TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEKSANDRA CHUCHVARA

REAL-TIME VIDEO-PLUS-DEPTH CONTENT CREATION UTILIZING TIME-OF-FLIGHT SENSOR – FROM CAPTURE TO DISPLAY

Master of Science Thesis

Examiners:
Professor Atanas Gotchev
Professor Jarmo Takala
Examiners and topic approved by the Council of the Faculty of Computing and Electrical Engineering on 9 April 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
CHUCHVARA, ALEKSANDRA: Real-time Video-plus-Depth Content Creation Utilizing Time-of-Flight Camera - from Capture to Display
Master of Science Thesis, 61 pages
October 2014
Major: Software Systems
Examiners: Professor Atanas Gotchev, Professor Jarmo Takala
Keywords: three-dimensional television, free-viewpoint television, depth-image-based rendering, video-plus-depth, time-of-flight, graphics processing unit

Recent developments in 3D camera technologies, display technologies and other related fields have been aiming to provide 3D experience for home user and establish services such as Three-Dimensional Television (3DTV) and Free-Viewpoint Television (FTV). Emerging multiview autostereoscopic displays do not require any eyewear and can be watched by multiple users at the same time, thus are very attractive for home environment usage. To provide a natural 3D impression, autostereoscopic 3D displays have been design to synthesize multi-perspective virtual views of a scene using Depth-Image-Based Rendering (DIBR) techniques. One key issue of DIBR is that scene depth information in a form of a depth map is required in order to synthesize virtual views. Acquiring this information is quite complex and challenging task and still an active research topic.

In this thesis, the problem of dynamic 3D video content creation of real-world visual scenes is addressed. The work assumed data acquisition setting including Time-of-Flight (ToF) depth sensor and a single conventional video camera. The main objective of the work is to develop efficient algorithms for the stages of synchronous data acquisition, color and ToF data fusion, and final view-plus-depth frame formatting and rendering.

The outcome of this thesis is a prototype 3DTV system capable for rendering live 3D video on a 3D autostereoscopic display. The presented system makes extensive use of the processing capabilities of modern Graphics Processing Units (GPUs) in order to achieve real-time processing rates while providing an acceptable visual quality. Furthermore, the issue of arbitrary view synthesis is investigated in the context of DIBR and a novel approach based on depth layering is proposed. The proposed approach is applicable for general virtual views synthesis, i.e. in terms of different camera parameters such as position, orientation, focal length and varying sensors spatial resolutions. The experimental results demonstrate real-time capability of the proposed method even for CPU-based implementations. It compares favorably to other view synthesis methods in terms of visual quality, while being more computationally efficient.

PREFACE

I would like to thank my supervisor Professor Atanas Gotchev for giving me such a great opportunity to complete my thesis, for his guidance and support during the work, and for introducing me to so many people in research community.

I would like to specially acknowledge my supervisor and friend Mihail Georgiev for his trust and support, for always giving me his valuable time and for allowing me to write this thesis at his chair. His advice has always been a great help for me.

Thanks to all members of 3D Media Group for creating such a friendly and pleasant working environment. Special thanks to Olli Suominen for guiding me around the laboratory and helping out with the experimental setup.

I also would like to thank Professor Jarmo Takala for reviewing this thesis and providing me with his fast feedback and helpful comments.

Finally, I want to thank my family for giving me this wonderful opportunity to do my studies in Tampere, for their love, and never-ending patience and support in all circumstances.

Tampere, 19th of August 2014

Aleksandra Chuchvara

TABLE OF CONTENTS

ABBREVIATIONS

2D	Two-Dimensional
3D	Three-Dimensional
3DTV	Three-Dimensional Television
3DV	Three-Dimensional Video
CPU	Central Processing Unit
DIBR	Depth Image Based Rendering
FBO	Frame Buffer Object
fps	frames per second
FTV	Free-viewpoint Television
GPU	Graphics Processing Unit
GLSL	OpenGL Shading Language
GT	Ground truth
HD	High Definition
JBF	Joint Bilateral Filter
LDV	Layered Depth Video
MVD	Multiview Video plus Depth
NDC	Normalized Device Coordinate
OpenGL	Open Graphics Library
OpenCV	Open Source Computer Vision
PMD	Photonic Mixer Device
PSNR	Peak Signal-to-Noise Ratio
RGB	Red, Green, Blue
ToF	Time-of-Flight
V+D	Video plus Depth

1. INTRODUCTION

3D video systems provide a viewer with 3D depth impression by displaying a scene from multiple perspectives, thus supporting stereoscopic views and some limited view parallax. 3D video technologies find place in wide variety of applications including entertainment, visualization, medicine, education and communication. In recent years, significant research interest has been shown in 3D video production for real-time applications such as Three-Dimensional Television (3DTV) and Free-Viewpoint Television (FTV). While 3D stereoscopic video is already well-established in cinemas, 3DTV would create a more natural and realistic viewing experience for home entertainment by providing the viewer with 3D feeling of visual content. FTV would allow for viewing 3D video content with freely adjustable viewpoints. With the recent advances in development of 3D camera technologies, display technologies and related data representation formats and standards, it became possible to attack the problem of dynamic 3D content creation of general, real-world scenes. Research in this area spans the whole media processing chain from capture to display [1].

New multiview autostereoscopic displays are very attractive for home environment usage as they do not require any eyewear and can be watched by multiple users at the same time. In traditional stereoscopic technology, one view for each eye is produced. When stereo views are properly created, human brain uses disparity in these two views to extract depth information of a scene. However, specific glasses are required to ensure that each eye gets its corresponding view. In contrast, autostereoscopic multiview displays emit multiple views at the same time and the light from each view is emitted in different directions, while the necessary optical elements for correct view separation, e.g. parallax barrier or lenticular lenses, are built-in in the display itself. Each view is visible only from a particular viewing angle, thus no glasses are required. By showing multiple perspective views, the effect of motion parallax is also supported, i.e. user can see a scene from different perspectives when moving in front of the screen, as it would be expected for real 3D objects. The more views are emitted the more natural and smooth the effect of motion parallax is. To provide such a 3D experience, videos of the same scene captured from different perspectives are required.

Since it is not practical to capture and transmit video of a scene from every view point, only a few views are provided, while the remaining views have to be reconstructed on the display side. Modern 3D displays are able to synthesize desired views using Depth-Image-Based Rendering (DIBR) techniques [2]. For DIBR algorithms, depth information of a rendered scene need to be provided in a form of a depth map associated with color, e.g. in a view-plus-depth format (Section 2.2.3). For each pixel in the color

image there is a corresponding depth value describing its depth in the scene. Such a 3D data representation allows rendering of virtual perspective views shifted with respect to the available color image. However, quality of virtual views decreases with the distance from the available view due to disocclusion artifacts.

When a 3D video system includes depth based view synthesis technique to provide more natural 3D effect, accurate depth map estimation is the most important aspect affecting the quality of the 3D output. Accurate depth sensing is a challenging task, especially when targeting real-time processing rates. Nowadays, depth sensing for 3DTV and FTV is made possible with new data acquisition devices, such as Time-of-Flight (ToF) sensors, which are designed to obtain scene range distances in real-time. However, such sensors can provide only low-resolution data.

In this thesis the problem of real-time 3D video content creation is addressed, focusing on the data acquisition scenario utilizing ToF depth sensor. Throughout the thesis a prototype 3DTV system capable for rendering viewing live 3d video on a 3D autostereoscopic display is presented. The presented system makes use of processing capabilities of modern general-purpose Graphics Processing Units (GPUs) in order to achieve real-time processing rates. Additionally, the matter of arbitrary view synthesis used to provide free-viewpoint functionality is investigated in the context of DIBR and a novel approach based on depth layering is proposed.

1.1 Objectives and Scope

This thesis work is focused on prototyping of a 3DTV system where 3D video of a scene can be captured in real time and viewed on a 3D autostereoscopic display. The work is motivated by the problem of real-time calculation of a dense depth map corresponding to a high-resolution image, which results in a view-plus-depth data representation used for 3D video rendering. Therefore, a real-time 3D imaging scenario where scene is captured by a conventional color camera and a depth sensor based on the time-of-flight principle is employed. The main objective of the work is to develop efficient algorithms for the stages of synchronous data acquisition, color and ToF data fusion and final view-plus-depth frame formatting and rendering on a 3D display. Since a real-time implementation of a full system from capture to display is targeted, the applied image processing and rendering algorithms are designed to match the programming paradigms of fast parallel GPUs in order to achieve real-time processing rates while providing good visual quality.

The following tasks fall within the scope of the project: study and utilization of camera calibration techniques in order to calibrate a multicamera system with respect to its intrinsic and extrinsic parameters; design and implementation of a 3D scene capture setting consisting of a color camera and a ToF depth sensor; efficient GPU based implementation of a chain of image processing algorithms for data denoising, optical distortions correction, data fusion and forming a view-plus-depth frame; efficient imple-

mentation of DIBR. GPU base synthesis of arbitrary virtual views based on view-plus-depth input has also been considered.

The outcome of this thesis is an efficient end-to-end system combining several methods to achieve real-time processing of the whole chain from 3D capture of a scene to its free-view display. The individual parts of the processing chain are implemented and integrated into a unified framework. Each component in the framework is presented in detail throughout the thesis. Necessary initialization setup and an efficient algorithmic approach for the creation of depth maps, depth and color data fusion and depth image-based rendering related to this framework are presented. Additionally, the issue of arbitrary view synthesis in the context of DIBR is investigated and a novel approach based on depth layering is proposed. Also various well known denoising techniques currently presented for ToF data and inpainting techniques used to cope with the disocclusion problems of DIBR synthesis are outlined. However, in-depth discussions of these broad research topics are beyond the scope of the presented work. Terminology is given in DIBR context. As the approach heavily relies on GPU resources, strong emphasis is made upon principal steps of GPU implementation with focus on real-time performance.

1.2 Related Work

Virtual views synthesis is a common task in 3D video applications. When a 3D video system relies on DIBR algorithms, depth information of the rendered scene in a form of a depth map associated with color is required. Accurate depth sensing is a challenging task especially for real-time applications. Dedicated sensors, i.e. ToF cameras, allow real-time depth sensing [3]. Such depth sensor is a key element of 3D video system presented in this thesis and will be described in more detail in the following chapter.

The major advantage ToF sensor compared to other depth estimation techniques is the ability to deliver depth map of the entire scene at a high frame rate and independently of textured surfaces and scene illumination. However, there are some drawbacks associated with the ToF working principle; namely, limited resolution and accuracy, as well as inability to capture color information. These limitations can be compensated by fusing ToF data with high-resolution video captured by one or more conventional video cameras. High-resolution texture data can provide important guidance for dense depth maps estimation from low-resolution ToF data.

Different combinations of high-resolution video cameras and low-resolution ToF sensors have been studied. Setups described in [4]-[7] utilize configuration with a single color view and a single range device. Approaches described in [5] and [6] upsample low-resolution depth maps by means of adaptive multilateral upsampling filtering schemes (similarly to [15]) aimed at preserving sharp edges in geometry data while smoothing it over homogenous regions. A GPU based approach proposed in [5] has been demonstrated to be feasible for real-time applications. A scene reconstruction method that iteratively adds every unique scene portion of scene, data utilizing both geometric and color properties of the already acquired parts of the scene has been pro-

posed in [6]. This technique enables capturing a scene with difficult geometry even with a small and low-cost depth sensor resulting in visually convincing 3D model of the scene. A rather straightforward data fusion scheme has been implemented in [4]. The data fusion stage is implemented by mapping the ToF data as 3D point clouds and projecting them onto the color camera sensor plane, resulting in pixel-to-pixel projection alignment of color and depth maps. Subsequently, the color image is back-projected as a texture onto the ToF sensor plane.

When multiple color cameras are available in a capturing setup, ToF data can be combined with disparities obtained by stereo-matching algorithms resulting in high quality disparity maps. Depth from stereo is a well-studied field of research [16]. Depth estimation from stereo is done by finding corresponding pixels in left and right views. When the correspondence between two views is found, the depth is inferred via triangulation. Properties of stereo matching estimations and ToF data are complementary in a way. Spatial resolution of depth map obtained from stereo matching can be very high. However, stereo matching algorithms often fail in textureless areas or areas with repetitive patterns. In such areas, ToF depth sensing is advantageous. An interesting solution of combining stereo and ToF data is described in [8]. The algorithm utilizes more than two images for stereo matching and produces a dense depth map for one of the available color views by calculating per-pixel cost function. In [9], ToF range measurements converted to stereo disparities are used to initialize a stereo matching algorithm. However, real-time implementation is not considered in these works.

Works that are focused on real-time performance of data fusion and DIBR are described in [10]-[13]. A rendering algorithm presented in [10] combines view blending with inpainting and achieves a real-time performance using GPU acceleration. A full real-time implementation of a multilateral filtering system for depth and color data fusion is presented in [11] and [13]. An approach proposed in [12] focuses on real-time implementation for depth-from-stereo estimation by utilizing FPGA hardware. Most relevant to this thesis is the system presented in [14]. The system has been designed for real-time multiview rendering and is based on a depth camera and a color camera. The multi-view rendering is done by means of 3D surface mesh representation of a scene and projective texture mapping. However, such problems as erroneous color mapping in regions hidden from the color camera or areas disoccluded in generated virtual views have not been addressed.

1.3 Thesis Structure

The thesis is structured as follows. Necessary background and fundamentals for the ideas discussed throughout the thesis are presented in Chapter 2. First, the basic pinhole camera model is briefly described and general concepts and practices related with DIBR are introduced. Further, the ToF technology, its working principles and limitations are presented. Finally, a high level overview of 3D graphics pipeline is explained for the needs of the rendering process.

Chapter 3 presents a system-level overview of the proposed 3D video system. It describes aspects such as system architecture and key functionalities, design and calibration of the data capturing setup, organization of data streaming between the cameras and the application and interfacing of an autostereoscopic display.

Chapter 4 provides step-by-step description of the processing chain for video-plus-depth content creation and arbitrary view synthesis in the context of GPU graphics pipeline. It includes steps for ToF data denoising, data reprojection and fusion for dense depth map generation, optical distortions correction, view-plus-depth frame formatting and rendering, and arbitrary view rendering from view-plus-depth data. At the end of the chapter, the performance evaluation of the presented system is provided.

In Chapter 5, a new approach for DIBR based on depth layering is proposed. It presents the depth layering procedure and the view synthesis algorithm along with obtained experimental results.

Finally, Chapter 6 concludes the presented work and discusses possible topics for future research.

2. THEORETICAL BACKGROUND

A common goal for 3DTV and FTV systems can be formulated as real-time rendering of realistic virtual views from available recorded information about real-world scenes. Provided that certain amount of 3D geometric information about the scene is available, the amount of recorded color information necessary to produce reliable images can be substantially reduced. Nowadays, real-time depth imaging of scenes can be facilitated by rapidly developing ToF depth sensing technologies, which become more and more popular within the DIBR context. From another side, increased computational power of modern GPUs allows for fast execution of computationally demanding image processing methods of depth and color necessary for realistic 3D visualization.

This chapter provides necessary background for the ideas presented throughout the thesis. The basic pinhole camera model is presented in Section 2.1. General concepts and practices related with DIBR are introduced in Section 2.2. Time-of-flight technology, its working principles and limitations are described in Section 2.3. Finally, in Section 2.4 a high level overview of 3D graphics pipeline is given.

2.1 Pinhole Camera Model

The pinhole camera is a simplified camera model where the camera aperture is assumed to be dimensionless, distortions caused by the actual lenses performing projections in real cameras are omitted, and projection plane is assumed to be parallel to the aperture plane [17]. The model parameters define the relation between 3D coordinates of the observed scene and their 2D projections. The plane where the image is projected is called the image plane or focal plane. The distance between the optical center C and the image plane is the focal length f and the line perpendicular to the image plane is the principal axis. The intersection of the principal axis with the image plane is called the principal point P and the plane containing optical center and parallel to the image plane is called principal plane. In practice, it is natural to place the image plane behind the optical center of a real camera. However, in case of theoretical model image plane can be moved in front of the optical center in order to work with aligned images as shown in Figure 2.1.

Assuming a pinhole camera model with the origin of the image plane coordinates at the principal point, the mapping between 3D world point $(x,y,z)^T$ and the point on the image plane $(u,v)^T$ can be calculated from similar triangles as follows:

$$u = \frac{x \cdot f}{z},\ v = \frac{y \cdot f}{z}. \tag{2.1}$$

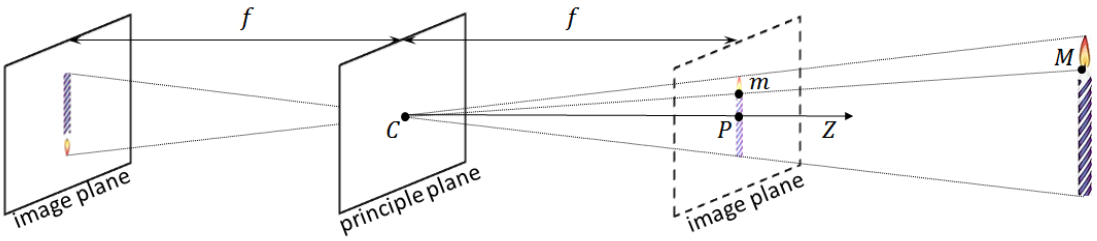

Figure 2.1 Pinhole camera model.

Using homogenous coordinates, (2.1) can be rewritten as:

$$z\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{2.2}$$

In matrix form, Eq. (2.2) can be written as:

$$zm = \boldsymbol{P}M \tag{2.3}$$

where $M = (x,y,z,1)^T$ and $m = (u,v,1)^T$ are homogenous coordinates, $\boldsymbol{P}_{[3\times4]}$ is a camera projection matrix of the central projection pinhole model.

In the general case, a camera projection matrix P can be represented as:

$$\boldsymbol{P} = \boldsymbol{K}\left[\boldsymbol{R}\middle|\, t\right] \tag{2.4}$$

where $\boldsymbol{K}$ contains intrinsic parameters and $[\boldsymbol{R}|t]$ contains extrinsic parameters of a camera. Intrinsic camera parameters define the correspondence between image pixel coordinates and camera coordinates. The matrix $\boldsymbol{K}_{[3\times3]}$ is called camera calibration matrix and is defined as:

$$K = \begin{bmatrix} f \cdot s_x & s & p_x \cdot s_x \\ 0 & f \cdot s_y & p_y \cdot s_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

where f is the focal distance; s_x and s_y are the numbers of pixels per unit distance; p_x and p_y are the principal point coordinates; and s is the skew parameter.

The extrinsic parameters are a rotation matrix $\boldsymbol{R}_{[3\times3]}$ and a translation vector $t_{[3\times1]}$, which describe the camera location and orientation regarding the world coordinate system:

$$\left[\boldsymbol{R}\middle|\, t\right] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]. \tag{2.6}$$

2.2 Depth-Image-Based Rendering

In the case of 3D video applications for home environment, when moving in front of a display, a user expects to see a 3D scene from a different viewpoint. To create such a

filling, in a multiview autostereoscopic display system, consecutive views of a scene are arranged properly in stereo pairs so that when the user moves around, a natural motion parallax can be observed. To provide smooth enough motion parallax, displays must be supplied with a fairly good number of views. The amount of required input data can be reduced by generating virtual views from a limited sub-set of inputs at the display side. Generating novel views from the available information is called view rendering or view synthesis. View rendering techniques are usually classified into three categories according to the amount of geometric information being used: geometry-based rendering, image-based rendering and depth-image-based rendering [18], [19].

Geometry-based rendering is a classical 3D computer graphics approach [22]. Usually, geometry-based rendering exploits 3D models of the objects. These are provided in a form of 3D surface meshes with an associated texture mapped onto them. A drawback of this approach is that human assistance is often required for content creation. Such models may contain millions of polygons. For realistic rendering, various visual effects such as highlights, reflections and transparency need to be imitated and realistic lighting models should be applied, which is often complex and time consuming, and even more complex for dynamic scenes.

