Design and Analysis of a Two-Dimensional Camera Array

by

Jason Chieh-Sheng Yang

B.S., Massachusetts Institute of Technology (1999)

M.Eng., Massachusetts Institute of Technology (2000)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
February 2005

Author .....
Department of Electrical Engineering and Computer Science
February 16, 2005

Certified by .....
Leonard McMillan
Associate Professor of Computer Science
[illegible] apel Hill
upervisor

Accepted by ....
[illegible] C. Smith
Chairman, Committee on Graduate Students
Department of Electrical Engineering and Computer Science

Design and Analysis of a Two-Dimensional Camera Array

by

Jason Chieh-Sheng Yang

Submitted to the Department of Electrical Engineering and Computer Science
on February 16, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

I present the design and analysis of a two-dimensional camera array for virtual studio applications. It is possible to substitute conventional cameras and motion control devices with a real-time, light field camera array. I discuss a variety of camera architectures and describe a prototype system based on the "finite-viewpoints" design that allows multiple viewers to navigate virtual cameras in a dynamically changing light field captured in real time. The light field camera consists of 64 commodity video cameras connected to off-the-shelf computers. I employ a distributed rendering algorithm that overcomes the data bandwidth problems inherent in capturing light fields by selectively transmitting only those portions of the video streams that contribute to the desired virtual view.

I also quantify the capabilities of a virtual camera rendered from a camera array in terms of the range of motion, range of rotation, and effective resolution. I compare these results to other configurations. From this analysis I provide a method for camera array designers to select and configure cameras to meet desired specifications. I demonstrate the system and the conclusions of the analysis with a number of examples that exploit dynamic light fields.

Thesis Supervisor: Leonard McMillan
Title: Associate Professor of Computer Science

Acknowledgments

I would first like to thank Professor Leonard McMillan. His guidance over the years has been invaluable. I especially want to thank him for his considerable support both in research and in life.

I would also like to thank Chris Buehler for his help on the camera system and algorithms and for our many discussions. My research would not have been possible without the assistance of Matthew Everett and Jingyi Yu and also Manu Seth and Nathan Ackerman.

Thanks to my committee members, Professors Seth Teller and Trevor Darrell, for their time and input to the thesis. And thanks to the entire Computer Graphics Group.

Finally, I would like to express my deepest gratitude to my parents, my brother, and my wife Maria for their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in computer generated imagery (CGI) have enabled an unprecedented level of realism in the virtual environments created for various visual mediums (e.g., motion picture, television, advertisements, etc.). Alongside imagined worlds and objects, artists have also created synthetic versions of real scenes. One advantage in modeling rather than filming, a building or a city for example, is the ease in manipulation, such as adding special effects in post-production. Another advantage is the flexibility in generating virtual camera models and camera movements versus the limited capabilities of conventional cameras.

The traditional method of generating images at virtual viewpoints is through the computation of the light interaction between geometric models and other scene elements. To render real-world scenes, a geometric model must be constructed. The most straightforward approach is to create models by hand. Another method is to employ computer vision algorithms to automatically generate geometry from images. However, these algorithms are often difficult to use and prone to errors.

Recently, an alternative approach has been introduced where scenes are rendered directly from acquired images with little or no geometry information. Multiple images of an environment or subject are captured, encapsulating all scene elements and features from geometry to light interaction. These images are treated as sampled rays rather than pixels, and virtual views are rendered by querying and interpolating the sampled data.

Image-based rendering has found widespread use because of the ease with which it produces photorealistic imagery. Commercial examples of image-based rendering systems can be found in movies and television with the most popular being the "time-freezing" effects of the motion picture *The Matrix*. Subjects are captured using a multi-camera configuration usually along a line or a curve. The primary application is to take a snap shot of a dynamic scene and move a synthetic camera along a pre-determined path.

Light field[17] and lumigraph[10] techniques are limited to static scenes, in large part because a single camera is often used to capture the images. However, unlike the previous methods, the cameras are not restricted to a linear path, but lie on a 2D plane. This allows for the free movement of the virtual camera beyond the sampling plane.

A logical extension to a static light field or lumigraph is to construct an image-based rendering system with multiple video cameras, thus allowing rendering in an evolving, photorealistic environment. The challenge of such a system, as well as the main disadvantage of light fields and lumigraphs in general, is managing the large volume of data.

This thesis will be about the development and analysis of dynamic, camera array systems with the introduction of a prototype system that interactively renders images from a light field captured in real time.

1.1 Motivation

The development of a real-time, light field camera is motivated by the virtual studio application. The goal of a virtual studio is to replace conventional, physical cameras with virtual cameras that can be rendered anywhere in space using a light field captured by an array of cameras. Video streams can be rendered and recorded in real time as the action is happening, or the light field can be stored for off-line processing.

Virtual cameras can do everything normal cameras can do such as zoom, pan, and tilt. For live events, the director or camera operator can position virtual cameras in

space just like normal cameras. Cameras can also be generated and destroyed at will.

Nowhere is the virtual studio more relevant than in an actual studio environment, especially for filming special effects. As an example, the movie *Sky Captain and the World of Tomorrow* is the first time where everything is digitally created except for the principle actors who are captured on film in front of a blue screen. In order to precisely merge the actors into the synthetic world the camera positions and motions during rendering and filming must match. This requires time and coordination. "The six week shooting schedule required a new camera setup every twelve and a half minutes." [9] In a virtual studio, the work of positioning cameras would not be needed. The entire light field video could be saved, and later any camera motion could be recreated. Ideally, the actors are positioned in a known coordinate system relative to the camera array. Then the mapping from the actor's space to the light field space is simply determined.

Sometimes storing the entire light field is not possible or needed, such as filming a television broadcast (e.g., a TV sitcom or drama). In this case the virtual camera can be previewed and the camera's motions programmed interactively. The video stream can then be directly recorded.

Beyond the virtual studio, a camera array can also be used in remote viewing applications such as teleconferencing or virtual tours.

1.2 Contribution

This thesis is about the development and analysis of camera array systems. Part of this includes the construction of a prototype camera array to interactively render images in real time. Several similar systems have been demonstrated before, but due to the large volume of data, relatively small numbers of widely spaced cameras are used. Light field techniques are not directly applicable in these configurations, so these systems generally use computer vision methods to reconstruct a geometric model of the scene to use for virtual view rendering. This reconstruction process is difficult and computationally expensive.

I propose a scalable architecture for a distributed image-based rendering system based on a large number of densely spaced video cameras. By using such a configuration of cameras, one can avoid the task of dynamic geometry creation and instead directly apply high-performance light field rendering techniques. A novel distributed light field rendering algorithm is used to reduce bandwidth issues and to provide a scalable system.

I will also analyze the capabilities of camera arrays. The analysis of light fields and their reconstruction has largely been limited to sampling of the camera plane. Instead, I analyze the sampling of the focal plane and how it relates to the resolution of the virtual cameras. In addition I will quantify the capabilities of the virtual camera in terms of movement and resolution. A related topic is the construction of a camera array given a desired specification.

To conclude, I will present examples of using a dynamic light field in a virtual studio setting. This will include traditional, image-based special effect applications. I will also demonstrate the simulation of motion controlled cameras. Finally, I will introduce other examples of using light fields such as mixing light fields, immersive environments, and as a replacement to environment mapping.

To summarize, the central thesis to my dissertation is that:
An interactive, real-time light field video camera array can substitute conventional cameras and motion control devices.

The following are my main contributions:

- I introduce a system architecture that allows multiple viewers to independently navigate a dynamic light field.

- I describe the implementation of a real-time, distributed light field camera consisting of 64 commodity video cameras arranged in a dense grid array.

- I introduce a distributed light field rendering algorithm with the potential of using bandwidth proportional to the number of viewers.

- I quantify the capabilities of a virtual camera in the environment of a light field

in terms of the possible camera positions and orientations and also the effective resolution.

- I describe how to design a camera array to match desired specifications.
- I demonstrate several sample applications.

1.3 Thesis Overview

Chapter 2 will cover the background materials related to image-based rendering, specifically to light fields and lumigraphs, image reconstruction, and light field sampling. I will also cover existing camera systems from static to dynamic systems and linear to higher-dimensional camera configurations. Chapter 3 will begin by looking at design goals for camera arrays and then discuss different architectures that meet those goals. I will also discuss a possible ideal system that encompasses many desired features. Chapter 4 introduces the prototype system. I will cover the construction and the rendering process, and I will also analyze the performance. Chapter 5 discusses virtual camera capabilities first by looking at the possible range of motions, then by the possible range of rotations, and finally the effective resolution. Chapter 6 makes use of the conclusions of Chapter 5 by describing the process of designing a camera array to meet certain specifications. I will also discuss how to capture an immersive environment and the applications for them. Chapter 7 covers the rendering results and potential uses of dynamic light fields. Finally, Chapter 8 summarizes my results and discusses future work.

Chapter 2

Previous Work

In the area of light fields and image based rendering there has been a great deal of research since the original Light Field Rendering [17] and The Lumigraph [10] papers. However, with regards to real-time camera array systems there is little prior work. In this chapter I will first discuss light field rendering in relation to camera systems, and in the process, I will define standard terminology and practices. Next, I introduce the analysis of light fields important in the construction of capture systems and rendering of images. Finally, I describe the progression of camera array systems for image based rendering from linear to planar arrays and static to video cameras.

2.1 Light Fields

A light field is an image-based scene representation that uses rays instead of geometries. It was first described by Levoy and Hanrahan [17] and Gortler et al. [10]. In a light field, rays are captured from a sequence of images taken on a regularly sampled 2D grid. This ray database, or "light field slab" [17], consists of all rays that intersect two parallel planes, a front and back plane.

Rays are indexed using a 4D or two plane parameterization (s, t, u, v). (Figure 2-1) I will consider the ST plane as the camera plane, where the real or sampling cameras are positioned to capture the scene. The UV plane can be thought of as the image or focal plane. For the rest of this thesis I will use this two plane formulation.

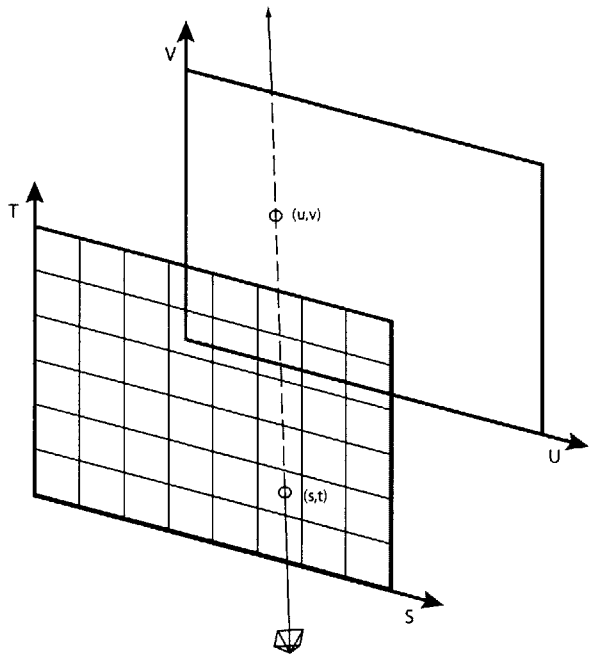


Figure 2-1: Two Plane Paramaterization. A ray from the virtual camera intersects the light field at (s, t, u, v) which are the points of intersection with the front plane (ST) and back plane (UV). Front plane can be referred to as the camera plane because cameras are positioned there to capture the scene.

Cameras on the ST plane will sometimes be referred to as the real cameras whether they are actual physical cameras or camera eye positions when rendering synthetic scenes. I will refer to eye positions and orientations used to render new images as virtual cameras.

Since the collection of rays in a space is formulated as essentially a ray database, rendering a novel view is as simple as querying for a desired ray. (Figure 2-1) To render an image using the light field parameterization, a ray is projected from the virtual camera to the parallel ST and UV planes. The point of intersection with the ST plane indicates the camera that sampled the ray, and the intersection with the UV plane determines the pixel sample. The (s, t, u, v) coordinates are the 4D parameters used to index the database where the ray's color is returned. Of course, the light field is only a sampled representation of a continuous space, therefore rendered rays must be interpolated from existing rays.

2.2 Sampling

As shown in 2.1, discrete sampling on the ST plane will lead to aliasing or ghosting of the rendered image. Levoy and Hanrahan [17] solved this by prefiltering the original images. I will discuss reconstruction more in Section 2.3. Another method of improving image quality is by increasing the sampling density of the ST plane. For light field rendering with no pre-determined geometry, Choi et al. [5] gives the minimum sampling required such that there is no aliasing or when the disparity, defined as the projection error using the optimal reconstruction filter, is less than one pixel. Ignoring scene features, [5] gives the minimum ST sampling or the maximum spacing between cameras as

$$\Delta ST_{\max} = \frac{2\Delta v}{f\left(\frac{1}{z_{\min}} - \frac{1}{z_{\max}}\right)} \tag{2.1}$$

where Δv is the resolution of the camera in terms of the pixel spacing, f is the focal length, and z_{min} and z_{max} are the minimum and maximum acceptable rendering distances. Also, if using a single focal plane the optimal position of this plane is

$$\frac{1}{z_{opt}} = \frac{1}{2} * \left(\frac{1}{z_{\min}} + \frac{1}{z_{\max}}\right) \tag{2.2}$$

2.3 Reconstruction Algorithms

Without a densely sampled ST plane, rendering images with no ghosting artifacts requires knowledge of the scene geometry. Choi et al. [5] demonstrated that with more depth information the less densely sampled the ST plane needs to be. However, with a physical camera array capturing real world scenes, depth is often difficult to acquire especially in real time. In this section, I discuss methods for improving reconstruction in the absence of depth information. There have been significant research in this area [6, 32], but I will be discussing work most related to this thesis.

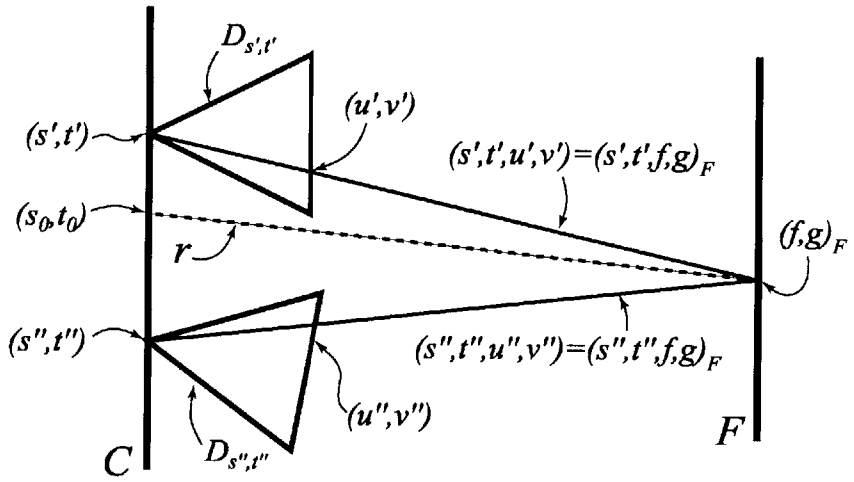


Figure 2-2: Reconstruction of the desired ray r. F is a variable focal plane. (s_0, t_0) and (f, g) are the points of intersection with the camera plane and the focal plane respectively. For each sampling camera D, the (u, v) coordinate representing (f, g) in the local image plane to that camera is solved. This represents the sampled contribution to the desired ray. [13]

2.3.1 Dynamically Reparamaterized Light Fields

Levoy and Hanrahan [17] used two fixed ST and UV planes in their parameterization. This leads to aliasing if the UV plane does not lie on the surface of the object. Isaksen et al. [13] introduced a variable focal plane for reconstruction through dynamic reparamaterization. From Figure 2-2, a desired ray intersects with the camera plane and a focal plane that can vary in position. Intersections with the camera plane (s, t), and the focal plane (f, g) are calculated. Since the cameras are fixed, a mapping is known that gives the corresponding (u, v) coordinate on the image plane of the sampling cameras for a (f, g) coordinate on the variable focal plane. These samples will contribute to the reconstruction of the desired ray. The conceptof using a variable focal plane for reconstruction will appear often in the following chapters.

2.3.2 Unstructured Lumigraph Rendering

Another limitation of light field rendering is that there is an implicit assumption that the light field was captured with cameras positioned on a strict plane. Otherwise,

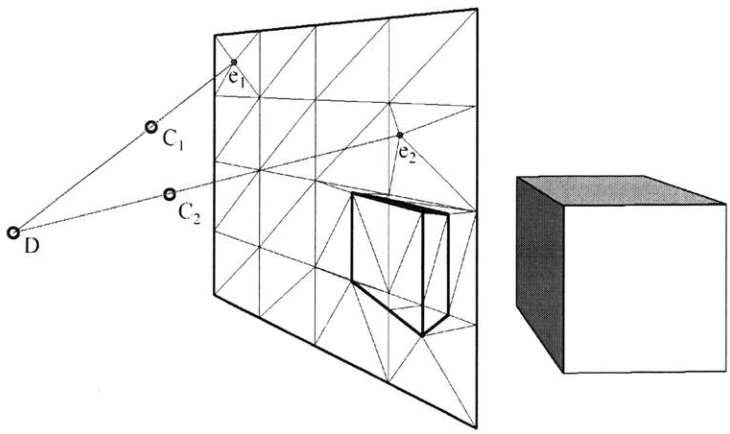


Figure 2-3: Unstructured lumigraph rendering uses an image plane (blending field) triangulated from the projections of the geometry proxy, the camera centers, and a regular grid. [4]

as is done by [17], the images must be rectified into the two plane arrangement. However, as I will later demonstrate, a physical camera system cannot guarantee such an arrangement. "Unstructured lumigraph rendering" [4] (Figure 2-3) is a generalized image based rendering algorithm that allows for cameras to be in general position. For each pixel to be rendered, the contribution from every camera is weighted by the angular deviation to the desired ray intersecting the geometry proxy, resolution mismatch, and field of view.

$$\text{penalty}_{comb}(i) = \alpha\text{penalty}_{ang}(i) + \beta\text{penalty}_{res}(i) + \gamma\text{penalty}_{fov}(i) \tag{2.3}$$

This process is optimized by using a camera blending field formed by projecting camera positions, the geometry proxy, and a regular sampling grid onto the image plane. The contribution weights are then calculated at these locations and finally rendered by using projective texture mapping.

Unstructured lumigraph rendering will be the basis for the rendering algorithm used in the prototype camera system I introduce in Chapter 4.

2.4 Camera Arrays

There are many examples of multi-camera systems for image-based rendering beyond light fields. In this section I will discuss a range of acquisition systems from static to dynamic cameras and also camera systems for light field rendering.

2.4.1 Static Camera Systems

Static camera systems capture light fields of static scenes (non-moving, not in real time, constant lighting conditions). With static scenes, the complexity of the camera system can be reduced to a single camera on a robotic platform. This configuration simplifies the costs of construction and the photometric and geometric calibration necessary for rendering.

Levoy and Hanrahan [17] (Figure 2-4) used a single camera, on a robotic arm, positioned at specific locations in space. The primarily application of their system is to capture light fields of objects. To adequately sample the object, six surrounding light field slabs are captured. During capture, the cameras are translated on a 2D plane, but in addition, the cameras are also rotated so as to fill the field of view with the object. Because of the rotation, since the focal plane is no longer parallel to the camera plane, which leads to non-uniform sampling in UV space, images are warped into the two plane parameterization.

The goal of the Lumigraph system [10] is to construct a light field, but instead of positioning the cameras strictly on a 2D grid, a hand-held camera is used (Figure 2-4). Multiple views of the scene and calibration objects are acquired by hand, enough to cover the object. Extrinsic calibration is performed as a post-process. Finally, where there are inadequate samples in the light field, a rebinning method called pull-push is used to fill in missing data.

Pull-push reconstructs a continuous light field equation by using lower resolutions of the sampled light field to fill in gaps. Pull-push has three phases - splatting, pull, and push. Splatting uses the captured data to initially reconstruct the continuous light field. The hand held camera cannot adequately sample the scene, therefore there

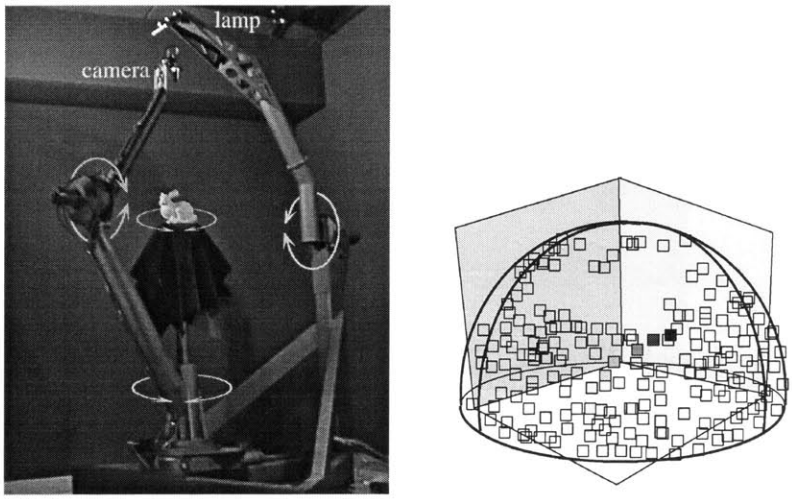


Figure 2-4: Camera systems used in [17] and [10]. [17] uses a robotic gantry whereas the Lumigraph uses a handheld camera to sample the target object.

are gaps in the reconstruction. The pull phase creates successive lower resolution approximations of the light field using the higher resolution. In the final push phase, areas in the higher resolution with missing or inadequate samples are blended with the lower resolution approximations.

Like [17], Isaksen et al. [13] used a robotic platform to capture the scene (Figure 2-5), but does not rotate the cameras.

The disadvantages of these systems are clear. By using a single camera, it takes some time to acquire a single light field (e.g., 30 minutes for [13]). Therefore, such systems can only capture static scenes.

2.4.2 Dynamic Camera Systems

Unlike static camera systems, dynamic systems capture scenes in real time using multiple cameras. The first such systems are linear arrays which can be interpreted as a three dimensional light field. I will also discuss both non-light field camera systems that use multiple cameras and finish with two-dimensional camera arrays.

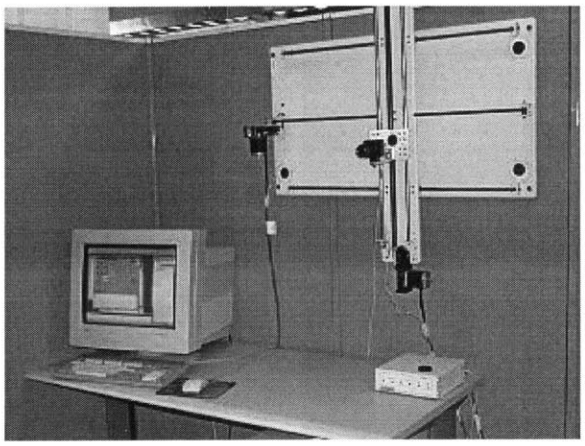


Figure 2-5: Camera gantry used by [13]. The camera is translated on a strict plane.

Linear Arrays

A linear camera array is a 3D light field system where the cameras are arranged on a curve. The parameterization is 3D because the ST plane has been reduced to a 1D line of cameras. Linear arrays have been used extensively in commercial applications, the most famous being the bullet time effect from the movie as popularized in the motion picture *The Matrix* using technologies pioneered by Dayton Taylor [34]. In this system (Figure 2-6) , cameras are positioned densely along a pre-determined camera path. Images are synchronously captured onto a single film strip for off-line processing.

Another commercial system called Eye Vision, developed by Takeo Kanade [15], has been used during sports broadcasts, most notably the Super Bowl. In this system, a small number of cameras are used to capture a single instance in time.

Yang et al. [42] used a linear video camera array (Figure 2-7) for real time video teleconferencing. Their system uses multiple CCD cameras arranged in a line with each camera capturing a video stream to its own dedicated computer. All the video streams are then transmitted to the final destination where virtual views are reconstructed.

To render a virtual a frame, [42, 43] uses a plane sweep algorithm by dynamically reparamaterizing the light field to generate views at multiple focal planes, in essence

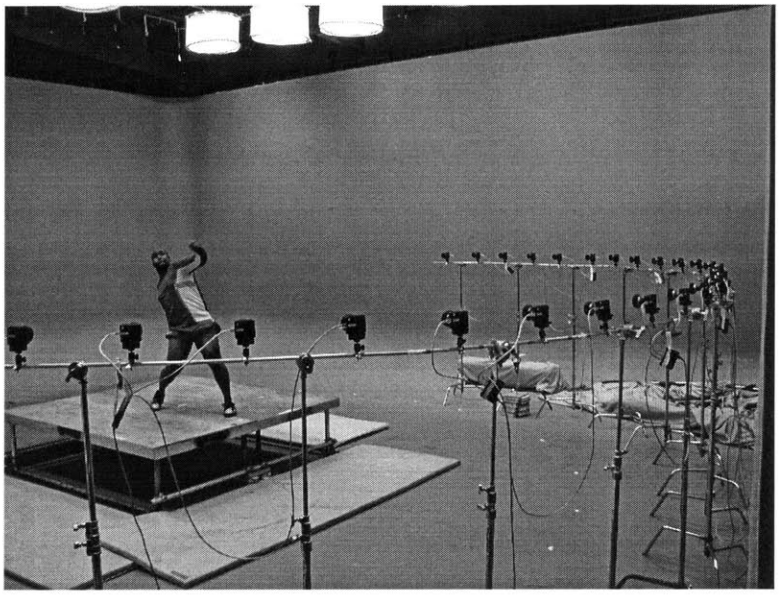


Figure 2-6: Linear camera system used for motion picture special effects.[21]

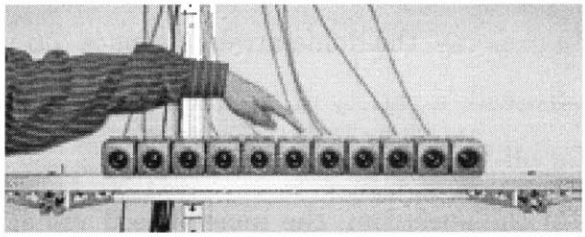


Figure 2-7: Linear camera array used in [42].

at multiple depths. The final image is assembled by deciding, on a per pixel basis, which focal plane best represents the correct scene depth based on a scoring function. Their scoring function is the smallest sum-of-squared difference between the input images and a base image (the input image closest in position to the desired view). The rendered color is the RGB mean of the input images.

Linear camera arrays are limited in their capabilities. These systems generate virtual views with a restricted range of motion and accurate parallax. Virtual camera motion is constrained to the line that the real cameras lie on. As a virtual camera leaves the camera line it requires rays that do not exist in the light field. A geometry proxy is needed even in the limit (infinite camera sampling) to move off the line. Motion parallax can still be viewed in this case, but view dependent features may be

lost.

A further drawback to both [34] and [15] is that they do not take advantage of the light field parameterization when rendering new views. Kanade does not render virtual views, but rather uses transitions to adjacent camera positions without interpolation. The primary goal of [34] is to generate a specific sequence along a specific path. They interpolate views between cameras by using both manual and computer input.

Non-Planer Systems

Separate from light field research, there has also been much interest in building multi-video camera dynamic image-based rendering systems. The Virtualized Reality system of Kanade et al. [16] has over 50 video cameras arranged in a dome shape. However, even with 50 cameras, the dome arrangement is too widely-spaced for pure light field rendering. Instead, a variety of techniques are used to construct dynamic scene geometry [29] for off-line rendering.

At the other end of the spectrum, the image-based visual hull system [19] uses only four cameras to construct an approximate geometric scene representation in real time. However, this system uses shape-from-silhouette techniques, and, thus, can only represent the foreground objects of a scene.

Two Dimensional Arrays

In recent years, researchers have begun investigating dense camera arrays for image-based rendering. The Lumi-Shelf [30] system uses six cameras in a two by three array. Cameras are oriented such that they see the complete object silhouette. During rendering, a mesh from the camera positions is projected into the image plane. From there, region-of interests (ROIs) are determined so that only a subset from each sensor is transmitted. These subsets are then projected into a final image. Most importantly, they use a stereo matching process for geometry correction, which limits the system's performance to one to two frames per second.

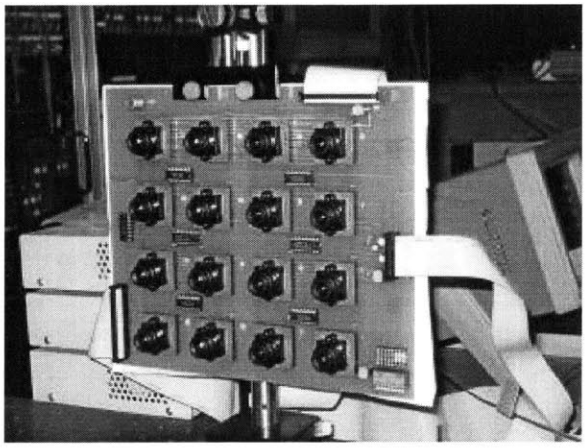


Figure 2-8: CMOS cameras developed by [25].

Oi et al. [25] designed a camera array (Figure 2-8) based on a random access camera. Their camera allows pixel addressing, but does not incorporate image warping and resampling. They demonstrate a prototype of the camera device in surveillance applications to simulate high resolution, variable field of view cameras.

Naemura et al. [22, 23] present a 16-camera array intended for interactive applications. They compensate for the small number of cameras by using real-time depth estimation hardware. They also employ a simple downsampling scheme (i.e., images are downsampled by 4 in both dimensions for a 4x4 array) to reduce the video bandwidth in the system, which keeps bandwidth low but does not scale well.

Wilburn et al. [40, 39] have developed a dense, multi-camera array using custom CMOS cameras (Figure 2-9). Each camera is designed to capture video at 30 fps and compressed to MPEG. Multiple MPEG streams are recorded to RAID hard drive arrays. The main distinction of this system is that their target application is the compression and storage of light fields, and not real-time interactivity.

Most recently, Zhang and Chen [45] introduced a sparse, multi-camera array using off-the-shelf cameras (Figure 2-10). Their system is set up with 48 cameras connected to one computer using region-of-interests, similar to [30], to reduce the transfer bandwidth. To render an image they use a plane sweep algorithm similar to [43] to evaluate pixel color and depth and a weighting similar to [4] to determine camera contribution.

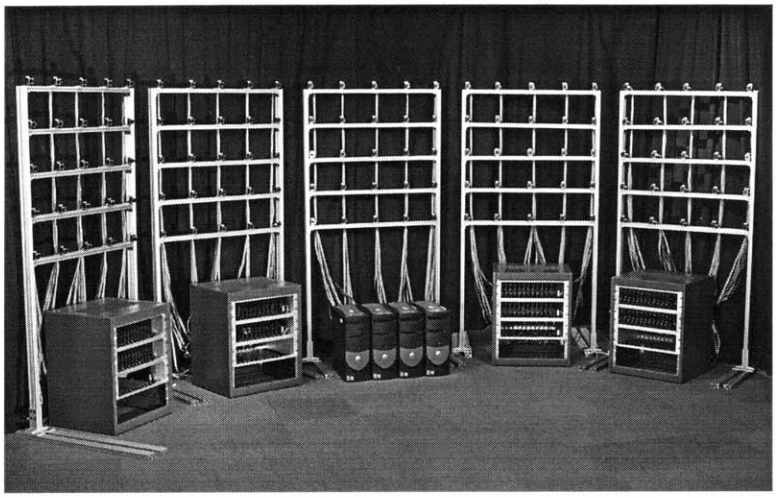


Figure 2-9: Wilburn et al. only records light field videos and processes them off-line. [40]

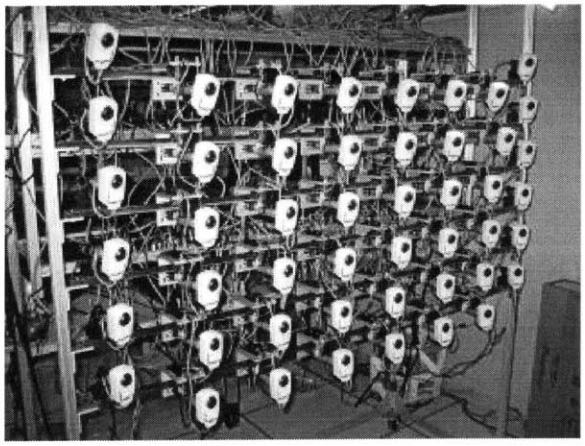


Figure 2-10: Zhang and Chen compensates for their sparse camera sampling through their reconstruction algorithm.[45]

System	Arrangement	Real-time
Dayton[34]	Linear/General	N
Eye Vision[15]	Linear/General	N
Yang[42]	Linear	Y
Light Field[17]	Rotated	N
Lumigraph[5]	General	N
DRLF[13]	Wall-Eyed	N
Kanade[16]	General	Y
Visual Hull[19]	General	Y
Lumi-Shelf[30]	Rotated	Y* (low fps)
Naemura[22, 23]	Wall-Eyed	Y* (low fps)
Oi[25]	Wall-Eyed	Y
Wilburn[40, 39]	Wall-Eyed	N* (capture)
Chang[45]	Wall-Eyed	Y

Table 2.1: The camera arrangement of each system and its rendering capability. A "general" arrangement is where there is no strict structure to the positioning. "Wall-eyed" means that the optical axis of all the cameras are parallel (generally orthogonal to the plane they lie on). A "rotated" configuration is where the camera positions may have structure, but that the optical axes are not parallel.

2.5 Summary

In this chapter I have outlined the areas of previous work. In the first half of the chapter, I covered light field rendering and aspects related to rendering images. In the second half of the chapter I discussed various camera systems. Table 2.1 summarizes the camera system in terms of camera configuration and type of rendering. In the next chapter I will discuss camera system architectures for capturing and rendering light fields for various applications.

Chapter 3

System Architectures

One of the goals of this thesis is to describe light field camera array design and construction specifically for studio applications. In this chapter, I will discuss design goals necessary for various video rate applications. Then I will describe the range of system architectures that meet the application's requirements and also how they relate to the previous work.

3.1 Design Considerations and Goals

A number of considerations affect the design of any light field camera array, including data bandwidth, processing, scalability, and desired uses for the system. Also considered are the overall system cost and camera timings when constructing a system.

3.1.1 Data Bandwidth

Light fields have notorious memory and bandwidth requirements. For example, the smallest light field example in [17] consisting of 16x16 images at 256x256 resolution had a raw size of 50 MB. This would correspond to a single frame in a light field video. In moving to a dynamic, video-based system, the data management problems are multiplied because each frame is a different light field. For every frame the system could be handling data from the cameras, hard drives, memory, etc. Thus, one of

the critical design criteria is to keep total data bandwidth (i.e., the amount of data transferred through the system at any given time) to a minimum.

