LIGHT FIELD COMPRESSION AND CONVERSION WITH IMAGE-BASED RENDERING

自由視点画像合成に基づく光線空間情報の符号化と変換

BY

YUICHI TAGUCHI
田口 裕一

A DOCTORAL DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL OF
INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF INFORMATION SCIENCE AND TECHNOLOGY

SUPERVISOR: ASSOCIATE PROF. TAKESHI NAEMURA

DECEMBER 2008

Abstract

Recent advances in camera, computer, and display technologies have led researchers to develop 3D TV systems, which provide a more natural and intuitive perception of real scenes than 2D TV systems. Such a system captures multi-view images of a scene by using an array of cameras or lenses, transmits them, and presents a free-viewpoint video on 2D displays or a 3D image on 3D displays by using image-based rendering techniques. A 4D function that represents the light rays included in the multi-view image set is called light field.

This dissertation focuses on handling light field data captured with relatively dense, planar multi-view imaging systems, which are used for reproducing an entire scene rather than only a set of objects, and addresses compression and conversion problems of the light field data. Efficient compression techniques are essential for transmission due to the vast amount of data, typically consisting of tens or hundreds of views. Conversion of light field data is also a core technology of 3D TV systems, because the light field data reproduced by displays is different from the data captured by imaging systems in most cases. Image-based rendering can be considered a basic light field conversion, generating a free-viewpoint image from multi-view images.

In both light field compression and conversion, geometry information of the scene plays an important role, because it provides the correspondence of light rays in the light field; it helps compression methods to improve coding efficiency, while enabling conversion methods to enhance the quality of converted views. The first part of this dissertation therefore addresses dense two-frame stereo matching, a fundamental problem for estimating scene geometry from a set of images. We present an over-segmentation-based stereo method that jointly estimates segmentation and depth to overcome limitations of traditional segmentation-based stereo methods. For mixed pixels on segment boundaries, the method computes foreground opacity (alpha), as well as color and depth for the foreground and background, which gives a more complete understanding of the scene structure than estimating a single depth value.

The next part explores issues of light field compression. In particular, we focus on compression methods that are suitable for image-based rendering. We first present two compression methods that provide a novel scalability, which we call view-dependent scalability. The scalability enables us to render high-quality views around a significant viewpoint even at low bit rates and to

improve the quality of views away from the viewpoint with increasing bit rate. One method performs image-based rendering before the encoding process to generate an image at the significant viewpoint, which is located at the head of the encoded bitstream and acts as a reference image for predicting the input multi-view images. The encoded bitstream can be used with three rendering methods depending on the bit rate. The other method uses region of interest (ROI) coding to provide more flexible control of the view-dependent scalability. It is designed for interactive streaming of free-viewpoint videos to compensate smooth movement of the viewpoint. We then explore how we can exploit inter-view correlation in image-based rendering systems while keeping the computational cost low and the system configuration simple. For this purpose, we use a distributed multi-view coding approach, in which the inter-view correlation is exploited only at the decoder, and propose an efficient method that jointly performs decoding and rendering processes in order to directly synthesize novel images without having to reconstruct all the input images.

The last part describes live 3D TV systems using real-time light field conversion. The system presented first in this part performs real-time video-based rendering using an array of 64 cameras and a single PC. The system estimates a view-dependent per-pixel depth map to render a high-quality novel view. The rendering method is fully implemented on the GPU, which allows the system to efficiently perform capturing and rendering processes as a pipeline by using the CPU and GPU independently. We then show a live end-to-end 3D TV system using the 64-camera array and an integral-photography-based 3D display with 60 viewing directions. We present a fast and flexible conversion method from the 64 multi-camera images to the integral photography format. The conversion method first renders 60 novel images corresponding to the viewing directions of the display by using the above rendering method, and then arranges the rendered pixels to produce an integral photography image. All the conversion processes are performed in real time on the GPU of a single PC. The conversion method also allows us to interactively control rendering parameters for reproducing the dynamic 3D scene with desirable viewing conditions.

Acknowledgments

I would first like to thank my advisor, Associate Prof. Takeshi Naemura, for his expert guidance and strong support during my Ph.D. course, and for introducing me to many of the topics in this dissertation. I would also like to thank Prof. Hiroshi Harashima for his valuable suggestions and continuous encouragement. They gave me an excellent research environment with many opportunities to pursue research on 3D image processing and the freedom to explore whatever I was interested in. I have learned a lot of things from five years of interaction with them, which will definitely be useful for my future career.

I would like to express my appreciation to my committee members, Prof. Mitsuru Ishizuka, Prof. Katsushi Ikeuchi, Associate Prof. Kenjiro Taura, and Assistant Prof. Toshihiko Yamasaki, for their insightful comments and valuable feedback on this dissertation. I would also like to thank Prof. Hiroshi Yasuda of Tokyo Denki University, Prof. Kiyoharu Aizawa, and Associate Prof. Yoichi Sato for giving me helpful comments and suggestions at research meetings and conferences.

I would like to express my gratitude to all the members of Harashima Naemura laboratory. It was a wonderful experience for me to discuss a broad range of research topics with them and to share enjoyable space and time in our laboratory. In particular, I would like to thank Keita Takahashi, who gave me a lot of valuable advice and suggestions on my research, writings, and presentations throughout my Ph.D. course. I would also thank Yasuaki Kakehi for his careful advice on my presentations and demonstrations, Takafumi Koike for valuable discussions about 3D display technologies, and Kumiko Morimura for her support and encouragement in technical English lessons. I would like to express my special appreciation to the members of the SIGLF group for working closely with me to explore interesting research topics on light field, and for helping me develop the camera array system presented in this dissertation. I am also very grateful to Takashi Tanaka and all secretaries of our laboratory for supporting my research activities.

I would like to express my deepest gratitude to Bennett Wilburn, my mentor during my internship at Microsoft Research Asia in Beijing, for his strong support and encouragement, and for giving me great suggestions and ideas in the field of computer vision and graphics. The research on stereo reconstruction with mixed pixels using adaptive over-segmentation, presented in this dis-

sertation, was pursued under his guidance at Microsoft Research Asia. I would also like to thank C. Lawrence Zitnick and Sing Bing Kang of Microsoft Research, and Jian Sun of Microsoft Research Asia, who gave me helpful comments and suggestions in completing the research project. Special thanks go to Yasuyuki Matsushita and Moshe Ben-Ezra of Microsoft Research Asia, and Jun Takamatsu of Nara Institute of Science and Technology, for their valuable advice and stimulating conversations, and a lot of intern students from various countries for letting me share their energy and motivation. The four months spent in Beijing were among the most exciting days in my Ph.D. course.

Finally, I would like to thank my family for their endless encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent advances in camera, computer, and display technologies have led researchers to develop 3D TV systems, which provide a more natural and intuitive perception of real scenes than 2D TV systems. Such a system captures multi-view images of a scene by using an array of cameras or lenses, transmits them, and presents a free-viewpoint video on 2D displays or a 3D image on 3D displays by using image-based rendering techniques. A 4D function that represents the light rays included in the multi-view image set is called light field. We mainly focus on handling light field data captured with relatively dense, planar multi-view imaging systems, which are used for reproducing an entire scene rather than only a set of objects, and address compression and conversion problems of light field data, as shown in Fig. 1.1.

Efficient compression methods are essential for transmitting the vast amount of light field data, typically consisting of tens or hundreds of views. As well as high compression efficiency, light field compression methods should have functionalities suitable for the output of the systems. In this dissertation, we explore compression methods that are suitable for image-based rendering systems, which generate free-viewpoint images from multi-view image sets. We assume that an image at a certain viewpoint is more significant than images at the other viewpoints in many applications of image-based rendering systems. For such applications, we propose two compression methods that provide a novel scalability, which we call view-dependent scalability. The scalability enables us to render high-quality views around a significant viewpoint even at low bit rates and to improve the quality of views away from the viewpoint with increasing bit rate. We also deal with a distributed coding architecture to exploit inter-view correlation in image-based rendering systems while keeping the computational cost low and the system configuration simple.

Conversion of light field data is also a core technology of 3D TV systems, because there are many types of displays and the light field data reproduced by displays is typically different from

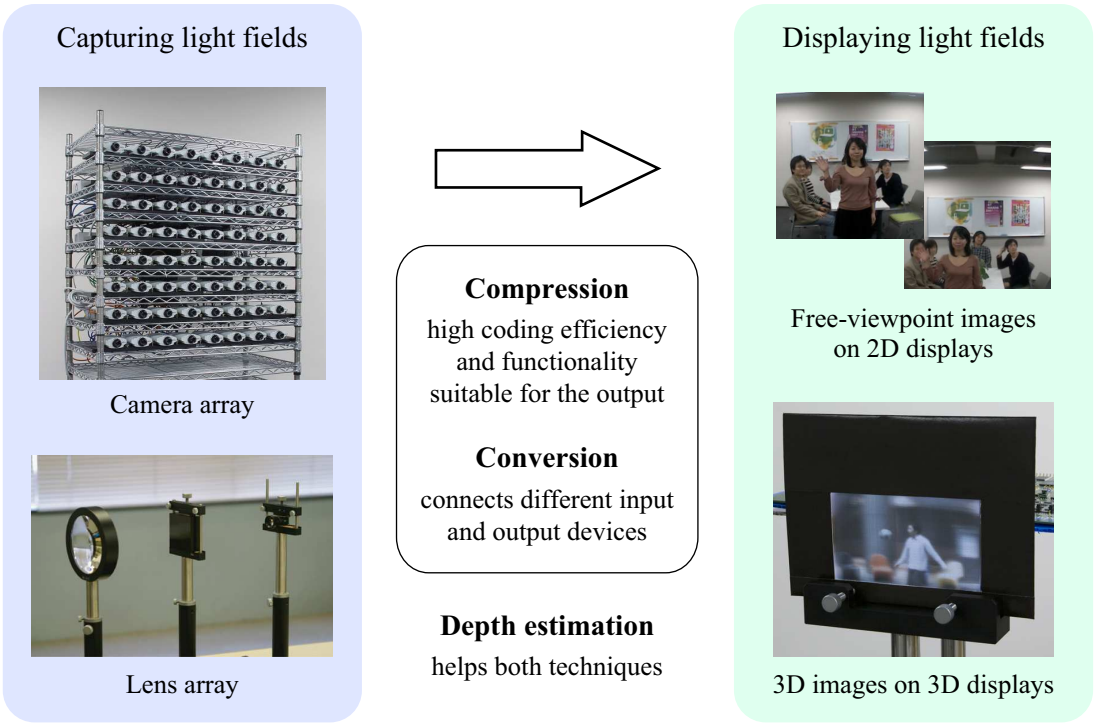


Figure 1.1: Topics of this dissertation.

the data captured by imaging systems. A basic light filed conversion is image-based rendering, which produces free-viewpoint images from multi-view image sets. Early image-based rendering systems often used static multi-view images captured by moving a single camera. More recently, many camera array systems have been developed with video-based rendering techniques to handle dynamic 3D scenes and produce interactive rendering applications. We present an online video-based rendering systems using an array of 64 network cameras, which generates a free-viewpoint video in real time from live multi-view videos. We also connect the camera array to an integral-photography-based 3D display using a real-time light field conversion method, which produces an end-to-end live 3D TV system. These live 3D TV systems could have a significant impact on many applications in communication, broadcasting, and entertainment.

In both light field compression and conversion, geometry information of the scene plays an important role, because it provides the correspondence of light rays in the light field. It enables compression methods to perform accurate prediction between views and improve compression efficiency. Meanwhile, it enables conversion methods to correctly interpolate light rays and enhance the quality of the converted views. Therefore, before describing the compression and conversion methods, this dissertation addresses dense two-frame stereo matching, a fundamental problem for

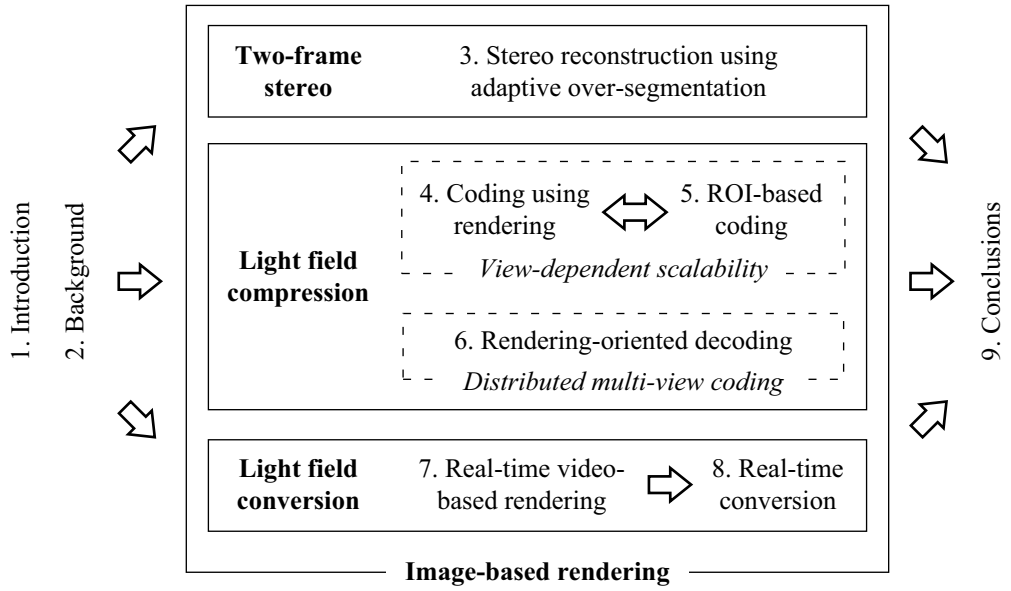


Figure 1.2: Organization of this dissertation.

estimating scene geometry from a pair of left and right images. The method we present for the two-frame stereo matching estimates an accurate depth map as an offline process, while the compression and conversion methods described in the following chapters use a method that estimates a relatively rough depth map, which is suitable for real-time processing.

1.2 Thesis Overview

This dissertation addresses the following three topics: two-frame stereo, light field compression, and light field conversion. Figure 1.2 shows the organization and the relationship between the chapters.

Chapter 2 describes background of our studies. We first review image-based rendering techniques based on a framework that represents visual information of a 3D scene as a collection of light rays. In particular, we detail a layer-based rendering method using view-dependent depth estimation, since the compression and conversion methods described in the following chapters use the rendering method. We then describe prior light field compression and conversion methods and clarify our contributions.

Two-Frame Stereo

Chapter 3 presents a stereo reconstruction method that estimates an accurate depth map from a pair of left and right images as an offline process. The method uses an over-segmentation approach and jointly estimates the segmentation and depth. For mixed pixels on segment boundaries, it computes foreground opacity (alpha), as well as color and depth for the foreground and background. We model the scene as a collection of fronto-parallel planar segments in a reference view, and use a generative model for image formation that handles mixed pixels at segment boundaries. Our method iteratively updates the segmentation based on color, depth and shape constraints using maximum a posteriori (MAP) estimation. Given a segmentation, the depth estimates are updated using belief propagation. We show that our method is competitive with the state-of-the-art using a standard stereo evaluation data set and that it overcomes limitations of traditional segmentation based methods while properly handling mixed pixels. Z-keying results show the advantages of combining opacity and depth estimation.

Light Field Compression

In the second part, we explore issues of light field compression. Chapter 4 presents a coding method that uses an image-based rendering method before the encoding process for providing the view-dependent scalability. The scalability enables us to render high-quality views around a significant viewpoint even at low bit rates and to improve the quality of views away from the viewpoint with increasing bit rate. The coder first synthesizes an image at the significant viewpoint, which we call representative viewpoint, and then predicts all input images by using the synthesized image as a reference. It produces a view-dependent scalable bitstream, which can be used with three rendering methods depending on the bit rate. Our experimental results show that the coder provides good coding efficiency, as well as the view-dependent scalability, for both multi-camera images and integral photography images, which are common light field representations.

In Chapter 5, we describe another coding method that provide more flexible control of the view-dependent scalability using a region of interest (ROI) approach. To synthesize a novel image at a certain viewpoint using multi-view images, typical renderers use only the part of the image segments that is aligned locally in a 4D light field. Our coding method exploits such locality of reference of light field when transmitting the multi-view data depending on a request from a remote user. It defines an ROI that includes the image segments used to synthesize the view requested by a user together with their neighboring segments. Since the data for the requested view are transmitted with the data for its neighboring views as the ROI, the user can render high-quality novel views around the requested viewpoint with this data. The coding method can thus compensate the smooth movement of a remote user even if the network has high latency. The user can arbitrarily choose the movable range of the viewpoint by changing the size of the ROI. Our

experimental results show that the ROI coding method keeps the reconstruction quality near the originally requested viewpoint high even at low bit rates and provides fine control of the view-dependent scalability.

Chapter 6 explores how we can exploit inter-view correlation in image-based rendering systems while keeping the computational cost low and the system configuration simple. For this purpose, we use a distributed multi-view coding approach, in which multi-view images are encoded independently at the encoder and the inter-view correlation is exploited only at the decoder. We present an efficient method for rendering a novel view from such data. The method combines decoding and rendering processes in order to directly synthesize the novel image without having to reconstruct all the input images. It jointly performs disparity compensation in the decoding process and geometry estimation in the rendering process, because they are essentially equivalent if the camera parameters for the input images are known. The method keeps both encoder and decoder complexity as low as that of a conventional intra-coding method, while attaining better coding performance owing to the inter-image decoding. We validate it by comparing the coding performance and the processing time of our method with those of an intra-coding method in experiments.

Light Field Conversion

The last part describes live 3D TV systems using real-time light field conversion. We first present a real-time video-based rendering system using a network camera array in Chapter 7. The system consists of 64 commodity network cameras that are connected to a single PC through a gigabit Ethernet. To render a high-quality novel view, the system estimates a view-dependent per-pixel depth map in real time by using a layered representation. The rendering algorithm is fully implemented on the GPU using GPGPU (General-Purpose computation on GPUs) techniques, which allows the system to efficiently perform capturing and rendering processes as a pipeline by using the CPU and GPU independently. Using QVGA input video resolution, the system renders a free-viewpoint video at up to 30 frames per second depending on the output video resolution and the number of depth layers.

We then present a live end-to-end 3D TV system using the 64-camera array and an integral photography display with 60 viewing directions in Chapter 8. The live 3D scene in front of the camera array is reproduced by the full-color, full-parallax autostereoscopic display, which gives users a perception of observing the 3D scene through a window without requiring them to wear special glasses. The main technical challenge is fast and flexible conversion of the data from the 64 multi-camera images to the integral photography format. Our conversion method first renders 60 novel images corresponding to the viewing directions of the display using the above rendering method, and then arranges the rendered pixels to produce an integral photography image.

For real-time processing on a single PC, all the conversion processes are implemented on the GPU. The conversion method also allows a user to interactively control viewing parameters of the displayed image for reproducing the dynamic 3D scene with desirable conditions. This control is performed as a software process, without reconfiguring the hardware system, by changing the rendering parameters such as the convergence point of the rendering cameras and the interval between the viewpoints of the rendering cameras.

Finally, Chapter 9 summarizes the contributions of the dissertation and discusses possible future directions.

Chapter 2

Background

2.1 Image-Based Rendering

If it is possible to capture all light rays filling a 3D space, we can generate correct views of the space at arbitrary viewpoints by selecting the necessary light rays from the captured light rays. Based on this concept, image-based rendering (IBR) techniques consider how to sample light rays of a 3D scene and how to render novel views by interpolating the sampled light rays. IBR techniques have attracted a lot of research interest since they reproduce photorealistic images of the 3D scene by using a set of images of the scene, which can be acquired easier than a 3D model and its properties of the scene required by traditional model-based rendering techniques.

To describe all light rays in a 3D space, Adelson and Bergen [2] introduced plenoptic function. The plenoptic function is a 7D function

$$P(V_x, V_y, V_z, \theta, \phi, \lambda, t), \tag{2.1}$$

which represents the intensity of light rays passing through the camera center at every 3D location (V_x, V_y, V_z), at every possible angle (θ, ϕ), for every wavelength λ, and at every time t. In practice, it is simplified by omitting dimensions, such as the wavelength and time, and by restricting the viewing space of the viewers. Table 2.1 summarizes example representations of simplified plenoptic function, based on Shum et al.'s [100] and Zhang and Chen's [140] survey papers.

IBR techniques have a tradeoff between the density of sample images and the amount of geometry information needed to be estimated or acquired for rendering high-quality views; we need no or rough geometry models when using densely sampled image sets, whereas we need accurate geometry models when using sparsely sampled image sets. Chai et al. [14] theoretically analyzed the tradeoff and formulated minimum sampling curves, which describe the relationship between the density of images and the number of depth layers required for rendering novel views without aliasing artifacts.

Table 2.1: Various representations of the plenoptic function.

Dimension	Example representations
7D	Plenoptic function
6D	Surface plenoptic function
5D	Plenoptic modeling, Light field video
4D	Light field/Lumigraph
3D	Concentric mosaics, Panoramic video, Video
2D	Panorama/Image mosaicing, Image

In the following subsections, we review representative IBR techniques by classifying them into three categories according to Shum et al.'s taxonomy [100]: rendering with no geometry, rendering with implicit geometry, and rendering with explicit geometry. We also explain a layer-based rendering method using view-dependent depth estimation, which we use in our compression and conversion methods described in the following chapters.

2.1.1 Rendering with No Geometry

The rendering methods classified into this category rely on densely sampled light ray data. McMillan and Bishop [75] introduced plenoptic modeling as a 5D plenoptic function by removing the time and wavelength from the 7D plenoptic function. They recorded cylindrical panoramas (2D) of a static scene at multiple camera positions (3D) to sample the 5D function. To render a novel view, they warped the recorded panoramas to the novel viewpoint based on computed epipolar geometry and disparity images between each image pair; therefore, they used implicit geometry to render novel views from sparsely sampled panoramas, although their plenoptic modeling framework can be used with no geometry for densely sampled panoramas.

Since it can be assumed that light intensity remains constant along its trajectory, arbitrary light rays can be described with light rays that pass through a boundary surface of an object or a scene, as long as viewers stay outside the boundary surface. Such light rays are parameterized by their intersections (2D positions) with two parallel planes, or by their positions (2D) and directions (2D) on a single plane. These 4D representations are called light field [59], lumigraph [39], or ray space [29]. Early systems [39, 59] captured the light rays of a static scene by moving a single camera. For rendering a novel view, they interpolated each light ray in the rendered view by using nearest samples of the light ray in the densely sampled images. More recently, many camera array systems have been developed to handle dynamic 3D scenes by recording light field video, a 5D plenoptic function. In such camera array systems, it is impractical to arrange the cameras enough densely. Therefore, if we use no geometry information (approximating the scene geometry as a single plane), novel images are synthesized with blur and ghosting (aliasing) artifacts [14, 45].

To avoid these artifacts, recent systems typically use rendering methods that estimate explicit geometry models, as we describe in the following subsections.

Shum and He [101] presented a 3D parameterization of the plenoptic function, called concentric mosaics. They captured a static scene along concentric circles by spinning cameras on a rotary table. For rendering a novel view, they interpolated each vertical slit in the novel view by using neighboring slits in the captured images. This representation allows viewers to move the viewpoint in the 2D circle.

Image mosaics, generated by stitching multiple image together, can be considered the simplest 2D plenoptic function having a single fixed viewpoint. Many systems have been built to produce image mosaics as cylindrical and spherical panoramas (e.g., [12, 18, 106]). Another interesting representation of image mosaics is called scene collages [83] or joiners [138]. The representation is geometrically incorrect, but gives us a comprehensive view of the scene like panoramas. Nomura et al. [83] and Zelnik-Manor and Perona [138] presented methods that automatically produce scene collages by extracting SIFT features [64] from the input images and computing an optimal similarity transform for each image, instead of homography between images for producing panoramas, so that the distances between corresponding features are minimized. Nomura et al. [83] also produced scene collages for dynamic scenes using camera arrays that can be physically flexed by the user to vary the composition of the scene.

2.1.2 Rendering with Implicit Geometry

The rendering methods classified into this category use positional correspondences across a small number of images to render novel views. A representative method, called view interpolation [19], computes dense optical flow between two input views and generates intermediate views based on the flow. The intermediate view may not necessarily be geometrically correct. View morphing [97] is a specialized version of view interpolation, except that the interpolated views are always geometrically correct. Wilburn et al. [123] presented a spatiotemporal view interpolation method that estimates optical flow among views captured from different viewpoints and at slightly different times.

Rendering methods that use geometric constraints between a relatively small number of input images to reproject image pixels at a novel camera viewpoint are called transfer methods [58, 100]. Laveau and Faugeras [58] used the epipolar constraints between a collection of input images to find corresponding pixels in two reference view, which are used to interpolate a pixel in a novel view. For rendering novel views using two or three input images, Avidan and Shashua [6] presented a method that computes dense point correspondences between two images, recovers trilinear tensor (when using only two images, the method replicates one of the input images), and generates novel views by using the point correspondences and a new trilinear tensor computed for

the novel viewpoints.

2.1.3 Rendering with Explicit Geometry

The rendering methods classified into this category use an explicit geometry model of the scene. A representative geometry model is a depth map, which indicates per-pixel depth values of an image. The depth map is typically used for representing the geometry of an entire scene. If we have depth maps for each image, 3D warping techniques (e.g., [71]) can be used to render novel views. These techniques project the pixels of the input images to their proper 3D locations and reproject them onto the novel viewpoint. To avoid the occlusion artifacts appeared with 3D warping methods, layered depth images (LDIs) [98] and LDI trees [16] were proposed, in which each pixel contains more than one depth and color values.

Another explicit geometry model is a 3D voxel or mesh model, which is suitable for representing an object rather than an entire scene. Such 3D models are reconstructed from multi-view images by using the silhouette of the objects [57] as well as color consistency between input views [56, 96]. If we use a single texture map for a single object model, as conventional texture mapping does, we cannot obtain view-dependent visual effects, such as highlights and reflections. To produce the view-dependent visual effect of the real scene, Debevec et al. [25, 26] used view-dependent texture mapping that warps and composites multiple texture maps captured from different viewpoints to the same object surface. Surface light field [20, 125] is another representation using an explicit object model, in which the light rays included in input images are remapped onto a 3D mesh model scanned by a range sensor.

Recent video-based rendering systems using camera arrays have often used these explicit geometry models, because the cameras cannot be arranged enough densely in practice. Since estimating accurate geometry models requires high computational cost, some systems reconstruct the geometry models as an offline process and perform real-time rendering using the reconstructed geometry model. Meanwhile, another type of systems estimates rough geometry models for online (real-time) rendering using live multi-view videos. We review the recent video-based rendering systems in Section 7.2 by comparing them with our camera array system.

2.1.4 Layer-Based Rendering Method Using View-Dependent Depth Estimation

This dissertation mainly focuses on handling light field data captured with planar multi-view imaging systems, which are used for reproducing an entire scene rather than only a set of objects. Since cameras cannot be arranged enough densely in practice, we need to estimate the scene geometry, rather than approximating it as a single plane, for synthesizing novel views without blur and ghosting artifacts [14, 45]. One of the most widely used and effective methods is plane sweeping [24], which computes view-dependent depth maps by evaluating color consistency among light rays.

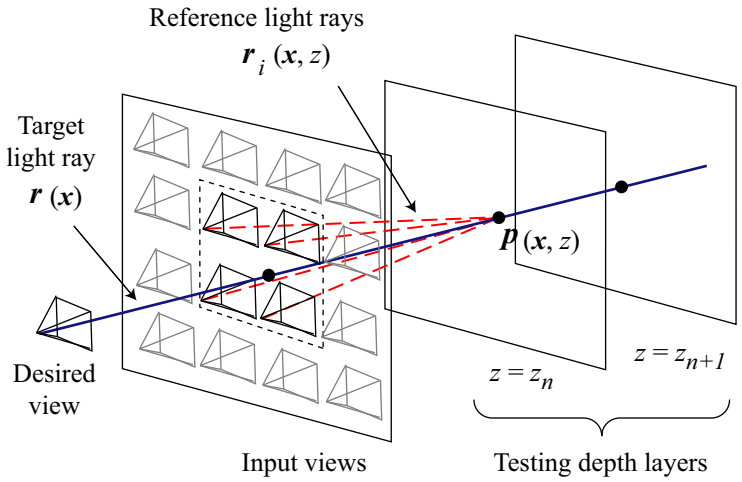


Figure 2.1: Configuration for rendering a desired view.

Here we explain a variant of the plane-sweeping methods presented in [107, 108, 109], because the light field compression and conversion methods described in the following chapters use the rendering method.

The rendering method assumes that multi-view images are captured with calibrated cameras that roughly lie on a plane and are arranged on a 2D grid, and that there is no prior knowledge of the scene geometry. As shown in Fig. 2.1, the method sets a layered depth model, $z = \{z_n | n = 1, 2, .., N\}$, in the object space to equally divide the disparity space as

$$\frac{1}{z_n} = \frac{1}{z_{max}} + \frac{n - 1/2}{N}\left(\frac{1}{z_{min}} - \frac{1}{z_{max}}\right), \tag{2.2}$$

where z_{max} and z_{min} are the maximum and minimum depths of the scene, respectively. As Chai et al. [14] showed, the number of depth layers required for appropriate interpolation of light field data depends on the camera intervals and resolutions. Using a small number of depth layers typically produces low-quality images with blur and ghosting artifacts. Meanwhile, using a larger number of depth layers produces higher-quality images, but needs more computational cost. We empirically choose an appropriate number of depth layers that produces enough visual quality while keeping the computational cost low.

The rendering method estimates the depth for each target light ray, $\boldsymbol{r}(\boldsymbol{x})$, where $\boldsymbol{x}$ represents the position of the light ray in the desired view. At the intersection of the target light ray with each of the depth layers ($\boldsymbol{p}(\boldsymbol{x}, z)$), the method evaluates the color consistency (focus measure in [107, 108, 109]) of the reference light rays, which correspond to the back-projections of the intersection point to the input cameras. The reference light rays are denoted by $\boldsymbol{r}_i(\boldsymbol{x}, z)$, where i is the camera index. In the color consistency evaluation, using a larger number of input (reference)

cameras makes the evaluation stable because of a larger stereo baseline [86]. However, it increases occlusion effects (some of the light rays may be occluded by foreground objects) and needs higher computational cost. To make the occlusion effects small and keep the computational cost low, we perform the evaluation only using the k-nearest reference cameras of the target light ray. The color consistency cost is therefore given by

$$C(\boldsymbol{x}, z) = consistency\left(I(\boldsymbol{r}_i(\boldsymbol{x}, z))|_{i \in V}\right), \tag{2.3}$$

where V is the set of camera indices near the target light ray and $I(\cdot)$ denotes the color of the light ray. We set $|V| = k = 4$, as shown in Fig. 2.1.

The color consistency cost is then smoothed in each depth layer in order to reduce noise effects. We average the cost over a square window

$$\bar{C}(\boldsymbol{x}, z) = \frac{1}{|W|} \sum_{\boldsymbol{x}' \in W} C(\boldsymbol{x}', z), \tag{2.4}$$

where W is a square window whose center is $\boldsymbol{x}$.

Finally, the depth value that minimizes the cost is selected for each target light ray:

$$z_{opt}(\boldsymbol{x}) = \arg\min_{z} \bar{C}(\boldsymbol{x}, z). \tag{2.5}$$

For the color interpolation of the target light ray, using too many reference cameras would produce an unnecessarily blurred result, because the object point from which the target light ray comes may not be completely diffusive and the projection of reference light rays may not be perfect. Therefore, we only use k-nearest reference light rays similarly to the depth estimation. The k-nearest approach can keep the view-dependent components of the target scene and prevent the blur [13]. We use bilinear interpolation of the colors of the reference light rays for the optimal depth:

$$I(\boldsymbol{r}(\boldsymbol{x})) = \sum_{i \in V} w_i(\boldsymbol{x})\, I(\boldsymbol{r}_i(\boldsymbol{x}, z_{opt}(\boldsymbol{x}))). \tag{2.6}$$

Here, $w_i(\boldsymbol{x})$ is the weight for the i-th reference light ray $\boldsymbol{r}_i(\boldsymbol{x}, z_{opt}(\boldsymbol{x}))$, and it takes a floating-point value between 0 and 1 depending on the positions of the reference cameras and the intersection point of the target light ray with the input camera plane; $w_i(\boldsymbol{x})$ takes 1 if the target light ray passes through the i-th camera position, while it takes 0 if it passes through the other neighboring camera positions, and $\sum_{i \in V} w_i(\boldsymbol{x}) = 1$. This weight considers the distance penalty between the reference camera and the intersection point of the target light ray with the input camera plane. As described in [13], the angular penalty between the reference light ray and the target light ray would be a more natural measure. However, we use the distance penalty because it can be efficiently implemented on a GPU by using texture mapping of an image that encodes the weight values (see Chapter 7 for implementation details).

Note that the reference camera set V depends on the position of each target light ray $\boldsymbol{x}$. Therefore, the number of input cameras used for rendering the entire view depends on the viewpoint. This rendering method, however, has constant computational complexity regardless of the number of input cameras, because it calculates the color and cost for each target light ray. The computational complexity is determined by the number of target light rays (i.e. the resolution of the desired view) and the number of depth layers.

The method may assign incorrect depth values in textureless regions, where several depth layers have similar color consistency costs. For image-based rendering, however, the depth values do not need to be correct as long as the interpolated color is visually correct. The method interpolates such visually correct colors by selecting the reference light rays with the minimum color variance.

2.2 Compression Techniques

As we described in Section 2.1, IBR techniques rely on a huge amount of image data. Efficient compression techniques are therefore essential for transmission and storage of IBR data sets. Since IBR systems capture an identical scene from slightly different viewpoints, significant correlations exist among the multi-view images. Implicit or explicit geometry models of the scene are used to exploit the correlations by inter-view prediction. Generally speaking, the compression efficiency becomes higher with increasing the accuracy of the geometry models.

Another requirement for the compression techniques is functionality that is suitable for the rendering. An important functionality is random access to the compressed data, because IBR techniques do not require all of the light rays for rendering a single view. Scalability is another functionality, which increase the quality of the data sets with increasing bit rate. In most cases, there is a trade-off between the compression efficiency and the random access; predictive methods typically improve the compression efficiency by exploiting the inter-view correlation, but they introduce dependencies between views, which restrict random access to the data. Researchers therefore explore appropriate prediction structures of compression methods, such as restricting the prediction level and using a hierarchical prediction order.

In the following subsections, we classify compression techniques into three categories, which are similar to those for the rendering techniques: compression with no geometry, compression with implicit geometry, and compression with explicit geometry.

2.2.1 Compression with No Geometry

The compression techniques classified into this category do not use inter-view prediction for supporting random access to the compressed data or for developing real-time encoding and decoding systems. Early IBR systems [59, 101] use vector quantization (VQ) to overcome the random ac-

cess problem. VQ first constructs a set of codewords that approximate input vectors of samples well during a training phase, and then quantizes the input vectors to one of the codewords. Since the decoding process just selects a codeword corresponding to the index of the vector, it provides random access and selective decoding of necessary light ray data.

Several camera array systems independently encode individual views by using conventional coding standards, such as motion JPEG [139] and MPEG-2 [74, 123], which means that they do not exploit the inter-view correlation. Although the compression efficiency is limited, such approaches are suitable for developing practical systems, because they keep the system configuration simple and make the video of each view independently decodable. They basically aim for real-time transmission and rendering, while the following approaches using inter-view prediction are suitable for storing the data.

2.2.2 Compression with Implicit Geometry

Conventional video coding standards use motion-compensated prediction (MCP) to exploit the inter-image correlation among temporal frames. Disparity-compensated prediction (DCP) [65] is a straightforward extension of MCP to the spatial (inter-view) direction. Since DCP methods find the correspondence between spatial views, they implicitly compute scene geometry as a disparity vector field.

It is important to design appropriate prediction structures to produce encoded bitstreams that are suitable for IBR while enabling high compression efficiency. Magnor and Girod [68] and Zhang and Li [141, 142, 143] restrict the prediction structure to one level (inter-coded views only refer to intra-coded views) to facilitate the random access. Another approach is using a hierarchical prediction structure that achieves progressive decoding of the light field data [67]. The hierarchical prediction method enables us to improve rendering quality with increasing bit rate.

For compressing multi-view videos, many techniques [15, 62], including currently developed MPEG standard for multi-view video coding (MVC) [76, 103], use MCP and DCP together. As well as attaining high compression efficiency, these techniques consider functionalities of 2D compatibility, which means that the video sequence of a main view is decodable as a conventional 2D video, and view scalability, which allows viewers to arbitrarily choose the number of decoded views depending on the bit rate.

2.2.3 Compression with Explicit Geometry

Compression methods using explicit geometry models of the scene can have higher compression efficiency than DCP methods. This is because the compression methods can perform per-pixel prediction between views, instead of per-block prediction of the DCP methods, if accurate geometry models are available. Moreover, global 3D models of the scene can be compactly encoded

compared to disparity vectors that DCP methods compute for all pairs of views.

Magnor et al. [66, 69, 70] presented compression methods for multi-view images of an object, captured on a hemisphere around the object. Their methods use a reconstructed 3D mesh model of the object in different ways. The model-aided predictive coding presented in [66, 70] uses the object model to warp and predict new views from already encoded images. The prediction is performed in a hierarchical manner for supporting progressive decoding. Meanwhile, the progressive texture-based coding presented in [69, 70] uses the object model to convert the input images to view-dependent texture maps, which have higher inter-image correlation than the original input images. The view-dependent texture maps are encoded with a 4D SPIHT [93] wavelet coding method. Their experimental results showed that the texture-based coding yields better compression performance if exact 3D scene geometry is available, while the model-aided coding attains better performance with approximate geometry. Ziegler et al. [146] extended the texture-based coding approach to handle multi-view videos of a dynamic object.

Surface light fields [20, 125] can be efficiently compressed, because the light rays defined on the surface of an object model are highly correlated. Wood et al. [125] presented two methods to efficiently compress the surface light fields based on generalization of vector quantization and principal component analysis. Chen et al. [20] presented another approach that approximates and compresses the surface light fields for hardware-accelerated rendering.

2.3 Conversion Techniques

In Section 2.1, we addressed how the imaging systems capture light rays of a 3D scene and IBR techniques generate free-viewpoint images using the light rays. Although free-viewpoint images are suitable for observing a 3D scene on 2D displays, presenting the 3D scene information on 3D displays gives us a more natural and intuitive perception. We can consider displays to be devices reproducing plenoptic functions; 2D displays reproduce a 2D plenoptic function, while 3D displays reproduce a higher-dimensional plenoptic function.

In particular, this dissertation focuses on autostereoscopic multi-view 3D displays. Because such displays present different views to different directions, they do not require the viewers to wear special glasses. Parallax-barrier displays and lenticular displays reproduce 2D views with 1D (either horizontal or vertical) parallax (i.e., 3D plenoptic function), whereas integral photography displays using microlens arrays reproduce 2D views with 2D (both horizontal and vertical) parallax (i.e., 4D plenoptic function). Figure 2.2 shows the principle of integral photography displays. In integral photography images presented on the displays, images behind each microlens are called elemental images, while images formed by extracting the pixels corresponding to the same position under each microlens are called sub-aperture images [82] (or subimages [3]). A set of elemental images is interchangeable with a set of sub-aperture images; the number of pixels in

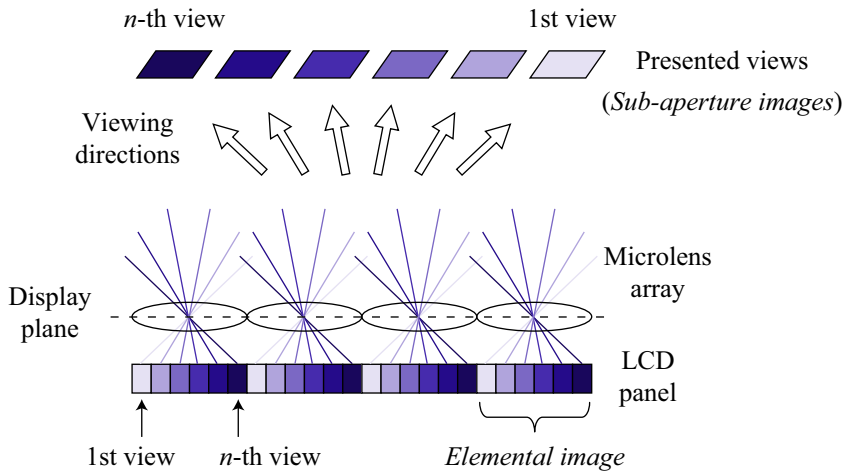


Figure 2.2: Principle of lenticular/integral photography displays. The number of pixels behind each microlens determines the number of the presented views, and a pixel position corresponds to a viewing direction.

the elemental images determines the number of the sub-aperture images (presented views), and a pixel position corresponds to a viewing direction.

Traditionally, capturing and display devices use an identical optical system [47, 85]; for example, stereoscopic cameras are used for stereoscopic displays, and two identical lens arrays are used for integral-photography-based systems. This restricts the combinations of the capturing and display devices, and requires precise calibration between them, which is impractical especially when we use multiple cameras and multi-view 3D displays. Data conversion techniques are therefore essential to develop flexible 3D TV systems using any combinations of the capturing and display devices. IBR can be considered a basic conversion method, generating a free-viewpoint image from multi-view images.

