Jan 2008.

[6] A. Akbarzadeh, J. M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys, "Towards urban 3D reconstruction from video," in *International Symposium on 3D Data Processing Visualization and Transmission*, Los Alamitos, CA, USA, 2006, pp. 1–8.

[7] Canesta Inc. web site. [Online]. Available: http://www.canesta.com/

[8] PMD Technologies web site. [Online]. Available: http://www.pmdtec.com/

[9] Mesa Imaging web site. [Online]. Available: http://www.mesa-imaging.ch/

[10] H. T. Nguyen and M. N. Do, "Image-based rendering with depth information using the propagation algorithm," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Mar 2005, pp. 589–592.

[11] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision, 2nd ed.* New York, NY, USA: Cambridge University Press, 2003.

[12] S. Chan, K. Ng, Z. Gan, K. Chan, and H. Shum, "The plenoptic video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 1650– 1659, Dec 2005.

[13] C. Fehn, "Depth-image-based rendering (DIBR), compression and transmission for a new approach on 3D-TV," in *Proc. SPIE Stereoscopic Displays and Virtual Reality Systems XI*, Jan 2004, pp. 93–104.

[14] C. Fehn and R. Pastoor, "Interactive 3DTV - Concepts and key technologies," in *Proceedings of the IEEE*, vol. 94, Mar 2006, pp. 524–538.

[15] L. McMillan, "An image-based approach to three dimensional computer graphics," Ph.D. dissertation, University of North Carolina at Chapel Hill, 1997.

[16] W.-Y. Chen, Y.-L. Chang, S.-F. Lin, L.-F. Ding, and L.-G. Chen, "Efficient depth image based rendering with edge dependent depth filter and interpolation," in *IEEE International Conference on Multimedia and Expo (ICME)*, Aug 2005, pp. 1314–1317.

[17] I. Daribo, C. Tillier, and B. Pesquet-Popescu, "Distance dependent depth filtering in 3D warping for 3DTV," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, Greece, 2007, pp. 312–315.

[18] J. Diebel and S. Thrun, "An application of Markov random fields to range sensing," in *Proceedings of Conference on Neural Information Processing Systems*, Cambridge, MA, USA, 2005, pp. 291–298.

[19] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2007, pp. 1–8.

[20] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. of the IEEE International Conference on Computer Vision*, Bombay, India, Jan 1998, p. 839.

[21] B. Zhang and J.P.Allebach, "Adaptive bilateral filter for sharpness enhancement and noise removal," *IEEE Transactions on Image Processing*, vol. 17, pp. 664–678, May 2008.

[22] W. C. Kao and Y. J. Chen, "Multistage bilateral noise filtering and edge detection for color image enhancement," *IEEE Transactions on Consumer Electronics*, vol. 51, pp. 1346–1351, Nov 2005.