3.1.2 Processing

One of the advantages of light field rendering is the fact that rendering complexity is independent of scene complexity and usually relative to the rendering resolution (a light field database lookup per pixel). However, like with data bandwidth, it is important to minimize processing time when dealing with a video or a real time system. For a given target framerate, the entire system has only so much time to process a single frame. In addition to rendering and handling the data, the system could be compressing or decompressing video streams, handling inputs, etc. Therefore, processing is not just limited to the rendering. Any improvement facilitates the inclusion of more features or increases the framerate.

3.1.3 Scalability

As shown by [5] in Chapter 2, the image quality of light field rendering improves with increasing number of images in the light field. Thus, the system should be able to accommodate a large number of video cameras for acceptable quality. Ideally, increasing the number of video cameras should not greatly increase the data bandwidth or the processing of the system.

3.1.4 Cost

One of the appeals in the static camera system is that they require only a single camera. However, a camera array multiples the cost in constructing a system by the number of cameras and therefore probably limits the number or the quality of sensor used in the system. A light field camera designer must factor in the cost of the system in deciding features and components. For example, lower quality cameras may be cheap, but they may also necessitate the need for increased computation to correct for image defects. More expensive cameras would improve image quality,

but the extra cost would reduce the total affordable number, which decreases final rendering quality.

3.1.5 Synchronous vs. Asynchronous

Ideally when recording video, all the cameras would be synchronized with each other or at least can be triggered externally. This means that all the sensors capture an image at the same instant, or can be controlled to capture at a specific time.

However, most off-the-shelf cameras, especially inexpensive ones, lack synchronization capability. Either that or the cost of a triggering feature makes the system too expensive. Therefore, the tradeoff in not having an asynchronous system is using software to correct for the difference in sensor timings which will potentially lead to errors, but hopefully small enough not to be noticed by the user.

3.1.6 Desired Uses

A dynamic light field system has many potential uses, some of which may be better suited to one design than another. The following is a range of possible uses for a multi-camera system. The distinguishing feature among the many applications is the bandwidth and storage requirements.

A. Snapshot In the last chapter I discussed single camera systems for static scenes. A snapshot is different in that although it generates a single light field (or one light field frame) it is an instance in time, therefore it can be used for both static and dynamic scenes. One application is for capturing light fields of humans or animals as in [7]. In their research they must acquire light fields of multiple human heads. A multi-camera system does not necessitate the target model to remain motionless for a prolonged period of time as in a single camera system, thus reducing errors in the light field.

B. Live Rendering in Real Time A live rendering is when novel views are generated for the user as the scene is in motion. An example of this application would be

a live television broadcast where a camera operator is controlling the motion of the virtual camera broadcasting a video stream. In this scenario, there is no recording of the entire light field, only the processing of those video fragments that contribute to the final image. Live rendering is also the primary application for the camera array prototype I describe in the next chapter.

C. Recording the Entire Light Field Video In this application, video streams from the entire camera array (the entire light field) is captured and stored for post-processing. The processing requirements in this scenario are minimal, but the transmission and storage bandwidths are high.

D. Real Time Playback of Finite Views Generally, only a single or a few video streams, such as stereo pairs for a head-mount display, are rendered from a captured light field. Processing and transfer loads would be dependent on the number of streams being rendered.

E. Real Time Playback of All Views Sometimes playback is not of a virtual view, but of the entire light field (transferring all the video files simultaneously). Having the entire light field is advantageous when information regarding the virtual views is not known. For example, a light field camera array may be used to drive a autostereoscopic viewing device (e.g., a true 3D television). An autostereoscopic display allows viewers to see 3D images without the need for special aids (ie glasses). Static versions of such devices have been demonstrated [13]. In this application, the camera must have the bandwidth to deliver the full light field to the display, whether from the cameras directly or from a storage device. An autostereoscopic device should be able to accommodate any number of users at any viewpoints, thus the need for the entire light field.

F. Post-production Processing Finally, light fields can be processed off-line in applications where real-time rendering is not required, such as in film post-production and animation.

System	Bandwidth	Processing	Scalable	Cost	Uses	Design
Dayton[34]	Low	Off-line	Y	Low	A,C,F	All
Eye Vision[15]	Low	Low	Y	Low	A	All
Yang[42]	Med to High	Medium	N	Med	B	All
Kanade[16]	High	Off-line	Y	High	B	All
Visual Hull[19]	Med	Med	Y	Med	B	All
Lumi-Shelf [30]	High	High	N	Med	B	Finite
Naemura[22, 23]	High	High	N	High	B	All
Oi[25]	Low to Med	Med	Y	High	B	Finite
Wilburn[40, 39]	High	High	Y	High	A,C,F	All
Chang[45]	Med to High	High	Y	High	B	Finite

Table 3.1: How camera systems from Chapter 2 meet design goals.

3.1.7 Comparing the Prior Work

Table 3.1 categorizes some of the relevant camera-array systems discussed in Chapter 2 with the design goals that must be met for a light field camera array.

3.2 General Camera System

Any camera array system can generally be broken down into the following components and interfaces in Figure 3-1.

1. Transfer between cameras and controller.
2. Transfer between controller and storage.
3. Transfer between controller and display.
4. Controller processing

The controller is the representation of one or many processing units that direct data traffic, control devices, compress video streams, render viewpoints, send video to the display, etc. It can perform all of these duties or only a few of them (i.e., pass data directly to an auto-stereoscopic display without processing).

In Table 3.2, I characterize the range of applications in Section 3.1.6 by the degree in which they exploit the above areas. The scoring will be a relative scaling from

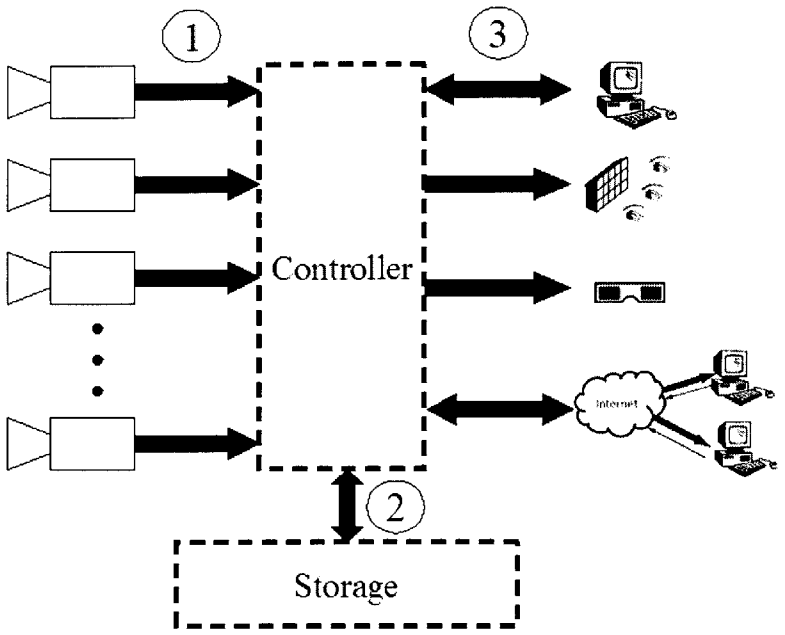


Figure 3-1: Transfer and processing points that can generally be found in any camera design.

low, medium, to high. For example, a live rendering application would have a low utilization of the link between the controller and the display whereas the bandwidth needed to transfer the entire light field to an auto-stereoscopic display would be high.

3.3 Range of Architectures

In designing the prototype system that will be presented in the next chapter, I evaluated two possible system configurations. The fundamental difference between these two designs is in the number of output views they can deliver to an end user. The first system is a straightforward design that essentially delivers all possible output views (i.e., the whole light field) to the end user at every time instant. I call this type of system an all-viewpoints system. The second system type is constrained to deliver only a small set of requested virtual views to the end user at any time instant. I call this system the finite-viewpoints system. The all-viewpoints system predictably has high bandwidth requirements and poor scalability. However, it offers the most flexibility in potential uses. The finite-viewpoints system is designed to be scalable

		Components and Connections			
		1. Transfer between cameras and controller	2. Transfer between controller and storage	3. Transfer between controller and display	4. Controller processing
Application	Snapshot	Low	Low	Low	Low to Medium
	Real time live rendering	Medium	--	Low to Medium	Medium
	Recording	High	High	--	Medium to High
	Real time playback (Finite Views)	--	Medium	Low to Medium	High
	Real time playback (All Views)	--	High	High	Medium to High
	Off-line Processing	--	--	Low	Low

Table 3.2: Relative bandwidth loads at various points in Figure 3-1.

and utilize low data bandwidth. On the other hand, it limits the potential uses of the system.

3.3.1 All-Viewpoints Design

The all-viewpoints design (Figure 3-2) is the simpler of the two architectures. In this design the camera array delivers all of the video streams to a display device. I call this the "all-viewpoints" design because any number of distinct virtual viewpoints be synthesized at the display device.

The advantage of this approach is that all of the light field data is available at the display device. This feature allows, for example, the light field to be recorded for later viewing and replay. In addition, this design could permit the light field to generate any number of virtual views simultaneously, given a suitable multi-user autostereoscopic display device. Also, by sending all the video streams, the overall processing in the system is reduced to transferring of data between components.

The primary disadvantage is that the data bandwidth required to transmit large

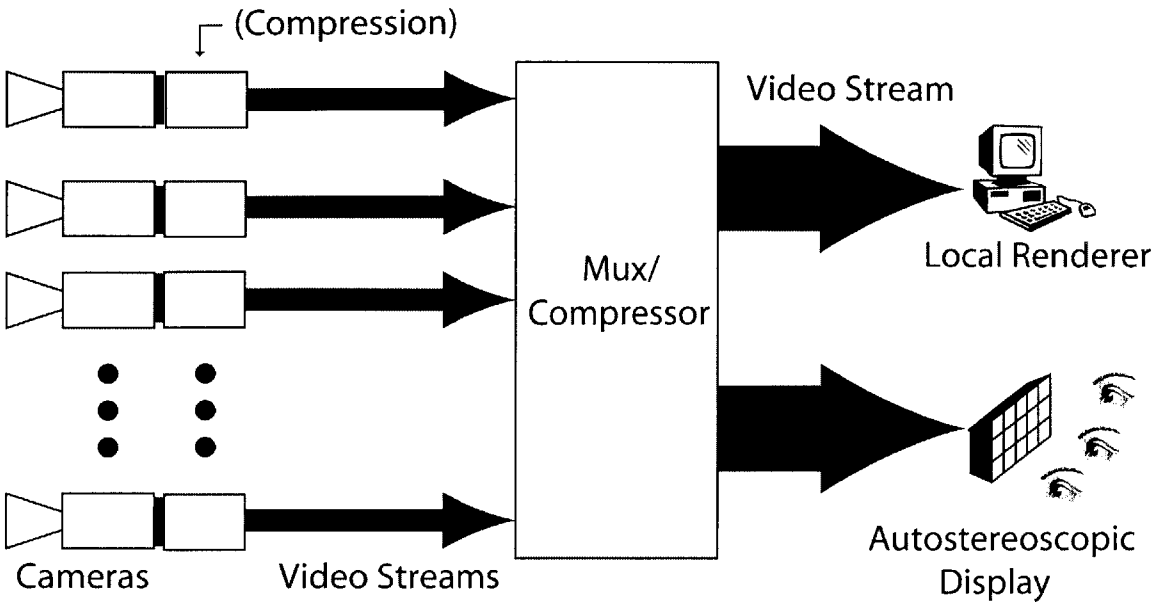


Figure 3-2: All-viewpoints design. High bandwidth requirements because the display requires the entire light field.

numbers of video streams is large. In addition, the bandwidth increases with the number of cameras, making the design difficult to scale.

Considering that a single uncompressed CCIR-601 NTSC video stream is ~165 Mbps[1], a system with multiple cameras would require aggressive compression capabilities. An optimal compression device would encode all video streams simultaneously, which could potentially lead to an asymptotic bound on the total output bandwidth as the number of cameras is increased. However, the input bandwidth to any such compression device would increase unbounded. The most practical solution is to place compression capabilities on the cameras themselves, which would reduce the bandwidth by a constant factor but not bound it, such as the design by [40].

3.3.2 Finite-Viewpoints Design

The finite viewpoint design (Figure 3-3) trades off viewing flexibility for a great reduction in bandwidth by treating cameras as "random access" video buffers whose contents are always changing. The cameras are individually addressable (such as in

[1]720x480 Luminance, 360x480 Chrominance, 4:2:2 sub-sampling, 60 fields/sec (interlaced) [38]

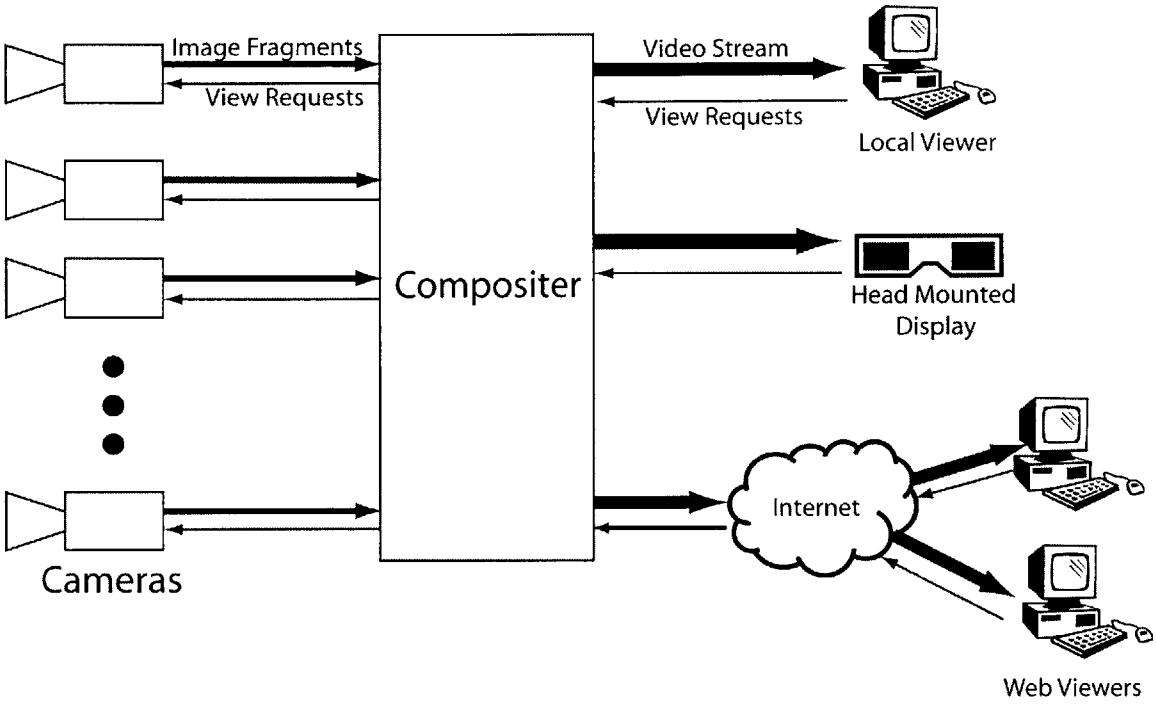


Figure 3-3: Finite-viewpoints design. Bandwidth needs are proportional to the number of distinct viewpoints requested.

[25]), and subsets of pixels can be copied from their video buffers. The key idea is that the individual contributions from each camera to the final output image can be determined independently. These "image fragments" can then be combined together later (e.g., at a compositing device or display) to arrive at the final image.

The system shown in Figure 3-3 operates as follows. An end user at a display manipulates a virtual view of the light field. This action sends a virtual view request to a central compositing engine. The compositor then sends the virtual view information to each of the cameras in the array, which reply with image fragments representing parts of the output image. The compositor combines the image fragments and sends the final image back to the user's display.

The system transfers only the data that is necessary to render a particular virtual view. In this way, the total bandwidth of the system is bounded to within a constant factor of the size of the output image, because only those pixels that contribute to the output image are transferred from the cameras. Ideally, cameras would either be

pixel-addressable in that a single pixel can be return on request or have intelligence to determine the exact contribution to a viewpoint. This is different from [30] and [45] where only a rectangular subset of the sensor's image can be returned and thus transferring extraneous data.

In general, the system may be designed to accommodate more than one virtual viewpoint, as long as the number of viewpoints is reasonably small. Note that there is some bandwidth associated with sending messages to the cameras. However, with the appropriate camera design, these messages can in fact be broadcast as a single message to all the cameras simultaneously.

Along with a reduction in bandwidth, this system design has other advantages. First, it more readily scales with additional cameras, as the total bandwidth is related to the number and size of the output streams instead of the number of input streams. However, scalability is also dependent on the rendering algorithm. This will be explained in more detail in the next section. Second, the display devices can be extremely lightweight, since only a single video stream is received. A light field camera employing this design could conceivably be manipulated over the internet in a web browser.

The main disadvantage of the finite viewpoint system is that the entire light field (all the camera images or video) is never completely transferred from the cameras. Thus, it is impossible to display all of the data at once (e.g., on a stereoscopic display).

3.4 An Ideal System

The all-viewpoints and finite-viewpoints systems are two opposite ends of a spectrum of possible designs, each with a specific application. Ideally, a complete camera array would be capable of all the desired uses outlined in Section 3.1.6. The major hurdle is overcoming physical limitations of current technologies.

- Network bandwidth is limited to 1 Gbs
- Peripheral connections, Firewire and USB, have a maximum data rates of 400

and 480 Mbs respectively

- Peripheral buses (PCI) have a maximum throughput of 133 MBs
- Today's fastest hard drive can read at a maximum sustained rate of 65 MBs

These transfer rates are also based on ideal, burst transfer situations where data is read or written sequentially. Random access transfers dramatically decreases performance (33 to 44 MBs).

The best system would incorporate the best features of the all- and finite-viewpoints designs - capable of storing all camera data, but also limit network bandwidth relative to the number of virtual views. (Figure 3-4) This system would have processing units close to the cameras (ideally with each sensor attached to its own CPU) to handle rendering and compression. These cameras would also have dedicated storage to store individual video streams and consequently overcoming physical hard drive limitations. I call a collective sensor, CPU, and storage configuration a smart camera. A smart camera would reduce the bandwidth requirements between the camera and the controller. The controller is the front end of the entire system which relays view requests between the viewing device and the smart cameras. However, the transfer of an entire light field is not possible by conventional means as outlined above due to network limitations. Therefore, a system requiring this ability such as an autostereoscopic display (such as in [20]) would need to be attached directly to the smart cameras.

Essentially, I contend that an ideal camera array distributes the rendering and storage functions and moves them closer to the cameras. In the following chapter I present a prototype system that tries to meet this description.

Scalability of this system and the finite-viewpoints design depends highly on the reconstruction algorithm. To render an image, each ray can be interpolated from a fixed number of cameras. In this case, the overall system bandwidth scales by resolution of the image and the number of virtual cameras. Another rendering method is to calculate all camera contributions to the desired view. In this case, overall system bandwidth will scale to the number of cameras. In the worst case all cameras will

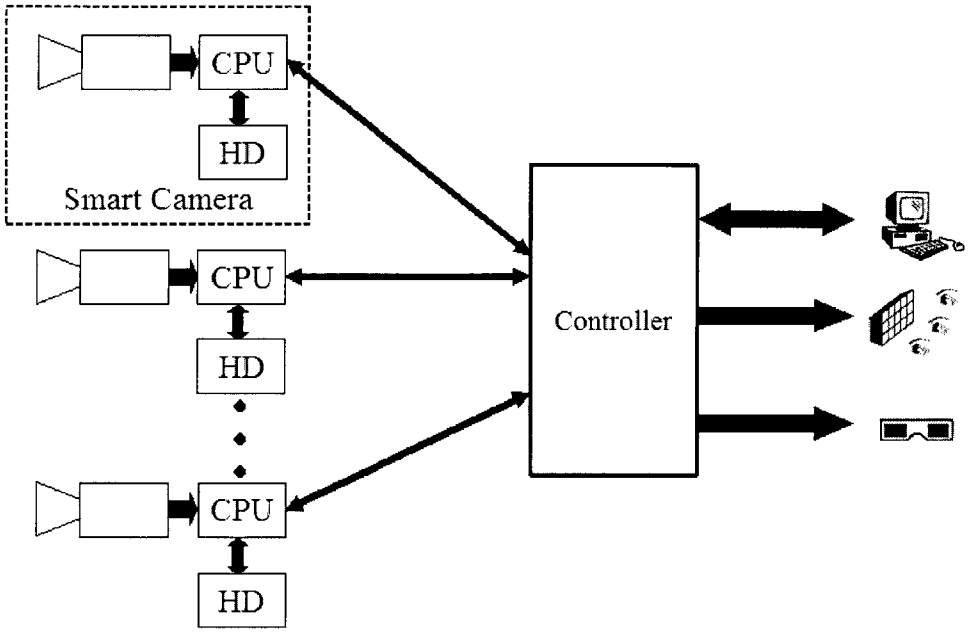


Figure 3-4: An ideal system that can render and record in real time of live scenes and also play back stored light fields in real time.

contribute to the final image. I will further discuss the issue of scalability in Chapter 4 and 5.

3.5 Summary

In this chapter I described various application types for light field cameras, the design goals needed to meet those specifications, and various architectures and how they satisfy those design goals. In the next chapter I will describe my prototype system based on the finite-viewpoints design.

Chapter 4

Prototype Camera Array

At the end of Chapter 3, I outlined the properties of an ideal camera array. In this chapter, I describe a prototype light field camera array [41] based on the finite-viewpoints design. The initial goal of this system was for live renderings in real time (rendering a limited number of virtual views in real time while the scene is in motion). While most of the discussion will revolve around this application, I will also discuss recording capabilities that were incorporated as an extension to the initial design. First, I describe the basic light field rendering algorithm that the system implements. Then I describe the two key system components, the random access cameras and the compositing engine, and how they interact. I will also discuss other details that are necessary for a complete system. Finally, I will analyze the results of the overall system.

4.1 Architecture

4.1.1 Overview

The prototype device I have developed is a real-time, distributed light field camera array. The system allows multiple viewers to independently navigate virtual cameras in a dynamically changing light field that is captured in real time. It consists of 64 commodity video cameras connected to off-the-shelf computers, which perform the

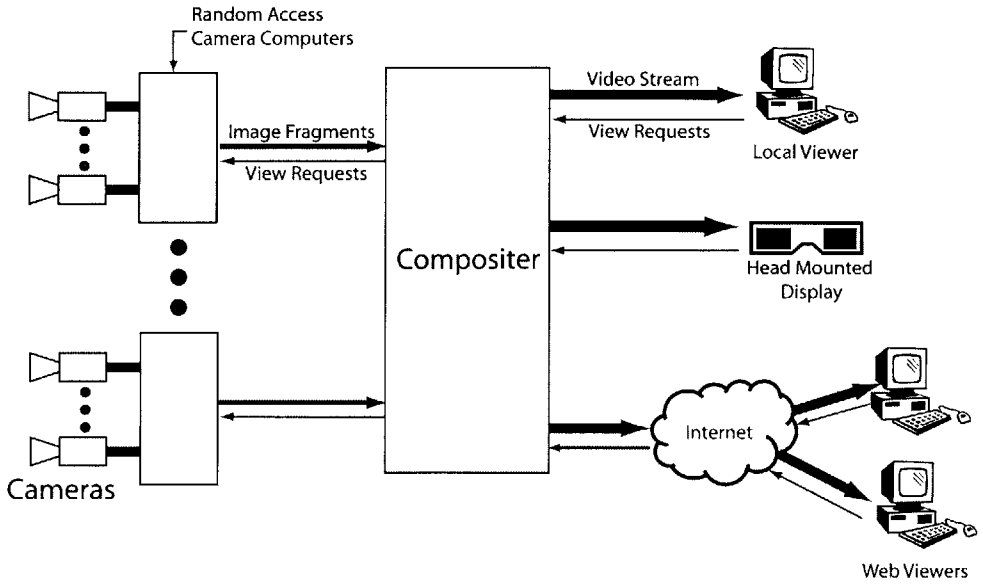


Figure 4-1: Overall System Architecture.

rendering. These computers are in turn connected to a single compositing computer that assembles final images for output to a display device. A distributed rendering algorithm overcomes the data bandwidth problems inherent to dynamic light fields. This algorithm selectively transmits only those portions of the camera videos that contribute to the desired view.

4.1.2 Rendering Algorithm

The actual implementation of my system is intimately tied to the choice of light field rendering algorithm. The algorithm that I use is related to the hardware-assisted algorithms described in [4, 31]. These algorithms are well-suited to my design since they construct the output image on a camera-by-camera basis.

First, the positions and orientations of all of the cameras are assumed to be known. While the camera positions are not required to strictly conform to a regular grid, I do assume that the cameras have a fixed regular topology. For example, in the prototype system, I have configured 64 video cameras in an eight by eight square grid. A fixed topology is used for rendering by creating a Delaunay triangulation on the grid (Figure 4-2).

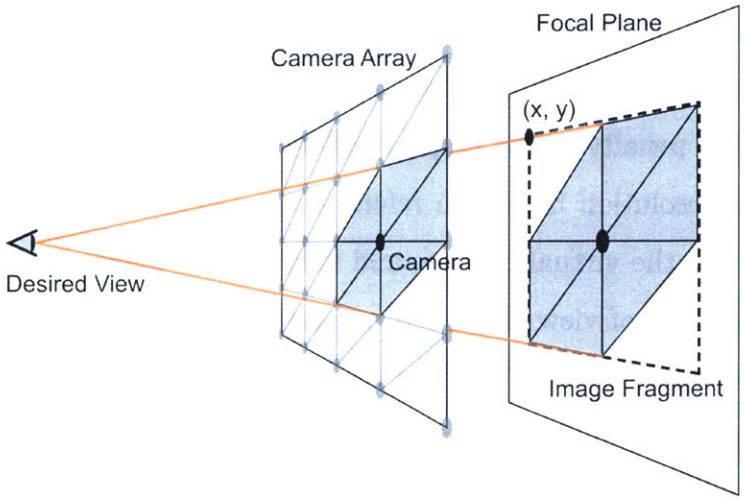


Figure 4-2: A diagram of the rendering algorithm. Cameras positions, in a regular topography, are first triangulated. All triangles associated with a camera are then projected onto the focal plane with projective texture mapping. The result is an image fragment that is transmitted for later compositing.

As in [4], the light field is rendered with respect to a user-specified focal plane. Conceptually, the rendering proceeds as follows. For each real camera (i.e., vertex) in the grid, the triangles that are connected to the camera are projected through the desired view onto the focal plane. These fixed triangles are rendered from the desired point of view using projective texture mapping of the camera's current image. The alpha values of the triangles are set such that the camera's vertex has an alpha value of one and all other vertices have alpha values of zero. These rendered triangles constitute the image fragment that this particular camera contributes to the final image. All image fragments from the contributing cameras are accumulated to arrive at the final image.

The alpha values of the rendered fragments are derived from the interpolation weights from the camera triangulation, used to combine the contributing rays from neighboring cameras. Projective texture mapping camera images to the focal plane is analogous to using the "dynamically reparameterized light field" algorithm where different rays from neighboring cameras intersect a variable focal plane.

The above rendering algorithm is based on the "unstructured lumigraph rendering" (ULR) [4] algorithm with the advantage that it does not require cameras to lie

strictly on a plane. To review, ULR weights the contribution of each camera to each desired ray based on a weighting of angular penalty, resolution mismatch, and field of view. Angular penalty is the angular difference between a contributing ray and the desired ray, resolution mismatch refers to the resolution difference between the source camera and the virtual camera, and field of view makes sure that the desired ray is within the field of view of the source cameras.

I can take advantage of the fact the there is a known, regular topology with the cameras loosely facing in the same direction. Due to the relative uniformity of the camera array, only angular penalty will affect the weighting of each ray. Angular penalty is determined based on the k-closest cameras in order to control weighting falloff from one to zero. I can optimize the rendering by using the three closest cameras and setting the weights of all other cameras to zero. This falls naturally from the triangulation and projective texture mapping. By setting the weight of the contributing camera node on the triangulation to one and zero everywhere else for each projective texture mapping pass of a camera, only three cameras will contribute to the rendering of a triangle or ray.

4.1.3 Random Access Cameras

A random access camera returns a portion of its current image in response to a request for data. The amount of "intelligence" required on the camera determines how the light field algorithm is distributed.

A simple random access camera knows nothing about itself or its neighbors and looks like a frame buffer to the compositor. In this case, the compositor knows all information about the system, including camera positions, focal plane position, etc. During rendering, the compositor determines which pixels are needed from each camera and directly reads them out of the cameras' frame buffers and performs all texture filtering, alpha blending, and compositing. This is the type of camera module described in [25].

However, a random access camera for light field applications can benefit greatly by applying image processing directly at each camera node. For example, as discussed

previously, rendering the light field involves projective texture mapping, which is a simple perspective transformation applied to a spatially coherent part of the camera's image buffer. Thus, a more intelligent random access camera would be able to warp its image fragment given a desired triangular output region and a 2D perspective transformation matrix. The compositor could then generate requests for triangular patches instead of individual pixels and perform only alpha compositing.

A random access camera could be further improved to have knowledge of its position in the world. In this case, the compositor could send just a desired 3D triangle along with the virtual view information and focal plane information. The required perspective image warp can then be derived from that information.

In my system, I assume that the camera has knowledge of its own position and the positions of its neighboring cameras. Given this information, the camera itself can determine its contribution to the final output image with just the virtual view information and the focal plane. All cameras require the same information, so it can simply be broadcasted to all the cameras at one time.

This simplified communication protocol has significant advantages. It reduces the bandwidth load between the cameras and the compositor. However, it is not clear that this choice is always best. For example, cameras that have no knowledge of their neighbors might be useful for a fault tolerant system. A sophisticated compositor could reconfigure the camera topology on-the-fly in the case of camera failures.

4.1.4 Simulating Random Access Cameras

Unfortunately, the type of random access camera that I envision is not yet commercially available. With the advent of inexpensive CMOS imagers, however, these cameras could certainly be built on a single chip.

In the meantime, I chose to simulate such cameras using a regular video camera connected to a computer equipped with commodity graphics hardware. Video frames are downloaded into the computer's memory. As view requests are received over the network, the computer warps a small part of the video frame according to the rendering algorithm outlined above and sends the resulting image fragment back to

the compositor.

The image fragment is a small rectangular patch with associated 2D coordinates (Figure 4-2). These coordinates tell the compositor where to position the image fragment in the final image. The fragments are also tagged, so when there are multiple users, the compositor can determine to which virtual view and focal plane they belong. Fragments can also be compressed, resulting in even further bandwidth reduction. My system uses simple JPEG compression for this purpose.

I have found that standard PCs are sufficient for simulating a random access camera. In fact, a single computer can easily simulate more than one camera. In my system, a single computer processes up to 16 simulated random access cameras. The computer stores the position and neighbor information and performs image processing and compression for each camera

4.1.5 Image Compositor

The image compositor is the central communication point in my system. End users connect to the compositor when they want to view novel images rendered from the light field data.

The user's display program sends a view request to the compositor whenever the virtual camera parameters require an update. The view request consists of a virtual view and focal plane. The virtual view is specified by position, orientation, and field-of-view, and the focal plane is specified by a plane equation.

The compositor broadcasts the view request information to the random access cameras. It then clears a new frame buffer and waits for the appropriate image fragments to return from the cameras. When a fragment arrives, the compositor aligns the upper-left corner of the fragment with the proper 2D coordinates returned by the camera. It then adds the fragment into the frame buffer.

When all fragments have been received, the completed output image is returned to the user's display. Because of the small amount of data involved, the entire process happens very quickly and there is very little latency. Also, the system maintains nearly constant overall throughput because light field rendering is insensitive to the

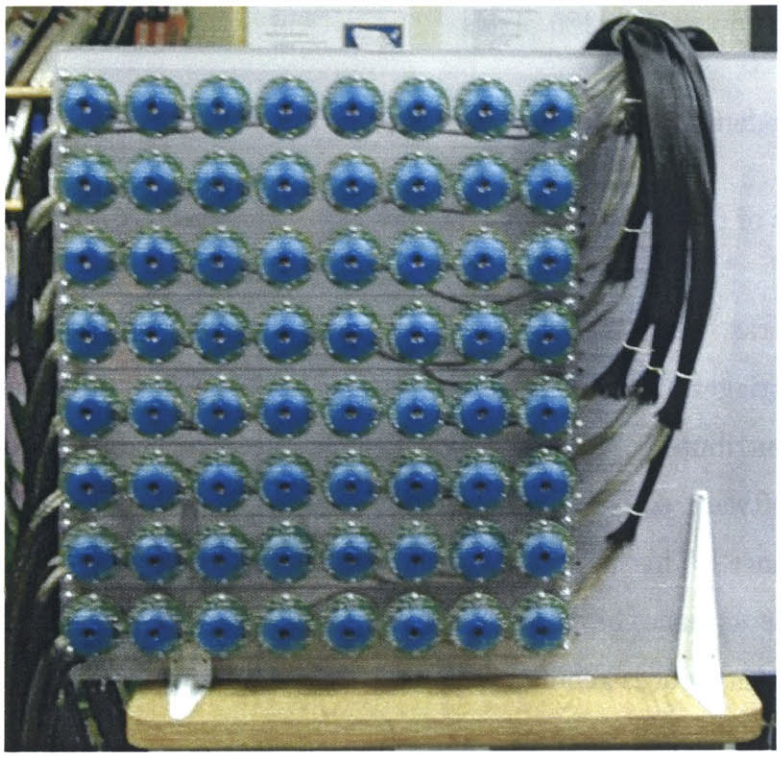


Figure 4-3: The 64-camera light field camera array. The cameras are grouped in rows of eight. One or two groups are connected to different computers.

virtual view selection or scene complexity with approximately a 3x overdraw of each pixel.

4.2 Construction

Cost was a significant consideration in choosing the hardware for the system. The camera array itself consists of 64 OrangeMicro iBot firewire video cameras (Figure 4-3) based on the Texas Instrument reference design [36, 27, 35, 26] with the following physical characteristics:

- $1/4''$ color CCD image sensor
- 62° field of view
- Video at rates of up to 30 frames per second (fps)

- Frame size up to 640x480 pixels
- Focusable lens from 1 cm to infinity
- Diameter of 60 mm

These cameras, while inexpensive, have a few disadvantages. First, they do not have external triggering capabilities and cannot be genlocked. This means that the images that contribute to the video light field are not synchronized exactly in time. Second, the software color controls on the cameras do not behave in a consistent or known manner. This makes color calibration across multiple cameras difficult. Third, the cameras lenses have significant radial distortion, which complicates camera calibration and introduces geometric distortions in renderings. I address each of these deficiencies in subsequent sections.