Isaksen et al. [45] presented a method that generates integral photography images from light fields captured with multiple cameras. Their method renders elemental images and composites them together to produce an integral photography image. As they stated, an advantage of such conversion methods using IBR is that a variety of different integral photography images can be generated from the same light field by changing the position and orientation of the rendering cameras. Matusik and Pfister [74] developed a live 3D TV system using 16 cameras and 16 projectors with lenticular screens. They used an IBR method to correct the misalignment of the input camera viewpoints and generate the aligned multi-view images presented on the lenticular displays. For CG scenes, Yang et al. [135] compared two rendering approaches for generating integral photography images: rendering a set of elemental images and rendering a set of sub-

aperture images. Their experimental results showed that rendering the image sets consisting of smaller number of images (i.e., having higher image resolution) is desirable to accelerate the rendering, because it reduces the number of rendering passes.

The interchangeability between elemental images and sub-aperture images is also used to efficiently compress integral photography images. Basically, compressing a set of images that have higher pixel density (resolution) provides higher efficiency, because the compression methods can exploit spatial correlations between pixels. Yeom et al. [137] encoded a set of elemental images with an MPEG-2 coder by considering the image set a video sequence, since they used a small number of high-resolution elemental images. On the other hand, Olsson et al. [87] encoded a set of sub-aperture images with an H.264 coder, since they used a large number of low-resolution elemental images. Miller et al. [77] used a similar approach for compressing surface light fields. They formed images with the light rays that have an identical direction on an object surface (corresponding to sub-aperture images) and encoded them, which allows them to efficiently decode necessary light rays for rendering novel views.

2.4 Summary

In this chapter, we have briefly reviewed IBR techniques, which can be considered basic conversion from multi-view images to a free-viewpoint image. We have also described more general conversion methods, which produce autostereoscopic views on multi-view 3D displays using the IBR techniques. Accurate geometry models enable us to render high-quality views from sparsely sampled input views, but estimating such geometry models requires high computational cost. The stereo reconstruction method we present in Chapter 3 aims to produce accurate depth maps as an offline process. Meanwhile, such time-consuming methods are not applicable for real-time systems. The live 3D TV systems we present in Chapters 7 and 8 therefore use the rendering method described in Section 2.1.4, which estimates a rough depth map at the rendering viewpoint in real time. As we have described in Section 2.3, conversion of light field data is essential for developing practical 3D TV systems, because it allows us to use any combinations of capturing and display devices. Chapters 7 and 8 show that we can use the light field captured with our camera array system to generate both a free-viewpoint video and an integral photography video with various viewing parameters.

This chapter has also described compression techniques for IBR data sets. As well as attaining high compression efficiency by using the geometry models of the scene, the compression techniques should have functionality suitable for the output of systems. In Chapters 4 and 5, we propose compression methods that have a novel functionality, which we call view-dependent scalability. The scalability extends the range of viewing area where we can render high-quality views with increasing bit rate, which is suitable for browsing a free-viewpoint video by using IBR

techniques. Meanwhile, view scalability, which we have explained in Section 2.2.2, increases the number of input views to be decoded with increasing bit rate. It is suitable for applications that present the decoded views themselves to 2D/3D displays. Chapter 6 considers how we can exploit inter-view correlation in IBR systems while keeping the computational cost low and the system configuration simple. The method presented in Chapter 6 uses a distributed coding approach, where the inter-view prediction is performed only at the decoder, and jointly performs the decoding and rendering processes to keep the system complexity as low as an intra-coding system. The method is designed for real-time rendering systems, whereas typical inter-view prediction methods described in Section 2.2 consider the encoding as an offline process.

Chapter 3

Stereo Reconstruction with Mixed Pixels Using Adaptive Over-Segmentation

3.1 Introduction

Dense stereo matching is challenging in the presence of occlusions and textureless image regions. Color segmentation based methods have been shown to effectively handle these cases [11, 43, 52, 111, 147]. These approaches assume that depth varies smoothly within regions of homogeneous color and that depth discontinuities coincide with color boundaries. This assumption helps resolve the depth ambiguity within textureless regions and allows for precise delineation of object boundaries corresponding to depth discontinuities.

A drawback of segmentation based stereo is that depth discontinuities may not lie along color boundaries. As a result, image segmentations based on color information may contain segments that span depth discontinuities. If the color segmentation is held fixed, errors will result in the final depth map [111, 147]. We present an approach that overcomes initial segmentation errors by jointly estimating depth and image segmentation.

Most dense stereo methods compute a single depth value for each pixel. For mixed pixels (pixels which span two objects at different depths), computing the depths of the foreground and background components of the pixel gives a more complete understanding of the scene structure. Moreover, one must compute the opacity (alpha) and foreground/background colors for mixed pixels in order to get high-quality results for Z-keying and view interpolation. Alpha estimation is usually done as a post-processing step [23, 42, 92, 147], given a presumed pixel-accurate depth map. We incorporate alpha estimation into the depth and segmentation computation to produce more accurate results. To be clear, our goal is not calculating matting and depth information for

The work presented in this chapter was done while I was visiting Microsoft Research Asia.

very fuzzy or hairy foreground objects. Instead, we focus on the mixed pixels that occur along the boundaries of nearly all objects in the scene, at all depths.

The stereo reconstruction method presented in this chapter jointly estimates image segmentation, depth, and matting/depth information for mixed pixels. We use an over-segmentation approach to represent a scene as a collection of fronto-parallel planar segments. The segments are characterized by their depth, 2D shape, and color. These parameters are jointly estimated by alternating the update of segment shapes and depths. To update the segment shapes, we use a generative model that accounts for mixed pixels at the segment boundary as well as the depth and shape probabilities. To update the segment depths, we define a pairwise Markov random field for the segments, and minimize its energy using belief propagation. The algorithm explicitly handles occlusions by checking the visibility of pixels based on the previous estimates of segment depths.

The rest of this chapter is organized as follows. In the next section, we review the prior art and identify our contributions. Section 3.3 describes our scene representation and stereo image model. Section 3.4 explains how we infer the scene structure. In Section 3.5, we validate our methods using the new Middlebury stereo evaluation [94]. Our method is ranked fourth best (as of December 2007), and performs well on image pairs that confound most segment-based approaches. A Z-keying example shows the ability of the algorithm to extract alpha values across the entire range of depths in the scene. Finally, we close with a discussion of the strengths and weaknesses of our method, and some comments on the treatment of mixed pixels in the Middlebury stereo evaluation.

3.2 Related Work

This section describes prior work related to segmentation-based stereo and alpha matting. For a comprehensive review of dense two-frame stereo methods, we refer the reader to Scharstein and Szeliski's taxonomy and evaluation [94]. Here, we review stereo methods that use planar scene representations. Wang and Adelson [120] decompose images into multiple layers for motion analysis. They iteratively update the layers using affine motion analysis and clustering. They cluster based on flow, which can be inaccurate near occlusion boundaries. Baker et al. [7] used a layered scene representation with alpha for stereo reconstruction. They represent a stereo scene as a collection of planes with per-pixel depth offsets. They refine estimates of the plane equation and depth offset for each layer using an algorithm that accounts for occlusion and mixed pixels, but the initialization of the scene layers is not automatic. Tao et al. [111] present a method using color over-segmentation and a piecewise planar scene representation that inspired many other researchers [11, 43, 52, 147]. These methods perform well for reasons discussed earlier, but they all segment the input images in a pre-processing step and cannot recover from segmentation errors. Deng et al. [27] partially overcome this vulnerability by subdividing segments from one

image using segment boundaries from the other, creating what they call patches. Their method improves the stereo estimation, but because it updates the patches, not the segmentations, it is still vulnerable to initial segmentation errors.

With the exception of Baker et al. [7], the work mentioned above does not account for the partial opacity of mixed pixels on object boundaries. Much of the work in this area has concentrated on digital matting (extracting a foreground object from an image or video). For example, Chuang et al. [23] and Ruzon and Tomasi [92] propose matting methods that use user-defined trimaps. These trimaps specify three regions in the image: background, foreground, and the undefined area in which the algorithm must compute the foreground opacity and color. These are matting, not stereo, methods, so they do not compute depth. They require a user-defined trimap, and they assume the image has clearly separable foreground and background components.

Some researchers have proposed stereo or optical flow methods that explicitly account for alpha and are fully automatic. Zitnick et al. [147] propose a video view interpolation method that computes depth using segmentation-based stereo, uses the depth map to automatically create a trimap, and then computes opacity and foreground/background color information using Bayesian matting [23]. Hasinoff et al. [42] also propose a multi-view stereo method that refines a depth map by modeling occlusion boundaries as 3D curves. Both methods first compute a depth map with a single depth value per pixel, then refine the depth and compute matting information. As such, they are vulnerable to errors in the depth map computation, although Hasinoff et al. can overcome small errors.

Zitnick et al. [148] present an optical flow method that computes a consistent segmentation of two or more images in a sequence and also accounts for mixed pixels. Their method produces good optical flow results and has the advantage of updating the image segmentations, but it is not directly applicable to stereo because it does not handle occlusions or account for stereo constraints. Finally, Xiong and Jia propose a method for stereo matching on objects with fractional boundaries [127]. Their method uses stereo image pairs to produce very impressive matting results, but they present no quantitative evaluation of the accuracy of their depth maps. They also formulate alpha estimation as a matting problem, separating the entire scene into one background layer and one foreground layer. With this assumption of two layers, they can handle objects with very large fractional boundaries (i.e. very fuzzy or hairy items), which our method does not. However, they are limited to two depth layers, so the method is not suitable for general scenes, which may have objects evenly distributed across many depths (for example, the Cones data set in the Middlebury stereo evaluation [94]).

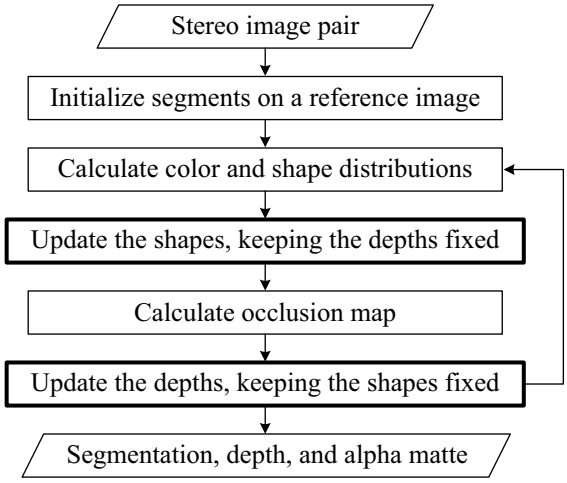


Figure 3.1: Overview of our algorithm.

3.3 Scene Representation and Stereo Image Model

Figure 3.1 shows an overview of our algorithm. The input is two stereo images that are rectified or calibrated with respect to each other. One of these images is considered the reference view, and we represent the scene as a collection of fronto-parallel planar segments in that view's coordinate system. We use an over-segmentation approach and assume that all pixels in each segment have the same depth. Slanted planes are therefore approximated by a set of small segments. The key to our algorithm is alternately updating the shape and depth of these segments. We use a generative model of an image to update the segment shapes based on maximum a posteriori (MAP) estimation. We model stereo constraints as a pairwise Markov random field (MRF) and update the segment depths using belief propagation. The following subsections introduce these models, and Section 3.4 describes our inference methods in detail.

3.3.1 A Generative Model of an Image for Updating Segment Shapes

We model an image as a set of potentially overlapping segments. Our generative model is inspired by Zitnick et al. [148]. In contrast to their work, however, we model stereo constraints. Moreover, we generate only one set of segments for the scene, instead of a segmentation of each input image.

Pixels in the reference image are mapped to segments by segment indices. To handle mixed pixels that commonly occur near segment boundaries, each pixel i is assigned to two segment indices, s_i^f and s_i^b, representing the foreground and background components, respectively. Pixels that do not lie near segment boundaries are captured by the case $s_i^f = s_i^b$.

Each segment is modeled by its depth, color and shape. We assume each segment has a

constant depth and its color is modeled by a Gaussian. A segment's spatial distribution is modeled using both a Gaussian and the set of pixels currently assigned as foreground to the segment. Thus, a segment s is described by the parameters

$$\mathbf{\Phi}_s = (d_s, \boldsymbol{\mu}_s, \mathbf{\Sigma}_s, \boldsymbol{\eta}_s, \mathbf{\Delta}_s, S_s), \tag{3.1}$$

where $(\boldsymbol{\mu}_s, \mathbf{\Sigma}_s)$ and $(\boldsymbol{\eta}_s, \mathbf{\Delta}_s)$ are mean and covariance matrix of the Gaussian distribution for the segment's color and shape, respectively, and d_s is the depth of the segment. S_s is a set of pixels over which segment s is believed to exist as foreground.

We express our generative model in a Bayesian framework and solve for the parameters using MAP estimation. Given the observed color $\boldsymbol{c}_i$ and position $\boldsymbol{x}_i$ of a pixel i, as well as the segment parameters $\mathbf{\Phi}$, we factorize the generative model as follows:

$$\begin{aligned} p(\boldsymbol{c}_i, \boldsymbol{x}_i, \boldsymbol{c}_i^f, \boldsymbol{c}_i^b, \alpha_i, s_i^f, s_i^b | \mathbf{\Phi}) \propto \\ p(\boldsymbol{c}_i | \boldsymbol{c}_i^f, \boldsymbol{c}_i^b, \alpha_i)\, p(\boldsymbol{c}_i^f | s_i^f, \mathbf{\Phi})\, p(\boldsymbol{c}_i^b | s_i^b, \mathbf{\Phi})\, p(\alpha_i) \\ p(\boldsymbol{x}_i | s_i^f, \mathbf{\Phi})\, p(\boldsymbol{x}_i | s_i^b, \mathbf{\Phi})\, p(s_i^f)\, p(s_i^b)\,. \end{aligned} \tag{3.2}$$

We model the first factor of this equation by

$$p(\boldsymbol{c}_i | \boldsymbol{c}_i^f, \boldsymbol{c}_i^b, \alpha_i) = \mathcal{N}(\boldsymbol{c}_i; \alpha_i \boldsymbol{c}_i^f + (1 - \alpha_i)\boldsymbol{c}_i^b, \boldsymbol{\psi}), \tag{3.3}$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \mathbf{\Sigma})$ is the normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$. This equation assumes the observed color of pixel i is generated by a noisy alpha-blending of the segment colors.

Given the segment indices s_i^f and s_i^b, the conditional distributions of the two hidden pixel colors $\boldsymbol{c}_i^f$ and $\boldsymbol{c}_i^b$ are computed using the segments' color models as

$$p(\boldsymbol{c}_i^f | s_i^f, \mathbf{\Phi}) = \mathcal{N}(\boldsymbol{c}_i^f; \boldsymbol{\mu}_{s_i^f}, \mathbf{\Sigma}_{s_i^f}), \tag{3.4}$$

and similarly for $\boldsymbol{c}_i^b$. The prior $p(\alpha_i)$ on α_i is set to be uniform and may be omitted.

The spatial likelihoods for a pixel i given segment indices s_i^f and s_i^b are split as

$$p(\boldsymbol{x}_i | s_i^f, \mathbf{\Phi}) \;=\; p(\boldsymbol{x}_i | s_i^f, \boldsymbol{\eta}_{s_i^f}, \mathbf{\Delta}_{s_i^f})\, p(\boldsymbol{x}_i | s_i^f, S_{s_i^f})\, p(\boldsymbol{x}_i | s_i^f, d_{s_i^f}) \tag{3.5}$$

$$p(\boldsymbol{x}_i | s_i^b, \mathbf{\Phi}) \;=\; p(\boldsymbol{x}_i | s_i^b, \boldsymbol{\eta}_{s_i^b}, \mathbf{\Delta}_{s_i^b})\, p(\boldsymbol{x}_i | s_i^b, S_{s_i^b}). \tag{3.6}$$

The first factor $p(\boldsymbol{x}_i | s_i^f, \boldsymbol{\eta}_{s_i^f}, \mathbf{\Delta}_{s_i^f})$ is equal to the normal distribution $\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\eta}_{s_i^f}, \mathbf{\Delta}_{s_i^f})$, and similarly for $p(\boldsymbol{x}_i | s_i^b, \boldsymbol{\eta}_{s_i^b}, \mathbf{\Delta}_{s_i^b})$. The second factors enforce the constraint that segments should be locally coherent. This is accomplished by favoring segment assignments with strong local support. Specifically, we define them to be proportional to the number of pixels within a small neighborhood ε_i of $\boldsymbol{x}_i$:

$$p(\boldsymbol{x}_i | s_i^f, S_{s_i^f}) \;\propto\; \sum_{j \in \varepsilon_i} h(j, S_{s_i^f}) \tag{3.7}$$

$$p(\boldsymbol{x}_i | s_i^b, S_{s_i^b}) \;\propto\; \sum_{j \in \varepsilon_i} h(j, S_{s_i^b}). \tag{3.8}$$

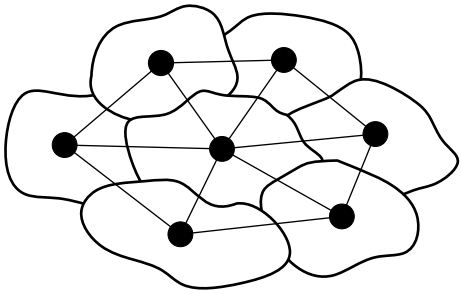


Figure 3.2: Stereo and smoothness constraints on the segment depths are modeled using a pairwise MRF, in which each segment corresponds to a node, and the nodes of neighboring segments are joined by edges.

The value of $h(j, S_s)$ is one if pixel j is a member of S_s and zero otherwise. Note that the background segment index is only influenced by the current assignment of the foreground segment index in the neighborhood ε_i. This constraint limits the extent of mixed pixels near the segment boundaries.

The final factor $p(\boldsymbol{x}_i|s_i^f, d_{s_i^f})$, only affected by the foreground segment index, ensures that the stereo matching cost for a pixel assigned to depth $d_{s_i^f}$ is small:

$$p(\boldsymbol{x}_i|s_i^f, d_{s_i^f}) \propto \exp(-C(i, d_{s_i^f})). \tag{3.9}$$

Here, $C(i, d_{s_i^f})$ is the matching cost described in detail in the next subsection. This formulation ensures that the pixel should belong to a segment whose estimated depth $d_{s_i^f}$ is likely given the depth probability distribution of the pixel.

We assume the segment priors are uniform. As a result $p(s_i^f)$ and $p(s_i^b) = \frac{1}{M}$, where M is the number of segments. Since $p(s_i^f)$ and $p(s_i^b)$ are uniform, they may be omitted when computing the MAP estimate.

3.3.2 Stereo Constraints for Updating Segment Depths

We model stereo and smoothness constraints using a pairwise MRF of segments, as shown in Fig. 3.2. Each node corresponds to a segment s and shares edges with neighboring segments t for $(s, t) \in N$, where N is the set of all adjacent segments. Here we define segments using only the foreground segment indices; i.e. the background segment indices are ignored in the depth update step. The state of each node is its corresponding segment's depth, so the number of states for each node is equal to the number of depth levels. Zitnick et al. [149] use this formulation with a fixed segmentation. Our algorithm, by contrast, updates the topology of the MRF in each iteration using the updated segment shapes.

We measure the quality of a depth assignment for the MRF using the following energy function:

$$E = \sum_{s} D_s(d_s) + \sum_{(s,t)\in N} V(d_s, d_t). \tag{3.10}$$

Here, $D_s(d_s)$ is the cost (commonly called the "data cost") of assigning a depth d_s to a segment s. This cost accounts for the matching of visible pixels between stereo views and a penalty for occluded regions. The term $V(d_s, d_t)$ is a cost (the "discontinuity cost") that penalizes depth assignments for neighboring segments s and t that violate a smoothness assumption.

Data Cost

We define the matching cost of a visible pixel i to be

$$C(i, d_s) = \rho_d(F(\boldsymbol{x}_i, d_s)). \tag{3.11}$$

The function F measures the intensity similarity between the pixel in the reference image whose coordinate is $\boldsymbol{x}_i$ and the pixel projected to the other image with a depth value d_s. For this function, we use Birchfield and Tomasi's pixel dissimilarity measure [10], which is insensitive to image sampling. The function ρ_d is an error function that is robust to outliers due to noise, occlusions, specularities, and so on. We use a truncated L1 norm [104, 105]

$$\rho_d(x) = -\ln((1 - e_d)\exp(-|x|/\sigma_d) + e_d), \tag{3.12}$$

where the parameters σ_d and e_d control the shape of the function.

Even with robust similarity measures, it is important to explicitly identify occluded pixels in the reference view so the algorithm does not match occluded regions in one view with pixels in the other. We incorporate an occlusion penalty in the data cost. We use a formulation that is similar to ones used in other segmentation-based stereo works [11, 121]. Before calculating the data cost for each iteration, we create an occlusion map by warping all of the pixels in the reference view to the non-reference view using currently estimated segment depths. The warped pixel depths (in the non-reference view coordinate system) are stored at the projected pixel coordinates. If more than one pixel from the reference view project to the same image coordinates in the non-reference views, they are sorted in depth order.

When calculating the data cost for each pixel, we project the pixel into the non-reference view and check its depth against the occlusion map. We distinguish the following three visibility cases: (a) the projected pixel is visible and occludes no other pixels; (b) the projected pixel is occluded by another pixel; and (c) the projected pixel is visible, but occludes another pixel. For each pixel,

the data cost $\bar{C}(i, d_s)$ for each visibility case is given by

$$\bar{C}(i, d_s) = \begin{cases} C(i, d_s) & : \text{case (a)} \\ \lambda_{occ} & : \text{case (b)} \\ C(i, d_s) + \lambda_{occ} - C(j, d'_s) & : \text{case (c)} \end{cases}. \tag{3.13}$$

For case (a), the pixel data cost is simply the matching cost. For case (b), an occluded pixel, the data cost is λ_{occ}, a positive constant that slightly penalizes occluded pixels. For case (c), the data cost favors low matching costs for the projected pixel, penalizes occlusions, and discourages occluding other pixels with low matching costs. $C(j, d'_s)$ is the matching cost of the occluded pixel j with (previously estimated) depth d'_s.

The data term of each segment is the sum of the matching costs of the pixels in the segment:

$$D_s(d_s) = \sum_{i \in s} \bar{C}(i, d_s). \tag{3.14}$$

Discontinuity Cost

Like many other stereo methods, ours assumes that depth varies smoothly almost everywhere, except at object boundaries. We also assume that neighboring segments with similar colors are likely to have similar depths. Moreover, the larger the shared boundary between two segments, the stronger the discontinuity penalty should be. We express this discontinuity cost using a truncated L2 norm of depth difference of neighboring segments:

$$V(d_s, d_t) = \lambda_{disc}\, b_{st}\, \min((d_s - d_t)^2, T_{st}). \tag{3.15}$$

The parameter λ_{disc} is a positive constant, b_{st} is the number of pixels on the boundary between segments s and t, and T_{st} is the truncation point for the L2 norm function. For each neighboring pair of segments, the truncation point is set such that pairs with large color differences have a small impact on the discontinuity cost, and pairs with small differences have a large impact. We use

$$T_{st} = \max(T_{max} \exp(-||\boldsymbol{\mu}_s - \boldsymbol{\mu}_t||^2 / 2\sigma_c^2), T_{min}), \tag{3.16}$$

where $\boldsymbol{\mu}_s$ and $\boldsymbol{\mu}_t$ are the mean colors of segments s and t. The parameter σ_c controls the influence of the segments' color difference. T_{min} is chosen to be a small value to ensure that each segment has at least some influence on its neighboring segments.

3.4 Inference Procedure

This section describes the details of our inference process using the models introduced in the previous section. For the initial segmentation, we use a mean-shift segmentation method [22]

with default parameters. The resulting large segments are partitioned into a grid of 8×8 pixels, because our method is based on over-segmentation. The initial depth of each segment is estimated using max-product belief propagation on our stereo MRF model. Because we have no occlusion information for this initial step, the message update order can strongly impact the inference. We use a synchronous update schedule [112], in which the messages are only updated at the end of each iteration, to ensure that the message update order does not affect the inference.

Next, we alternate between updating segment shapes and segment depths. To update segment shapes, we must find parameters which maximize the probability in Eq. (3.2) for each pixel i. To do this, we first choose candidate segment indices $(\hat{s}_i^f, \hat{s}_i^b)$ for the pixel based on the current estimate of segment assignments S and the constraints in Eqs. (3.7) and (3.8). The candidate segment indices can be any pair of two segments found within the neighborhood ε_i of pixel i. For each index pair $(\hat{s}_i^f, \hat{s}_i^b)$, we approximate alpha using the estimated segment colors $\boldsymbol{\mu}_s$ as

$$\hat{\alpha}_i = \frac{(\boldsymbol{c}_i - \boldsymbol{\mu}_{\hat{s}_i^b}) \cdot (\boldsymbol{\mu}_{\hat{s}_i^f} - \boldsymbol{\mu}_{\hat{s}_i^b})}{||\boldsymbol{\mu}_{\hat{s}_i^f} - \boldsymbol{\mu}_{\hat{s}_i^b}||^2}. \tag{3.17}$$

We use another approximation that the background color is same as the background segment color, i.e. $\hat{\boldsymbol{c}}_i^b = \boldsymbol{\mu}_{\hat{s}_i^b}$. Given $\hat{\boldsymbol{c}}_i^b$ and $\hat{\alpha}_i$, we can compute the foreground color using the alpha matting equation. Finally, we choose the parameters that maximize the probability of Eq. (3.2). For each iteration, to roughly estimate $\boldsymbol{\psi}$, the color noise covariance matrix, we compute and average the noise covariance matrices of all of the segments. If at any point a segment becomes too small (assigned to fewer than 12 pixels), it is discarded from the segmentation map, and the pixels within that segment are merged into the neighboring segments.

After each shape update step, we recalculate the occlusion map with the newly estimated segment depths. We then use belief propagation to update the segment depths. In contrast to the initial step, we now have occlusion information (based on previously estimated depths), so we can use an accelerated update schedule in which updated messages are immediately used to calculate the messages of neighboring segments. This scheme makes the inference fast, even with many segments. In our experiments, we only need two message propagation steps in each depth update step. We store the messages at the end of each depth update step, and use them for the initial messages in the next iteration. Whenever a new edge appears due to segment shape updates, the message for that edge is initialized to zero.

3.5 Experiments

In this section, we evaluate our method with the following experiments. First, we show the accuracy of our stereo algorithm using the new (second version) Middlebury stereo evaluation [94]. Next, we present the robustness of our adaptive segmentation method. Our method recovers from

Table 3.1: Parameters used for the new Middlebury stereo evaluation.

ε_i	λ_{occ}	λ_{disc}	e_d	σ_d	σ_c	T_{max}	T_{min}
5×5	1.1	0.1	0.01	4.0	12.0	64.0	0.9

Table 3.2: Results on the new Middlebury stereo evaluation [94], comparing the percentage of "bad pixels" in non-occluded regions ($R_{\bar{O}}$), all regions except for unknown pixels (R_A), and regions near depth discontinuities (R_D). The best result in each column is in bold print. Subscript numbers for our method are the relative ranks in each column. The average rank of our algorithm is fourth best on the evaluation as of December 2007.

Algorithm	Tsukuba			Venus		
	$R_{\bar{O}}$	R_A	R_D	$R_{\bar{O}}$	R_A	R_D
AdaptingBP [52]	1.11	1.37	5.79	**0.10**	0.21	1.44
DoubleBP [132]	**0.88**	**1.29**	**4.76**	0.14	0.60	2.00
SubPixDoubleBP [133]	1.24	1.76	5.98	0.12	0.46	1.74
Ours ($\alpha_{th} = 0.0$)	1.52 $_{16}$	1.93 $_{13}$	4.77 $_{2}$	0.11 $_{2}$	0.22 $_{2}$	**1.07** $_{1}$
Ours ($\alpha_{th} = 0.5$)	1.69 $_{17}$	2.04 $_{16}$	5.64 $_{4}$	0.14 $_{3}$	**0.20** $_{1}$	1.47 $_{2}$
SymBP+occ [104]	0.97	1.75	5.09	0.16	0.33	2.19
SO+borders [72]	1.29	1.71	6.83	0.25	0.53	2.26
Segm+visib [11]	1.30	1.57	6.92	0.79	1.06	6.76
Algorithm	Teddy			Cones		
	$R_{\bar{O}}$	R_A	R_D	$R_{\bar{O}}$	R_A	R_D
AdaptingBP [52]	4.22	7.06	11.8	**2.48**	7.92	**7.32**
DoubleBP [132]	3.55	8.71	**9.70**	2.90	9.24	7.80
SubPixDoubleBP [133]	**3.45**	8.38	10.0	2.93	8.73	7.91
Ours ($\alpha_{th} = 0.0$)	7.10 $_{11}$	11.3 $_{6}$	16.6 $_{10}$	3.75 $_{10}$	9.21 $_{8}$	9.28 $_{11}$
Ours ($\alpha_{th} = 0.5$)	7.04 $_{11}$	11.1 $_{6}$	16.4 $_{9}$	3.60 $_{9}$	8.96 $_{8}$	8.84 $_{9}$
SymBP+occ [104]	6.47	10.7	17.0	4.79	10.7	10.9
SO+borders [72]	7.02	12.2	16.3	3.90	9.85	10.2
Segm+visib [11]	5.00	**6.54**	12.3	3.72	8.62	10.2

initial segmentation errors by updating segment shapes, and performs well for the Map image pair, which is known to be difficult for fixed segmentation methods. Finally, we show a Z-keying example to demonstrate the quality of our alpha matting results.

3.5.1 Stereo Reconstruction Accuracy

The error metric for the Middlebury stereo evaluation is the percentage of "bad pixels" (pixels for which the absolute disparity error is greater than 1 pixel) in the following three regions: non-occluded regions ($R_{\bar{O}}$), all regions except for unknown pixels (R_A), and regions near depth discontinuities (R_D). We used the same parameters, shown in Table 3.1, for all stereo pairs. We discretized the disparity space with an interval of 0.5 pixels, and performed 20 iterations of shape

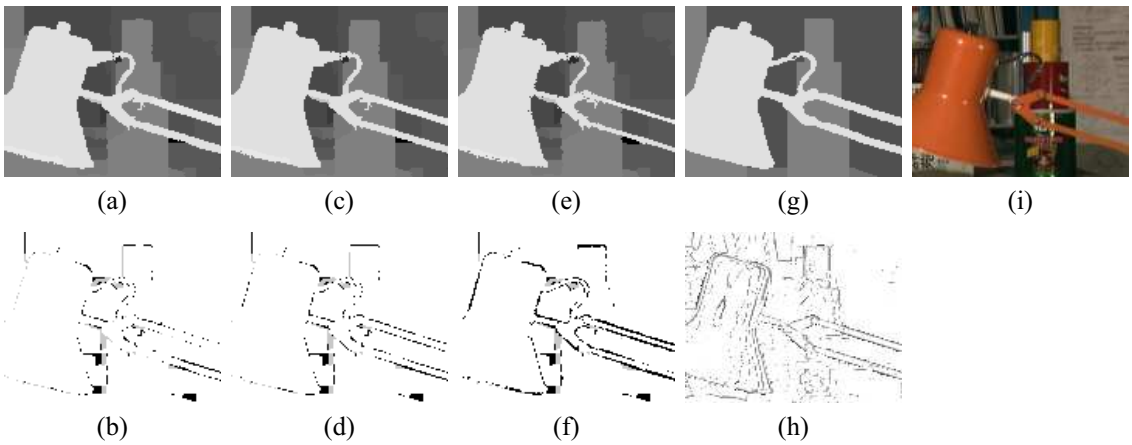


Figure 3.3: Depth maps and bad pixels for an inset of the Tsukuba data set, for different alpha thresholds: (a, b) $\alpha_{th} = 0.0$, (c, d) $\alpha_{th} = 0.5$, and (e, f) $\alpha_{th} = 1.0$. (g) Ground truth. (h) Estimated alpha matte. (i) Original left image. According to the evaluation ground truth data, best results are obtained with $\alpha_{th} = 0.0$. Although visually the ground truth depth map (g) matches our $\alpha_{th} = 0.0$ depth map (a), the actual left Tsukuba input image (i) seems to more closely resemble our $\alpha_{th} = 0.5$ depth map.

and depth updates. The running times were about 90 seconds for the Tsukuba data set (384×288 pixels, 31 depth levels) and 20 minutes for the Cones data set (450×375 pixels, 119 depth levels) on a 3.2 GHz PC.

Since the foreground and background segment indices for mixed pixels at segment boundaries differ, we have two different depth values for those pixels. The Middlebury evaluation, however, requires a single-valued depth map (one with one depth value per pixel). We use a threshold α_{th} to select a depth value for mixed pixels: for pixel i with $s_i^f \neq s_i^b$, if $\alpha_i \geq \alpha_{th}$ then select $d_{s_i^f}$, otherwise use $d_{s_i^b}$.

Table 3.2 summarizes the results of our method with two fixed thresholds (0.0 and 0.5) for all data sets, compared with the other state-of-the-art methods. Figure 3.3 shows depth maps and their "bad pixels" (shown as black for non-occluded regions and gray for occluded regions) using different alpha thresholds, for an inset from the Tsukuba image. The table and figure show that for the evaluation, the best threshold differs for different data sets. In particular, our results suggest that the Tsukuba depth map is biased toward foreground depth values for mixed pixels. This is confirmed by the insets in Fig. 3.3; the left Tsukuba input image more closely resembles our $(\alpha_{th} = 0.5)$ depth map than the $(\alpha_{th} = 0.0)$ one. Using a fixed alpha threshold value of 0.5 for all stereo pairs, the average rank of our algorithm is the fourth best in the Middlebury evaluation as of December 2007.

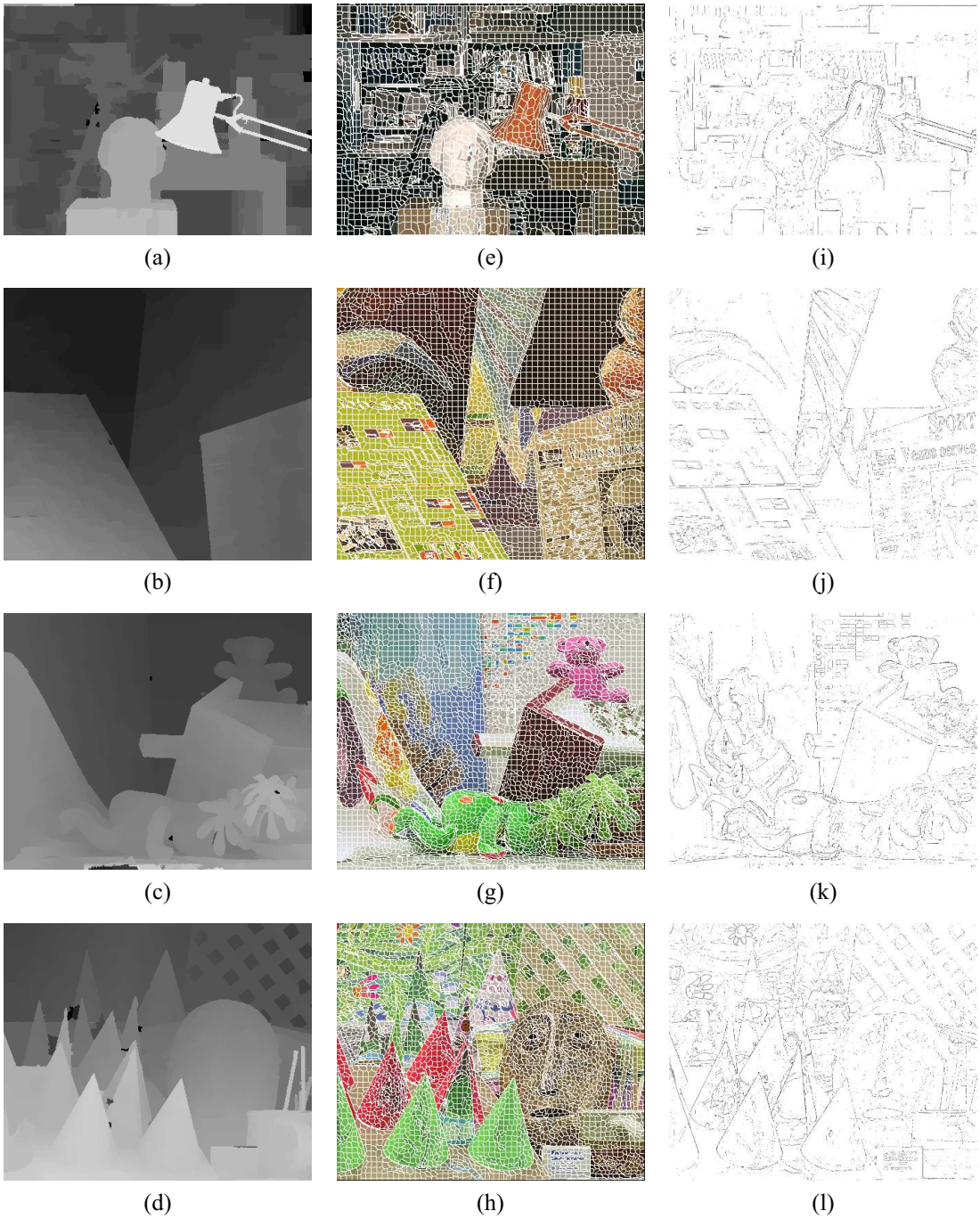
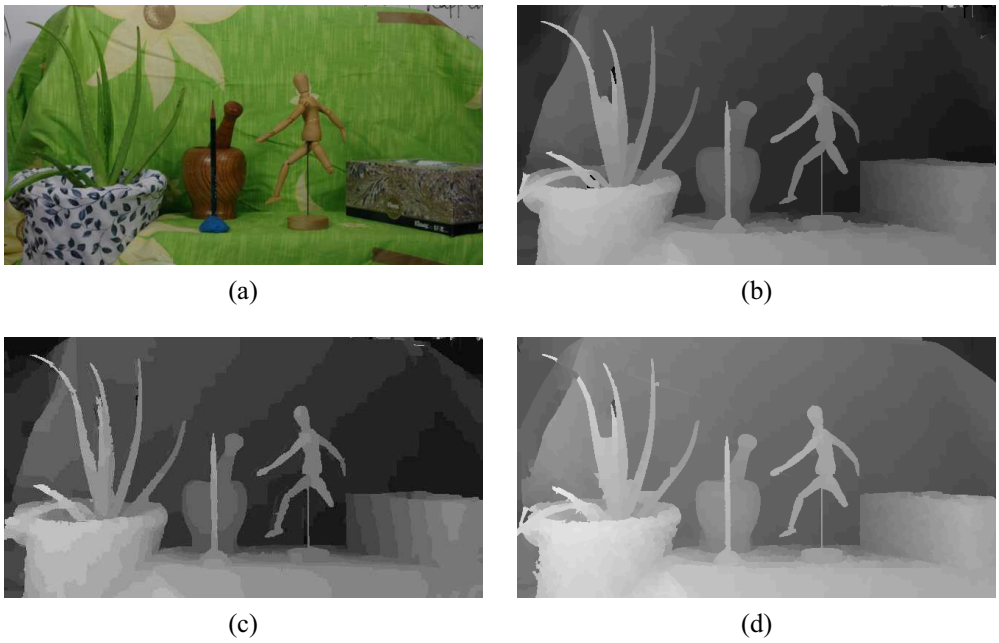


Figure 3.4: Output images. (a–d) Depth map ($\alpha_{th} = 0.5$), (e–h) segmentation map, and (i–l) alpha matte for each data set.



(a) (b) (c) (d)

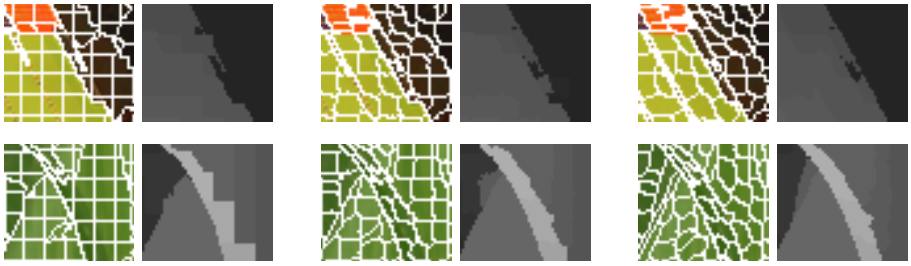
Figure 3.5: Result from Puppet image pair. (a) Original left image. (b) Our depth map ($\alpha_{th} = 0.5$). (c) Sun et al. [104]. (d) Zitnick and Kang [149].

The final depth maps, segmentation maps, and alpha mattes for the Middlebury image pairs are shown in Fig. 3.4. Sharp object boundaries are recovered for all four data sets. Although slanted planes are approximated well with small segments of constant depth, our method fails for heavily slanted planes, such as the floor in the Teddy data set. Figure 3.5 depicts another result obtained using the Puppet image pair from [149], which is comparable to the results in [149].