On the other hand, image-based rendering does not require any geometric information at all [19]. In this case, instead of 3D geometric primitives, a collection of available real camera views are used to generate virtual intermediate views. Potentially high quality realistic virtual views can be synthesized by interpolation, avoiding any 3D reconstruction and complex visual effects calculations. However, the lack of geometric information has to be compensated by very dense sampling of the real world. If the number of used input views is not high enough, interpolation artifacts will appear affecting the quality of the synthesized views. Thus, data acquisition becomes highly complicated as large numbers of cameras have to be used to capture a scene and huge amount of images have to be processed thereafter.

Depth image-based rendering approaches rely on explicit geometry information for new view generation [20], [21]. The geometric constraints can be for example in a form of disparity or depth maps. Such maps assign a depth value for every pixel in the image. The depth map combined together with the original view provides a 3D-like representation (often called 2.5D). Virtual views can be synthesized from captured scene image and associated depth. Depth-image-based rendering techniques are described in more details in the following sections.

2.2.1 3D Image Warping

Depth-image-based rendering utilizes scene depth information provided in the form of depth maps to synthesize new virtual views. The basic idea of most DIBR rendering methods is to perform 3D warping [23]. 3D warping is a two-step process (Figure 2.2). First, given a reference color image and its associated depth map along with camera calibration information, pixels of the reference image are back projected to their original 3D locations in space using their respective depth values (2D-to-3D). Second, the 3D

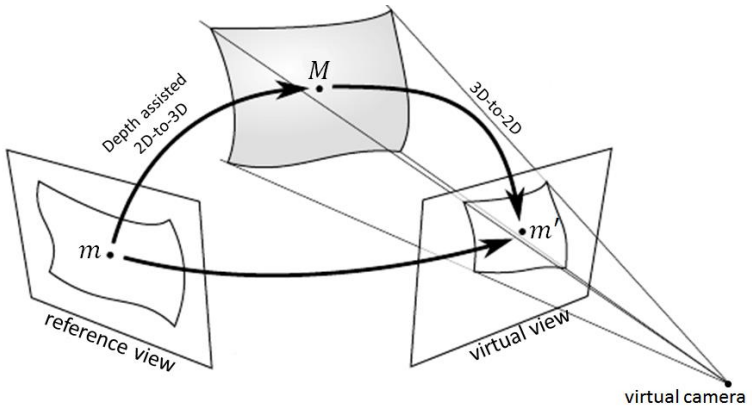

Figure 2.2 3D warping.

points are projected again onto a virtual camera image plane at the required viewing position (3D-to-2D). As a result, the pixels of the reference image are shifted to generate a new view as if they were captured from the virtual viewpoint.

A 3D warping algorithm can be formalized as follows. Let $M = (x,y,z,1)^T$ be a world point and $m = (u,v,1)^T$ be its projection in a camera with projection matrix $\boldsymbol{P}$ in homogeneous coordinates. If a camera is modeled as a pinhole camera, the relationship between M and m is:

$$zm = PM \tag{2.7}$$

where z is depth and $\boldsymbol{P}_{[3\times4]}$ is the camera projection matrix. Using Eq. (2.7), the 3D point M can be reconstructed from the image point m using inverse projection matrix $\boldsymbol{P}^{-1}$ and its depth value z. Then, the reconstructed 3D point is projected to the virtual image plane with the projection matrix of the virtual camera $\boldsymbol{P}$.

Usually, the input pixel locations are not mapped to integer positions of the virtual view after 3D warping and the novel view must be accurately resampled. The main difference between various resampling approaches lies in the interpolation kernel used, e.g. nearest neighbor, linear, cubic spline, etc. Furthermore, disocclusions and other rendering artifacts may appear in the novel views. To avoid artifacts in the newly generated virtual views, DIBR requires efficient techniques to fill these disocclusions. This is discussed in the next section.

2.2.2 Rendering Artifacts

The quality of the virtual view strongly depends on the accuracy of the depth data. After 3D warping, rendering artifacts may appear in virtual views mainly caused by depth discontinuities. The amount of artifacts increases with the distance of a virtual view from the original view. Rendering artifacts result in a reduced visual quality, thus some post-processing should be performed to cope with the artifacts. There are three major types of artifacts [24].

The first type of artifacts is caused by the fact that the size of objects may change depending on the viewpoint, so after warping an object from one viewpoint to another viewpoint, cracks on the object surface will appear (Figure 2.3 left). Cracks can be emp-

Figure 2.3 Rendering artifacts. Left: disocclusions and cracks. Right: contour artifacts.

ty, i.e. containing pixels with no values, or can contain pixels from the background. In most cases, cracks can be filled by applying a median filter (if there are background pixels, they should be detected and ignored).

Secondly, inaccuracy in depth maps, depth-texture misalignments or smooth texture edges causes pixels to be warped to wrong places in the virtual view. Foreground edge pixels, where the discontinuities in depth are high, are warped onto background creating contour artifacts (Figure 2.3 right). When the wrongly warped edge pixels are not removed, the resulting rendered image will have unnatural ghost contours of the foreground objects.

The main drawback of DIBR techniques is that virtual view can contain holes also often referred as disocclusions (Figure 2.3 left). Disocclusions are previously invisible scene parts, which become visible in the virtual view. Disocclusions usually appear at the object boundaries, where the depth discontinuities are larger. Neither color nor depth information related to these regions is known. This is the most problematic type of artifacts to be handled and some advanced inpainting techniques are required to fill in the disoccluded areas.

The disocclusion problem is often considered as missing texture information, which is filled by using inpainting techniques. A number of different inpainting methods can be found in the literature. When the holes are small enough, simple interpolation and extrapolation can offer acceptable results. These would create obvious artifacts for larger holes [25]. In order to get better visual results for large holes, more sophisticated methods based on structural inpainting [26] or textural inpainting [27] are suggested. Another type of algorithms is based on exemplar-based inpainting technique first introduced in [28], which combines both structure propagation and texture synthesis to effectively fill in missing pixels. However, depending on the characteristics of the scene and the size of the holes, all the inpainting approaches produce more or less annoying visible artifacts in the synthesized views [29]. Overviews of inpainting techniques for DIBR can be found in [30] and [31].

2.2.3 Depth-Based Data Formats

The primary design choice of any 3DV system is a 3D scene representation format, as the scene representation consequently affects all other modules in the 3DV system processing chain. On the sender side, it determines requirements for data acquisition, e.g. number and setting of cameras, data extraction algorithms, coding and transmission schemas. On the receiver side, it determines the rendering algorithm, and subsequently the output quality, navigation range, interactivity, etc. Depth-based data representations have a number of advantages over conventional stereoscopic video data formats:

- Backward compatibility to existing 2D TV services.
- Efficient compression capabilities: a depth map adds transmission overhead less than 20% of the original color video bitrate, whereas in case of stereoscopic format, transmitting two views would cause up to 100% overhead.
- The final stereo images are generated during rendering at the receiver side, this allows to adjust the parameters of the DIBR algorithm and to customize the 3D depth impression according to the viewer's personal preferences.

2.2.3.1 Video-plus-Depth

The simplest of the depth based formats is the video-plus-depth format (V+D or 2D+Z) [32], which augments a regular 2D video with its associated depth video. The 2D video provides the color intensity, while the depth video represents per-pixel distances between the camera and points in the scene. The depth map is a gray scale image usually represented with 8-bit unsigned integers, i.e. the value 255 represents the closest point and the value 0 represents the most distant point (see Figure 2.4).

Acquisition and transmission of a high number of views is inefficient, V+D format supports rendering of virtual views by DIBR at the receiver side. However, V+D format can support view synthesis only in a very limited neighborhood of the available original view, as the amount of disocclusion artifacts significantly increases with distance of the virtual view from the available view. Information about the originally hidden scene parts can neither be inferred from the single color image nor from the corresponding depth image, and the missing image parts must be filled during the post-processing.

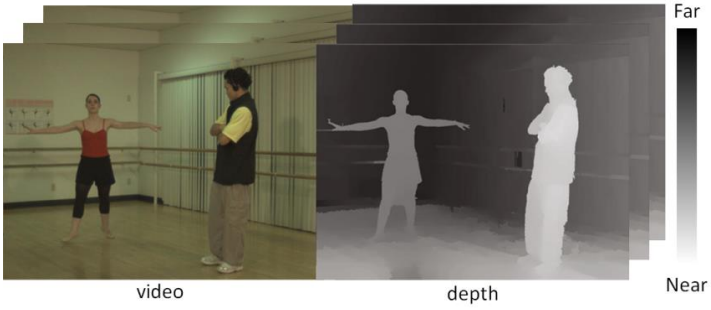

Figure 2.4 Video-plus-depth format.

2.2.3.2 Multiview Video-plus-Depth

The problem of a single video-plus-depth stream is a small navigation range due to the growth of rendering artifacts with the distance from the original view. The V+D format can be easily extended to multiview video-plus-depth (MVD) [33]. MVD consists of two or more V+D data streams. In this case, a scene has to be captured by multiple cameras and depth data has to be estimated for each of the original camera views (Figure 2.5). Such data representation supports generation of the high quality virtual views and extends the potential navigation range for 3DV applications.

For transmission of multiple V+D streams an appropriate multiview coding (MVC) algorithm can be applied. It allows efficient data compression by exploiting the inter-view dependencies [33]. After transmitting and decoding, the received V+D streams are converted to the display views by using DIBR techniques. However, because of the possible difference between capturing and display configurations, such conversions from *N* source views to *M* display views introduce additional practical challenges compared to the single V+D format. First, the number of camera views *N* usually is not equal to the number of display views *M* (for practical reasons, *N*is usually significantly smaller than *M*). This means intermediate views have to be interpolated anyway. Second, the arrangement of the cameras capturing the scene may be different from the view geometry of a 3D display. The virtual views of a 3D display are usually perpendicular to the display surface and evenly spaced, which is hard to achieve in practice for a real capturing rig, due to mechanical inaccuracies and different optical characteristics of the cameras.

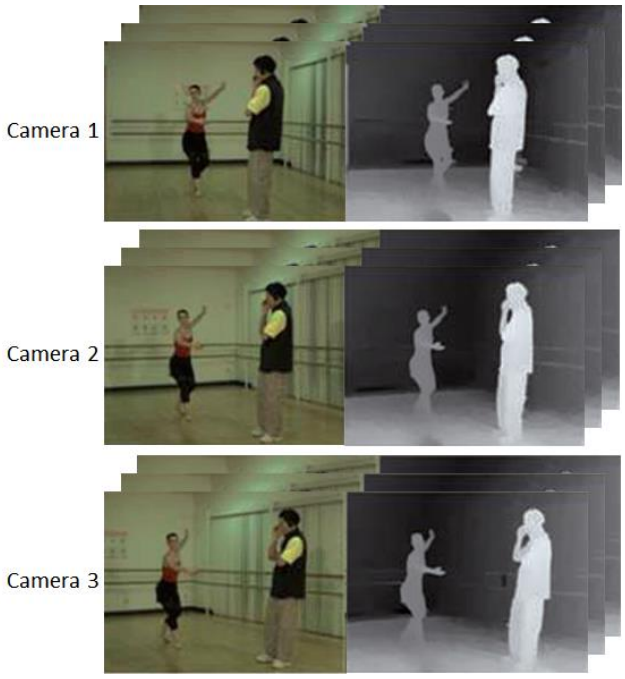

Figure 2.5 Multiview video-plus-depth.

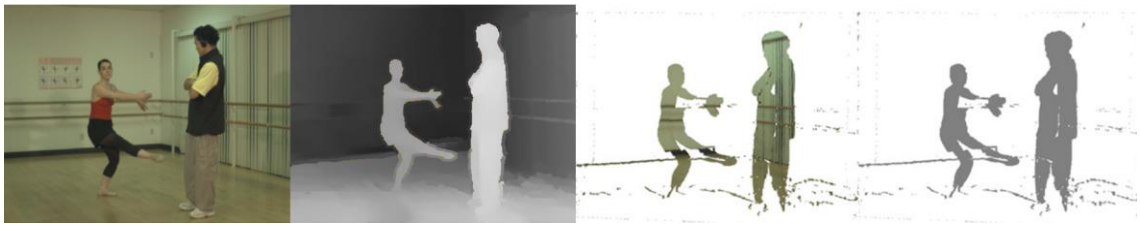

Figure 2.6 Layered depth video with a main and an occlusion layer.

2.2.3.3 Layered Depth Video

Although MVD is a powerful representation format for 3DV, it requires transmission of a vast amount of redundant data even if compression is applied. An alternative data representation is layered depth video (LDV) [34]. LDV representation allows further reduction of the overhead associated with transmission of multiple V+D streams. To do so, the available views are projected onto each other in order to detect and eliminate duplicated information. The reduced information is then stored in different layers: the main layer is one full (usually central) video-plus-depth view, and a number of additional occlusion layers consisting of residual texture and associated depth data of side views (Figure 2.6). Afterwards, the non-transmitted side views are generated by view synthesis, i.e. the main view is projected onto side views by DIBR. This way, LDV representation can achieve a more compact representation of the data compared to MVD.

However, LDV representation also has a number of drawbacks. First of all, the layer extraction for LDV strongly depends on depth quality as it is based on view warping, and is prone to errors in case of inaccurate depth estimation. Furthermore, lighting effects that are different in different views, e.g. shadows, reflections, etc., are ignored in the LDV representation, while preserved in the MVD representation. Finally, when the main view is projected, the generated side view may still suffer from cracks occurrence, as not every pixel necessarily exists in the main view.

2.3 Time-of-Flight Technology

Depth sensing of real world scenes is used in many computer graphics and computer vision applications, such as virtual reality, object reconstruction, intermediate view synthesis, human machine interaction, robotic navigation, etc. At present, there is no off-the-shelf system that can provide real-time high resolution high quality range information. Depth sensing can be carried out using laser scanners or stereo vision systems. While those techniques are able to provide high-resolution, accurate estimates of depth information, they still possess several drawbacks. Systems involving laser scanners, for example, produce high quality measurements by scanning the environment with a laser beam row by row, consequently, because of this processing scheme, laser scanning is rather time-consuming and thus not suitable for dynamic scenes. Stereo vision systems, on the other hand, analyze the scene as viewed in different images to obtain depth measurements, a process, which is strongly depending on the scene's texturing and,

therefore, tending to fail in image regions with homogeneous or repetitive textures, resulting in gaps in the depth measurements.

An alternative, suitable for real-time applications is depth sensing based on the time-of-flight (ToF) principle [3]. ToF cameras work by measuring the phase-delay of reflected infrared light and provide range estimates at each pixel in parallel. These devices are relatively compact and inexpensive, capable to provide depth images at a rate equal to or higher than real-time speed. In contrast to stereo systems, depth accuracy provided by ToF sensors is independent of the scene texturing. Nevertheless, new challenges introduced by the nature of time-of-flight mechanism itself are still to be solved, such as: low sensor resolution (e.g. 200×200 pixels) compared to Full High-Definition (HD) pixel standard (1920×1080), inability to capture color information along with depth and inaccuracies in depth measurements containing systematic and other errors. In practice, before using the range data from a ToF sensor, some pre-processing of the input data is usually required, e.g. denoising and upscaling. Further details on the working principles and error sources in time-of-flight depth measurements are presented in the next two sections.

2.3.1 Operation Principle of ToF Depth Sensors

A typical ToF device consists of an illumination source, an electronic light modulator and a sensor array. The whole scene is illuminated by the illumination source with modulated infrared light. The light reflected from objects in the scene is sensed back by sensing elements of the sensor array. Every pixel in the sensor array independently calculates depth, amplitude and intensity information, by measuring the phase difference between the emitted sinusoidal light wave and incoming signal (Figure 2.7 and [35]).

Assuming ideal sinusoidal modulation of the infrared light source, emitted and incoming signals, denoted as $g(t)$ and $s(t)$ respectively, can be represented as:

$$g(t) = \cos(\omega t); \tag{2.8}$$

$$s(t) = h + a \cdot \cos(\omega t - \Delta\varphi) \tag{2.9}$$

where ω represents the modulation frequency, phase offset $\Delta\varphi$ depends on the duration of light travel to the object and back, additional constant component h is used to model background illumination and an amplitude component a is used to model power loss due to light absorption.

Cross-correlation function between emitted and incoming signals is defined by:

$$c(\tau) = (s * g)(\tau) = \frac{a}{2} \cdot \cos(\Delta\varphi + \tau). \tag{2.10}$$

By sampling the correlation function (2.10) four times at phases with $\pi/2$ shift from each other, phase offset between emitted and incoming signals can be calculated as:

$$\Delta\varphi = \arctan\left(\frac{c(270°) - c(90°)}{c(0°) - c(180°)}\right). \tag{2.11}$$

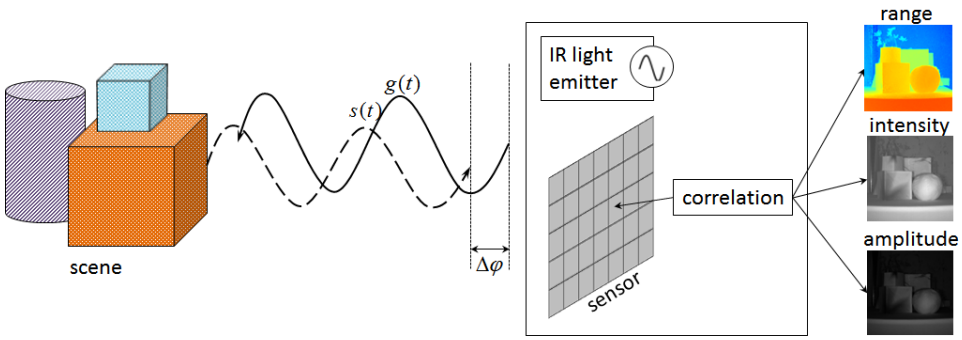

Figure 2.7 Time-of-flight measurement principle.

The depth value d can be derived from the obtained phase information φ as:

$$d = \frac{c_L}{4\pi\omega}\Delta\varphi \tag{2.12}$$

where $c_L \approx 3 \cdot 10^8\, m/s$ is the speed of light.

The amplitude of the received signal a and the background intensity I are calculated by:

$$a = \frac{\sqrt{\left(c(270\,°) - c(90\,°)\right)^2 + \left(c(0\,°) - c(180\,°)\right)^2}}{2}; \tag{2.13}$$

$$I = \frac{c(0\,°) + c(90\,°) + c(180\,°) + c(270\,°)}{4}. \tag{2.14}$$

2.3.2 Error Sources in Depth Measurements

Depth images produced by a ToF sensor contain both systematic and non-systematic errors [3]. Measurement accuracy of a ToF sensor is limited by the power of the emitted signal and depends on many factors, such as light intensity, different reflectivity of surfaces, distances to objects in a scene, etc.

Systematic errors are directly related to the time-of-flight working principle introduced in the previous section. Since calculation of the distance assumes a perfectly sinusoidal light source, which in practice is not possible, the measured depth contains an error component, referred to as "wiggling". Possible methods to correct systematic depth errors are discussed in [3], e.g. technique based on look-up-tables [36] or B-splines approximation for the error [37].

Non-systematic errors are related to the scene-content. So called "flying pixels" can be observed in regions with inhomogeneous depth, e.g. at object boundaries. Because of low sensor resolution and systematic noise, pixels that observe object boundaries get mixed signals leading to wrong distance values falling in between foreground and background. Motion artifacts are caused by either camera or object motion. Cross-correlation for phase estimation requires sampling of the incoming signal at least four times, thus depth values at object boundaries become erroneous for a dynamic scene. Other errors are intensity related distance errors, as the amplitude of the reflected signal can also varies depending on the material and color of the object surface. Object areas with very low

reflectivity or objects far from the sensor generate a low signal, while areas with high reflectivity may cause over-saturation.