To build the array, I stripped the cameras of their casings and rigidly mounted them to a Plexiglas board. The cameras are organized into groups of eight, which are connected to two firewire hubs. Rows of eight cameras are then connected to PCs, which act as the random access camera simulators for those cameras. Currently, the cameras are arranged in an eight by eight grid; however, the intention is to be able to reconfigure the array by rearranging the eight-camera modules.

Currently, I use six differently configured computers to act as random access camera simulators. They range from 1.5 GHz to dual 1.7 GHz Pentium 4 PCs. Each computer has two firewire channels for grabbing video from the cameras. Each camera is identified by a unique firewire device tag. The two fastest computers each simulate 16 random access cameras, while the other four computers each simulate 8 cameras. The computers are equipped with Nvidia Quadro2 Pro graphics cards, and they are all connected to the compositing machine via a dedicated gigabit ethernet network.

The compositor computer is a slower 1.5 GHz Pentium 4 computer, and it does not require an advanced graphics card. The compositor drives a display that users can use to view the light field. This system is also allows users to connect remotely (e.g., over the web) to the compositor machine.

4.3 Calibration

In order to render images, the cameras must undergo both geometric and photometric calibration. Geometric calibration involves determining intrinsic and extrinsic parameters of each camera. The intrinsic camera properties include focal length, aspect ratios, image plane skew, optical center, and radial distortion. Extrinsic calibration refers to determining each camera's position and orientation relative to each other. Geometric calibration is required to render or select rays correctly. Photometric calibration on the other hand is required to normalize the response and color space of each camera. Since multiple cameras contribute to the final rendering, corresponding colors for each color must be reproduced equally.

4.3.1 Geometric

For quality light field rendering, it is important to know accurately the positions, the orientations, and the internal parameters of the cameras. Calibrating a single camera is a tedious process; individually calibrating 64 (or more) cameras is even more difficult. To make the process easier, I have developed a largely automatic process.

First, I manually calibrate the intrinsics of one of the cameras using Zhang's calibration software [46], which solves for focal length, aspect ratios, optical center, and radial distortion. Initially, I assume that all of the cameras have the same intrinsics as this first camera. This assumption is relaxed in a later stage of the calibration process.

Next, to determine extrinsic parameters I need to find corresponding points that are visible to all the cameras. I darken the room and move a point light source in front of the camera array. Each camera tracks the position of this light source, which is easy to see in the dark environment. I acquire about 100 points of data.

I then run standard structure-from-motion computer vision algorithms [33, 37] on this point dataset to compute an initial estimate of the cameras' positions and orientations as well as the 3D positions of the point light sources. In this step, I have

assumed identical intrinsics, which causes the structure-from-motion process to yield noisy results.

Finally, I relax the identical intrinsics assumption and refine the initial estimate using a large nonlinear optimization. This optimization improves the position, orientation, and intrinsics of each camera. (The 3D point light source positions are improved as well, but I do not use this data.) Using this process, I generally achieve a calibration with reprojection errors of less than 1 pixel, which is suitable for image-based rendering applications.

4.3.2 Photometric

When combining images from many different cameras, it is important to match the response and colors between cameras. There are many controls available in adjusting the images captured by a digital camera. I focus primarily on contrast, brightness, and white balance. Since, the sensor response of each camera is unique, independently adjusting the controls for 64 cameras is a tedious process. To assist in this task, I have implemented the simple automatic process described in [24].

The relationship between observed and displayed intensity is governed by the following equation.

$$intensity\,(x, y) = brightness + contrast * i\,(x, y) \tag{4.1}$$

Brightness is an offset added to each pixel, and contrast is the scale. A color value is composed of three channels: red, green, and blue. Brightness and contrast is applied to each channel equally. White balance is the process of adjusting the individual color channel's responses such that, simply put, "white looks white".

The color matching process begins by selecting a reference camera where the software controls have been manually adjusted so that the output image has a reasonable appearance. For simplicity, the cameras are all looking at a white background. Then, for each camera, the color parameters are adjusted in software until the total amount of red, green, and blue is close to matching the reference camera.

Unfortunately, some low-cost cameras have defective color controls (similar to the problem reported in [24]), so the automatic process does not always succeed on all cameras. First, there is only one white balance parameter to control for three parameters. Second, the brightness parameter does not equally apply an offset to each channel. Instead there is a color shift in the output images. For some cases, I must adjust some color controls manually.

Although I do not implement them, I will describe two more calibration steps that can be implemented. Adjusting the camera controls is one method of color matching. Another method is to use a color pattern and apply a per-pixel mask to the images such that the response to different colors are the same versus just white. Also, each camera sensor has inherent fixed-position noise called dark current. Again another mask is used to subtract out the noise by capturing images with the lens cap on (no light response). Since the goal of the camera array was real-time capture and rendering, I did not implement these two methods.

4.4 Synchronization

Synchronization usually refers to running the system on a single global clock such that the sequence of operations are known - capture, rendering, transmission, etc. However, the cameras cannot be genlocked. This means that images are not captured at the same instance. Furthermore, rendering is distributed and occurs asynchronously from capture. Even if the cameras of one computer are in synch, the rendering on each computer must be performed on data that was capture at the same time. Therefore in this case, synchronization refers to both the capture of the images and the rendering across the computers.

The only way to synchronize cameras is to provide an external clock. This is not possible with the cameras that I use. Firewire does not provide a mechanism either to do this. In effect each camera is running asynchronously with the firewire channel, capturing images and blindly sending them through the firewire bus. Fortunately, the cameras can be started simultaneously and through experimentation, frames arrive

at the rendering computers within a frame of each other.

As cameras are capturing images, the camera computers render the light field on a loose global clock. When the system first starts up, all the computers synchronize their internal clocks. Once they finish this step, their clocks differ by no more than 5 to 10 ms. Then, the camera computers agree on a schedule for when they will render images from the cameras. Since their clocks are aligned, they will all render an image from their respective cameras within 5-10ms of each other. Because the cameras run at only 15 fps, this scheme is sufficient to keep the light field images to within a frame of one another. See Figures 4-12 and 4-13 in Section 4.5.6 for more details.

4.5 Analysis

So far I have presented the architecture and construction of a prototype camera array system based on the finite-viewpoints design. Now I will present and analyze the results.

4.5.1 Performance

The system renders images at an average of 18 fps at 320x240 resolution while capturing video from the cameras at 15 fps. I can achieve this rendering rate because rendering and capture happen asynchronously. Unfortunately, the frame rate of the system is limited by the slowest camera computer due to the synchronization of the camera computers. For each frame, the computers agree on a schedule to render frames. Since the rendering must be performed on images that were captured at the same time, the entire process is limited to the computer with the slowest rendering speed. The fastest computer can easily run at 30 fps, so the system can attain faster speeds with minor computer upgrades.

Latency is only 80 milliseconds, which makes the system extremely responsive, but it is probably still inadequate for head-tracked display [28]. Figure 4-5 breaks down the latency of the system into its various components. The most time is spent copying video data to (Texture Update) and rendered images from (Pixel Readback)

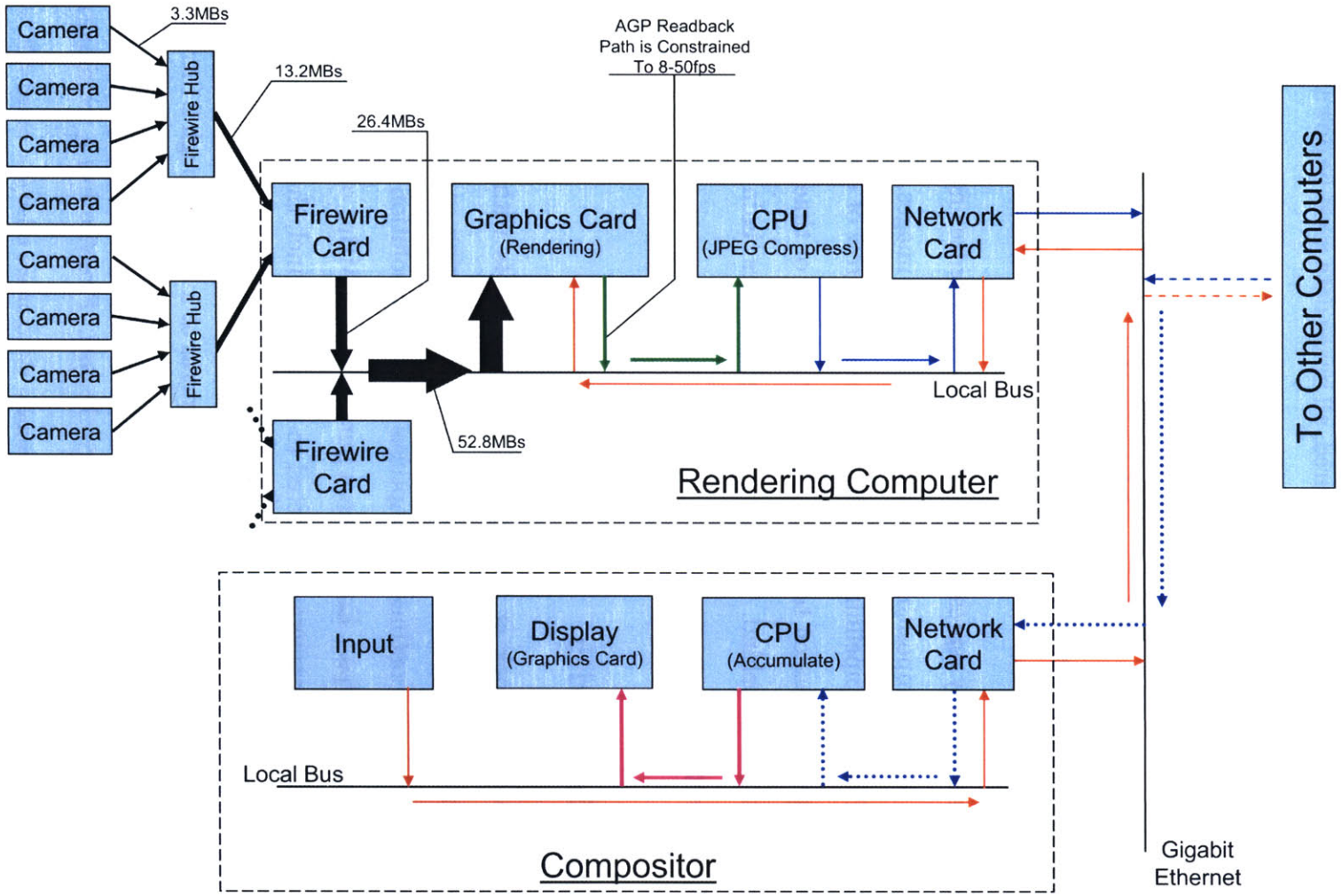


Figure 4-4: **Detailed System Diagram.** Arrow colors represent different data. Arrow sizes represent relative bandwidths.

the graphics cards over the AGP bus. This is an unfortunate artifact of simulating random access cameras; on-chip readout and rendering of the CMOS sensor would not suffer this penalty. For every frame to be rendered, all image data must be downloaded to the local memory of the graphics card. This can be improved since it is simple to conservatively determine which cameras directly contribute to the final image. However, in the worst case, all images could be used. Furthermore, pixel readback is a non-optimzed path of the graphics card because the primarily application is to render images to a display. Next generation graphics hardware should not have this problem.

The synchronization delay represents the method to synchronize ungenlocked cameras. Much time is spent compositing and displaying the final images. However, this time is essentially free with pipelining as shown by the second row of the graph. The rendering and image compression tasks take the least amount of time, with network communication not too far behind. The "setup" category covers timings that did not belong in other categories.

4.5.2 Data Bandwidth

It is instructive to briefly look at the bandwidth requirements within the system. Multiple cameras transmit their video streams to their rendering computers over a Firewire bus. Each sensor continuously captures 320x240 resolution video with 24 bits of color per pixel at 15 fps.[1] Each camera puts about 3.3 MBps of data onto the Firewire bus. So with eight cameras connected to a single channel, there is a total of 26.4 MBps of video on the shared firewire bus at all times.

Each rendering computer handles at most 16 cameras at a time through 2 firewire cards (up to 8 cameras per card[2]). Each card is an independent bus, therefore the 64 cameras are not all sharing the same firewire channel. Firewire has a maximum

[1]For the remainder of the chapter, all references to camera video unless otherwise noted will be at 320x240 pixels, 24 bit color, and 15 fps.

[2]To those interested in firewire cards, only those cards with Lucent/Agere chipset can run with 8 DMA channels meaning it can handle 8 cameras at a time. The TI chipset only has 4 DMA channels. I used the Maxtor brand Firewire Card.

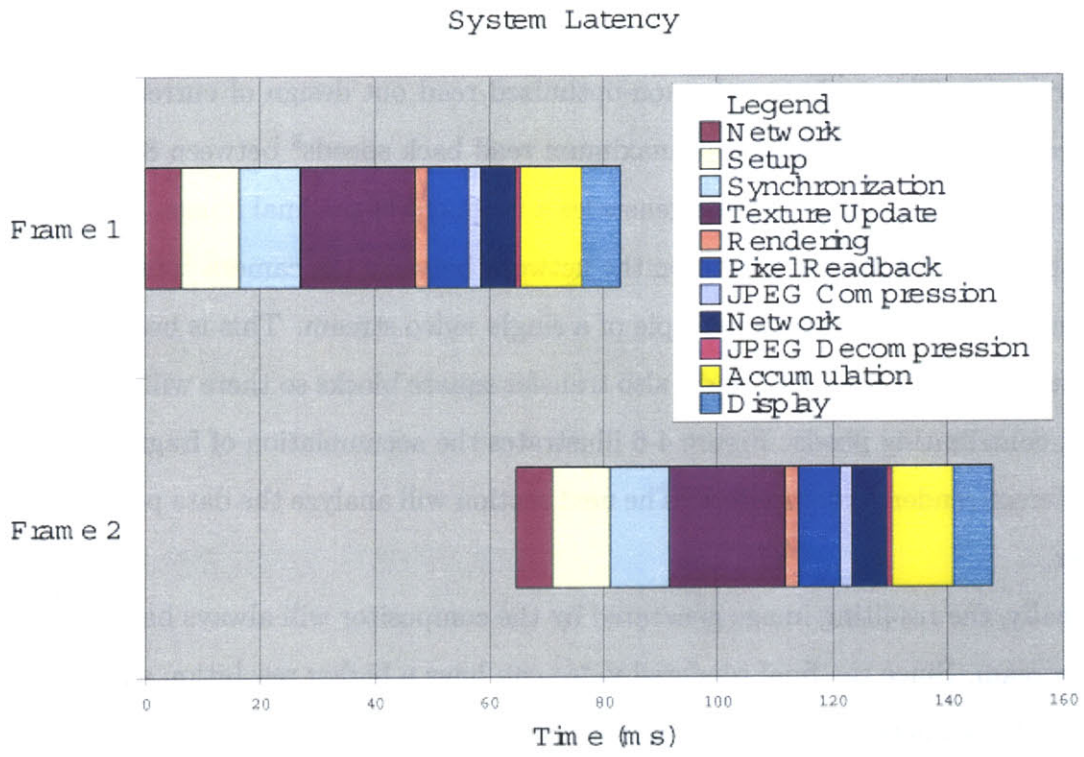


Figure 4-5: The round trip latency of the system broken down into its components. Grabbing images from the cameras happens asynchronously and occurs in the background. Therefore, the time involved in this operation does not appear in the latency graph.

transfer rate of 50 MBps. Even though this is at peak (burst) transfer speeds, since video data is only flowing in one direction, from camera to computer, my eight cameras would run in this mode and fall well under the limit.

The firewire cards, along with the graphics card, are connected to the PCI local bus of the rendering computer. PCI has a maximum transfer rate of 132 MBps. Again, the majority of the data traffic will be of the burst transfer nature, therefore moving 16 video streams to the graphics card (52.8 MBps) is acceptable.

Figure 4-5, however shows a bottleneck in the data path into and out of the graphics card. This is due to the non-optimized read out design of current graphics architectures and drivers. I found maximum read back speeds[3] between 8 to 50 fps.

Each rendering computer only generates a fragment of the final image. The idea is to limit the total data bandwidth on the network between the camera computers and the compositor to a constant multiple of a single video stream. This is because three cameras contribute to a pixel and I also transfer square blocks so there will sometimes be non contributing pixels. Figure 4-6 illustrates the accumulation of fragments from the different rendering computers. The next section will analyze the data performance further.

Finally, the resulting image generated by the compositor will always be a constant video stream. Since the final rendered video can have a higher resolution and framerate than the camera streams, the resulting bandwidth is based on the user's request. The example images were rendered at 512x512 resolution. With JPEG compression the network bandwidth has been on average less than 1 MBs.

4.5.3 Scalability

I conducted experiments to verify the scalability of the system so that as more cameras are added the overall bandwidth should stay constant. I ran the system with varying numbers of cameras and measured the data bandwidth of the image fragments entering the compositor. I took care to always use cameras spread over the whole array so that I could render the same size output image. For example, when

[3]Nvidia Geforce2 series graphics cards.

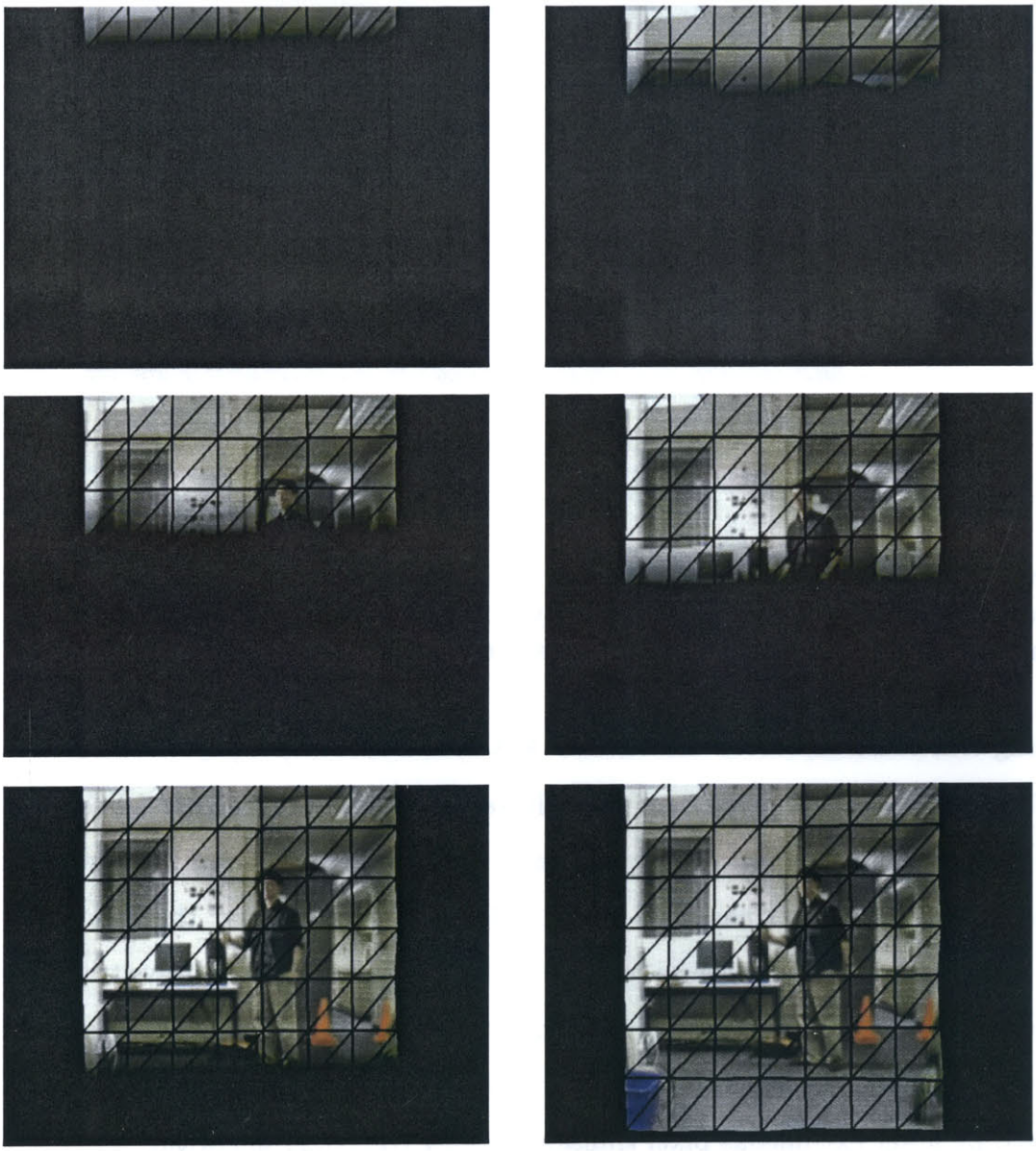


Figure 4-6: Rendering fragments and their accumulation as contributed by each rendering computer. Overlaid is the triangulation of the cameras. Notice that the cameras are not on in a strict regular grid.

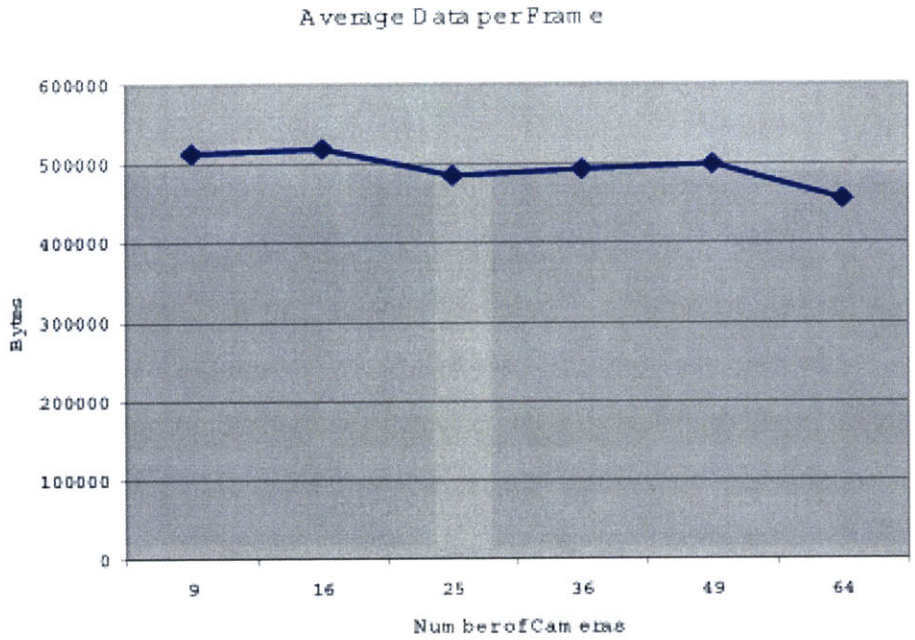


Figure 4-7: The average data per frame that is transmitted from the camera computers to the compositor. Bandwidth stays roughly constant for different numbers of cameras. These values reflect uncompressed data.

testing four cameras, I used the cameras at the corners of the array.

Using a pre-programmed camera path, results were averaged over 20 seconds of run time. Ideally, this measure should show that the bandwidth of the system does not increase as more cameras are added (Figure 4-7). Note that the bandwidth is not identical in all cases because rectangular image fragments are always transferred, which might contain extraneous pixels that are known to be zero. There is actually a reduction in the bandwidth as the number of cameras is increased. This may be caused by fewer extraneous pixel transfers at finer camera granularities.

In Chapter 3, I highlighted how scalability is dependent on the reconstruction method. In this system, rendered images are based on the contribution of each camera projected onto the focal plane. In the worst case, all the cameras could contribute to the final image. The rendering algorithm therefore scales with the number cameras so that if there were billions of cameras, then there would possibly be billions of contributions. For a rendered resolution of 640x480, each pixel would have sub-pixel

contributions from thousands of cameras.

One solution would be to use a different rendering algorithm that scales independent of the number of cameras. An example of this would be the synthetic renderer used in Chapter 7 and explained in Appendix A.

An alternative solution would be to chain multiple rendering computers and compositors. The prototype system departs from the finite-viewpoints design by adding rendering computers. Therefore, scaling is actually dependent on the number of rendering computers. A tree of rendering computers and compositors could be constructed so that performance and bandwidth utilization is then based on the number of virtual cameras.

The issue of scalability and rendering will be further discussed in Chapter 5.

4.5.4 Cost

Since I discussed Data Bandwidth and Scalability, two of the design goals for a camera array, I will also briefly mention cost. To keep the cost low I used internet cameras with limited capabilities and slower than top-of-the-line computers at the time of construction. However, the key point to note in this design is that the components are heterogeneous. By using off-the-shelf components, my system can be easily upgraded to better cameras and faster computers without any need to change software. In my experiments, I was able to only use the four 1.5 GHz machines (16 cameras each) and add in the additional 1.7 GHz machines to improve performance. Therefore, no design or software changes are needed as new technologies are introduced.

4.5.5 Rendering Quality

The following are sample renderings from the system. More examples can be found in Chapter 7. Figure 4-8 is a sample rendering taken at four different virtual camera positions. There is clear parallax between the balls, the juggler and the background.

Figure 4-9 demonstrates the variable focus capability of the system. These figures also reveal one of the system's rendering artifacts: double images are due to alias-

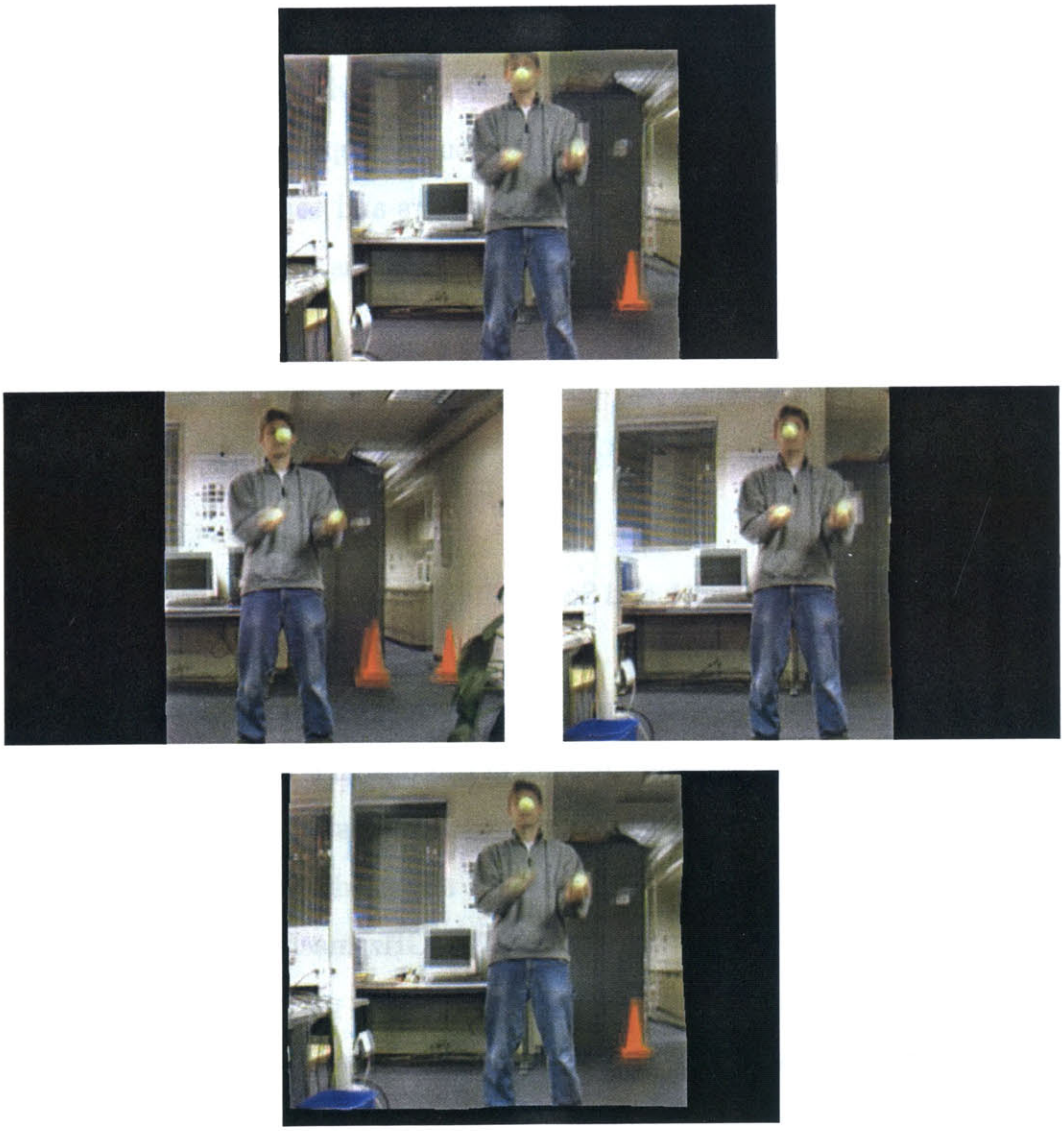


Figure 4-8: Final renderings from the compositor at four different virtual camera positions. Parallax between the balls, juggler, and background should be clear.

Figure 4-9: Final renderings at different focal planes. Artifacts are due to an undersampled light field. Image quality can be improved with more densely packed cameras or by better reconstruction algorithms.

ing. These types of artifacts are inherent to light field rendering due to inadequate sampling of the scene. From Chapter 2,

$$\Delta ST_{\max} = \frac{2\Delta v}{f\left(\frac{1}{z_{\min}} - \frac{1}{z_{\max}}\right)} \tag{4.2}$$

By assuming the maximum depth plane to be infinite, this simplifies to

$$z_{\min} = \frac{\Delta ST_{\max} * f}{2\Delta v} \tag{4.3}$$

The physical characteristics of the cameras and the camera array are 62° field of view, 320 horizontal pixel resolution, and 6 cm spacing between sensors. With $f = 1$,

$$\Delta v = \frac{2f \tan\left(\frac{\phi}{2}\right)}{resolution} \tag{4.4}$$

solves to $\Delta v = .037554$ mm. Then using Equation 4.3, the array can only generate non-aliased images with a minimum object depth of almost 800 cm (26 feet) away from the camera array.

The image quality can be improved by increasing the number of cameras and redesigning the spacing between them. For example, by using Point Grey's Firefly cameras in the extended CCD form factor, the cameras could be packed much more tightly than the current design. Additionally, the double images could be replaced with perhaps less objectionable blurring artifacts by using a larger aperture and blending together more than three images per pixel [13]. Changing the aperture size and filter changes the depth of field, thereby filtering out the aliasing artifacts not on the focal plane. This type of blending could be implemented by using a finer triangle grid with multiple blending weights at each vertex, as is done in [4]. Or, improved reconstruction algorithms could be used, such as those mentioned in Chapter 2 and others.

Figure 4-10 shows the 64 raw images taken in an instant in time. The person in these images is jumping in mid-air while the camera records him. Note that these images are unrectified, so the variations in the cameras' viewing directions are readily

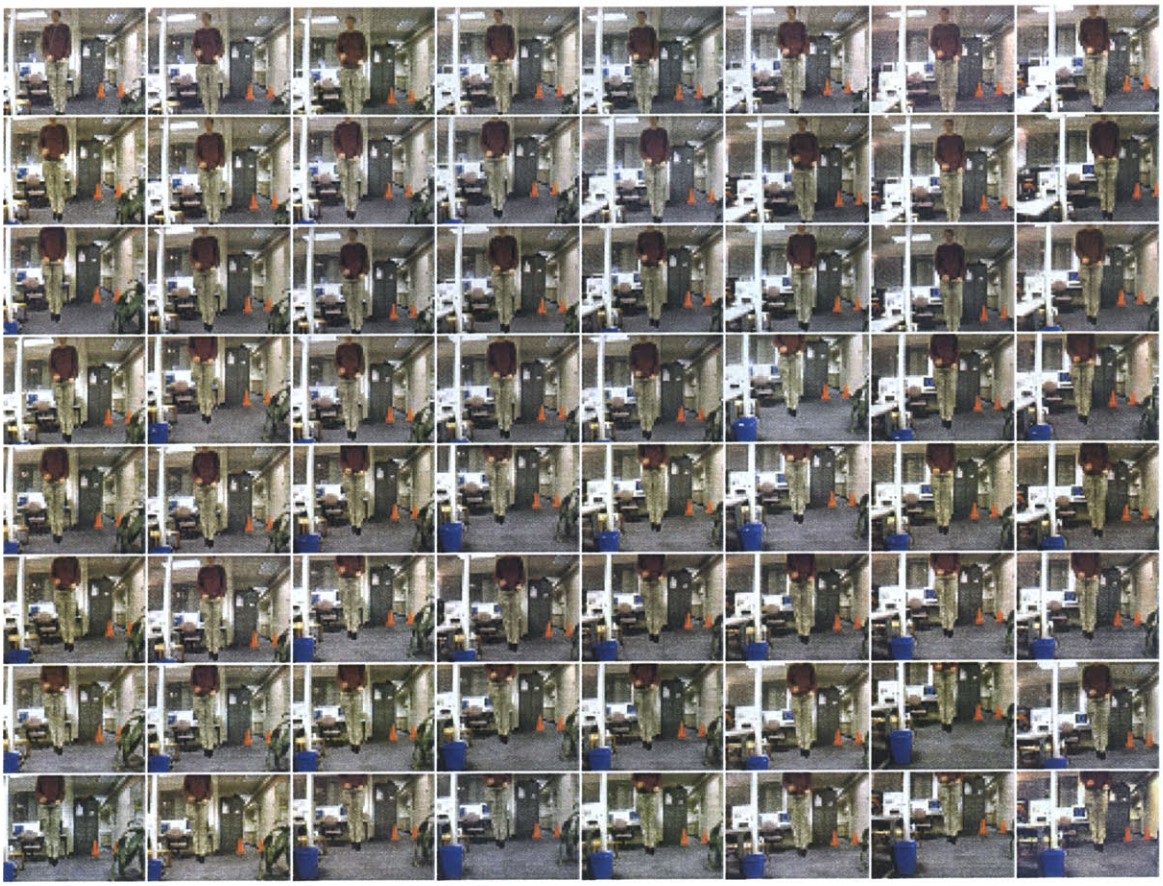


Figure 4-10: Raw data from the 64 cameras. It is clear from the images along the rows and columns that camera positions and orientations are not aligned.

apparent. Because of this there exists camera positions where some rays cannot be properly rendered form the light field such as in Figure 4-11.

Also evident from the images in Figure 4-10, is the radial distortion (most clear in the pole) inherent to the cameras. During the calibration process, radial distortion parameters are calculated, therefore it is possible to undistort the images. However, this would involve simultaneously processing multiple dynamic images ($16 * 30$ fps for each camera computer) and decrease the overall frame rate of the system.