3.5.2 Robustness of Adaptive Over-Segmentation

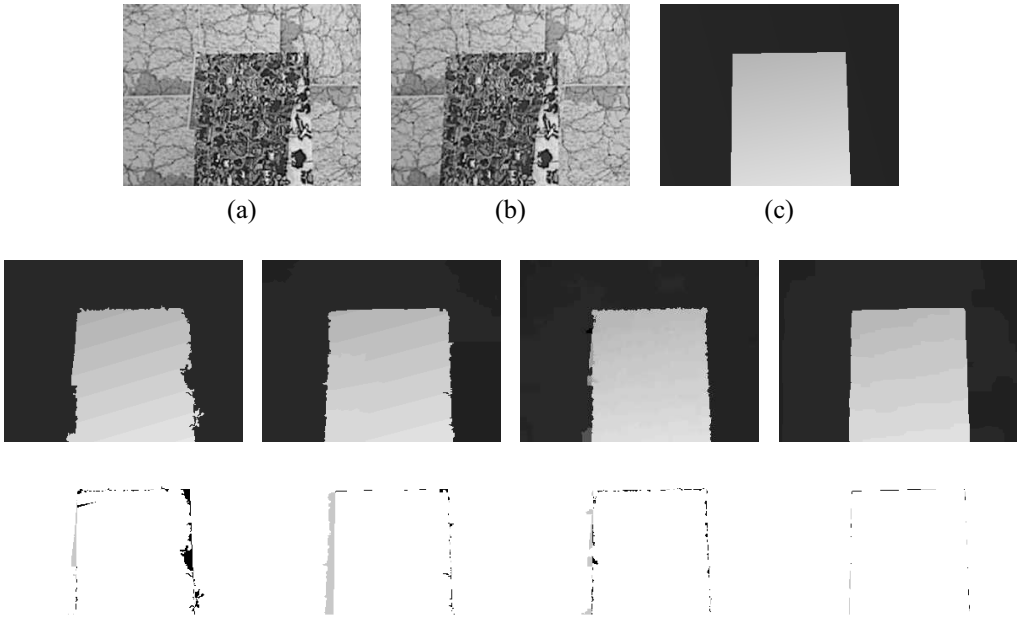
Figure 3.6 shows close-up views of the segmentation and depth maps at different iterations and demonstrates the robustness of our adaptive over-segmentation. The mean-shift segmentation method (with default parameters) [22] labels objects at different depths as one segment due to their similar colors (Fig. 3.6 (a)), causing errors for methods that use fixed segmentations. Our method, by contrast, recovers from these errors, producing better depth maps (Figs. 3.6 (b) and (c)).

Figure 3.7 shows stereo reconstruction results for the Map data set from the old Middlebury stereo evaluation. This data set is difficult for typical segmentation-based methods [11, 43, 121, 149], because color segmentation fails at object boundaries with similar foreground and background colors. For example, Fig. 3.7 (d) shows the results from Hong and Chen's method [43],



(a) (b) (c)

Figure 3.6: Close-up views (The right edge of the left plane in the Venus image (top) and the left leaf in the Puppet image (bottom)) of segmentation and depth maps at (a) initial step, (b) after 2 iterations, and (c) after 10 iterations. Our method recovers from initial segmentation errors, where objects at different depths are labeled as one segment, although there are still small regions that have wrong depth values.



(a) (b) (c)

(d) (e) (f) (g)

Figure 3.7: Results for the Map image pair, known to be difficult for segmentation based stereo methods. (a) Left input image. (b) Right input image. (c) Ground truth depth map. (d–g) Computed depth maps (middle row) and their bad pixels (bottom row). (d) Hong and Chen [43]. (e) Deng et al. [27]. (f) Our method ($\alpha_{th} = 0.5$). (g) Sun et al. [104].

(a) (b) (c)

Figure 3.8: Z-keying example. (a) The teddy bear is extracted from the Teddy data set, and composited to the Cones data set. (b) Close-up views of rectangles in (a). (c) Close-up views of the result with a single depth map ($\alpha_{th} = 0.5$) and no alpha matte.

a color-segmentation based algorithm that ranked third on the old Middlebury evaluation. Deng et al.'s patch-based approach [27] overcomes many of these errors, as shown in Fig. 3.7 (e). (Deng et al. fill occluded regions with neighboring depth values because these regions were not considered in the old Middlebury evaluation.) Our result is similar in quality to Deng et al.'s (Fig. 3.7 (f)). Currently, Sun et al. [104] obtain the best results for this image pair by using segmentation as a soft constraint (Fig. 3.7 (g)).

3.5.3 Z-Keying

Figure 3.8 shows a Z-keying result using estimated depth maps and alpha mattes for the Teddy and Cones image pairs. We extracted the teddy bear from the left Teddy image and composited it into the left Cones image. Because we use alpha mattes for both extraction and composition, there is no color bleeding on boundaries between the teddy bears and other objects (Figs. 3.8 (a) and (b)). By comparison, the matting results using a single depth map (calculated with $\alpha_{th} = 0.5$) and no alpha matte (Fig. 3.8 (c)) have artifacts.

3.6 Discussion and Conclusions

Our adaptive over-segmentation based stereo algorithm overcomes limitations of traditional segmentation based methods while properly handling mixed pixels on object boundaries. Our depth maps are not only accurate according to accepted standards (Middlebury) but in fact more complete, because we produce opacity information and foreground/background colors and depths for mixed pixels. In contrast to most matting methods, we produce this information along depth discontinuities throughout the scene, not only for foreground objects. Currently, the most significant limitation of our method is that it assumes a constant depth for all pixels in each segment, so it does not handle heavily slanted planes well. In future work, we could attempt to address this problem by using oriented planes or parametric surfaces instead of fronto-parallel segments.

To compare our stereo results with other researchers, we create single-valued depth maps to use with the Middlebury stereo evaluation. In doing so, we discovered that the Tsukuba ground truth depth map is biased toward the foreground depths of mixed pixels. Our performance on the Middlebury evaluation gives us good confidence in our depth reconstruction, but it does not fully evaluate the quality of our matting results. Computing depth and matting information is clearly important for applications like view interpolation and Z-keying. In the future, we believe it would be useful to create a new stereo evaluation with ground truth opacities, and foreground/background colors and depths.

Chapter 4

View-Dependent Light Field Coding Using Image-Based Rendering

4.1 Introduction

Image-based rendering (IBR) techniques have attracted a lot of research interest, since they have a great potential for synthesizing photorealistic 3D scenes. IBR data sets, such as light fields [39, 59], are often constructed from multi-view images captured with an array of cameras or lenslets. Since hundreds or thousands of images are necessary for high-quality rendering, efficient coding schemes are required to transmit or store such a large amount of image data.

A number of light field compression techniques have been developed, as we reviewed in Section 2.2. The techniques are commonly designed to compress light field data uniformly; therefore, the ease of random access and the reconstruction quality of synthesized images are approximately constant regardless of the viewpoint. However, there are many applications in which a certain viewpoint image is significant and required fast decoding. For example, as shown in Fig. 4.1 (a), when we transmit light fields for heterogeneous clients, some of them require only a 2D view, while the others require all the light field data, depending on their bandwidth, computational power, and display devices. In this scenario, the sender can choose the significant viewpoint at which a representative view of the 3D scene can be generated. Figure 4.1 (b) shows another interesting scenario. When we interactively browse synthesized views over a network, the data for the current view is more important than the data for the other views. If the current viewpoint image is only transmitted, however, we can not change the viewpoint immediately due to the network latency. The coding method presented in this chapter, as well as another approach presented in Chapter 5, provides a functionality suitable for these scenarios.

We present a scalable light field coder that performs image-based rendering before the encoding process. We call it view-dependent coder, because it places priority on a given significant viewpoint, which we call *representative viewpoint*. The coder first synthesizes an image at the

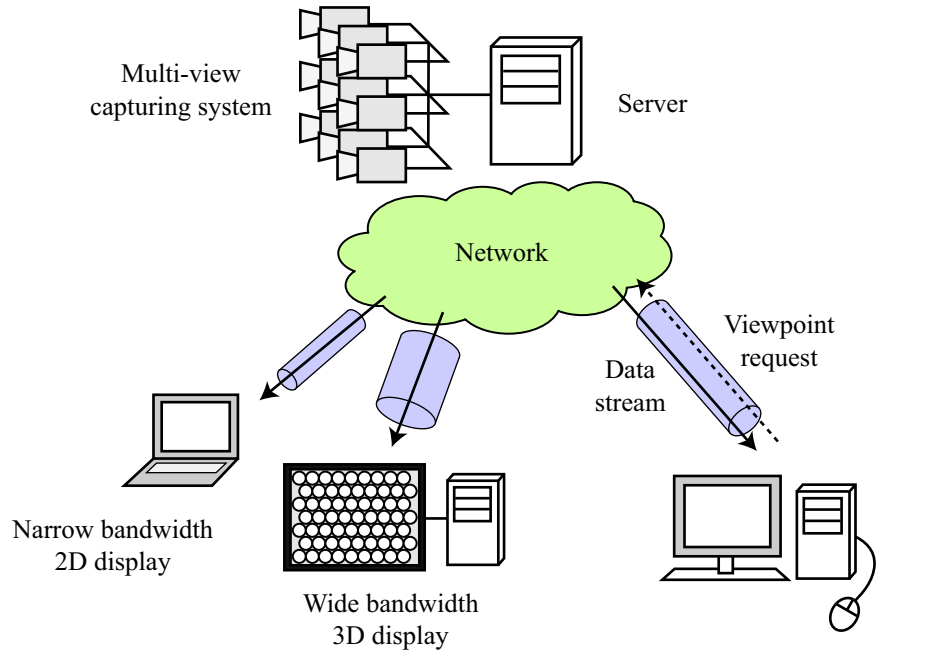


Figure 4.1: Applications of our coding method.

representative viewpoint and then encodes the input multi-view images by using the synthesized image as a reference image. It produces a scalable bitstream, which has 2D compatibility (the representative viewpoint image is decodable as a 2D image) and can be used with three rendering methods depending on the bit rate. The scalable bitstream enables us to render high-quality views around the representative viewpoint even at low bit rates, and to improve the quality of views away from the viewpoint with increasing bit rate. We call this novel scalability view-dependent scalability. Our experimental results show that the coder also provides good coding efficiency for both multi-camera images and integral photography images.

4.2 View-Dependent Coder

4.2.1 Coding Procedure

Figure 4.2 shows a block diagram of our view-dependent coder. The coder first synthesizes an image at a representative viewpoint using an image-based rendering method. A view-dependent geometry model is estimated in the rendering process to produce a high-quality image. The coder

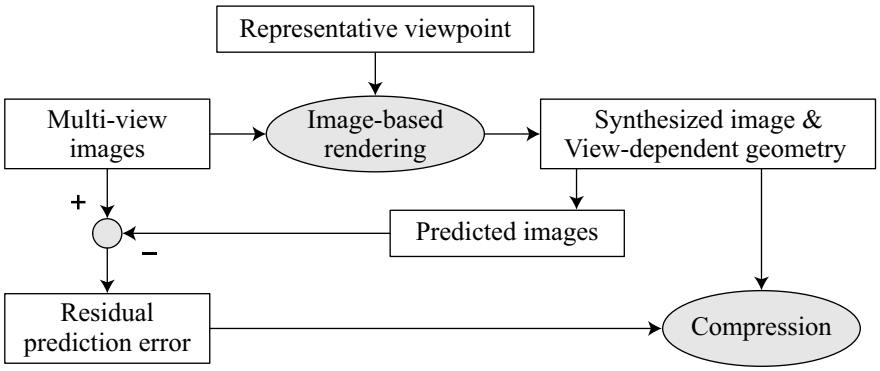


Figure 4.2: Block diagram of our coding scheme.

then predicts all of the input images by using the synthesized image and the estimated geometry model. The residual prediction error is generated if the quality of the predicted image is not sufficient. Finally, the synthesized image, view-dependent geometry model, and residual prediction errors are compressed and stored into a hierarchical bitstream shown in Fig. 4.3.

This prediction process is similar to that used in the model-aided predictive coding [66, 70], but we take a novel approach that uses a synthesized image at an arbitrary viewpoint as the reference image. Thus this coder provides both direct access to the representative viewpoint image, and good coding efficiency by using a predictive method with a view-dependent geometry model.

4.2.2 Hierarchical Bitstream

Our coder produces a hierarchical bitstream shown in Fig. 4.3. This bitstream can be used with three rendering methods depending on the bit rate or decoding time as follows. The layer 1 is a synthesized image at a representative viewpoint. It acts as a *thumbnail* of the light field because we can see an overview of the 3D scene. The layer 2 includes a view-dependent geometry model. Using layers 1 and 2, we can synthesize novel views by model-based rendering techniques. However, the quality of views away from the representative viewpoint would not be high enough due to the occlusions and geometry errors. The residual information is stored in the layer 3. Using all the layers, we can render high-quality views by reconstructing the light field data and using an image-based rendering method.

In this way, our coder provides the view-dependent scalability. When the bit rate of the residual data in the layer 3 increases, the quality of views away from the representative viewpoint improves. The coder also has the compatibility with conventional 2D image formats by using the layer 1 image as a base image and the data of the layers 2 and 3 as its extension information. We show an implementation of JPEG-compatible bitstreams in Section 4.4.1.

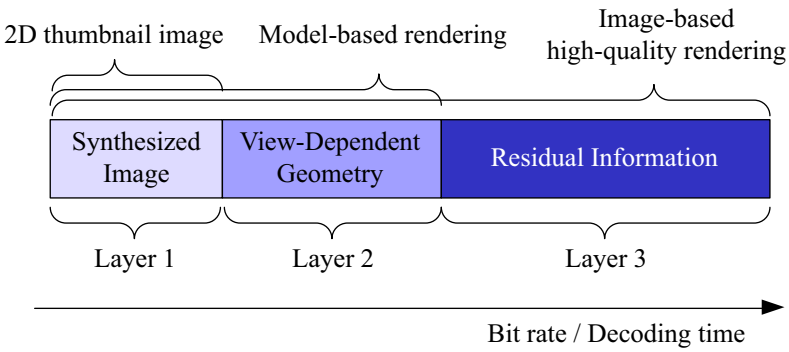


Figure 4.3: Hierarchical structure of the encoded bitstream.

4.2.3 Evaluation Method

The rate-distortion performance of light field coding methods is typically evaluated by reconstructing the input multi-view images and measuring their quality with respect to the original images. This means that the quality of the synthesized images is not evaluated directly. Since our coder is designed to provide view-dependent reconstruction quality of synthesized images, it is essential to evaluate the quality of synthesized images themselves, rather than the quality of the input multi-view images.

Figure 4.4 shows our evaluation method for the synthesized images. To show the view-dependent reconstruction quality of our coder, we evaluate the quality of synthesized images with the different distance of the rendering viewpoint from the representative viewpoint, denoted as r in Fig. 4.4. At multiple viewpoints with a fixed distance r from the representative viewpoint, we synthesize images from the original and compressed light fields using the same rendering method, and measure the reconstruction quality by comparing these two images. The reconstruction quality decreases when the distance r becomes longer at low bit rates, while it improves at high bit rates.

4.3 Implementation

We implemented the view-dependent coder for two different types of light fields: multi-camera images and integral photography images. Figure 4.5 shows examples of them, which we use for measuring the coding performance of our method.

Our coder first renders a novel image with estimating a view-dependent geometry model of the scene at the representative viewpoint. For the rendering, we used the rendering method described in Section 2.1.4 for multi-camera images, and a method presented by Mitsuda et al. [79] for integral photography images. The rendered image is once encoded using a standard block-DCT

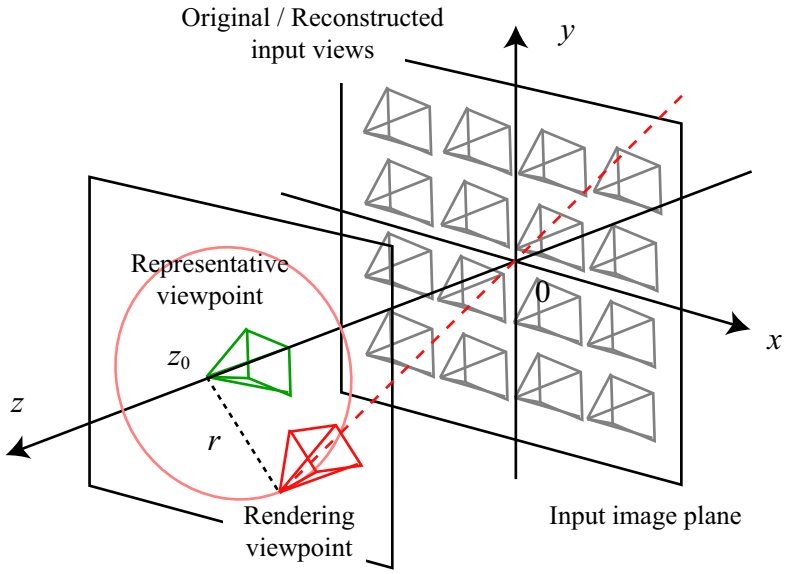


Figure 4.4: Evaluation method for synthesized images.

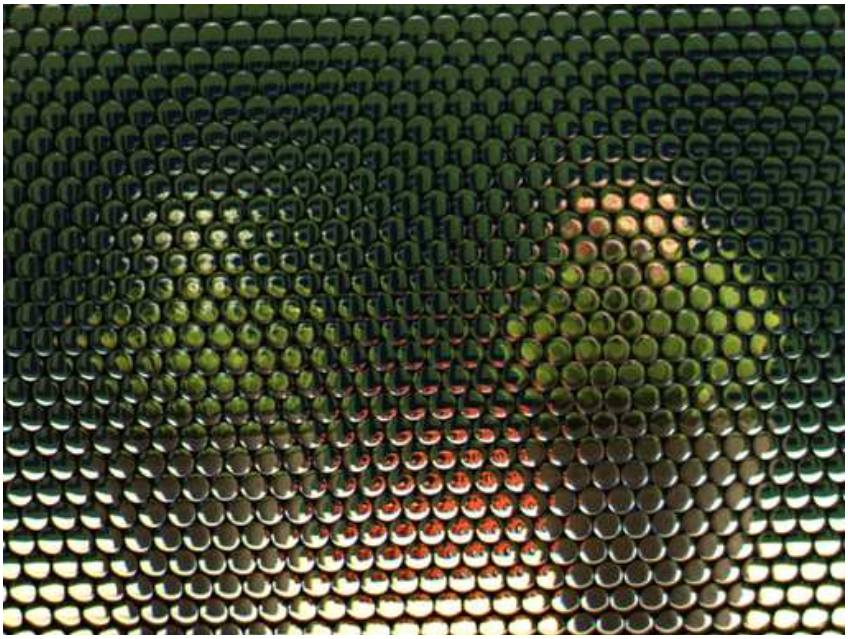
scheme, and then locally decoded to be used as a reference image. The input multi-view images are predicted by warping the reference image with the geometry model. Since the prediction accuracy varies widely, a coding mode is selected for each macroblock of 16 $\times$ 16 pixels from the following modes:

- Only predicted
- Predicted and residual coded
- Intra-coded

If a predicted macroblock meets a preset minimum reconstruction quality q_{min}, the only predicted mode is selected and no further information for this macroblock is encoded. Otherwise, the coder compares a cost value between the residual prediction error and the original input macroblock, and decides which should be coded. The intra-coded mode is selected if the cost value of the original image is better than that of the residual error. This decision process for the latter two modes is equivalent to that used in MPEG-2 Test Model 5 [154].

Finally, the residual errors are encoded using a block-DCT scheme as well as the synthesized reference image. The geometry model is losslessly compressed with a DPCM method. In the experiments described in Section 4.5, those bits are taken into account for calculating the bit rate of our coder.

(a) *Santa*



(b) *Car and Flowers*

Figure 4.5: Input multi-view image sets. (a) *Santa* is a multi-camera image set consisting of 9×9 views of 640×480 pixels. (b) *Car and Flowers* is an integral photography image consisting of 26×23 views of 31×31 pixels.

4.4 Applications

The hierarchical bitstream produced by our coder has compatibility with conventional 2D image formats, and is available for three rendering methods depending on the bit rate. This section describes the applications and rendering results using these characteristics of our coder.

4.4.1 JPEG-Compatible Bitstream

Since a 2D representative view is located at the head of the hierarchical bitstream, the bitstream has compatibility with conventional 2D image formats. Typical 2D image formats have application data segments; for example, in the JPEG standard, the APP_n segments are reserved for application use [151]. We use the segments to produce a JPEG-compatible bitstream from the output of our coder.

Here we encoded the representative image of the layer 1 as a JPEG image, and stored the data of the layers 2 and 3 into the application data segments of the JPEG image, as shown in Table 4.1. As shown in Fig. 4.6, we can see a 2D thumbnail image of a 3D scene by opening the JPEG file with common JPEG viewers. Meanwhile, our specialized viewer enables us to generate free-viewpoint images by using all of the data stored in the same file.

4.4.2 Synthesized Images Obtained Using Different Rendering Methods

Figure 4.7 shows synthesized images obtained using different rendering methods. If the data in the layers 1 and 2 are available, we can render novel views using texture-mapped polygon rendering, as shown in Fig. 4.7 (a). This rendering method produces good-looking images when the rendering viewpoint is close to the representative viewpoint, but the rendering quality decreases when the rendering viewpoint is away from the representative viewpoint due to the occlusions and geometry errors. Figure 4.7 (b) shows the effect of occlusions and geometry error more clearly by rendering these images as point clouds. On the other hand, Fig. 4.7 (c) shows images rendered by using all the layers. In this case, we first reconstruct input multi-view images and then render the images using the rendering method that is used for generating the representative image. This rendering method produces high-quality images regardless of the position of the viewpoint.

4.5 Experiments

We evaluate the performance of our coder using two different image sets shown in Fig. 4.5: (a) *Santa* and (b) *Car and Flowers*, which are examples of multi-camera images and integral photography, respectively. The *Santa* image set, which is from the multi-view image database provided by University of Tsukuba, Japan, consists of 81 (9×9) images of 640×480 pixels.

Table 4.1: Contents in application data segments of the JPEG image.

Marker	Contents
APP_1	Geometry data
APP_2	Macroblock mode information
APP_3	Elemental image parameters (e.g., width and height)
APP_4	Quantization tables
APP_5	Huffman tables
APP_6	Encoded elemental image data

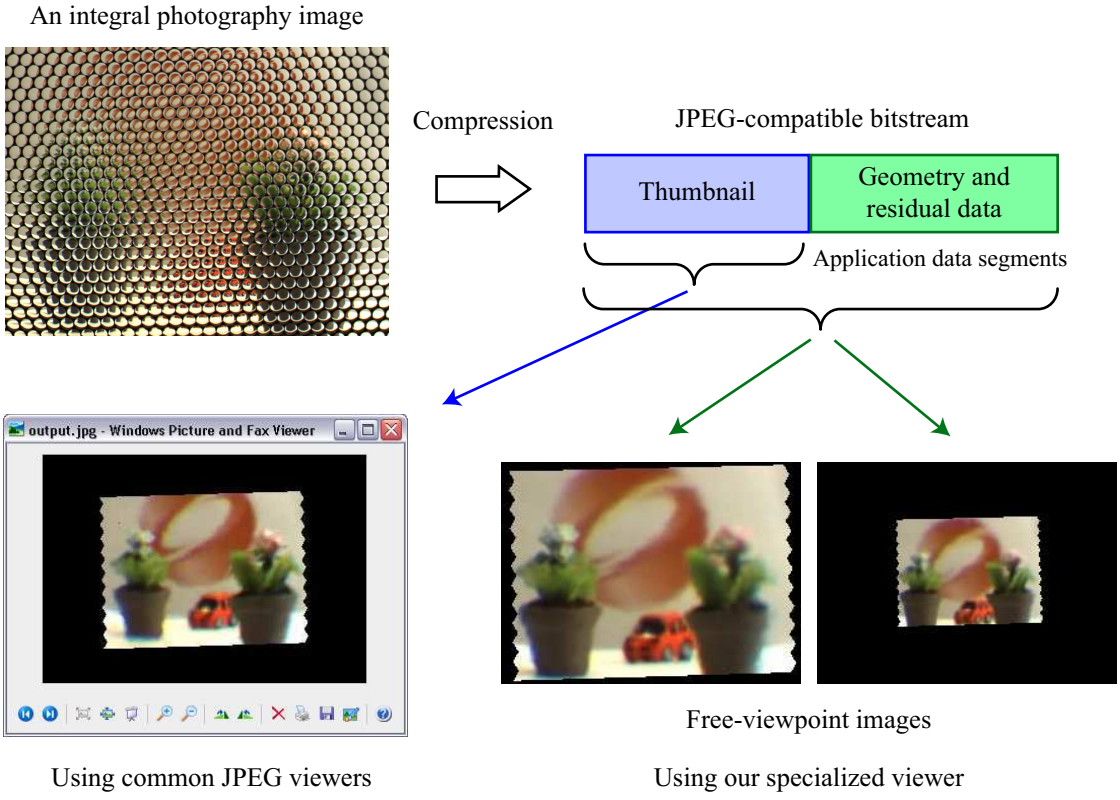
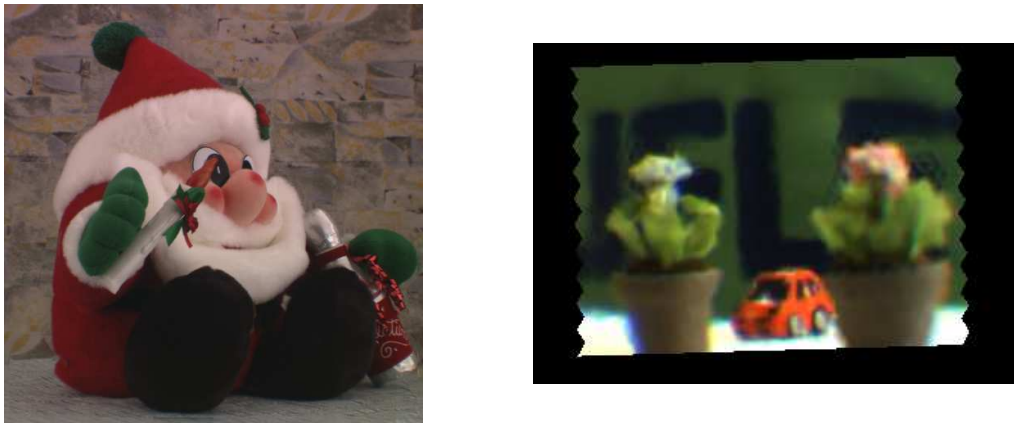


Figure 4.6: Applications using the JPEG compatible bitstream.

Figure 4.7: Synthesized images obtained using different rendering methods. Images in the same row were rendered at the same viewpoint.



(a) *Santa* (b) *Car and Flowers*

Figure 4.8: Synthesized images at the representative viewpoint.

The recording camera positions lie in a plane and are arranged in a regular 2D grid. On the other hand, the *Car and Flowers* image set is created from an integral photography image captured with a real-time video-based rendering system named LIFLET [129, 130], which employs an array of lenslets and an XGA camera. An integral photography image consists of a set of small circle images, and each image is called an elemental image. We used 598 (26×23) elemental images of 31×31 pixels as the image set. Invalid pixels among the elemental images (i.e., the exterior portion of the circular region) were padded with the nearest valid pixel color in order to reduce the high-frequency components. The representative viewpoint was set behind the center of the input image plane. Figure 4.8 shows the synthesized images at the viewpoint, which is clear and sharp in the whole area thanks to the geometry estimation method.

In the following subsections, we first show the rate-distortion performance of the input images, which is a typical evaluation of light field coding methods. We then evaluate the view-dependent reconstruction quality of synthesized images using the method described in Section 4.2.3.

4.5.1 Rate-distortion Performance of the Input Images

Figure 4.9 shows the rate-distortion performance measured for the input images. We compared our coder with two conventional coders: one is JPEG coder that encodes the input images independently, and the other is MPEG-2 coder that encodes them as a sequence of moving pictures. They are simple implementations of an intra-image coder and an inter-image coder using disparity-compensated prediction, respectively. The performance of our coder was measured by keeping the quality of the synthesized image and geometry model constant and changing the quality of the residual information only. We controlled the quality of the residual information by changing

the parameter q_{min}, which is a threshold value for selecting the macroblock modes, and the quantization factor. We calculated the peak signal-to-noise ratio (PSNR) for each input image of the luminance value, and expressed its average and standard deviation as the reconstruction quality.

As shown in Fig. 4.9, our coder shows better performance than the other coders especially at low bit rates. The minimum bit rate of our coder is much lower than that of the MPEG-2 coder, because the geometry model of our coder introduces less overhead bits than the motion vectors of the MPEG-2 coder for the inter-view prediction. Although the MPEG-2 coder exceeds our coder at high bit rates for the *Santa* image set, note that the MPEG-2 coder does not have the view-dependent scalability. The performance of our coder at high bit rates could be improved by using predictive coding methods between the residual errors.

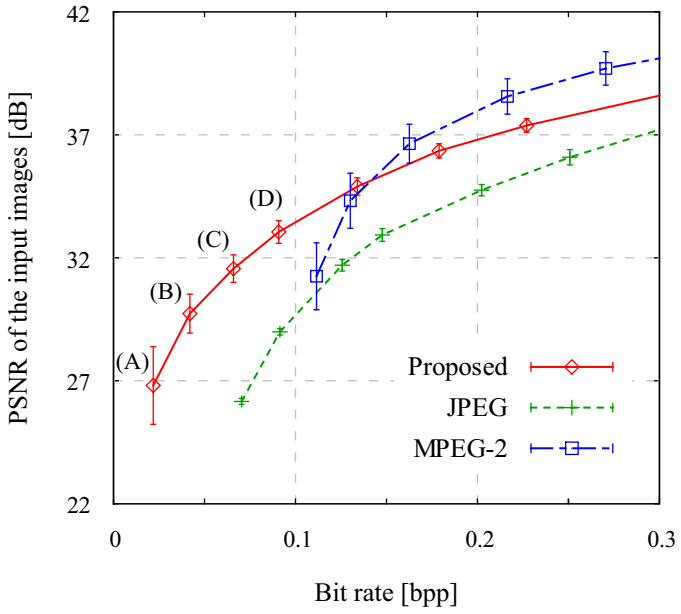
It can also be seen that the performance of the MPEG-2 coder is worse than that of the JPEG coder for the *Car and Flowers* image set. Since the elemental images of the integral photography record a small part of the scene separately, they have less inter-image correlations than the multi-camera images. Therefore, the prediction between the elemental images does not work efficiently. Our coder, on the other hand, shows good coding performance because it can perform efficient prediction using the representative viewpoint image that records an overview of the scene as the reference image (Fig. 4.8 (b)).

4.5.2 View-dependent Reconstruction Quality of the Synthesized Images

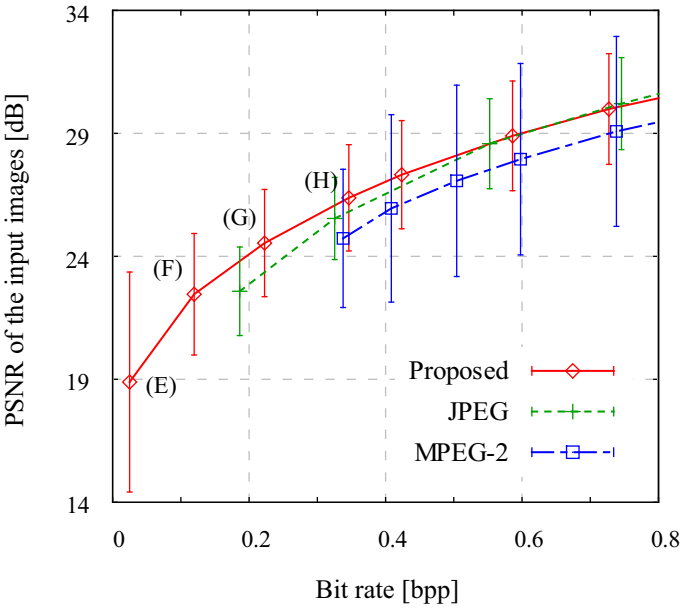
Figure 4.10 shows the reconstruction quality of the synthesized images using our coder. The reconstruction quality was measured at 36 viewpoints for each r, which is the distance between the representative viewpoint and the rendering viewpoint (see Fig. 4.4). These rendering viewpoints were placed on a circumference at regular intervals. The average and the standard deviation of the PSNR at these viewpoints are depicted against r. The bit rates of (A) to (H) correspond to those in Fig. 4.9. At the rates (A) and (E), no residual information was used; i.e., the synthesized image and the geometry model were only used to reconstruct the synthesized image. The bit rate of the residual information increases from (B) and (F) to (D) and (H), respectively.

The view-dependency of the reconstruction quality can be observed for both image sets; that is, the PSNR value of the synthesized image decreases with increasing the distance r. The quality of views around the representative viewpoint is kept high even at low bit rates, and the quality of views away from the representative viewpoint improves according to the increase of the residual bits. Our coder provides the view-dependent scalability since it produces a hierarchical bitstream whose reconstruction quality is view-dependent as shown in this experiment.

Figure 4.11 shows synthesized images from the reconstructed multi-view images. At low bit rates, the rendering quality decreases with increasing the distance between the rendering viewpoint and the representative viewpoint. For the *Santa* image set, the object is rendered with high quality

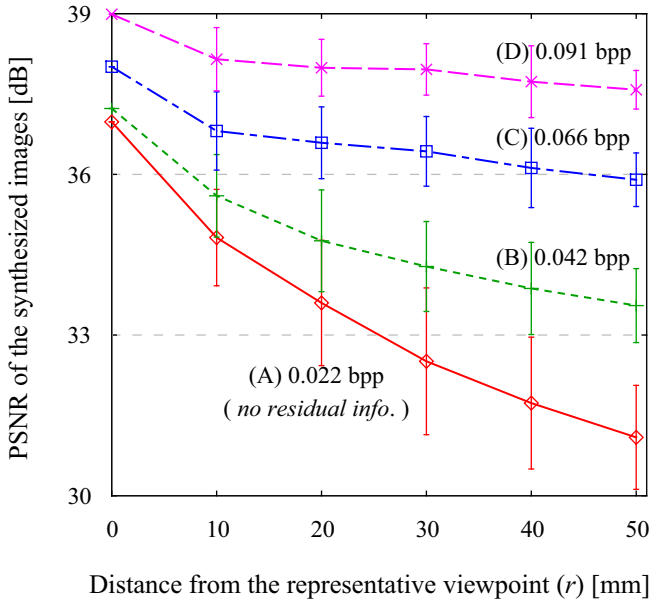


(a) *Santa*

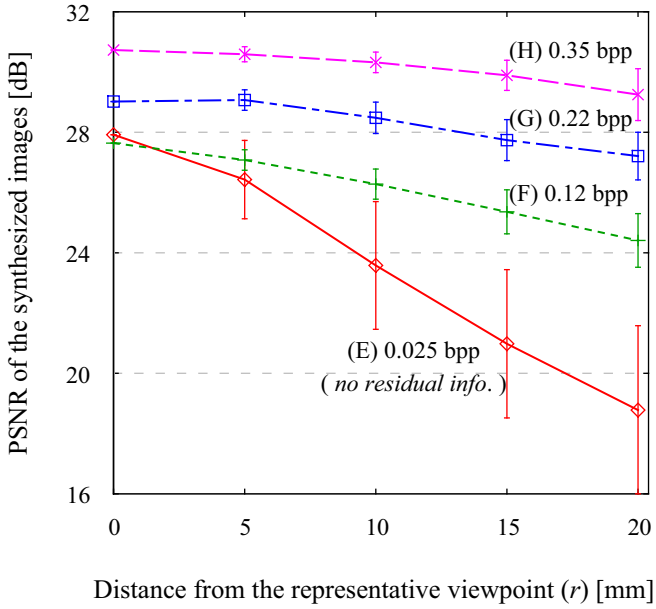


(b) *Car and Flowers*

Figure 4.9: Rate-distortion curves of the input images.



(a) *Santa*

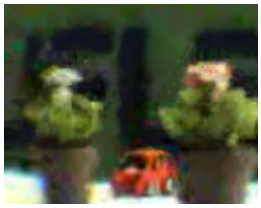


(b) *Car and Flowers*

Figure 4.10: Reconstruction quality of the synthesized image against the position of the rendering viewpoint. The bit rates of (A) to (H) correspond to those in Fig. 4.9.

	$(x, y, z) = (0, 0, 400)$	$(x, y, z) = (20, 0, 400)$	$(x, y, z) = (40, 0, 400)$
(B) 0.042 bpp	 PSNR: 37.23 dB	 PSNR: 34.23 dB	 PSNR: 33.11 dB
(D) 0.091 bpp	 PSNR: 38.99 dB	 PSNR: 38.01 dB	 PSNR: 37.46 dB

(a) *Santa*

	$(x, y, z) = (0, 0, 200)$	$(x, y, z) = (10, 0, 200)$	$(x, y, z) = (20, 0, 200)$
(F) 0.12 bpp	 PSNR: 27.64 dB	 PSNR: 26.92 dB	 PSNR: 24.95 dB
(H) 0.35 bpp	 PSNR: 30.73 dB	 PSNR: 30.38 dB	 PSNR: 29.13 dB

(b) *Car and Flowers*

Figure 4.11: Synthesized images from reconstructed multi-view images.

even at distant viewpoints and most errors occur in occluded regions, because the geometry model is relatively accurate and the prediction works well for the visible regions. For the *Car and Flower* image set, on the other hand, errors occur in whole areas, because the geometry model is not very accurate. The rendering quality at distant viewpoints improves with increasing the bit rate for both image sets.

4.6 Conclusions

In this chapter, we have presented a view-dependent light field coding method, which performs image-based rendering before the encoding process to generate an image at the representative viewpoint. It produces a view-dependent scalable bitstream, which can be used with three rendering methods depending on the bit rate. The experimental results showed that the images synthesized from the bitstream show view-dependent reconstruction quality, and the method also provides good coding performance for both multi-view images and integral photography images.

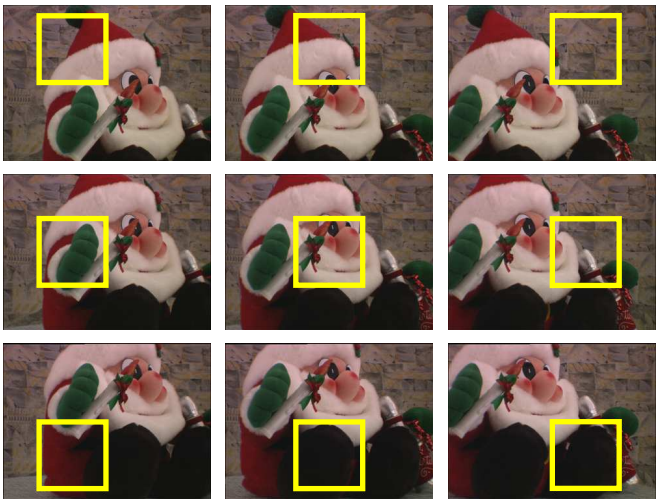
Chapter 5

ROI-Based Light Field Coding for View-Dependent Scalable Streaming

5.1 Introduction

A typical scenario of image-based rendering systems is interactive streaming of free-viewpoint images, in which a remote user browses novel 2D views of a scene over a network. The user sends requests to a server to synthesize a novel image at a viewpoint, and the server returns requested data. In this scenario, the simplest strategy for the server is transmitting all of the captured multi-view images regardless of the requested viewpoint. The user can render novel views at arbitrary viewpoints with this data, but this strategy requires huge bandwidth. To reduce the data amount, several camera array systems [95, 131, 139] use the fact that only a part of image segments in the multi-view images, which we call *reference regions*, is used to synthesize a novel image at a certain viewpoint (see Fig. 5.1, for example). By transmitting the reference regions depending on the requested viewpoint, the user can synthesize a novel image at that viewpoint. However, this transmission strategy restricts the movement of the user's viewpoint, since the data for the other views do not arrive until after the round-trip time of the network has elapsed, which results in less interactivity if the network has high latency. This viewpoint-movement restriction is the problem that we solve in this chapter.

We propose a scalable coding method for the interactive streaming of dynamic light fields. Our method assumes that the remote user moves the viewpoint smoothly from the previous viewpoint. It defines a region of interest (ROI) that includes the reference region together with its neighboring region. By encoding and transmitting this ROI with high priority, the user can render high-quality novel views near the requested viewpoint before the arrival of the next frame data. Our method can thus compensate the smooth movement of the user's viewpoint. The user can arbitrarily choose the movable range of the viewpoint by controlling the ROI size. Moreover, with increasing bandwidth, the user can extend the ROI to include the larger neighborhood of the reference region while



(a) Parts of input images and reference regions

(b) Synthesized image

Figure 5.1: (a) Parts of input multi-view images of *Santa* image set. Rectangles in the input images represent reference regions that contribute to synthesizing (b) a novel image at a certain viewpoint.

keeping the rendering quality at the requested viewpoint fixed. In this way, our method provides the functionality of view-dependent scalability. As shown in Table 5.1, our ROI method achieves view-dependent scalable streaming that takes account of the bandwidth and latency of the network. It is an intermediate approach between two typical transmission strategies.