The problem of denoising of ToF data has been addressed in a number of works [38], [39], [44]. Modern denoising approaches, such as edge-preserving bilateral filtering [40] and non-local (patch based) filtering [41], have been modified to deal with ToF data [38], [44]. It has been proved that the error in distance measurements is proportional to the square inverse of the amplitude and measured amplitude can serve as an estimate of the reliability of depth measurements [42], [43]. The technique proposed in [44] utilizes a non-local means filtering [41], modified to work in a complex-variable domain. Phase delay and amplitude of the sensed signal are regarded as components of a complex-valued variable and processed together in a single step. This imposes better filter adaptivity and weighting with reduced computational complexity, and additionally improves the noise confidence parameter given by amplitude.

A specific case of interest is the so-called low-sensing mode, in which the sensor is more restricted, e.g. by size restrictions leading to limited beamer size and decreased number of light-emitting elements, requirements for low power consumption, etc. For such mode the noise presence becomes a dominant problem, which should be addressed by dedicated denoising methods [45], [46].

2.4 3D Computer Graphics Basics

In computer graphics, rendering is the process of producing a digital image based on three-dimensional scene description. A 3D rendering algorithm takes as input a stream of primitives (triangles, lines, points, etc.) that define the geometry of a scene. The primitives are processed in a series of steps, where each step forwards the results to the next one. The sequence of steps required to implement the rendering algorithm is called rendering pipeline or graphics pipeline [47]. Some components in the modern rendering pipeline are fixed and implemented using dedicated logic, while others are programmable and can be provided to a graphical processor in a form of special shader programs. Fixed parts of the pipeline are controlled through graphics APIs, such as OpenGL and Direct3D, which modify the rendering state and supply input to the pipeline. In the thesis project, OpenGL library is used. Using shaders, it is possible to customize the functionalities of the graphics pipeline. Programming shaders is possible with high level shading languages; e.g. OpenGL Shading Language (GLSL) [48], which is based on the syntax of the C programming language.

2.4.1 Programmable GPU Rendering Pipeline with OpenGL

A simplified representation of the modern 3D pipeline is shown in Figure 2.8. The dark boxes stand for programmable parts, while the others provide fixed functionality.

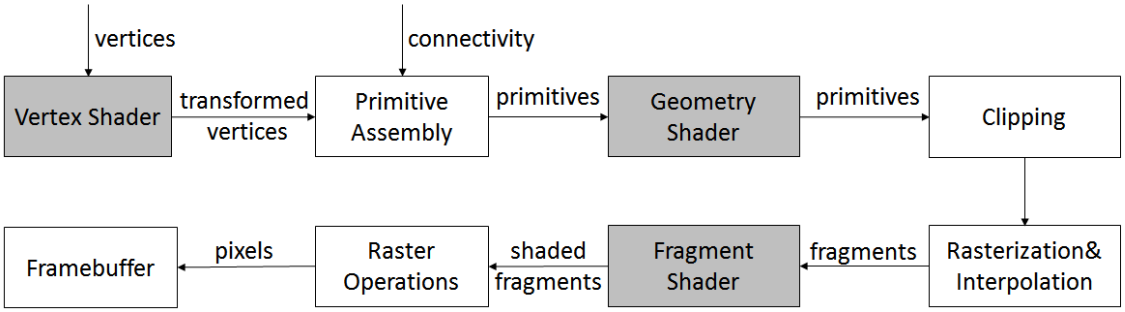

Figure 2.8 Simplified rendering pipeline.

The input of the rendering pipeline is a set of geometric primitives. In most cases triangles are used but also rectangles or other polygons are possible. A primitive is defined by its vertices. Each vertex contain information about vertex properties, this may include: location of the vertex in space, surface normal vectors, texturing and coloring information, and material properties.

A vertex shader program can perform arbitrary operations on individual vertices, i.e. it runs once for each vertex given to the graphics processor. A vertex shader can manipulate properties such as position, color and texture coordinates (vertex coordinate transformations are typically performed at this stage). All the computations for one vertex have to be independent from all other vertices. The vertex shader cannot create new vertices: for each input vertex, the shader has to output exactly one output vertex.

The primitive assembly stage is not programmable and it is responsible for assembling the transformed vertices into proper primitives. The output of assembly process is an ordered sequence of simple primitives (lines, points, or triangles). The type of the output primitives depends on the type of the primitives provided by a user in the rendering procedure call.

A geometry shader is an optional stage of the pipeline. If presented, it takes as input a whole primitive and has access to all the vertices that make up the primitive. Vertex information can be provided also for adjacent vertices. The output of a geometry shader can be zero or more primitives: some primitives can be filtered out or new primitives can be generated, e.g. to optimize speed or to increase rendering quality.

The clipping stage divides partially visible primitives into their visible and invisible parts. Here “invisible” means those parts of primitives that lie outside the area that is visible in the final frame. This process reduces the workload for the following stages.

Rasterisation and interpolation are fixed stages of the pipeline. Rasterisation maps portions of primitives to pixels on the screen, i.e. defines a set of pixel-size fragments that are part of a primitive. Fragments are used to compute the final data for a pixel in the output framebuffer. Every fragment includes its position in screen-space and a list of vertex attributes (position, color, etc.) that were output from the previous stage: for each fragment in a primitive, values of these attributes are interpolated between the data values of the vertices composing the primitive. Commonly, the value of the attribute is calculated as a weighted average between the attribute values of the primitive vertices.

Fragment shader (or pixel shader) computes the final color value of each fragment. It can access the fragment position, and all the interpolated data computed in the rasterisation process (only current fragment and its associated data). It can change the depth of a fragment and use textures to produce special effects such as fog, lighting, etc. Moreover, fragment shader can act as a postprocessor or filter for the output data after it has been rasterized. Although fragment shader can only operate on a single fragment, the screen coordinate being drawn is known. So if the content of the entire screen is passed as a texture to the shader the values of the nearby pixels can be accessed. This allows a wide variety of post-processing effects, such as blur, enhancement, etc.

Raster operations are a sequence of stages, which determine the final color of pixels based on the fragments produced by the fragment shader. First various culling tests are performed to determine if the fragment is visible and needs to be added to the framebuffer. Then blending is performed to blend the color of the fragment with the color of the already rendered pixel.

2.4.2 OpenGL Coordinate System Transformations

Just like the rendering pipeline, transforming vertex coordinates from the original 3D coordinates to 2D screen coordinates is done step-by-step. Each transformation maps vertex coordinates onto a new coordinate system, moving to the next step (Figure 2.9). There are multiple coordinate systems involved in 3D graphics:

- Object space;
- World space (Model space);
- Camera space (Eye space/ View space);
- Screen space (Clip space).

Object space is the local coordinate system of a geometrical object. The model matrix transforms a position in the object coordinates to the desired position in world coordinate system. Usually it is used to move the object somewhere in the world: every vertex of the object is multiplied by the model matrix, which transforms the vertex to its new position and orientation.

The view matrix transforms world coordinates into eye coordinates, i.e. it corresponds to placing and orienting the observed scene relative to the camera position (or the eye of an observer). The camera is an abstract thing, which is placed at the origin of the camera coordinate system, looks down the negative direction of the *Z*-axis and its up-direction is given by the positive *Y*-axis.

The projection matrix transforms eye coordinates into clip coordinates. It determines how a 3D vertex coordinate is projected on a 2D screen coordinate and the viewing volume of the scene (frustum). The resulting coordinates are called clip coordinates because all parts of primitives that are outside the frustum are clipped away. It should be the last transformation that is applied to a vertex in a vertex shader.

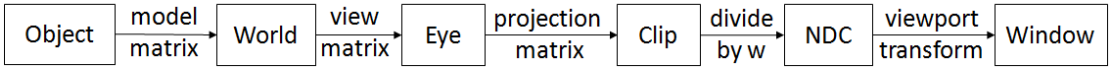

Figure 2.9 OpenGL transformations.

The modeling, viewing and projection transformation are applied in the vertex shader. For a vertex represented by homogenous coordinates $v = (x, y, z, w)$, w=1, the final vertex transformation is the product of these three matrices:

$$v_{clip} = \boldsymbol{M}_{proj} \cdot \boldsymbol{M}_{view} \cdot \boldsymbol{M}_{model} \cdot v\,. \tag{2.15}$$

Next, each clip coordinate is transformed into a normalized device coordinate (NDC) by perspective division, which is dividing the clip coordinate by w_{clip}. Perspective division is automatically applied in the fixed-function stage after the vertex shader. The resulting range of values is from -1 to 1 in all three axes.

$$\begin{pmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \end{pmatrix} = \begin{pmatrix} x_{clip} / w_{clip} \\ y_{clip} / w_{clip} \\ z_{clip} / w_{clip} \end{pmatrix}. \tag{2.16}$$

NDC coordinates are then mapped to screen coordinates (or window coordinates) by the viewport transformation. The viewport transformation is also applied automatically in the fixed-function stage. Finally, the window coordinates are passed to the rasterisation process to become a fragment.

2.4.3 Calibrated Cameras in OpenGL

When working with data originated from real calibrated cameras, it is often useful to be able to display things on screen from the point of view of a virtual camera that resembles properties of a real camera providing the data. To simulate a calibrated camera in OpenGL original camera's parameters (focal length, principal point coordinates, etc.) should be incorporated into OpenGL projection matrix, which defines how 2D projection of a 3D scene is formed [49].

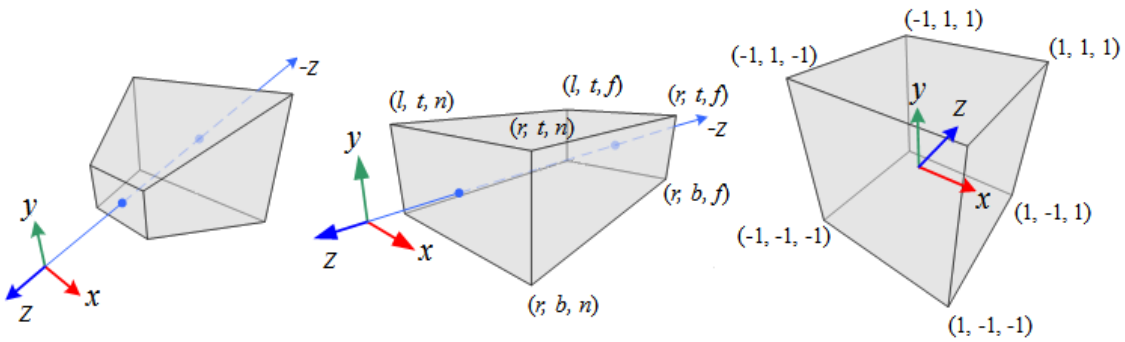

Figure 2.10 Projective transformation: Frustum→Cube→NDC.

Given that the calibration matrix $\boldsymbol{K}$ is known, OpenGL projection matrix can be calculated by multiplying two matrices:

$$\boldsymbol{M}_{proj} = \boldsymbol{M}_{NDC} \cdot \boldsymbol{M}_{persp}. \tag{2.17}$$

The $\boldsymbol{M}_{persp}$ matrix converts a frustum-shaped space into a cubic shape, and $\boldsymbol{M}_{NDC}$ converts the cubic space to normalized device coordinates (Figure 2.10). The intrinsic camera matrix $\boldsymbol{K}$ specifies the camera perspective projection transformation and can be incorporated into $\boldsymbol{M}_{persp}$ as follows:

$$M_{persp} = \begin{pmatrix} f_y & 0 & p_y & 0 \\ 0 & f_x & p_x & 0 \\ 0 & 0 & near + far & near \cdot far \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2.18}$$

where f_x, f_y are focal length in pixels, p_x, p_y are principal point coordinates, *near* and *far* correspond to back and front clipping planes of the viewing frustum. While $\boldsymbol{M}_{NDC}$ should be of a form:

$$\boldsymbol{M}_{NDC} = \begin{pmatrix} \frac{2}{right - left} & 0 & 0 & \frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & \frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{-2}{far - near} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.19}$$

The choice of *top*, *bottom*, *left*, and *right* clipping planes correspond to the dimensions of the calibrated camera sensor and the coordinate conventions used during calibration.

2.5 Summary

Basic principles and concepts related with the research objectives of this thesis were presented in this chapter. The main accent in the thesis is made on real-time realization of DIBR techniques and practices utilizing fast ToF depth sensors and the computational power of modern GPUs. ToF sensors are able of delivering depth images in real-time, however, numerous errors of various kind as well as other issues associated with the ToF sensing principle need to be taken into account and handled. One major drawback besides noise presence in depth measurements is the missing color information from the sensor: it does not allow capturing color and light from the scene as a conventional video camera does. In order to deal with the limitations of the ToF sensor, various combinations of high resolution video cameras with ToF sensors are possible. The following chapter presents such a system. The quality of the recorded depth information is enhanced by applying a dedicated denoising procedure and by further fusing depth data with the high-resolution color modality.

3. SYSTEM OVERVIEW

This chapter presents a system-level overview of the proposed 3D video system capable of live 3D video rendering on a multiview autostereoscopic display and arbitrary view synthesis. The entire processing chain, including data capture, image processing algorithms and rendering, is composed into a unified end-to-end framework and runs in real time. The input data is received from a multisensor setup combining a ToF depth sensor and a high-resolution color camera. The ToF depth sensor can provide depth information in real-time, while the color camera is added in order to capture the color information. The input data from the camera setup is being processed in a series of steps in order to produce a view-plus-depth frame for an autostereoscopic 3D display or to generate a novel view of the captured scene from an arbitrary viewpoint. Processing power of modern GPUs and their programming tools enable for efficient high speed realization of data processing and rendering algorithms.

The chapter is organized as follows. The overview of the system architecture and key functionality is given in Section 3.1. Section 3.2 describes the design and calibration of data capturing setup that enables simultaneous recording of dynamic scene. Details on data streaming between the cameras and the application are provided in Section 3.3. Finally, technical details and the interface specification of the autostereoscopic display are given in Section 3.4.

3.1 System Design and Functionality

The full 3D video processing chain from data capture to visualization on the 3D display is composed into an end-to-end system. Figure 3.1 shows the principal block diagram of the implemented 3D video system. The system software implementation is done in C++ using cross platform libraries, namely OpenCV and OpenGL. The hardware specification of the system includes Intel Core i7-3770 CPU and graphics card equipped with NVIDIA GeForce GT640 GPU, 64-bit Windows 7 operating system. Technically, the system implementation can be divided into two functional modules CPU-based data acquisition and GPU-based data processing and rendering. Each functional unit in the framework is presented in detail within the current and the next chapters.

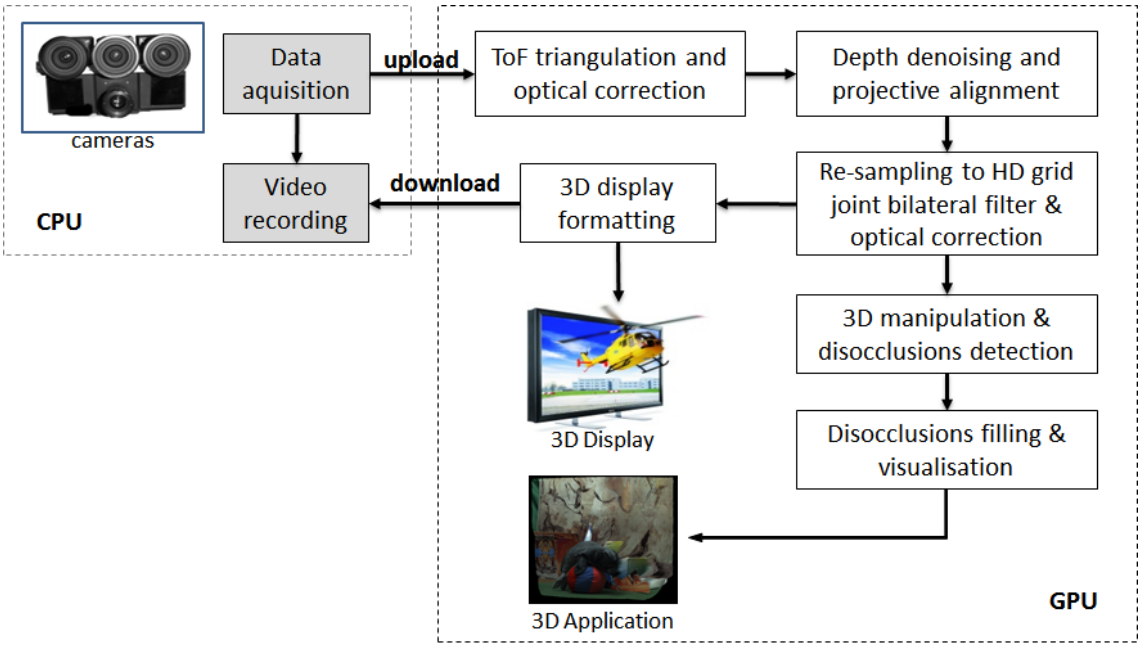

Figure 3.1 3D video system design.

The acquisition module provides a software-synchronized data acquisition from the camera setup and uploads the data to the GPU for further processing as explained in Section 3.3.The code responsible for camera interfacing and data capturing is implemented in a dedicated class and is executed in a separate process using a simple interface for the data transfer and synchronization with the main program. The main program is responsible for creation of OpenGL context and for uploading the data to the GPU memory, as well as for managing the rendering loop and processing user input. A simple user interface is implemented to provide the ability to interactively switch between rendering modes and adjust various system and rendering parameters, such as camera capture settings, display 3D depth impression, calibration parameters of the camera setup, OpenGL virtual camera position, parameters of image processing algorithms, etc. Available system controls are summarized in Table 3.1.

The data processing and rendering modules are implemented solely on GPU. The stages of the data processing chain are implemented as shaders using OpenGL Shader Language (GLSL). Being uploaded to GPU, a series of processing steps including denoising, undistortion, and data fusion is performed in order to generate a dense depth map for the input color image. Finally, the color and corresponding depth map are combined in a video-plus-depth frame to be displayed on the 3D display. As an alternative, fused data can be rendered as 3D textured surface mesh for viewing from an arbitrary viewpoint. In this case, after the dense depth map has been obtained, an additional post-processing step has to be performed to fill in disocclusions inevitable during the view synthesis. Rendered output can be downloaded from the GPU back to system memory and saved on the disk.

Table 3.1 User interface controls

Control key	Description
Camera capture settings	
i/I e/E	Increase/Decrease ToF sensor integration time Increase/Decrease 2D camera exposure time
Camera calibration parameters tuning	
x/X, y/Y, z/Z p/P, w/W, r/R	Adjust translation vector between cameras in the setup Adjust rotation parameters of color camera relative to ToF sensor
Virtual camera position manipulation	
Mouse left button Mouse right button Mouse wheel	Rotate virtual camera Translate virtual camera by X or Y Translate virtual camera by Z
Processing chain controls	
b d h Space	On/Off joint bilateral filter for depth map On/Off disocclusion filling On/Off Dimenco header (switch between 2D and 3D mode) Switch between textured surface and view-plus-depth representation
Algorithms parameters	
t/T j/J, k/K, l/L	Increase/Decrease threshold parameter τ for disocclusion detection Increase/Decrease parameters of JBF: radius, range σ_r, spatial σ_s.
Other	
Arrows ←,→, ↑,↓ f v ? q	Control display offset and depth impression On/Off full screen mode Video recording Print interface and current system parameters Exit program

3.2 Camera Setup and Calibration

The multisensor system used in this work for the experiments consists of a PMD[vision]® CamCube 2.0 sensor for depth measurements and an Allied Vision Technologies Prosilica GE 1900C camera, which is used to obtain color image (Figure 3.2 left and middle). Binocular camera setup is used: color camera is mounted on top of the PMD sensor as presented in Figure 3.2 right (although there are three color cameras in the picture, only the center camera is used in this work). The PMD camera has two light sources symmetrically placed on both sides of the sensor array to illuminate the scene. It can operate in a range up to seven meters. The camera sensor array has resolution of 204×204 pixels: for each pixel the camera delivers distance, amplitude and intensity information simultaneously with a frame-rate up to 28 fps. Technical characteristics of the cameras used are summarized in Table 3.2.