4.5.6 Deviation from Ideal

At the end of Chapter 3, I described a potential ideal architecture that allows for both real-time viewing, but also recording and playback capabilities. My design

Figure 4-11: Rendering errors due to the alignment of the physical cameras.

comes close to that ideal by moving the "rendering intelligence" closer to the cameras thereby reducing the network bandwidth. In this section I will discuss my experiments with recording and playback.

The biggest hurdle to overcome in recording and playback is hard drive read and write throughput. In experiments on my hard drives , the average read rates is 20 MBps and the average write rate is 15 MBps. Therefore, for my current design the average commodity hard drive cannot even sustain the 55 MBps and 28 MBps required to save the raw data of 16 or 8 cameras respectively.

Wilburn et al. [40] solves this problem by externally compressing the video using special hardware into MPEG (8:1) streams and by employing a striped disk array (32 cameras per array). Without dedicated hardware, my solution is to use JPEG compression, performed by the CPU, as the frames arrive at the computer and store them into memory. Then as a post process these frames are written to disk. JPEG compresses each 230.4 KB frame to 10 KB for a ratio of 22:1. With 16 video streams, a system with 1 GB of ram can hold 6 minutes of video. However, the problem now becomes one of CPU speed vs. hard drive speed.

Figures 4-12 and 4-13 are charts that show when frames arrive at the CPU and the difference in the times. The time stamp indicates when a frame arrives in relation to the start of the cameras. In this example I capture 15 seconds of video into memory.

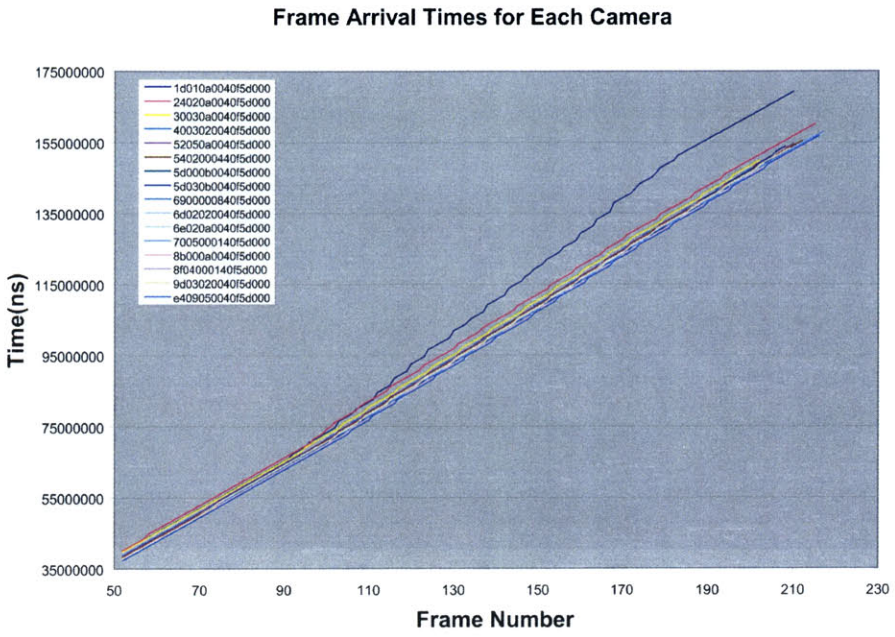


Figure 4-12: Arrival times of video frames from the cameras to the CPU measured in nanoseconds.

For the first three second (45 frames), all of the cameras drop some frames due to application and camera startup. Then the system reaches a steady state where it continuously captures and compresses frames into memory.

Except for one outlier, the cameras are generally in sync with each other. Overall, frames arrive within 18.7 ms of each other which is well within one frame time (66.6 ms).

Finally, it is obvious that the prototype system cannot render images directly from the hard drive without more hardware investment. The current simulated setup requires data for 16 cameras to be read off disk into local memory, decompressed, moved into video card memory, read back out of video memory, and transferred onto the network. For an ideal camera, which is essentially one computer per camera, this is not unreasonable since a single video stream using native video hardware compression (DXT1 6:1) is 0.576 MBps (4.6 MBps at NTSC).

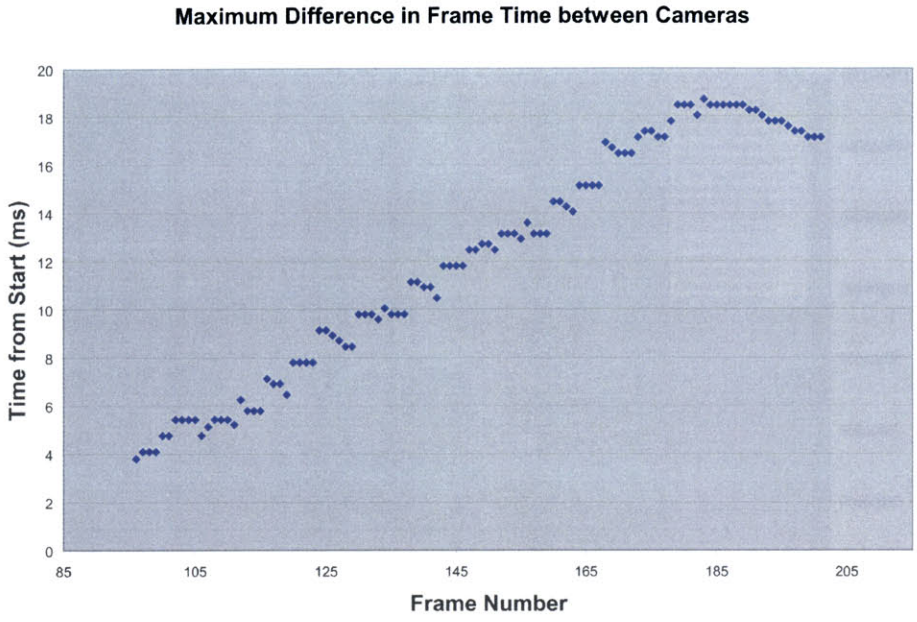


Figure 4-13: Maximum difference in arrival time for each frame number. Maximum difference is 18.7 ms which is within one frame time (66.6 ms).

4.6 Summary

Because it is simply a query into a ray database, light field rendering is essentially a large data management problem. The main conclusion to draw from this chapter is that the best way to render light fields in real time whether it is from live or stored data is to use a distributed algorithm. As technology advances a distributed system, such as the prototype described here, can easily handle and take advantage of device changes to improve performance and rendering quality. For example, the video cards I used are not pixel programmable (shaders). A programmable GPU could be used to perform improved reconstruction algorithms or correct for radial distortion and color irregularities at little performance cost. The next couple of chapters will look into light field characteristics and capabilities, some of which my aid in taming the enormity of light fields.

Chapter 5

Virtual Camera Capabilities

In this chapter I will cover two topics that have not been extensively discussed in the existing light field literature: range of motion and effective rendering resolution. Range of motion refers to the range of possible virtual camera positions and orientations. Camera plane sampling in order to build an adequate light field is discussed in [5]. Effective resolution refers to the sampling of existing light fields in terms of the resolution of the image plane.

I begin the analysis by introducing epipolar-plane images as a dual space for analyzing rays. Then I will describe the range of possible motions and orientations for 2D cameras (i.e. a "Flatland"[1] world) and later extend that into 3D. Finally, using the same 2D camera configuration I will analyze the sampling of light fields as it relates to the effective resolution of rendered images.

5.1 Introducing a Dual Space

The use of dual spaces for analyzing light fields and lumigraphs was introduced in [17, 10] and they have been to analyze light fields [11]. However, it is instructive to give an overview as I will often refer to a dual space in the later discussion. Levoy and Hanrahan [17] use Line Space (Hough transform) to analyze their light fields. I will be using epipolar plane images (EPI). Before discussing EPIs, I will first describe the parameterization I use in my analysis.

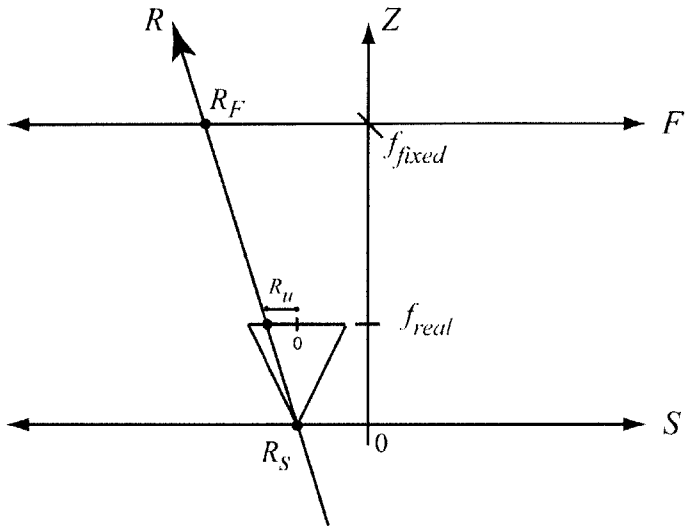


Figure 5-1: The relationship between *fixed* and *relative* parameterization. A ray R will intersect the two fixed planes S and F at (R_s, R_F). The ray will also intersect the image plane at R_u relative to the camera at R_s. f_{fixed} is the distance between the two planes and f_{real} is the focal length of the camera.

5.1.1 Parameterization

As discussed in Chapter 2, light field coordinates are based on a two plane parameterization, which I will call the *fixed parameterization* because the two planes are fixed in space. The 4D coordinates are (s,t,u,v) where (s,t) is on the camera plane and (u,v) is on the image plane.

Another parameterization method is what I will call the *relative parameterization* where there is no common image plane. Instead, (u,v) image-plane coordinates are relative to the camera at (s,t). See Figure 5-1.

The mapping between the fixed and relative parameterization is governed by the following relationship:

$$R_F = R_s + R_u * \frac{f_{fixed}}{f_{real}} \tag{5.1}$$

For example, if the plane F is placed at the same distance as the focal length of the cameras, then the ray would intersect the image plane of camera R_s at a relative coordinate of R_u. The same ray would also intersect the F plane at $R_s + R_u$.

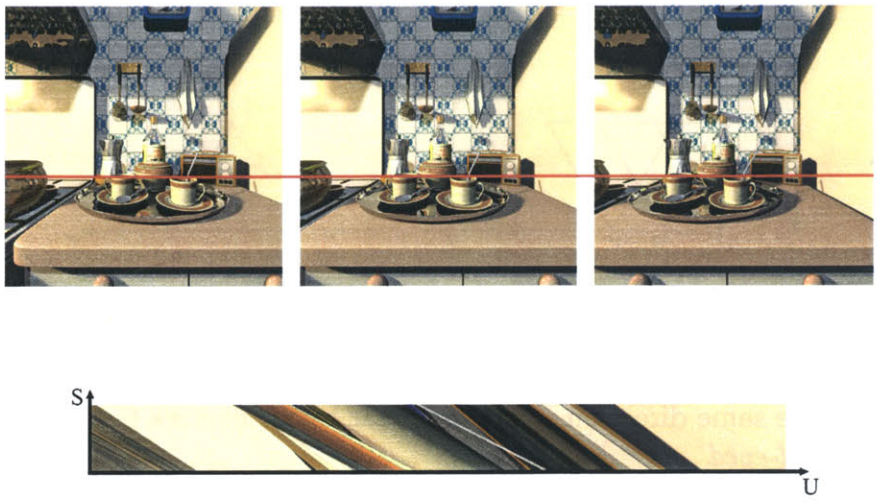


Figure 5-2: Constructing an Epipolar Plane Image (EPI) from a series of photographs from a line of cameras. A line of pixels from each photograph is stacked on top of each other. Pixels representing objects further away from the camera have a steeper slope than those closer to the camera. EPIs can also be constructed directly from the depth of the objects if known.

5.1.2 Epipolar Plane Images

Epipolar plane images (EPIs) are slices of a 3D ray space comprised of images taken from a line of cameras [3]. EPIs capture a particular 2D slice of an observed 3D scene and are equivalent to particular 2D slices through a 4D light field. EPIs can be constructed directly from the scene geometry or from images, such as from a camera array or from ray tracing. In my analysis, I will be using EPIs from images. An additional feature of using images is that they also characterize the sampling of the scene as it relates to a discreetly sampled light field.

Figure 5-2 gives an example of an EPI by taking corresponding horizontal lines from a row of images of a ray traced light field (varying s and u while keeping t and v constant). The S-axis is the camera plane and the U-axis is the focal or image plane.

To illustrate the relationships between "real" and EPI space, I will use the 2D case of a light field where the cameras are modeled as 2D pinhole cameras and arranged uniformly on a line. Real space is the coordinate system relative to the physical camera array. All rays from the cameras lie on a common plane. (Figure 5-3)

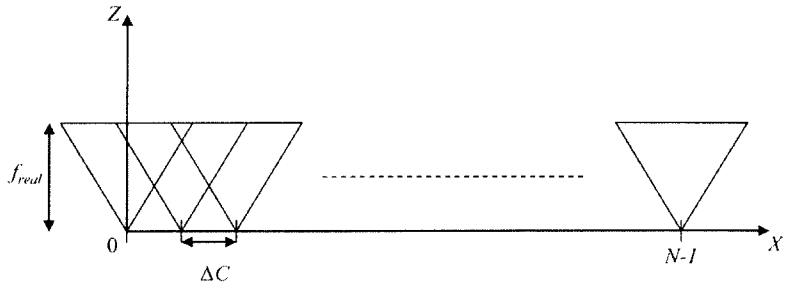


Figure 5-3: Cameras are arranged such that they lie on a strict plane or line and are oriented in the same direction - perpendicular to the camera plane. This is often referred to as *wall-eyed.*

5.1.3 Points Map to Lines

Given a point (P_x, P_z) in the 2D space (Figure 5-4), then by similar triangles that point is projected onto the image plane of camera n at

$$U(n) = \frac{f_{real} * (P_x - C_x(n))}{P_z} + f_{real} * \tan\left(\frac{\phi}{2}\right) \tag{5.2}$$

where $C_x(n)$ is the x-coordinate of the camera n (C_z is always zero). This projected image plane position corresponds to the U parameter of the 2D, relative parameterization.

The line formed on the EPI representing the point is

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} U(n) \\ C_x(n) \end{bmatrix} + \alpha * \begin{bmatrix} \Delta U = -\frac{\Delta C * f_{real}}{P_z} \\ \Delta C \end{bmatrix} \tag{5.3}$$

The slope is

$$m = -\frac{P_z}{f_{real}} \tag{5.4}$$

and is just $-P_z$ if $f_{real} = 1$. This means that a point in front of the camera plane is a line on the EPI with a negative slope, and a point in behind of the camera plane is a line with a positive slope.

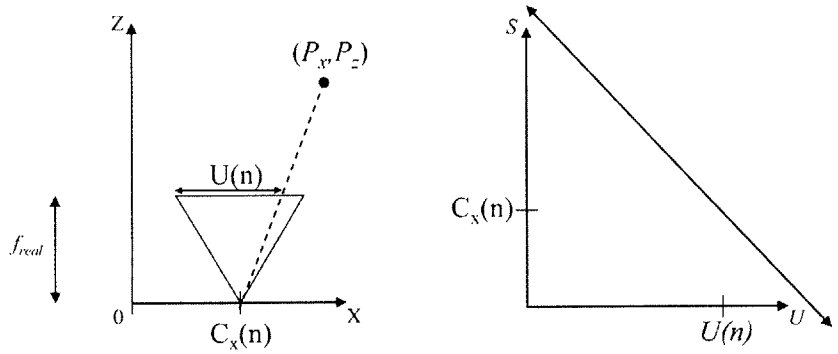


Figure 5-4: Points in real space map to lines in EPI space. The slope of the line is determined by the depth of the point from the camera plane. Points further away have a steeper slope than points closer to the camera plane.

5.1.4 Lines Map to Points

Now given a line in real space

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} P_x \\ P_z \end{bmatrix} + \beta \begin{bmatrix} d_x \\ d_z \end{bmatrix} \tag{5.5}$$

This line maps to a point (u, s) in the EPI where s is the intersection with the camera line $z = 0$ and u is the intersection with the image plane of the camera at s. From (5.5) and setting $z = 0$

$$s \Rightarrow x = P_x - P_z * \frac{d_x}{d_z} \tag{5.6}$$

From Equation (5.2) and substituting s for $C_x(n)$ using (5.6)

$$u = \frac{f_{real} * d_x}{d_z} + f_{real} * \tan\left(\frac{\phi}{2}\right) \tag{5.7}$$

In summary, a line (5.1.5) maps to a point on the EPI at

$$\left(f_{real} * \frac{d_x}{d_z} + f_{real} * \tan\left(\frac{\phi}{2}\right), P_x - P_z * \frac{d_x}{d_z}\right)$$

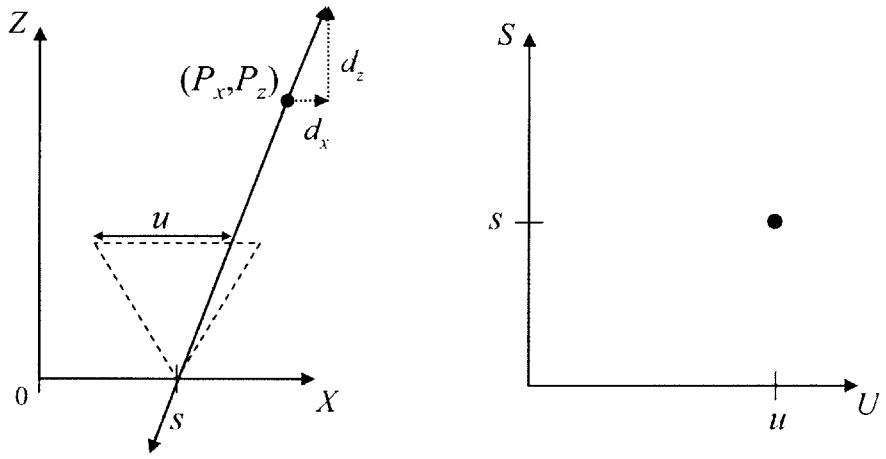


Figure 5-5: The image of a point in real space maps to a point in the EPI. EPI coordinates are the intersections with the camera line S and the image plane U of the camera at s.

5.1.5 Mapping Segments

When a line segment is parallel to the camera plane (Figure 5-6) then the EPI representation is a region bounded by two parallel lines representing the endpoints of the segment. All other types of segments map to two wedges formed by the two lines in the EPI representing the endpoints of that segment (Figure 5-7). The point of intersection in EPI space represents the line of sight from which the segment would project as a point. The lines in EPI space that pass through the intersection point between the "endpoint lines" represent all the points on the segment.

5.1.6 Relating Virtual Cameras to EPIs

A virtual camera is defined by a position (P_x, P_z) in space, a field of view (ϕ), and an image plane. Equivalently, the virtual camera position is a point, the image plane is a segment, and the field of view is defined by two lines that intersect the position and the endpoints of the segment. Therefore, the light field or EPI samples needed to render a virtual camera lie on the intersection of the line in the EPI representing the camera position and the space representing the image plane (Figure 5-8).

Since the slopes of the lines in the EPI are independent of P_x, then any movement

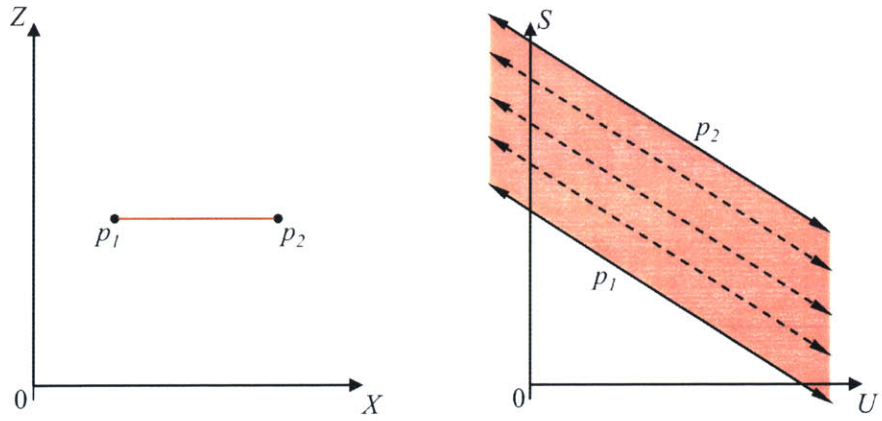


Figure 5-6: The image of a line segment parallel to the camera plane. The EPI representation is a space bounded by two parallel lines representing the endpoints. Points on the segment are lines in the EPI parallel and lying in between the endpoint liens.

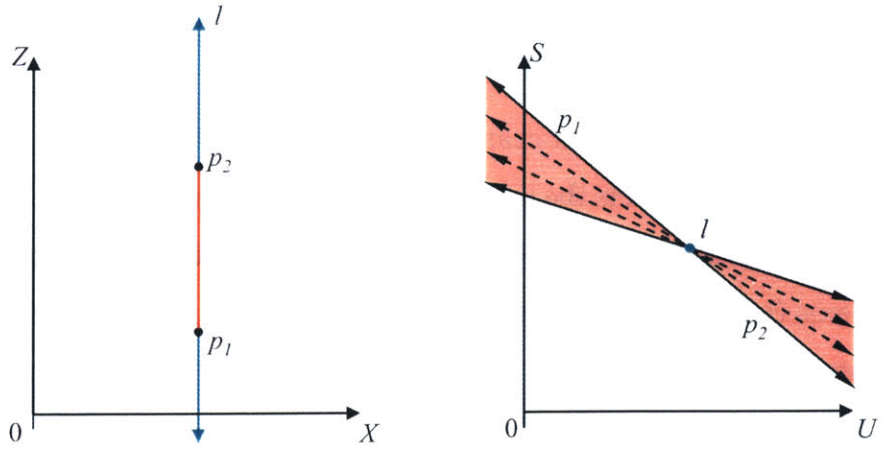


Figure 5-7: Any segment not parallel to the camera plane will map to two wedges. The line on which the segment lies will be the coincident point l. Lines representing the endpoints bound the wedges. Points on the segement map to lines that intersect l.

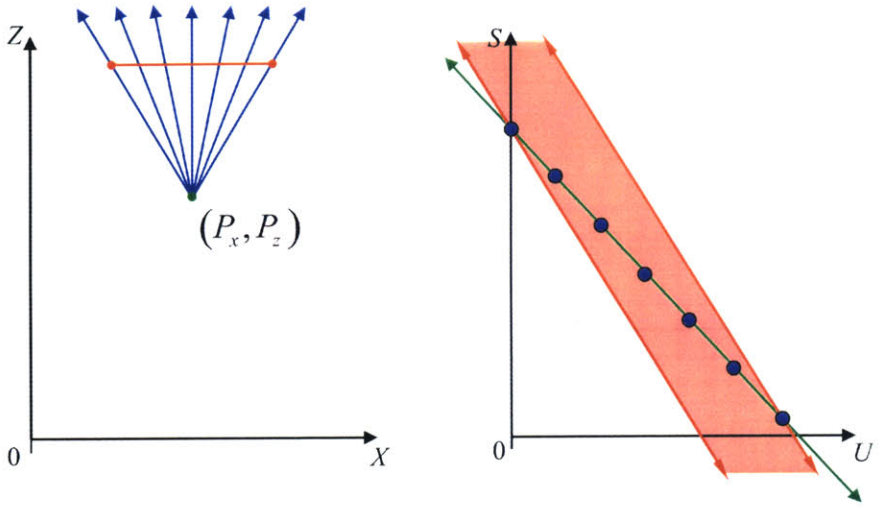


Figure 5-8: Virtual camera representation in the EPI Space. The camera position is a line in EPI space. Rays that sample the camera plane are represented by points on the camera line in the EPI bounded by lines representing the endpoints of the image plane.

of the virtual camera parallel to the camera plane maintains the structure of the EPI. The only difference is a shift parallel to the S-axis. For example, if the virtual camera moves left, then the EPI structures move down because the samples will sample the same U coordinates, just at different cameras (Figure 5-9.

If the virtual camera's position changes in depth, then the slopes of the lines in the EPI change. Equation 5.4 is the relationship between slope and depth. The effect is a rotation of the EPI structures. Another characteristic of translations in general is that the u-axis (horizontal) distance between the first and last samples in the EPI remains the same.

Finally, during a rotation of the virtual camera, the camera position is fixed, so the sampling line on the EPI stays the same. In the example figures, the image plane begins parallel to the camera plane, so during a rotation the corresponding EPI structure changes from a space bounded by parallel lines to two wedges. In the case of Figure 5-10, because the image plane is no longer parallel to the camera plane, there will also be non-uniform sampling of the light field as shown in the EPI.

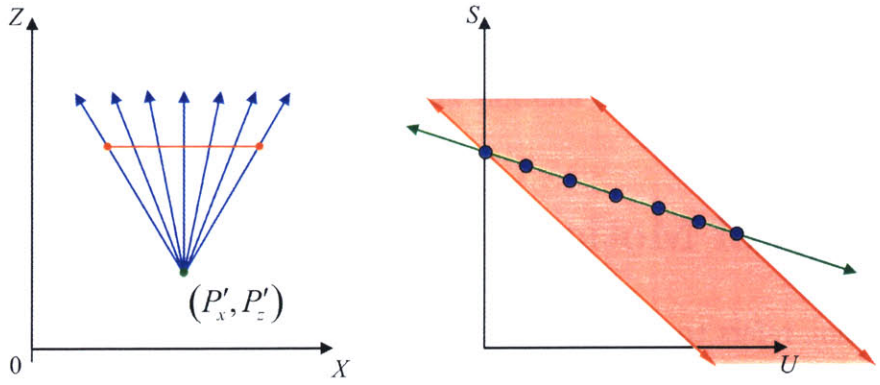


Figure 5-9: Continuing from Figure 5-8, changing the depth of the virtual camera rotates the corresponding lines in the EPI.

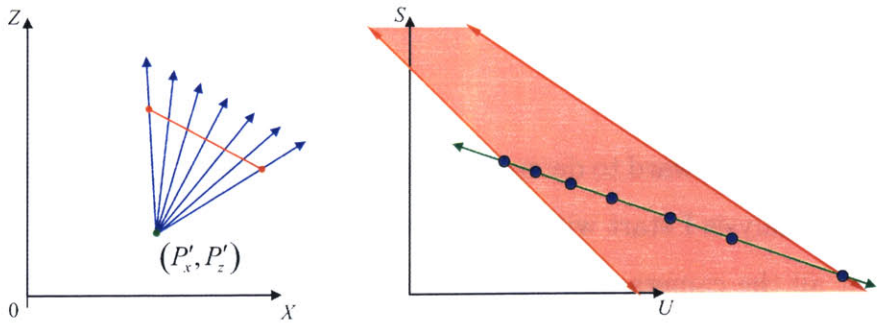


Figure 5-10: Rotation of the virtual camera. Notice that uniform sampling of the image plane will result in non-uniform sampling of the EPI.

5.1.7 Relating 3D Cameras to EPIs

The relationships outlined above easily extends to higher dimensions. Deriving the relationships is beyond the scope of this introduction (and can be found in [11]) except to note that sampling depends on the characteristics of the virtual camera. A 3D camera can be thought of as multiple 2D cameras sampling different EPIs.

5.2 Range of Motion

Range of motion refers to the possible positions and orientations of virtual cameras in real space. Given an array of real cameras, the virtual camera is restricted in movement to a region in space such that that the rays required to render that view can be found in the light field. The virtual camera is further restricted in the range of rotations that it can perform. In this analysis, I will start with positioning of the virtual camera in the 2D case from Figure 5-3 where the real cameras are two-dimensional and are positioned in a linear fashion on a plane. The results can easily be extended to a 4D light field configurations. Afterwards I will discuss the ability to rotate the virtual camera.

5.2.1 Position

2D Cameras

It is relatively straightforward to determine the range of possible positions for a virtual camera. In this analysis I start with the same configuration as in Section 5.1 (Figure 5-3) such that all the camera characteristics are the same, they lie strictly in a line, and they are facing the same direction (orthogonal to the line they are positioned on). This configuration can also be thought of as a 2D light field where one scanline is used from a row of cameras thus sampling S and U. The EPI for this scene naturally follows as each horizontal line of samples represents a separate real camera. For this analysis I also restrict the virtual camera so that its orientation is the same as the real cameras and cannot rotate. Although this simplified setup is never fully realized

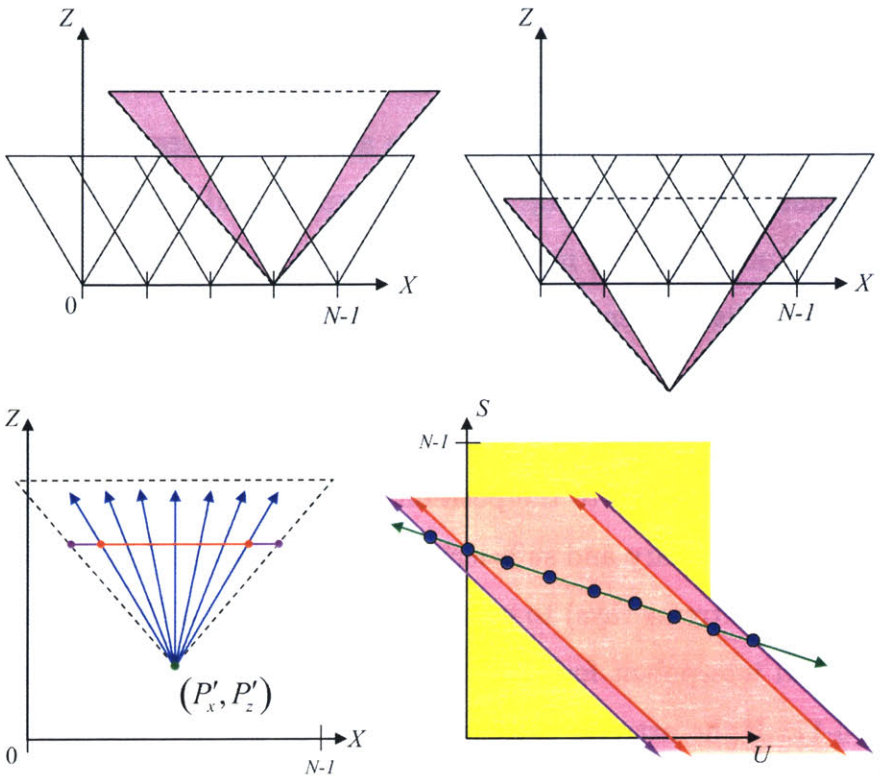


Figure 5-11: *(above)* Maximum field of view of the virtual camera is restricted to be that of the real cameras. It should be clear that all the rays in purple are at angles outside of the light field. *(below)* Using the example from Figure 5-9, as the field of view increases, the virtual camera will sample outside of the existing camera samples represented by the yellow box (this is the stacked camera images).

in reality, it gives a clear picture of the relationship among the various light field parameters and can be easily extended into higher dimensional light fields.

First, I will show that *the field of view of the virtual camera cannot be greater than that of the real or sampling cameras.* This conclusion is clear from Figure 5-11. If the virtual camera is placed at the same position as a real camera, then ray angles greater than those found in the light field do not exist. Again, I assume that a valid rendering or virtual view is where every desired ray can be found in the light field. Therefore the virtual camera in Figure 5-11 would not be a valid rendering by the above definition. The field of view restriction also holds as the virtual camera moves on or off the camera line. This is because all rays oriented at an angle larger than the real cameras do not exist in the light field.

For now, I assume that the field of view of the virtual camera is the same as the real camera. I will relax this assumption later in this section and demonstrate the effect to the range of motion. I also make no assumptions regarding the sampling of the camera line. The relationship between sampling and valid views depends on the reconstruction method. For now I can assume camera or S-line sampling to be infinite or at least satisfies plenoptic sampling. Summarizing: I solve for the range of possible camera positions for a virtual camera that has the same field of view as the real cameras configured as wall-eyed along a line in 2D space.

The range of motion refers to the positions in space where valid virtual cameras can be rendered. A necessary and sufficient condition is for the first and last rays (or what I also call the extreme rays) to exist in the light field. If the first and last rays exist, then a valid interpolant for all of the rays will exist. This is should be clear from Figure 5-12. If the extreme rays, represented by the first and last samples on the line in the EPI, fall in the region of existing samples, then all other rays of the virtual camera must also lie in the same region.

The space can be divided into three regions: on the camera line, in front of the camera line, and behind the camera line. When the virtual camera is on the camera line then the range of motion is clearly bounded by the first and last real cameras. The virtual camera can be positioned anywhere between those two points with images rendered by interpolating the rays between real cameras.

As I move the virtual camera off the camera line, images are rendered by interpolating rays among more cameras (Figure 5-12c). Then, by using the rule that a valid image is one where all rays are found in the light field, the range of motion of a virtual camera is bounded by the limits of where the extreme rays of the virtual camera exist in the light field. This space is simply the intersection of the spaces of where the extreme rays can be rendered. Figure 5-13a shows the valid position where the first ray can be rendered, and Figure 5-13b shows the valid positions for the last ray. Figure 5-13c is the intersection of the two spaces and represents the complete range of motion.

Up till now, I have assumed that the field of view of the virtual camera is the

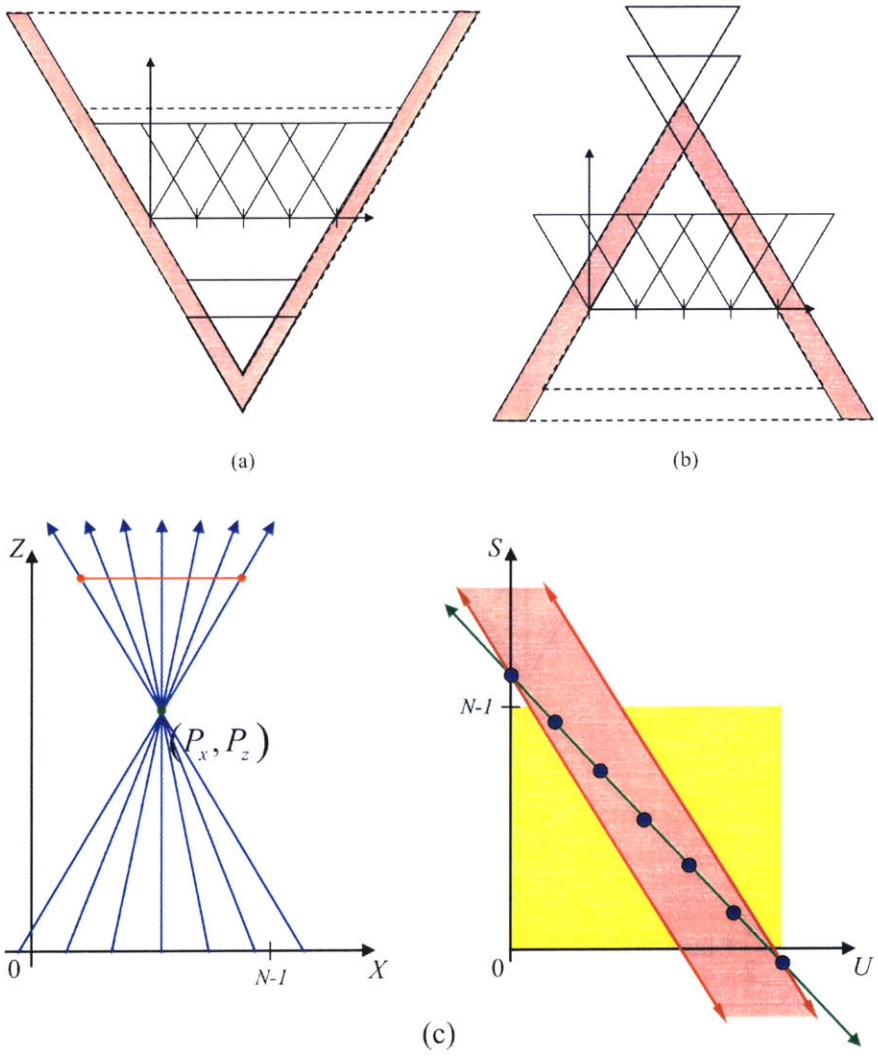


Figure 5-12: Camera motion is restricted to where the first and last rays of the virtual camera exist in the light field. (a) Moving behind the camera plane. (b) Moving in front of the camera plane. (c) On the EPI the effect is to sample outside of the existing camera samples in yellow.