To our knowledge, the most closely related work to our coding method is rate-distortion optimized streaming of light fields [17, 90]. In these methods, the server determines the bit rate of each input image [90] or image segment [17] to be transmitted using a rate-distortion optimization method so that the distortion of the requested view can be minimized. Our method, by contrast, aims to provide high-quality rendering results near the requested viewpoint, as well as the requested view itself, so that the movement of the remote user can be compensated even under high network latency. We therefore use ROI-based techniques when performing the bit allocation, which depends on the requested viewpoint like the related work. In addition to these approaches, a number of multi-view compression schemes using temporal and spatial (inter-view) prediction have been proposed, as we reviewed in Section 2.2. To reduce the data amount, these prediction-based schemes consider how to exploit inter-image correlations, while our method considers which image segments are encoded and transmitted with high priority. Compared to the coding method presented in Chapter 4, which also provides the view-dependent scalability, the method presented in this chapter allows the user to more flexibly control the scalability by changing the ROI size.

Table 5.1: Comparison of transmission strategies.

Strategy	Features
Transmitting all multi-view images	The user can render novel views at arbitrary viewpoints, but it requires huge bandwidth.
Transmitting reference regions only	It reduces bandwidth, but the user cannot immediately change the viewpoint if the network has high latency.
Transmitting ROI (our method)	The user can arbitrarily choose the movable range of the viewpoint depending on the bandwidth and latency of the network.

However, it does not provide 2D compatibility, since novel views are synthesized at the client.

The rest of this chapter is organized as follows. Section 5.2 explains the rendering method and the structure of the reference region. Section 5.3 presents our ROI coding framework and the method used to provide view-dependent scalability. In Section 5.4, we evaluate the view-dependent reconstruction quality of our coding method using a modified JPEG2000 codec, and the chapter is concluded in Section 5.5.

5.2 Reference Regions in Multi-View Images

We assume that multi-view images are captured with many cameras that lie on a plane and are arranged in a regular 2D grid, and that there is no prior knowledge of the scene geometry except for the minimum depth of the scene. The light field constructed from those images can be parameterized with (s, t, u, v), where (s, t) and (u, v) denote the positions of cameras and pixels, respectively. For simplicity, we discuss a 2D subspace (s, u) as shown in Fig. 5.2.

To synthesize a novel image at a requested viewpoint (s_0, z_0), light rays that pass through the viewpoint need to be gathered. They must satisfy

$$u = \frac{f}{z_0}(s - s_0), \tag{5.1}$$

where f is the focal length of the input cameras. Since a light field is usually composed of a finite number of images, depth estimation is widely adopted to appropriately interpolate the light rays that are not actually captured with the input cameras.

As we described in Section 2.1.4, typical rendering methods of real-time systems perform view-dependent depth estimation and color interpolation using k-nearest cameras as reference cameras to prevent an unnecessarily blurred result and keep computational cost low. Figure 5.3 shows the method for interpolating a desired light ray in the (s, u) subspace. The rendering method assumes a layered depth model $z_n (n = 1, 2, .., N)$ in the object space, and estimates the depth for each target light ray. At the intersection of the target light ray with each of depth layers, it

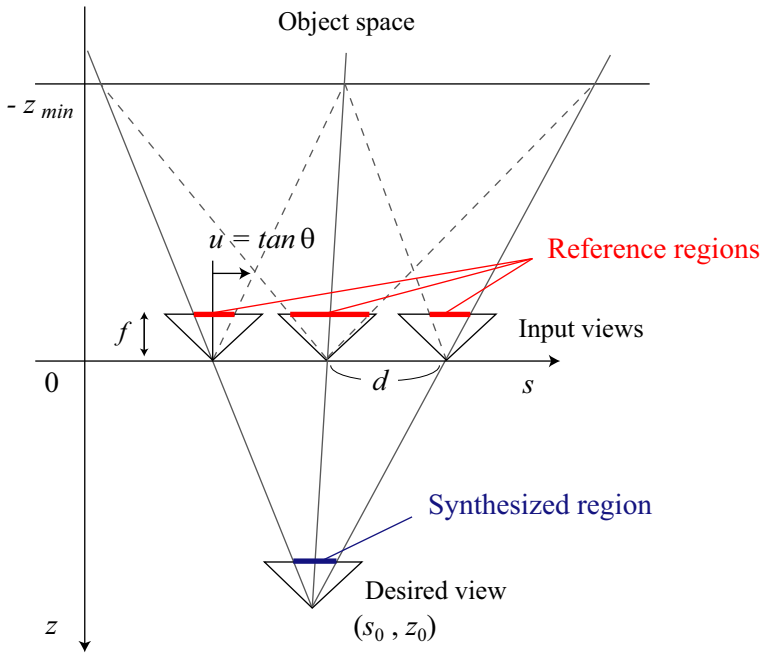


Figure 5.2: Reference regions used to interpolate the synthesized region in the desired view.

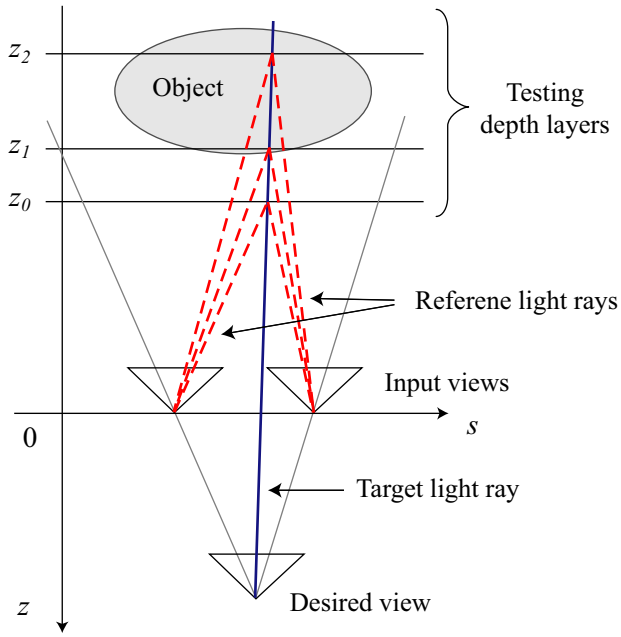


Figure 5.3: Interpolation method for synthesizing a target light ray.

evaluates the similarity of the reference light rays, which correspond to the back-projections of each intersection point to the nearest cameras. In the case of Fig. 5.3, for example, the reference light rays depicted as dotted lines are used to estimate the depth of the target light ray. Finally, the depth z_1 would be selected because the light rays coming from the same point on an actual object have similar colors.

The reference region defined in this chapter includes all of the reference light rays used to synthesize a requested view. Using the above rendering method, the reference region is limited by the minimum depth of the scene, z_{min}, as shown in Fig. 5.2, and it is described by

$$\left| u - \frac{f}{z_0}(s - s_0) \right| \leq \frac{z_{min} + z_0}{z_{min}\,|z_0|}\, fd, \tag{5.2}$$

where d is the distance between the input cameras. It changes according to the requested viewpoint (s_0, z_0). A notable point is that the reference region moves only around its previous location if the movement of the user's viewpoint is smooth. With a displacement of the viewpoint Δs_0 in the s direction, the reference region shifts as $-\Delta s_0\, f/z_0$ in the u direction. Meanwhile, with Δz_0 in the z direction, the slope of the reference region changes by $-\Delta z_0\, f/\left(z_0(z_0 + \Delta z_0)\right)$, and its size changes by $-\Delta z_0\, fd\,/\left(z_0\,|z_0 + \Delta z_0|\right)$.

5.3 ROI-Based Light Field Coding

Figure 5.4 shows the reference region discussed in Section 5.2 in the (s, u) subspace of a light field. The reference region lies along the line denoted by Eq. (5.1), and moves only around its previous location if the user smoothly changes the viewpoint. We use this locality of the reference region in a 4D light field to construct our coding framework for view-dependent scalable streaming.

5.3.1 Definition of ROI

We define an ROI to include the reference region and its neighborhood, as shown in Fig. 5.4. In this definition, ROI includes not only the image segments used for the requested viewpoint, but also the segments needed for nearby viewpoints around the requested viewpoint. Smooth movement of the user can be compensated by encoding and transmitting the ROI data with high priority. The size of ROI corresponds to the movable range of the user. Note that the ROI is a 4D segment in the light field. In practice, however, our ROI method encodes and transmits the cross sections between the ROI and each of the input images with high priority. This means that each of the input images has a 2D ROI determined by a cross section with the 4D ROI in the light field. The light field data can therefore be considered a set of 2D images with 2D ROIs.

We can design several types of weight functions for the ROI, as shown in Fig. 5.5 (see Section 5.4.1 for implementation details). The movable range of the viewpoint enlarges with the

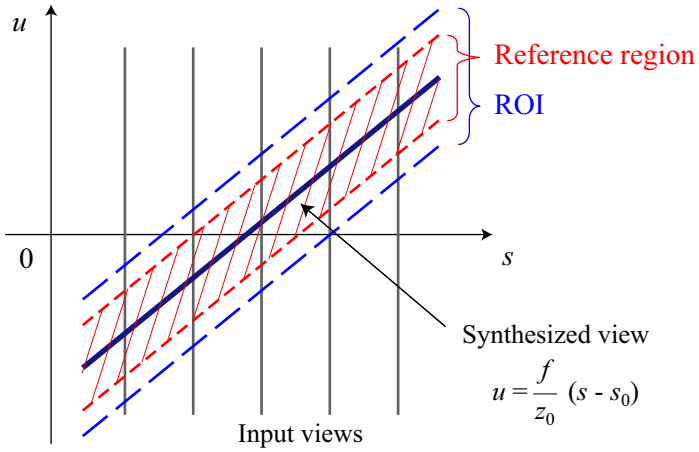


Figure 5.4: Reference region and ROI for synthesizing a view in the (s, u) subspace of a light field. The lines perpendicular to the s axis represent the actual light rays captured with the input cameras.

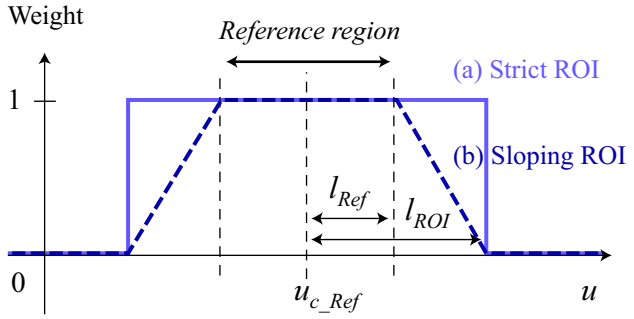


Figure 5.5: Example settings of ROI size and weight in an input image.

increase of the ROI size (l_{ROI} in Fig. 5.5). The weight value affects the quality of synthesized images at nearby viewpoints. In this study, we consider symmetric ROI settings about the center of the reference region (u_{c_Ref}) in both (u, v) directions in each input image.

5.3.2 View-Dependent Scalability

Figure 5.6 shows the scalable bitstreams of our coding method, in which the size of the ROI is enlarged with increasing bit rate. By collecting these incremental contributions from all input images into the layers, the coding method provides view-dependent scalability (i.e., the data near the reference region has higher priority than the data away from it). Using this structure, the reconstruction quality around the requested viewpoint is kept high even if the bitstream is truncated

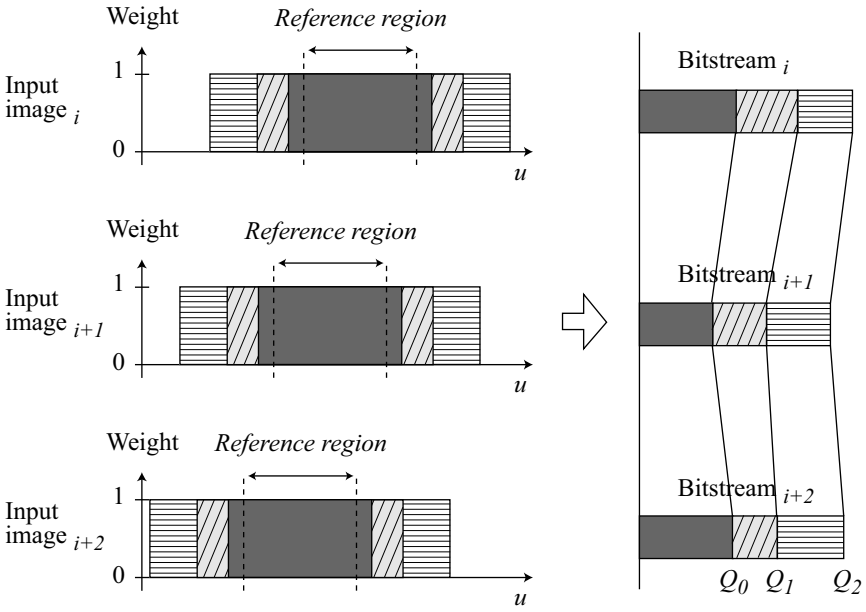


Figure 5.6: View-dependent scalable bitstreams using ROI.

to an intermediate layer at low bit rates. It is similar to the quality layer in JPEG2000 [113], which provides SNR scalability and resolution scalability.

Some previous works [34, 62, 99] define *view scalability* so that users can arbitrarily choose the number of camera views to be decoded. These methods reconstruct one of the input images as the minimum unit. Such functionality is useful for applications that use the input camera views directly. On the other hand, we define *view-dependent scalability* so that users can arbitrarily choose the movable range of the viewpoint, which is not restricted to the input camera viewpoints. Our method reconstructs the ROI that includes the image segments needed for rendering in a certain viewing area. It is useful for interactive browsing of a 3D scene in which the user moves the viewpoint smoothly.

5.4 Experiments

We evaluated the view-dependent reconstruction quality of our coding method using a static multi-view image set, *Santa* (Fig. 5.1 (a)), which is from the multi-view image database provided by University of Tsukuba, Japan. The image set consists of 81 (9×9) still images of 640×480 pixels. The cameras are arranged in a regular 2D grid on a plane, where the distance between them (d in Fig. 5.2) is 20 mm. The minimum depth of the target scene, z_{min}, is 519 mm. For rendering novel views, we used the rendering method described in Section 2.1.4, in which the

number of testing depth layers, N, was set to 10. The resolution of the synthesized image was set to 512 $\times$ 512 pixels. The *original viewpoint*, which we define as the viewpoint originally requested by a remote user, was set at $(s_0, t_0, z_0) = (0, 0, 500)$ mm. Figure 5.1 (b) shows the image synthesized at this original viewpoint. We encoded the input images independently at the same compression ratio, because the reference region appears in all input images for the rendering at this original viewpoint, as shown in Fig. 5.1 (a). The reference region occupies 13.8% of the whole image in this configuration.

5.4.1 Implementation of ROI Coding

We implemented our coding method by modifying a JPEG2000 codec, JasPer [153], because the ROI coding with flexible weight control can be performed. JPEG2000 first decomposes the image into $3L + 1$ subbands using a discrete wavelet transform. The wavelet coefficients of each subband are then partitioned into small rectangular blocks, called code-blocks. Each code-block, B_i, is independently encoded into a finely embedded bitstream, which has a sequence of distortion value, $D_i^{n_i}$, and bit rate, $R_i^{n_i}$, for each truncation point, n_i. Finally, rate-distortion optimization is performed with the embedded block coding with optimized truncation (EBCOT) algorithm [113] for each code-block to minimize its cost function given as

$$D_i^{n_i^\lambda} + \lambda R_i^{n_i^\lambda}, \tag{5.3}$$

where λ is a weight parameter determined by the available bit rate. An increase of λ yields a decrease of bit rate.

We set the resolution level, L, to 2 and the size of code-block to 16×16 in this implementation. For ROI coding, we modified the cost function as

$$D_i^{n_i^\lambda} + \frac{\lambda}{weight\,(i)}\, R_i^{n_i^\lambda}, \tag{5.4}$$

where $weight\,(i)$ is a weight function depending on the location of the code-block B_i in the image. As shown in Fig. 5.5, we implemented two types of ROIs, which we call *strict ROI* and *sloping ROI*, as follows. Let $|u_i'|$ be the distance between the center of the reference region (u_{c_Ref} in Fig. 5.5) and that of the code-block, $u_{c\_B_i}$, i.e., $|u_i'| = |u_{c\_B_i} - u_{c\_Ref}|$. The strict ROI, which definitely separates the ROI from other regions, employs the following weight function:

$$weight\,(i) = \begin{cases} 1 & \text{if } |u_i'| \le l_{ROI} \\ 0 & \text{otherwise} \end{cases}. \tag{5.5}$$

Meanwhile, the sloping ROI, in which the weight value gradually decreases to zero according to

the distance from the reference region, employs

$$weight\,(i) \;=\; \begin{cases} 1 & \text{if } |u_i'| \le l_{Ref} \\ a^{\,b\,\frac{(\,|u_i'|-l_{Ref}\,)}{(\,l_{ROI}-l_{Ref}\,)}} & \text{if } l_{Ref} < |u_i'| \le l_{ROI} \\ 0 & \text{otherwise} \end{cases} , \tag{5.6}$$

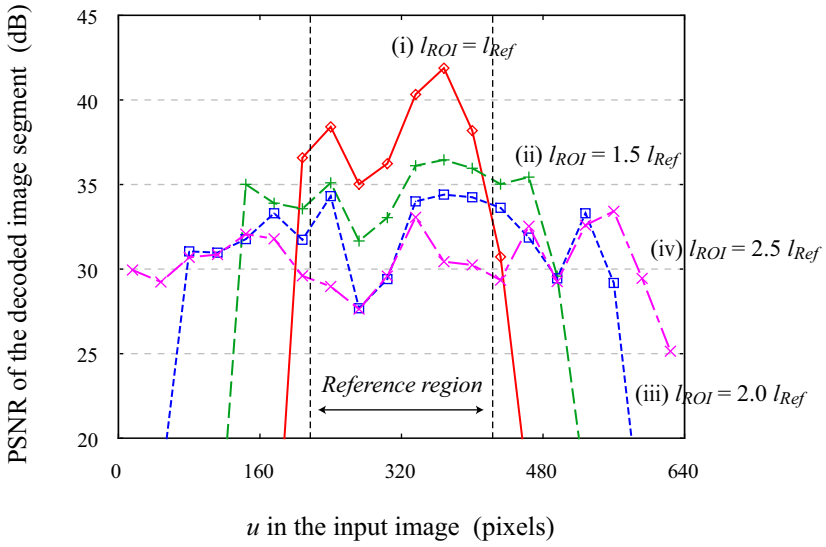
where a and b are constant numbers empirically set to $1/2$ and 10, respectively. The default ROI coding method of JPEG2000, known as the Maxshift method [5], does not deal with a weight function like Eq. (5.6), so we modified the codec to perform flexible bit allocation.

Figure 5.7 shows coding performance of the above ROI methods for different ROI sizes at the same compression ratio. The center image of the *Santa* image set was used in this experiment, and the reconstructed images for two types of ROIs are depicted in Fig. 5.8. In Fig. 5.7, the reconstruction quality is expressed as the peak signal-to-noise ratio (PSNR) of the luminance value, which was calculated for the image segment of 32 $\times$ 32 pixels along the center line of the image (the dotted lines in Fig. 5.8). These plots show that the ROI has higher quality than the other region, and its size is controlled by l_{ROI}. In the reference region, the (b) sloping ROI has higher reconstruction quality than the (a) strict one at the same compression ratio and the same ROI size. For the outside of the reference region, the (b) sloping ROI shows gradual degradation of the reconstruction quality, while the (a) strict one keeps almost the same quality as in the reference region. This is because the (b) sloping ROI devotes higher amount of data for the inside of the reference region than the outside, while the (a) strict one makes no distinction between the inside and the outside of the reference region in the ROI.

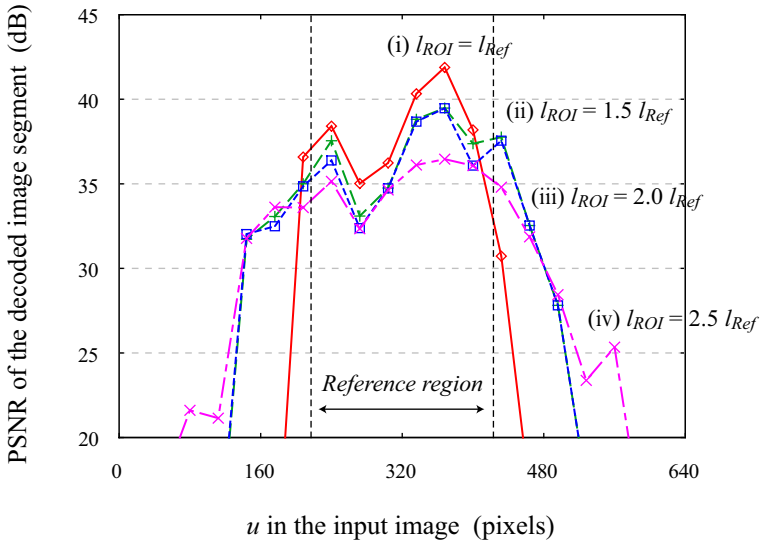
5.4.2 Evaluation Method for View-Dependent Reconstruction Quality

For measuring the view-dependent reconstruction quality of our method, we used the same evaluation method that we used in Chapter 4. In the evaluation method, the reconstruction quality is not the quality of decoded multi-view images themselves, as in the case of Fig. 5.7, but that of novel views synthesized from the decoded multi-view images. As shown in Fig. 4.4, we compared images synthesized from the decoded multi-view images and those from the original multi-view images. To clarify the view-dependency of reconstruction quality, we focused on the relation between the rendering quality and the distance of the rendering viewpoint from the original viewpoint, which is denoted as r in Fig. 4.4. This evaluation shows the range in which high-quality images can be rendered by using the decoded multi-view images. At low bit rates, the reconstruction quality would be lower at larger distance r.

In the following results, the reconstruction quality was measured at 36 viewpoints for each distance r. These viewpoints were placed at regular intervals on the circumference whose z coordinate is constant at z_0. As the measure of quality, we adopted the PSNR of the luminance value averaged over the 36 viewpoints for each value of r.

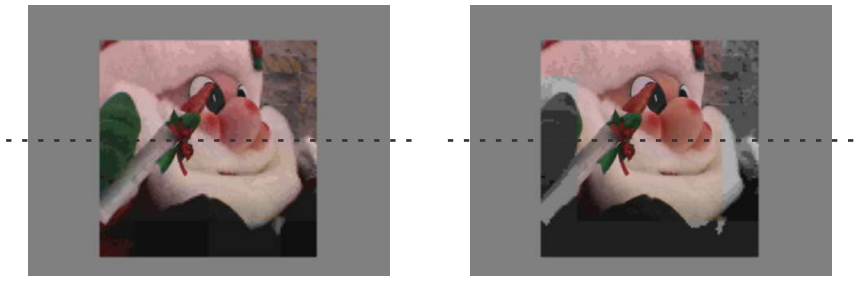


(a) Strict ROI



(b) Sloping ROI

Figure 5.7: Reconstruction quality of decoded image segments for different ROI sizes. The center image of the *Santa* image set was used in this experiment. Compression ratio was constant at 0.005 for all curves.



(a) Strict ROI (b) Sloping ROI

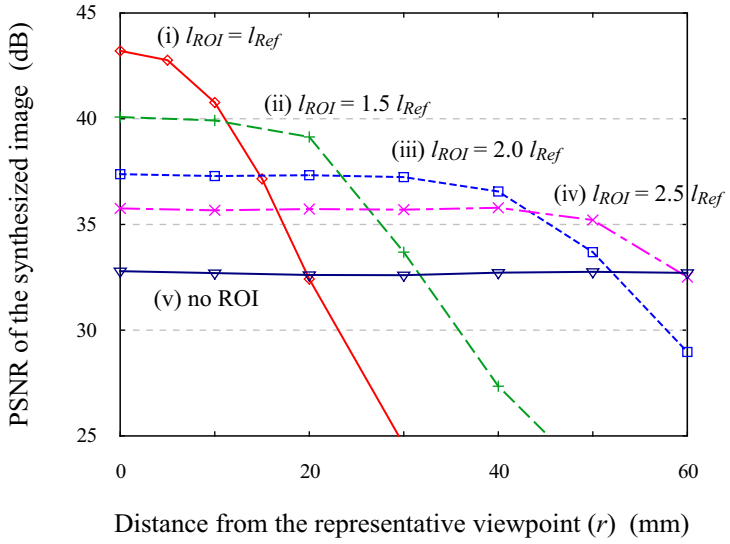
Figure 5.8: Reconstructed images of the center image of the *Santa* image set ($l_{ROI} = 1.5 l_{Ref}$, compression ratio: 0.005). The gray area around the image had no information to be decoded. Reconstruction quality in Fig. 5.7 was measured along the dotted line in this figure.

5.4.3 Results

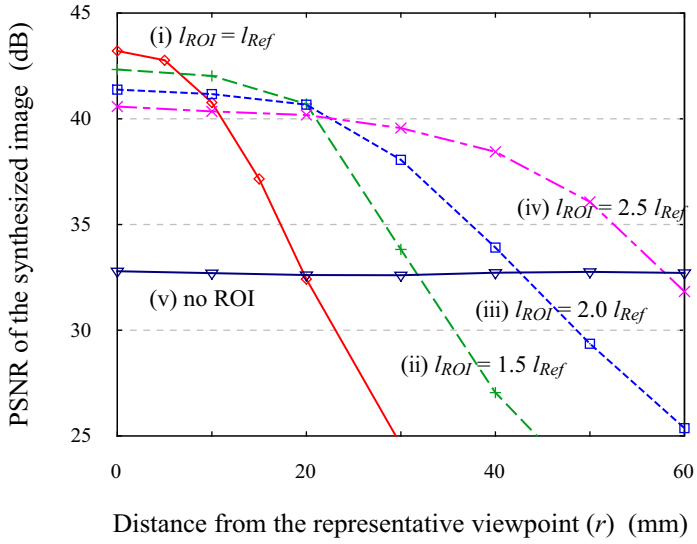
Figure 5.9 shows the view-dependent reconstruction quality obtained using different ROI sizes and a constant compression ratio of 0.005. In Figs. 5.9 (a) and (b), curve (i) represents the result where the ROI is exactly the same as the reference region. The ROI size expands from curves (ii) to (iv), and curve (v) represents the result where the whole image is encoded uniformly. Thanks to the ROI coding method, the reconstruction quality around the original viewpoint is kept higher than that of the other viewpoints. By changing the ROI size, we can control the trade-off between the maximum rendering quality and the range of the high-quality rendering area.

Figure 5.10 shows the result of another setting, in which we changed the bit rate and the ROI size simultaneously. The bit rate was increased with increasing the ROI size so that the reconstruction quality at the original viewpoint was kept approximately constant. Using this control scheme, we can extend the movable range of a remote user with increasing bit rate, while keeping the maximum rendering quality. In this way, our coding method provides view-dependent scalable streaming, in which the maximum rendering quality and the viewing area can be adaptively controlled according to the bit rate.

Figure 5.11 depicts synthesized images obtained using the sloping ROI. At the original viewpoint, the quality of the synthesized image is kept high even at low bit rates (Fig. 5.11 (a)). With a large distance r, however, some of the image segments needed for rendering the view are not included in the bitstream; therefore, the synthesized image has some missing parts (Fig. 5.11 (b)). The reconstruction quality at that distance improves with the increase of both the ROI size and bit rate, as shown in Figs. 5.11 (c) and (d).

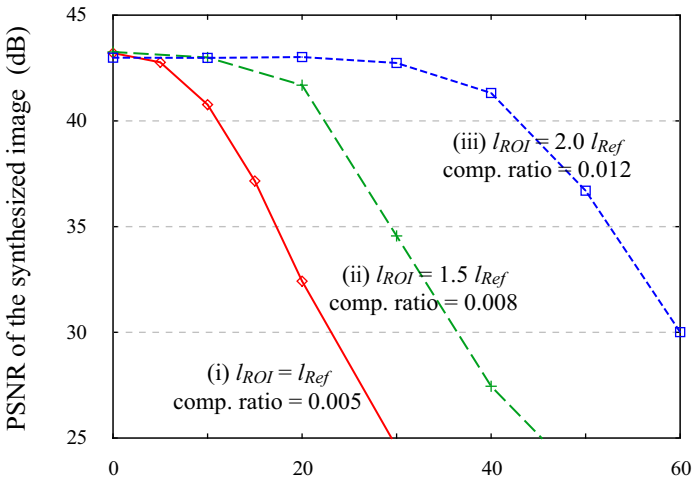


(a) Strict ROI

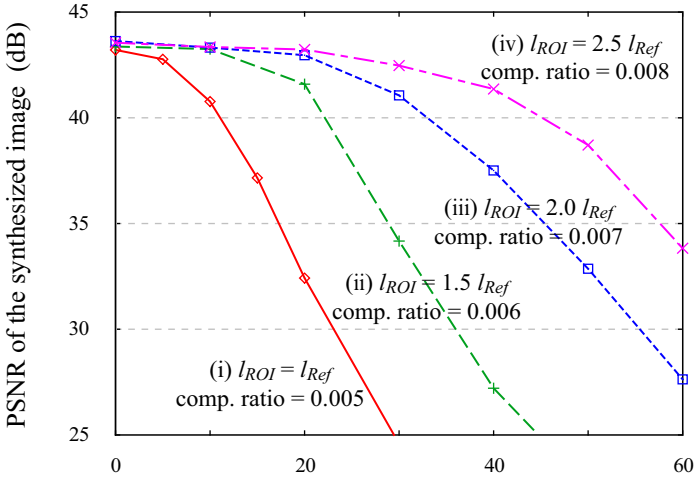


(b) Sloping ROI

Figure 5.9: View-dependent reconstruction quality obtained using different ROI sizes for the *Santa* image set. Compression ratio was constant at 0.005 for all curves.



(a) Strict ROI



(b) Sloping ROI

Figure 5.10: View-dependent reconstruction quality obtained using different ROI sizes and compression ratios for the *Santa* image set. The reconstruction quality at the original viewpoint was kept approximately constant by adjusting the compression ratio.

(a) comp. ratio: 0.005
r: 0 mm

PSNR: 43.21 dB PSNR: 40.23 dB

(b) comp. ratio: 0.005
r: 30 mm

PSNR: 24.85 dB PSNR: 23.43 dB

(c) comp. ratio: 0.006
r: 30 mm

PSNR: 34.72 dB PSNR: 33.00 dB

(d) comp. ratio: 0.008
r: 30 mm

PSNR: 41.94 dB PSNR: 39.69 dB

Figure 5.11: Synthesized images obtained using the sloping ROI for the *Santa* (left) and *Plant* (right) image sets. The *Plant* image set has same parameters as the *Santa* image set.

5.4.4 Discussion

As described in Section 5.4.3, our coding method controls the view-dependency of the reconstruction quality by changing the ROI size. The method enables view-dependent scalable streaming of light fields in which the movement of a remote user can be adaptively compensated depending on the bit rate.

Compared to the strict ROI, the sloping ROI provides more gradual degradation of the reconstruction quality as the distance r increases, as shown in Fig. 5.9. Moreover, the sloping ROI attains better reconstruction quality at the same bit rate, as shown in Fig. 5.10. This observation reflects the fact that the region near the center of the ROI has more chances to be used for synthesizing light rays than the near-boundary region in the ROI; that is, if the actual depth of a desired light ray is behind the minimum depth of the scene (see Fig. 5.3), only the region near the center of the ROI contributes to synthesizing the final color of that light ray, while the near-boundary region is only used to estimate the depth of that light ray. It is therefore effective to assign more bits to the region near the center of the ROI like the sloping ROI.

Our ROI coding method would be more efficient if combined with some models that describe the movement of the remote user's viewpoint. If the user is moving in a certain direction, more bits can be assigned by extending the ROI to that direction, instead of using the symmetric ROI settings used in the experiments. Moreover, if the user is moving slowly, small ROI can be used to achieve higher-quality rendering in exchange for the shrinkage of the movable range. By considering the moving direction and the speed of the user's viewpoint in this way, effective bit allocation to compensate these movements could be achieved.

In the experiments, we only dealt with the case in which all input multi-view images have the ROI. If the original viewpoint is near to the plane where the input images are captured, however, the ROI appears in a subset of the input images. In that case, our ROI scheme could be available by modifying the weight function ($weight\,(i)$ in Eq. (5.4)) to take account of the distance between the code-block and the reference region in (s, t) directions, as well as the distance in (u, v) directions ($|u_i'|$) in our current implementation. With such weight function, rate-distortion optimization need to be performed for all input images together, and the resulting bit rate of each input image would be different according to the distance from the reference region.

5.5 Conclusions

We have presented a scalable coding method for interactive streaming of dynamic light fields. The method uses ROI-based techniques to provide view-dependent scalability, with which the smooth movement of the remote user's viewpoint can be compensated even under high network latency. Experimental results show that our ROI method can adaptively control the view-dependency of

the reconstruction quality by changing the ROI size. Developing a streaming system of dynamic light fields considering the user's movement is an interesting future direction. It is also an important issue to combine predictive methods, which exploit the correlations between images both in temporal and spatial directions, with our ROI framework to attain higher compression ratio.

Chapter 6

Rendering-Oriented Decoding for a Distributed Multi-View Coding System Using a Coset Code

6.1 Introduction

Camera array systems can capture multi-view images of a 3D scene, which allow a viewer to observe the scene from arbitrary viewpoints by using image-based rendering techniques [55, 100, 140]. Such systems require efficient coding schemes owing to the large amount of data, typically consisting of hundreds of views. Since they capture an identical scene from slightly different viewpoints, significant correlations exist among the multi-view images. Most of conventional coding methods, as well as currently developed MPEG standard [76, 103], exploit these correlations at the encoder using the concept of disparity compensation. However, they require high encoding complexity and communication between cameras with large data volume.

Distributed multi-view coding schemes provide a solution for such problems [1, 40, 46, 48, 145]. In these methods, each image is encoded independently, but decoded jointly at a central decoder. Since the inter-camera communication is avoided, low-complexity encoding and a simple system configuration can be achieved. The inter-image correlation is exploited at the decoder. Therefore, compression efficiency is still higher than that possible by conventional intra-coding methods. In previous works, however, the decoder seems to pay an unnecessary computational cost when the viewer only observes a novel image synthesized at a desired viewpoint, instead of the decoded images themselves. This is because it first reconstructs input camera images and then synthesizes the novel image with a general renderer using the decoded images. To our knowledge, there is no approach so far that synthesizes a novel image directly from the encoded data.

In this chapter, we consider a system in which multi-view images are captured and encoded in a distributed fashion and a viewer synthesizes a novel image at a desired viewpoint by using

this data. We propose an efficient method that combines decoding and rendering processes so that the novel image can be directly synthesized without having to reconstruct all the input images. This method, called rendering-oriented decoding, jointly performs two key techniques, disparity compensation in the decoding process and geometry estimation in the rendering process, because they are essentially equivalent if the camera parameters for the multi-view images are known. When the viewer only synthesizes a novel image, our method requires low computational cost compared to a typical method that performs the two above processes separately. Our method keeps the complexity of both the encoder and decoder as low as a conventional intra-coding method, while attaining better coding performance thanks to the inter-image decoding.

The rest of this chapter is organized as follows. Section 6.2 briefly describes two basic schemes for this study: distributed multi-view coding techniques and an image-based rendering algorithm. Section 6.3 presents our rendering-oriented decoding method. Section 6.4 evaluates the coding efficiency and processing time of our method compared to a conventional intra-coding method, and Section 6.5 concludes the chapter.

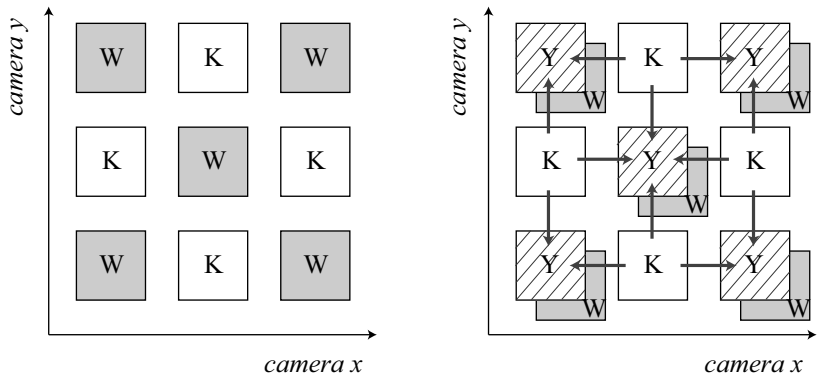
6.2 Background

6.2.1 Distributed Video/Multi-View Coding

Current video coding standards perform inter-frame prediction at the encoder to exploit the similarities among temporally successive frames. The motion compensation used in the inter-frame prediction makes the encoder much more complex than the decoder. Distributed video coding is a new paradigm that reverses the complexity balance between the encoder and decoder; it encodes the frames independently, but decodes jointly using the motion compensation at the decoder [33]. Two information-theoretic results presented by Slepian and Wolf [102] and Wyner and Ziv [126] suggest that such an intra-frame encoder and inter-frame decoder system can have close coding efficiency to a typical inter-frame encoder and decoder system.

The distributed coding approach can be also used for multi-view coding, where the inter-view prediction is only performed at the decoder. It makes the configuration of multi-view capturing systems simple, because the encoder can avoid the inter-camera communication with large data volume. It also allows the decoder to arbitrarily determine the prediction structure for decoding a frame; this is suitable for a multi-view streaming application that allows users to interactively switch between different viewpoint videos, called free-viewpoint switching [21, 41].

Figure 6.1 shows a typical structure of a distributed multi-view coding system. The images are classified into two categories: key images (K) and Wyner-Ziv images (W). The key images are encoded and decoded independently with a conventional intra-image coder. The Wyner-Ziv images are encoded independently by applying a channel coder for their pixel values or transformed



(a) Encoder (b) Decoder

Figure 6.1: A typical structure of distributed multi-view coding systems.

coefficients, and the resulting parity bits are transmitted to the decoder. To decode the Wyner-Ziv image, its estimate, called side information (Y), is generated through disparity-compensated prediction using the previously decoded key images, and the prediction error is corrected by using the parity bits of the image.

The compression efficiency of the distributed coding method greatly depends on the accuracy of the side information, because only a few parity bits are needed to correct small prediction errors. If a geometry model of the target scene is available, accurate side information can be generated by warping the neighboring views [1]. For multi-view video sequences, to improve the quality of side information, the motion-compensated prediction can be combined with the disparity-compensated one [40, 48].

6.2.2 Rendering Using Multi-View Images

We use the method described in Section 2.1.4 as the rendering method of the system presented in this chapter. Here we briefly review the method.

The rendering method assumes that multi-view images are captured with calibrated cameras that roughly lie on a plane and are arranged on a 2D grid, and that there is no prior knowledge of the scene geometry. As shown in Fig. 2.1, it sets a layered depth model, $z = \{z_n | n = 1, 2, .., N\}$, in the object space to equally divide the disparity space as

$$\frac{1}{z_n} = \frac{1}{z_{max}} + \frac{n - 1/2}{N}(\frac{1}{z_{min}} - \frac{1}{z_{max}}), \tag{6.1}$$

where z_{max} and z_{min} are the maximum and minimum depths of the scene.

The method estimates the depth for each target light ray, $\boldsymbol{r}(\boldsymbol{x})$, where $\boldsymbol{x}$ represents the position of the light ray in the desired view. At the intersection of the target light ray with each of the depth

layers ($\boldsymbol{p}(\boldsymbol{x}, z)$), it evaluates the color consistency of the reference light rays, denoted by $\boldsymbol{r}_i(\boldsymbol{x}, z)$ where i is the camera index. To prevent the occlusion effect and keep computational cost low, this evaluation is only performed on the k-nearest cameras (reference cameras). The color consistency cost is therefore given by

$$C(\boldsymbol{x}, z) = consistency\left(I(\boldsymbol{r}_i(\boldsymbol{x}, z))|_{i \in V}\right), \tag{6.2}$$

where V is the set of camera indices near the target light ray and $I(\cdot)$ denotes the color of the light ray. We used the sum of variances for each RGB component as the consistency measure, and set $|V| = k = 4$ as shown in Fig. 2.1.