Figure 3.2 Left: Prosilica GE 1900C. Middle: CamCube 2.0. Right: camera setup.

Table 3.2 Camera technical specifications

Model	**PMD CamCube 2.0**	**Prosilica GE 1900C**
Type	time-of-flight	Color
Resolution [pixels]	204×204	1920×1080
Measurement range [m]	0.3-7.0	-
Frame rate [fps]	28	30
Field-of-View [°]	~39	~55
Data Interface	USB 2.0	Gigabit Ethernet

The cameras come with their own C++ software development kits [50][51]. The SDK allows capture triggering and controlling variety of camera capture properties such as exposure time, white balance, frame rate, resolution, and more. Due to the different fields of view (see Table 3.2), the two sensors observe slightly different regions of the scene. Thus, the capturing settings of the color camera had to be justified with respect to the PMD camera field of view. In order to acquire an appropriate overlapping area of color and depth data of the captured scene, the color camera was set to capture 800×800 central part of the available full resolution.

Data fusion process of distance information and high-resolution color information requires reliable estimation of the relative positions of the cameras and their internal parameters. Moreover, most of the algorithms involved in depth estimation assume data with no optical distortions. Thus, a camera calibration procedure has to be performed in order to estimate all necessary camera parameters so that misalignments, optical distortions and so on can be corrected in software.

Once cameras are set up they need to be calibrated intrinsically for focal length, principal point and distortion coefficients and extrinsically for relative orientation between the cameras. Camera parameters can be estimated by taking a set of images: when certain image coordinates from the set of images are known exactly, some pixels among images can be matched, which allows to estimate projection matrix and then find intrinsic and extrinsic parameters. Deduction of camera parameters from image sets is referred to as *calibration problem*. Point correspondences can be obtained by imaging a simple pattern with certain easy-to-extract features, e.g. a checkerboard.

ToF sensors use standard optics to focus the reflected light onto the sensor array, thus, effects like lens distortions and shifted optical center need to be compensated, i.e. intrinsic calibration is required. Time-of-flight sensors provide low-resolution intensity images, which makes calibration possible. Camera Calibration Toolbox for Matlab [52] was used to calibrate the cameras. The calibration is performed by using the calibration technique presented in [53] and lens distortion coefficients are calculated by the method presented in [54]. First intrinsic and extrinsic parameters of each camera are calculated independently, an after that, the extrinsic parameters of the stereo system characterizing the relative location and orientation between the two cameras are estimated.

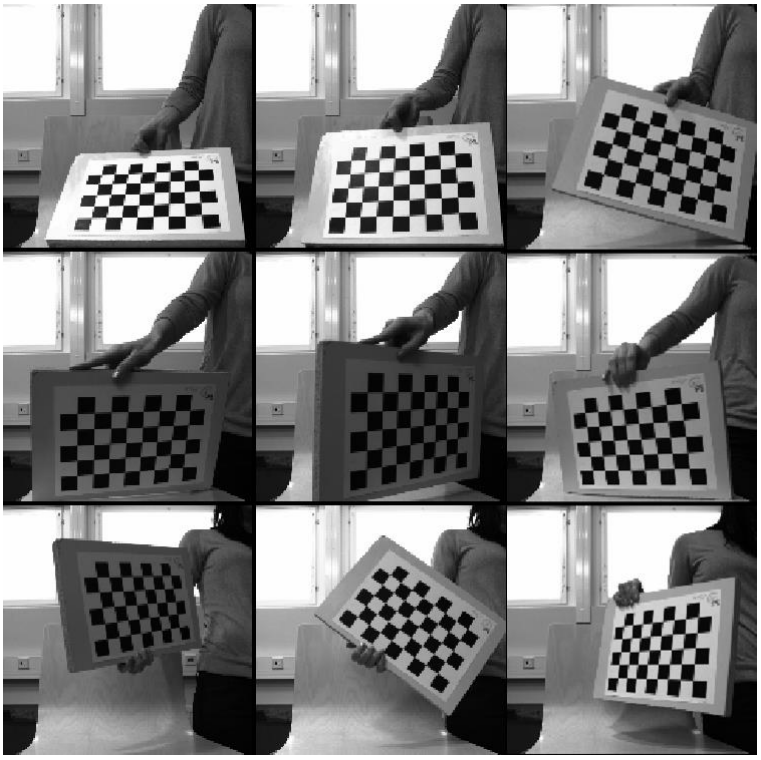

Figure 3.3 Captured calibration pattern with different orientations.

The calibration technique requires a camera to observe a checkered pattern shown at different orientations. A set of 30 images was taken simultaneously from the ToF sensor and the color camera (nine of them are shown in Figure 3.3). For the ToF sensor, intensity images taken with an appropriate integration time were used. The calibration pattern should be placed such that it can be fully and clearly observed by both cameras. The more pictures are taken, the higher the accuracy of the result is. The pattern consists of black and white 30×30 mm squares. Point correspondence calculation requires manual click on pattern's corners, after that internal and external parameters of a camera can be estimated. Several iterations can be taken: the pattern's corners coordinates can be recomputed using estimated parameters, and parameters can be estimated again. Table 3.3 includes obtained intrinsic calibration results for ToF sensor and color camera and the relative orientation for the color camera with respect to the ToF sensor. Due to the fixed setup, these parameters have to be determined only once during an initialization stage.

Table 3.3 Calibration results.

Model	**PMD CamCube 2.0**	**Prosilica GE 1900C**
Focal Length [mm]	(284.76, 287.45)	(1116.48, 1116.1)
Principal point [pixel]	(101.86, 100.75)	(403.94, 406.97)
Distortion coefficients	-0.43, 0.33, 0.006, -0.008	-0.13, -0.03, -0.002, 0.009
Rotation vector[°]		-0.001, -0.046, 0.004
Translation vector [mm]		9.47, 63.786, -4.234

After the calibration, the intrinsic and extrinsic parameters of the system are as follows:

$$\boldsymbol{K}_{TOF} = \begin{bmatrix} 284.76 & 0 & 101.86 \\ 0 & 287.45 & 100.75 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{K}_{2D} = \begin{bmatrix} 1116.48 & 0 & 403.94 \\ 0 & 1116.1 & 406.97 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\left[\boldsymbol{R}|t\right]_{TOF\rightarrow 2D} = \left[\begin{array}{ccc|c} 0.9989 & -0.0040 & -0.0460 & 9.47 \\ 0.0040 & 1.0000 & 0.0009 & 63.786 \\ 0.0460 & -0.0011 & 0.9989 & -4.234 \end{array}\right].$$

3.3 Data Capture and Streaming

A two-sensor setup is used for scene capture. It is important that the two captured data streams from the color camera and ToF sensor are synchronized in time. Without time synchronization, as the observed scene is dynamic, even a slight difference in the capture time may introduce visual artifacts to the 3D output due to temporal data inconsistency. To ensure synchronized data acquisition, software triggering mode is used for both cameras. When this mode is enabled, a software command triggers the capture of an image, which is then transferred to the PC. To minimize the time-lag between the two capture triggering events, a separate thread is created to trigger each camera whenever the system is ready to update. Triggering is performed as an atomic operation, i.e. no interruption is allowed in between, so both cameras receive the signal to start capturing effectively at the same time and no delay is observed between the input two images.

In order to decouple the acquisition and processing parts, data transfer between the cameras and application is organized in producer-consumer model using queue buffers and fixed amount of reusable frames (Figure 3.4). Both producer and consumer run on CPU. Whenever an unused frame is available, the producer thread fills the frame with new input data and pushes the updated frame into the ready-to-use queue. The frames are popped off the ready-to-use queue by the consumer thread, which task is to upload the frame data to the GPU memory where the data is being processed and rendered. Mutual exclusion of producer and consumer treads accessing the buffers is maintained using a mutex. After the data uploading, the frame is pushed into the queue of available frames and can be refilled again by producer thread. This allows data transfers between the cameras and CPU to occur concurrently with data transfers between the CPU and GPU.

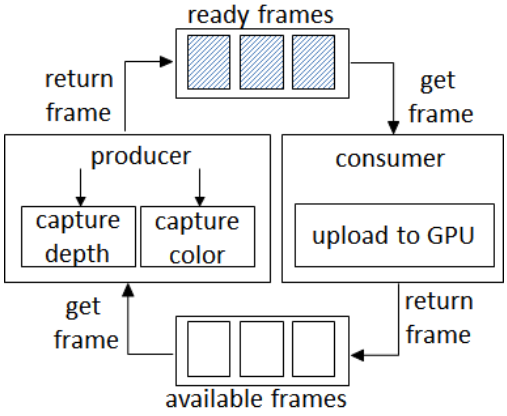

Figure 3.4 Capturing and streaming.

3.4 Autostereoscopic Display

In this work, an autostereoscopic multiview Dimenco display is used for 3D video rendering and display [55]. This section provides technical details about the display, its input data format and multiview rendering process. Technical characteristics of the display are summarized in the Table 3.4.

Table 3.4 Technical characteristics of Dimenco display

Model	**Dimenco 42 Inch**
Type	Multiview lenticular
Diagonal screen size	42 inch
Panel resolution	1920×1080 pixels
Aspect ratio	16:9
Input Format	2D-plus-Depth
Image type	true color (24 bit)
Depth quantization	0-255(8 bit)
Number of views	28
Data Interface	DVI

The input data format of the Dimenco display is video-plus-depth format, which is referred to as ‘2D-plus-depth’ in Dimenco’s technical documentation. The display also supports layered depth video format with a single occlusion layer, which is referred as ‘Declipse’ format and is realized as an extension of the 2D-plus-Depth format. A display frame has to contain the following data:

- Header;
- 2D sub-image of resolution 960×540;
- Depth sub-image of resolution 960×540;
- Background 2D sub-image of resolution 960×540;
- Background Depth sub-image of resolution 960×540.

In case of the Declipse format, the latter two sub-images contain information about background areas occluded by the foreground object, enabling the rendering algorithm to fill in the occluded areas. The 2D and Depth sub-images of the foreground and back-

ground data are line interleaved, i.e. every second line is filled with the corresponding data from the background. For the 2D-plus-Depth format the background lines are not used for rendering. In any case a line should be inserted below each line of the 2D and Depth, but it can contain any information or just be left blank. This way vertical resolution doubles from 540 to 1080. Frame layout is depicted in Figure 3.5.

As the depth map is a grayscale image, the display uses only red sub-pixels of the Depth sub-image and green and blue sub-pixels are discarded. The depth sub-image contains disparity values with a range of 0 to 255, where a value of 0 corresponds to objects at the furthest distance and 255 corresponds to objects located closest to the viewer. Here disparity refers to the differences between the images perceived by the left and the right eye, which human's brain uses as a binocular cue to determine depth of an object. According to Dimenco interface specification [55], the disparity values should be calculated from depth values as follows:

$$d = \mathrm{M}\cdot\left(1-\frac{\mathrm{VZ}}{z-\mathrm{ZD}+\mathrm{VZ}}\right)+\mathrm{C} \tag{3.1}$$

where d is disparity; z is normalized depth [0, 1]; ZD is the depth of display plane; VZ is view distance in coordinate units; M and C are constants. For a 42 inch Dimenco display the optimal values for the disparity calculation are: ZD = 0.467, VZ = 7.655, M = -1960.37, C = 127.5 [55].

The header is located in the upper left corner of a frame, i.e. beginning of the 2D sub-image first line. The header indicates the type of content being displayed and contains settings for rendering processing to customize the 3D depth impression. The display has two operation modes: 2D mode and 3D mode. When the display detects the header it switches to 3D mode. When no header is detected, the display switches back to 2D mode. The display interprets the header of each frame, so changes in header values directly change the displaying mode.

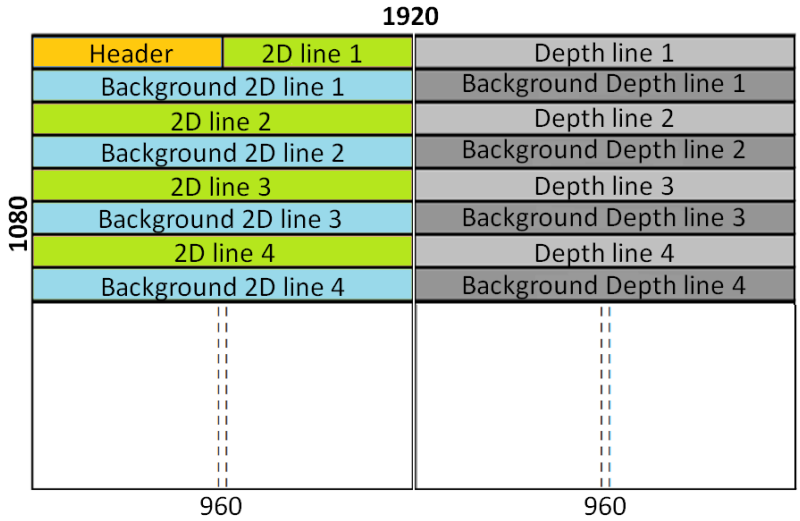

Figure 3.5 Dimenco's 3D frame layout.

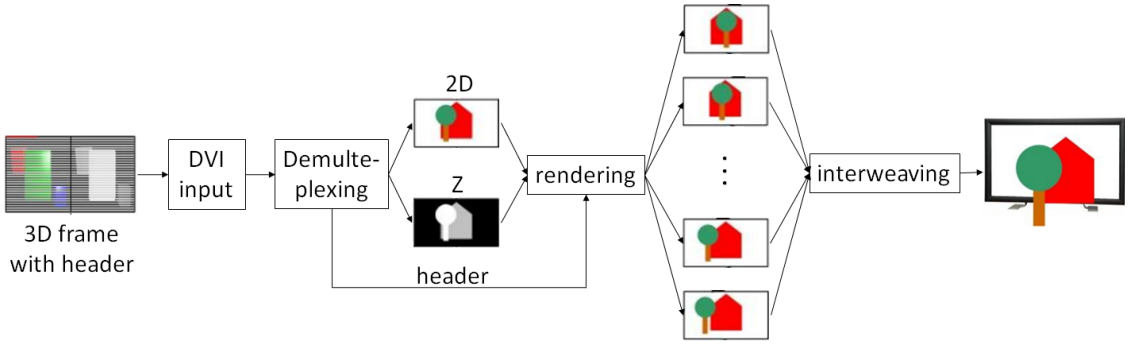

Figure 3.6 Dimenco's 2D-plus-depth to multiple views rendering flowchart.

When the Dimenco display receives a 3D frame, the demultiplexing block decomposes it into the header and the 2D and Depth sub-images. The rendering block generates multiple views, corresponding to slightly different camera positions. The amount of perceived depth and other depth related parameters are controlled by the values in the header. Generated views are interweaved by the interweaving block, which ensures that each sub-pixel is located under the right lens for the best 3D experience. Finally, interweaved images are sent to the auto-stereoscopic multiview display. Flowchart of a 2D-plus-depth frame processing within the display can be seen in Figure 3.6.

3.5 Summary

The overview of the system design and functionality, the main components composing the system and the necessary camera setup calibration procedure have been presented in this chapter. For the system calibration, a software package widely adopted for calibration of conventional cameras and stereo systems was used [52]. While it provides sufficiently accurate results for the standard camera, the accurate intrinsic calibration of the ToF sensor is more complicated, mostly because of its low resolution, relatively high noise level and systematic errors in measurements, which lead to not very accurate intrinsic calibration results and consequently to roughly calibrated extrinsic systems parameters. Thus, as a part of future work ToF-specific calibration techniques, e.g. [56], [57], should be considered in order to obtain more accurate calibration parameters. For now, functional buttons were added to the user interface in order to be able to manually tune the extrinsic system parameters.

Further, as a part of future work, the two side-cameras presented in the camera setup can be included in the scene-capture scenario. The additional scene information captured from the different viewpoints can be used to enhance the low-resolution ToF depth with the high-resolution depth-from-stereo estimations, as well as for the faster and more realistic disocclusions filling in case of the virtual view generation. However, the increased number of the sensors will cause additional calibration and capture synchronization challenges. Furthermore, the current state-of-the-art algorithms for depth-from-stereo estimation [58] are of high computational complexity and hard to implement within the real-time framework. Thus, an optimal trade-off between quality and computational complexity has to be found.

4. DATA PROCESSING

This chapter describes all steps of the processing chain for video-plus-depth content creation. The video-plus-depth data is obtained by fusing high-resolution color image with low-resolution ToF distance data, acquired using a binocular camera setup. Video-plus-depth format requires color and depth data corresponding to the same viewpoint and of the same spatial resolution and. Therefore, the main goal of the data fusion process is the calculation of a dense depth map for the high-resolution color image. The color and corresponding depth data accurately align in a common video-plus-depth frame is further processed within the 3D display using DIBR techniques in order to render multiple virtual views of the observed scene. The quality of the virtual views generated by DIBR algorithms depends on the accuracy of the depth data. Using a computer graphics based approach, the system takes advantage of a 3D mesh representation of the scene, which can be obtained from the ToF depth to generate the depth map corresponding to the viewpoint of the color camera.

Furthermore, to provide free-viewpoint functionality, a 3D scene is generated using mesh triangulation with depth information obtained after data fusion process. Color image is then used to texture the 3D surface. High rendering speed of such mesh-based representation allows reconstructing 3D dynamic scenes in real time. Virtual views can be synthesized by rendering the 3D scene geometry from the requested viewpoint. However, the mesh-based representation causes rubber-sheet artifacts at the virtual viewpoint covering disoccluded areas. Thus, disocclusion areas still need to be detected and filled in a proper fashion.

The data processing is implemented inside the graphics pipeline on GPU. By using the shaders, the full processing chain is incorporated in a several rendering passes, which are summarized in Table 4.1. Both inputs from color and depth sensors are stored on the GPU memory in 2D texture format [59]. The output of each intermediate rendering pass is also saved as a texture, so that it can be accessed later inside the following rendering pass. At each processing step, different algorithms/techniques are applied. The algorithms used for data processing are well suited for a parallel implementation as all the computations are performed for each pixel or vertex without data dependencies.

Section 4.1 describes the method used to denoise the initial ToF depth data. Section 4.2 explains the steps aimed at generating dense depth map corresponding to the viewpoint of the color camera. The problem of optical distortions presented in the input data is covered in the Section 4.3. The Section 4.4 provides details on how the final view-plus-depth frame is rendered in the format recognized by the 3D display. An arbitrary view rendering from view-plus-depth data is also described in Section 4.4. Section 4.5

addresses the problem of disocclusions appearing in the rendered virtual view, while Section 4.6 presents an evaluation of the system's performance.

Table 4.1 Multipass data processing.