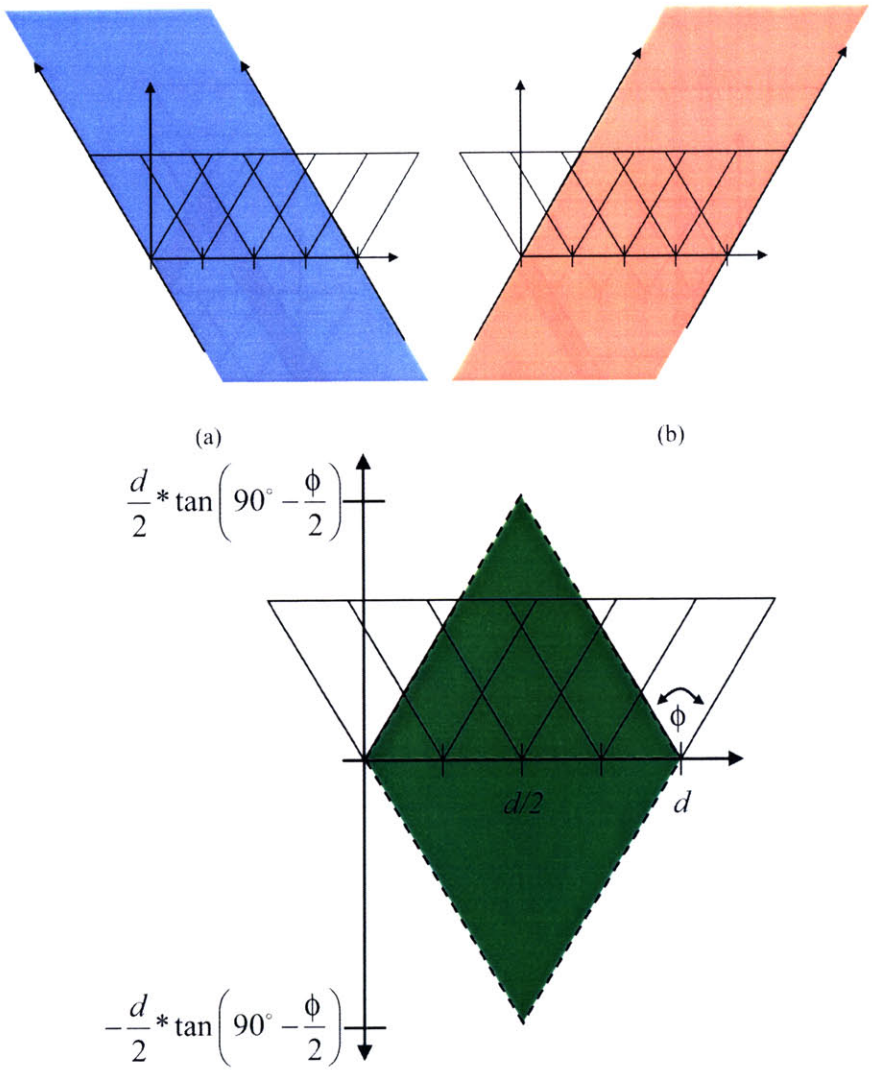


Figure 5-13: (a) In this space all rays corresponding to the first ray angle in the virtual camera can be rendered. (b) This space represents the valid positions for the last ray angle. (c) Intersection of the spaces forms the complete range of motion for a translating, non-rotating virtual camera with a field of view that is the same as the real cameras.

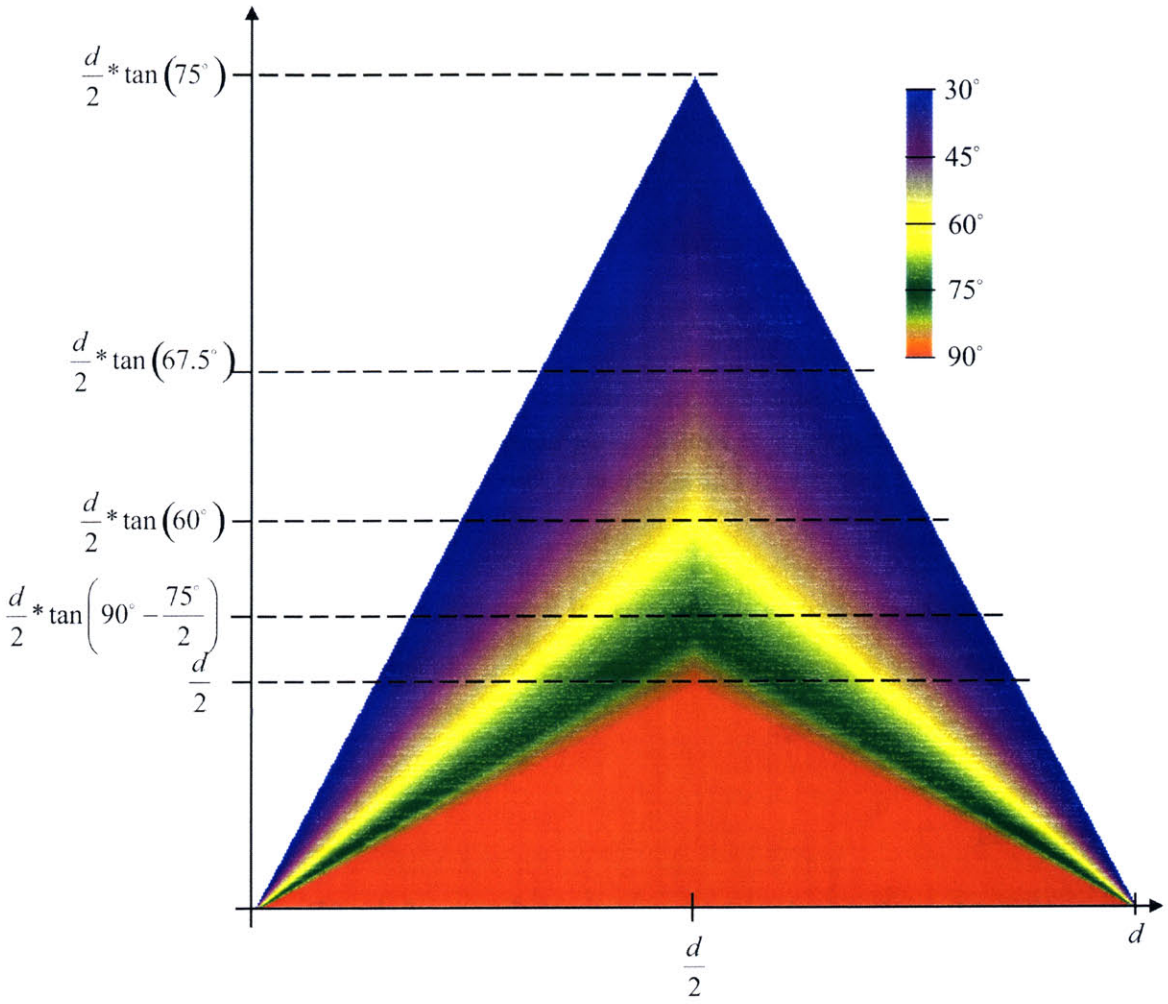


Figure 5-14: This plot shows the possible camera positions relative to the camera array (field of view 90°) of length d for a range of field of views.

same as the real cameras. Now I relax this assumption by allowing the field of view to be any value less than that of the real cameras. The range of motion for the virtual camera is still bounded by the extreme ray angles of the virtual camera that exist in the first and last cameras. However, by decreasing the field of view, the depth at which I can position the virtual camera increases, thus increasing the range of motion. Figure 5-14 shows how range of motion changes as the field of view changes.

In summary:

With a field of view less than that of the real cameras, the range of motion (no rotations) for a virtual camera is the region formed by the intersection of the spaces

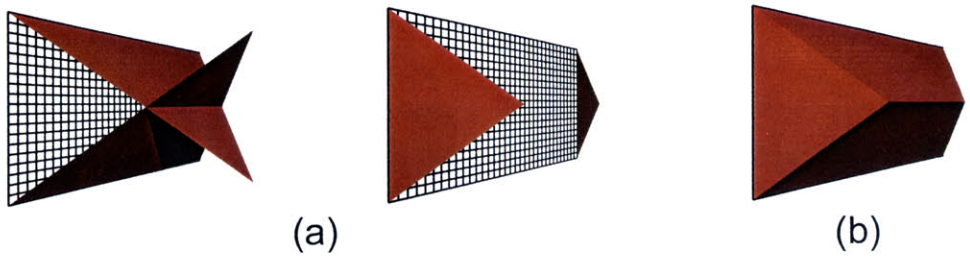


Figure 5-15: (In this example the field of view is 90°.)(a) Constructing the range of motion. Since one dimension is longer than another, extreme rays do not meet at a point. (b) Range of motion in 3D space is a volume symmetrical about the camera plane.

where the extreme ray angles of the virtual camera can be rendered. This space is bounded by the lines representing the extreme ray angles found in the first and last cameras of the camera array.

3D Cameras

A 4D light field can be analyzed as an extended case of the 2D version. The virtual camera is actually multiple 2D cameras sampling different EPI slices of the 4D light field. If thought of this way the range of motion rules derived in the 2D case holds, but in both directions (horizontal and vertical).

As in the 2D case the limits of the range of motion are determined by the extreme rays of the cameras in the array. In the 3D case they are the corner rays. Each corner ray is an edge of a triangle formed by two neighboring corner rays and the line between corner cameras. The range of motion bounds is a convex polyhedron enclosed by these triangles.

Figure 5-15 is an example of the range of motion for a regularly sampled 2D camera array where the number of cameras in each dimension is not equal. The bounds are formed by the corner rays. Notice that with the added dimension the rays do not all meet at a single point. This is because the sampling distance along the x-axis is greater than that of the y-axis. Thinking back to the 2D case, with more cameras, the range of motion is larger, so triangles will intersect at a segment as shown.

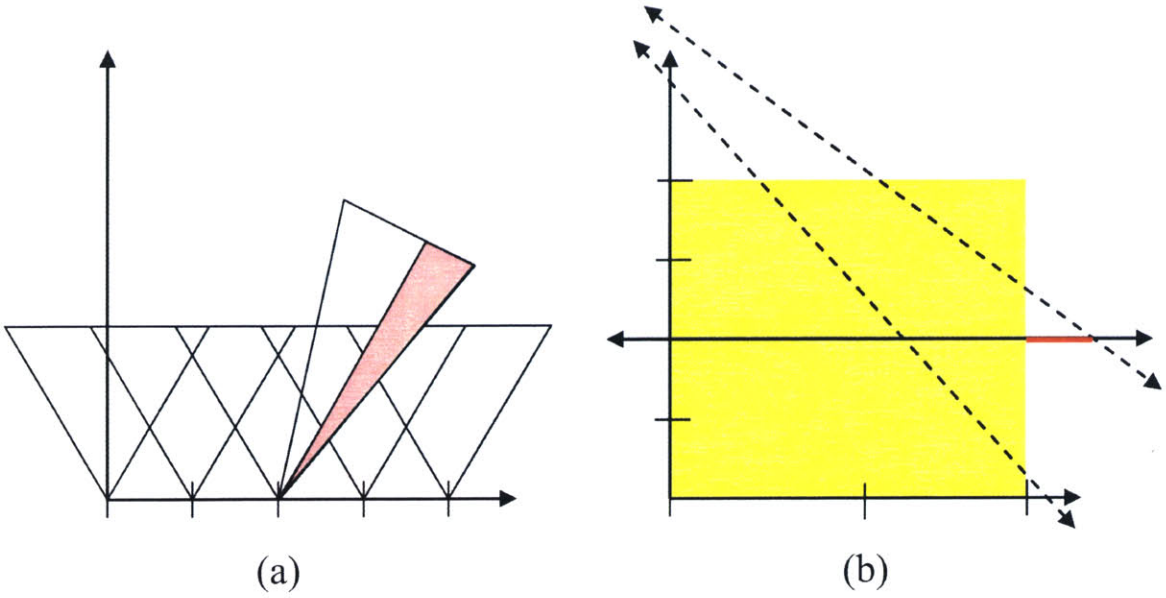


Figure 5-16: The ability to rotate the camera and still render a valid image is restricted by the field of view of the real cameras. (a) Rays in red do not exist in the light field. (b) The yellow box represents the EPI. The horizontal line represents the camera position. The dotted lines bound the field of view on the line. The red segment represents the missing rays outside of the sampled region.

5.2.2 Rotations

2D Cameras

When the virtual camera rotates about its center of projection, it captures the same set of rays, but sampling on the image plane varies. Previously I showed that the field of view of the virtual camera cannot be greater than that of the real cameras because the extreme ray angles are not found in the light field. Therefore, in order for the virtual camera to rotate, the field of view of the virtual camera must be less than the real cameras. Also, the range of rotation is bounded by the field of view of the real cameras. This is illustrated in Figure 5-16. Samples that are beyond the field of view of the real cameras are outside of the EPI.

Just as the range of motion changes as the field of view decreases, the range of motion changes as the virtual camera rotates. Determining the bounds is similar to before. When the virtual camera is on the camera plane the motion is bounded by the position of the first and last real camera. When the virtual camera is in front

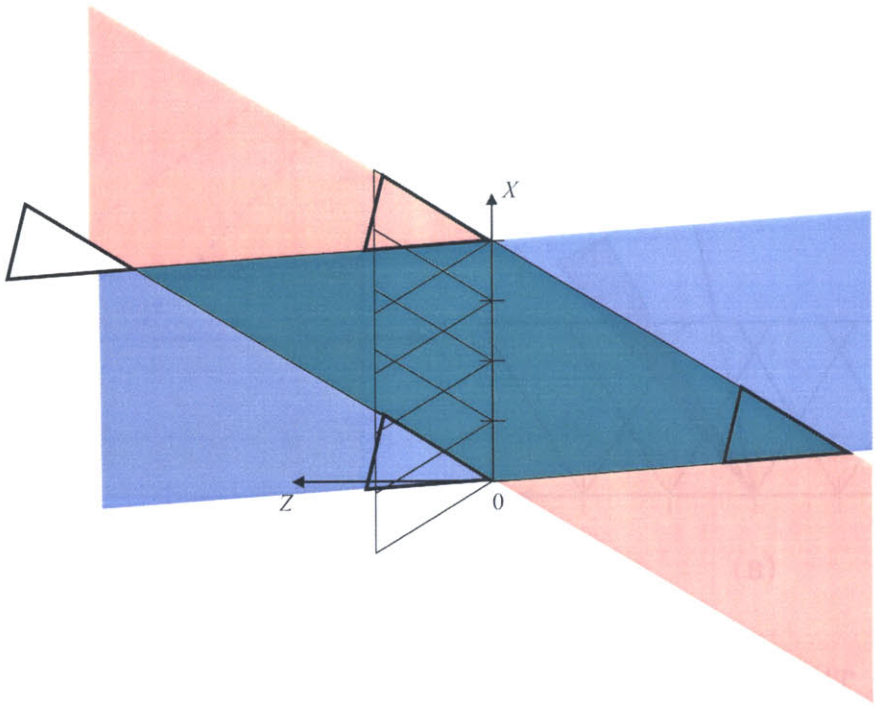


Figure 5-17: Range of motion construction with a rotated camera. The area in blue represent where the first ray of the now rotated virtual camera can be rendered. The area in red represent the last ray of the virtual camera. The intersection of the two spaces form the range of motion for a virtual camera rotated at a certain angle. (Note that he overhead view is different from before and is rotated.)

of the camera plane the boundaries are the ray of the first camera with the same orientation as the first sampled ray in the virtual camera and conversely with the last sampled ray orientation and the last real camera. The opposite occurs for positions behind the camera, therefore the rule from Section 5.2.1 still holds as long as the camera does not rotate past the field of view of the real cameras. Figure 5-17 shows an example of how the range of motion is bounded as the virtual camera rotates.

3D Cameras

After discussing the various virtual camera scenarios, the range of rotation for a virtual camera sampling a 4D light field becomes evident. First, the virtual camera cannot rotate beyond the field of view of the real cameras in the camera array. Second, the range of motion is again bounded by the extreme rays of the camera array.

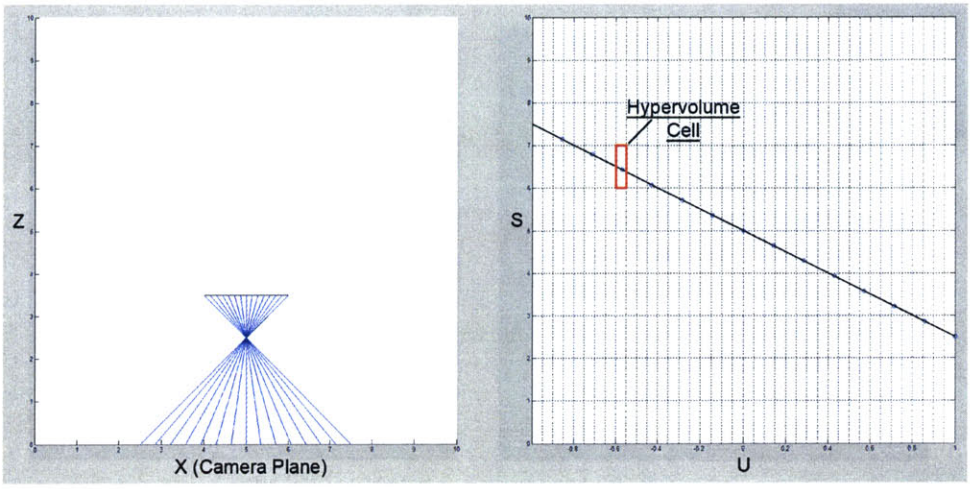


Figure 5-18: The hypervolume cell is the grouping of real samples that contribute to the desired ray. In the figure, four samples (two from neighboring cameras) contribute to a ray.

5.3 Resolution

Plenoptic Sampling [5] determined the minimum sampling on the ST (camera) plane such that the projection error on the image plane is less than one pixel. However, there was no analysis of the UV sampling (resolution) only that the virtual camera resolution was set to the same resolution as the real cameras.

The goal of the following analysis is to determine the sampling rate (or maximum rendering resolution) such that the there is no oversampling or undersampling of the light field. Like the range of motion analysis, I will primarily discuss the 2D case.

5.3.1 Effective Resolution

Virtual views can be rendered at any resolution. When the resolution is greater than that of the real cameras then adjacent rays in the rendered image are possibly being interpolated from the same set of rays in the light field. In other words, adjacent pixels in the rendered image are constructed from the same light field samples. Therefore, I define the effective resolution of an image to be the number of rays that are interpolated from distinct samples in the light field.

Figure 5-18 further demonstrates the relationship between the virtual camera and

Figure 5-19: Enlarging an exactly sampled image (left) vs. rendering at a high resolution (right). Although the images are not exact, the effect is still a blurry image.

EPI sampling. A hypervolume cell is the interpolation kernel that contributes to a particular ray. This figure uses bilnear interpolation of the closest two pixels from the closest two cameras. If this was a 3D camera then the interpolation would be quadralinear (closest four pixels from the closest four cameras).

It should be clear how the effective resolution is defined. As the resolution of the virtual camera increases, the number of samples in the EPI increases. Even if the camera line in the EPI is densely sampled, only a finite number of hypervolume cells contribute to the rendered image. This is the effective resolution.

When a high resolution image is rendered from a low resolution light field the result is a blurred image. The same effect could have been achieved by rendering at the effective resolution and enlarging the image. The same set of light field samples are used in both images. Figure 5-19 is an example of this effect. Although the weighting of the light field samples are different between the two images, the result is still two blurry images. The benefit is eliminating multiple, expensive fetches for the same light field samples.

To achieve optimal sampling, each sample should fall in a separate cell. Distance between each sample must be at least ΔU in the U direction (physical resolution) or ΔC in the S direction (camera spacing). This corresponds to the Nyquist limit of

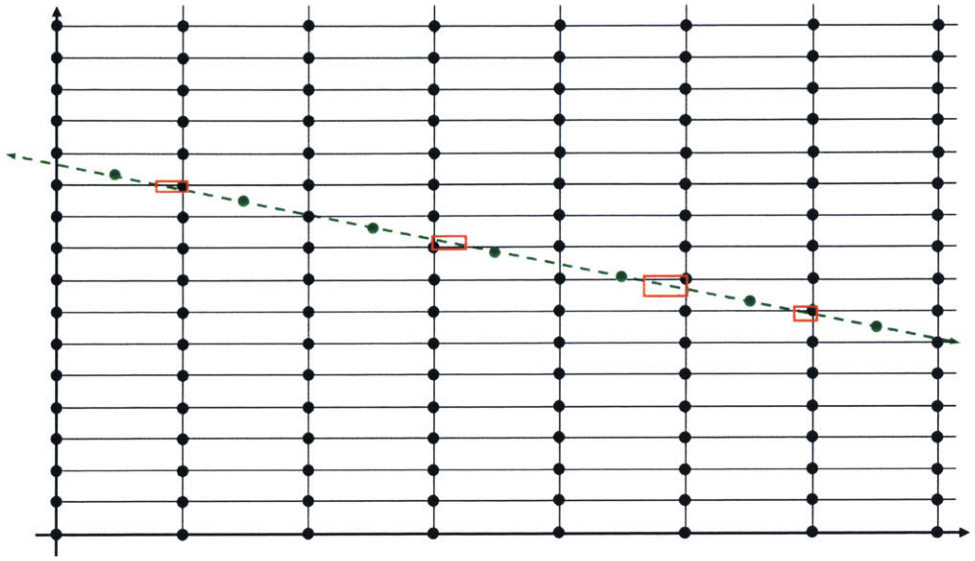


Figure 5-20: Sampling at the Nyquist limit. The red boxes show where the lines clip a cell that is not sampled. The issue here is the reconstruction filter. The filter could have been enlarged to encompass those small areas, but would result in aliasing of the filter.

the EPI. The maximum frequency would be every other box (a high and a low for each two box block) or $\frac{1}{2\Delta U}$. Two times the maximum frequency is $\frac{1}{\Delta U}$ which is one sample per adjacent hypervolume cells. (Analysis is the same for the S-axis.)

There are situations where the sampling line intersects cells that do not contribute to the rendered image (Figure 5-20). This is still within the Nyquist limit. As the slope changes, there is a corresponding change in the distance between samples. In order to account for those samples the reconstruction filter could have been increased, which would result in aliasing due to the reconstruction filter.

5.3.2 General Sampling of the Light Field

I will derive a general sampling equation relating virtual camera sampling to UV sampling (or in the 2D case U sampling). I begin with the camera model from [14] (Figure 5-21).

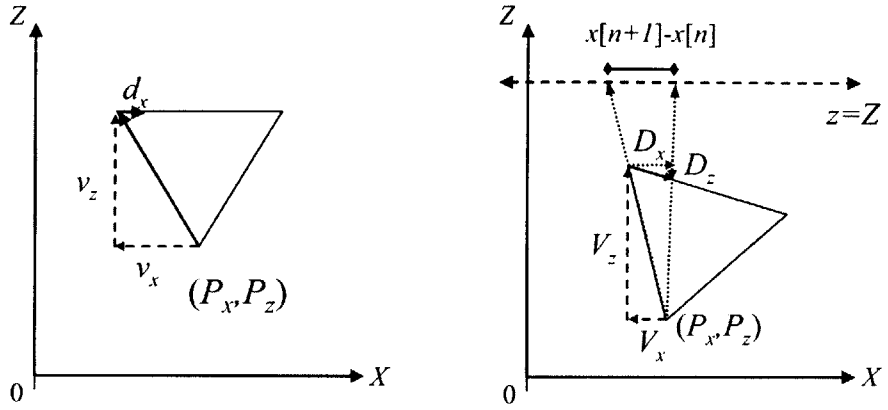


Figure 5-21: (a) Camera model. P is the camera position. $\vec{v}$ is the vector to the first pixel. $\vec{d}$ is the sampling direction and size. (b) Projection of two adjacent sampling position on the image plane onto a line in space.

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} P_x \\ P_z \end{bmatrix} + \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} d_x & v_x \\ d_z & v_z \end{bmatrix} \bullet \begin{bmatrix} n \\ 1 \end{bmatrix}$$
$$\Rightarrow \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} P_x \\ P_z \end{bmatrix} + \begin{bmatrix} D_x & V_x \\ D_z & V_z \end{bmatrix} \bullet \begin{bmatrix} n \\ 1 \end{bmatrix} \tag{5.8}$$
$$\begin{bmatrix} D_x & V_x \\ D_z & V_z \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} d_x & v_x \\ d_z & v_z \end{bmatrix}$$

P and θ is the camera position and orientation respectively. $\vec{V}$ is the vector to the first pixel sample and $\vec{D}$ is the sampling vector. (x, y) is the sample position in space of the n^{th} pixel.

Cameras at zero degrees of rotation are oriented in the same direction as the z-axis. The vector to the first sample before rotation is then

$$\begin{aligned} v_x &= -f_{virt} * \tan\left(\frac{\phi}{2}\right) \\ v_z &= f_{virt} \end{aligned} \tag{5.9}$$

where ϕ is the field of view. In the case where the image plane is parallel to the x-axis, the sampling vector of the camera will be at zero degrees of rotation and $\vec{d}$

$$\begin{aligned} d_x &= \frac{2*\tan\left(\frac{\phi}{2}\right)}{res} \\ d_z &= 0 \end{aligned} \tag{5.10}$$

A 2D camera samples along an image line in space (or projects camera rays onto a line in space). For simplicity this line will be parallel to the camera (S) line. For a sample (n) projected onto the line $z = Z$, the x coordinate is, from (5.8), then

$$x[n] = P_x + \frac{(Z - P_z) * (V_x + n * D_x)}{V_z + n * D_z} \tag{5.11}$$

After substitution and expansion, the sampling rate on this line (distance between samples) is then

$$\begin{aligned} \Delta x[n] &= x[n+1] - x[n] \\ &= (Z - P_z) * \\ &\left(\begin{array}{c} \frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + (n+1)\cdot d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + (n+1)\cdot d_x\sin(\theta)} \\ -\frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + n\cdot d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + n\cdot d_x\sin(\theta)} \end{array} \right) \end{aligned} \tag{5.12}$$

This equation gives the sampling rate on a line in space parallel to the line of real cameras using a virtual camera with position (P_x, P_z), rotation of θ degrees, field of view of ϕ degrees, and a sampling rate of d_x.

From the analysis in Section 5.1.3, the slope of the sampling line in the EPI gives the relationship between the sampling of the S and U-lines. Therefore, camera samples are projected onto the S-line $z = 0$. Sampling rate of the S-line is then (from Equation (5.12))

$$\begin{aligned} \Delta S[n] &= -P_z * \\ &\left(\begin{array}{c} \frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + (n+1)d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + (n+1)d_x\sin(\theta)} \\ -\frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + nd_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + nd_x\sin(\theta)} \end{array} \right) \end{aligned} \tag{5.13}$$

Using the slope of the sampling line in the EPI and (5.4), the relationship between sampling on the U-line and sampling on the S-line is

$$\Delta u = \Delta s * \left(-\frac{f_{real}}{P_z} \right) \tag{5.14}$$

Combining (5.13) and (5.14) gives the relationship between sampling in the virtual camera and sampling in the EPI.

$$\Delta u\,[n] = f_{real} * \begin{pmatrix} \frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + (n+1)d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + (n+1)d_x\sin(\theta)} \\ -\frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + nd_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + nd_x\sin(\theta)} \end{pmatrix} \tag{5.15}$$

In Equation 5.15, P_z has dropped out. The significance of this result is that *sampling of the UV plane is independent of the position of the virtual camera.* This can be explained intuitively through the EPI. As the virtual camera moves away from the camera array, the sampling interval on the camera line (S-axis) increases, but at the same time the slope of the sampling line changes at the same rate. (Figure 5-22)

5.3.3 Camera Plane vs. Focal Plane Resolution

Sampling of the light field is also dependent on the camera plane and the focal plane resolutions. Looking at the EPI, changing either resolution changes the sampling rate along each axis. This in turn alters the hypervolume cells the camera line crosses. For example, when the virtual camera lies on the camera plane, the image plane (UV) sampling "dominates" so that the effective resolution of the virtual camera is the same as the physical cameras. Whereas when the virtual camera is at infinity, meaning the corresponding line in the EPI is vertical, then the effective resolution would be the sampling rate of the camera plane.

Therefore, from Figure 5-23, when analyzing light field sampling there are two cases to consider. One is when the absolute value of the slope of the sampling line is less than $\frac{\Delta C}{\Delta U}$ where ΔC is the distance between cameras and ΔU is the sampling rate of the camera. This is when image plane sampling dominates. The other case is when the absolute value of the slope is greater than $\frac{\Delta C}{\Delta U}$ which corresponds to camera plane sampling importance.

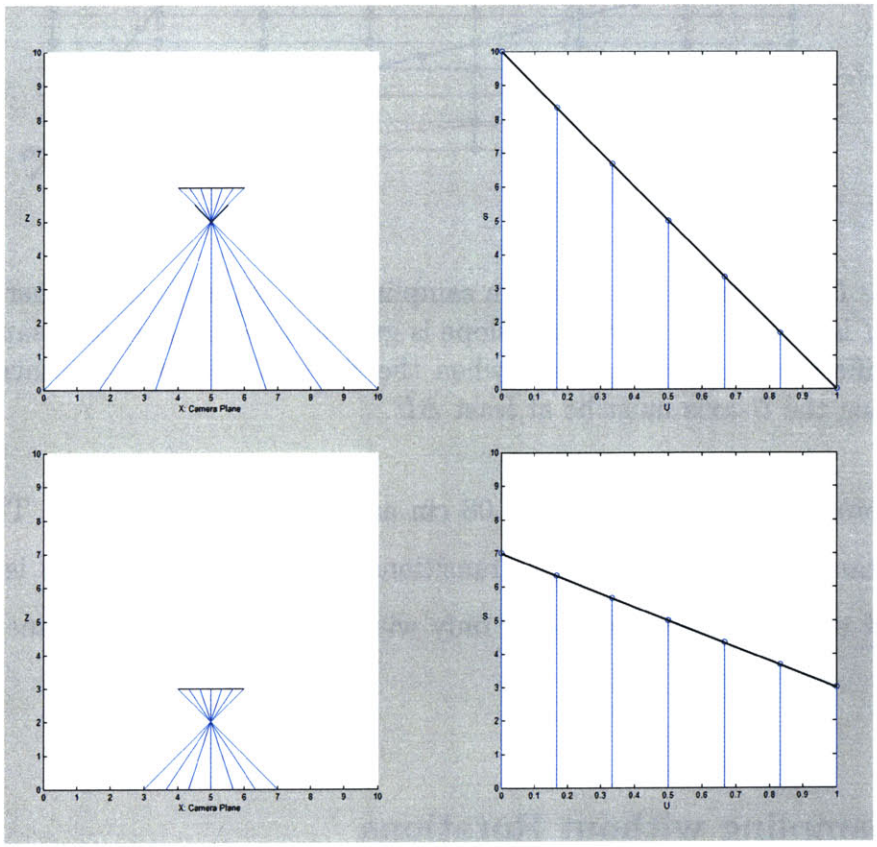


Figure 5-22: Sampling rate of the light field is independent of the position of the camera. As the virtual camera moves away from the camera plane (above), the S sampling interval increases. However, since the slope of the sampling line changes at the same rate as the camera movement, U sampling stays the same.

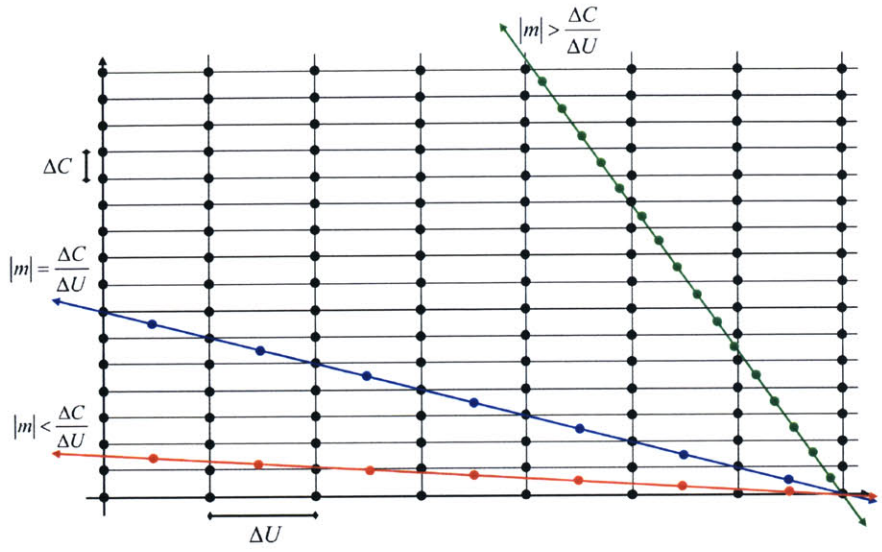


Figure 5-23: Determining the minimum sampling rate is based on whether the slope is greater or less than $\frac{\Delta C}{\Delta U}$. When the slope is greater (green) then each sample must sample a different camera. Whereas when the slope is less (red) distance between samples along the U-axis must be at least ΔU.

In the prototype system, $\Delta C = 5.08$ cm and $\Delta U = 0.001804$ cm. The virtual camera distance at which the slope transitions between the two cases is 28.16 m. For the rest of the chapter I will deal only with areas where image plane sampling dominates.