This cost function is smoothed in each depth layer in order to reduce noise effects. For this smoothing, we use a normal block filter

$$\bar{C}(\boldsymbol{x}, z) = \frac{1}{|S|} \sum_{\boldsymbol{x}' \in S} C(\boldsymbol{x}', z), \tag{6.3}$$

where S is a rectangular window whose center is $\boldsymbol{x}$. Finally, the depth value that minimizes the cost is selected for each target light ray:

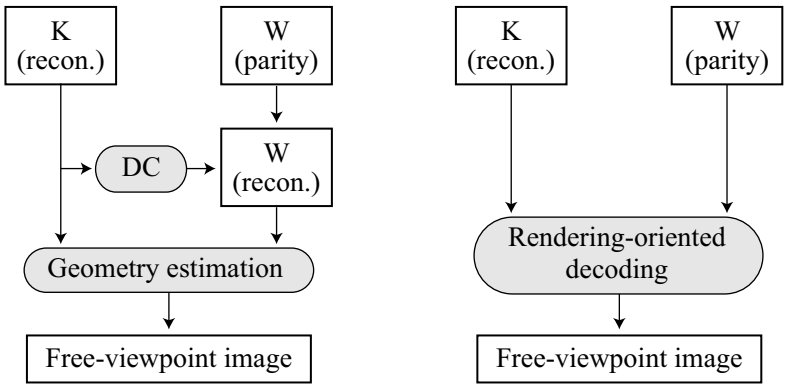
$$z_{opt}(\boldsymbol{x}) = \arg\min_{z} \bar{C}(\boldsymbol{x}, z). \tag{6.4}$$

As in the depth estimation, we use k-nearest reference light rays to interpolate the color of the target light ray. This approach keeps the view-dependent components of the target scene and prevents an unnecessarily blurred result [13]. We use bilinear interpolation of the colors of the reference light rays for the optimal depth:

$$I(\boldsymbol{r}(\boldsymbol{x})) = \sum_{i \in V} w_i(\boldsymbol{x})\, I(\boldsymbol{r}_i(\boldsymbol{x}, z_{opt}(\boldsymbol{x}))). \tag{6.5}$$

Here, $w_i(\boldsymbol{x})$ is the weight for the i-th reference light ray $\boldsymbol{r}_i(\boldsymbol{x}, z_{opt}(\boldsymbol{x}))$, and it takes a floating-point value between 0 and 1 depending on the positions of the reference cameras and the target light ray; $w_i(\boldsymbol{x})$ takes 1 if the target light ray passes through the i-th camera position, while it takes 0 if it passes through the other neighboring camera positions, and $\sum_{i \in V} w_i(\boldsymbol{x}) = 1$.

Note that the reference camera set V depends on the position of each target light ray $\boldsymbol{x}$. Therefore, the number of input cameras used for rendering the entire view depends on the desired viewpoint. This rendering method, however, has constant computational complexity regardless of the number of input cameras, because it calculates the color and cost for each target light ray. The computational complexity is determined by the number of target light rays (i.e., the resolution of the desired view) and the number of depth layers.



(a) Typical method (b) Our method

Figure 6.2: Process flow for synthesizing a free-viewpoint image using (a) a typical method (DC: disparity compensation) and (b) our method.

Reference Region

For synthesizing a novel image, the above rendering method does not require all light rays acquired with the input cameras; instead, as we showed in Section 5.2, it only requires the light rays in reference regions, which we define as segments in the input images that include all of the reference light rays used to synthesize a desired view. The reference region in an input image is a rectangular segment given by Eq. (5.2) when we use a regular camera arrangement. For an irregular (practical) camera arrangement, the reference regions are similarly defined as quadrangular segments in the input images.

Based on the locality of the reference regions, several camera array systems [95, 131, 139] use an approach that only transmits or decodes image segments including the reference regions to reduce the data amount. Our ROI-based coding method presented in Chapter 5 also uses this concept. However, they do not address inter-view prediction. The method described in this chapter, by contrast, decodes the light rays in the reference regions with inter-view prediction based on a distributed coding approach. Moreover, since the inter-view prediction is incorporated into the geometry estimation in the rendering process, the method keeps the decoder complexity as low as an intra-coding method.

6.3 Rendering-Oriented Decoding

The rendering method described in Section 6.2.2 is applicable if all reference regions are reconstructed and available. Therefore, as shown in Fig. 6.2 (a), typical methods first reconstruct the

multi-view images by using the decoding method described in Section 6.2.1, and then perform rendering using the reconstructed images. However, they seem to pay an unnecessary computational cost, because disparity compensation in the decoding process and geometry estimation in the rendering process are essentially equivalent if the camera parameters for the multi-view images are known, and not all the reconstructed images are used for the rendering.

To synthesize a desired view directly, we propose rendering-oriented decoding method, in which the decoding of the Wyner-Ziv images is incorporated into the rendering process, as shown in Fig. 6.2 (b). The Wyner-Ziv images are therefore not reconstructed explicitly, and only the reference light rays in the Wyner-Ziv images are reconstructed implicitly in the rendering process. Our method uses a simple coset code for the Wyner-Ziv images. As with a conventional intra-coding method, it keeps both the encoder and decoder low-complexity.

6.3.1 Rendering Method with a Coset Code

The input multi-view images are divided into key images and Wyner-Ziv images. At the encoder, the key images are encoded using a conventional intra-image coder. For the Wyner-Ziv images, each RGB value of a pixel is represented by M cosets, $C_m (m = 1, 2, .., M)$, in a memoryless fashion [89].

At the decoder, we first reconstruct the key images and coset indices for the Wyner-Ziv images. The side information for each target light ray and each depth layer, $Y(\boldsymbol{x}, z)$, is then calculated by interpolating the colors of the reference light rays in the key images as follows:

$$Y(\boldsymbol{x}, z) = \frac{\sum_{i \in V_K} w_i(\boldsymbol{x})\, I(\boldsymbol{r}_i(\boldsymbol{x}, z))}{\sum_{i \in V_K} w_i(\boldsymbol{x})}. \tag{6.6}$$

Here, V_K is the set of camera indices for the key images in the reference camera set V. This side information is used to reconstruct the reference light rays of near Wyner-Ziv images in a maximum likelihood sense by

$$\hat{I}(\boldsymbol{r}_i(\boldsymbol{x}, z)|_{i \in V_W}) = \arg \min_{c_j \in C_{m,q}} (c_j - Y_q(\boldsymbol{x}, z))^2|_{q \in \{R,G,B\}}, \tag{6.7}$$

where V_W is the set of camera indices for the Wyner-Ziv images in V, and c_j is a codeword in the coset $C_{m,q}$ of the light ray $\boldsymbol{r}_i(\boldsymbol{x}, z)|_{i \in V_W}$ for each RGB component q. This equation means that our method only reconstructs the reference light rays in the Wyner-Ziv images. We then evaluate the color consistency cost of the reconstructed reference light rays (Eq. (6.2)), smooth the cost (Eq. (6.3)), and estimate the depth and color for each target light ray (Eqs. (6.4) and (6.5)). Since the extra computational cost for Eqs. (6.6) and (6.7) is not too high, we can keep the complexity of this rendering method as low as that of the original one described in Section 6.2.2. In the experiments, we arranged the key images and Wyner-Ziv images as shown in Fig. 6.1; therefore, $|V_K| = |V_W| = 2$ for all target light rays.

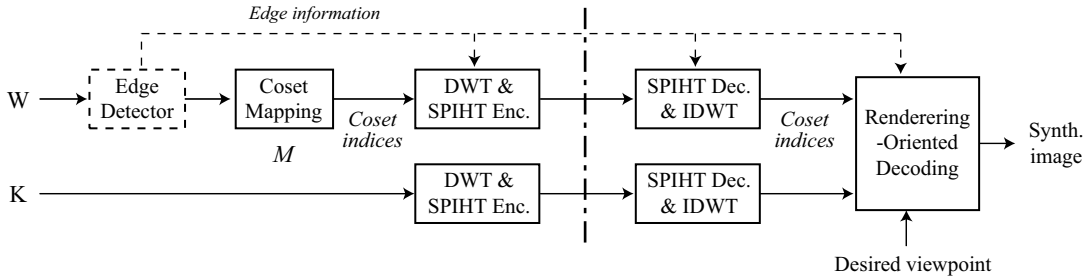


Figure 6.3: Implementation diagram.

6.3.2 Improving Coding Efficiency by Using Edge Information

When the side information for the Wyner-Ziv images is generated, smooth regions can be easily predicted, while edge regions are difficult to predict because of occlusions. In other words, the predicted color (side information) given by Eq. (6.6) is accurate enough in the smooth regions, but it includes a larger error in the edge regions [48]. We therefore use an algorithm that performs the coset decoding only in the edge regions and uses the predicted color itself as the interpolated color in the smooth regions. This reconstruction algorithm is described as follows:

$$\hat{I}(\boldsymbol{r}_i(\boldsymbol{x},z)|_{i\in V_W}) = \begin{cases} \arg\min\limits_{c_j \in C_{m,q}} (c_j - Y_q(\boldsymbol{x},z))^2|_{q\in\{R,G,B\}} & \\ \qquad\qquad \text{if } \boldsymbol{r}_i(\boldsymbol{x},z) \text{ is in edge regions} & \\ Y(\boldsymbol{x},z) \qquad \text{otherwise} & \end{cases} \tag{6.8}$$

The encoder only needs to send coset indices that correspond to edge regions of the Wyner-Ziv images, as well as mask information that indicates the position of the edge regions. This algorithm therefore improves coding efficiency.

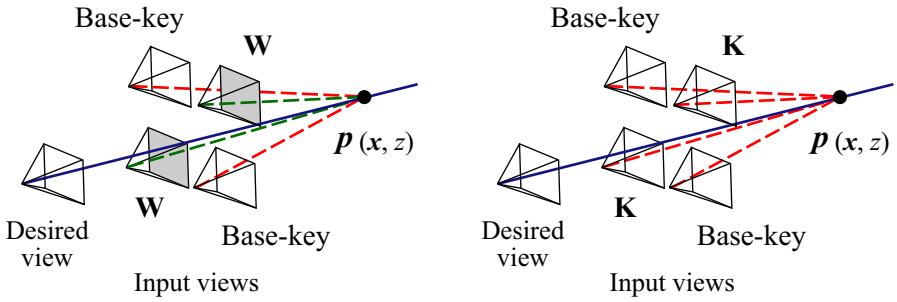
6.3.3 Implementation

Figure 6.3 shows the implementation diagram of our method. We encode the key images by using a standard intra-image coder consisting of discrete wavelet transform (DWT) and SPIHT [93] * for each RGB component. For the Wyner-Ziv images, we first map each RGB value of a pixel, v_q, to a coset $C_{m,q}$ by the following function:

$$C_{m,q} = \begin{cases} v_q \bmod M & \text{if } \lfloor v_q/M \rfloor \text{ is even} \\ M - 1 - (v_q \bmod M) & \text{otherwise} \end{cases} \tag{6.9}$$

The coset indices are then encoded with DWT and SPIHT for each RGB component. Since we use the lossy coder for encoding the coset indices, we choose the above mapping function, instead

*We used the implementation in QccPack [155].



(a) Our method (b) All-key method

Figure 6.4: Methods compared in the experiments. Both methods share *base-key* images encoded in the same way at the same positions. The other images, referred to as *non-base* images, are encoded in different ways.

of the regular modulo M function, to prevent drastic changes in codewords with a small error in the coset index. A similar technique is also used in [9]. At the decoder, we decode the SPIHT and perform the rendering-oriented decoding with the key images and the decoded coset indices of the Wyner-Ziv images. In the experiments, we only set M to numbers to the power of two, which is described as $\bar{M} = \log M$.

For exploiting edge information as described in Section 6.3.2, we implemented a simple edge detector for the Wyner-Ziv images. The Wyner-Ziv images are divided into a set of small rectangular blocks. If the sum of RGB color variances within a block exceeds a threshold, the block is considered as an edge region. The coset indices within the extracted edge regions are encoded by using shape-adaptive SPIHT [78] with a mask image for the edge regions.

6.4 Experiments

Compared to a typical method that performs a straightforward decoding and rendering, as shown in Fig. 6.2 (a), our rendering-oriented decoding method is low-complexity because it does not perform disparity compensation explicitly and does not reconstruct all of the light rays in the Wyner-Ziv images. Instead, our method has a similar complexity to a method that encodes all input images as the key images and synthesizes a novel image with a normal renderer described in Section 6.2.2, which is referred to as *all-key method*. In the following experiments, we therefore compare the coding performance and processing time of these two methods, as shown in Fig. 6.4.

We used two different types of input image sets, as shown in Figs. 6.5 and 6.6. The *City* and *Santa* image sets (Fig. 6.5), provided by University of Tsukuba, Japan, are captured by moving a single camera on a control stage, which is an ideal condition for generating accurate side infor-

(a) *City* (b) *Santa*

Figure 6.5: Parts of (a) *City* and (b) *Santa* image sets, which are captured on a regular 2D grid by moving a single camera.

Figure 6.6: Parts of *Meeting room* image set, which are captured with multiple cameras that roughly lie on a 2D grid.

Table 6.1: Specifications of the input image sets and the parameters for the edge detection and rendering used in the experiments.

	City, *Santa*	*Meeting room*
Number of input images	81 (9×9)	64 (8×8)
Resolution of input images	640×480	320×240
Edge detection block size	32×32	16×16
Edge detection threshold	200	200
Resolution of synthesized images	640×480	300×300
Number of depth layers (N)	20	15
Smoothing window size (S)	15×15	11×11

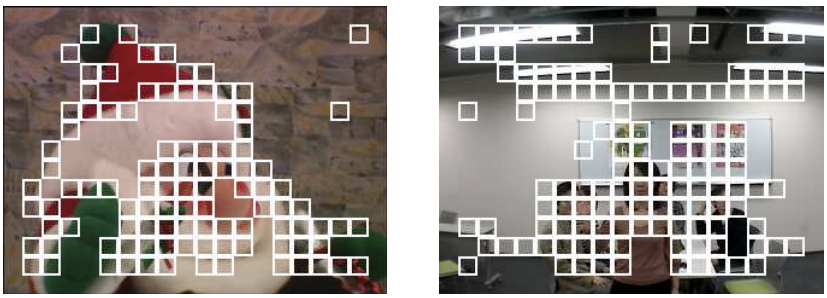
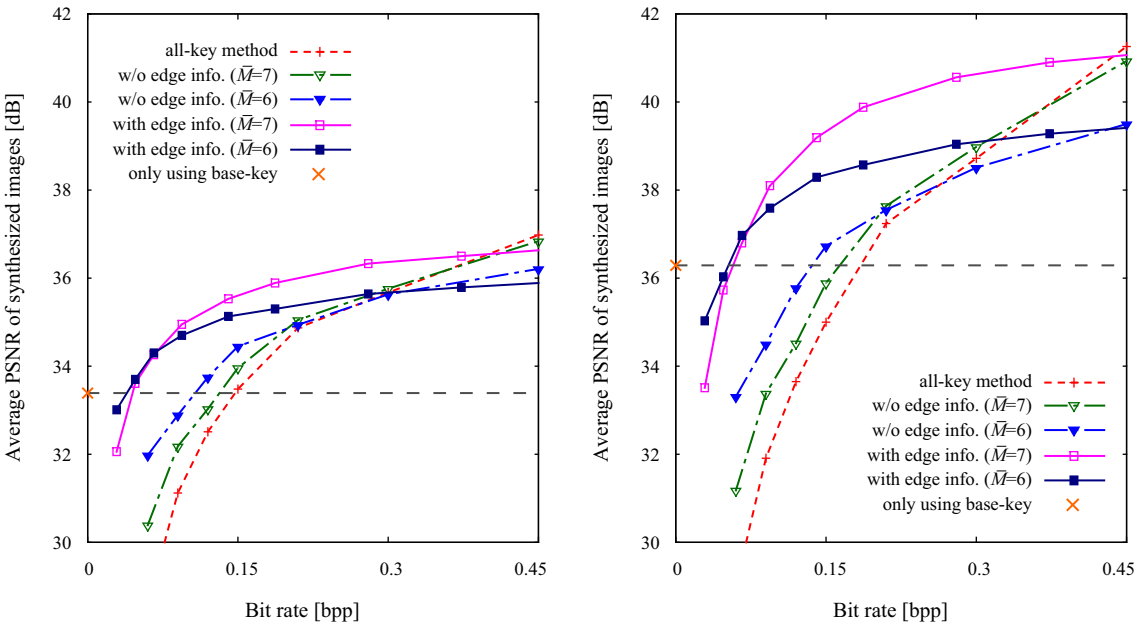


Figure 6.7: Extracted edge regions in an input image of the *Santa* (left) and *Meeting room* (right) image sets.

mation. Since they are captured on a regular 2D grid with a fixed camera pose, we used a simple geometry for calculating the position of the reference light rays in the input images. On the other hand, the *Meeting room* image set (Fig. 6.6) is captured with our 64-camera array presented in Chapter 7, which corresponds to a more practical situation. The image set has large color variations due to individual differences between cameras, and some of them suffer from lens blur. We performed geometry calibration of the cameras by using Tsai's method [114]. For the *Meeting room* image set, we implemented our rendering-oriented decoding method and the all-key method on a GPU (described in Section 6.4.2 in detail) and measured the coding performance and processing time using the GPU implementations. Table 6.1 summarizes the parameters used in the following experiments, and Fig. 6.7 shows some examples of the edge regions extracted with these parameters.



(a) Using lossy base-key images (0.45 bpp, 35.77 dB)

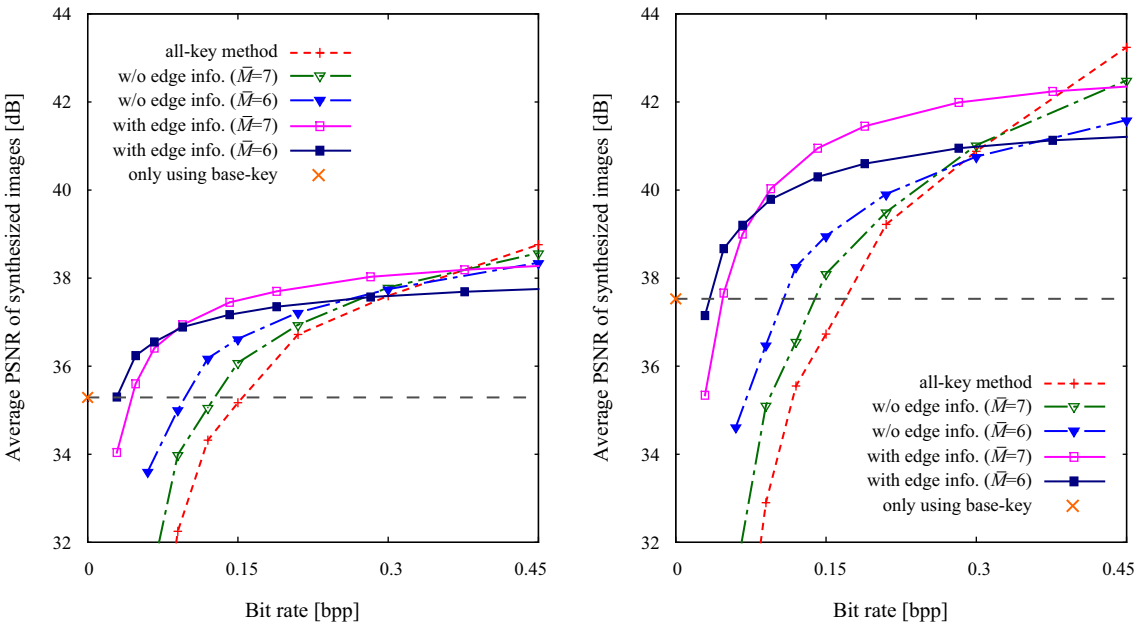
(b) Using lossless base-key images

Figure 6.8: Rate-distortion curves for the *City* image set obtained using (a) lossy and (b) lossless base-key images. The bit rate of the lossy base-key images was 0.45 bpp and their average quality was 35.77 dB.

6.4.1 Coding Performance

As shown in Fig. 6.4, we divided input images into *base-key* images and the other (*non-base*) images. The base-key images were identical in both our method and the all-key method; they were encoded by using DWT and SPIHT or assumed to be losslessly available for comparing the influence of the quality of the base-key images on the rendering quality. The non-base images were encoded as Wyner-Ziv images in our method, as shown in Fig. 6.3, while as key images in the all-key method. The only difference between the two encoding methods is therefore whether they use the coset mapping and edge detection or not. In the experiments, the bit rate of the base-key images was fixed, while that of the non-base images was controlled by truncating the SPIHT bitstream.

Figures 6.8, 6.9, and 6.10 plot the rate-distortion performance of our method either with or without the edge detector (our method without the edge detector encodes the coset indices in all regions of the Wyner-Ziv images) and that of the all-key method for different image sets, obtained using lossy and lossless base-key images. The plots show the reconstruction quality of



(a) Using lossy base-key images (0.45 bpp, 36.75 dB)

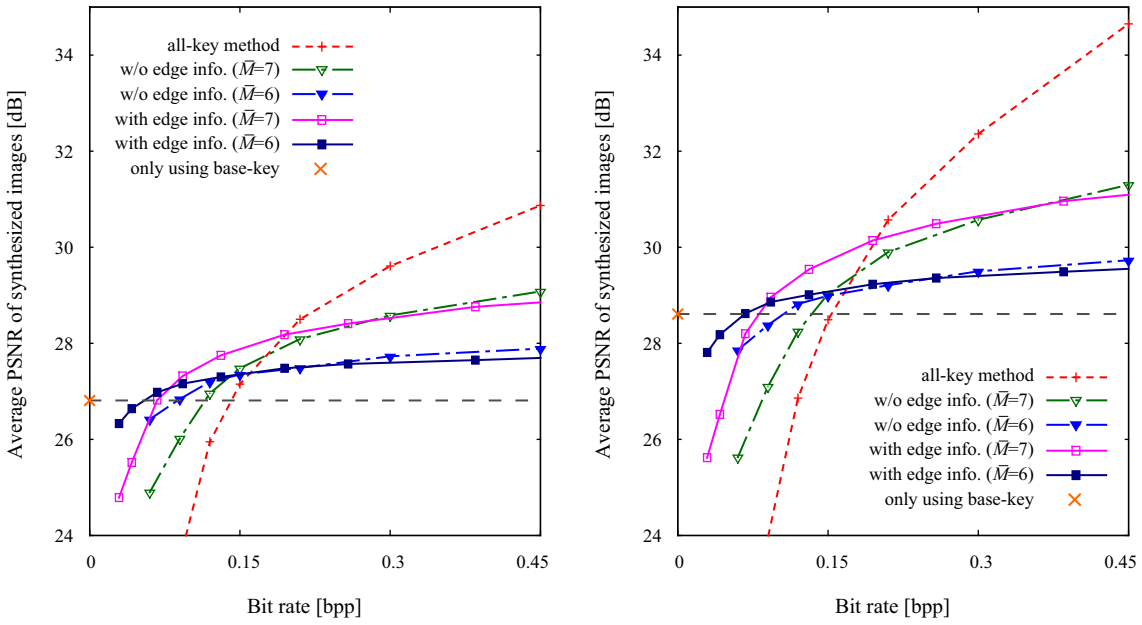
(b) Using lossless base-key images

Figure 6.9: Rate-distortion curves for the *Santa* image set obtained using (a) lossy and (b) lossless base-key images. The bit rate of the lossy base-key images was 0.45 bpp and their average quality was 36.75 dB.

synthesized images averaged for 10 random viewpoints (except the original viewpoints of the key and Wyner-Ziv images), where the quality is calculated with respect to the image synthesized from the uncompressed data and expressed as peak signal-to-noise ratio (PSNR). The bit rate of the non-base images is expressed on the horizontal axis. The bit rate of edge information is included in the plots of our method using it.

As it can be seen from the plots, our method shows superior coding performance compared to the all-key method especially at low bit rates. Smaller $\bar{M}$ yields better performance at low bit rates, because small errors in the smooth regions can be corrected by a coset code with small $\bar{M}$, but it restricts the maximum quality which is important at high bit rates. As for our method, the edge information provides additional gain at low bit rates, since the edge regions include larger errors than the smooth regions. When comparing the results obtained using the lossy and lossless base-key images, we can see that all of the methods similarly benefit from the increase of the quality of the base-key images, and the shapes of the rate-distortion curves maintain their relationship to each other regardless of the quality of the base-key images.

The plot “only using base-key” in each graph shows the reconstruction quality when we ren-



(a) Using lossy base-key images (0.45 bpp, 29.23 dB)

(b) Using lossless base-key images

Figure 6.10: Rate-distortion curves for the *Meeting room* image set obtained using (a) lossy and (b) lossless base-key images. The bit rate of the lossy base-key images was 0.45 bpp and their average quality was 29.23 dB.

der the novel image by using the base-key images only (i.e., the bit rate of the non-base images is zero). In this case, the color is interpolated in the same way as for generating the side information (Eq. (6.6)), and the color consistency cost is calculated as the sum of absolute difference of the reference light ray's colors in the base-key images. This reconstruction quality therefore corresponds to the quality of the side information without error correction. At very low bit rates, our method and the all-key method produce lower-quality images than the side information (under the dashed line). This means that the novel images synthesized at those bit rates are negatively affected from the reconstructed low-quality non-base images.

This negative effect can be explained with the reconstructed synthesized images and their error images (difference from the synthesized image obtained using uncompressed data), as shown in Figs. 6.11 and 6.12. Here, we used lossless base-key images and set the bit rate of the non-base images to 0.15 bpp for all methods. If we only use the base-key images, many of the errors appear in the edge regions; in particular, some large structure errors can be seen in those regions (e.g., the bottom-left building in Fig. 6.11 (a) and around the head of the candle in Fig. 6.12 (a)). The all-key method produces larger errors in the smooth regions than the rendering method only using

(a) Only using base-key
36.91 dB

(b) All-key method
35.51 dB

(c) Ours w/o edge info. ($\bar{M}$=7)
36.49 dB

(d) Ours with edge info. ($\bar{M}$=7)
39.79 dB

Figure 6.11: Synthesized images and their difference from that obtained using uncompressed data (multiplied by 8) for the *City* image sets.

(a) Only using base-key
38.74 dB

(b) All-key method
36.52 dB

(c) Ours w/o edge info.
($\bar{M}$=7)
38.73 dB

(d) Ours with edge info.
($\bar{M}$=7)
42.16 dB

Figure 6.12: Synthesized images and their difference from that obtained using uncompressed data (multiplied by 8) for the *Santa* image sets.

(a) All-key method

27.16 dB

(b) Ours with edge info. ($\bar{M}$=7)

29.47 dB

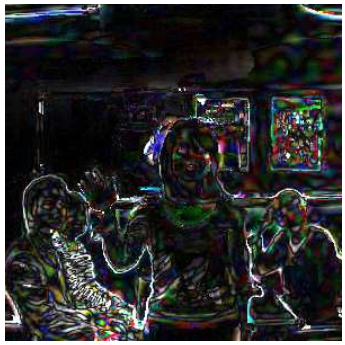


Figure 6.13: Synthesized images and their difference from that obtained using uncompressed data (multiplied by 8) for the *Meeting room* image set.

the base-key images (e.g., the top-right part (background) in Fig. 6.11 (b)), because it synthesizes the interpolated color with the low-quality non-base images. The resulting images look blurred, as shown in Figs. 6.11 (b) and 6.12 (b). Our method without edge information also produces the errors in the smooth regions, but has better PSNR than the all-key method (Figs. 6.11 (c) and 6.12 (c)). Our method with edge information provides the best reconstruction quality, where the smooth regions keep high quality as using the base-key images only, and errors in the edge regions are reduced (Figs. 6.11 (d) and 6.12 (d)). The synthesized images obtained using the *Meeting room* image set, depicted in Fig. 6.13, also show similar results: the all-key method produces too blurred images, while our method with edge information produces higher-quality images.

6.4.2 Processing Time

To compare the processing times of our method and the all-key method, we implemented the two methods on a GPU. For the all-key method, we used the GPU implementation of the rendering algorithm presented in Chapter 7, since all the input images are reconstructed and available before rendering. For the rendering-oriented decoding method, we modified the GPU implementation so

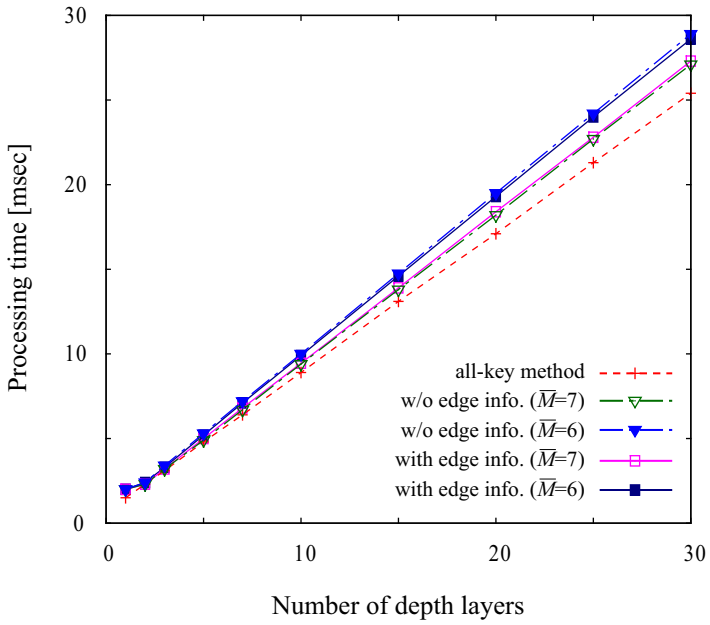


Figure 6.14: Processing time for different numbers of depth layers.

that it can perform coset decoding before evaluating the color consistency of reference light rays. The reconstructed coset indices in the Wyner-Ziv image are uploaded to the GPU texture memory as a texture in the RGB channels, as well as the reconstructed key images. When we use edge information, the edge mask for each Wyner-Ziv image is also uploaded as a texture in the alpha channel together with the coset indices in the RGB channels. We used OpenGL and fragment programs with Cg [152] for the GPU implementation. The measurements were performed on an Intel Xeon 5160 (3 GHz) dual processor machine with 3 GB main memory and an NVIDIA GeForce 8800 Ultra graphics card.

Figure 6.14 shows the processing time versus the number of depth layers for our method and the all-key method. We measured the average processing time for 100 executions of both rendering methods for the *Meeting room* image set. The processing time only includes the coset decoding and rendering processes; that is, the key images and the coset indices in the Wyner-Ziv images were decoded and uploaded to the GPU texture memory before rendering.

The processing time of our rendering-oriented decoding method is proportional to the number of depth layers. This result is the same as that in the case of the original rendering method, which is used for the all-key method. The processing times of our methods with $\bar{M} = 6$ and 7 are different. This is because we only need to check two candidates in coset decoding for $\bar{M} = 7$, while we need to check $2^{(8-\bar{M})}$ candidates (or determine which two candidates should be evaluated based

on the higher-order bits of the side information) for $\bar{M} < 7$, resulting in higher complexity. The difference between our method and the all-key method is small: our method takes about 7% and 14% more processing time than the all-key method for $\bar{M} = 7$ and 6, respectively. When our method uses edge information, the processing time becomes slightly faster than that without edge information for $\bar{M} = 6$, because we do not need to correct the reference light rays that are not in the edge regions. On the other hand, the processing time becomes slightly slower for $\bar{M} = 7$, because there are only two candidates for the coset decoding and checking if the reference light ray is in the edge regions causes an overhead.

6.4.3 Discussion

The experimental results show that our method has better coding performance than the all-key method especially at low bit rates, while performing the decoding and rendering as fast as the all-key method. In particular, the coding performance for the *City* and *Santa* image sets shows a clearer advantage of our method than that for the *Meeting room* image set, because the former image sets are suitable for generating accurate side information. Although the *Meeting room* image set has large color variations among input images, which makes it difficult to generate accurate side information, our method still provides higher quality than the all-key method at low bit rates. In such a case, incorporating a color compensation method among input views (e.g. [51, 128]) into the decoding algorithm could help improve coding efficiency.

The experimental results also show that, at very low bit rates, the rendering method only using base-key images provides higher quality than our method and the all-key method. This means that we can choose an appropriate rendering method depending on the bit rate: the rendering method only using base-key images at very low bit rates, our method with the edge detector and a proper number of cosets ($\bar{M}$) at low and medium bit rates, and the all-key method at high bit rates. Since we do not use a feedback channel to control the bit rate of the Wyner-Ziv images [1, 40], to determine the proper number of cosets at the encoder is still difficult and it would be an interesting future work.

Our rendering-oriented decoding method has the same feature of the original rendering method; that is, the processing time is proportional to the number of depth layers and target light rays. This is because the coset decoding (Eqs. (6.6)–(6.8)) can be performed for each target light ray in a desired view, as well as the original rendering process (Eqs. (6.2)–(6.5)). This feature is suitable for implementing the decoding and rendering processes all on a GPU, because the GPU can efficiently perform the same instructions for all the target pixels in parallel. Thanks to this implementation, our rendering-oriented decoding is fast enough for real-time processing as well as the original rendering method. As we present in Chapter 7, we have developed a camera array system that enables real-time video-based rendering with the original rendering method. Therefore,

if the cameras have a function that maps pixel values to coset indices and encodes them with an intra-image coder (e.g., the Axis 210 camera we used for the camera array has a built-in JPEG encoding function), we could construct a system that performs real-time video-based rendering with improved synthetic quality.

Since our method uses a simple memoryless coset code, it would have worse coding performance than modern distributed multi-view coding methods that use more sophisticated coding tools (e.g., high-performance channel codes [91, 116] and transform-domain encoding [1, 40, 46, 48]). Therefore, our method would have worse coding performance than conventional methods that perform disparity-compensated prediction at the encoder, which typically have better coding performance than the modern distributed coding methods. However, for the scenario described in this chapter (rendering a novel view from encoded data), our method has a clear advantage in computational cost as follows. The conventional method that performs disparity compensation at the encoder needs to separately perform geometry estimation at the decoder for rendering a novel view; there is no way to jointly perform these two processes because the encoder and decoder are separated. The typical distributed multi-view coding method performs disparity compensation at the decoder, but still separately performs geometry estimation at the decoder for the rendering, as shown in Fig. 6.2 (a). Our method, by contrast, jointly performs disparity compensation and geometry estimation at the decoder, which can make the total computational cost of the encoder and decoder lower than the above two methods.

We compared the coding performance of our method and the all-key method at novel viewpoints, instead of at the viewpoints of the Wyner-Ziv images, because of the following two reasons: (1) To our knowledge, all existing works about distributed multi-view coding focus on reconstructing the Wyner-Ziv images; they therefore measure the reconstruction quality at the viewpoints of the Wyner-Ziv images. However, for the free-viewpoint rendering scenario described in this paper, it is more natural to select novel viewpoints that are different from the original viewpoints of the key and Wyner-Ziv images. (2) Image-based rendering techniques tend to produce images having low PSNR (this does not necessarily mean low visual quality), when we compare the rendered image with the image captured by an actual camera. This is because they do not correctly synthesize view-dependent effects, such as specular components and occluded regions in the scene. Therefore, if we evaluate the reconstruction quality in PSNR at the original viewpoints of the Wyner-Ziv images, our method, which uses an image-based rendering method for reconstructing the images, has a disadvantage compared to the all-key method, which uses the encoded key images themselves as the reconstructed images. If we evaluate the quality at novel viewpoints, as we did in this paper, the disadvantage is avoided, because both our method and the all-key method use an image-based rendering method for the reconstruction and the reference images are also synthesized with the same image-based rendering method (i.e., the view-dependent effects decrease in

both the reference images and the reconstructed images).

6.5 Conclusions

In this chapter, we have presented rendering-oriented decoding method for a distributed multi-view coding system using a coset code. By incorporating the reconstruction of reference light rays in the Wyner-Ziv images into the rendering process, our method directly synthesizes a novel image without reconstructing all the Wyner-Ziv images explicitly. Our method keeps both encoder and decoder complexity as low as that of a conventional intra-coding method, while attaining better coding performance especially at low bit rates. Our future work will be focused on finding a way to incorporate the rendering-oriented decoding method into a real-time video-based rendering system.

Chapter 7

Real-Time All-in-Focus Video-Based Rendering Using a Network Camera Array

7.1 Introduction

Image-based rendering (IBR) has attracted a lot of research interest, because photorealistic rendering quality is affordable by using a set of images of a 3D scene captured from multiple viewpoints. IBR is based on a framework in which visual information of a 3D scene is represented as a collection of light rays, such as the 7D plenoptic function [2] and 4D parameterizations using planes called ray space [29] and light field [39, 59]. Early IBR techniques often used static multi-view images captured by moving a single camera. More recently, many camera array systems have been developed with video-based rendering techniques to handle dynamic 3D scenes and produce interactive rendering applications, commonly called free-viewpoint video and 3D TV.

This chapter presents the design and implementation of a system that renders a free-viewpoint video in real time from the live multi-view videos captured with a network camera array. As shown in Fig. 7.1, our system consists of 64 (8×8) commodity network cameras that are connected to a single PC through a gigabit Ethernet. The cameras are mounted on a mobile cart so that we can easily move them to capture various scenes. We present an on-the-fly depth estimation method to synthesize high-quality novel views. Using a layered representation, our method reconstructs a view-dependent per-pixel depth map based on a color consistency measure as well as a temporal smoothness constraint. The rendering algorithm is fully implemented on the GPU using GPGPU (General-Purpose computation on GPUs) techniques. This approach has the following advantages: 1) the GPU is suitable for parallel processing of the same instructions for each pixel, which accelerates our rendering algorithm; and 2) the software can use CPU and GPU independently and in parallel for real-time processing. Using QVGA (320×240) input video resolution, our system en-

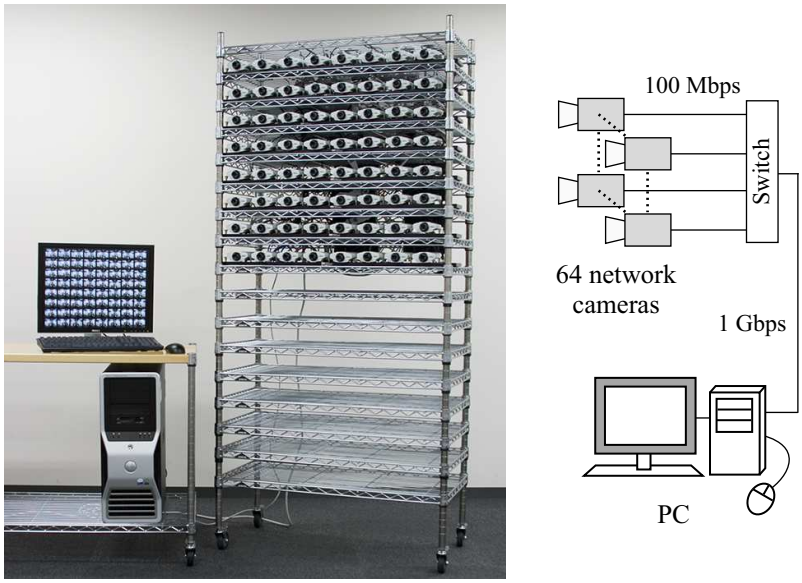


Figure 7.1: Our camera array system consists of 64 (8×8) network cameras connected to a single PC through a gigabit Ethernet. The array can be easily moved on a mobile cart.

ables a rendering rate of up to 30 frames per second (fps) depending on the output video resolution and the number of depth layers.

7.2 Related Work

This section reviews prior multi-view video capturing systems and their rendering algorithms, and identifies our contribution.

One of the pioneering studies in this area is Kanade et al.'s Virtualized Reality project [50], in which 51 cameras were mounted on a 5-meter geodesic dome. Systems using such a sparse, circular camera arrangement basically aim at rendering objects inside the capturing volume, and they often reconstruct voxel models by using the silhouette of the objects as well as color consistency between views [35, 36, 56, 57, 73, 117, 118]. Einarsson et al. [28] presented a dome-type system that captures cyclic human motion from multiple viewpoints under a sequence of controlled lighting conditions. The system enables rendering of objects under variable illumination (image-based relighting) as well as from variable viewpoints.

On the other hand, the following camera arrays (including ours) aim at capturing whole scenes by using a relatively dense, planar camera arrangement. Wilburn et al. [123, 124] developed 100 custom video cameras that have accurate timing control using a trigger signal. Using multi-view videos captured by their first prototype, Goldluecke et al. [37] presented warping-based dynamic

light field rendering. In [123], Wilburn et al. presented a spatiotemporal optimal sampling method and an optical flow algorithm to improve view interpolation quality. They also presented several high-performance imaging applications, such as high-speed imaging [122] and synthetic aperture photography [115]. Zitnick et al. [147] arranged eight cameras along a 1D arc and presented high-quality view interpolation using a sophisticated stereo reconstruction method followed by matting at object boundaries. Tanimoto et al. presented a large camera array system with 100 high-definition video cameras [30, 110] and a view interpolation method based on view-dependent depth estimation [31]. Ng et al. [82] developed a hand-held plenoptic camera, in which a microlens array is placed in front of the photosensor. Using the captured static (not video) light fields, they performed refocusing and all-in-focus rendering by producing refocused images at multiple depths and gathering in-focus parts from them. Their work inspired many researchers to develop light field cameras using microlens arrays or masks with conventional 2D cameras [32, 60, 119]. All of the systems described in this paragraph need to process the captured data offline before interactive rendering, because they use complex geometry reconstruction algorithms or handle a large amount of data.

In contrast to the above offline rendering systems, the systems described in the next paragraph perform capturing and rendering all in real time. Such online rendering systems typically use simpler geometry reconstruction and rendering algorithms than the offline rendering systems for real-time processing. Our goal is to develop an online rendering system that renders higher-quality novel views and has higher system performance than the existing systems.