	Task	Input	Output
I pass	Vertex Shader: - Denoising; - Data reprojection (alignment with color camera)	- ToF depth image; - ToF amplitude image; - ToF intensity image; - Bilateral filter parameters; - Calibration parameters	Low-resolution depth map corresponding to the color camera viewpoint
II pass	Fragment shader: - Upsampling; - Joint bilateral filtering	- Low-resolution depth map corresponding to the color camera viewpoint; - Bilateral filter parameters	High-resolution depth map corresponding to the color camera viewpoint
III pass (for view-plus-depth mode)	Fragment shader: - Header encoding; - Depth map to disparity map conversion	- Color image; - Corresponding depth map	View-plus-depth frame
III pass (for 3D surface mode)	Vertex shader: - Data reprojection; - Texture mapping; Geometry shader: - Disocclusion detection	- Color image; - Corresponding depth map; - Position and orientation of the virtual camera;	Arbitrary view of 3D scene model
IV pass (for 3D surface mode)	Fragment shader: - Disocclusion filling	Arbitrary view of 3D scene model with disocclusions	Arbitrary view of 3D scene model without disocclusions

4.1 Depth Denoising

Due to the principles of operation of the ToF range sensor, a significant amount of noise is present in the captured range data (see Section 2.3.2). Prior to the actual steps of fusion, denoising of depth data is performed in order to reduce the noise and remove outliers. A popular denoising method commonly used for ToF data is the bilateral filter [60], [61]. The key idea of bilateral filter is that for a pixel to influence another pixel it should not only occupy a nearby location (as it is in standard Gaussian filter) but also it should have a similar value. The advantage of this technique is that it preserves depth discontinuities while smoothing continuous regions. The bilateral filter is a combination of spatial and range weightings: the intensity weighting for similar pixels preserves the edges whereas the distance dependent weighting preserves the spatial dependence of the neighbor pixels. It is defined as follows:

$$\begin{aligned} BF\left[I_p\right] &= \frac{1}{W_p}\sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot G_r\left(I_p - I_q\right)\cdot I_q, \\ W_p &= \sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot G_r\left(I_p - I_q\right) \end{aligned} \tag{4.1}$$

where I_p is the intensity of pixel p in input image I, Ω is a spatial neighborhood around p, $BF[I_p]$ is the filtered value of pixel p, and W_p is a normalization factor ensuring pixel weights sum to 1.0, $G_s(\|p-q\|)$ is a spatial Gaussian weighting with standard deviation σ_s that decreases the influence of distant pixels, $G_r(I_p-I_q)$ is a range Gaussian with standard deviation σ_r that decreases the influence of pixels q when their intensity values differ from I_p. In other words, Eq. (4.1) is a normalized bilateral weighted average over the filter support Ω. In case of ToF data, the depth information of the pixel is taken into account instead of gray level intensities. The filtering procedure remains the same as that for intensity images.

There are still ways to apply a bilateral filter by incorporating other information provided by ToF sensor, such as intensity or amplitude data. So called joint- or cross-bilateral filters [60] do not use the primary data to determine the range term weight but calculate it from an additional image. So, instead of evaluating both weighting components of the filter on ToF depth image, intensity or amplitude image is used in the range term. Another alternative, which is applied here, is to use the amplitude image in order to introduce adaptivity in the range term of bilateral filter [62]. Regarding ToF depth data, the noise level varies strongly over the image, depending on the amplitude of the recorded signal [42], [43]. Following the noise model presented in [42], the error variance in distance measurements σ_D^2 is proportional to the square inverse of the amplitude of the recorded IR signal – A:

$$\sigma_D^2 \propto \left(\frac{1}{A^2}\right). \tag{4.2}$$

The amplitude term can be included in weight calculation as follows [63]:

$$\begin{aligned} BF\left[I_p\right] &= \frac{1}{W_p}\sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot\left(G_r\left(I_p-I_q\right)+G_A\left(A_p-A_q\right)\right)\cdot I_q, \\ W_p &= \sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot\left(G_r\left(I_p-I_q\right)+G_A\left(A_p-A_q\right)\right). \end{aligned} \tag{4.3}$$

Here, either depth or intensity data can be used to calculate the range term. The described denoising approach can be efficiently implemented on a GPU. Depth denoising

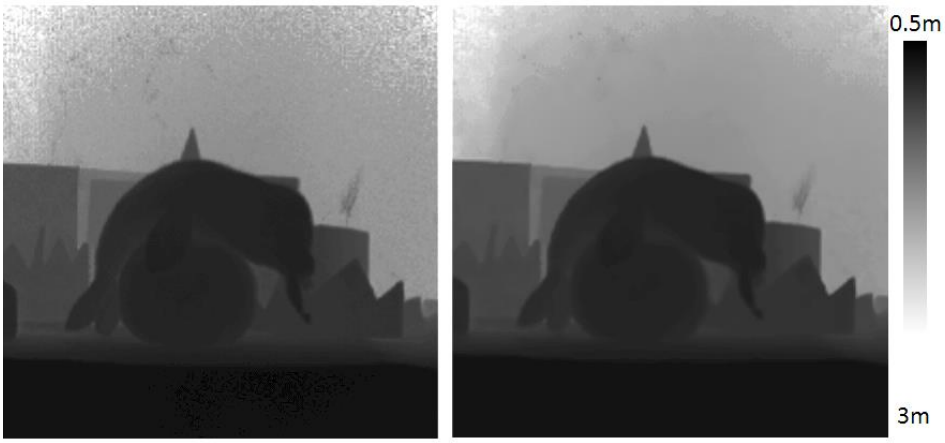

Figure 4.1 Denoising of ToF data. Left: noisy image. Right: denoised.

is combined in a single rendering pass with depth reprojection discussed in the next section. Before any manipulation with vertices in a vertex shader takes place, the denoised depth value is calculated using Eq. (4.3). All data required for this operation, i.e. depth, intensity and amplitude, is provided to the vertex shader in a form of 2D textures. The texture data in vertex shader can be accessed just as in fragment shader using vertex texture fetch [64]. Denoising results are shown in Figure 4.1.

4.2 Depth and Color Fusion

The two cameras in the setup cannot be located at the same position, so their viewpoints and viewing directions are slightly different. Combining directly the depth information with the color image will result in mismatch between the depth perception and the displayed image. Therefore, a transformation is required to align the depth information with the color image. Further, in order to obtain view-plus-depth frame the depth data should be upscaled using high-resolution color information to compensate for inaccuracies in mapping process and the low resolution of the depth data. This process is referred to as data fusion.

Projective alignment

Within the GPU rendering framework, the depth map projective alignment to the viewpoint of the color camera can be obtained by creating a 3D mesh from the data captured by the ToF sensor and observing the result from the viewpoint of the color camera. To put into practice this idea, the following steps should be taken. First of all, triangles are formed in-between the neighboring pixels of the depth image grid as shown in Figure 4.2. Next, each pixel of the depth image is converted into a corresponding 3D coordinate. It should be noted, that ToF delivers radial depth, which cannot be used directly as a vertex depth and also has to be converted into *z*-depth. So, given the focal length f_{TOF}, principal point coordinates p_x, p_y (i.e. in (0, 0)) and radial depth $d(u, v)$, the 3D vertex coordinates (x, y, z) can be computed as:

$$\begin{aligned} z &= f_{TOF} \cdot d(u,v) \Big/ \sqrt{f_{TOF}^2 + u^2 + v^2}\,, \\ x &= u \cdot z / f_{TOF}\,, y = v \cdot z / f_{TOF}\,, w = 1. \end{aligned} \tag{4.4}$$

This set of 3D points is defined in a coordinate system where the origin is the depth sensor's position. The extrinsic parameters $[\boldsymbol{R}|t]_{TOF \to 2D}$ of the camera setup, obtained during the calibration stage (see Section 3.2), are used to set up the view matrix $\boldsymbol{M}_{view}$ so

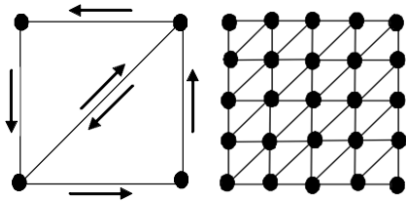

Figure 4.2 Triangulation of the image grid.

that the rendering viewpoint matches the color camera viewpoint. While the intrinsic parameters of the ToF sensor are used to calculate a proper projection matrix for the OpenGL virtual camera, $\boldsymbol{M}_{proj}$ as explained in Section 2.4.3. The mesh is finally rendered using this configuration (Figure 4.3). As all points are connected by triangles into a single surface mesh, the generated depth map contains no gaps: the disoccluded regions seen from the new viewpoint are replaced by an automatic bilinear interpolation across the triangle face connecting two different depths. However, depth data can be missing on the border of the mesh.

Rendering can be done as an off-screen rendering pass by means of Frame Buffer Object (FBO) [65]. Using FBO, a scene can be rendered directly onto a texture, so that the additional copy of intermediate results from frame buffer to texture memory can be avoided. The aim is to get the depth buffer of the rendered scene, so only the depth component of the scene needs to be rendered, i.e. rendering of the color component of the scene can be omitted, which also reduces the processing time. During the rendering process depth testing is performed automatically and only minimal per-pixel z-distance is stored in the depth buffer. However, the depth buffer does not store the actual z-value of every rendered pixel. The values stored in the depth buffer are in the range [0, 1] and are not linear due to a non-linear transformation [66]. Real depth values, i.e. the distance from the point to the camera plane, can be recovered from the depth buffer value as follows:

$$z = \frac{2 \cdot far \cdot near}{(far + near - (far - near) \cdot (2 \cdot z_{buf} - 1))} \tag{4.5}$$

where z_{buf} is the value stored in the depth buffer, *near* and *far* are parameters used to calculate the first camera's projection matrix $\boldsymbol{M}_{proj}$ (these parameters were set to be consistent with the measurement range of the PMD sensor, e.g. *near*=0.3, *far*=7.0).

Depth map refinement

After a depth map corresponding to the viewpoint of the color camera is obtained it should be upscaled to match the resolution of the color image. Modern GPUs provide

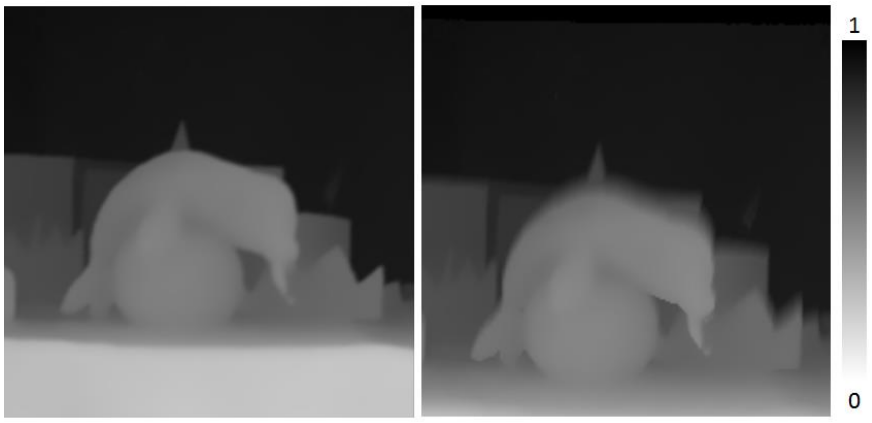

Figure 4.3 Projective alignment. Left: initial view. Right: view from the color camera position.

hardware support for bilinear interpolation: during rasterisation of a triangle (see Section 2.4.1), all attributes of the triangle's vertices, such as position, normal, texture coordinates and so on, are interpolated for each fragment across the triangle face using fast hardwired bilinear interpolation. However, special attention has to be paid to the different nature of the input data and large difference in resolution. Because of the low resolution of initial ToF data, simple linear interpolation of depth values blurs the sharp depth transitions at object borders, in addition small or thin objects might not be observed by the ToF camera. All this leads to misalignments at the objects boundaries when combining with high-resolution color and causes disturbing DIBR artifacts. Thus, the depth map obtained with bilinear interpolation needs to be further refined.

Accuracy on object borders can be increased using color controlled bilateral filter (joint- or cross bilateral filter) [60], so that color edges and depth discontinuities become aligned. The assumption is made that when neighboring pixels in the reference color image have similar color, they also have a similar depth. Joint bilateral filter is a version of the bilateral filter (see Section 4.1). Given an image I, the joint bilateral filter smooth I while aligning it to the edges of a second image E (in our case, the depth map I needs to be refined based on the color information E). In practice, the range weight is computed using E instead of I:

$$JBF\left[I_p\right] = \frac{1}{W_p}\sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot G_r\left(E_p - E_q\right)\cdot I_q,$$
$$W_p = \sum_{q\in\Omega} G_s\left(\|p-q\|\right)\cdot G_r\left(E_p - E_q\right) \tag{4.6}$$

where W_p is a normalization factor.

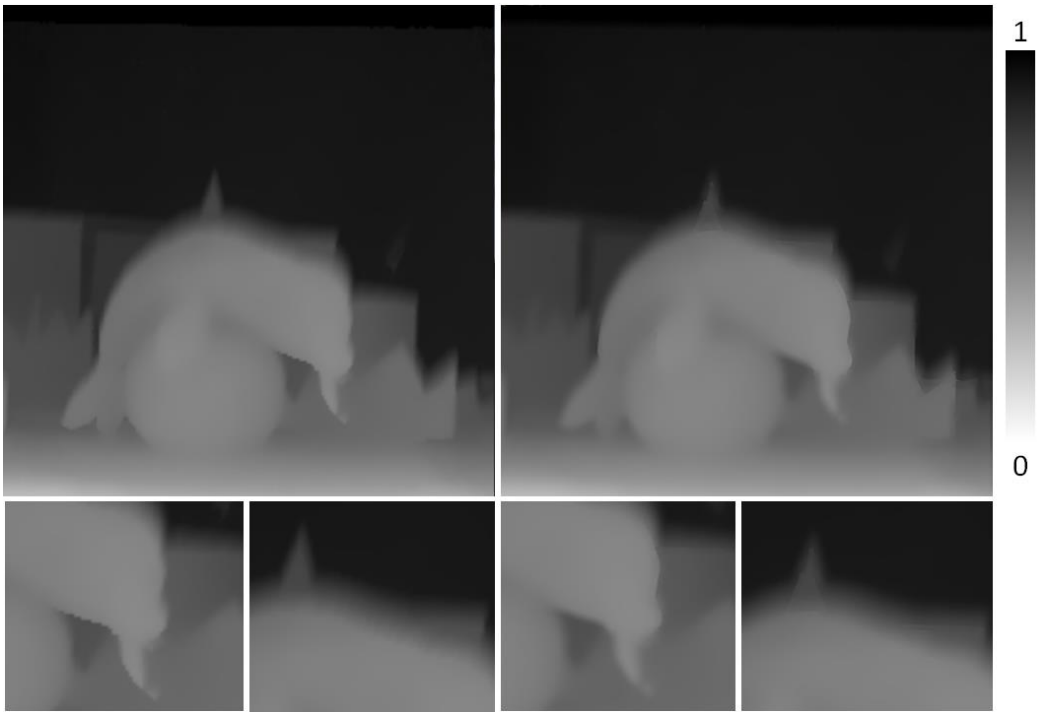

Figure 4.4 Left: Bilinear interpolation. Right: With JBF (parameters are: window size = 11; σ_s = 3; σ_r = 0.075). Bottom: zoomed parts.

Image filtering on GPU is usually done in the fragment shader, as the fragment shader is responsible for the per fragment processing. A simple way to perform upscaling and filtering of the depth data is to render a quadrant spanning the full viewport, which size is equal to the desired output resolution. In this way, there is a one-to-one correspondence between fragments of the rendered quadrant and pixels of the output buffer. Similarly to the case of depth map rendering, the output can be rendered off-screen directly to a texture using FBO (Figure 4.4).

The color image and low-resolution depth map are provided to the fragment shader as textures. OpenGL uses normalized texture coordinates; this means that the size of a texture maps to the coordinates on the range [0, 1] in each dimension. The value for any texture coordinate within the range can be sampled automatically using bilinear interpolation. Thus, Eq. (4.6) is used to calculate output color for every fragment. Samples from the low-resolution depth texture that falls in between the texels are bilinearly interpolated, and as the output resolution matches the color image size, the samples from the color texture correspond to actual pixels of the color image.

4.3 Lens Distortion Correction

In practice, lenses used in the real cameras do not act exactly like the pinhole model (Section 2.1), and create geometrical distortions not accounted by the model. The result of geometrical distortions is that straight lines in the real world appear curved in the image. This effect increases with the distance from the distortion center.

ToF sensors, like regular cameras, are modeled by the pinhole camera model. Therefore, their images are corrupted by lens distortion effects. This distortion leads to shape mismatches between the color image and the corresponding depth image. Thus, in order to achieve more precise data fusion results, both color and depth images should be undistorted before the fusion procedure.

In the light of the real-time constraint, undistortion of the high-resolution color image is an undesirable time-consuming operation as it leads to remapping and interpolating of each pixel in the image grid. However, optical distortions of the input data can be handled by processing only the values of the depth image in two steps (see Figure 4.5), so that undistortion procedure of the color image can be omitted completely.

First, to cope with distortions of the depth image, instead of initial grid coordinates

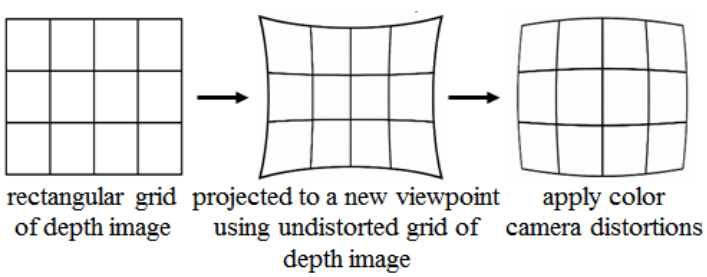

Figure 4.5 Optical distortion correction scheme.

(u, v) the undistorted coordinates of the depth image grid (u_u,v_u) should be used in Eq. (4.4) to calculate 3D point coordinates of each pixel, so that after reprojection to the new view point depth image containing no distortions is obtained. The distortion parameters are constant parameters of the sensor, so the undistorted image grid of the depth image can be calculated only once during the initialization stage before the streaming is started or even pre-calculated and stored along with calibration parameters.

Second, when joint bilateral filter is applied, to take into account distortions caused by the color camera optics, the sampling coordinates of the depth texture should be adjusted (distorted) using distortion coefficients of the color camera, so that the correspondence with the distorted color image is maintained.

Optical distortions of the cameras are modeled as radial and tangential effects [17]:

$$p_u = \left(1 + d_1 \cdot r^2 + d_2 \cdot r^4 + d_5 \cdot r^6\right) \cdot p_d + dx \tag{4.7}$$

where $r^2 = p_x^2 + p_y^2$ is the distance to the distortion center; dx define the tangential distortion, which can be omitted in our case; d_1, d_2, d_3 are distortion coefficients; p_d is the distorted point, p_u is the corresponding undistorted coordinate. This function is used for remapping process. The coordinates of distortion centers and the distortion coefficients for both cameras are estimated during the calibration procedure (Section 3.2).

4.4 Rendering

As stated before, the application supports two rendering modes: view-plus-depth frame for the autostereoscopic display and 3D model of the scene enabling the free-viewpoint functionality. Rendering is done in a separate rendering pass after the dense depth map corresponding to the color image has been calculated and saved to a texture.

Having two corresponding color and depth textures, the final view-plus-depth frame can be rendered as two side-by-side textured quadrants (Figure 4.6) spanning the whole viewport: the color texture is applied to the left quadrant, and depth texture to the right quadrant. As stated in Section 3.4, depth sub-image of the Dimenco display frame

Figure 4.6 View-plus-Depth frame.

should contain disparity values with a range of 0 to 255, so depth values should be normalized to [0..1] and converted into disparities using Eq. (3.1). However, as mentioned in Section 4.2, depth texture contains depth-buffer specific values, which are not actual depth. Before calculating disparities, the actual depth values should be recovered using Eq. (4.5). Depth texture values can be converted to disparities in the fragment shader.

The frame header includes user provided parameters to adjust depth impression. It is calculated on CPU side and passed as a uniform variable to the fragment shader, where it is copied to the upper left of the output buffer. In order for the display to identify the header, the header should be located at the beginning of the first pixel row of the display. To locate the header properly, a frame is rendered in a full screen borderless OpenGL window with its upper left corner coincide with the upper left pixel of the Dimenco display. The initial size of the textures is 800×800, when resizing the window to full screen mode (i.e. to full display resolution 1920×1080) automatic bilinear interpolation is applied.