5.3.4 Sampling without Rotations

I define oversampling of the light field to be when the virtual camera samples the same hypervolume cell for consecutive pixels of the rendered image. Undersampling on the other hand occurs when consecutive pixels of the virtual camera do not sample adjacent hypervolume cells. (Figure 5-24)

The question to answer is: What is the minimum sampling rate (or maximum resolution) such that there is no oversampling of the light field, and the opposite for undersampling. Before adding rotations, I start with the simple and common case where the virtual camera orientation is $\theta = 0$. By setting $\Delta u = \Delta U$, Equation (5.15) reduces to

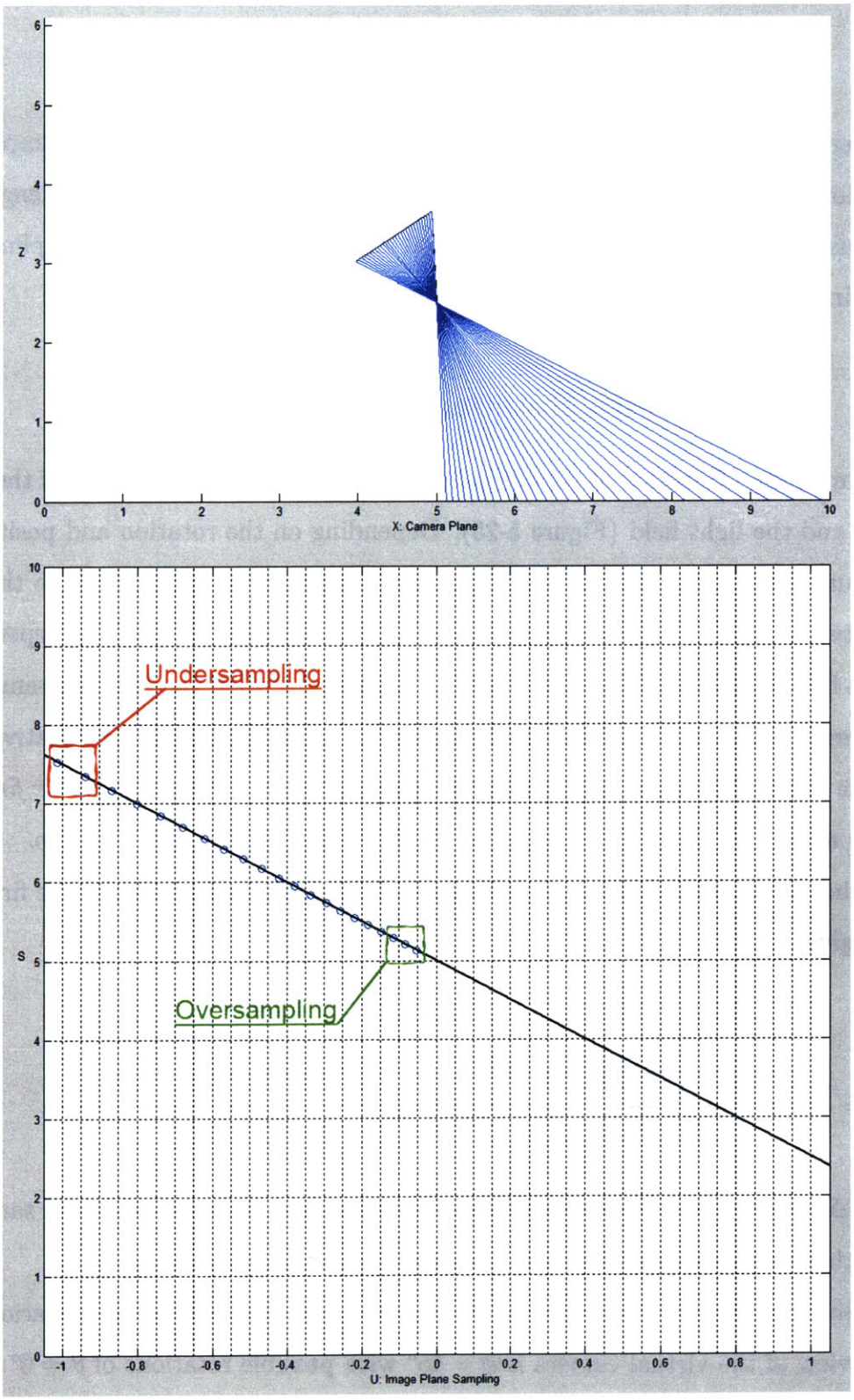


Figure 5-24: Oversampling (in green) is when adjacent samples fall in the same cell. Undersampling (in red) is when adjacent samples do not fall in adjacent cells.

$$d_x = \Delta U * \frac{f_{virt}}{f_{real}} \tag{5.16}$$

Under the condition that there is no rotation in the virtual camera, sampling is proportional by a scale factor based only on the ratio between the focal lengths of the virtual and real cameras. When the focal lengths are the same the sampling rate of the virtual camera should be the same as the real cameras.

5.3.5 Sampling with Rotations

Oversampling Rotation of the camera results in non-uniform sampling of the camera line and the light field (Figure 5-25). Depending on the rotation and position of the virtual camera, the smallest or largest sampling interval corresponds to the first or last two samples. With a positive rotation, the smallest interval corresponds to the first two samples and conversely with a negative rotation the last two samples.

However, sampling is mirrored when rotating either positively or negatively by the same angle. Therefore, *the smallest sampling interval corresponds to the first two samples using a rotation of the negative absolute value of the given rotation.*

Solving for d_x from (5.14) and setting $n = 0$ (sampling rate between the first two pixels) gives

$$d_x = \frac{f_{virt} * \Delta u * \left(\tan\left(\frac{\phi}{2}\right)^2 \sin(\theta)^2 - 2\tan\left(\frac{\phi}{2}\right)\sin(\theta)\cos(\theta) + \cos(\theta)^2\right)}{f_{real} + \Delta u * \left(\sin(\theta)^2 \tan\left(\frac{\phi}{2}\right) - \sin(\theta)\cos(\theta)\right)} \tag{5.17}$$

Setting Δu equal to the U-axis sampling of the light field gives the minimum sampling rate of the virtual camera such that the light field is never oversampled.

As an example, here is a reasonable camera array and rendering scenario. The field of view of the virtual camera is $\phi = 45°$ with possible rotations of $\theta = 0° \rightarrow 15°$ degrees (which implies that the real cameras have at least a 60° field of view). I set the focal lengths to be the same and set the resolution of the real cameras to be 640 pixels with a field of view of 60° ($\Delta U = 0.0018$).

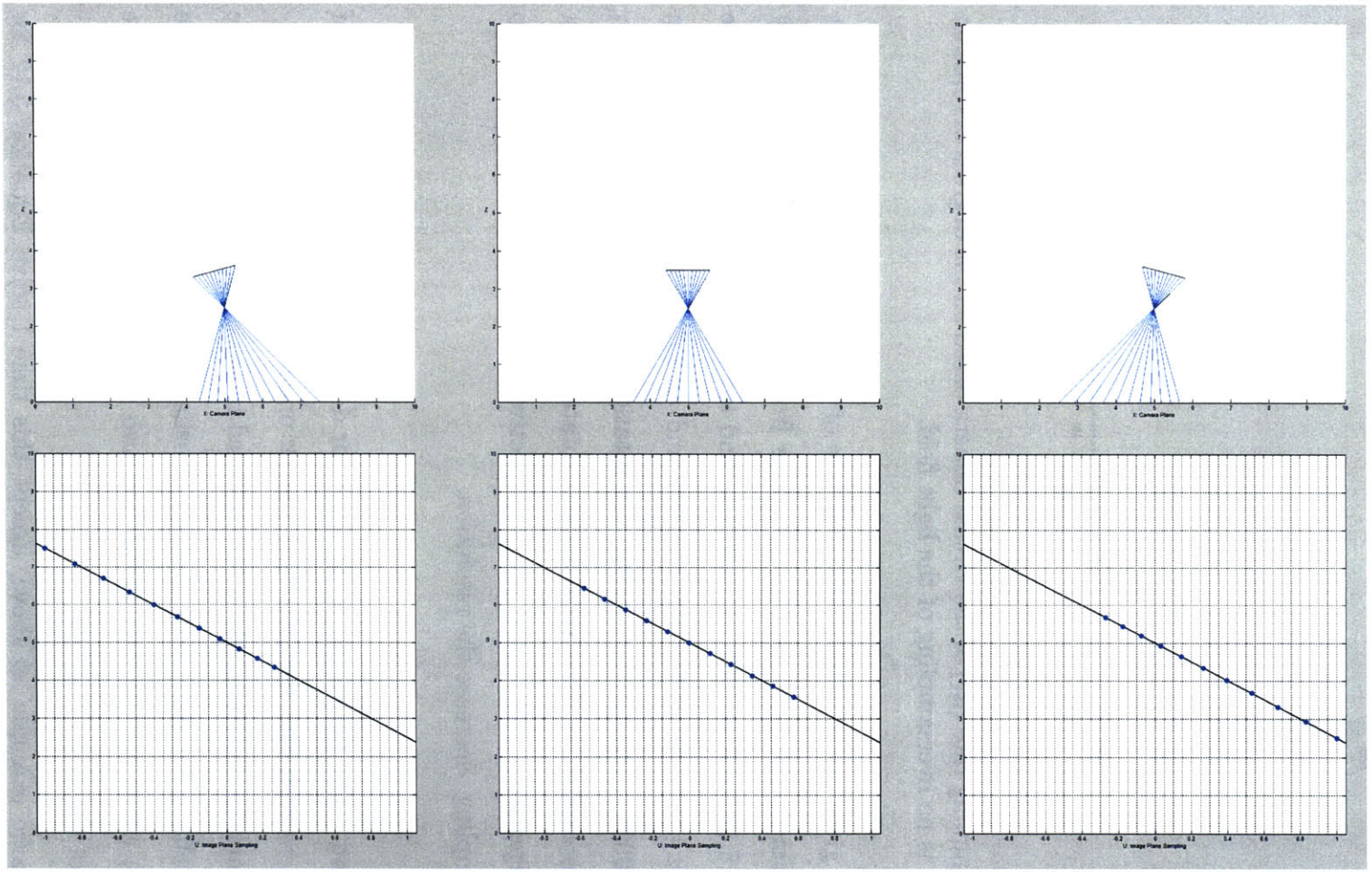


Figure 5-25: Rotating the virtual camera will result in nonuniform sampling of the light field. The first camera-EPI pair is of a positive rotation. The last pair is of a negative rotation. Notice that in a negative rotation the first two samples have the smallest interval and the last two have the largest interval. The reverse occurs during a positive rotation.

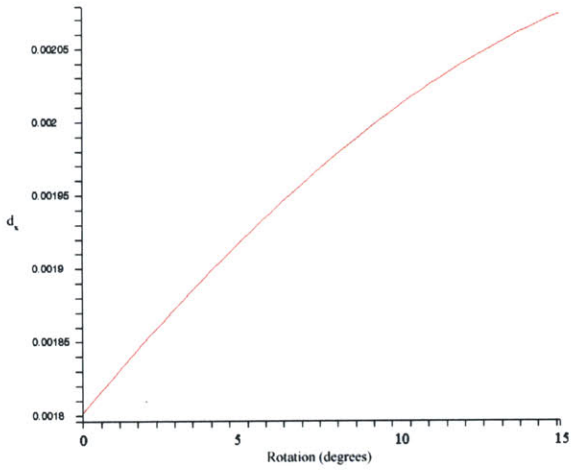


Figure 5-26: Sampling rate (d_x) of the virtual camera based on the degrees of rotation such that there is no oversampling of the light field.

Figure 5-26 gives the minimum sampling rate of the virtual camera such that it does not oversample the light field for rotations from 0° to 15°. (I must actually solve Equation 5.17 for $\theta = -|0°| \rightarrow -|15°|$ to find the smallest interval.) At 0° the sampling rate is the same as the real cameras and at 15° of rotation the sampling rate is $d_x \approx 0.0021$ - greater than before the rotation. Looking at it another way, to prevent oversampling as the virtual camera rotates from 0° to 15°, the rendering resolution changes from 459 to 400 pixels respectively. This result is obvious - to prevent oversampling, decrease the resolution.

Undersampling The derivation is the same for determining the sampling rate of the virtual camera such that the light field is never undersampled. I define undersampling to be when consecutive pixels in the rendered image do not sample consecutive hyper-volume cells in the light field. Instead of using the smallest interval I use the largest interval, which corresponds to the first two samples using a rotation of the absolute value of the given rotation.

I use the same example as above, except this time I solve Equation 5.17 for $\theta = |0°| \rightarrow |15°|$. Figures 5-27 demonstrate that as the virtual camera rotates, the sampling rate must increase in order to prevent undersampling of the light field.

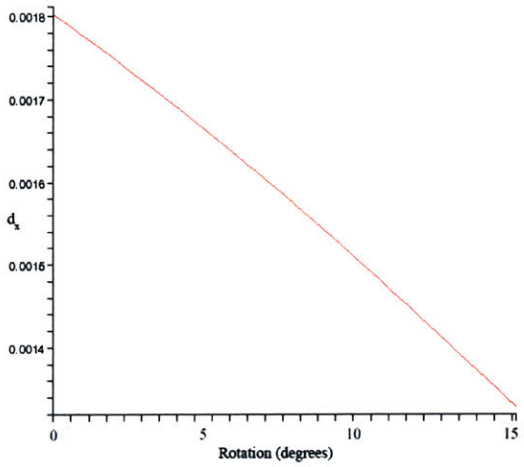


Figure 5-27: Sampling rate (d_x) of the virtual camera such that there is no undersampling of the light field when rotated.

Again this should be obvious - to prevent undersampling, increase the resolution.

5.3.6 Sampling, Scalability, and the Prototype System

In Chapter 4, I explained that due to the rendering algorithm, there is a limit to the scalability because if the number of cameras exceed the desired resolution it is possible that there would be sub-pixel contributions to the rendered image. In the worst case all cameras would contribute to the final image. The EPI explanation is presented in Figure 5-28. This is the case of rendering at a lower resolution than the light field.

Figure 5-29 is an example of rendering a low resolution image using the reconstruction filter of this chapter. A pixel is the integral of all rays that intersect the pixel from the center of projection. To render at a lower resolution, neighboring samples must be interpolated. The effect is to render at the highest resolution and downsample to a lower resolution.

On the other hand, I have previously been treating pixels and rays as the same, which is possible because the light fields were of low resolution. This is equivalent to a low-pass pre-filter vs. a post-filter in the above case.

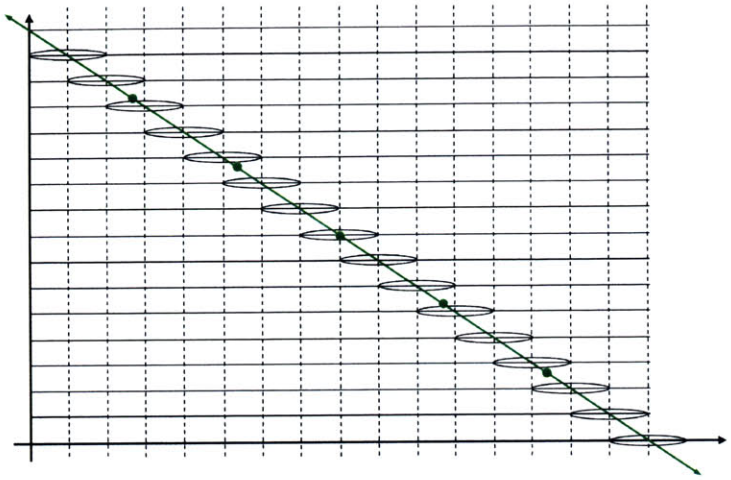


Figure 5-28: In this figure, the vertical lines are not image samples. They represent the intersection between the sampling line and the real camera lines (horizontal). When the light field is at a higher resolution (either on the camera or image plane) than the desired image, then the rendering algorithm in Chapter 4 will have sub-pixel contributions. Assuming infinite image plane resolution, the ovals represent the contribution of each camera to the final image. The center of the oval has a weighting of one (at the intersection) and falls off to zero (neighbor intersections). The green dots represent actual pixel centers. With a low resolution rendering, multiple ovals (which represents multiple pixels) contribute to each pixel.

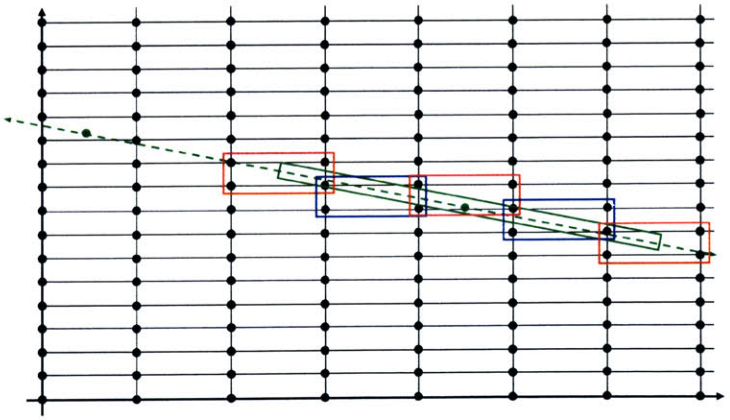


Figure 5-29: The green dots represent the pixel centers of the final low resolution image. A pixel is a contribution of multiple rays falling within the green box. The red and blue boxes surround the samples of the light field that are used to compute those rays and the final pixel.

Essentially, there are tradeoffs in terms of the light field sampling methodology. The finite-viewpoint architecture can still scale to the number of virtual cameras depending on the rendering algorithm and capture settings. For low resolution images, the system could capture at a lower resolution and with a fewer number of cameras or sample at exact ray intersections of the EPI and filter out aliasing artifacts.

It is possible to alter the prototype system presented in Chapter 4 to sample at different rates (both on the ST and UV planes). The simplest solution would be to render at the proper resolution and use the graphics hardware to map the rendered frames into higher or lower resolutions. The number of cameras used to render can also be controlled in software. Another solution is to replace the graphics hardware with ones that support programmable shaders. Then rendering can be performed as a per pixel computation, such as using the renderer in Appendix A.

5.3.7 3D Cameras

As stated before, 3D cameras are closely related to 2D cameras. Therefore the previous analysis easily extends to the 3D case. Each 2D slice of the 3D camera use a different EPI, but the sampling rate is the same so the equations still hold. The most important result to take away is that the rendering resolution or the sampling of the light field is independent of the depth of the virtual camera from the camera array up until the point where camera plane resolution is more important than image plane resolution.

5.4 General Position

Up till now I have been analyzing the case of the wall-eyed configuration. This not the only possible camera setup. Levoy and Hanrahan [17] rotate the cameras such that the entire target object stays within the field of view of the cameras. These images are then rectified into the fixed, two plane parameterization. This section will discuss how changing the camera orientation affects range of motion and resolution.

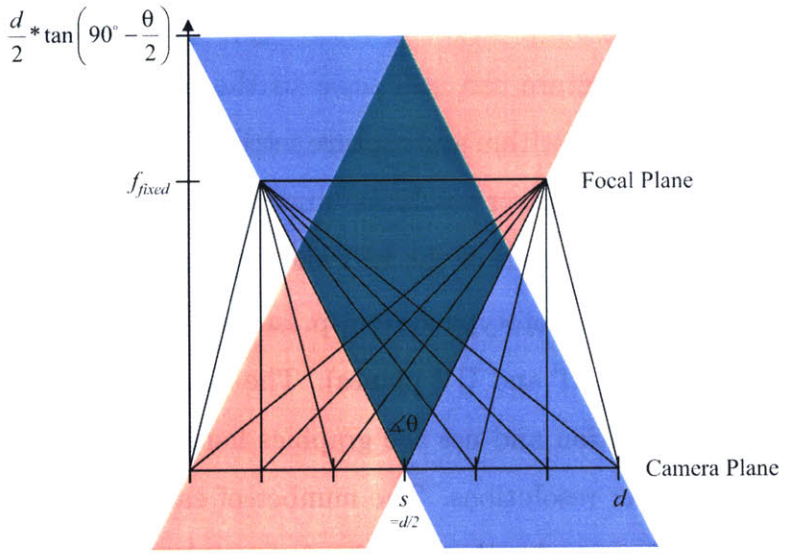


Figure 5-30: A camera array with skewed perspective cameras with a fixed focal plane a finite distance away. The green is the valid positions for virtual cameras with the same field of view as camera s.

Range of Motion The first question to ask is "How does this change the range of motion?" If the goal is still to restrict valid images to be those where the rays can be found in the LF, then the construction is still the same. For example, looking at Figure 5-30, if I wanted to determine the valid positions for a virtual camera with the same field of view as camera s, I would previously find where the first and last rays intersect the last and first camera. However, the area of valid cameras for specific ray angles is now smaller due to the rotation of the cameras. The first ray angle of camera s does not exist in cameras to the left of it, but only exist in the blue region. The converse occurs for the last ray angle in camera s. In fact, it should be clear from Figure 5-30, that the only valid area of movement is in front of the camera plane with the same maximum depth equal to the maximum depth from before. Therefore, in terms of range of motion, if the valid camera assumption is the same, then the wall-eyed configuration has a greater area of movement.

However, since the goal of this camera configuration is usually to capture an object and not a scene, it would be reasonable to relax the valid image definition. Basically as I translate a virtual camera, I will begin to have invalid (not found in the light field) rays, but those rays would not have seen the object anyways.

Resolution: Skewed Perspective Cameras Now looking at the sampling, instead of using rotated cameras, I will substitute with cameras that have a skewed perspective projection (the physical embodiment is a camera with a bellows). (Figure 5-30) Earlier in this chapter I made a distinction between a fixed vs. a relative parameterization. However, relative coordinates is actually a special case of the fixed parameterization when the image plane is at infinity. Adding a skew factor, Equation 5.2 becomes

$$u'(n) = u(n) + C(n) * \frac{f_{real}}{f_{fixed}} \tag{5.18}$$

where f_{fixed} is the distance of the fixed focal plane. As $f_{fixed} \rightarrow \infty$, $u'(n) = u(n)$.

Construction of the EPI remains the same, except now a vertical line (infinite slope) represents the depth of the fixed focal plane, whereas before, it represented a focal plane at infinity. The slope equation becomes

$$m = -\frac{1}{f_{real} * \left(\frac{1}{P_z} - \frac{1}{f_{fixed}}\right)} \tag{5.19}$$

As $f_{fixed} \rightarrow \infty$, $m = -P_z/f_{real}$, which is the same as (5.4).

Rearranging (5.19),

$$\Delta u = -\Delta s * f_{real} * \left(\frac{1}{P_z} - \frac{1}{f_{fixed}}\right) \tag{5.20}$$

Substituting (5.20) into (5.13)

$$\Delta u[n] = f_{real} * \left(1 - \frac{P_z}{f_{fixed}}\right) * \begin{pmatrix} \frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + (n+1)d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + (n+1)d_x\sin(\theta)} \\ -\frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + nd_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + nd_x\sin(\theta)} \end{pmatrix} \tag{5.21}$$

Setting $\theta = 0$ (image plane is parallel to the camera plane) and assuming the focal length of the virtual camera (f_{virt}) is the same as the real cameras (f_{real})

$$d_x = \frac{\Delta u}{\left(1 - \frac{P_z}{f_{fixed}}\right)} \tag{5.22}$$

This means that the sampling rate of the virtual camera changes as it moves away from the camera plane, which is different from the results based on a wall-eyed array. The effective resolution is only the same as the real cameras when it is on the camera plane or if the focal plane is at infinity.

From the last section, sampling changes as the slope of the sampling line increases past $\frac{\Delta C}{\Delta U}$. In this case, this occurs when $|m| = \frac{\Delta C}{\Delta U}$. Substituting with (5.19)

$$\frac{1}{P_z} = \frac{\Delta U}{f_{real} * \Delta C} + \frac{1}{f_{fixed}} \tag{5.23}$$

This result and (5.19) show that the slope changes faster than the earlier case since infinite slope occurs at a finite distance. Assuming $\theta = 0$ and $f_{virt} = f_{real}$

$$d_x = -\frac{\Delta C * f_{virt}}{P_z}$$

Which is the same result as substituting $\theta = 0$ and ΔC for ΔS into Equation 5.13.

Resolution: Rotated Cameras Replacing the skewed cameras, with rotated cameras has two effects. First, it should be clear from the analysis of Section 5.3.5 how sampling the fixed focal plane is non-uniform due to the camera rotations. Second, as the camera rotates and translates away from the center, the distance to the object inevitably increases. If the camera model is the same at every position in the array, then the sampling density on the focal plane is different for each camera. See Figure 5-31. This is unlike the previous camera configurations where the image plane is parallel to the camera plane.

The EPI can be constructed by projecting samples, but the EPI will be non-uniform. Levoy and Hanrahan [17] instead warps each image into the two plane parameterization. Each sample in the new EPI is an interpolated value.

Analyzing resolution is difficult when the EPI is non-uniform. Figure 5-32 is the

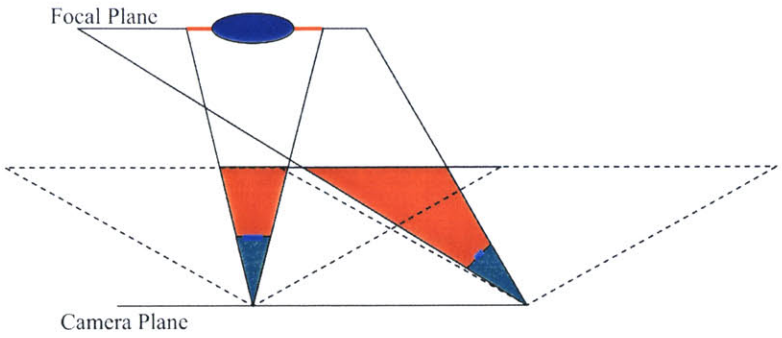


Figure 5-31: As cameras are rotated and translated away from the center, the fixed focal plane will be non-uniformly sampled. For the same area on the focal plane (i.e., the blue oval) each camera will have a different sampling density.

EPI using the projected samples of the rotated cameras onto a common plane.

The main conclusions to draw regarding resolution and rotated or skewed cameras are that as the virtual camera moves away from the camera plane, the effective resolution decreases and also the sampling of the scene rays is non-uniform.

5.5 Summary

In this chapter, I have used EPIs to analyze the virtual camera capabilities of a light field. Under the wall-eyed configuration, I showed how to derive the range of motion and described the effective resolution of the camera in terms of oversampling and undersampling. I then performed a similar analysis regarding the rotated camera configuration. One difference between the two setups is that the range of motion is reduced when the cameras are rotated, but the advantage or disadvantage is unclear because of targeted applications. The other difference is that with the fixed focal plane, the effective resolution decreases as the camera moves away from the focal plane, whereas it is constant when using a wall-eyed array.

In Chapter 6, will be about designing a camera array give a set of specifications. I will also further discuss these advantages or disadvantages between the rotated and wall-eyed configurations.

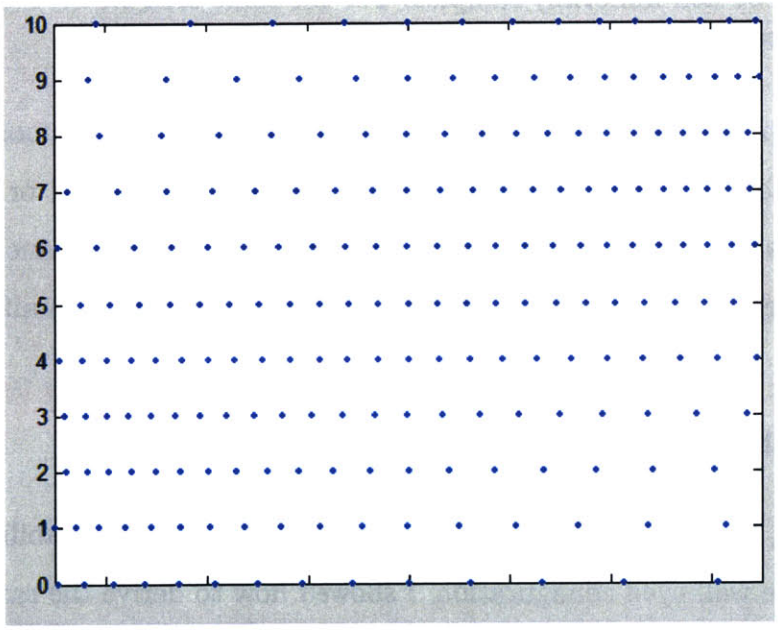


Figure 5-32: What the EPI would look like using rotated cameras constructed by projected samples from each camera onto a common plane. The blue dots are the actual rays sampled. Each camera line is non-uniform.

Chapter 6

Camera Array Configurations

One driving application for a real-time camera system is a virtual studio. In a virtual studio application, virtual cameras rendered from a light field captured by wall of cameras would replace conventional video cameras. The entire light field could be recorded for off-line processing, such as capturing a movie scene or a subject in front of a blue screen, or any number of virtual cameras could be rendered "live" in real time. The engineering problem that camera array builders need to solve is positioning and selecting the type of cameras such that they meet the desired specifications.

The goal of this chapter will be to answer the question of "Given a set of specifications, what cameras are needed and how to arrange them?" and will draw on the conclusions reached in Chapter 5. In that chapter, I answered the converse question by analyzing capabilities of a given light field in terms of virtual camera movement and resolution. The first part of this chapter will outline the steps required in designing the camera array. Then I will extend the discussion beyond a plane of cameras by capturing rays for immersive applications.

6.1 Camera Array Design

In this section, I will pull together the results from Chapter 5 by discussing the design of a camera array in a wall-eyed arrangement. This is the same type of arrangement used in Chapters 4 and 5. The section begins by describing how a camera array

should be specified for applications. Next I will describe how to build the array to meet the desired goals. Finally, I will briefly discuss non-walleyed camera arrays.

6.1.1 Application Parameters

To construct the camera array, a designer requires the desired virtual camera characteristics and the desired movement volume. Virtual camera characteristics (the type of camera to be rendered) can be defined in terms of the desired minimum and maximum field of views and the maximum image resolution.

The movement volume (where the camera will be positioned) can be defined in terms of the desired range of motion of the virtual camera along with the maximum rotation. Range of motion refers to the space where the virtual camera will be positioned. Since I am limiting the camera array to be wall-eyed and targeting mainly for the virtual studio application, the range of motion will be given relative to the camera plane (e.g., a wall).

6.1.2 Designing to Specifications

To meet the above goals, the designer needs to select the type of physical camera and to arrange them on the camera plane. There are many variables to consider. These include

- Camera Characteristics - resolution, field of view, etc.
- Camera Positions - dimensions and placement of the camera array
- Camera Orientation - how the cameras are oriented
- Cost

The application parameters affect the selection of cameras and the size of the camera array. Table 6.1 illustrates the relationship between application requirements from above and the array construction parameters.

Array Parameters	Dependencies
Physical Camera Field of View	Virtual Camera Field of View Range of Rotation
Physical Camera Resolution	Min Virtual Camera Field of View Max Virtual Camera Resolution Physical Camera Field of View
Array Dimension	Range of Motion Range of Rotation *or* Physical Camera Field of View
Number of Cameras	Scene Distance Resolution Range of Motion Limited by Cost

Table 6.1: This table illustrates the dependencies between selecting and building the camera array and the application conditions.

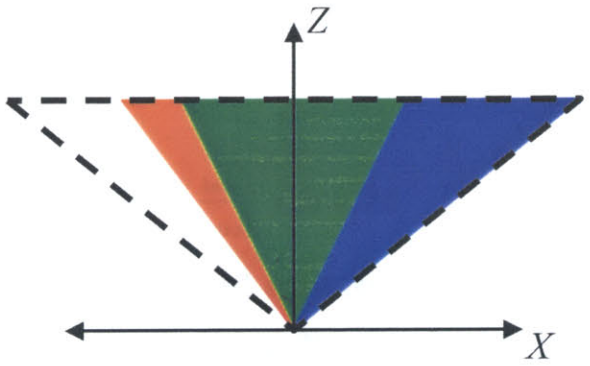


Figure 6-1: The field of view is the sum of the desired field of view (green) of the virtual camera and the maximum rotation (blue).

Field of View Section 5.2.1 illustrates that without rotation, the field of view of the virtual camera can be at most that of the virtual camera. Section 5.2.2 further demonstrates that the range of possible virtual camera rotations is limited by the field of view of the camera array. *Therefore, the field of view required is the sum of the desired maximum rendering field of view and the desired maximum rotation.* (Figure 6-1)

$$fov_{real} = fov_{desired} + 2 * max(|rotation_{desired}|) \tag{6.1}$$

Resolution From Chapter 5, the relationship between physical camera sampling or resolution (Δu) and virtual camera sampling (d_x) is the following

$$\Delta u = f_{real} * \begin{pmatrix} \frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta) + d_x\cos(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta) + d_x\sin(\theta)} \\ -\frac{-f_{virt}\tan\left(\frac{\phi}{2}\right)\cos(\theta) - f_{virt}\sin(\theta)}{-f_{virt}\tan\left(\frac{\phi}{2}\right)\sin(\theta) + f_{virt}\cos(\theta)} \end{pmatrix} \tag{6.2}$$

where d_x is determined from the minimum desired virtual field of view and the maximum desired virtual resolution:

$$d_x = \frac{2 * f_{virt} * \tan\left(\frac{1}{2}\min(\phi)\right)}{\max(res)} \tag{6.3}$$

If the virtual camera never rotates, to exactly sample the light field the physical cameras should have the same sampling rate as the virtual cameras. This occurs when $\theta = 0$ and

$$\Delta u = d_x * \frac{f_{real}}{f_{virt}} \tag{6.4}$$

Otherwise to prevent oversampling, solve Equation 6.2 for $-|max(\theta_{desired})|$, and to prevent undersampling solve for $|max(\theta_{desired})|$.

Array Dimensions The dimensions of the camera array refer to the distance between the first and last cameras in the horizontal and vertical directions. Of course the camera array could be built to cover an entire wall, but there are other reasons for knowing the required dimensions, such as only recording a part of the scene if the rest of the cameras only view the blue screen.