Yang et al.'s distributed light field camera [131] performed real-time rendering at a rate of 18 fps using 64 FireWire cameras (the camera capture rate was 15 fps). Since their rendering method approximates the scene geometry as a single plane, their system produces low-quality synthesized images in which only the objects at the depth of that plane are clear (in-focus) and the objects at other depths are blurred or appear with ghosting artifacts [14, 45]. This single-plane rendering would produce enough results if we use light field data densely sampled with a small camera interval [14]. However, because the cameras cannot be arranged enough densely in practice, we need to estimate the scene geometry (e.g., depth maps) for higher-quality rendering. Schirmacher et al. [95] used an array of six FireWire cameras and generated views at 1–2 fps with dense depth maps estimated from the stereo camera pairs, but the rendering quality was limited due to wrong depth reconstruction. Zhang and Chen [139] presented a self-reconfigurable camera array using 48 network cameras, each of which can move sideways and pan using servo motors. They estimate depth values only on a multi-resolution 2D mesh for rendering novel views at 4–10 fps. Our method, by contrast, estimates per-pixel depth maps to improve rendering quality. Using commodity graphics cards, Yang et al. [134] presented an efficient GPU implementation of a depth estimation method using plane-sweeping [24, 96] and view synthesis. Their rendering

rate was 15 fps using five input cameras. We use similar, but more recent GPGPU techniques to perform depth estimation and rendering fully on a GPU. Moreover, they use all input cameras evenly as reference cameras for the depth estimation and color interpolation. Their approach is reasonable since their system has only five cameras. However, such an approach is not suitable for directly applying to a larger camera array system consisting of a larger number of cameras in a larger spatial arrangement, because its computational complexity increases with the increase of input cameras and it is more likely to be subject to occlusions due to the larger baseline. Our method, by contrast, uses only neighboring input cameras of each target light ray as the reference cameras, which can keep the computational complexity constant regardless of the number of the input cameras and make the occlusion effects small.

Finally, we summarize the previous work by our research group. For real-time capturing and rendering, Naemura et al. developed an array of 16 cameras [80] with hardware specialized to estimate depth maps in real time [81]. They render novel views at 10 fps by approximating the depth maps with three layers. Our system described in this chapter does not require such additional hardware. Yamamoto et al. [129, 130] developed a system called LIFLET that captures a 3D scene through a micro-lens array with an XGA video camera. The captured image, called integral photography, includes thousands of different viewpoint images. They estimate a single depth value for each viewpoint image and render novel views at 15 fps. Although their system records dense light fields thanks to the dense microlens array, the capturing volume is relatively small. Takahashi et al. [107, 108, 109] presented a layer-based rendering algorithm that generates images and their corresponding costs (what they call focus measure) at each depth layer, detects in-focus parts from the images by evaluating the cost, and synthesizes an all-in-focus novel view. Their rendering rate was 13.9 fps from 81 static input views without offline processing. Our rendering algorithm is based on this layer-based approach, but we use 1) a more straightforward measure (i.e. variance) for evaluating the similarity of light rays, instead of their focus measure; and 2) a temporal smoothness constraint to make the depth estimation more stable. We also show a complete system implementation of the rendering algorithm from live videos, while they use static multi-view data sets.

7.3 Rendering Algorithm

7.3.1 View-Dependent Depth Estimation

We assume that multi-view videos are captured with calibrated cameras that roughly lie on a plane and are arranged on a 2D grid, and that there is no prior knowledge of the scene geometry. As described in Section 7.2, because the cameras cannot be arranged enough densely in practice, we need to estimate the scene geometry (depth), rather than approximating it as a single plane, for

synthesizing novel views without blur and ghosting artifacts [14, 45].

For synthesizing high-quality views, we use the layer-based rendering method described in Section 2.1.4. The method assumes a layered depth model and estimates the depth of each target light ray in the rendered view. At the intersection of the target light ray with each of the depth layers, it first computes color consistency cost among reference light rays, which correspond to the back projections of the intersection point to reference cameras. We use as the reference cameras the four-nearest input cameras for each target light ray to keep the computational cost low and make occlusion effects small, and use as the color consistency measure the sum of variances for each RGB component. The method then smoothes the cost in each depth layer by using an average filter (we use an 11×11 window in the experiments in this chapter), and finally assigns the optimal depth value that has the minimum cost for each target light ray. The color of each target light ray is generated by using bilinear interpolation of the colors of the reference light rays corresponding to the estimated depth value. As we describe in Section 7.4.4, the rendering method is efficiently implemented on the GPU, because all of the rendering processes are independently performed on each light ray (pixel) in the rendered image.

7.3.2 Temporal Smoothness Constraint

If we use the above algorithm to estimate depth maps independently for each time frame, the estimate becomes unstable, especially in textureless regions, due to camera noises. This causes flickering artifacts on synthesized videos. To reduce the artifacts and make our depth estimation stable, we incorporate a temporal smoothness constraint into the cost function. Given the depth estimate at the previous time frame, $z_{opt}^{t-1}(\boldsymbol{x})$, this constraint is described as

$$\hat{C}(\boldsymbol{x}, z) = \begin{cases} \bar{C}(\boldsymbol{x}, z) & \text{if } z^t(\boldsymbol{x}) = z_{opt}^{t-1}(\boldsymbol{x}) \\ \bar{C}(\boldsymbol{x}, z) + \lambda & \text{otherwise} \end{cases} , \tag{7.1}$$

where λ is a small positive constant. We set it to 40 in the experiments. The minimum cost search is then applied to $\hat{C}(\boldsymbol{x}, z)$, instead of Eq. (2.5). This constraint encourages that the current depth is similar to the previous depth, while it allows for large depth changes (i.e. motion) because the penalty λ is equal for all depths with $z^t(\boldsymbol{x}) \neq z_{opt}^{t-1}(\boldsymbol{x})$. If the rendering viewpoint smoothly moves (as is often the case in interactive rendering applications), we can still use this constraint because the depth map should change smoothly. If the rendering viewpoint dynamically changes, we can simply ignore this constraint.

7.4 Implementation

7.4.1 System Setup

As shown in Fig. 7.1, our camera array consists of 64 (8×8) Axis 210 network cameras. The distance between cameras is about 100 mm both horizontally and vertically. The camera employs a built-in HTTP server, which sends motion JPEG sequences in response to HTTP requests from clients. Its capture rate is up to 640×480 pixels at 30 fps. The JPEG compression factor can be adjusted between 0 (the best quality) and 100 (the worst quality). In our experiments, we set the image resolution to 320×240 and the JPEG compression factor to 10. This compression factor was enough to prevent visible compression artifacts such as blocking. In practice, we chose the relatively high-quality factor because the network bandwidth was not a bottleneck in our system.

The cameras have 100 Mbps Ethernet ports. We connected them to a single PC using gigabit Ethernet switches. We used an Intel Xeon 5160 (3 GHz) dual processor machine with 3 GB main memory and an NVIDIA GeForce 8800 Ultra graphics card.

7.4.2 Camera Calibration

For geometric calibration of the cameras, we tried two standard methods proposed by Tsai [114] and Zhang [144]. Although Zhang's method can be easily used by capturing a checkerboard pattern at several arbitrary positions, we found that the calibration parameters computed by the method produced good rendering results only within the volume where the checkerboard was placed during the calibration, as pointed out in [115]. For our camera array system that aims to capture and render a large space, Tsai's method, which requires known 3D geometry of a calibration object, worked better. We therefore used Tsai's method by capturing a checkerboard pattern at several depth positions with known translations.

We did not use color calibration and only relied on automatic white balance and exposure control of each camera, which is a similar setting to [139]. This is because the cameras do not have the flexible control of capturing parameters. In Section 7.5.1, we show that our method synthesizes visually good images even from input images that have large variations.

7.4.3 Software Architecture

Figure 7.2 shows the software architecture and data flow of our system. The system performs network I/O (receiving 64 motion JPEG sequences from cameras) and JPEG decoding in parallel on the CPU. The decoded images are uploaded to the GPU texture memory. The following rendering process is fully implemented on the GPU, so there is no data transmission from the GPU to the CPU. This allows our system to efficiently perform the processes as a pipeline by using the CPU and GPU independently.

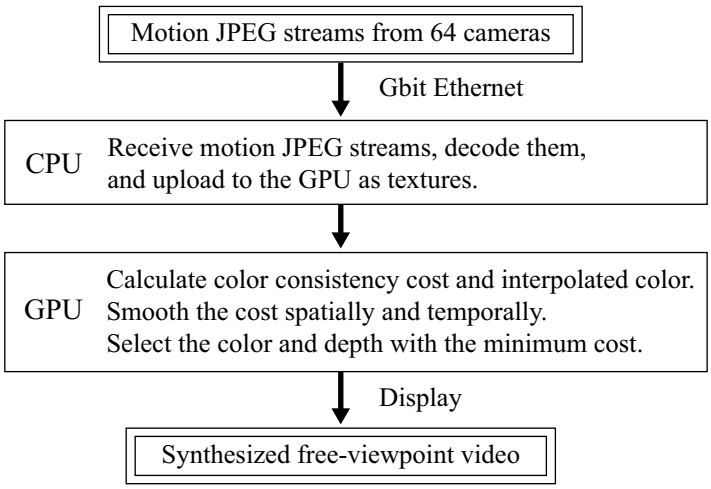


Figure 7.2: Software architecture and data flow. No need for data transmission from the GPU to the CPU allows our system to efficiently perform the processes as a pipeline by using these processors independently.

The Axis 210 cameras do not have a synchronization function. Our rendering process therefore uses the most recently uploaded images. This approach mostly works well because our method interpolates the colors with minimum variance in any case and we use a relatively high frame rate.

7.4.4 GPU Implementation of Rendering

We implemented the rendering algorithm using OpenGL and fragment programs with Cg (C for graphics) [152]. For rendering a novel view, we first calculate the color consistency costs (Eq. (2.3)) and the interpolated colors for each depth layer in a single rendering pass. The interpolated colors are therefore generated for all depth values, instead of only for the optimal depth values (Eq. (2.6)), for computational efficiency. By using the currently uploaded input image textures, we perform projective texture mapping with a fragment program that calculates the interpolated colors and the costs at the same time, and stores them in the RGB channel and the alpha channel of the GPU frame buffer, respectively, as in [134]. For the color interpolation, Yang et al. [134] use the average color of reference light rays. Our method uses bilinear interpolation, which provides better rendering quality, by using textures that give weights to each reference light ray ($w_i(\boldsymbol{x})$ in Eq. (2.6)). Note that too large cost values are automatically truncated when the calculated colors and costs are rendered to the GPU frame buffer, because the frame buffer limits the calculated floating-point values to a range between 0 and 1. This suppresses noise and outliers for the following smoothing operation.

In the next rendering pass, the costs in each depth layer are spatially and temporally smoothed

(Eqs. (2.4) and (7.1)), and then the optimal depth for each pixel are selected by comparing the smoothed cost with the current optimal cost (Eq. (2.5)). To apply the same instructions to each pixel, we set the projection matrix to orthogonal and use texture mapping of the data to be processed, which is a standard GPGPU technique. We use a fragment program that aggregates neighboring alpha values and add λ if the current depth layer is not equal to the previous depth estimate. The current optimal depths are stored in the texture memory together with their costs and are updated by iteration over the depth layers. The optimal color for each pixel is also selected similarly to the optimal depth in another rendering pass. Consequently, our algorithm uses $3N$ rendering passes for rendering a novel view, where N is the number of depth layers.

Pseudocode of the above implementation is listed in Section 7.7.

7.5 Experiments

In this section, we first show rendering results and validate the effectiveness of the depth estimation and bilinear color interpolation. The performance of our rendering algorithm is then evaluated by changing the number of depth layers and target light rays (output resolution). The processing time was proportional to the numbers of layers and target light rays, as discussed in Section 7.3.1, and our system rendered a free-viewpoint video at up to 30 fps depending on those parameters. Throughout the experiments, we used 320×240 pixels as input image resolution and a JPEG compression factor of 10.

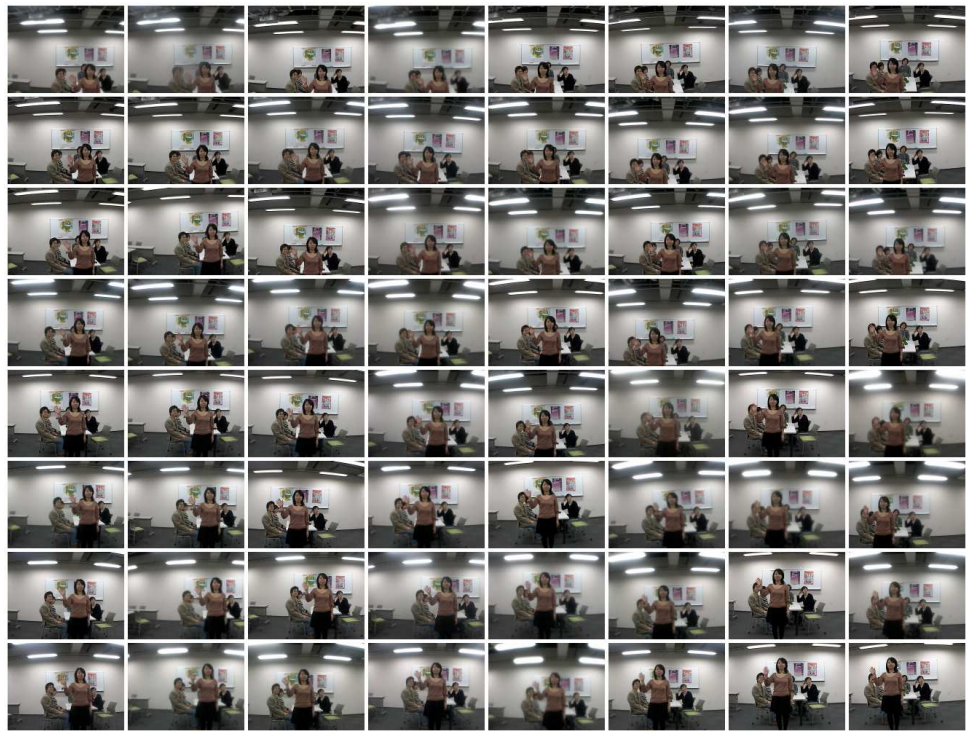
Note that output free-viewpoint videos, as well as comparison videos that show the effectiveness of the temporal smoothness constraint, are available at

http://www.hc.ic.i.u-tokyo.ac.jp/project/camera-array/

7.5.1 Rendering Results

Figure 7.3 shows input and output images from *Meeting room* sequence. We can see correct parallax in the synthesized images from different viewpoints. Objects in the synthesized images are all-in-focus, although some artifacts are still visible as we discuss in Section 7.5.3. Although some estimated depth values are incorrect in textureless regions, the rendered colors are visually correct.

Figure 7.4 compares images synthesized with different numbers of depth layers. The image synthesized with a single depth layer (Fig. 7.4 (a)) suffers from blur and ghosting artifacts except for the left person on which the depth layer is placed. The rendering quality increases with the increase of the number of depth layers. However, as we have reported in [108, 109], the rendering quality gradually reaches a ceiling; that is, the earlier depth layers have a larger impact on the rendering quality. Therefore, the image synthesized with 7 depth layers (Fig. 7.4 (c)) and the one



(a)

(b) (c) (d)

Figure 7.3: Example images from *Meeting room* sequence. (a) Input 64 images. (b–d) Output synthesized images and their corresponding depth maps from various viewpoints.

Figure 7.4: Synthesized images and their corresponding depth maps using (a) a single layer, (b) 3 layers, (c) 7 layers, and (d) 15 layers. Although the rendering quality increases with the increase of the number of depth layers, it gradually reaches a ceiling. Therefore, the synthesized images in (c) and (d) have no significant visual difference.

(a) (b) (c)

Figure 7.5: Comparison of the color interpolation methods. Synthesized images with (a) average interpolation and (b) bilinear interpolation. (c) The camera grid superimposed on the synthesized image, where the intersections of horizontal and vertical lines correspond to the position of input cameras. To render a rectangle region of the grid, four cameras at its corners are used as reference cameras. Average interpolation causes annoying color discontinuities at camera boundaries, while bilinear interpolation generates better-looking images.

synthesized with 15 depth layers (Fig. 7.4 (d)) have no significant visual difference.

As shown in Fig. 7.3 (a), the captured images have large variations due to individual differences between cameras. Therefore, as shown in Fig. 7.5 (a), average color interpolation produces annoying color discontinuities at the reference camera boundaries shown in Fig. 7.5 (c). Our method using bilinear interpolation, by contrast, produces better-looking images (Fig. 7.5 (b)).

7.5.2 Performance Measurement

Figures 7.6 and 7.7 plot the processing time of our rendering algorithm versus the number of target light rays and depth layers, respectively. Here we used a set of static input images (i.e. no texture uploading) and measured the average processing time of 100 executions of the rendering algorithm. The smoothing window size was fixed to 11×11. We measured the time by rendering the results to back buffers, because the refresh rate of the display (60 Hz) limits the processing time if we draw the resulting images on the display.

The processing time was proportional to both the number of target light rays (output resolution) and that of the depth layers, as discussed in Section 7.3.1. These times are fast enough for real-time interactive rendering. The difference in processing time between bilinear and average interpolations is small, which means that bilinear interpolation can be used for higher-quality rendering. The most time-consuming process is the spatial cost smoothing, which needs to aggregate all neighboring values.

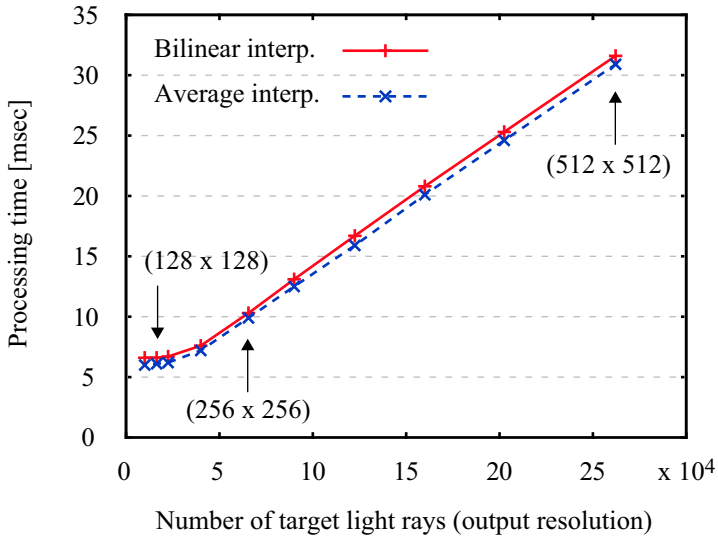


Figure 7.6: Processing time of the rendering algorithm for different numbers of target light rays (output resolution). The number of depth layers was fixed to 15.

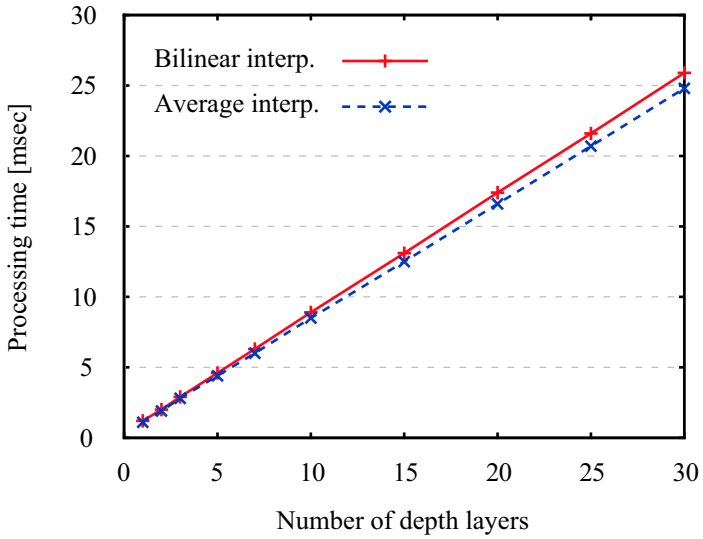


Figure 7.7: Processing time of the rendering algorithm for different numbers of depth layers. The output resolution was fixed to 300×300.

The total throughput of our system, including the CPU processing and texture uploading, was 30 fps using 300×300 output resolution and 7 depth layers, and 20 fps using 500×500 output resolution and 20 depth layers, for example. At a throughput of 30 fps, the total network bandwidth from the camera array to the PC was about 270–330 Mbps. The system throughput was limited by the GPU processing (rendering and texture uploading). The network I/O and JPEG decoding were not bottlenecks even at 30 fps in our system. When we tried to use higher resolution input images, the JPEG decoding became the bottleneck as well as the GPU processing.

7.5.3 Discussion

Our system renders all-in-focus novel views in real time by using the view-dependent depth estimation and bilinear color interpolation methods. The interpolation method prevents color discontinuities at reference camera boundaries and generates visually natural images. We use bilinear interpolation for higher-quality rendering because the difference in processing time between average and bilinear interpolations is small. However, some artifacts are still visible in the rendered views. They are caused by the following three sources.

- *Incorrect depth estimation at object boundaries*: Since we use a normal square window for smoothing the color consistency cost, the regions near depth boundaries tend to prefer the foreground depth value (the "foreground fattening" effect [94]). This incorrect depth estimation produces halo artifacts near objects boundaries (e.g., around the head of the foreground person in Fig. 7.3). A simple method for preventing the artifacts is using a shiftable window [94], which can be also implemented on a GPU efficiently (Gong [38], for example, showed a GPU implementation of the shiftable window for trinocular stereo sequences). In our system, however, the shiftable window often produced worse-looking images than the normal square window, because the outliers in the color consistency cost have a larger influence to the shiftable window. We therefore used the normal square window in our implementation.

- *Unsynchronized input videos*: Figure 7.8 shows images synthesized from two sequences, *Soccer* and *Juggling*, which have fast motions. Since our camera array has no synchronization function, the frames used in the rendering process may be captured at different times. This causes motion blur artifacts in the fast moving parts (e.g., the soccer ball and the juggling clubs in Fig. 7.8). Even in such parts, the bilinear interpolation produces better-looking results than the average interpolation, because it blends the misaligned parts more naturally. Moreover, because our system can run at a relatively high frame rate, the output videos are not so disturbing.

- *Blur in the input images*: Network cameras are suitable for building a compact camera array system that can run with only a single PC, because they can reduce the amount of data sent to

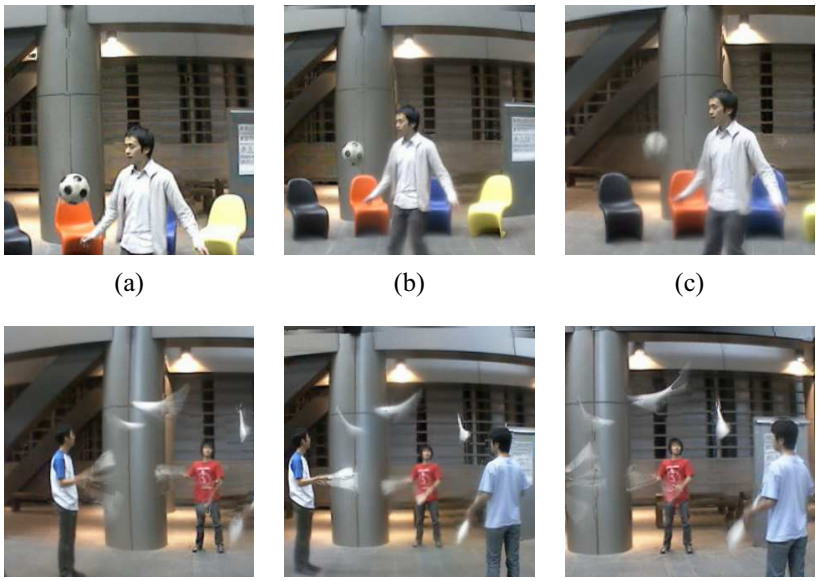


Figure 7.8: Synthesized images from (a–c) *Soccer* and (d–f) *Juggling* sequences. Images in each row are rendered from different viewpoints at the same time frame. Although these images have motion blur artifacts due to the fast motions, the output videos have visually acceptable quality.

the PC to a reasonable level by compressing the captured images. Unfortunately, however, the image quality of network cameras is generally not very high. Some of the input images of our system, in fact, suffer from blur and look unclean, as it can be seen in Fig. 7.3 (a). They produce low-quality regions in the final output images (e.g., the right person and chair in Fig. 7.3 (c) and the chairs in Fig. 7.8 (c)). We found that the degradation of the camera sensor causes the inevitable blur, and the only way to prevent this problem was replacing the low-quality camera with a better one. Such input images also increase outliers in the color consistency cost. We therefore used a relatively larger window of 11×11 pixels for smoothing the cost.

The processing time of our rendering method is proportional to the output resolution and the number of depth layers, as described in Section 7.5.2. However, note that synthesizing a higher resolution image requires more depth layers [14] as well as a larger smoothing kernel. Therefore, the total computational cost for a higher resolution is typically not proportional. Moreover, although the processing time itself is independent of the number of input cameras, a smaller number of input cameras (i.e. larger camera intervals) requires more depth layers, resulting in longer processing time.

Since synthesizing a novel view requires only parts of segments in the input images, several systems [95, 131, 139] use the region of interest (ROI) approach to reduce the amount of processed data. We could, for example, partially decode the received JPEG images and only upload the decoded segments to the GPU memory by using the ROI approach. However, we did not use such an approach because the current viewpoint is needed to determine the ROI. In this case, the rendering result reflects the current viewpoint after the CPU processing (i.e. JPEG decoding and texture uploading), which causes less interactivity. Meanwhile, our implementation performs the JPEG decoding, texture uploading, and novel view rendering independently. Because the rendering process only requires the current viewpoint, our approach has less delay than the ROI approach. Moreover, our system has a sufficient rendering rate without reducing the data amount thanks to the advancement of hardware and GPU functions.

7.6 Conclusions

We have presented a real-time video-based rendering system using an array of 64 commodity network cameras. Our system renders all-in-focus novel views from live multi-view videos by using an on-the-fly per-pixel depth estimation method. The rendering algorithm is fully implemented on the GPU, which allows our system to efficiently perform the capturing and rendering processes as a pipeline and to render novel views at up to 30 fps depending on the rendering parameters. Since our system setup is simple, we can capture various scenes by moving the camera array. The rendering results show that our method produces visually natural images even from the frames that have large variations and are captured at slightly different timings.

7.7 Appendix

Algorithm 7.1 is pseudocode of OpenGL commands for our rendering algorithm, and Algorithm 7.2 shows the details of the fragment programs. Texture names in Algorithm 7.2 are defined in Algorithm 7.1. The textures are passed to the fragment programs by texture mapping functions, and the fragment programs load the value corresponding to each pixel on the rendered image from the textures. We could use an OpenGL extension EXT_framebuffer_object for outputting the processed data directly to the textures, instead of copying the output data from the rendered frame buffer to the textures in Algorithm 7.1. In the experiments in Section 7.5.2, however, we did not use it because the processing times were similar.

Algorithm 7.1 Rendering algorithm

```
//Allocate texture memory on GPU
Texture tInputImage [64], tLayer, tCurOptImage, tCurOptDepth, tPrevDepth
Texture tBilinearWeight //Initialized with weight values

procedure RENDERING
  for all camera idx do
    UploadTexture (tInputImage [camera idx])
  end for
  //The view synthesis procedure is invoked after each uploading of 64 input views
  VIEWSYNTHESIS(current viewpoint)
end procedure

procedure VIEWSYNTHESIS(viewpoint)
  for all depth layer do
    SetFragmentProgram (CALCCOSTINTERPCOLOR)
    //Neighboring 4 cameras are used for the target light rays passing within the positions
    //of the cameras
    for all neighboring 4 cameras do
      for idx ← 1, 4 do
        cIdx ← FindCameraIdx (target 4 cameras, idx)
        //Set matrices based on the current viewpoint and the calibration parameters of the camera
        SetMatrices (viewpoint, CalibParams [cIdx])
        ProjectiveTextureMapping (tInputImage [cIdx])
        //The weight texture is mapped with different directions for each index
        SetMatrices (viewpoint, CalibParams [cIdx], idx)
        TextureMapping (tBilinearWeight)
      end for
    end for
    CopyFrameToTexture (tLayer)

    //Set matrices to orthogonal for applying the same instructions for each pixel in the textures
    SetMatrices (Orthogonal 2D)

    //Smooth the cost and select the depth with the minimum cost
    SetFragmentProgram (SMOOTHCOSTRECONDEPTH)
    TextureMapping (tLayer, tCurOptDepth, tPrevDepth)
    CopyFrameToTexture (tCurOptDepth)
    //Store the estimated depth map for the next iteration
    if last depth layer then
      CopyFrameToTexture (tPrevDepth)
    end if

    //Select the color with the minimum cost
    SetFragmentProgram (COMPOSITELAYERCOLOR)
    TextureMapping (tLayer, tCurOptDepth, tCurOptImage)
    CopyFrameToTexture (tCurOptImage)
  end for
end procedure
```

Algorithm 7.2 Fragment programs

```
procedure CALCCOSTINTERPCOLOR
  //Calculate color consistency cost
  cost ← Sum_{c←r,g,b} (Variance_i (tInputImage[i].c))
  //Interpolate color
  color ← Sum_i (tInputImage[i].rgb · tBilinearWeight[i])

  //Output the calculated values to (RGB, A) channels of the frame buffer
  Output (color, cost)
end procedure

procedure SMOOTHCOSTRECONDEPTH
  //Smooth cost in the neighborhood window W
  smoothCost ← Average_W (tLayer.a)

  //Temporal depth smoothing
  if current depth value ≠ tPrevDepth then
    smoothCost ← smoothCost + λ
  end if

  //Update the current depth map
  if smoothCost < tCurOptDepth.a then
    Output (current depth value, smoothCost)
  else
    Output tCurOptDepth
  end if
end procedure

procedure COMPOSITELAYERCOLOR
  //Update the current color image
  //Smoothed cost is stored in tCurOptDepth.a
  if tCurOptDepth.a < tCurOptImage.a then
    Output (tLayer.rgb, tCurOptDepth.a)
  else
    Output tCurOptImage
  end if
end procedure
```

Chapter 8

Live Transmission of Light Field from a Camera Array to an Integral Photography Display

8.1 Introduction

Three-dimensional TV is a promising technology for providing a more natural and intuitive perception of 3D scenes than two-dimensional TV. In particular, live 3D TV systems, which transmit 3D visual information in real time, could have a significant impact on many applications in communication, broadcasting, and entertainment. In principle, such a system can be implemented by transmitting all of the light rays that pass through a plane, called light field [39, 59]. However, developing a practical live 3D TV system has been only enabled by recent advances in imaging devices, computers, and display technologies.

This chapter presents a live 3D TV system, TransCAIP, using an array of 64 video cameras and an integral photography display with 60 viewing directions, as shown in Fig. 8.1. The live 3D scene in front of the camera array is reproduced by the full-color, full-parallax autostereoscopic display, which gives users a perception of observing the 3D scene through a window without requiring them to wear special glasses. Unlike existing live 3D TV systems using symmetric input and output devices [4, 74, 84], our system connects asymmetric input and output devices by using a light field conversion method. For the conversion, our system first uses an image-based rendering method to render 60 novel views that correspond to the viewing directions of the display, and then arranges the rendered pixels to produce an integral photography image. All the conversion processes are performed in real time on a GPU of a single PC. The conversion method also allows a user to interactively control viewing parameters of the displayed 3D image by changing the rendering parameters. The controllable viewing parameters include the depth

A demonstration video is available at http://www.hc.ic.i.u-tokyo.ac.jp/project/TransCAIP/

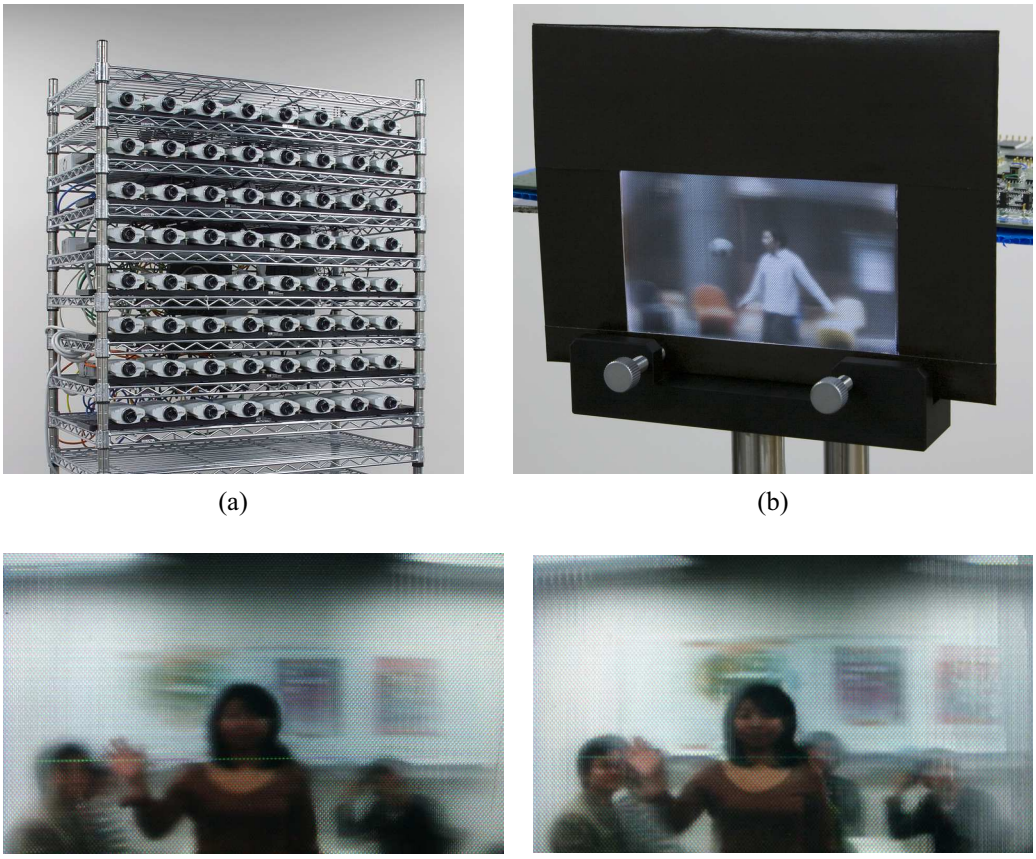


Figure 8.1: Our live 3D TV system consists of (a) an array of 64 cameras that captures multi-view videos of a dynamic scene and (b) an integral photography display with 60 viewing directions that reproduces (c) a full-color and full-parallax autostereoscopic 3D video of the scene.

at which objects are reproduced with the highest spatial resolution of the display, the amount of depth reproduced on the display, and the portion of the captured scene reproduced on the display. Since integral photography displays have a limited depth of field with the display plane having the highest spatial resolution [44, 150], it is essential to control the parameters and reproduce an interesting part of the scene near the display plane.

In summary, we make the following contributions:

- **Real-time light field conversion**. Our system performs real-time light field conversion from multi-camera images to an integral photography image by using an image-based rendering method fully implemented on a GPU.

- **Interactive control of viewing parameters**. Our system allows a user to interactively control viewing parameters of the displayed image for reproducing the dynamic 3D scene with desirable parameters.

8.2 Related Work

8.2.1 Video-Based Rendering

Free-viewpoint videos can be generated from multi-view videos captured with a camera array or a lens array by using video-based rendering techniques. As we reviewed in Section 7.2, some systems [50, 123, 147] reconstruct a geometry model from the captured multi-view videos using a sophisticated algorithm as an offline process, and perform interactive rendering using the estimated geometry. On the other hand, another type of systems aims for real-time (online) rendering from live multi-view videos. Earlier systems of the latter type perform rendering by approximating the scene geometry as a single plane [80, 131]. Such a rendering algorithm, however, produces low-quality synthesized images in which only the objects at the depth of that plane are clear and the objects at other depths are blurred or appear with ghosting artifacts [14, 45]. For higher-quality rendering, more recent real-time systems estimate the scene geometry [95, 130, 134, 139] or use specialized hardware to acquire it [81]. In the previous chapter, we have presented a real-time video-based rendering system that uses an array of 64 network cameras, estimates a view-dependent per-pixel depth map, and produces free-viewpoint video at rates up to 30 fps. We use the camera array of that system as the input device of the system described in this chapter. Compared to the existing video-based rendering systems, which render a single view in real time, the system described in this chapter performs much more complex processes (rendering 60 views and generating an integral photography image) in real time on a single PC.

8.2.2 Autostereoscopic 3D Displays

Autostereoscopic 3D displays present 3D images without requiring the viewers to wear special glasses. They reproduce different viewpoint images for different viewing directions. Although the basic principles of autostereoscopic displays were developed over a century ago [63], creating practical autostereoscopic video displays was difficult because such displays need to reproduce huge number of light rays (the resolution of a view times the number of views). Recent advances in digital imaging and display technologies, however, have made possible the creation of such displays [8, 54].

Some displays are designed to present different views for 360-degrees directions by rotating an LED array [136] or a screen [49, 88]. They are suitable for observing an object within that volume. Meanwhile, others using a lenticular lens or a microlens array are suitable for reproducing an entire

scene. Lenticular displays are commercially available but basically provide only 1D parallax (either horizontal or vertical). Integral photography displays, by contrast, provide both horizontal and vertical parallax by using a microlens array. Recently developed integral photography displays use multiple projectors [61, 135] or a high-density LCD panel [4, 53, 84]. The system we describe in this chapter uses as its output device the integral photography display reported by Koike et al. [53], but our light field conversion method is applicable to any kind of integral photography and multi-view 3D displays.

8.2.3 Live 3D TV Systems

Okano et al. presented integral-photography-based systems that acquire and display light field in real time [4, 84]. Their systems capture a 3D scene by using a high-resolution video camera and a lens array consisting of gradient-index (GRIN) lenses, and present the captured video on an integral photography display. In integral photography displays, the 3D images reproduced near the display plane have a higher spatial resolution than those farther from the plane (i.e., such displays have a limited depth of field) [44, 150]. Okano et al. therefore set a large-aperture convex lens, called depth control lens, in front of the lens array in their capturing systems, so that the real images of objects are formed near the lens array and their 3D images are reproduced around the display plane of the integral photography display. The position of the depth control lens determines the depth at which objects appear with the highest spatial resolution. Our system controls this effect by changing rendering parameters as a software process, which provides more flexible control than the hardware reconfiguration that their systems need. Moreover, their systems need symmetric input and output devices (i.e., the number of lenses of the array for capturing is same as that of the array for displaying), whereas our system can use asymmetric input and output devices, which have different viewpoint layouts, thanks to the conversion method.

Matusik and Pfister [74] also developed a live 3D TV system using symmetric input and output devices: 16 cameras for the input, and 16 projectors with lenticular screens for the output. Although lenticular screens need multi-view images captured at regularly spaced viewpoints, exactly aligning the input cameras in such a manner is impractical. They therefore correct the misalignment of the camera viewpoints by using image-based rendering with approximating the scene geometry as a single plane. Our system uses image-based rendering not only for the viewpoint correction, but also for converting between the completely different viewpoint layouts of the input and output devices, and for interactively controlling the viewing parameters of the displayed 3D image. Moreover, our system is designed to produce 3D images that have both horizontal and vertical parallax, while their system shows images that have only horizontal parallax. Finally, our system uses a rendering method that estimates a per-pixel depth map, which synthesizes higher-quality images than their rendering method that approximates the scene geometry as a single plane.

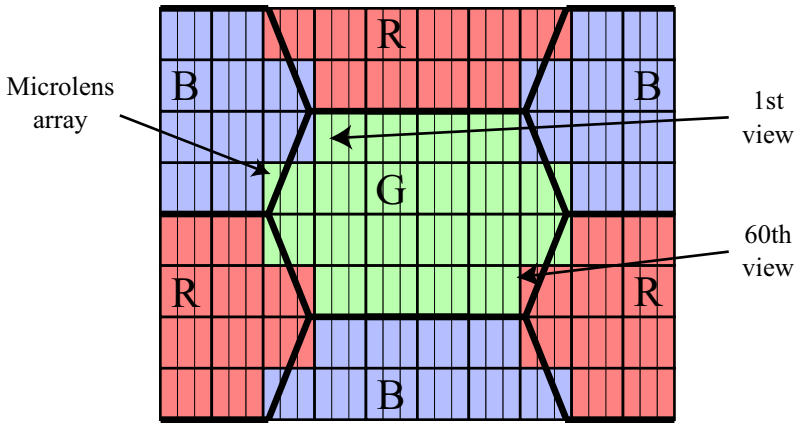


Figure 8.2: The dot layout and the modified color-filter arrangement of the integral photography display [53].

8.3 System Overview

Our system consists of an array of 64 cameras that captures multi-view videos of a dynamic scene (Fig. 8.1 (a)) and an integral photography display with 60 viewing directions that reproduces a full-color, full-parallax autostereoscopic video of the scene (Fig. 8.1 (b)). To connect these asymmetric input and output devices, the system performs light field conversion in real time on a single PC.

The camera array is composed of 8×8 Axis 210 network cameras. The distance between cameras is about 100 mm both horizontally and vertically. The camera employs a built-in HTTP server, which sends motion JPEG sequences in response to HTTP requests from clients. Its capture rate can be as high as 640×480 pixels at 30 fps. The JPEG compression factor can be adjusted between 0 (the best quality) and 100 (the worst quality). In our setup, we used an image resolution of 320×240 pixels and a JPEG compression factor of 10. We performed geometric calibration of the cameras with Tsai's method [114] by detecting the corner points of a checkerboard pattern at several known positions.