In case of arbitrary view rendering, during the third rendering pass instead of a view-plus-depth frame a textured surface mesh representing the scene is rendered. The calculated dense depth map is turned into a triangulated surface in the same manner as it was done before for the ToF depth image but using the color camera intrinsic parameters. As there is a pixel-to-pixel correspondence between depth and color textures, every 3D vertex can be assign a proper texture coordinate simply based on its position in the image grid. The texture is then cast onto every triangle in the mesh using the camera position as the center of projection. This is called projective texture mapping. Projective texture mapping was first introduced in [67] and now it is part of the OpenGL standard.

Whenever a new view of the scene has to be rendered, the coordinate system transformation of the new view is used as a view-matrix transformation (see Section 2.4.2) to place the scene surface where needed in the world coordinate system. The camera view point can be modified by changing the view matrix, so that it is possible to rotate or translate the virtual camera in any direction, to go into the 3D scene and looking around (Figure 4.7).

Figure 4.7 Textured surface mesh views from a virtual camera.

4.5 Disocclusion Detection

One of the most important issues of DIBR is dealing with disoccluded areas in newly generated views. In case of mesh-based view synthesis, the disocclusion areas are filled automatically by linear surface interpolation. However, this method causes geometric distortions. The triangulation technique explained above treats the entire depth map as a continuous surface that contains stretched triangles connecting data on background with boundaries of front situated objects (Figure 4.7). The artificially-elongated triangles at object boundaries (also called rubber-sheet triangles) appear large and visually unrealistic, hiding objects and background behind them. Thus, it is important to determine whether triangles represent actual object surfaces or not.

To detect artificial triangles of the mesh, the orthogonality test proposed in [68] is employed. This test is based on the observation that the artificial triangles, introduced during the triangulation, have the following property: a triangle normal is almost perpendicular to the vector from the initial viewpoint to the center of the triangle, i.e. dot product of the vectors is close to zero (Figure 4.8 bottom-left). Problematic triangles can be discarded from the mesh during the surface rendering pass inside a geometry shader with the following inequality using a threshold to control the amount of discarded triangles:

$$n_t \bullet p < \tau \tag{4.8}$$

where t is a triangle index, n_t is the normal of t, p is the direction from camera position to the center of the triangle, symbol $\bullet$ denotes vector dot product operation, and τ is a tunable threshold parameter, the bigger the threshold τ the more triangles are discarded. However, discarding triangles by the above criterion introduce new edges in the rendered view, which may look unacceptably jagged due to complete discarding of triangles at the boundaries of disocclusion areas and may negatively affect the disocclusion filling results (Figure 4.8 bottom-right). This problem will be further addressed in Chapter 5.

After artificial triangles are removed, a virtual view containing holes is obtained (Figure 4.8 top-right). The image is rendered directly to a texture and further processed during the next rendering pass where the disocclusions should be filled. The pixels belonging to disoccluded areas can be easily detected by masking them in alpha channel of the RGBA texture. At the beginning of the rendering process the frame buffer is initialized with the color (0, 0, 0, 0), so that all unpainted pixels have alpha value equals to 0. When a fragment is assigned a color value, the alpha value is set to 1, and for pixels belonging to holes alpha value will remain 0.

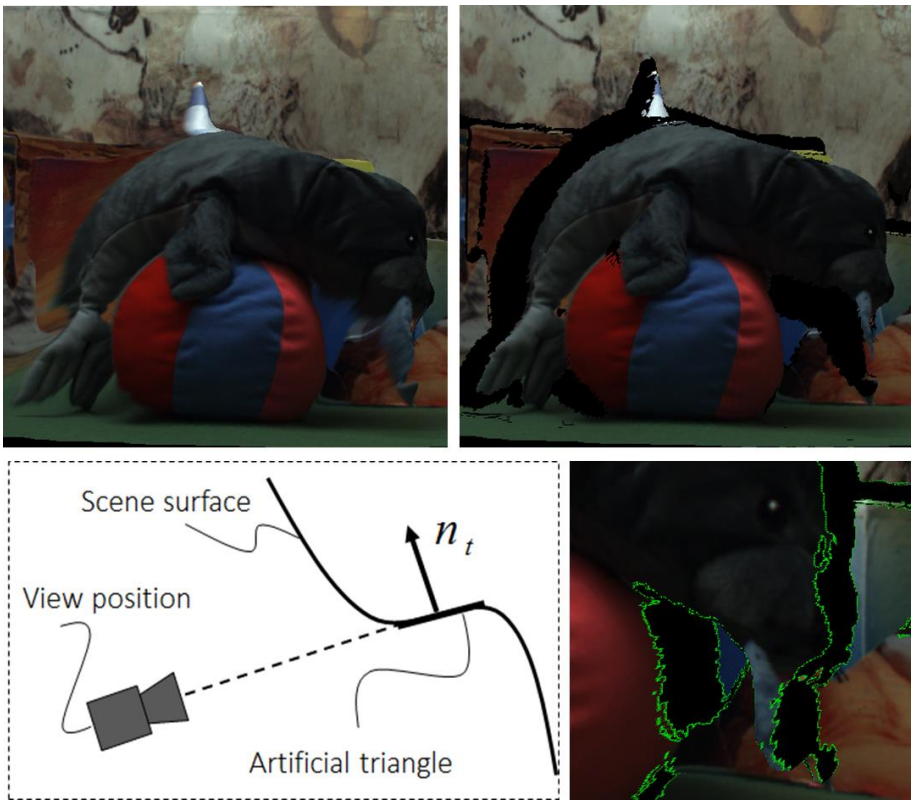

Figure 4.8 Omitting artificial triangles in 3D scene surface. Bottom-Left: principle. Top-Left: no triangles are discarded. Top-Right: artificial triangles are removed. Bottom-Right: jugged edges marked green.

During the disocclusion filling step appropriate color for the missing pixels in the final picture should be determined by analyzing their surrounding areas. Although a number of inpainting techniques have been proposed in the literature (see Section 2.2.2), there are not many algorithms, which are designed to run on GPU. Usually iterative algorithms based on sequential prioritized filling are used for inpainting. Namely, there is an increasing interest in exemplar-based methods due to their ability to provide high-accuracy results.

The basic idea of exemplar-based methods is to assign a priority to each patch on the border of unfilled region and to search in the prioritized order for the best patch in source region of the image that can be used to effectively fill the missing pixels. Further depth information can be used in order to reduce search range to only background pixels of the synthesized image. Such methods produce good visual results for the price of high computational complexity and can hardly be parallelized for GPU-based implementation. Several papers address the efficiency issues of exemplar-based inpainting [69], [70], and propose partially parallel implementations to accelerate the sequential inpainting process while maintaining its high accuracy. Although a noticeable speedup can be achieved, still the methods are not really suitable for real-time applications.

When real-time performance of a system is a definite requirement and GPU-based implementation is desired, intensity interpolation or intensity spreading techniques more suitable for parallelization can be applied. In [71], intensity spreading algorithm is proposed: the set of border pixels of all holes is determined and for each border pixel, its intensity is propagated into the hole in a fixed set of directions (typically 16, equally

Figure 4.9 Result of disocclusion filling by background propagation.

distributed over the 360 degree range). Such approach yields some blurring in the filled region as both foreground and background pixel intensities are accounted. In our case, disocclusions regions are not foreground objects, but newly discovered background areas. Following this assumption, depth information at the holes edges can be used for filling with more accurate pixel values. As suggested in [72], for every pixel in a disoccluded region the nearest edge pixels are searched in eight directions and only the edge pixels with the furthest depth value, i.e. corresponding to the background, are taken into account when a weighted average is calculated to obtain pixel's value.

Although disocclusion filling is considered to be out of scope of the current work, for the sake of processing chain completeness, a simple background color interpolation method was implemented to fill in disocclusions. For every unfilled pixel of a newly synthesized view, two nearest filled pixels in horizontal direction are searched, and the unfilled pixel gets a color value of the pixel with the furthest depth value, i.e. the one on background. The result can be seen in Figure 4.9.

4.6 Performance Evaluation

The proposed processing chain is tested with a color and a ToF cameras and a real-time application is developed. This section describes the experimental evaluation carried out to validate the algorithmic solutions presented throughout this chapter. The codes are written based on C++ and GLSL using cross platform libraries, namely OpenCV and OpenGL. The experiments are carried out on NVIDIA GeForce GT640 GPU and 3.4 GHz Intel Core i7-3770 hosted by64-bit Windows7.

Considering the execution speed of the system, for both view-plus-depth frame rendering and arbitrary view rendering scenarios the rendering time is only limited by the frame-rate of the cameras. To estimate actual frame rates, the measurements were done

using off-line data recorded from the camera setup. In Table 4.2 the average execution time and frame rate over 100 frames is presented. The view-plus-depth performance for 1080×1920 video is well above 50 fps. For arbitrary view output, performance of 36 fps frame rate is achieved.

Table 4.2 Average execution timing and frame rate

	View-plus-Depth 1080×1920 pixels	**Arbitrary View** 800×800 pixels
Denoising	1.3 ms	1.3 ms
Projection alignment	4.4 ms	4.4 ms
Depth/color fusion (JBF)	5.7 ms	5.7 ms
Disocclusion detection	-	0.4 ms
Disocclusion filling	-	6.6 ms
Rendering	5.5 ms	7.5 ms
Total	16.9 ms	25.9 ms
Overhead	0.96 ms	1.9 ms
FPS	56	36

In addition, the approximate estimation of computation time of each individual processing step on average is calculated (Table 4.2). Although the processing chain is implemented as a sequence of several rendering passes, it is not straightforward to measure precisely what is going on within the OpenGL rendering pipeline. For example, when an OpenGL call is returned, it does not mean that rendering is finished: typically data sent to graphics card is placed into the first-in-first-out (FIFO) queue at the front end of the graphics card and is processed and rendered sometime later, while the call is returned to the application right away. Thus, simply starting the clock before and stopping them after an OpenGL call can tell very little about performance. As described in [73], a solution is this case is to use *'glFinish'* command before the clock is started as well as just before the clock is stopped. First, it forces to finish all the tasks waiting in the FIFO, so that the graphics card is not busy with something else when the clock is started. Second, it forces the rendering to be completed before it returns. This can give an accurate idea of how long each rendering pass takes. In case when there are multiple tasks are performed in a single rendering pass, e.g. denoising and projection alignment, two timings can be measured: with denoising part and without (also there is no need to transfer intensity and amplitude data when denoising is switched off). Then, denoising time can be calculated as difference between these two timings. Finally, the *overhead* in this case is calculated as difference between the execution time of the full processing chain and the sum of measured timings of each processing step.

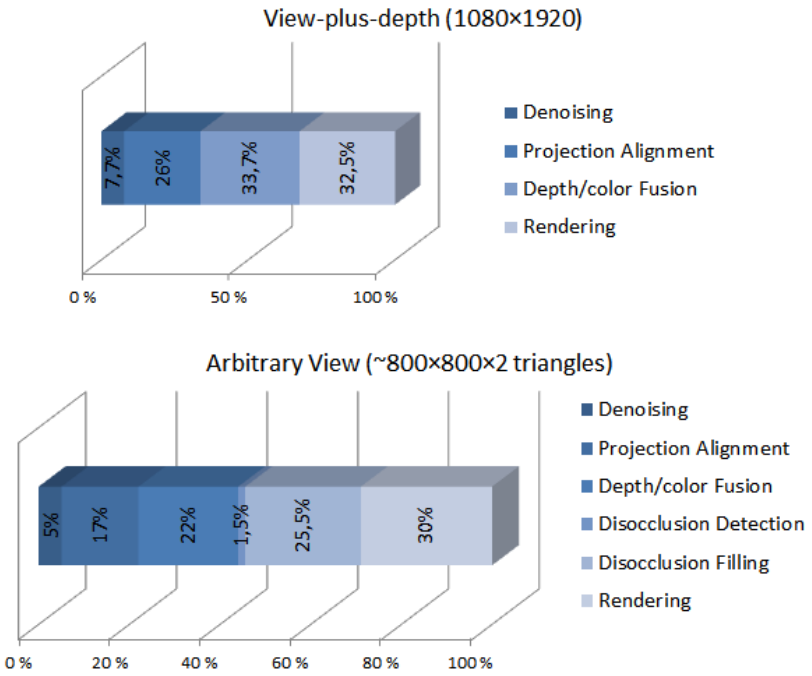

Figure 4.10 Workload distribution.

Figure 4.10 shows the workload distribution of all processing steps in percentage. It can be observed from the figure that, apart from the rendering stage itself, joint bilateral filtering (JBF) and disocclusion filling process are the most computationally expensive modules in the system chain. For the rendering part, the timings are mostly depending on the resolution of the output data. For the high resolution 1080×1920 view-plus-depth frame, the rendering is done in the simplest possible way and very little can be done to improve the performance. In case of arbitrary view rendering, to reduce the number of triangles a more sophisticated approach for triangulation of the depth map can be considered, which however would increase the algorithmic complexity, but nevertheless can improve processing time.

Time consumption of a JBF straightforward implementation depends on the filter window size as $O(n^2)$, where n is the window size. Dependency of the overall view-plus-depth rendering frame rate on the window size of JBF is depicted in Figure 4.11. However, bigger size of the filter window does not give better quality results in general. For the bigger window size, influence of the color term of the filter becomes more pronounced and some texture artifacts appear in the homogenous regions of the depth image, while for a very small window size, the color term does not give enough visible improvement of the depth map, and the filtering result looks like simple smoothing effect. Results presented in Table 4.2 have been obtained with the JBF window size set to 11, which was chosen as an optimal window size providing visible improvement of the depth map at the depth discontinuities, yet keeping the homogenous depth regions smooth.

Disocclusion filling process can create a substantial performance bottleneck: even using a very simple background propagation algorithm it takes up more than a quarter of the total computation time. This is due to the step-by-step neighborhood search, which is difficult to parallelize and computationally depends on the width of the hole to be

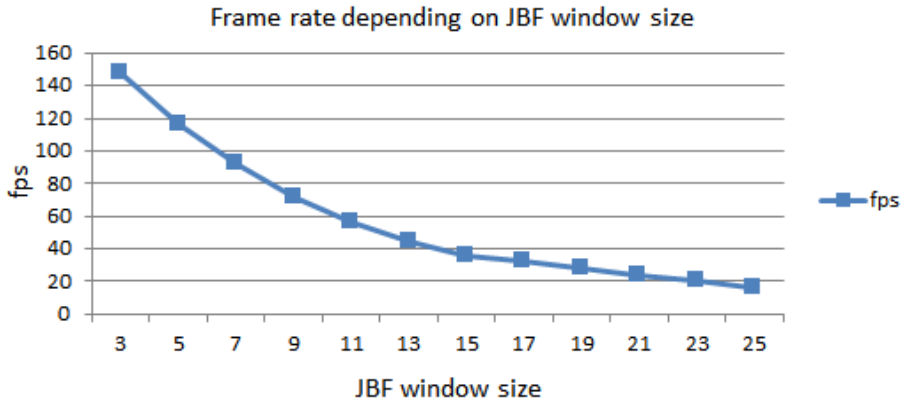

Figure 4.11 View-plus-depth rendering frame rate depending on JBF window size.

filled. However, as was pointed out before, this simple approach for disocclusion filling would not yield acceptable visual results for big disocclusion holes and, thus, have to be substituted with more complex algorithms to achieve better visual quality, which will increase processing time. One possible approach to address this problem is to utilize the information from the two conventional side-cameras presented in the camera setup: currently the system uses the color information only from the central camera in the setup. The two side views of the scene can provide true texture information for the disoccluded regions; while the still-left smaller holes in turn can be filled by simpler algorithms, such as background propagation, without much degrading the visual quality of the rendered virtual view. However, this additional information does not come for free: first of all, the data traffic between GPU and system memory will increase almost three times and, further, additional computational efforts will be associated with the fusion and processing the data from multiple texture sources.

4.7 Summary

In this chapter, a GPU-based implementation of the processing chain of algorithms necessary for noisy low-resolution ToF depth images fusion with the high-resolution color data was demonstrated. The resulting system can render a view-plus-depth frame or an arbitrary view of a scene in real-time. This shows that real-time free-viewpoint DIBR is feasible; the demonstrated system can be used as a base for a future 3DTV system. As an alternative approach, it would be interesting to investigate the possible use of additional color views, which can be provided by the side-cameras in the setup, for more reliable depth estimation and disocclusions handling. Finding the best balance between performance and visual quality for such setup is an important part of the future work.

5. ARBITRARY VIEW SYNTHESIS BASED ON DEPTH LAYERING

In DIBR context, a 3D warping technique (Section 2.2.1) is used for rendering virtual views from view-plus-depth representation. The 3D warping projections result in data points containing known values from the original view scattered on the virtual camera image plane. This means that the unknown values at the regular pixel positions of the virtual view need to be estimated from irregular data, imposing a non-uniform to uniform resampling. Performing an accurate non-uniform to uniform resampling in real time is a challenging task and still an open-research problem [74].

On the other hand, in GPU-based rendering context arbitrary view synthesis can be implemented in real time, as was demonstrated throughout this work. When a mesh-based representation is used, a textured surface of a scene can be rendered form an arbitrary point of view. However, virtual views generated with this approach contain rubber sheet artefacts (artificial surfaces that were not presented in the original 3D scene, but were introduced during the triangulation process). Artificial surfaces look unrealistic and degrade the quality of generated virtual views, therefore they should be detected and removed. As proposed in Section 4.5, problematic surfaces can be detected and discarded by employing orthogonality test. Nevertheless, the problem of jugged edges still exists.

In this chapter, a new approach for DIBR based on depth layering is proposed. The propose method is a computationally efficient workaround of the "3D warp → non-uniform resampling" scheme, which is usually employed by traditional DIBR approaches for virtual view synthesis from view-plus-depth data. The proposed method avoids the non-uniform resampling stage by employing depth layering, which facilitates a resampling at the uniform grid of the given reference camera. Another limitation of the classical approach is that non-uniform data cannot provide accurate estimation and masking of disoccluded and hidden data, i.e. disocclusions detection and z-ordering, while the depth layering approach naturally embeds z-ordering and disocclusion detection within the view-generation process.

The experimental results demonstrate its real-time capability even for CPU-based implementations, while the quality is comparable with other view synthesis approaches but for lower computational cost. It is also suitable for general free-viewpoint rendering scenario in terms of position, orientation, focal length, and varying sensor spatial resolutions of reference and virtual cameras. The main benefits of the proposed method can be summarized as follows: a high-quality texture resampling, avoiding non-uniform to uniform resampling, on-the-fly disocclusion detection and z-ordering.

The chapter is organized as follows. The depth layering procedure is described in Section 5.1 and the view synthesis algorithm is in Section 5.2. Some optimization considerations are presented in Section 5.3. Finally, Section 5.4 provides some experimental results.

5.1 Layering in Disparity Domain

The input is formed by the aligned color and range images (Figure 5.1 a and b), where the range image represents the depth map $z(x,y)$. For the layer based rendering the given depth map is first transformed into a disparity map, which is subsequently divided into layers. Here *disparity* means the amount of displacement between projections of a point in two images from different viewpoints. The disparity map is calculated with respect to the targeted virtual view, which is determined by the new camera position and the corresponding baseline b between the reference and the new camera positions. The disparity displacement is calculated in pixels as follows:

$$d(x,y) = \frac{b \cdot f}{z(x,y)} \tag{5.1}$$

where f is the focal length of the reference camera. The required number of layers depends on the minimum and maximum disparity displacements D_{min} and D_{max}. A layered map with L layers, l=1…L, is constructed by assigning pixels with disparity displacements within the interval [D_{min}, D_{min}+1) to the first layer, within the interval [D_{min}+1, D_{min}+2) to the second layer, and so on until $D_{min}+L \geq D_{max}$ and the interval [$D_{min}+L$-1, D_{max}) corresponds to the last layer. Figure 5.1c illustrates a layered disparity map.