For example, Figure 6-2 is a 2D overhead view of a potential virtual studio. In the back of the studio there is background scenery or simply a blue screen. In the middle there are set pieces where the action takes place. The director could have fixed virtual cameras such as for interviews or monologues where the people are stationary. Or there could be live action segments with larger camera movements. Sometimes, such as for a weekly sitcom, the camera motion is unknown, but is limited to a certain

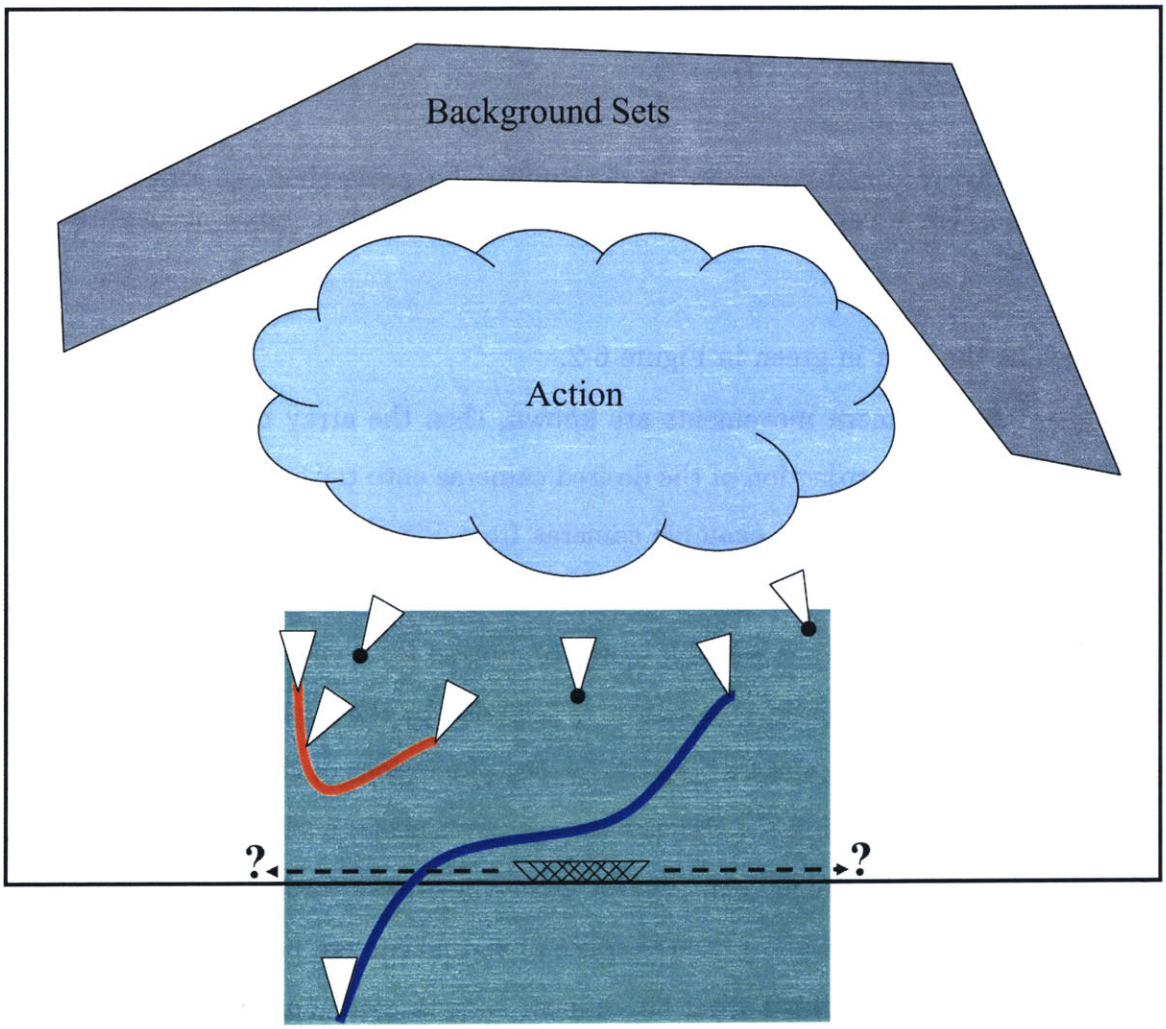


Figure 6-2: An overhead view of a sample virtual studio. In the background are sets or a blue screen. The action occurs in the foreground. There are desired camera movements or possibly a range of camera positions (green square). The unknown is the camera array dimensions.

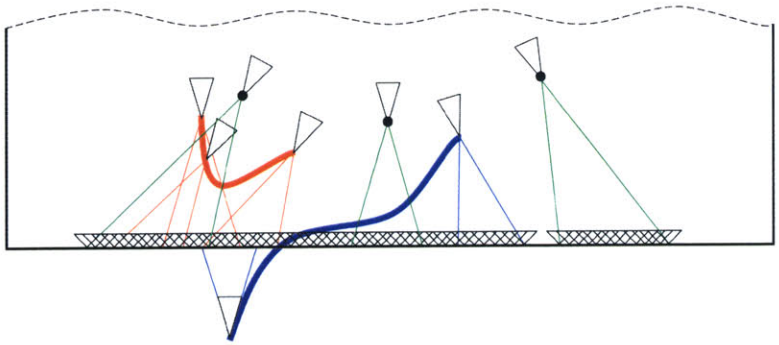


Figure 6-3: Array dimensions can be determined by projecting cameras onto the camera plane. For a camera path, the end cameras and any changes in orientation are projected.

area, such as the area in green in Figure 6-2.

If the desired camera movements are known, then the array dimensions can be determined from the projection of the desired cameras onto the camera plane. Figure 6-3 illustrates projecting the example cameras from Figure 6-2 onto the wall of the studio.

Section 5.2 illustrates the possible range of motion for a virtual camera rendered from a camera array. The range of motion is bounded by the rays of the first and last camera in the array. Therefore, alternatively, the dimensions of the camera array can be determined by bounding the desired camera movements, such as those in Figure 6-2, with the complete range of motion for a certain camera array.

For example, Figure 6-4 is a desired space of virtual camera positions. This space could also have been formed from the known camera positions (e.g., by building a convex hull). Now, given either a particular physical camera or calculating the proper field of view from earlier, the first camera position on the camera plane is where the extreme (first and last) camera rays enclose the desired space of virtual camera positions. The last camera position is determined in the same fashion.

Camera Density The number of cameras or the camera density is determined by the plenoptic sampling equations [5], effective resolution and virtual camera depth, and limited by the overall cost or budget.

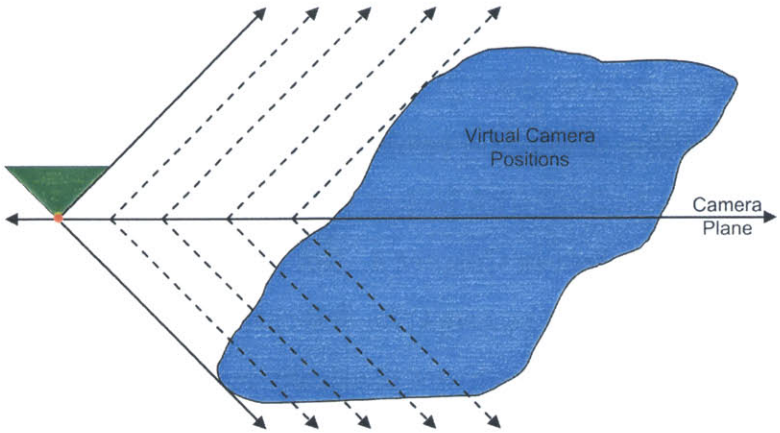


Figure 6-4: If the exact camera positions are unknown, but instead a range of motion is given, the analysis from Chapter 5 can be used to bound the area in green.

Plenoptic sampling [5], as discussed in Chapter 2, shows that the maximum camera spacing (Equation 6.5) and optical focal depth (Equation 6.6) is determined by the minimum and maximum scene depths.

$$\Delta ST_{\max} = \frac{2\Delta v}{f\left(\frac{1}{z_{\min}} - \frac{1}{z_{\max}}\right)} \tag{6.5}$$

$$\frac{1}{z_{opt}} = \frac{1}{2} * \left(\frac{1}{z_{\min}} + \frac{1}{z_{\max}}\right) \tag{6.6}$$

There is an additional factor to consider, which is the desired effective resolution. At some distance, the virtual camera resolution is dependent on the sampling of the camera plane. This occurs when the absolute value of the slope representing the virtual camera in the EPI is greater than $\frac{\Delta C}{\Delta U}$ (from Section 5.3.3). Assuming $z_{\max} = \infty$ and $f = 1$ then Equation 6.5 becomes

$$\frac{\Delta C}{\Delta U} \Rightarrow \frac{\Delta ST_{\max}}{\Delta v} = 2 * z_{\min} \tag{6.7}$$

This means that by following plenoptic sampling, effective resolution due to camera plane sampling becomes a concern at two times the minimum scene depth (slope is a function of the virtual camera's depth from Section 5.1.3).

Since the dimensions of the camera array are independent of the number or density

of cameras, it is up to the designer to determine the proper camera sampling. The straightforward approach is to determine camera spacing from plenoptic sampling and then adjust according to effective resolution and maximum camera depth.

There is of course a tradeoff with budget constraints. Cost was a primary motivator for the prototype system in Chapter 4. Not only will it dictate the number of cameras, but also the quality of cameras. As demonstrated in previous chapters, the calculated camera spacing is generally not physically realizable. Therefore, to achieve the best reconstruction, the environment should be sampled as densely as possible.

6.1.3 Wall-Eyed vs. Rotated Camera Arrangements

Cameras can be oriented either in the wall-eyed (infinite focal plane) or the rotated configuration [17] (fixed focal plane). In Chapter 5, I demonstrated that wall-eyed arrays allow virtual cameras to have a uniform effective resolution equal to the real cameras, whereas in the rotated case, the effective resolution of the virtual camera decreases away from the camera plane. I also showed the differences in the range of motion, with the rotated configuration having a smaller range. Therefore, in terms of virtual camera movement and virtual camera resolution the wall-eyed configuration is optimal.

The above conclusions, however, do not take into account target applications or intended acquisition subjects. For example, if the capture subject is an object (i.e., outside looking in light field), then in a wall-eyed setup, many rays will not even sample the object. Therefore, the "valid" camera definition from previous chapters does not apply.

Levoy and Hanrahan [17] analyzed the differences by transforming rays into line space. Line space illustrates the sampling range and density of the real cameras. Figure 6-5 shows that the wall-eyed configuration has a larger sampling area than the rotated case (with the same array dimensions and cameras). However, with a fixed focal plane placed at the depth of an object, the rotated cameras will sample more ray angles.

The conclusion is that in general, the wall-eyed configuration is optimal. Rotated

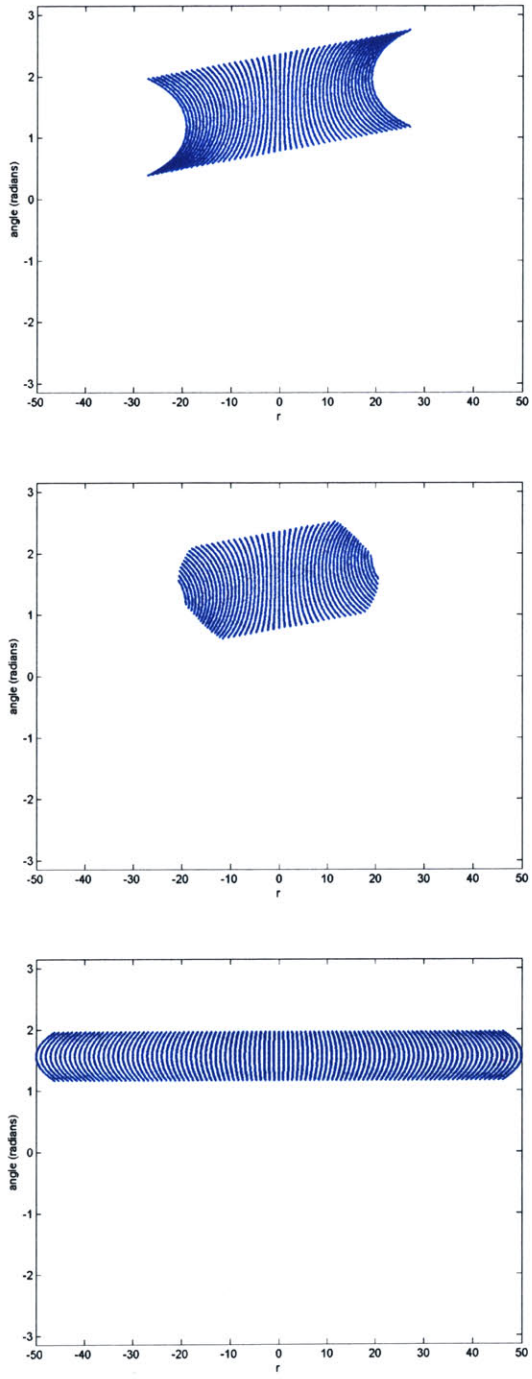


Figure 6-5: Rays are transformed into line space. In the rotated case (above), more ray angles are sampled, but they sample a smaller area. However, since some rays falls outside the fixed 2PP volume, the actual sampling density is reduced (middle).Using the same array dimension and cameras, the wall-eyed cameras (bottom) uniformly sample a larger area, but fewer ray angles.

cameras has a more scene specific application and is advantageous only when capturing an object at which the fixed focal plane can be placed, but only in the sense that there are fewer "wasted" rays.

6.1.4 A Virtual Studio Example

Figure 6-6 is a simulated demonstration of a virtual studio. The virtual camera is rendered at different points inside and outside the range of motion of a camera array. I also render images at different resolutions to demonstrate sampling of the light field.

In my analysis, I make no claims regarding the quality of the renderings or more accurately the sampling of the scene frequencies. For example, in Figure 6-6 the images are "blurrier" as the virtual camera moves toward the subject. This is due to the sampling of the scene (resolution) by the physical cameras. Choi et al. [5] also discusses the tradeoffs between resolution of the cameras and the scene frequencies. In all cases, the primary goal is the prevention of aliasing artifacts.

6.2 Immersive Environments

The 2PP is limited in that a light field slab encodes only a subset of all the rays in a volume (i.e., only rays entering and leaving the ST and UV planes) thus limiting virtual camera movement. Additional rays can be sampled by extending the ST and UV planes and changing the field of view of the cameras. However, the sampling gains are still limited; therefore, multiple light field slabs are required to completely capture an object or environment. Levoy and Hanrahan [17] primarily focused on using multiple slabs for capturing an object (inward looking light field). I will discuss outward looking light fields to capture an entire immersive environment.

With an inward looking light field, in order to completely capture an object, a ray leaving the object must intersect at least one slab. Six adjacent slabs (cube) are enough to completely capture an object as long as the distance from the slab to the object is such that the maximum angle of any ray to the slab is within the bounds of the field of view of the cameras. This guarantees that any virtual view of the object

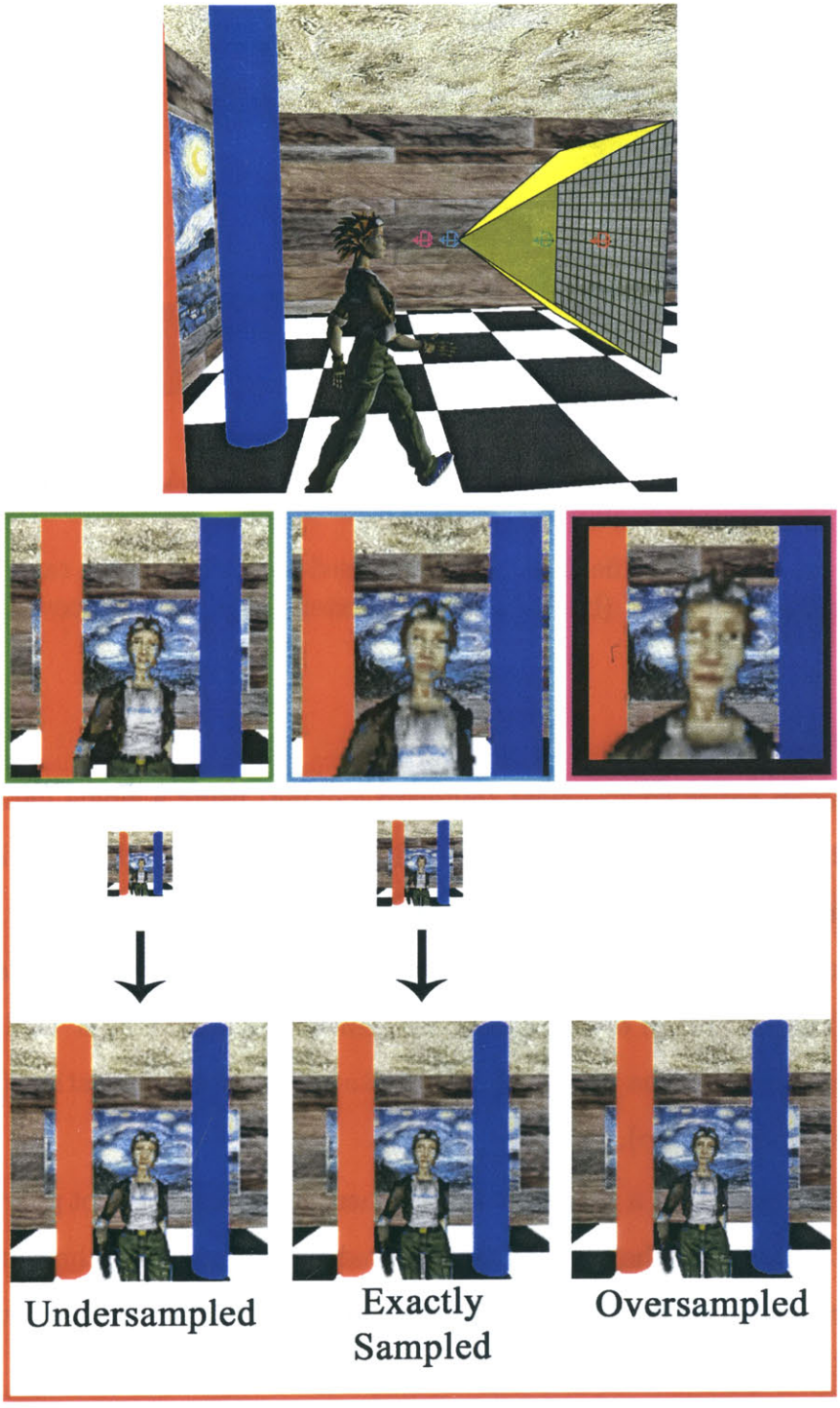
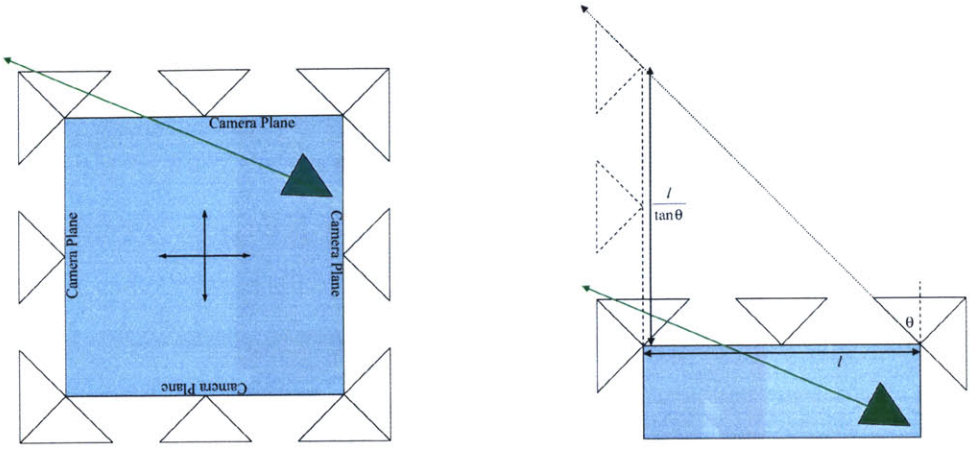


Figure 6-6: This is a simulated demonstration of a virtual studio. The rendered images are color coded to match the virtual cameras in the top image of the studio. The bottom images are renderings of the same virtual camera at different resolutions. This is to illustrate the sampling of the light field. The images have also been enlarged to match the highest resolution image.



(a) (b)

Figure 6-7: (a) If the light field slabs are arranged as a cube, some rays cannot be queried from the light field. (b) Slabs must be extended in order to cover all rays in the volume.

will produce the correct rays.

Capturing an entire environment (outward looking) using a single light field can rarely be achieved. However, multiple light fields can be used such that within some volume a virtual camera will render any view of the environment. Levoy and Hanrahan [17] uses a cross arrangement, whereas I arrange the light fields as six faces of a cube. The advantage of the cube arrangement is that the light field slabs can surround an object and be queried for accurate reflectance information (similar to using an environment map).

Figure 6-7a shows that a pure cube arrangement, however, does not guarantee that all rays passing through the box will be captured (unless of course the cameras have very wide field of views. One way to solve this problem is to extend the dimension of the light fields on each face.

For simplicity, I assume all the cameras are of the same field-of-view 2θ . If I want to move a virtual camera freely in a volume of l^3, I first create a bounding box with length l. To guarantee no ray passing through the bounding box is missing, I extend the slabs by length $l_{extend} = \frac{1}{\tan(\theta)}$. If $\theta > \frac{\pi}{4}$, then the lengthened light field slabs can capture all rays passing through the bounding box. In practice, cameras with smaller

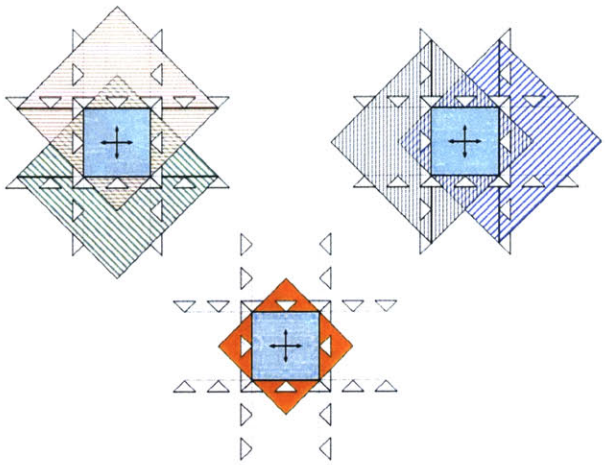


Figure 6-8: Knowing the range of motion for the slabs, the overlaps determine where all views exist. This area (brown) is actually greater than the desired area (blue).

field of views are preferred because it increases the UV resolution. Therefore, the minimum setup to allow free movement within the volume is $\theta = \frac{\pi}{4}$ with orthogonal slabs of length $3l$. (Figure 6-7b)

As Figure 6-8 shows, virtual cameras can actually be rendered beyond the desired volume. In fact, the cross arrangement is the optimal configuration. While valid virtual cameras can be rendered in these extra regions, the goal is to be able to render valid views of the environment (inside looking out). Figure 6-9 shows how objects outside of the defined space can pose problems during rendering.

In the discussion so far I have assumed synthetic light fields. Unfortunately, neither the cross nor the extended cube is physically realizable due to the camera planes obstructing the views of other planes. Using solely a simple cube with cameras of $\frac{\pi}{2}$ field of view, the only possible position where all rays can be rendered is at the exact center of the volume. In order to increase the range of motion, the field of view of the cameras must increase. However, physical cameras with a field of view greater than $\frac{\pi}{2}$ are generally not practical.

Other than increasing the field of view, the only possible solution is to rotate the camera planes, creating a convex polyhedron. However, as Figure 6-10 demonstrates, all rays can never captured under this arrangement either. Therefore, capturing

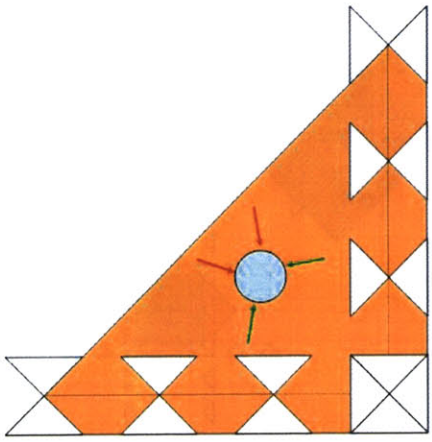


Figure 6-9: A cross-section of the above arrangement. Although all rays can be queried in this region, the object (blue circle) is only capture by two slabs. A virtual camera can only render half of this object. This also poses a problem for reflectance since only half of the object can be rendered.

synthetic environments can only be realized in a synthetic or static environment so that the array can be repositioned to capture multiple slabs.

6.3 Summary

This chapter focused on designing the arrangement of the cameras in the array. The first half demonstrated the steps needed to select the physical cameras, determine the dimensions of the array, and decide between the wall-eyed and rotated camera configuration. The second half of the chapter discussed using multiple arrays to capture an immersive environment using synthetic light fields. The next chapter will give additional application examples of a virtual studio.

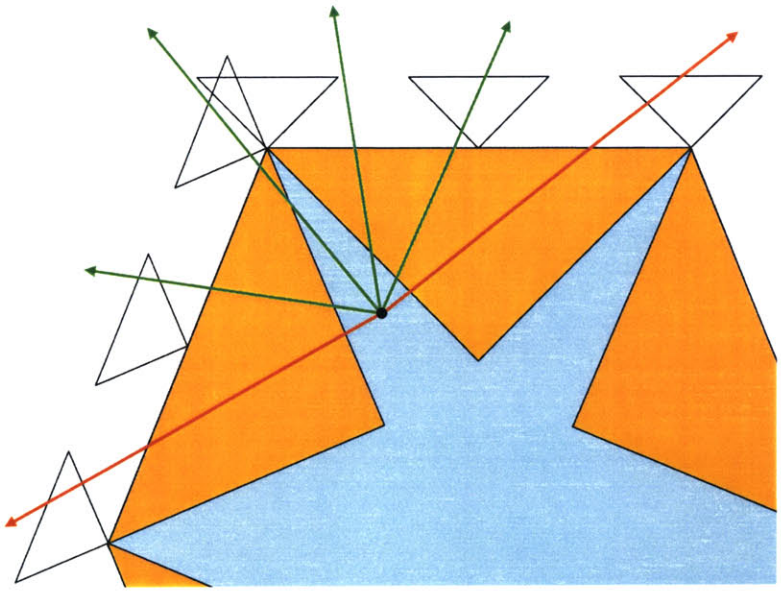


Figure 6-10: The result of rotating the camera planes to try to capture all rays. For the given position in the volume, the rays in red are not captured.

Chapter 7

Exploiting Dynamic Light Fields for Virtual Studio Effects

In Chapter 6, I described the process of building a camera array for a virtual studio. In this chapter I will describe various example applications using dynamic light fields captured by a real-time camera array. Some of the examples are rendered from data captured by the prototype system in Chapter 4 and others will be rendered using synthetic data.

[Note: Renderings will appear at the end of the chapter.]

7.1 Freeze Motion

Figure 7-3 is an example of the time-stopping effect introduced by Dayton Taylor and popularized in the movie *The Matrix.* This effect is only possible with multiple cameras. Images are rendered at different viewpoints from the same light field captured at a moment in time.

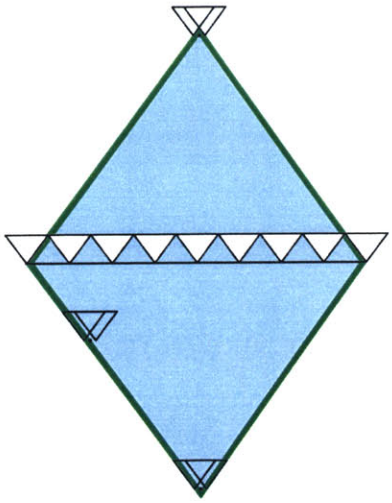


Figure 7-1: The area in green is the range of motion for a single camera. The area in blue is the range of motion for the stereo pair. The shapes are the same, but the smaller area takes into account the "true" position, which is in between the two cameras.

7.2 Stereo Images

Depending on the system design and hardware limitations, multiple virtual cameras can be rendered simultaneously. Therefore, stereo pairs can be generated for 3D viewing suitable for use on a head-mounted display. Figure 7-5 shows image pairs from a video sequence.

Determining the range of motion for a stereo pair is similar to that of a single virtual camera, except there are two centers of projection and parallel optical axes. I consider the position of the stereo pair as the midpoint between the two "eyes". Extreme rays can still be used to construct the range of motion, taking care to bound the motion to the true position. Figure 7-1 shows the difference between the range of motion for a single virtual camera and a stereo pair with the same fields of view.

7.3 Defined Camera Paths

7.3.1 Tracked Camera Motion

Figure 7-4 demonstrates that the light field can render video where virtual camera simulates a tracked camera. Tracked camera are camera motions where the camera moves along a rail or "track" on the ground. This effect is frequently used in motion picture productions. The disadvantage of using a tracked camera is the inflexibility of the camera path whereas the virtual camera can have any motion. Tracked cameras can also affect the lighting of the scene.

7.3.2 Robotic Platform

A superset of tracked cameras are motion controlled cameras on a robotic arm. The robotic arm can be pre-programmed to move on a defined path or controlled by an operator. The limitation of using the robotic arm is that the mechanics restrict the motion to a certain movement and a certain speed. For example, the Milo Motion Control System [8] has a maximum arm extension of 1 m, extension speed of 38 cm per second, track speed of 2 m per second, and camera rotation speed 120° per second. These limitations do not exist for a virtual camera in the virtual studio. Camera movement is confined only to the capabilities of the light field (Chapter 5). This means that it is possible to place the virtual camera in positions that are not natural for the robotic arm, and the virtual camera can move at any speed. There are also no safety concerns when using a camera array versus the robotic camera. Figure 7-6 demonstrates rendering a virtual camera on a pre-programmed camera path.

7.3.3 Vertigo Effect

The vertigo effect is a special effect where the field of view changes as the camera moves toward or away from an object. (Figure 7-7) The effect is to keep the object the same size in the frame while the background changes in size relative to the object. By knowing the capabilities of the light field, this effect can be planned and implemented

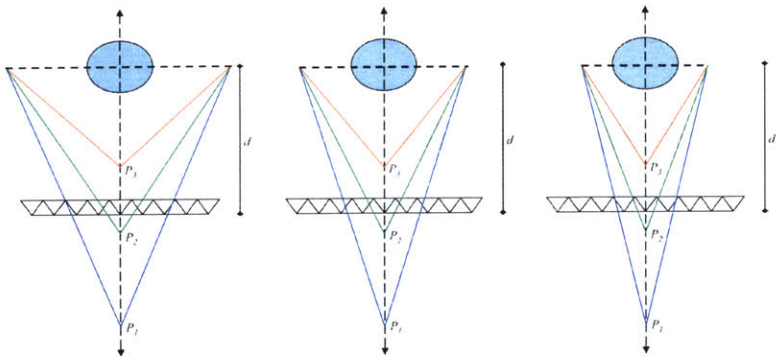


Figure 7-2: For the vertigo effect, the image plane (derived from a single camera) will determine the field of view for all camera positions.

with a virtual camera.

To plan the correct camera motion, first, the location of the target is required in order to fix the target size in the image. Then the field of view of a camera on a linear camera path will determine the field of view for every position on the same path by fixing an image plane at the depth of the object.

$$\phi = 2 * tan^{-1}\left(\frac{(d - P'_z)}{(d - P_z)} \tan\left(\frac{\phi'}{2}\right)\right) \tag{7.1}$$

Equation 7.1 gives the field of view for a camera at P_z given the depth of the target relative to the camera array and the field of view of camera at pre-determined position P'_z. Figure 7-2 demonstrates how the the field of view of all the camera in the vertigo effect are related to each other.

7.4 Mixing Light Fields

Film production frequently uses compositing where different elements (e.g., matte paintings, rendered images, blue screened subjects) are combined into a single frame. Figure 7-8 is a simple example where different light fields are rendered together. The first row shows different camera positions of a live light field. In the second row, the left half is a single stored light field and the right half is a different live light field. Since the camera array is fixed, the background rendering will be seamless.

Since environment size and position relative to the camera array can be determined, it is also possible to composite real world with synthetic light fields.

7.5 Real-Time Rendering

This section will utilize multiple light fields discussed in Chapter 6 for real-time rendering effects.

7.5.1 Immersive Environment

This example illustrates the use of multiple light fields to capture immersive environments. With six overlapping light field slabs surrounding a cube any view can be generated within a certain volume. Figure 7-9 demonstrates a virtual camera traversing a light field volume. With multiple light field volumes, virtual camera movement can be expanded.

7.5.2 Light Fields for Reflectance

Environment mapping is the most commonly used method for rendering approximate reflections interactively [2]. An implicit assumption in environment mapping is that all scene elements are located infinitely far away from the reflecting surface (equivalently, the reflector is modeled as a single point). When scene elements are relatively close to the reflectors, the results of environment mapping are inaccurate.

Up till now, I have been using light fields to render objects or scenes through a virtual camera. However, light fields can be used wherever light rays are needed, such as in reflectance [44]. To render accurate reflections, reflectors are placed within the volume surrounded by the multiple light field slabs. Each reflected ray is rendered by indexing into the correct light field. The gain in using light fields versus environment maps is the motion parallax observed in the reflections which is lacking when using only the environment map. Correct reflections are observed at any viewpoint in space, whereas reflector movement is constrained within the volume. This technique

is similar to the one proposed by [12]. (Figure 7-10) are some rendered examples.

7.5.3 Mixing Light Fields and Geometry

An extension to mixing light fields is mixing light fields with geometry. In the introduction, I mention that cameras that capture real scenes must be precisely aligned in order to composite with rendered effects. This is simplified with a camera array because novel camera paths can be rendered. Figures 7-11 and 7-12 are examples of combining light fields and geometry.

Figure 7-11 combines the immersive light field environment from above and geometric models. Reflections are rendered using the same light fields.

Figure 7-12 combines a video light field of a walking person and a geometric environment. The light field is a simulation of a real person captured in a virtual studio in front of a blue screen. The geometry and the light field are composited using alpha blending.

7.6 Summary

In this chapter, I have discussed and presented applications of the virtual studio and real-time light fields. The complete range of applications is not limited to those covered in this chapter. Most importantly, I describe possible uses of the virtual camera beyond the capabilities of conventional cameras.

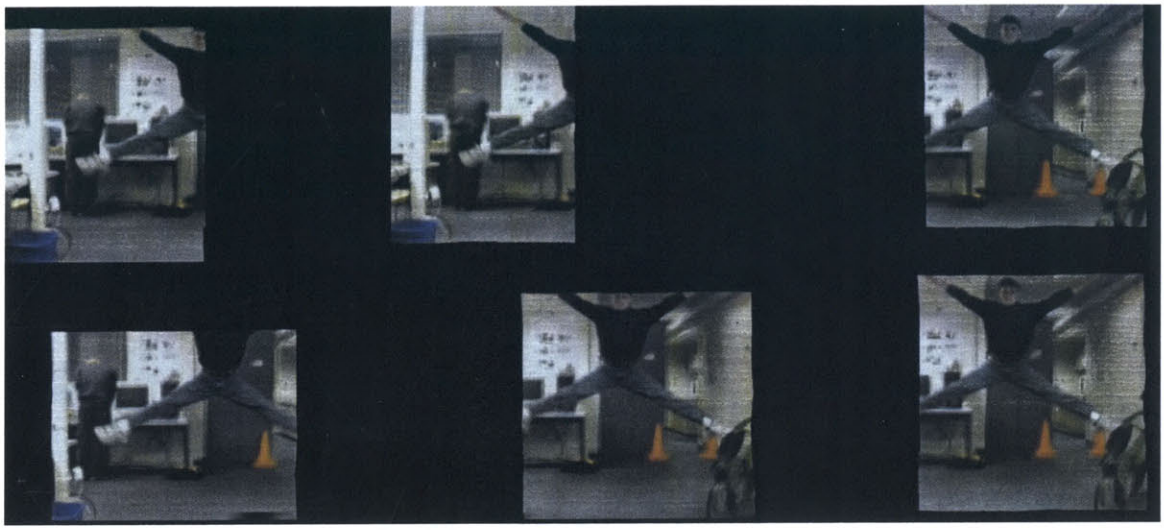


Figure 7-3: **Freeze Motion.** Different camera viewpoints of the light field. The virtual camera is moving in a circular motion.