The cameras are connected to a single PC through gigabit Ethernet switches. We used an Intel Xeon 5160 (3 GHz) dual-processor machine (totally having 4 CPU cores) with 3 GB of main memory and an NVIDIA GeForce 8800 Ultra graphics card. The PC converts 64 input views captured with the camera array to an integral photography image consisting of 60 views. All of the conversion processes are implemented on the GPU using OpenGL and Cg (C for graphics) and performed in real time, as described in Section 8.4. The conversion method also allows users to interactively control viewing parameters of the displayed image, as described in Section 8.5.

The integral photography display presents an autostereoscopic 3D video with 60 views of 256×192 pixels. As shown in Fig. 8.2, each lens in the integral photography display [53] covers

60 dots (four rows of 12, 18, 18, and 12) of an LCD panel of 3840×2304 dots (1280×768 pixels) *, and this dot layout corresponds to the viewing zone of the display. To increase the density of the viewing directions without color moire artifacts, the display uses a modified color-filter layout in which each microlens presents only one RGB component; therefore, the number of dots, instead of the number of pixels as we described in Section 2.3, behind each microlens corresponds to the number of the presented views. Since the display uses a high-density microlens array, users can observe natural full-color 3D images even with the modified color-filter layout.

8.4 Real-Time Light Field Conversion

Figure 8.3 shows an overview of our light field conversion method. Using the 64 input views (Fig. 8.3 (a)), the method first renders 60 novel views (Fig. 8.3 (b)) corresponding to the viewing directions of the display by using an image-based rendering method, and then arranges the rendered pixels to produce an integral photography image (Fig. 8.3 (c)). As shown in Fig. 8.4, the rendering cameras are placed at a regular interval such that their viewing directions converge at the same point. As described in detail in Section 8.5, the camera arrangement determines the viewing parameters of the displayed image. For generating high-quality novel views, our method estimates a view-dependent per-pixel depth map based on a layered representation. The conversion method is fully implemented on a GPU for real-time processing on a single PC.

In the following subsections, we first describe the method for rendering 60 novel views, and then show the pixel arrangement method for generating an integral photography image from the rendered views. Next, we present an accelerated rendering method by using depth map interpolation. Finally, we discuss the rendering results and the performance of our system.

8.4.1 Rendering 60 Novel Views

The camera array captures multi-view videos at 8×8 viewpoints that are roughly aligned on a 2D plane, while the integral photography display presents 60 views that need to be regularly aligned such that they correspond to the viewing directions of the display. We therefore use an image-based rendering method to render the 60 views.

For this rendering, we use the GPU rendering method that was presented in Chapter 7 for real-time video-based rendering. Here we do not use the temporal smoothness constraints, because the frame rate of the system presented in this chapter is relatively low due to higher computational cost than the system that only renders a single view. The rendering method is performed at the 60 viewpoints, and the rendered images and depth maps are stored in the GPU texture memory.

*The high-density LCD panel tends to have some defective lines of dots, since it is a research prototype. This hardware defect causes the horizontal or vertical line artifacts that appear in some of the displayed images shown in this chapter.

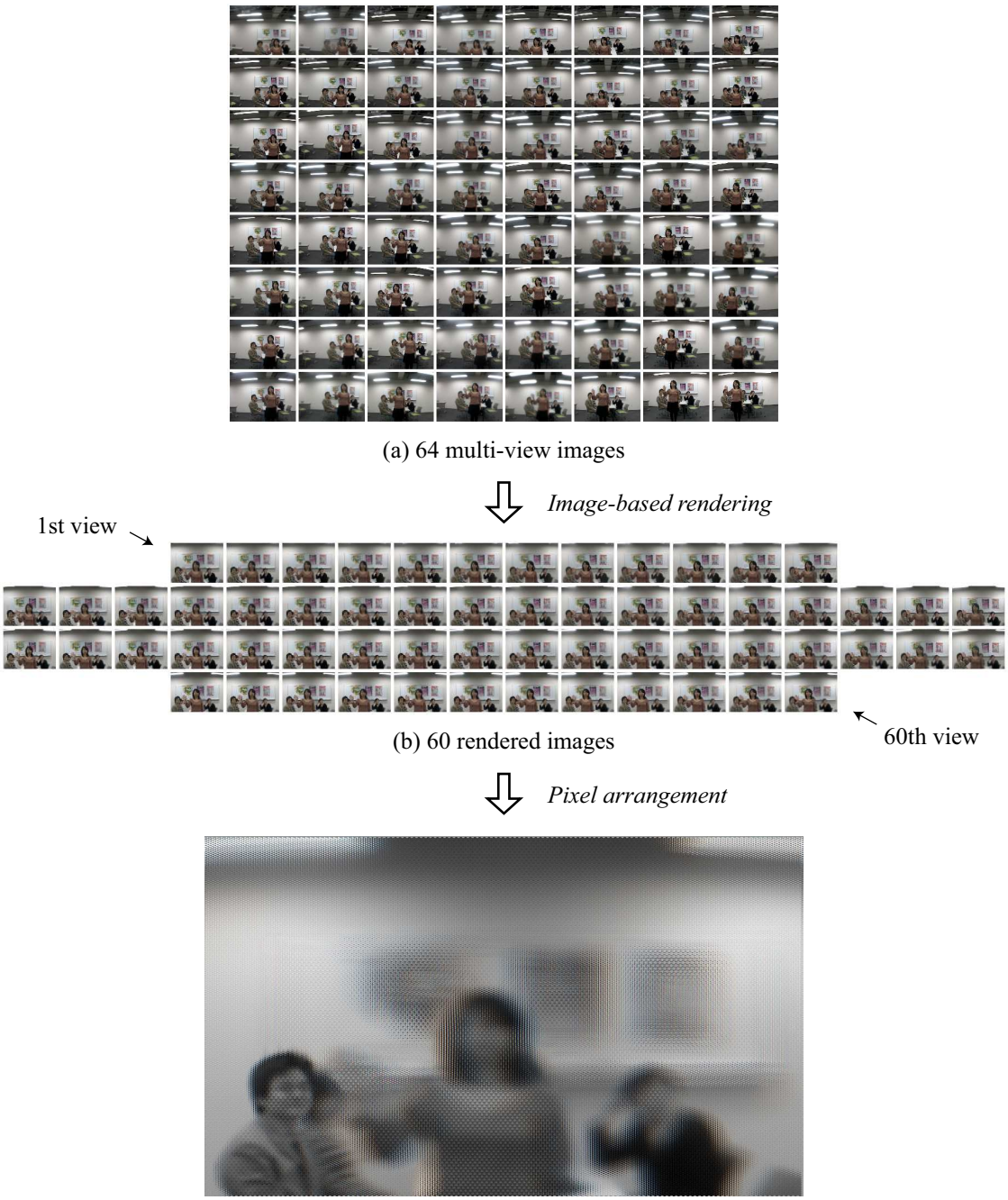


Figure 8.3: Overview of our conversion method. Using (a) 64 input views, the method first renders (b) 60 novel views corresponding to the viewing directions of the integral photography display. The rendered pixels are then arranged to produce (c) an integral photography image. All of the conversion processes are performed on a GPU.

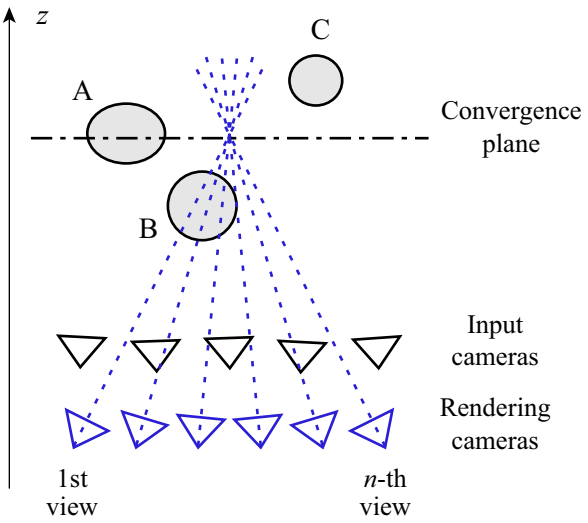


Figure 8.4: Configuration for rendering 60 views. The convergence plane corresponds to the display plane of the integral photography display.

Figure 8.3 (a) shows an example of the input multi-views in which the objects appear at different positions because the input cameras are not aligned regularly. Figure 8.3 (b), on the other hand, shows that the objects appear regularly in the 60 rendered views because the rendering cameras are aligned at a regular interval. As shown in these figures, the rendering method produces exactly aligned images that are suitable for presenting on integral photography displays, as well as converting the positions of viewpoints from 8×8 to four rows of 12, 18, 18, and 12.

8.4.2 Generating an Integral Photography Image

To produce an integral photography image from the 60 rendered images, we arrange the pixels such that a color component of a single pixel in each rendered image contributes to forming the set of 60 dots within a microlens (Fig. 8.2). We implemented the pixel arrangement on the GPU, because such an implementation does not require the transmission of the 60 rendered images from the GPU texture memory to the main memory for the CPU and enables fast conversion.

To efficiently perform the pixel arrangement on the GPU, we create a look-up table as an RGBA texture, as shown in Table 8.1. This look-up table describes which rendered pixel is used for each target pixel in the integral photography image. The rendered view index takes an integer in $[1 \ldots 20]$, because a single pixel (consisting of three dots) includes three viewing directions described with three sequential view indices. The X and Y coordinates describe the position of the rendered pixel within each rendered image. Because of the delta layout of microlenses on

Table 8.1: Look-up table for pixel arrangement on a GPU. Each color component of an RGBA texture can encode 8 bit information.

R	Rendered view index (5 bits)
G	X coordinate (8 bits)
B	Y coordinate (high 8 bits)
A	Y coordinate (low 1 bit) / Color flag (2 bits)

the display, the Y coordinate requires half-pixel accuracy; that is, it takes an integer in the range $[1 \ldots 384]$ and is represented by 9 bits. The color flag describes which color component (RGB) in the rendered pixel is used for the pixel of the integral photography image.

Once the look-up table is uploaded to the texture memory, the pixel arrangement is performed by using texture mapping of the 60 rendered images and the look-up table with a fragment program. Because of the limitation of the number of textures that can be used in a single rendering pass, the process is divided into a few passes (our implementation uses five rendering passes). The resulting integral photography image is directly presented on the display, so all the conversion processes are completed on the GPU.

8.4.3 Accelerating the Rendering by Using Depth Map Interpolation

Since the 60 novel views are rendered at closely-aligned viewpoints, the depth map estimated at one viewpoint is similar to that estimated at the neighboring viewpoints. We therefore use a depth interpolation method that exploits the similarity of the neighboring depth maps for accelerating the rendering.

For the depth interpolation, we first estimate the depth maps at each single viewpoint out of M viewpoints by using the method described in Section 8.4.1, and then interpolate the depth maps at the rest of viewpoints; therefore, the number of viewpoints whose depth maps are interpolated increases with the increase of M, while $M = 1$ means that the depth interpolation is not used (the depth maps are estimated at all viewpoints). The depth interpolation is performed by computing the weighted mean of the estimated depth maps at the two-nearest viewpoints according to the distance between viewpoints. We use only the estimated depth maps at the left and right viewpoints (not the top and bottom ones) because, in the dot layout of the integral photography display shown in Fig. 8.2, the distance between horizontally neighboring viewpoints is one-third of that between vertically neighboring viewpoints.

Although the depth map is interpolated from the neighboring views, instead of interpolating the color from the neighboring views, we use the color interpolation method of the original rendering method described in Section 8.4.1. This approach keeps the view-dependent component of the scene, because it uses the colors of light rays that are the closest to the target viewpoint. As in

[13], we use the interpolated depth map as a geometric proxy and interpolate the color of light rays in the rendered image by using the four-nearest reference cameras for each target light ray. This rendering method using the depth interpolation is also implemented on the GPU by modifying the depth estimation process of the original rendering method.

8.4.4 Rendering Results

Figure 8.5 compares synthesized images and depth maps obtained using different number of depth layers and different rendering methods at an identical viewpoint. The results in Figs. 8.5 (a)–(c) were generated by using the depth estimation method at the viewpoint ($M = 1$) with (a) a single depth layer, (b) 3 depth layers, and (c) 5 depth layers. The smoothing window size was fixed at 7×7 pixels. The result with a single depth layer (a) suffers from blur and ghosting artifacts except for the left person on which the depth layer is placed. The rendering quality increases with the increase of the number of depth layers. Using too many layers, however, does not contribute to the rendering quality [14] and only increases the computational cost of the depth estimation. We therefore empirically choose an appropriate number of depth layers that produces enough visual quality while keeping the computational cost low (we used 5 layers in the results shown in this paper).

Figures 8.5 (d)–(f), on the other hand, show the results obtained using the interpolated depth maps with $M = 2$, 3, and 4, respectively. The interpolated depth maps (d)–(f) are similar to the one estimated at the viewpoint (c), and the synthesized images are of comparable quality. Although the interpolation quality depends on rendering parameters (e.g., interval between the rendering cameras), with our typical parameters, the depth interpolation using these values of M works well.

Limitations

A limitation of our rendering method is that it produces halo artifacts near depth boundaries (e.g., around the head of the center person in Fig. 8.5 (c)). This is because our depth estimation method smoothes the color consistency cost by using a square window, which tends to prefer the foreground depth value at the boundary regions (the foreground fattening effect [94]). Such artifacts also appear in the images rendered with interpolated depth maps, as shown in Figs. 8.5 (d')–(f'), since the interpolated depth maps are generated from the depth maps that have incorrect depth estimates at depth boundaries. A simple method for preventing the artifact is using a shiftable window instead of a square window [94]. However, we did not use it, because it requires additional computational cost and the artifact does not decrease the visual quality so much when we see the images on the integral photography display.

Another limitation comes from the fact that the camera array does not have synchronization

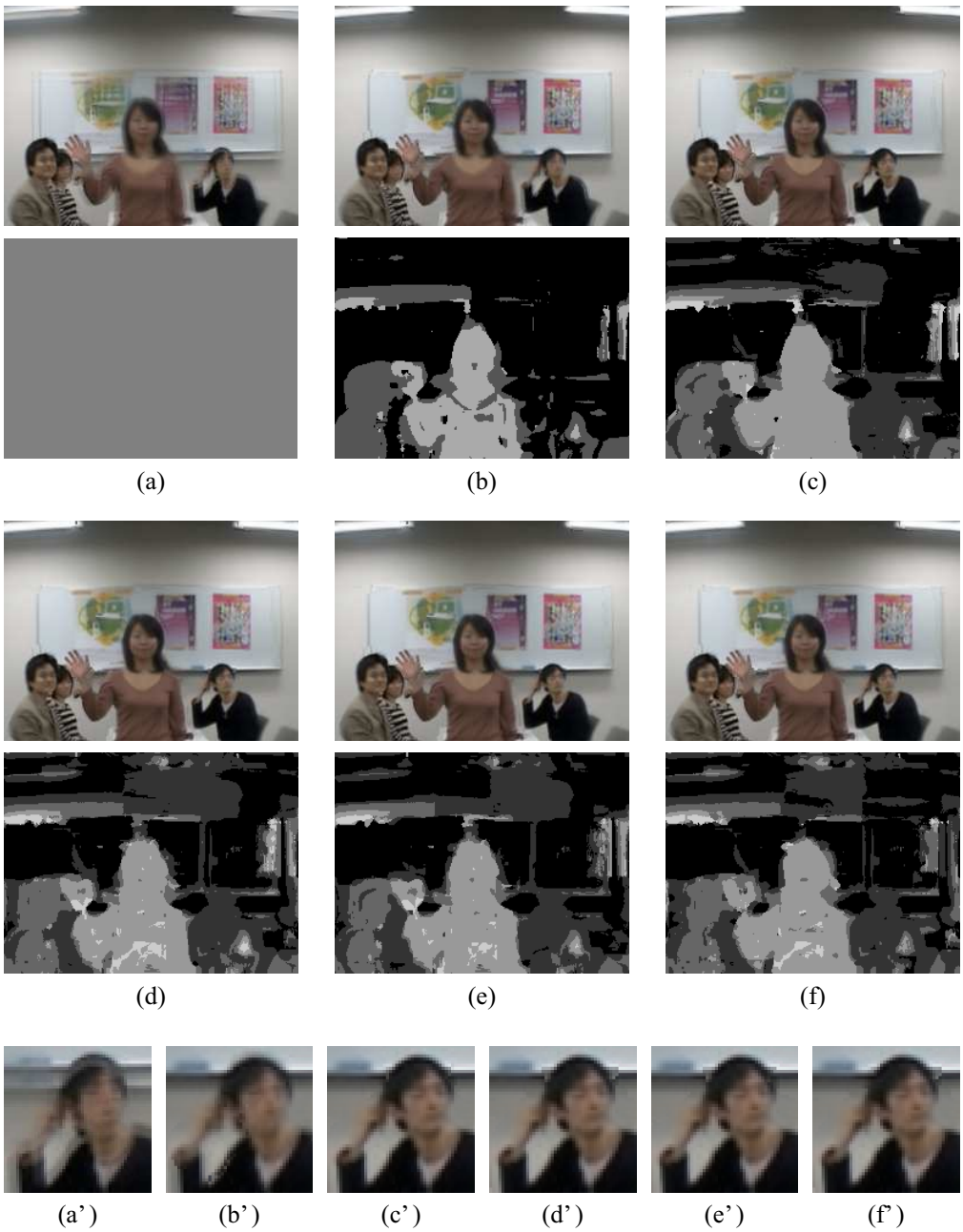


Figure 8.5: (a–f) Synthesized images and depth maps at the 38th viewpoint obtained using different number of depth layers and different rendering methods, and (a'–f') close-up views of the right person in the corresponding images. (a–c) The depth estimation method at the viewpoint ($M = 1$) using (a) a single layer, (b) 3 layers, and (c) 5 layers. The rendering quality increases with the increase of the number of depth layers. (d–f) The depth interpolation method with (d) $M = 2$, (e) $M = 3$, and (f) $M = 4$, using 5 depth layers. The interpolated depth maps (d–f) are similar to the one estimated at the viewpoint (c), and the synthesized images are of comparable quality.

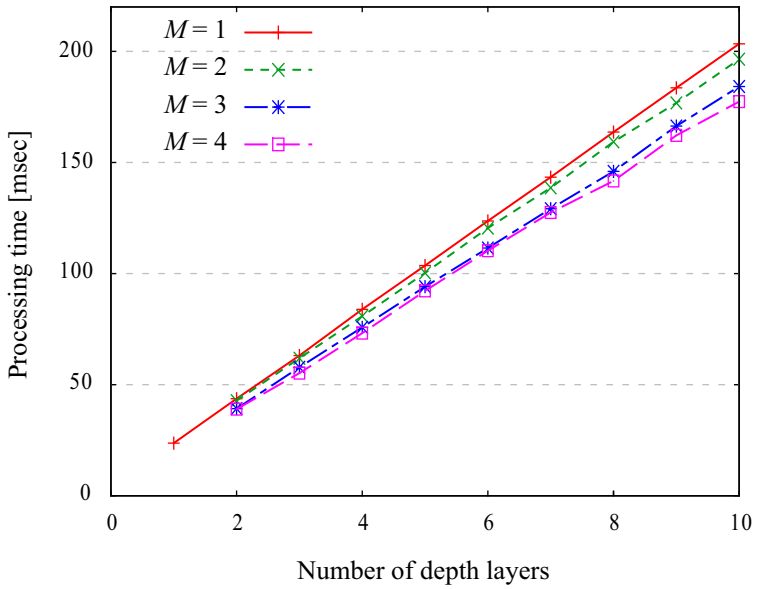


Figure 8.6: Relations between the number of depth layers and the processing time for rendering 60 views with different parameters M.

function. Since the input images used for the rendering may be captured at slightly different times, fast moving objects in the scene are not aligned well in the rendered 60 views. The misaligned objects produce visible artifacts when we see a single time frame of the scene on the integral photography display. However, when we see the scene as a video sequence on the display, moving objects are reproduced with visually acceptable quality as shown in the demonstration video.

8.4.5 Performance

Figure 8.6 plot the processing time for rendering 60 views against the number of depth layers. We obtained the data plotted there by using a set of static input images (i.e., no texture uploading) and measuring the average processing time for 100 executions of the 60-view rendering. For each of the rendering methods, the average processing times are proportional to the number of depth layers. The depth interpolation method reduces processing time more when M is larger; the rendering method with $M = 4$ reduces the processing time below that of the method with $M = 1$ (estimating the depth maps at all viewpoints) by about 12%.

Figure 8.7 shows the overall process flow of our system and the processing time of each process. The system first captures 64 motion JPEGs from the cameras and inserts them into a queue. The network bandwidth between the camera array and the PC was about 8–10 Mbps at 1 fps. Although the bandwidth increases in proportion to the frame rate, it was not a problem even when

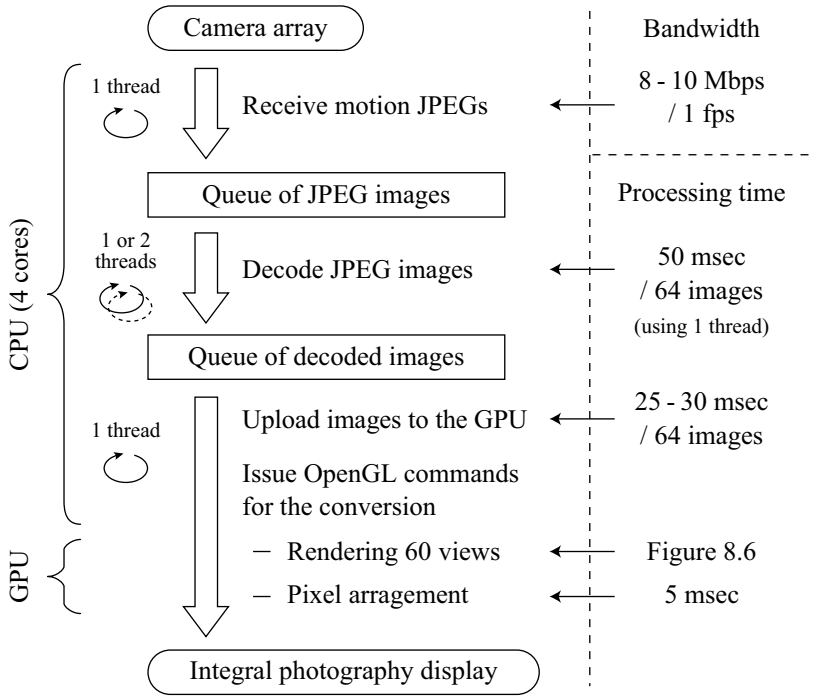


Figure 8.7: The process flow of our system and the processing time of each process.

we capture the 64 videos at 30 fps (the maximum capture rate of the camera), as we described in Chapter 7. The JPEG images are then decoded and inserted into another queue. Finally, the decoded images are uploaded to the GPU as textures, and the conversion is performed on the GPU using the textures. The system uses multiple threads to perform these processes independently and in parallel on different CPU cores, as shown in Fig. 8.7.

The bottleneck of our system is the most time-consuming thread, i.e., the thread that performs texture uploading and the light field conversion. Since these processes need to be sequentially performed, the processing time for generating an integral photography image is the sum of the following processing times: 25–30 milliseconds for uploading 64 input images to the GPU, the processing time shown in Fig. 8.6 for rendering 60 views (e.g., about 100 milliseconds using 5 depth layers), and 5 milliseconds for the pixel arrangement. The throughput of our system therefore depends on the rendering method and its parameters. When we use the depth estimation method at all viewpoints ($M = 1$) with 5 depth layers, the throughput was 6 fps. Using the depth interpolation method ($M = 4$) with 5 depth layers increased the throughput to 7 fps, which means that the depth interpolation method enables to increase the system throughput without reducing the visual quality of the presented views. If we use a single depth layer for the rendering, as Matusik and Pfister's system does [74], our system has a throughput of 15 fps but the presented images are

rendered with lower quality. The latency of our system was within 0.5 seconds, which is caused by all of the processes performed on the PC and the data transmission from the cameras.

Discussion

Here we discuss how the resolution and number of input and output images affect the system performance.

- **Resolution of input and output images**. Currently we use the input image resolution of 320×240 pixels, since it roughly matches the output image resolution of 256×192 pixels. When using higher-resolution displays, we need to increase the input resolution as well to capture images with sufficient spatial frequency. In this case, the network bandwidth increases in proportion to the input resolution. The processing time for decoding JPEG images and that for uploading the decoded images to the GPU also increase linearly. Meanwhile, the processing time of the rendering process increases more than in proportion to the output resolution, because synthesizing a higher resolution image requires more depth layers (basically, the number of depth layers required for high-quality rendering is proportional to the input/output resolution [14]) as well as a larger window for smoothing the color consistency cost, and both the output resolution and the number of depth layers linearly increase the processing time of the rendering method as we showed in Section 7.5.2. Therefore, the rendering process becomes the bigger bottleneck when we create a higher-resolution system.

- **Number of input images**. Briefly speaking, the number of depth layers required for high-quality rendering is proportional to the interval between input cameras (as well as the input/output resolution as described above) [14]. Therefore, if we use a quarter of the input cameras (4×4) with double the interval between them for keeping the size of the capturing volume fixed, we need to use double the number of depth layers. In this case, uploading decoded images to the GPU requires a quarter of the processing time (6–8 milliseconds), but rendering 60 views takes double the processing time (about 200 milliseconds using 5 depth layers); as a result, the system throughput decreases. This means that using all 64 cameras is reasonable in our current system.

- **Number of output images**. Our system sequentially renders the 60 views using a single thread, because the PC employs a single-GPU graphics card. The processing time of the rendering process is therefore proportional to the number of output views, when we use the depth estimation method at all viewpoints. The depth interpolation method accelerates the rendering process as shown in Fig. 8.6, but the improvement of the processing time is

currently limited (about 10 milliseconds per 60 views when we use 5 depth layers). For accelerating the rendering more, developing a method that exploits recent multi-GPU graphics cards to render multiple views in parallel would be interesting future work.

8.5 Interactive Control of Viewing Parameters

Our light field conversion method not only enables the viewpoint conversion from the camera array to the integral photography display as described in Section 8.4, but also provides interactive control of viewing parameters of the displayed 3D image. As shown in Fig. 8.4, the conversion method places the rendering cameras at a regular interval such that their viewing directions converge at the same point. The plane whose depth is equal to that of this point is called the convergence plane. The viewing parameters of the displayed image are controlled by changing the position of the convergence plane and the other parameters of the rendering cameras. This viewing parameter control is therefore performed as a software process without reconfiguring the hardware system.

Integral photography displays reproduce objects near the display plane with a higher spatial resolution than those farther from the plane; that is, the displays have a limited depth of field with the display plane having the highest spatial resolution (detailed analyses of the depth of field of such displays are described in [44, 150]). It is therefore important to reproduce an interesting part of the scene near the display plane. Moreover, since our system presents a dynamic 3D scene in real time, we let a user choose desirable viewing parameters that depend on the contents of the scene as well as user's preference. To our knowledge, existing works using integral photography or multi-view 3D displays do not consider such interactive control, because they mostly show a pre-rendered CG scene, where suitable parameters can be determined before displaying it.

The following subsections first describe controllable rendering parameters and their effect on the displayed image, and then show the resulting images presented on the integral photography display.

8.5.1 Controllable Parameters

Convergence plane

The convergence plane corresponds to the display plane of the integral photography display. Because of the limited depth of field of the display, it is important to set the convergence plane at an appropriate position in the target scene. The position of an object relative to the display plane is also determined by the convergence plane. In the configuration shown in Fig. 8.4, for example, the object A appears on the display plane with the highest spatial resolution of the display, and the objects B and C respectively appear in front of and in back of the display plane. When the user's viewpoint changes, the position of the object A is fixed because it has no parallax, while the

objects B and C move in opposite directions to each other. Our system enables users to control such 3D effects interactively by adjusting the position of the convergence plane along the z axis.

Interval between the rendering cameras

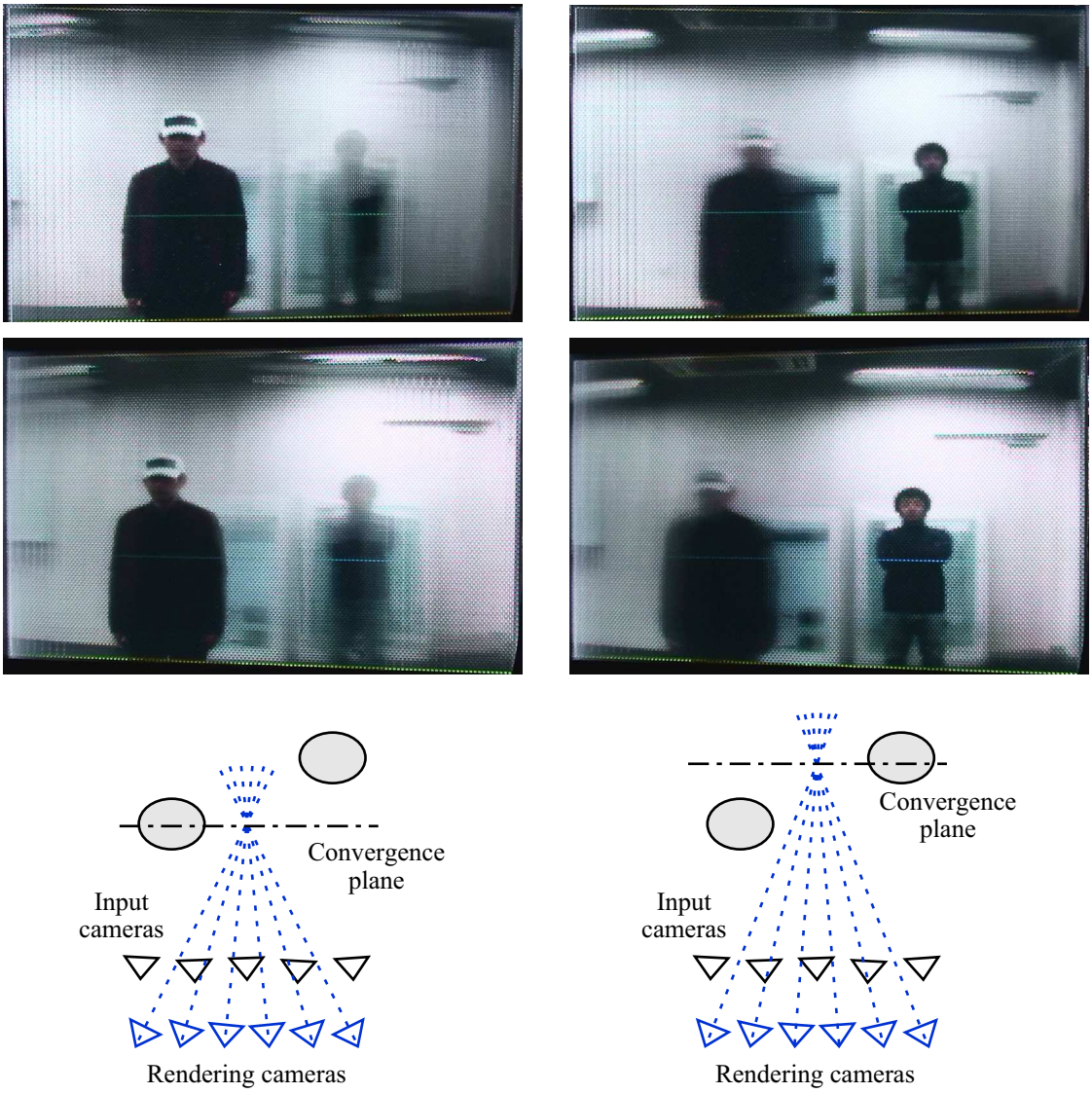
For reproducing a 3D image with the correct 3D scale on the integral photography display, the appropriate camera interval can be calculated from the specifications of the display. A correct-scale conversion, however, may not provide a good viewing experience, because it may reproduce objects farther from the display plane with blur and ghosting artifacts due to the limited depth of field of the display. To avoid this problem, our system controls the amount of depth reproduced on the display by changing the interval of the rendering cameras. The interval corresponds to how much depth amount is reproduced on the display. A larger camera interval produces more disparities, resulting in more amount of depth on the display. On the other hand, if we set all of the rendering cameras at the same position (i.e., no interval between cameras), the display acts as a 2D display because the same image is presented for all of the viewing directions. Our system allows users to adjust the trade-off between the depth amount and the spatial resolution of the reproduced 3D image, which is inherent in integral photography displays, by changing the interval of the rendering cameras.

Position and view angle of the rendering cameras

In our setup, the display often shows a 3D image of only a part of the captured scene, because the input cameras are aligned on an 8×8 grid, while the rendering cameras are aligned within a horizontally-long rectangular grid. We can control the location of the part of the scene reproduced on the display by changing the positions of the rendering cameras. The target part of the scene also depends on the view angle of the rendering cameras. A smaller view angle yields a narrower field of view on the display.

8.5.2 Results

Figure 8.8 shows displayed images obtained using different positions of the convergence plane. The person at the convergence plane (the left person in Fig. 8.8 (a) and the right person in Fig. 8.8 (b)) is clearly reproduced with the highest spatial resolution of the display and appears at the same position in the display regardless of the user's viewpoint. Meanwhile, the person not at the convergence plane appears with a lower spatial resolution but at the correct position relative to the person at the convergence plane. Since these results were captured by a single-viewpoint camera, they look like synthetic aperture photography [45, 82, 123], which adjusts focal plane to change the in-focus depth in rendered 2D images. However, our method that adjusts the convergence plane is different from the synthetic aperture photography; the rendered 60 images them-



(a) Convergence plane set at the left (near) person

(b) Convergence plane set at the right (far) person

Figure 8.8: Displayed images obtained using different positions of the convergence plane (top images captured from a left viewpoint and bottom images captured from a right viewpoint). In both cases, the person at the convergence plane appears with the highest spatial resolution of the display and at the same position in the display regardless of the user's viewpoint. Meanwhile, the person not at the convergence plane appears with a lower spatial resolution but at the correct position relative to the person at the convergence plane.

(a) No interval

(b) Small interval

(c) Large interval

Figure 8.9: Displayed images obtained using different rendering camera intervals (left images captured from a left viewpoint and right images captured from a right viewpoint). (a) When all the rendering cameras are set at the same viewpoint, the display acts as a 2D display (i.e., no parallax). (b) A small interval yields a small parallax (a small amount of depth perception). (c) A large interval yields a large parallax. Note, however, that objects farther from the display plane are reproduced with a lower spatial resolution.

Figure 8.10: Displayed images obtained using different positions and view angles of the rendering cameras. Note that the display presents autostereoscopic 3D images for any target part of the scene.

selves are all-in-focus (objects at all depths are clearly rendered), but the integral photography display produces the focusing effect due to the limited depth of field. Note that this focusing effect of the display is inevitable as we described above, so we provide the interactive control so that users can select the depth plane reproduced with the highest spatial resolution.

Figure 8.9 shows displayed images obtained using different rendering camera intervals. When all rendering cameras are set at the same viewpoint (Fig. 8.9 (a)), the display acts as a 2D display and users can observe the same image from any viewpoint. In this case, objects at all depths are reproduced on the display plane with the highest spatial resolution but without depth perception. When we use a small interval (Fig. 8.9 (b)), only a small amount of depth is reproduced on the display. A larger interval increases the reproduced depth amount (Fig. 8.9 (c)). However, we need to consider that objects farther from the display plane are reproduced with a lower spatial resolution (more blur).

Figure 8.10 shows displayed images obtained using different positions and view angles of the rendering cameras. The part of the scene reproduced on the display can be adjusted within the capturing volume of the camera array by changing these parameters. Although existing video-based rendering systems produce such a free-viewpoint effect, our system extends this effect with a 3D display; that is, users can observe autostereoscopic 3D images of the scene at any viewpoint.

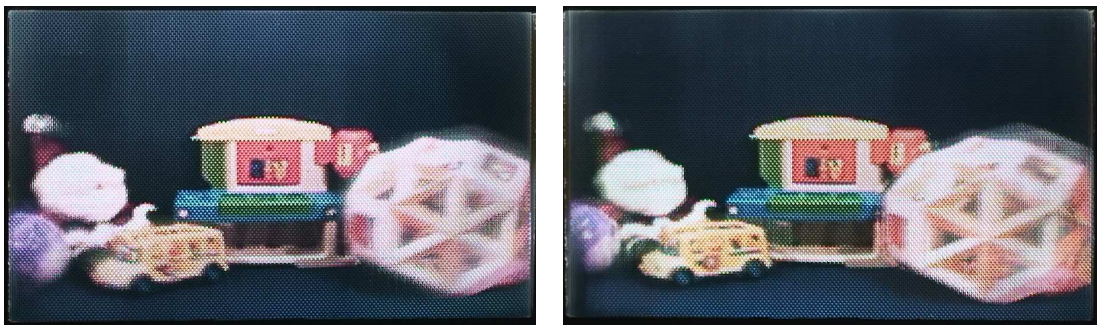


Figure 8.11: Displayed images obtained from the Toys image set (the left image captured from a left viewpoint and the right image captured from a right viewpoint), indicating the generality of our conversion method.

The demonstration video shows more results of changing these conversion parameters in dynamic scenes. However, all of the image and video results are captured by a single-viewpoint camera and do not fully convey the advantage of our system. We believe our system provides an attractive and convincing 3D experience for most users.

8.6 Discussion and Conclusions

Our system enables real-time light field transmission from an array of 64 cameras to an integral photography display with 60 viewing directions by using a light field conversion method. The conversion method provides a general way for converting the different viewpoint layouts between input and output devices. It also provides interactive control of viewing parameters, which is essential for reproducing a dynamic 3D scene with desirable parameters that depend on the contents of the scene, user's preference, and the display specifications. Our light field conversion method is therefore applicable to general combinations of camera arrays and integral photography (and multi-view 3D) displays.

Figure 8.11 confirms the generality of the conversion method by using another input image set, *Toys*, recorded by Zhang and Chen's self-reconfigurable camera array and provided on their website [139]. The image set consists of 48 (8×6) views of 320×240 pixels and are provided with the camera calibration parameters. The conversion method correctly generates an integral photography image from the input image set that has a different capturing parameter, and reproduces the 3D scene on the display with a good viewing condition by using the convergence plane set at the position of the center object and an appropriate rendering camera interval.

For capturing integral photography images directly using a microlens array, existing systems use a depth control lens (a large-aperture convex lens) so that the captured objects are reproduced

near the display plane with a higher spatial resolution [4, 84, 130]. Because the real images of objects need to be formed near the microlens array, capturing a large volume with these systems would require impractical optical systems, such as a huge convex lens and a long imaging distance between the convex lens and the microlens array. On the other hand, our method that uses a camera array for generating integral photography images does not require such a large lens and a long imaging distance. This advantage is similar to that of synthetic aperture photography using a camera array, which simulates a high-performance camera that has a large aperture [123]. Moreover, existing systems using the depth control lens require hardware reconfiguration for changing the convergence plane, while our system changes the convergence plane interactively as a software process.

There are basically two approaches for generating an integral photography image: one is to render the presented views corresponding to the viewing directions of the display and arrange the rendered pixels (as our system does), and the other is to directly render novel views from the viewpoints of each microlens. In our display, the number of presented views (60) is much smaller than the number of microlenses (256×192), so the former approach is computationally efficient because it requires fewer rendering passes. Yang et al. [135] pointed out this advantage for CG scenes. In our system, this approach has more advantages as follows. First, our display reproduces 3D images both in front and back of the display plane, which means direct generation of microlens views needs a complex rendering and depth estimation method. Second, the viewing parameter control can be performed simply and intuitively for the presented views.

Chapter 9

Conclusions

9.1 Summary of Contributions

This dissertation has explored compression and conversion techniques for light field data sets, both of which are essential for practical 3D TV systems. Our compression methods consider free-viewpoint videos as the output of the system and provide functionality that is suitable for rendering the free-viewpoint videos. We have presented two view-dependent scalable coding methods, which enable us to render high-quality views around a significant viewpoint even at low bit rates and to improve the quality of views away from the viewpoint with increasing bit rate. We have also dealt with a distributed coding architecture to exploit inter-view correlation in image-based rendering systems while keeping the system complexity as low as intra-coding systems. Our conversion methods, meanwhile, generate free-viewpoint videos and autostereoscopic 3D views of a scene from live multi-view videos in real time. The conversion methods allow us to construct 3D TV systems using general combinations of camera arrays and 2D/multi-view 3D displays. Moreover, we have presented an accurate two-frame stereo reconstruction method. Accurate geometry information estimated by the method offline could be used for improving the coding efficiency of the compression methods and enhancing the rendering quality of the conversion methods.

The contributions of this dissertation are summarized as follows.