Such layering of the depth map of a scene allows to approximate depth volume of the scene by a set of planes, with each layer corresponds to a plane in space parallel to the reference camera image and located at depth z_l (world coordinate system origin is considered placed at the reference camera position), i.e. the plane equation is $Z=z_l$, for l=1..L. The depth value z_l can be chosen as the average depth value between minimum and maximum depth values of the pixels belonging to the layer. This idea is illustrated in Figure 5.2.

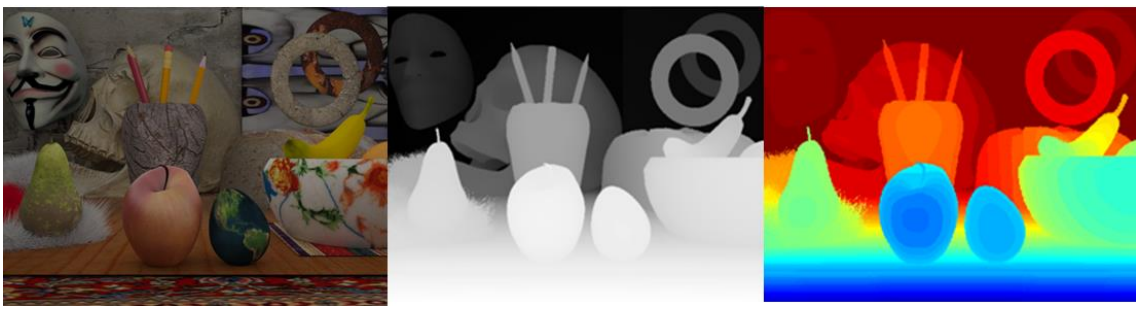

Figure 5.1 Test scene: a) color, b) depth map, c) layered map.

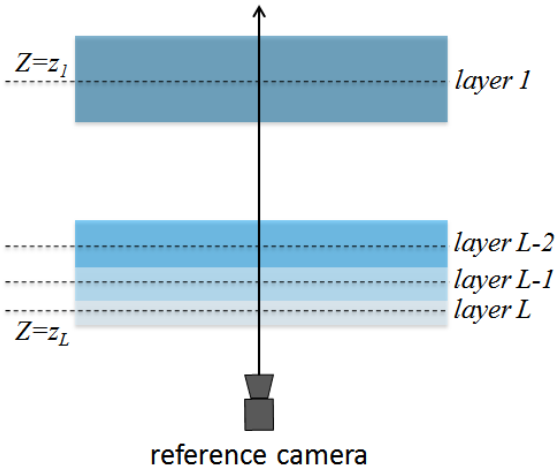

Figure 5.2 Representation of a scene by a set of parallel planes.

Such flat layered representation of the scene geometry allows approximating the 3D transformations as a series of 2D planar perspective projections. This feature is used to render an arbitrary view as described in the next section.

5.2 View Synthesis

Having the scene depth volume approximated by a set of planes, a projective transformation, i.e. a homography, exists for each plane, which can be used to map a projection of a point, which lies on the plane from the reference camera image plane to the virtual camera image plane (Figure 5.4). Thus, the 3D warping procedure can be approximated as a series of 2D perspective projections performed in a reversed way.

Consider the pinhole model camera model (Section 2.1). The projection matrices for the reference and the virtual cameras are defined as $\boldsymbol{P}_{[3\times4]}=\boldsymbol{K}\cdot[\boldsymbol{I}|0]$ (world origin is at the camera) and $\boldsymbol{P}'_{[3\times4]}=\boldsymbol{K}'\cdot[\boldsymbol{R}|t]$ correspondingly. Here $\boldsymbol{I}_{[3\times3]}$ is an identity matrix, $\boldsymbol{K}_{[3\times3]}$ and $\boldsymbol{K}'_{[3\times3]}$ are the reference and the virtual cameras' calibration matrices of the form:

$$K_{3x3}=\begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

where f_x, f_y are the focal length components measured in pixels, p_x, p_y are the principal point coordinates; and $[\boldsymbol{R}|t]$ is the combined rotation matrix $\boldsymbol{R}_{[3\times3]}$ and the translation column vector $t_{[3\times1]}$ of the virtual camera with respect to the reference camera. Then a 3×3 homography transformation $\boldsymbol{H}_l$ which maps points of the plane $Z=z_l$ from reference image plane to the virtual image plane, is:

$$\boldsymbol{H}_l=\boldsymbol{K}'\cdot\left(\boldsymbol{R}+t\cdot[0\,0\,1/z_l]\right)\cdot\boldsymbol{K}^{-1}. \tag{5.3}$$

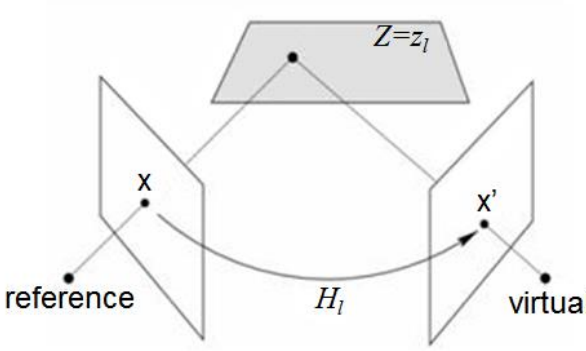

Figure 5.4 Homography induced by a plane.

To render a view from the virtual camera position, for every plane in the set approximating the scene projections of the virtual camera image grid onto the reference camera grid are calculated, so that the values of the projected points can be interpolated within the regular grid of the reference image. To calculate the projections, for each depth z_l, points of the virtual view image grid are back-projected onto the reference image plane using the inverse homography $\boldsymbol{H}_l^{-1}$. In this way, projections of the virtual image grid points are obtained as if they all were located on the plane $Z=z_l$ in space, or, according to our layering, if they all belong to the layer l of the reference image. In order to decide which of the projected points do actually belong to the layer l and how their values can be interpolated, the neighborhood of the projected points is analyzed (Figure 5.3). First, the four-point neighborhood of the projected point is considered, which is referred to as *layer support*. If the layer support of the projected point (u,v) contains pixels from current, next or previous layers and there is at least one pixel from the current layer, the value P of the projected point (u,v) can be interpolated. Second, the $n\times n$-point neighborhood is checked where n is determined by the chosen interpolation support. If $P(u,v)$ can be interpolated within the current layer, use for the interpolation only those pixels from the $n\times n$ resampling support which belong to the layers $l\text{-}n/2..l+n/2$ qualified as valid for interpolation. If all pixels within the resampling support are valid, the desired interpolator is directly used. If there are some invalid pixels, an interpolator with a smaller support is applied.

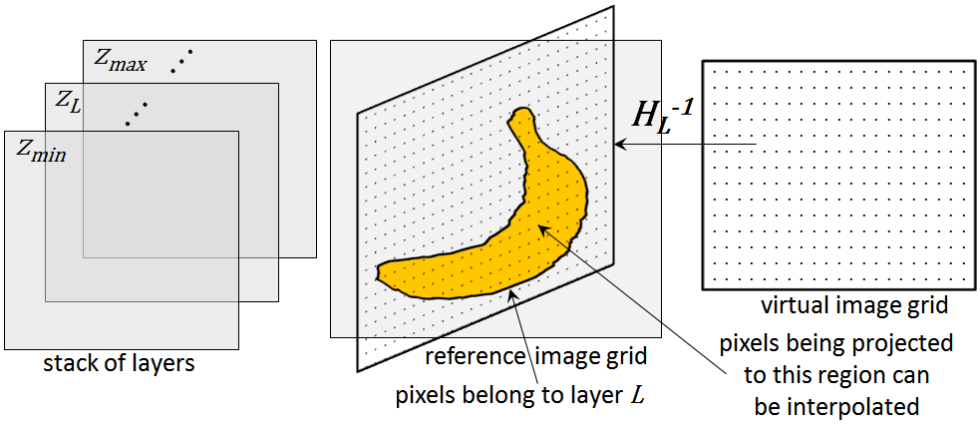

Figure 5.3 Layered rendering.

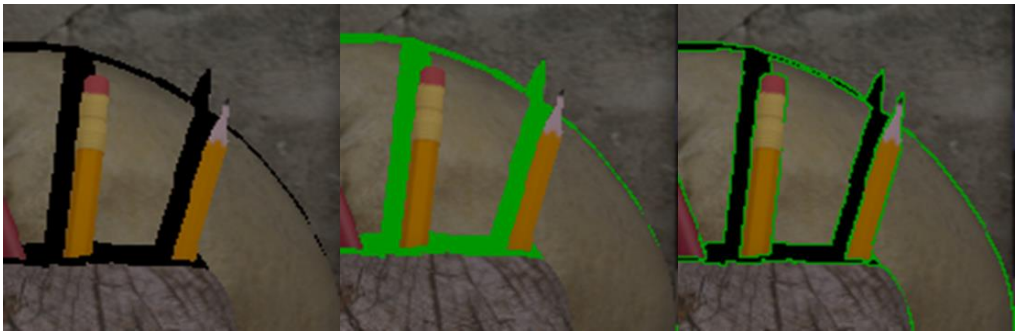

Figure 5.5 Layered rendering: a) result for translation vector (0.1, 0.1, 0.1) in meters, b) disoccluded regions, c) edge detection.

The obtained interpolated values are used to fill in the corresponding values on the virtual view image grid. In this way, by processing all layers, a novel view can be rendered as illustrated in Figure 5.5a. By processing layers from back to front *z*-ordering is maintained. Disoccluded regions are determined by pixels where no value was assigned, see Figure 5.5b. Edge pixels of disoccluded regions can also be detected as pixels without values that were projected in such a way, that they have pixels from current layer in their layer support, but some of the pixels in the layer support do not belong to the current, next or previous layers (Figure 5.5c). The projection incorporates all parameters of a two-camera system; therefore the rendering script can be configured for different camera arrangements, asymmetric camera parameters in terms of f, $\boldsymbol{R}$, t, and varying sensor spatial resolutions, which works also as rescaling or zooming of the rendered view.

The described method can be regarded as plane sweeping [75], [76] but implemented in a reverse manner. The plane sweeping approach uses layers to estimate depth, while the proposed method uses depth to find optimal depth layering. Furthermore, by the inverse projection from the virtual to the reference view, pixels being synthesized within the reference camera grid, thus avoiding non-uniform resampling.

5.3 Back Projection Optimization

The virtual image grid is back-projected for each value of z_l. Instead of applying the homography $\boldsymbol{H}_l^{-1}$ every time, it is more efficient to compute the projection of the virtual image grid on some initial plane, e.g. Z=1, and then modify this projection to calculate the projection of the virtual image grid on any other plane Z=z_i [76]. Consider two projections $\boldsymbol{H}_1$ and $\boldsymbol{H}_i$ that map points from the planes Z=1 and Z=z_i correspondingly to the virtual image grid:

$$\begin{aligned} \boldsymbol{H}_1 &= \boldsymbol{K}' \cdot \left(\boldsymbol{R} + t \cdot [0\,0\,1] \right) \\ \boldsymbol{H}_i &= \boldsymbol{K}' \cdot \left(\boldsymbol{R} + t \cdot [0\,0\,1/z_i] \right) \end{aligned}. \tag{5.4}$$

The homography $\boldsymbol{H}_{i1} = \boldsymbol{H}_i^{-1} \cdot \boldsymbol{H}_1$ maps points between planes Z=1 and Z=z_i directly, by first applying forward projection form Z=1 to the virtual image grid, and then back-projecting them onto the Z=z_i. $\boldsymbol{H}_{i1}$ has a rather simple structure: if (u_1, v_1, w_1) is a point projected onto the plane Z=1, the corresponding point on the plane Z=z_i is:

$$\begin{pmatrix} u_i \\ v_i \\ w_i \end{pmatrix} = H_i^{-1} H_0 \begin{pmatrix} u_1 \\ v_1 \\ w_1 \end{pmatrix} = \begin{pmatrix} (1+c_3\delta)u_1 + (1-\delta)c_1 w_1 \\ (1+c_3\delta)v_1 + (1-\delta)c_2 w_1 \\ (1+c_3)w_1 \end{pmatrix}. \quad (5.5)$$

Here $\delta=1/z$ and $(c_1, c_2, c_3) = (r_1t, r_2t, r_3t)$, where t is the translation column vector, and $\boldsymbol{r_i}$ is a row vector of transposed rotation matrix $\boldsymbol{R}^T$. This way all projections can be calculated as linear combination of some pre-calculated projection of the virtual image grid.

5.4 Experimental Results

The proposed resampling technique is demonstrated by an experiment performed on a photorealistic scene synthetically rendered by the Blender software [77]. A rendering script has been configured for different camera arrangements, asymmetric camera parameters in terms of f, $\boldsymbol{R}$, t, and varying sensor spatial resolutions. The designed scene is of high depth contrast varying within the range of 0.5-7 m, which resembles a typical range of the available ToF depth sensing devices [3]. The scene contains objects of different reflection surfaces, materials and facing directions, and illumination noise. The scene is given in Figure 2a and 2b.

Using the script, a rather extreme case has been simulated. A horizontally aligned camera setup has been misaligned in forward/backward direction within the range 0-0.25m for an arbitrary chosen relative pose in terms of $\boldsymbol{R}$ and t, and for different resolution downscaling of the reference view, both for width and height from two to ten times in order to evaluate upsampling capabilities of the method.

We compared the rendered results of the proposed approach against previous approaches by measuring the difference to ground-truth data in terms of peak signal-to-noise ratio (PSNR) for the rendered color view. The comparative tests included a direct bilinear plane-fit resampling (bilinear triangulation) as used in a graphic accelerator hardware supported by OpenGL, and a method proposed in [46], where an intermediate resampling step by bilinear triangulation is used to place the virtual camera to the desired camera position, and a 2D image back-projective up-sampling to the desired camera grid is followed (denoted as VC method).

Ground truth data (GT) was rendered in Blender with the same tested baseline, relative pose and misalignment, and with disoccluded pixels being masked. All re-sampling approaches were programmed in Matlab. For the 2D interpolation (VC) and the proposed method (Proposed), bi-cubic interpolation was implemented [78]. The results plotted in Figure 5.6, show that for each tested case, our approach achieves comparable or better quality. Visual results for some scene details are given in Figure 5.7. From the experimental results it can be concluded, that the method is limited to relatively small free-view shifts (e.g. <0.25m). For larger shifts, the number of layers increases and more pixels are treated as boundary conditions, which degrades the performance.

To measure the performance of the proposed method, it was implemented in C++ and tested on a computer with Intel Core i7-3770 CPU. The performance results are

measured for the 480×640 resolution of reference and virtual image grids and can be seen in Figure 5.8. The results demonstrate that pure CPU-based C++ implementation of the proposed approach without any processor-specific optimization gives a fair real-time performance (~20 fps), the approach has superior quality and in contrast to the other approach it also inherits disocclusion detection.

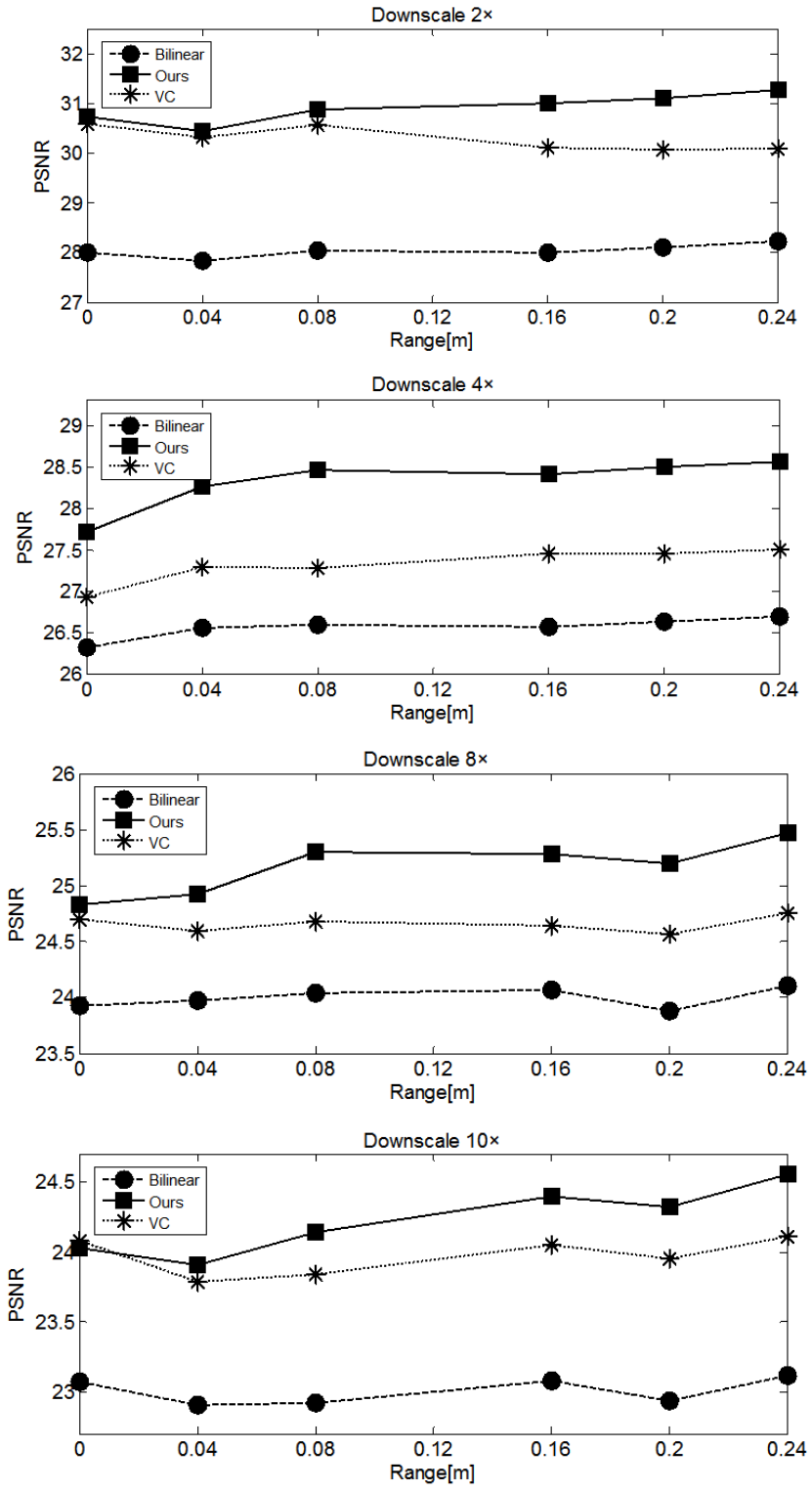

Figure 5.6 Performance comparison for asymmetric camera setup between proposed (Ours), general re-sampling approach (Bilinear) and virtual camera inter-view rendering approach (VC).

Figure 5.7 Visual results on scene details for 4x asymmetric downscale and 0.25m setup misalignment of methods (by columns): a) Bilinear, b) VC, c) proposed, d) GT

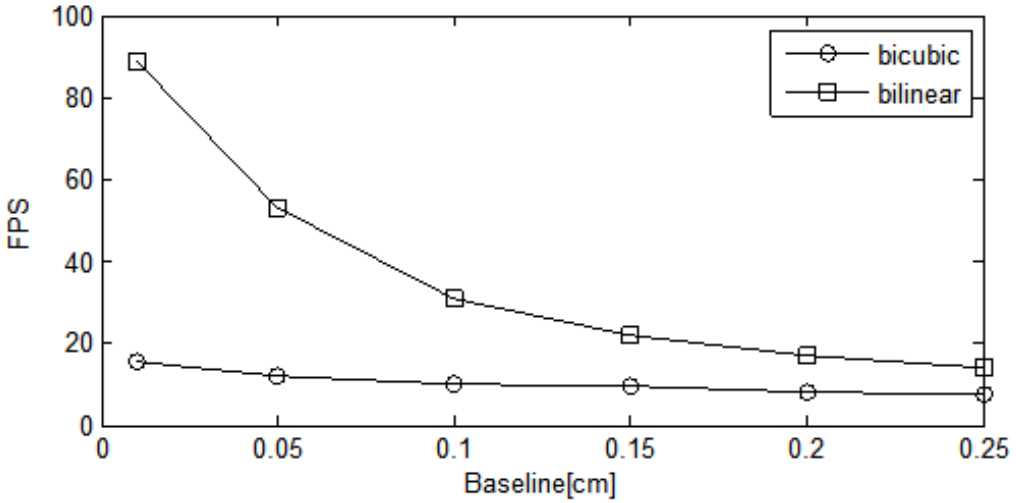

Figure 5.8 Performance results of the method for bilinear and bicubic interpolation kernel.