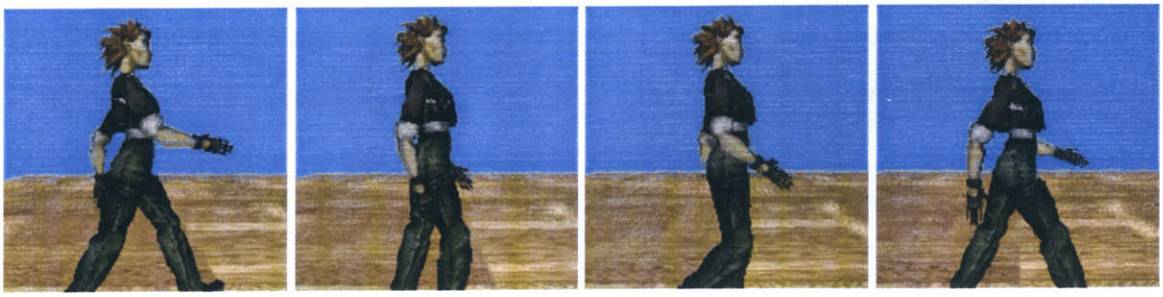


Figure 7-4: **Tracked Camera Motion.** Tracked camera from a synthetically generated light field video sequence.

Figure 7-5: **Stereo Images.** Sequence of stereo pairs from a video capture from the prototype camera array. 3D viewing is possible by cross-eyed fusing each image pair.

Figure 7-6: **Robotic Platform.** A sequence of images representing 3.5 seconds of video from a virtual camera. The virtual camera is on a pre-programmed path (the camera moves in and out of the scene). The intent is to emulate a motion controlled camera.

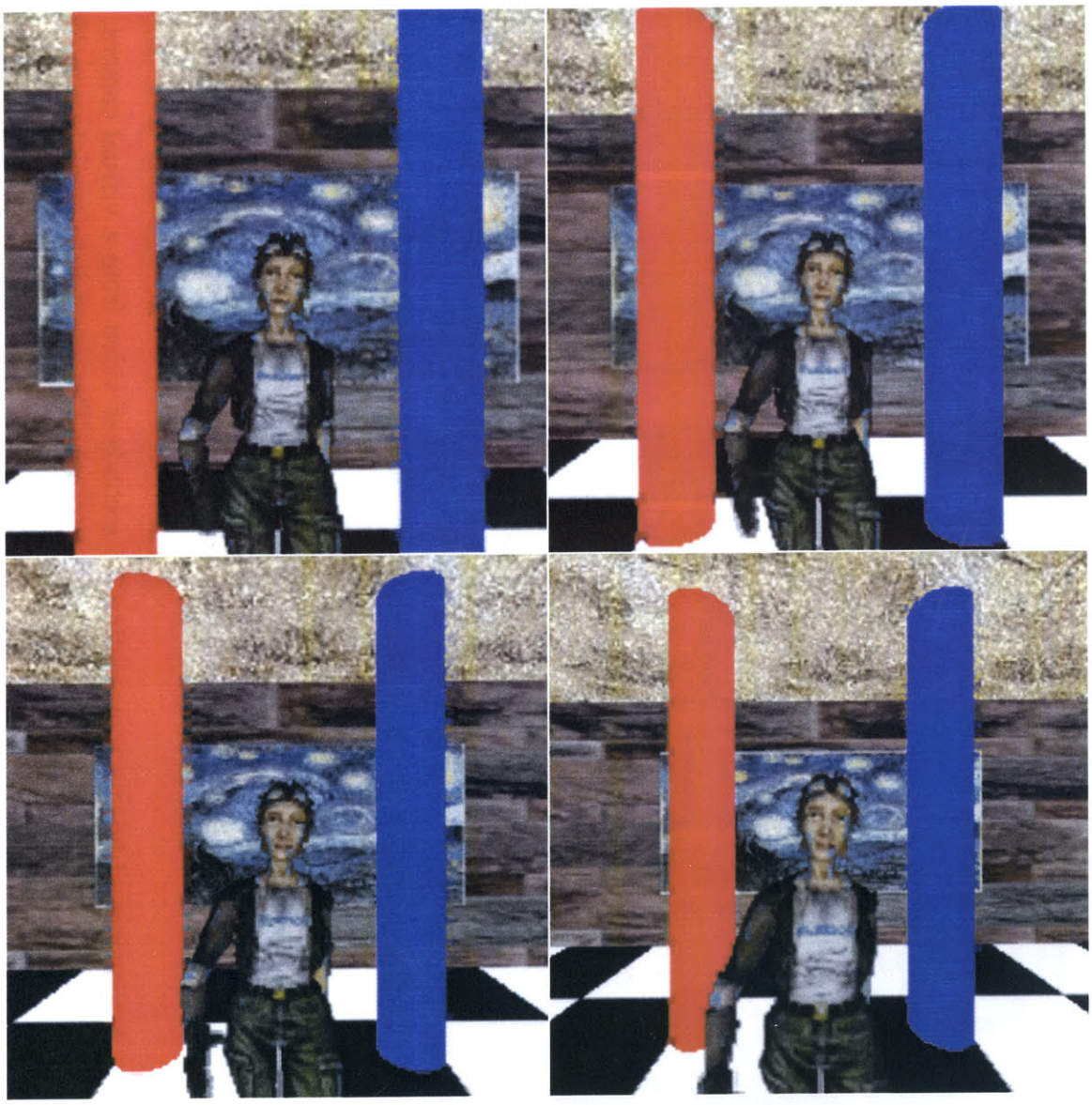


Figure 7-7: **Vertigo Effect.** As the camera translates forward, the field of view of the virtual camera widens. This sequence is also composed of two light fields. The foreground is the character and the background is the environment. Each light field is rendered at a different focal plane.

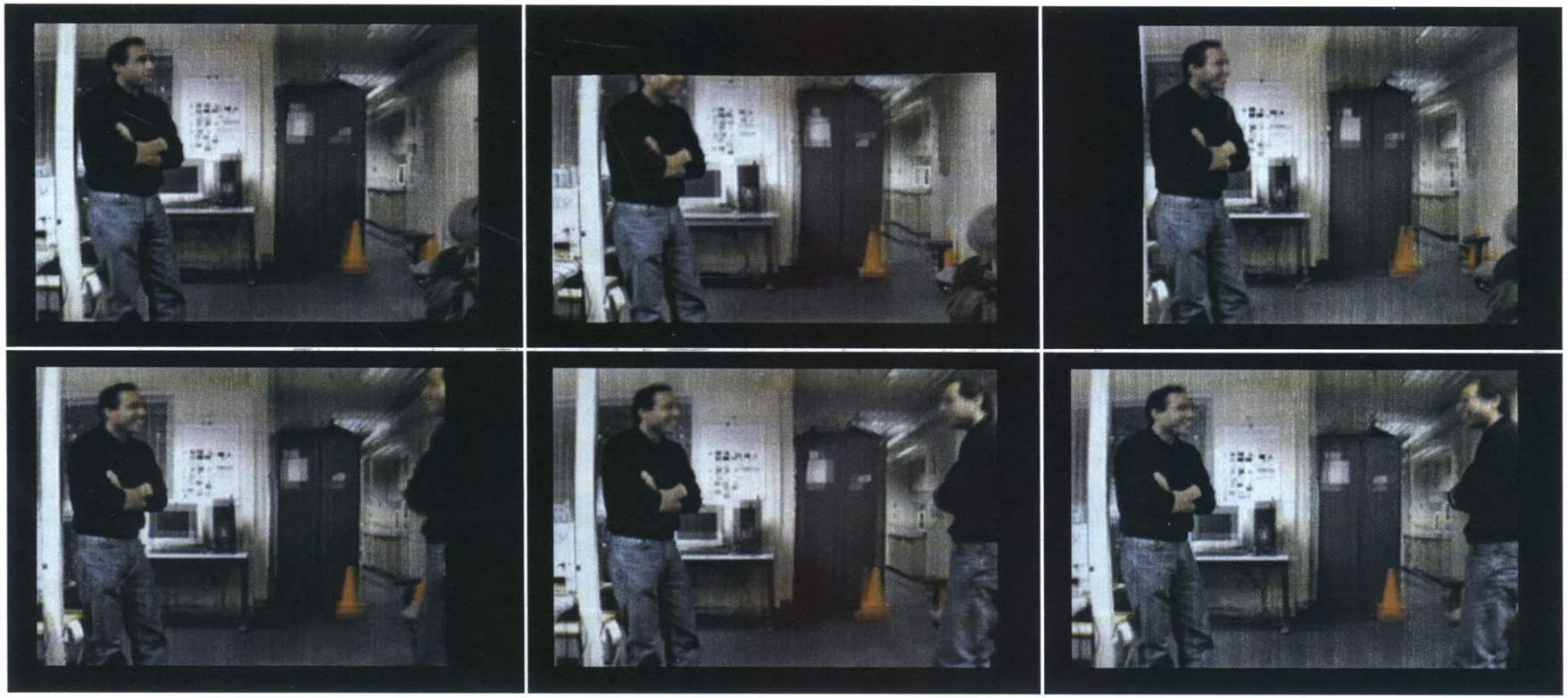
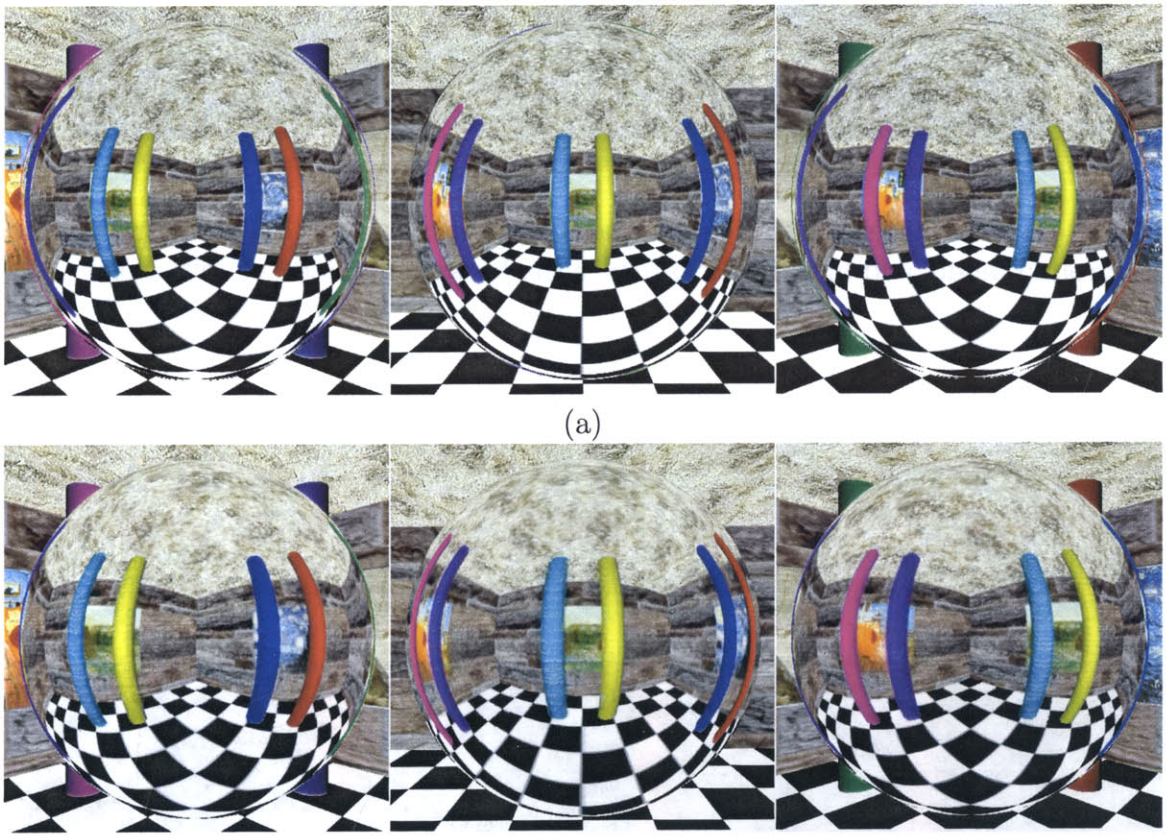


Figure 7-8: **Mixing Light Fields.** The top row are frames from a light field captured live. Cameras are at different viewpoints. The second row mixes a single stored light field (left half of image) with another live light field (right half). Notice that the background rendering, although rendered at different points in time, is continuous.

Figure 7-9: **Immersive Environment.** An immersive environment can be rendered using multiple light fields. These images are of virtual cameras at different positions and orientations. (Images are rendered using programmable graphics hardware. See Appendix A.)



(a)

(b)

Figure 7-10: **Light Fields for Reflectance.** Comparison between rendering reflectance using an environment map (a) or multiple light fields (b).

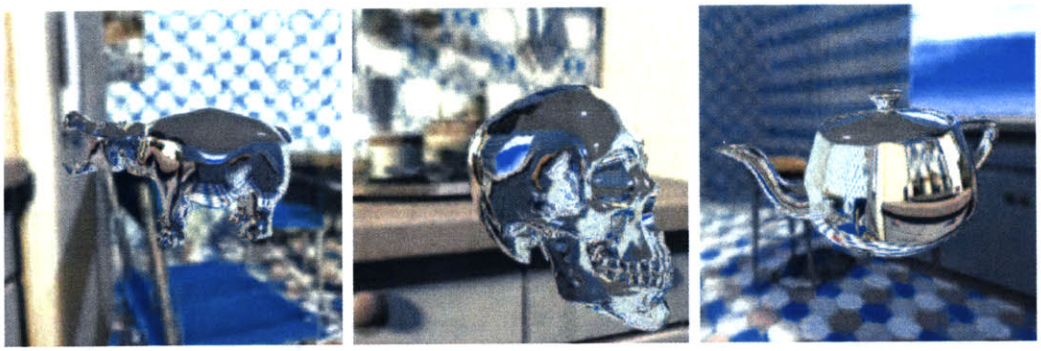


Figure 7-11: **Mixing Light Fields and Geometry.** A mix of geometry for the foreground model and light fields to render the background and the reflections.

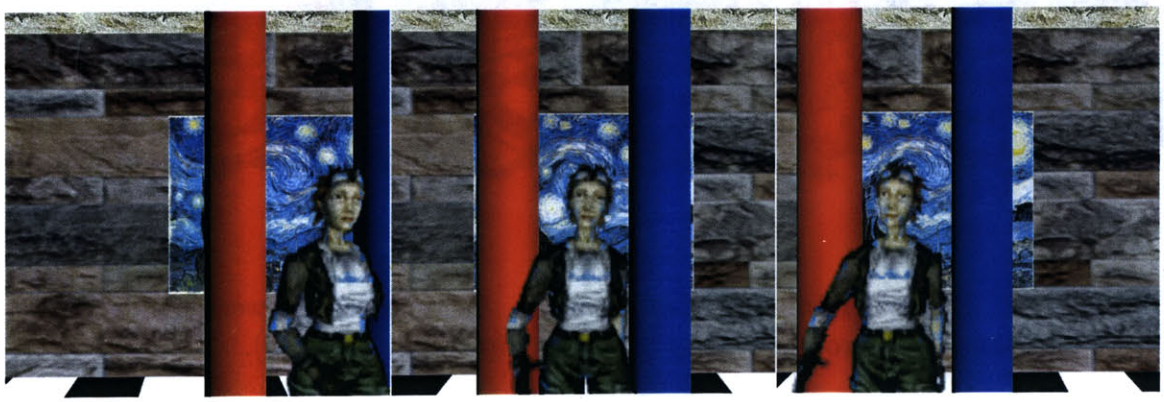


Figure 7-12: **Mixing Light Fields and Geometry.** Different views of a scene where the background is geometry and the person is a light fields.

Chapter 8

Conclusions and Future Work

In this chapter, I summarize the contributions presented in this dissertation from the prototype system to the analysis of the virtual camera capabilities within a light field camera array environment. I will also discuss areas of future work.

8.1 Conclusions

This dissertation describes the virtual studio application which replaces conventional cameras with synthetic cameras rendered from data collected by an array of cameras.

The virtual studio has a variety of applications, each with specific design goals. The primary applications include rendering live virtual cameras, recording light fields, and driving an autostereoscopic display. The two types of camera system designs that supports these applications are the "all-viewpoints" and the "finite-viewpoints" designs.

The all-viewpoints design transmits all video streams to the display device (e.g., an autostereoscopic display) in order to support all possible viewpoints. On the other hand, the finite-viewpoints design only transmits the fragments required to render requested views (i.e., live virtual cameras). However, due to the large bandwidth requirements, recording and playing back an entire light field is difficult. An "intelligent" camera proposed in Chapter 3 and 4 would partly solve the bandwidth problem by coupling rendering and storage to the physical cameras while reducing

overall rendering bandwidth.

Based on the finite-viewpoints design, I introduce a prototype system that renders live virtual cameras, meaning views are rendered from a dynamic light field. The prototype system consists of 64, densely spaced, firewire cameras in a wall-eyed arrangement. The cameras are attached to 4 rendering computers (16 cameras each). The rendering computers are connected to a single compositing computer over gigabit ethernet. The compositing computer can drive a display or further transmit rendered images (e.g., over the internet).

The prototype system uses a distributed rendering algorithm. The rendering computers asynchronously capture video streams from the cameras and render contributing image fragments to the final image. These fragments are transmitted to the compositing computer to be assembled and displayed. The system must also be photometrically, geometrically, and temporally calibrated.

Previously, the analysis of the light field has been limited to the sampling of the camera plane in the two plane parameterization [5]. I analyze the capabilities of virtual cameras in terms of the range of motion (position and orientation) and rendering resolution. Range of motion is defined by the dimensions of the camera array and the field of view of the cameras. Resolution refers to the effective resolution such that the light field is sampled so that no two rays sample the same hypervolume cell in the light field. When the image plane of the virtual camera is parallel to the camera plane, then the sampling is uniform, and the sampling rate of virtual camera should be the same as the real cameras in order to exactly sample the light field. Otherwise, under rotation the light field is non-uniformly sampled and the sampling rate must be determined such that the light field is either not oversampled or undersampled.

Using these results, I outline the steps required in order for a camera array designer to arrange the cameras to meet desired specifications. In doing so, I compared the wall-eyed (infinite focal plane) with the rotated (fixed focal plane) configuration. The only advantage of using the rotated configuration is that more angles are sampled (using the same array dimensions). In addition to a single plane, multiple camera

planes can be used to build immersive environments.

Finally, I present examples that take advantage of a dynamic light field. I demonstrate how the virtual camera can be used to simulate tracked or motion controlled cameras knowing the virtual camera capabilities. I also show examples of using dynamic light fields for special effects such as mixing light fields with other light fields or rendering light fields with geometry.

8.2 Future Work

The main challenge of light fields is handling the large bandwidth requirements. Fewer cameras, mean less data, but require geometry or improved reconstruction. The immediate area of further research would be the implementation of the "intelligent" camera and the ideal system that I propose in Section 3.4. This would address the problem of recording and playing back the light field by efficiently processing the data.

On the other hand, compression is still needed to reduce the size of the data. Levoy and Hanrahan [17] used vector quantization and LZW to achieve up to 118:1[1] compression. Magnor and Girod [18] achieve up to 2000:1 compression, but with loss of quality. In order to support rendering in graphics hardware, a few of the examples in Chapter 7 are compressed with DXT1 (6:1) compression although vector quantization could also be used.

However, current light field compression methods only compress a single light field. Ideally, dynamic light fields would include temporal compression. This is important because a single light field can be stored in memory, but the difficulty in handling light field videos is streaming the data out of storage at an interactive rate and keeping multiple light fields in memory. Standard video compression techniques such as motion flow and segmentation could be exploited, but there has been little research in this area.

Another further area of research is real-time rendering using light fields and pro-

[1] 4 slabs at 32x16x256x256 using 4D VQ (23:1) and LZW (5:1) to go from 402.7 MB to 3.4 MB

grammable graphics hardware. I have some examples of this in Chapter 7 with a complete description in Appendix A. However, I will highlight considerations relevant to rendering on current and future generations of programmable graphics hardware.

In Chapter 4, the prototype camera system used the texture mapping ability of commodity graphics hardware to accelerate rendering. For reflectance rendering, I used the programmable GPU of the current generation of graphics hardware to render rays on a per pixel basis. The advantages of using graphics hardware are that multiple computations (including filtering) and fetches are performed in parallel and that light fields can be used with traditional geometry rendering. The disadvantages are that the GPU is limited in programmability, has constraints on texture dimensions, and has limited addressable RAM.

An open area of research would therefore be developing a memory management strategy for light fields to control data flow between hard drive, CPU memory, and GPU memory. One method would be to use the rendering technique in Chapter 4 of only transferring the fragments that contribute to the final image. For example, knowing the camera parameters, a conservative estimate of the continuous slices of the light field required to render an image can easily be determined. Furthermore, in a multiple light field arrangement, it is known which light fields are required at any time. There is also a high degree of spatial coherency during rendering which can be exploited, meaning light fields can be used optimally in GPU caches.

In Chapter 6, I discussed camera arrangements to build immersive environments. I simulated this using a synthetic light fields. A simple extension would be to capture real world scenes. This would possibly require calibration across multiple camera arrays or across different light fields (one camera array successively repositioned).

Finally, another direction in the analysis of light fields and camera arrays is studying the tradeoffs between spatial camera resolution (ST sampling) to reduce aliasing vs. imager resolution to improve the visual fidelity. The approach would be to fix the total video bandwidth and determine the appropriate number of cameras and their resolution.

Appendix A

DirectX Shaders for Light Field Rendering

In the synthetic examples in Chapter 7, I used the programmable GPU to render the light fields and the reflections. The environment is rendered using six light fields (32x32x128x128). For efficiency I store multiple light fields in a single texture volume. Due to the limitations of the hardware I must render the image using two shader passes. I present the code here as an example of rendering light fields using graphics hardware.

First Pixel Shader Pass A quad representing the image plane is passed to the shader and rasterized. A ray from the eye to each pixel position is calculated. STUV coordinates are determined from this ray and reordered depending on which light field slab contributes to the pixel. The correct slab is determined by the same method as cube maps (largest coordinate value of the normalized ray determines the slab). STUV coordinates are written to a off screen render target along including a flag whether the ray is outside of all the slabs. The following is the shader for the first pass.

```
float   disparity : register(c0);
float4  offset : register(c2);
```

```
float4 UVscale : register(c3);
float SToffset : register(c4);
float3 eyepos : register(c5);

static const float4 UVcenter = float4(0.5, 0.5, 0.5, 0.5);

struct OutStruct {
    float4 STUV: COLOR0;
    float4 Level: COLOR1;
};

OutStruct main( float4 Vert : TEXCOORD0 )
{
    OutStruct output;

    float2 axisadj = float2(1.0, 1.0);
    float LFnum = 1;

    //-----Calculate Reflected Ray
    float3 Reflect = Vert.xyz - eyepos;

    //-----Reorder-----//
    float3 newReflect = Reflect;
    float3 newEyepos = eyepos;
    float3 aReflect = abs(Reflect);

    if(aReflect.x >= aReflect.y && aReflect.x >= aReflect.z)
    {
    //x axis
        newReflect = Reflect.zyx;
        newEyepos = eyepos.zyx;
        LFnum = 3;
        if(newReflect.z > 0)
            axisadj.x = -1;
    }
    else if(aReflect.y > aReflect.x && aReflect.y >= aReflect.z)
```

```
{
//y axis
    newReflect = Reflect.xzy;
    newEyepos = eyepos.xzy;
    LFnum = 5;
    if(newReflect.z > 0)
        axisadj.y = -1;
}
else
{
    if(newReflect.z < 0)
        axisadj.x = -1;
}

Reflect = newReflect;
eyepos = newEyepos;

if(Reflect.z > 0)
{
    eyepos.z = -eyepos.z;
    LFnum -= 1;
}

//-----Calculate STUV
Reflect = Reflect/abs(Reflect.z);

float4 STUV;
STUV.xy = ((eyepos.xy + ( SToffset + eyepos.z) * Reflect.xy)
                * axisadj) + offset;
STUV.zw = Reflect.xy * UVscale * axisadj + UVcenter;

output.STUV = STUV;

float4 Level = float4(0,0,disparity,0);
Level.x = LFnum;
if( STUV.x >= 0 && STUV.y >= 0 && STUV.x <= 15 && STUV.y <= 15)
```

```
        Level.y = 1;
    else
        Level.y = 0;

    output.Level = Level;

    return output;
}
```

Second Pixel Shader Pass In the second pass, a quad filling the viewport is passed to the shader. Pixels will correspond to STUV coordinates determined earlier. In this pass I calculate the correct texture coordinate and the correct volume slab. This is how I avoid conditional branches, where all branches are executed by the hardware. The texture fetch return a color and an alpha. Therefore, I can perform alpha blending with the framebuffer. To mix light fields I simply render multiple times. The following is the code for the second pass.

```
sampler3D LightField : register (s0);
sampler2D STUVTable : register (s6);
sampler2D LFTable : register (s7);

static const float  s = 128.0f / 1024.0f;
static const int dimface = 8;
static const int numgroup = 4;

float4 LFFetch( float4 ST, float4 UV, float LFnum, float disparity )
{
    //------Calculate closest integer ST coords
    int4 vST = floor(ST)+int4(0,0,1,1);

    //------Find interpolaion coeff
    float4 delta = frac(ST)-float4(0,0,1,1);

    //------Claculate UV with disparity offset
```

```
    float4 UVD = UV - disparity * delta;
    UVD = clamp(UVD, 1.0/256.0, 255.0/256.0);

    //-----Index Light Field
    float4 fcoord1 = vST%dimface + UVD;
    fcoord1 *= s;

    int4 a = vST/dimface;
    float4 fdepth = 1/128.0f * (a.yyww * numgroup + a.xzxz)
                  + 1/256.0f + LFnum * 32.0f/256.0f;

    float4 color1 = tex3D(LightField,
                          float3( fcoord1.x, fcoord1.y, fdepth.x ));
    float4 color2 = tex3D(LightField,
                          float3( fcoord1.z, fcoord1.y, fdepth.y ));
    float4 color3 = tex3D(LightField,
                          float3( fcoord1.x, fcoord1.w, fdepth.z ));
    float4 color4 = tex3D(LightField,
                          float3( fcoord1.z, fcoord1.w, fdepth.w ));


    //Interpolate
    float4 temp1 = lerp(color1, color2, delta.x);
    float4 temp2 = lerp(color3, color4, delta.x);

    return lerp(temp1, temp2, delta.y);
}

float4 main( float2 Tex : TEXCOORD0) : COLOR {
    float4 STUV = tex2D(STUVTable, Tex);
    float4 LFindex = tex2D(LFTable, Tex);

    float4 ST = float4(2,2,2,2)*STUV.xyxy;
    float4 UV = STUV.zwzw;

    float4 outcolor = LFFetch( ST, UV, LFindex.x, LFindex.z );
```

```
    if( LFindex.x > 0.5 || LFindex.y < 0.5)
        outcolor.w = 0.0f;
    return outcolor;
}
```

Bibliography

[1] Edwin A. Abbott. *Flatland : A Romance of Many Dimensions.* Little, Brown, and Co., 1899.

[2] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, 1976.

[3] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1:7–55, 1987.

[4] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 425–432. ACM Press / ACM SIGGRAPH, 2001. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[5] Jin-Xiang Chai, Shing-Chow Chan, Heung-Yeung Shum, and Xin Tong. Plenoptic sampling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318. ACM Press/Addison-Wesley Publishing Co., 2000.

[6] Shing-Chow Chan and Heung-Yeung Shum. A spectral analysis for light field rendering. In *7th IEEE Intn'l Conference on Image Processing (ICIP 2000)*, volume 2, pages 25–28, September 2000.

[7] Chris Mario Christoudias, Louis-Philippe Morency, and Trevor Darrell. Light field appearance manifolds. In *8th European Conference on Computer Vision*, pages 481–493, 2004.

[8] Mark Roberts Motion Control. Milo. http://www.mrmoco.com/.

[9] Joe Fordham. Brave new world. *Cinefex*, (98):15–33, July 2004.

[10] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[11] Xianfeng Gu, Steven J. Gortler, and Michael F. Cohen. Polyhedral geometry and the two-plane parameterization. In Julie Dorsey and Phillipp Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 1–12, New York, NY, 1997. Springer Wien.

[12] Wolfgang Heidrich, Hendrik Lensch, Michael F. Cohen, and Hans-Peter Seidel. Light field techniques for reflections and refractions. In *Rendering Techniques '99 (Proceedings of Eurographics Rendering Workshop)*, June 1999.

[13] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 297–306. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[14] Leonard McMillan Jr. *An Image-based Approach to Three-Dimensional Computer Graphics.* PhD thesis, University of North Carolina at Chapel Hill, 1997.

[15] Takeo Kanade. Carnegie mellon goes to the super bowl. http://www.ri.cmu.edu/events/sb35/tksuperbowl.html.

[16] Takeo Kanade, Peter Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE MultiMedia*, 4(1):34–47, – 1997.

[17] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[18] M. Magnor and B. Girod. Data compression for light field rendering. *IEEE Trans. Circuits and Systems for Video Technology*, 2000.

[19] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.

[20] Wojciech Matusik and Hanspeter Pfister. 3d tv: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *Proceedings of SIGGRAPH 2004*, 2004.

[21] Movia. http://www.movia.com/. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[22] Takeshi Naemura and Hiroshi Harashima. Real-time video-based rendering for augmented spatial communication. *Visual Communication and Image Processing 1999 (SPIE)*, pages 620–631, 1999.

[23] Takeshi Naemura, Junji Tago, and Hiroshi Harashima. Real-time video-based modeling and rendering of 3d scenes. *IEEE Computer Graphics and Applications*, pages 66–73, 2002.

[24] Harsh Nanda and Ross Cutler. Practical calibrations for a real-time digital omnidirectional camera. *CVPR 2001 Technical Sketch*, 2001.

[25] R. Oi, M. Magnor, and K. Aizawa. A solid-state, simultaneous wide angle-detailed view video surveillance camera. *Proc. Vision, Modeling, and Visualization (VMV-2003)*, Munich, Germany, pages 329–336, November 2003. Images

[26] Mixed Signal Products. *TSB15LV01 Video Signal Processor With IEEE-1394 Link Layer Controller: Data Manual.* Texas Instruments, 2000. Document No. SLLS425.

[27] Keith Quiring. Development of a tsb15lv01-based digital camera. Technical Report SLLA086, Texas Instruments: 1394 Peripheral Applications Team, August 2000.

[28] Matthew J. P. Regan, Gavin S. P. Miller, Steven M. Rubin, and Chris Kogelnik. A real-time low-latency hardware light-field renderer. *SIGGRAPH '99*, pages 287–290, 1999.

[29] Hideo Saito, Shigeyuki Baba, Makoto Kimura, Sundar Vedula, and Takeo Kanade. Appearance-based virtual view generation of temporally-varying events from multi-camera images in the 3d room. In *Proceedings of Second International Conference on 3-D Digital Imaging and Modeling*, pages 516 – 525, October 1999.

[30] Hartmut Schirmacher, Ming Li, and Hans-Peter Seidel. On-the-fly processing of generalized Lumigraphs. In *Proc. Eurographics 2001*, volume 20 of *Computer Graphics Forum*, pages C165–C173;C543. Eurographics Association, Blackwell, 2001.

[31] Peter-Pike Sloan, Michael F. Cohen, and Steven J. Gortler. Time critical lumigraph rendering. *1997 Symposium on Interactive 3D Graphics*, pages 17–24, 1997.

[32] J. Stewart, J. Yu, S. J. Gortler, and L. McMillan. A new reconstruction filter for undersampled light fields. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 150–156. Eurographics Association, 2003.

[33] Richard Szeliski and Sing Bing Kang. Recovering 3d shape and motion from image streams using nonlinear least squares. *JVCIP*, 5(1):10–28, 1994.

[34] Dayton V. Taylor. System for producing time-independent virtual camera movement in motion pictures and other media. US Patent 5659323, August 1997. Assignee: Digital Air, Inc.; filed December 21, 1994.

[35] 1394 Peripheral Applications Team. *1394 Digital Camera DCConnect: Software Manual.* Texas Instruments, December 2000. Document No. SLLU015.

[36] 1394 Peripheral Applications Team. *TSB15LV01 Digital Camera Evaluation Module: User's Guide.* Texas Instruments, 2000. Document No. SLLU014.

[37] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment–a modern synthesis. *Vision Algorithms: Theory and Practice*, pages 298–373, 1999.

[38] International Telecommuncation Union. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. Recommendation BT.601-5 (10/95).

[39] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Marc Levoy, and Mark Horowitz. High speed video using a dense camera array. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[40] Bennett Wilburn, Michal Smulski, Kevin Lee, and Mark A. Horowitz. The light field video camera. *Proceedings of Media Processors 2002, SPIE Electronic Imaging 2002*, 2002. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[41] Jason C. Yang, Matthew Everett, Chris Buehler, and Leonard McMillan. A real-time distributed light field camera. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 77–86. Eurographics Association, 2002.

[42] Ruigang Yang, Celso Kurashima, Andrew Nashel, Herman Towles, Anselmo Lastra, and Henry Fuchs. Creating adaptive views for group video teleconferencing

– an image-based approach. In *International Workshop on Immersive Telepresence (ITP 2002)*, December 2002. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[43] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 225. IEEE Computer Society, 2002.

[44] Jingyi Yu, Jason Yang, and Leonard McMillan. Real-time reflection mapping with parallax. In *(to) appear ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, 2005.

[45] Cha Zhang and Tsuhan Chen. A self-reconfigurable camera array. In *Eurographics Symposium on Rendering*, 2004. Images reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[46] Z. Zhang. A flexible new technique for camera calibration. Technical Report MSR-TR-98-71, Microsoft Research, 1998.