Accurate Stereo Reconstruction

In Chapter 3, we have presented a stereo reconstruction method that estimates accurate depth maps using adaptive over-segmentation as an offline process. The method jointly estimates the segmentation and depth to overcome limitations of traditional segmentation-based methods while properly handling mixed pixels on object boundaries. The resulting depth maps are not only accurate according to accepted standards (the second version Middlebury stereo evaluation [94]) but in fact more complete, because we produce opacity information and foreground/background colors and

depths for mixed pixels. In contrast to most matting methods, we produce this information along depth discontinuities throughout the scene, not only for foreground objects. Our scene representation is suitable for the applications of Z-keying and view interpolation to produce high-quality results.

View-Dependent Scalable Coding

We have proposed a novel scalability for multi-view coding, called view-dependent scalability. The scalability enables us to render high-quality views around a significant viewpoint even at low bit rates and to improve the quality of views away from the viewpoint with increasing bit rate. The method presented in Chapter 4 uses an image-based rendering method before the encoding process to generate an image at the significant viewpoint, which acts as a reference image for predicting the input multi-view images. It produces a view-dependent scalable bitstream that can be used with three rendering methods depending on the bit rate. The method described in Chapter 5, on the other hand, provides more flexible control of the view-dependent scalability by using ROI-based techniques. It compensates the smooth movement of the remote user's viewpoint in interactive streaming of free-viewpoint videos.

Joint Decoding and Rendering Method for Distributed Multi-View Coding Systems

In Chapter 6, we have presented rendering-oriented decoding method for a distributed multi-view coding system using a coset code. The method considers how we can exploit inter-view correlation in image-based rendering systems while keeping the computational cost low and the system configuration simple. It uses a distributed coding approach and combines the decoding and rendering processes to directly synthesize the novel image without having to reconstruct all the input images. It keeps both encoder and decoder complexity as low as that of a conventional intra-coding system, while attaining better coding performance especially at low bit rates.

Real-Time Light Field Conversion Systems

We have developed real-time light field conversion systems using an array of 64 network cameras. The video-based rendering system presented in Chapter 7 renders high-quality novel views from live multi-view videos by using an on-the-fly per-pixel depth estimation method. The rendering algorithm is fully implemented on the GPU, which allows the system to efficiently perform the capturing and rendering processes as a pipeline and to render novel views at up to 30 fps depending on the rendering parameters. The rendering results show that our method produces visually natural images even from the frames that have large variations and are captured at slightly different timings. In Chapter 8, we have presented an end-to-end live 3D TV system that enables real-time

light field transmission from the 64-camera array to an integral photography display with 60 viewing directions by using a light field conversion method. The conversion method provides a general way for converting the different viewpoint layouts between input and output devices. It also provides interactive control of viewing parameters, which is essential for reproducing a dynamic 3D scene with desirable parameters that depend on the contents of the scene, user's preference, and the display specifications. The light field conversion method is therefore applicable to general combinations of camera arrays and multi-view 3D displays.

9.2 Future Directions

We conclude this dissertation with a discussion of open problems and future improvements that we are interested in pursuing.

Light Field Compression

Fine view-dependent scalability with 2D compatibility. The view-dependent coding method described in Chapter 4 produces a scalable bitstream that has 2D compatibility and can be used with three rendering methods, but the scalability control is limited because it controls the quality of residual information uniformly. On the other hand, the method described in Chapter 5 provides finer control of the view-dependent scalability by using ROI-based techniques, but it does not have 2D compatibility. Developing a coding method that provides fine view-dependent scalability with 2D compatibility by combining these two methods would be interesting; we could, for example, compress the residual information of the coding method described in Chapter 4 with an ROI coding method.

Scalable streaming with high compression efficiency. The coding methods described in Chapters 5 and 6 could be used together. The method described in Chapter 5 defines and encodes an ROI in each input image independently. Since it does not consider inter-view prediction, the compression efficiency is limited. Meanwhile, the rendering-oriented decoding method described in Chapter 6 exploits inter-view correlation at the decoder by using a distributed coding approach, but it currently transmits all of the input images, which are not necessary when we render a single view. We could encode only the ROIs in the input images using the distributed coding method, and render a novel view directly using the rendering-oriented decoding method for the ROIs. Such a method could enable scalable streaming of light field with improved compression efficiency, while keeping the simple system configuration that does not require inter-camera communication at the encoder.

Improving compression efficiency by using accurate depth estimation methods. We used a view-dependent depth estimation method for the inter-view prediction process of the methods described in Chapters 4 and 6. The depth estimation method is suitable for real-time processing, but the estimated depth map is not very accurate. When the system allows an offline process, we can use more accurate geometry estimation methods, such as the method described in Chapter 3. Using accurate geometry models would improve the prediction accuracy, resulting in higher compression efficiency.

Implementation for practical systems. Existing camera array systems independently encode individual views by using conventional video coding standards, such as motion JPEG ([139] and ours) and MPEG-2 ([74, 123]). This approach is suitable for practical real-time systems because of the simple system configuration, but the compression efficiency is limited. Currently developed MPEG standard for multi-view video coding (MVC) [76, 103], meanwhile, provides high compression efficiency by fully exploiting the inter-view correlation, but it is difficult to apply it for real-time systems. For developing real-time systems with improved compression efficiency, we could use the methods described in Chapters 5 and 6, because these methods encode individual views independently while reducing the data amount by selecting ROIs or by exploiting the inter-view correlation at the decoder. Implementing these methods on practical camera array systems would be interesting. In that case, it would be desirable that the cameras themselves have programmable units to manipulate the raw image data before encoding it.

Light Field Conversion

Display-oriented rendering. Our rendering method synthesizes objects at all depths clearly by estimating a view-dependent per-pixel depth map. It generates high-quality free-viewpoint videos for 2D displays. However, when we use integral photography or the other multi-view 3D displays as the output, objects away from the display plane are reproduced with blur and ghosting (aliasing) artifacts due to the limited depth of field of the displays. When we use a small rendering camera interval, it is necessary to render all the objects clearly, because all the objects are reproduced near the display plane; however, when we use a large rendering camera interval, we could reduce the number of depth layers and set them only near the convergence plane, which could make the rendering speed faster. Zwicker et al. [150] proposed a prefiltering method to suppress high-frequency components and avoid the aliasing on the multi-view 3D displays. Incorporating such a technique into our conversion method could improve the visual quality.

Viewing parameter control for each object in the 3D scene. Our conversion method controls rendering parameters, such as the convergence plane and the rendering camera interval, for the

entire scene. This determines which part of the scene is reproduced clearly on the integral photography display with an appropriate depth amount. Changing these parameters for parts of the scene or each object in the scene would produce more attractive 3D images, such as enhancing the depth amount of particular objects and changing relative positions of objects.

System design for specific applications. Our live 3D TV system provides flexible control of viewing parameters, which gives users an attractive and convincing 3D experience. However, the capability of the flexible control means that the system only uses a part of light rays captured by the camera array and discards the rest of light rays to produce a 3D image at a certain time. To efficiently use system resources, we could discard the data including unnecessary light rays at an early stage of the conversion process, although such an approach may reduce the interactivity of the system as we discussed in Section 7.5.3. Moreover, instead of using a general-purpose camera array for an integral photography display, designing an optimal camera placement for a specific display and application would be interesting future work.

References

[1] A. Aaron, P. Ramanathan, and B. Girod, "Wyner-Ziv coding of light fields for random access," in *Proc. IEEE Int. Workshop on Multimedia Signal Process. (MMSP 2004)*, Sept. 2004, pp. 323–326.

[2] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*, M. Landy and J. A. Movshon, Eds. Cambridge, MA: MIT Press, 1991, pp. 3–20.

[3] E. H. Adelson and J. Y. A. Wang, "Single lens stereo with a plenoptic camera," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 99–106, Feb. 1992.

[4] J. Arai, M. Okui, T. Yamashita, and F. Okano, "Integral three-dimensional television using a 2000-scanning-line video system," *Appl. Opt.*, vol. 45, no. 8, pp. 1704–1712, Mar. 2006.

[5] J. Askelof, M. L. Carlander, and C. Christopoulos, "Region of interest coding in JPEG 2000," *EURASIP Signal Process.: Image Commun.*, vol. 17, no. 1, pp. 105–111, Jan. 2002.

[6] S. Avidan and A. Shashua, "Novel view synthesis by cascading trilinear tensors," *IEEE Trans. Visual. Comput. Graphics*, vol. 4, no. 4, pp. 293–306, Oct.–Dec. 1998.

[7] S. Baker, R. Szeliski, and P. Anandan, "A layered approach to stereo reconstruction," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR '98)*, June 1998, pp. 434–441.

[8] P. Benzie, J. Watson, P. Surman, I. Rakkolainen, K. Hopf, H. Urey, V. Sainov, and C. von Kopylow, "A survey of 3DTV displays: Techniques and technologies," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1647–1658, Nov. 2007.

[9] R. Bernardini, R. Rinaldo, P. Zontone, D. Alfonso, and A. Vitali, "Wavelet domain distributed coding for video," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2006)*, Oct. 2006, pp. 245–248.

[10] S. Birchfield and C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 4, pp. 401–406, Apr. 1998.

[11] M. Bleyer and M. Gelautz, "A layered stereo algorithm using image segmentation and global visibility constraints," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2004)*, vol. 5, Oct. 2004, pp. 2997–300.

[12] M. Brown and D. G. Lowe, "Recognising panoramas," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV 2003)*, vol. 2, Oct. 2003, pp. 1218–1224.

[13] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen, "Unstructured lumigraph rendering," in *Proc. ACM SIGGRAPH 2001*, Aug. 2001, pp. 425–432.

[14] J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum, "Plenoptic sampling," in *Proc. ACM SIGGRAPH 2000*, July 2000, pp. 307–318.

[15] S.-C. Chan, K.-T. Ng, Z.-F. Gan, K.-L. Chan, and H.-Y. Shum, "The compression of simplified dynamic light fields," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process. (ICASSP 2003)*, Apr. 2003, pp. 653–656.

[16] C.-F. Chang, G. Bishop, and A. Lastra, "LDI tree: A hierarchical representation for image-based rendering," in *Proc. ACM SIGGRAPH 99*, Aug. 1999, pp. 291–298.

[17] C.-L. Chang and B. Girod, "Rate-distortion optimized interactive streaming for scalable bitstreams of light fields," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2004)*, vol. 5308, Jan. 2004, pp. 222–233.

[18] S. E. Chen, "QuickTime VR — an image-based approach to virtual environment navigation," in *Proc. ACM SIGGRAPH 95*, Aug. 1995, pp. 29–38.

[19] S. E. Chen and L. Williams, "View interpolation for image synthesis," in *Proc. ACM SIGGRAPH 93*, Aug. 1993, pp. 279–288.

[20] W.-C. Chen, J.-Y. Bouguet, M. H. Chu, and R. Grzeszczuk, "Light field mapping: Efficient representation and hardware rendering of surface light fields," in *Proc. ACM SIGGRAPH 2002*, July 2002, pp. 447–456.

[21] N.-M. Cheung and A. Ortega, "Distributed source coding applications to low-delay free viewpoint switching in multiview video compression," in *Proc. Picture Coding Symp. (PCS 2007)*, Nov. 2007.

[22] C. M. Christoudias, B. Georgescu, and P. Meer, "Synergism in low level vision," in *Proc. Int. Conf. Pattern Recog. (ICPR 2002)*, vol. 4, Aug. 2002, pp. 150–155.

[23] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski, "A bayesian approach to digital matting," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2001)*, vol. 2, Dec. 2001, pp. 264–271.

[24] R. T. Collins, "A space-sweep approach to true multi-image matching," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR '96)*, June 1996, pp. 358–363.

[25] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach," in *Proc. ACM SIGGRAPH 96*, Aug. 1996, pp. 11–20.

[26] P. E. Debevec, Y. Yu, and G. Borshukov, "Efficient view-dependent image-based rendering with projective texture-mapping," in *Proc. 9th Eurographics Workshop on Rendering*, June 1998, pp. 105–116.

[27] Y. Deng, Q. Yang, X. Lin, and X. Tang, "A symmetric patch-based correspondence model for occlusion handling," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV 2005)*, vol. 2, Oct. 2005, pp. 1316–1322.

[28] P. Einarsson, C.-F. Chabert, A. Jones, W.-C. Ma, B. Lamond, T. Hawkins, M. Bolas, S. Sylwan, and P. Debevec, "Relighting human locomotion with flowed reflectance fields," in *Proc. 17th Eurographics Symposium on Rendering*, June 2006.

[29] T. Fujii, "A basic study on the integrated 3D visual communication," Ph.D. dissertation, Department of Electrical Engineering, School of Engineering, The University of Tokyo, 1994 (in Japanese).

[30] T. Fujii, K. Mori, K. Takeda, K. Mase, M. Tanimoto, and Y. Suenaga, "Multipoint measuring system for video and sound — 100-camera and microphone system," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME 2006)*, July 2006, pp. 437–440.

[31] N. Fukushima, T. Yendo, T. Fujii, and M. Tanimoto, "Free viewpoint image generation using multi-pass dynamic programming," in *Proc. SPIE Stereoscopic Displays and Virtual Reality Systems XIV*, vol. 6490, Mar. 2007, pp. 460–470.

[32] T. Georgeiv, K. C. Zheng, B. Curless, D. Salesin, S. Nayar, and C. Intwala, "Spatio-angular resolution tradeoff in integral photography," in *Proc. 17th Eurographics Symposium on Rendering*, June 2006, pp. 263–272.

[33] B. Girod, A. Aaron, S. Rane, and D. Rebollo-Monedero, "Distributed video coding," *Proc. IEEE*, vol. 93, no. 1, pp. 71–83, Jan. 2005.

[34] B. Girod, C.-L. Chang, P. Ramanathan, and X. Zhu, "Light field compression using disparity-compensated lifting," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process. (ICASSP 2003)*, vol. 4, Apr. 2003, pp. 760–763.

[35] B. Goldluecke and M. Magnor, "Real-time microfacet billboarding for free-viewpoint video rendering," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2003)*, vol. 3, Sept. 2003, pp. 713–716.

[36] ——, "Space-time isosurface evolution for temporally coherent 3D reconstruction," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2004)*, vol. 1, June 2004, pp. 350–355.

[37] B. Goldluecke, M. Magnor, and B. Wilburn, "Hardware-accelerated dynamic light field rendering," in *Proc. Vision, Modeling, and Visualization (VMV 2002)*, Nov. 2002, pp. 455–462.

[38] M. Gong, "A GPU-based algorithm for estimating 3D geometry and motion in near real-time," in *Proc. 3rd Canadian Conf. Comput. Robot Vision*, June 2006.

[39] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *Proc. ACM SIGGRAPH 96*, Aug. 1996, pp. 43–54.

[40] X. Guo, Y. Lu, F. Wu, W. Gao, and S. Li, "Distributed multi-view video coding," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2006)*, vol. 6077, Jan. 2006.

[41] ——, "Free viewpoint switching in multi-view video streaming using Wyner-Ziv video coding," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2006)*, vol. 6077, Jan. 2006.

[42] S. W. Hasinoff, S. B. Kang, and R. Szeliski, "Boundary matting for view synthesis," *Comput. Vision Image Underst.*, vol. 103, no. 1, pp. 22–32, July 2006.

[43] L. Hong and G. Chen, "Segment-based stereo matching using graph cuts," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2004)*, vol. 1, July 2004, pp. 74–81.

[44] H. Hoshino, F. Okano, H. Isono, and I. Yuyama, "Analysis of resolution limitation of integral photography," *J. Opt. Soc. Am. A*, vol. 15, no. 8, pp. 2059–2065, Aug. 1998.

[45] A. Isaksen, L. McMillan, and S. J. Gortler, "Dynamically reparameterized light fields," in *Proc. ACM SIGGRAPH 2000*, July 2000, pp. 297–306.

[46] A. Jagmohan, A. Sehgal, and N. Ahuja, "Compression of lightfield rendered images using coset codes," in *Proc. Asilomar Conf. Signals, Systems, and Computers*, vol. 1, Nov. 2003, pp. 830–834.

[47] B. Javidi and F. Okano, Eds., *Three-Dimensional Television, Video, and Display Technologies*. Springer, 2002.

[48] Z. Jin, M. Yu, G. Jiang, X. Zeng, and Y.-D. Kim, "ROI-based Wyner-Ziv coding with low encoding complexity for wireless multiview video sensor array," in *Proc. Picture Coding Symp. (PCS 2006)*, Apr. 2006.

[49] A. Jones, I. McDowall, H. Yamada, M. Bolas, and P. Debevec, "Rendering for an interactive 360° light field display," *ACM Trans. Graphics*, vol. 26, no. 3, pp. 40:1–40:10, July 2007.

[50] T. Kanade, P. Rander, and P. J. Narayanan, "Virtualized reality: Constructing virtual worlds from real scenes," *IEEE Multimedia*, vol. 4, no. 1, pp. 34–47, Jan. 1997.

[51] J. H. Kim, P. Lai, J. Lopez, A. Ortega, Y. Su, P. Yin, and C. Gomila, "New coding tools for illumination and focus mismatch compensation in multiview video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1519–1535, Nov. 2007.

[52] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," in *Proc. Int. Conf. Pattern Recog. (ICPR 2006)*, vol. 3, Aug. 2006, pp. 15–18.

[53] T. Koike, M. Oikawa, K. Utsugi, M. Kobayashi, and M. Yamasaki, "Autostereoscopic display with 60 ray directions using LCD with optimized color filter layout," in *Proc. SPIE Stereoscopic Displays and Applications XVIII*, vol. 6490A, Jan. 2007.

[54] J. Konrad and M. Halle, "3-D displays and signal processing," *IEEE Signal Process. Mag.*, vol. 24, no. 6, pp. 97–111, Nov. 2007.

[55] A. Kubota, A. Smolic, M. Magnor, M. Tanimoto, T. Chen, and C. Zhang, "Multiview imaging and 3DTV," *IEEE Signal Process. Mag.*, vol. 24, no. 6, pp. 10–21, Nov. 2007.

[56] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *Int. J. Comput. Vision*, vol. 38, no. 3, pp. 199–218, July 2000.

[57] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 2, pp. 150–162, Feb. 1994.

[58] S. Laveau and O. D. Faugeras, "3-D scene representation as a collection of images," in *Proc. Int. Conf. Pattern Recog. (ICPR '94)*, vol. 1, Oct. 1994, pp. 689–691.

[59] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. ACM SIGGRAPH 96*, Aug. 1996, pp. 31–42.

[60] C.-K. Liang, T.-H. Lin, B.-Y. Wong, C. Liu, and H. H. Chen, "Programmable aperture photography: Multiplexed light field acquisition," *ACM Trans. Graphics*, vol. 27, no. 3, pp. 55:1–55:10, Aug. 2008.

[61] H. Liao, M. Iwahara, T. Koike, N. Hata, I. Sakuma, and T. Dohi, "Scalable high-resolution integral videography autostereoscopic display with a seamless multiprojection system," *Appl. Opt.*, vol. 44, no. 3, pp. 305–315, Jan. 2005.

[62] J. Lim, K. N. Ngan, W. Yang, and K. Sohn, "A multiview sequence CODEC with view scalability," *EURASIP Signal Process.: Image Commun.*, vol. 19, no. 3, pp. 239–256, Mar. 2004.

[63] G. Lippmann, "Epreuves reversibles donnant la sensation du relief," *J. Phys.*, vol. 7, no. 4, pp. 821–825, Nov. 1908.

[64] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[65] M. E. Lukacs, "Predictive coding of multi-viewpoint image sets," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process. (ICASSP '86)*, 1986, pp. 521–524.

[66] M. Magnor, P. Eisert, and B. Girod, "Model-aided coding of multi-viewpoint image data," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2000)*, Sept. 2000, pp. 919–922.

[67] M. Magnor and B. Girod, "Hierarchical coding of light fields with disparity maps," in *Proc. IEEE Int. Conf. Image Process. (ICIP '99)*, Oct. 1999, pp. 334–338.

[68] ——, "Data compression for light-field rendering," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 3, pp. 338–343, Apr. 2000.

[69] ——, "Model-based coding of multi-viewpoint imagery," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2000)*, vol. 4067, June 2000, pp. 14–22.

[70] M. Magnor, P. Ramanathan, and B. Girod, "Multi-view coding for image-based rendering using 3-D scene geometry," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1092–1106, Nov. 2003.

[71] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D warping," in *Proc. Symposium on Interactive 3D Graphics*, Apr. 1997, pp. 7–16.

[72] S. Mattoccia, F. Tombari, and L. D. Stefano, "Stereo vision enabling precise border localization within a scanline optimization framework," in *Proc. 8th Asian Conf. Comput. Vision (ACCV 2007)*, Nov. 2007, pp. 517–527.

[73] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan, "Image-based visual hulls," in *Proc. ACM SIGGRAPH 2000*, July 2000, pp. 369–374.

[74] W. Matusik and H. Pfister, "3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 814–824, Aug. 2004.

[75] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," in *Proc. ACM SIGGRAPH 95*, Aug. 1995, pp. 39–46.

[76] P. Merkle, A. Smolic, K. Muller, and T. Wiegand, "Efficient prediction structures for multiview video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1461–1473, Nov. 2007.

[77] G. Miller, S. Rubin, and D. Ponceleon, "Lazy decompression of surface light fields for precomputed global illumination," in *Proc. 9th Eurographics Workshop on Rendering*, June 1998, pp. 281–292.

[78] G. Minami, Z. Xiong, A. Wang, and S. Mehrotra, "3-D wavelet coding of video with arbitrary regions of support," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 9, pp. 1063–1068, Sept. 2001.

[79] S. Mitsuda, T. Yamamoto, K. Takahashi, T. Naemura, and H. Harashima, "Interactive view synthesis from integral photography using estimated depth information," in *Proc. SPIE Three-Dimensional TV, Video, and Display II*, vol. 5243, Sept. 2003, pp. 116–124.

[80] T. Naemura and H. Harashima, "Real-time video-based rendering for augmented spatial communication," in *Proc. SPIE Visual Commun. Image Process. (VCIP '99)*, vol. 3653, Jan. 1999, pp. 620–631.

[81] T. Naemura, J. Tago, and H. Harashima, "Real-time video-based modeling and rendering of 3D scenes," *IEEE Comput. Graph. Appl.*, vol. 22, no. 2, pp. 66–73, Mar. 2002.

[82] R. Ng, M. Levoy, M. Bredif, G. Duval, M. Horowitz, and P. Hanrahan, "Light field photography with a hand-held plenoptic camera," Stanford University Computer Science, Tech. Rep. CSTR 2005-02, Apr. 2005.

[83] Y. Nomura, L. Zhang, and S. K. Nayar, "Scene collages and flexible camera arrays," in *Proc. 18th Eurographics Symposium on Rendering*, June 2007, pp. 127–138.

[84] F. Okano, J. Arai, H. Hoshino, and I. Yuyama, "Three-dimensional video system based on integral photography," *Opt. Eng.*, vol. 38, no. 6, pp. 1072–1077, June 1999.

[85] T. Okoshi, *Three-Dimensional Imaging Techniques*. Academic Press, 1976.

[86] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 4, pp. 353–363, Apr. 1993.

[87] R. Olsson, M. Sjostrom, and Y. Xu, "A combined pre-processing and H.264-compression scheme for 3D integral images," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2006)*, Oct. 2006, pp. 513–516.

[88] R. Otsuka, T. Hoshino, and Y. Horry, "Transpost: A novel approach to the display and transmission of 360 degrees-viewable 3D solid images," *IEEE Trans. Visual. Comput. Graphics*, vol. 12, no. 2, pp. 178–185, Mar.–Apr. 2006.

[89] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): Design and construction," *IEEE Trans. Inf. Theory*, vol. 49, no. 3, pp. 626–643, Mar. 2003.

[90] P. Ramanathan and B. Girod, "Rate-distortion analysis for light field coding and streaming," *EURASIP Signal Process.: Image Commun.*, vol. 21, no. 6, pp. 462–475, July 2006.

[91] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Trans. Commun.*, vol. 48, no. 6, pp. 948–959, June 2000.

[92] M. A. Ruzon and C. Tomasi, "Alpha estimation in natural images," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2000)*, vol. 1, June 2000, pp. 18–25.

[93] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243–250, June 1996.

[94] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, no. 1–3, pp. 7–42, Apr.–June 2002. [Online]. Available: http://vision.middlebury.edu/stereo/

[95] H. Schirmacher, M. Li, and H.-P. Seidel, "On-the-fly processing of generalized lumigraphs," in *Proc. Eurographics 2001*, vol. 20, no. 3, 2001, pp. 165–173.

[96] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR '97)*, June 1997, pp. 1067–1073.

[97] ——, "View morphing," in *Proc. ACM SIGGRAPH 96*, Aug. 1996, pp. 21–30.

[98] J. Shade, S. Gortler, L.-W. He, and R. Szeliski, "Layered depth images," in *Proc. ACM SIGGRAPH 98*, July 1998, pp. 231–242.

[99] S. Shimizu, M. Kitahara, K. Kamikura, and Y. Yashima, "Multi-view video coding based on 3-D warping with depth map," in *Proc. Picture Coding Symp. (PCS 2006)*, Apr. 2006.

[100] H.-Y. Shum, S. B. Kang, and S.-C. Chan, "Survey of image-based representations and compression techniques," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1020–1037, Nov. 2003.

[101] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," in *Proc. ACM SIGGRAPH 99*, Aug. 1999, pp. 299–306.

[102] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. 19, no. 4, pp. 471–480, July 1973.

[103] A. Smolic, K. Muller, N. Stefanoski, J. Ostermann, A. Gotchev, G. B. Akar, G. Triantafyllidis, and A. Koz, "Coding algorithms for 3DTV — a survey," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1606–1621, Nov. 2007.

[104] J. Sun, Y. Li, S. B. Kang, and H.-Y. Shum, "Symmetric stereo matching for occlusion handling," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2005)*, vol. 2, June 2005, pp. 399–406.

[105] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 7, pp. 787–800, July 2003.

[106] R. Szeliski and H.-Y. Shum, "Creating full view panoramic image mosaics and environment maps," in *Proc. ACM SIGGRAPH 97*, Aug. 1997, pp. 251–258.

[107] K. Takahashi, A. Kubota, and T. Naemura, "All in-focus view synthesis from under-sampled light fields," in *Proc. Int. Conf. Artificial Reality and Telexistence (ICAT 2003)*, Dec. 2003, pp. 249–256.

[108] ——, "A focus measure for light field rendering," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2004)*, vol. 4, Oct. 2004, pp. 2475–2478.

[109] K. Takahashi and T. Naemura, "Layered light-field rendering with focus measurement," *EURASIP Signal Process.: Image Commun.*, vol. 21, no. 6, pp. 519–530, July 2006.

[110] M. Tanimoto, "FTV (free viewpoint television) creating ray-based image engineering," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2005)*, vol. 2, Oct. 2005, pp. 25–28.

[111] H. Tao, H. S. Sawhney, and R. Kumar, "A global matching framework for stereo computation," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV 2001)*, vol. 1, July 2001, pp. 532–539.

[112] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV 2003)*, vol. 2, Oct. 2003, pp. 900–906.

[113] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[114] R. Y. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE J. Robot. Autom.*, vol. 3, no. 4, pp. 323–344, Aug. 1987.

[115] V. Vaish, B. Wilburn, N. Joshi, and M. Levoy, "Using plane + parallax for calibrating dense camera arrays," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2004)*, vol. 1, June 2004, pp. 2–9.

[116] D. Varodayan, A. Aaron, and B. Girod, "Rate-adaptive codes for distributed source coding," *EURASIP Signal Process.*, vol. 86, no. 11, pp. 3123–3130, Nov. 2006.

[117] S. Vedula, S. Baker, S. Seitz, and T. Kanade, "Shape and motion carving in 6D," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2000)*, vol. 2, June 2000, pp. 592–598.

[118] S. Vedula, S. Baker, and T. Kanade, "Spatio-temporal view interpolation," in *Proc. 13th Eurographics Workshop on Rendering*, June 2002, pp. 65–76.

[119] A. Veeraraghavan, R. Raskar, A. Agrawal, A. Mohan, and J. Tumblin, "Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing," *ACM Trans. Graphics*, vol. 26, no. 3, pp. 69:1–69:12, Aug. 2007.

[120] J. Y. A. Wang and E. H. Adelson, "Representing moving images with layers," *IEEE Trans. Image Process.*, vol. 3, no. 5, pp. 625–638, Sept. 1994.

[121] Y. Wei and L. Quan, "Region-based progressive stereo matching," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2004)*, vol. 1, July 2004, pp. 106–113.

[122] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz, "High-speed videography using a dense camera array," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2004)*, vol. 2, July 2004, pp. 294–301.

[123] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 765–776, July 2005.

[124] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. A. Horowitz, "The light field video camera," in *Proc. SPIE Media Processors 2002*, vol. 4674, Jan. 2002, pp. 29–36.

[125] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle, "Surface light fields for 3D photography," in *Proc. ACM SIGGRAPH 2000*, July 2000, pp. 287–296.

[126] A. D. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Trans. Inf. Theory*, vol. 22, no. 1, pp. 1–10, Jan. 1976.

[127] W. Xiong and J. Jia, "Stereo matching on objects with fractional boundary," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2007)*, June 2007.

[128] K. Yamamoto, M. Kitahara, H. Kimata, T. Yendo, T. Fujii, M. Tanimoto, S. Shimizu, K. Kamikura, and Y. Yashima, "Multiview video coding using view interpolation and color correction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 11, pp. 1436–1449, Nov. 2007.

[129] T. Yamamoto and T. Naemura, "Real-time capturing and interactive synthesis of 3D scenes using integral photography," in *Proc. SPIE Stereoscopic Displays and Virtual Reality Systems XI*, vol. 5291, Jan. 2004, pp. 155–166.

[130] T. Yamamoto, M. Kojima, and T. Naemura, "LIFLET: Light field live with thousands of lenslets," *ACM SIGGRAPH 2004 Emerging Technologies*, Aug. 2004.

[131] J. C. Yang, M. Everett, C. Buehler, and L. McMillan, "A real-time distributed light field camera," in *Proc. 13th Eurographics Workshop on Rendering*, June 2002, pp. 77–85.

[132] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation and occlusion handling," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2006)*, vol. 2, June 2006, pp. 2347–2354.

[133] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2007)*, June 2007.

[134] R. Yang, G. Welch, and G. Bishop, "Real-time consensus-based scene reconstruction using commodity graphics hardware," in *Proc. Pacific Graphics 2002*, Oct. 2002, pp. 225–235.

[135] R. Yang, X. Huang, S. Li, and C. Jaynes, "Toward the light field display: Autostereoscopic rendering via a cluster of projectors," *IEEE Trans. Visual. Comput. Graphics*, vol. 14, no. 1, pp. 84–96, Jan.–Feb. 2008.

[136] T. Yendo, N. Kawakami, and S. Tachi, "Seelinder: The cylindrical lightfield display," *ACM SIGGRAPH 2005 Emerging Technologies*, Aug. 2005.

[137] S. Yeom, A. Stern, and B. Javidi, "Compression of 3D color integral images," *Opt. Exp.*, vol. 12, no. 8, pp. 1632–1642, Apr. 2004.

[138] L. Zelnik-Manor and P. Perona, "Automating joiners," in *Proc. Int. Symp. Non-Photorealistic Animation and Rendering (NPAR 2007)*, Aug. 2007, pp. 121–131.

[139] C. Zhang and T. Chen, "A self-reconfigurable camera array," in *Proc. 15th Eurographics Symposium on Rendering*, June 2004, pp. 243–254. [Online]. Available: http://amp.ece.cmu.edu/projects/MobileCamArray/

[140] ——, "A survey on image-based rendering — representation, sampling and compression," *EURASIP Signal Process.: Image Commun.*, vol. 19, no. 1, pp. 1–28, Jan. 2004.

[141] C. Zhang and J. Li, "Compression and rendering of concentric mosaics with reference block codec (RBC)," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2000)*, vol. 4067, June 2000, pp. 43–55.

[142] ——, "Compression of lumigraph with multiple reference frame (MRF) prediction and just-in-time rendering," in *Proc. Data Compression Conf. (DCC 2000)*, Mar. 2000, pp. 253–262.

[143] ——, "Interactive browsing of 3D environment over the internet," in *Proc. SPIE Visual Commun. Image Process. (VCIP 2001)*, vol. 4310, Jan. 2001, pp. 509–520.

[144] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.

[145] X. Zhu, A. Aaron, and B. Girod, "Distributed compression for large camera arrays," in *Proc. IEEE Workshop on Statistical Signal Process. (SSP 2003)*, Sept. 2003, pp. 30–33.

[146] G. Ziegler, H. P. A. Lensch, N. Ahmed, M. Magnor, and H.-P. Seidel, "Multi-video compression in texture space," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2004)*, Oct. 2004, pp. 2467–2470.

[147] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 600–608, Aug. 2004.

[148] C. L. Zitnick, N. Jojic, and S. B. Kang, "Consistent segmentation for optical flow estimation," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV 2005)*, vol. 2, Oct. 2005, pp. 1308–1315.

[149] C. L. Zitnick and S. B. Kang, "Stereo for image-based rendering using image oversegmentation," *Int. J. Comput. Vision*, vol. 75, no. 1, pp. 49–65, Oct. 2007.

[150] M. Zwicker, A. Vetro, S. Yea, W. Matusik, H. Pfister, and F. Durand, "Resampling, antialiasing, and compression in multiview 3-D displays," *IEEE Signal Process. Mag.*, vol. 24, no. 6, pp. 88–96, Nov. 2007.

[151] "Information technology — digital compression and coding of continuous-tone still images — requirements and guidelines," ITU-T Recommendation T.81, Sept. 1992.

[152] Cg. [Online]. Available: http://developer.nvidia.com/page/cg_main.html

[153] JasPer software 1.701.0. [Online]. Available: http://www.ece.uvic.ca/~mdadams/jasper/

[154] MPEG-2 Test Model 5. [Online]. Available: http://www.mpeg.org/MPEG/MSSG/tm5/

[155] QccPack — quantization, compression, and coding library. [Online]. Available: http://qccpack.sourceforge.net/

List of Publications

Journal Papers

[1] Yuichi Taguchi, Takafumi Koike, Keita Takahashi, and Takeshi Naemura, “TransCAIP: A live 3D TV system using a camera array and an integral photography display with interactive control of viewing parameters,” accepted to *IEEE Trans. Visual. Comput. Graphics*.

[2] Yuichi Taguchi and Takeshi Naemura, “Rendering-oriented decoding for a distributed multi-view coding system using a coset code,” accepted to *EURASIP J. Image Video Process. (Special Issue on Distributed Video Coding)*.

[3] Yuichi Taguchi, Keita Takahashi, and Takeshi Naemura, “Design and implementation of a real-time all-in-focus video-based rendering system using a network camera array,” submitted to *IEICE Trans. Inf. Syst.*.

[4] 金 時煥, 河 宗玄, 田口 裕一, 高橋 桂太, 苗村 健, “自由視点画像合成における見え方を考慮したレンズアレイ撮像系の設定”, 映像情報メディア学会誌, vol. 60, no. 10, pp. 1658–1663, Oct. 2006.

[5] 田口 裕一, 苗村 健, “自由視点画像合成に基づく光線空間符号化”, 映像情報メディア学会誌, vol. 60, no. 4, pp. 569–575, Apr. 2006.

International Conferences

[6] Yuichi Taguchi, Takafumi Koike, Keita Takahashi, and Takeshi Naemura, “TransCAIP: Live transmission of light field from a camera array to an integral photography display,” *ACM SIGGRAPH ASIA 2008 Emerging Technologies*, Dec. 2008.

[7] Yuichi Taguchi, Bennett Wilburn, and C. Lawrence Zitnick, “Stereo reconstruction with mixed pixels using adaptive over-segmentation,” in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR 2008)*, June 2008.

[8] Yuichi Taguchi, Keita Takahashi, and Takeshi Naemura, "Real-time all-in-focus video-based rendering using a network camera array," in *Proc. 3DTV-Conference 2008*, May 2008, pp. 241–244.

[9] Yuichi Taguchi and Takeshi Naemura, "Rendering-oriented decoding for distributed multi-view coding system," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2007)*, vol. 1, Sept. 2007, pp. 213–216.

[10] Toru Ando, Yuichi Taguchi, and Takeshi Naemura, "GPU-oriented light field compression for real-time streaming," *ACM SIGGRAPH 2007 Posters*, no. 70, Aug. 2007.

[11] Yuichi Taguchi and Takeshi Naemura, "View-dependent coding of light fields based on free-viewpoint image synthesis," in *Proc. IEEE Int. Conf. Image Process. (ICIP 2006)*, Oct. 2006, pp. 509–512.

[12] Yuichi Taguchi, Keita Takahashi, and Takeshi Naemura, "View-dependent scalable coding of light fields using ROI-based techniques," in *Proc. SPIE Three-Dimensional TV, Video, and Display V*, vol. 6392, Oct. 2006.

[13] Yuichi Taguchi and Takeshi Naemura, "Free-viewpoint thumbnail for light field compression," *ACM SIGGRAPH 2005 Posters*, no. 63, Aug. 2005.

[14] Yuichi Taguchi, Shunsuke Saruwatari, Mikio Hasegawa, Masugi Inoue, Hiroyuki Morikawa, and Tomonori Aoyama, "U[1]-Chip: Wireless communication module for fast service discovery," in *Adjunct Proc. 6th Int. Conf. Ubiquitous Computing (UbiComp 2004)*, Interactive Posters, Sept. 2004.

Domestic Conferences

[15] 永塚 遼, 田口 裕一, 苗村 健, "裸眼立体映像システムにおけるオブジェクトの切り抜きと見え方の操作", 信学技報, IE2008-204, Feb. 2009, pp. 1–6.

[16] 田口 裕一, 山本 和明, 小池 崇文, 高橋 桂太, 苗村 健, "TransCAIP: カメラアレイからインテグラルフォトグラフィディスプレイへのインタラクティブな 3 次元映像提示", 3 次元画像コンファレンス 2008, July 2008, P-10, pp. 121–124. **優秀論文賞 受賞**

[17] 田口 裕一, 高橋 桂太, 苗村 健, "ネットワークカメラアレイを用いた実時間全焦点自由視点映像合成システム", 情処研報, CVIM-162-14, Mar. 2008, pp. 79–86.

[18] 安藤 徹, 田口 裕一, 苗村 健, "グラフィクスプロセッサを用いた 3 次元空間情報の実時間符号化手法の検討", 3 次元画像コンファレンス 2007, July 2007, 4-3, pp. 67–70.

[19] 王 金戈, 田口 裕一, 高橋 桂太, 苗村 健, “実時間自由視点画像合成のためのカメラアレイシステムの構築とキャリブレーション手法の検討”, 3 次元画像コンファレンス 2007, July 2007, P-1, pp. 97–100. **優秀論文賞 受賞**

[20] 田口 裕一, 苗村 健, “Distributed Coding と自由視点画像合成の連携に関する一検討”, 画像符号化シンポジウム (PCSJ 2006), Nov. 2006, P-1.01, pp. 29–30.

[21] 田口 裕一, 高橋 桂太, 苗村 健, “光線空間の視点依存階層符号化の基礎検討”, 3 次元画像コンファレンス 2006, July 2006, 5-2, pp. 77–80.

[22] 金 時煥, 河 宗玄, 田口 裕一, 高橋 桂太, 苗村 健, “自由視点画像合成における見え方を考慮したレンズアレイ撮像系の設計”, 3 次元画像コンファレンス 2006, July 2006, 5-1, pp. 73–76.

[23] 金 時煥, 河 宗玄, 田口 裕一, 高橋 桂太, 苗村 健, “自由視点画像合成における見え方の変化を考慮したレンズアレイ撮像系の設計に関する基礎検討”, 信学総大, Mar. 2006, D-11-63.

[24] 田口 裕一, 苗村 健, “自由視点画像合成に基づく光線空間符号化 — カメラアレイ入力への適用 —”, 画像符号化シンポジウム (PCSJ 2005), Nov. 2005, P-2.12, pp. 31–32.

[25] 高橋 桂太, 田口 裕一, 呉 炳俊, 一松 隆平, 飯田 誠, 苗村 健, “空間共有通信のための多眼カメラアレイ構築に向けた基礎検討”, 日本バーチャルリアリティ学会第 10 回大会, Sept. 2005, 3A2-2, pp. 475–476.

[26] 田口 裕一, 苗村 健, “自由視点画像合成に基づく光線空間符号化手法の提案と評価方法の検討”, 第 4 回情報科学技術フォーラム (FIT 2005), Sept. 2005, J-009.

[27] 田口 裕一, 苗村 健, “光線空間符号化のための自由視点画像合成に関する検討”, 3 次元画像コンファレンス 2005, July 2005, 1-3, pp. 9–12.

[28] 田口 裕一, 苗村 健, “自由視点画像合成に基づく光線空間符号化方式の基礎検討”, 信学総大, Mar. 2005, D-11-144. **学術奨励賞 受賞**

[29] 田口 裕一, 猿渡 俊介, 長谷川 幹雄, 川原 圭博, 井上 真杉, 森川 博之, 青山 友紀, “U^1-Chip: インスタントサービス実現に向けての無線通信モジュール”, 信学総大, Mar. 2004, B-15-40.