5.5 Summary

In this chapter, a layered resampling approach for free-viewpoint rendering from view-plus-depth data is proposed. While achieving high quality texture resampling results, avoiding non-uniform to uniform resampling and providing on-the-fly disocclusion detection and z-ordering, the speed of the proposed method has similar processing time compared to some other 2D image resampling methods, and is attractive for CPU-based applications. However, the method is limited to relatively small free-view shifts (e.g. <0.25m). In case for larger shifts, more pixels are treated as boundary conditions as the number of layers increases degrading the performance. Future work will focus upon investigating the possibilities of an optimal depth layering and a speed optimized implementation utilizing general-purpose computing means of modern GPUs.

6. CONCLUSION

In this work, a prototype 3DTV system capable for rendering live video on a 3D autostereoscopic display has been presented. The entire processing chain from a scene capture to its rendering on a 3D display has been implemented and composed into a unified end-to-end framework. Each component in the framework has been presented in details throughout the thesis. The depth acquisition employs a ToF sensor, which can provide depth information in real-time. Since the depth camera is not able to provide color information, a conventional color camera is added in order to capture a scene textures. The input data from the camera setup is being processed in a series of steps in order to produce a view-plus-depth frame for an autostereoscopic 3D display or to generate a novel view of the captured scene from an arbitrary viewpoint.

The presented system makes extensive use of processing capabilities of modern Graphics Processing Units (GPUs) in order to achieve real-time processing rates while providing an acceptable quality of rendering results. Using a computer graphics based approach, the system takes advantage of a 3D mesh representation of the scene, which can be reconstructed from the ToF depth to generate the depth map corresponding to the viewpoint of the color camera. By fusing it with high-resolution color, a dense depth map is obtained resulting in a video-plus-depth data representation used for DIBR rendering on the display side.

To provide free-viewpoint functionality, a 3D scene is generated using mesh triangulation with depth information obtained after data fusion process. Color image is then used to texture the 3D surface. High rendering speed of such mesh-based representation allows reconstructing 3D dynamic scenes in real time. Virtual views can be synthesized by rendering the 3D scene geometry from the requested viewpoint. However, such mesh-based representation has a major drawback causing rubber-sheet artifacts at the virtual viewpoint. Thus, disocclusion areas still need to be detected and filled in a proper fashion.

This problem is addressed further and a layered resampling approach for free-viewpoint rendering from view-plus-depth data is proposed. The proposed method is suitable for general rendering scenario in terms of position, orientation, focal length and varying sensors spatial resolutions. The main benefits of the method can be summarized as follows: a high-quality texture resampling, avoiding non-uniform to uniform resampling, on-the-fly disocclusion detection and z-ordering. The experimental results demonstrate real-time capability of the proposed method even for CPU-based implementations, while the quality is comparable with other view synthesis approaches but for lower computational cost.

Future work will focus upon further optimizations of the proposed method. An optimal depth layering, which can be obtained by more careful analysis of the depth data of a scene, resulting in a less number of necessary layers would boost speed and quality performance of the resampling approach, as well as can provide a valuable information for the post-processing procedures, such as disocclusions filling. The future work also involves the speed optimized implementation of the proposed method utilizing general-purpose computing means of modern GPUs.

The current system operates only with one color view. As part of future work, it would be interesting to investigate the possible use of additional color views, which can be provided by the two side-cameras presented in the setup. Additional scene information captured from the different viewpoints can be used for more reliable depth estimation and for faster and more realistic disocclusions filling results in case of the virtual view generation. However, the increased number of active sensors in the setup would need more precise approaches for calibration and synchronization, e.g. hardware synchronization; additional two views of the scene would increase almost three times the data traffic between GPU and system memory; further, increasing computational complexity associated with the fusion and processing the data from multiple sources. Thus, an optimal trade-off between performance and visual quality of such system has to be found.

REFERENCES

[1] A. Smolic, "3D video and free-viewpoint video - From capture to display," in *Journal of Pattern Recognition*, vol. 44, no. 9, pp. 1958-1968, 2011.

[2] W. Sun, L. Xu, O. Au, S. Chui, and C. Kwok, "An overview of free view-point depth-image-based rendering (DIBR)," in *Proceedings of APSIPA Annual Summit and Conference*, pp. 1023-1030, 2010.

[3] A. Kolb, E. Barth, R. Koch, and R. Larsen, "Time-of-flight cameras in computer graphics," in *Proceedings of Eurographics Computer Graphics Forum*, vol. 29, no. 1, pp. 141-159, 2010.

[4] M. Lindner, A. Kolb, and K. Hartmann, "Data-fusion of PMD-based distance-information and high-resolution RGB-images," in *Proceedings of International Symposium on Signals Circuits and Systems (ISSCS)*, pp. 121-124, 2007.

[5] D. Chan, H. Buisman, C. Theobalt, and S. Thrun, "A noise-aware filter for real-time depth upsampling," in *Proceedings of European Conference on Computer Vision (ECCV), Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications*, 2008.

[6] B. Huhle, P. Jenke, and W. Strasser, "On-the-fly scene acquisition with a handy multi-sensor system," *International Journal on Intelligent Systems Technologies and Applications*, vol. 5, no. 3, pp. 255-263, 2008.

[7] A. Linarth, J. Penne, B. Liu, O. Jesorsky, and R. Kompe, "Fast fusion of range and video sensor data," *International Forum on Advanced Microsystems for Automotive Applications*, pp. 119-134, 2007.

[8] B. Bartczak and R. Koch, "Dense depth maps from low-resolution time-of-flight depth and high resolution color views," in *Proceedings of the International Symposium on Visual Computing (ISVC)*, pp. 228-239, 2009.

[9] S. Guðmundsson, H. Aanæs, and R. Larsen, "Fusion of stereo vision and time-of-flight imaging for improved 3D estimation," *Journal of Intelligent Systems Technologies and Applications*, vol. 5, no. 3, pp. 425-433, 2008.

[10] L. Do, G. Bravo, S. Zinger, and P. With, "GPU-accelerated real-time free-viewpoint DIBR for 3DTV," in *IEEE Transactions on Consumer Electronics*, vol. 58, no. 2, pp. 633-640, 2012.

[11] F. Garcia, D. Aouada, T. Solignac, B. Mirbach, and B. Ottersten, "Real-time depth enhancement by fusion for RGB-D cameras," *Journal by The Institution of Engineering and Technology(IET) Computer Vision*, vol. 7, no.5, pp. 1-11, 2013.

[12] L. Zhang, K. Zhang, J. Lu, T.-S.Chang, and G. Lafruit, "Low-complexity stereo matching and viewpoint interpolation in embedded consumer applications," in

Depth Map and 3D Imaging Applications: Algorithms and Technologies, IGI Global, 2010.

[13] C. Richardt, C. Stoll, N. Dogdgson, H.-P.Seidel, and C. Theobalt, "Coherent spatiotemporal filtering, upsampling and rendering of RGBZ videos," in *Proceedings of Eurographics Computer Graphics Forum*, vol. 31, no. 2, 2012.

[14] F. Sorbier, Y. Uematsu, and H. Saito, "Depth camera based system for autostereoscopic displays," in *Proceedings of IEEE International Workshop on Multimedia Signal Processing (MMSP)*, pp. 361-366, 2010.

[15] J. Kopf, M. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," in *IEEE Transactions on Computer Graphics (SIGGRAPH)*, vol. 26, no. 3, pp. 96–100, 2007.

[16] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision (IJCV)*, vol. 47, no. 1, pp. 7-42, 2002.

[17] R. Hartley and A. Zisserman, *Multiple-view geometry in computer vision, 2nd Edition*, Cambridge University Press, 2003.

[18] R. Koch and J.-F. Evers-Senne, "View synthesis and rendering methods," in *3D Video communication: Algorithms, concepts and real-time systems in human centered communication*, pp. 235–260, 2005.

[19] H.-Y. Shum and S. B. Kang, "A review of image-based rendering techniques," in *Proceedings of SPIE Visual Communications and Image Processing (VCIP)*, pp. 2-13, 2000.

[20] S. Zinger, L. Do, and P. de With, "Free-viewpoint depth image based rendering," *Journal Visual Communication and Image Representation*, vol.21, no. 5-6, pp. 533-541, 2010.

[21] P. Kauff, N. Atzpadin, and C. Fehn, "Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability," *Journal Signal Processing: Image Communication*, vol. 22, no. 2, pp. 217–234, 2007.

[22] A. Kubota, A. Smolic, M. Magnor, T. Chen, and M. Tanimoto, "Multi-view imaging and 3DTV," *IEEE Signal Processing Magazine, Special Issue on Multi-view Imaging and 3DTV*, vol. 24, no. 6, 2007.

[23] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," in *Proceedings of Computer Graphics (SIGGRAPH)*, pp. 39–46, 1995.

[24] L. Do, S. Zinger, and P. de With, "Quality improving techniques for free-viewpoint DIBR," in *Proceedings of Signal Processing and Electronic Imaging (SPIE)*, vol. 7524, 2010.

[25] A. Smolic, P. Kauff, S. Knorr, A. Hornung, M. Kunter, M. Muller, and M. Lang, "Three-dimensional video postproduction and processing," in *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, vol.4, pp. 607–625, 2011.

[26] Z. Tauber, Z. Li, and M. Drew, "Review and preview: Disocclusion by inpainting for image-based rendering," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* vol. 37, no. 4 pp. 527–540, 2007.

[27] C. Cheng, S. Lin, and S. Lai, "Spatio-temporally consistent novel view synthesis algorithm from video-plus-depth sequences for autostereoscopic displays," in *IEEE Transactions on Broadcasting*, vol. 57, no. 2, pp. 523–532, 2011.

[28] A. Criminisi, P. Pérez, and K. Toyama, "Object removal by exemplar-based inpainting," in *Proceedings of International Conference Computer Vison Pattern Recognition.*, vol. 2, no. 2, pp. 721–728, 2003.

[29] C. Vazquez, W. Tam, and F. Speranza, "Stereoscopic imaging: Filling disoccluded areas in depth image-based rendering," in *Proceedings of Signal Processing and Electronic Imaging (SPIE)*, vol. 6392, no. 5, p. 63920D*,* 2006.

[30] Y. Zhao, C. Zhu, and L. Yu, "Virtual view synthesis and artifact reduction techniques," in *3D-TV System with Depth-Image-Based Rendering – Architectures, Techniques and Challenges*, pp. 145–167, 2013.

[31] I. Daribo, H. Saito, R. Furukawa, S. Hiura, and N. Asada. "Hole filling for view synthesis," in *3D-TV System with Depth-Image-Based Rendering – Architectures, Techniques and Challenges*, pp. 169–189, 2013.

[32] "3D Content Creation Guidelines," "3D Interface Specification," Dimenco, B.V., [Online] Cited 18.08.2014 www.dimenco.eu.

[33] P. Merkle, A. Smolic, K. Müller, and T. Wiegand, "Multi-view video plus depth representation and coding," in *Proceedings of the IEEE International Conference on Image Processing*, vol. 1, pp. 201-204, 2007.

[34] P. Kerbiriou, G. Boisson, K. Sidibé, and Q. Huynhthu, "Depth-based representations: which coding format for 3D video broadcast applications?," *in Proceedings of Signal Processing and Electronic Imaging (SPIE)*, pp. 78630D-78630D-10, 2011.

[35] R. Lange and P. Seitz, "Solid state ToF range camera," *Journal of Quantum Electronics*, vol. 37, no. 3, pp. 390-297, 2001.

[36] T. Kahlmann, F. Remondino, and S. Guillaume, "Range imaging technology: new developments and applications for people identification and tracking," in *Proceedings of Video metrics IX - SPIE-IS&T Electronic Imaging*, vol. 6491, 2007.

[37] M. Linder and A. Kolb, "Calibration of the intensity-related distance error of the PMD ToF-camera," in *Proceedings SPIE, Intelligent Robots and Computer Vision*, vol. 6764, pp. 6764–35, 2007.

[38] B. Huhle, T. Schraier, P. Jenke, and W. Strasser, "Fusion of range and color images for denoising and resolution enhancement with a non-local filter," *Journal of Computer Vision and Image Understanding*, vol. 114, no. 12, pp. 1336-1345, 2010.

[39] D. Chan, H. Buisman, C. Theobalt, and S. Thrun, "A noise-aware filter for real-time depth upsampling," *Workshop on European Conference on Computer Vision (ECCV)*, vol. 1, no.1, pp. 1-12, 2008.

[40] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 839-847, 1998.

[41] A. Buades and J. Morel, "A non-local algorithm for image denoising," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 60-65, 2005.

[42] M. Frank, M. Plaue, H. Rapp, U. Koethe, B. Jaehne, and F. Hamprecht, "Theoretical and experimental error analysis of continuous-wave time-of-flight range cameras," *Journal of Optical Engineering*, vol. 48, no. 1, pp. 1-24, 2009.

[43] M. Frank, M. Plaue, and F. Hamprecht, "Denoising of continuous-wave time-of-flight depth images using confidence measures," *Journal of Optical Engineering*, vol. 48, no. 7, 2009.

[44] M. Georgiev, A. Gotchev, and M. Hannuksela, "Denoising of distance maps sensed by time-of-flight devices in poor sensing environment," *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[45] M. Georgiev, A. Gotchev, and M. Hannuksela, "Joint denoising and fusion of 2D video and depth map sequences sensed by low-powered ToF range sensor," in *Proceedings of International Conference on Media and Expo (ICME)*, pp. 1-4, 2013.

[46] A. Chuchvara, M. Georgiev, and A. Gotchev, "A speed-optimized RGB-Z capture system with improved denoising capabilities," in *Proceedings of Signal Processing and Electronic Imaging (SPIE)*, vol. 9019, 2014.

[47] Industry's Foundation of High Performance graphics, "Rendering pipeline overview,"[Online] Cited 18.08.2014 www.opengl.org/wiki/Rendering_Pipeline_Overview.

[48] Industry's Foundation of High Performance graphics, "OpenGL shading language," [Online] Cited 18.08.2014 www.opengl.org/wiki/OpenGL_Shading_Language.

[49] K. Simek, "Calibrated cameras in OpenGL without glFrustum," [Online] Cited 18.08.2014 www.ksimek.github.io.

[50] PMD Technologies GmbH, "PMD[Vision] CamCube 2.0," a PMD camera vendor in Siegen, Germany, 2010.

[51] Allied Vision Technologies Canada Inc., "AVT/Prosilica PvAPI programmers' reference manual," Canada, 2011.

[52] J. Bouguet, "Camera calibration toolbox for Matlab," [Online] Cited 18.08.2014 www.vision.caltech.edu.

[53] Z. Zhang, "A flexible new technique for camera calibration," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1330-1334, 2000.

[54] J. Heikkilä and O. Silvén, "A four-step camera calibration procedure with implicit image correction," in *Proceedings of IEEE Computer Vision and Pattern Recognition*, pp. 1106-1112, 1997.

[55] "3D Content Creation Guidelines," "3D interface specification," Dimenco, B.V., [Online] Cited 18.08.2014 www.dimenco.eu.

[56] C. Zhang and Z. Zhang, "Calibration between depth and color sensors for commodity depth cameras," in *Proceedings of International Conference on Media and Expo (ICME)*, pp. 1–6, 2011.

[57] C. Herrera and J. Kannala, "Joint depth and color camera calibration with distortion correction," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 2058–2064, 2012.

[58] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1, pp.7–42, 2002.

[59] Industry's Foundation of High Performance graphics, "Texture," [Online] Cited 18.08.2014 www.opengl.org/wiki/Texture.

[60] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "Bilateral filtering: Theory and applications," *Journal Foundations and Trends in Computer Graphics and Vision*, vol. 4, no. 1, pp. 1-73, 2008.

[61] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth International Conference on Computer Vision (ICCV)*, page 839, 1998.

[62] F. Lenzen, K. I. Kim, R. Nair, S. Meister, H. Schäfer, F. Becker, C. Garbe, and C. Theobalt, "Denoising strategies for time-of-flight data," in *Time-of-Flight Imaging: Algorithms, Sensors and Applications*, 2013.

[63] F. Lenzen, H. Schäfer, and C. Garbe, "Denoising time-of-flight data with adaptive total variation," in *Proceedings of International Symposium on Visual Computing (ISVC)*, vol. 1, pp. 337-346, 2011.

[64] Industry's Foundation of High Performance graphics, "Vertex texture fetch," [Online] Cited 18.08.2014 www.opengl.org/wiki/Vertex_Texture_Fetch.

[65] Industry's Foundation of High Performance graphics, "Framebuffer object," [Online] Cited 18.08.2014 www.opengl.org/wiki/Framebuffer_Object.

[66] Industry's Foundation of High Performance graphics, "Category: depth buffering," [Online] Cited 18.08.2014 www.opengl.org/wiki/Category:Depth_Buffering.

[67] M. Segal, C. Korobkin, R. Vanvidenfelt, J. Foran, and P. Haeberli, "Fast shadows and lighting effects using texture mapping," in *Proceedings of Computer Graphics (SIGGRAPH)*, pp. 249–252, 1992.

[68] R. Pajarola, M. Sainz, and Y. Meng, "Dmesh: Fast depth-image meshing and warping," *International Journal of Image and Graphics*, vol. 4, no. 4, pp. 653–681, 2004.

[69] Z. Tian, C. Xu, and X. Deng, "A parallel depth-aided exemplar-based inpainting for real-time view synthesis on GPU," in *Proceedings of International Conference on High Performance Computing and Communications (HPCC)/ International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 1739-1744, 2013.

[70] D. Pradhan, M. Naveen., A. Sai Hareesh, P. Baruah, and V. Chandrasekaran, "A computationally efficient approach for exemplar-based color image inpainting using GPU," in *Proceeding of 8th Annual IEEE International Conference on High Performance Computing HiPC 2011 Student Research Symposium*, 2011.

[71] J. Rosner, H. Fassold, P. Schallauer, and W. Bailer, "Fast GPU-based image warping and inpainting for frame interpolation," in *Proceedings of Computer Graphics, Computer Vision and Mathematics (GraVisMa) Workshop*, 2010.

[72] L. Do, G. Bravo, and S. Zinger, "GPU-accelerated real-time free-viewpoint DIBR for 3DTV," in *IEEE Transactions on Consumer Electronics*, vol. 58, no. 2, pp. 633-640, 2012.

[73] Industry's Foundation of High Performance graphics, "Performance," [Online] Cited 08.09.2014 www.opengl.org/wiki/Performance.

[74] H. Sankaran, M. Georgiev, A. Gotchev, and K. Egiazarian, "Non-uniform to uniform image resampling utilizing a 2D farrow structure," in *Proceedings of Spectral Methods and Multirate Signal Processing (SMMSP)*, 2007.

[75] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

[76] R. Collins, "A space-sweep approach to true multi-image matching," in *IEEE Transactions on Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 358–363, 1996.

[77] Blender Foundation, "Free and open source 3D animation suite," [Online] Cited 18.08.2014 www.blender.org.

[78] R. Keys, "Cubic convolution interpolation for digital image processing," in *Transactions on Signal Processing, Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.