Dynamically Reparameterized Light Fields

by

Aaron Isaksen

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

November 2000

Author .....
Department of Electrical Engineering and Computer Science
November 28, 2000

Certified by .....
Leonard McMillan
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Dynamically Reparameterized Light Fields

by

Aaron Isaksen

Submitted to the Department of Electrical Engineering and Computer Science
on November 28, 2000, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This research further develops the light field and lumigraph image-based rendering methods and extends their utility. I present alternate parameterizations that permit 1) interactive rendering of moderately sampled light fields with significant, unknown depth variation and 2) low-cost, passive autostereoscopic viewing. Using a dynamic reparameterization, these techniques can be used to interactively render photographic effects such as variable focus and depth-of-field within a light field. The dynamic parameterization works independently of scene geometry and does not require actual or approximate geometry of the scene for focusing. I explore the frequency domain and ray-space aspects of dynamic reparameterization, and present an interactive rendering technique that takes advantage of today's commodity rendering hardware.

Thesis Supervisor: Leonard McMillan
Title: Assistant Professor

Acknowledgments

I would like to thank many people, agencies, and companies who helped with this research. Firstly, I would like to thank my advisor Professor Leonard McMillan, whose mentoring, leadership, and friendship was very important to me while I worked on this project. Both Professor McMillan and Professor Steven Gortler of Harvard University were instrumental in developing key sections of this thesis and the dynamically reparameterized light field work in general. Without their help, I would never have been able to complete this work. Also, their help in writing the Siggraph 2000 paper [13] and Siggraph 2000 presentation on this subject formed the textual basis for this thesis.

This work was funded through support from NTT, Hughes Research, and a NSF fellowship. In addition, Fresnel Technologies donated many lens arrays for the autostereoscopic displays.

Thanks to Ramy Sadek and Annie Sun Choi for illustrations, assistance, support, and proofreading of early versions of this thesis. Thanks to Robert W. Sumner for his LaTeX experience and to Chris Buehler for many helpful light field discussions and DirectX help. Thanks to Neil Alexander for the "Alexander Bay" tree scene and to Christopher Carlson for permission to use Bumpé in our scenes.

I would like to thank Annie Sun Choi, Robert W. Sumner, Benjamin Gross, Jeffery Narvid, the MIT LCS Computer Graphics Group, and my family for their emotional support and friendship while I worked on this research. Their support made it possible to keep going when things got especially difficult.

Once again, I would like to thank Professor Leonard McMillian, because he most of all made this work possible.

Contents

List of Figures

Chapter 1

Introduction

Traditionally, to render an image, one models a scene as geometry to some level of detail and then performs a simulation which calculates how light reacts with that scene. The quality, realism, and rendering time of the resulting image is directly related to the modeling and simulation process. More complex modeling and simulation leads to higher quality images; however, they also lead to longer rendering times. Even with today's most advanced computers running today's most powerful rendering algorithms, it is still fairly easy to distinguish between a synthetic photograph of a scene and an actual photograph of that same scene.

In recent years, a new approach to computer graphics has been developing: image-based rendering. Instead of simulating a scene using some approximate physical model, novel images are created through the process of signal reconstruction. Starting with a database of source images, a new image is constructed by querying the database for information about the scene. Typically, this produces higher quality, photorealistic imagery at much faster rates than simulation. Especially when the database of source images is composed of a series of photographs, the output images can appear with as high quality as the source images.

There are a variety of image-based rendering algorithms. Some algorithms require some geometry of the scene while others try to limit the amount of a priori knowledge of the scene. There is a wide range of flexibility in the reconstruction process as well: some algorithms allow translation, others only allow rotation. Often, image-based

rendering algorithms are closely related to computer vision algorithms, as computing structure from the collection of images is one way to aid the reconstruction of a new image, especially when translating through the scene. Another major factor in the flexibility, speed, and quality of reconstruction is the parameterization of the database that stores the reference images.

However, rendering is only part of the problem as there must be a image-based modeling step to supply input to the renderer. Some rendering systems require a coarse geometric model of the scene in addition to the images. Because of this requirement, people may be forced to use fragile vision algorithms or use synthetic models where their desire may be to use actual imagery. Some algorithms reduce the geometry requirement by greatly increasing the size of the ray database or by imposing restrictions on the movement of the virtual camera.

The light field [14] and lumigraph [7] rendering methods are two similar algorithms that synthesize novel images from a database of reference images. In these systems, rays of light are stored, indexed, and queried using a two-parallel plane parameterization [8]. Novel images exhibiting view-dependent shading effects are synthesized from this ray database by querying it for each ray needed to construct a desired view. The two-parallel plane parameterization was chosen because it allows very quick access and simplifies ray reconstruction when a sample is not available in the database. In addition, the two-parallel plane parameterization has a reasonably uniform sampling density that effectively covers the scene of interest. Furthermore, at very high sampling rates, the modeling step can be ignored.

Several shortcomings of the light field and lumigraph methods are addressed in this thesis. At low to moderate sampling rates, a light field is only suitable for storing scenes with an approximately constant depth. A lumigraph uses depth-correction to reconstruct scenes with greater depth variation. However, it requires an approximate geometry of the scene which may be hard to obtain. Both systems exhibit static focus because they only produce a single reconstruction for a given queried ray. Thus, the pose and focal length of the desired view uniquely determine the image that is synthesized.

This thesis shows that a dynamic reparameterization of the light field ray database can improve the quality and enhance the flexibility of image reconstruction from light field representations.

My goal is to represent light fields with wide variations in depth, without requiring geometry. This requires a more flexible parameterization of the ray database, based on a general mathematical formulation for a planar data camera array. To render novel views, my parameterization uses a generalized depth-correction based on focal surfaces. Because of the additional degrees of freedom expressed in the focal surfaces, my system interactively renders images with dynamic photographic effects, such as depth-of-field and apparent focus. The presented dynamic reparameterization is as efficient as the static lumigraph and light field parameterizations, but permits more flexibility at almost no cost. To enable this additional flexibility, I do not perform aperture filtering as presented in [14], because aperture filtering imposes a narrow depth of field on typical scenes. I present a frequency domain analysis justifying this departure in Chapter 6.

Furthermore, my reparameterization techniques allow the creation of directly-viewable light fields which are passively autostereoscopic. By using a fly's-eye lens array attached to a flat display surface, the computation for synthesizing a novel view is solved directly by the optics of the display device. This three-dimensional display, based on integral photography [17, 23], requires no eye-tracking or special hardware attached to a viewer, and it can be viewed by multiple viewers simultaneously under variable lighting conditions.

In this thesis, I first describe pertinent background information, as well as the relevant previous work in this area of image-based rendering (Chapter 2). Next, I discuss the dynamic reparameterization approach to light field processing (Chapter 3) and explain of how the variable parameters of the system affect reconstruction of the light field (Chapter 4). It is instructive to look at dynamic light field reparameterization in other domains: I explore the effects of reparameterization in ray space (Chapter 5) and in the frequency domain (Chapter 6). I then discuss various methods to render dynamically reparameterized light fields (Chapter 7) as well as ways to display them

in three-dimensions to multiple viewers (Chapter 8). I describe the methods used to capture light fields as well as various particulars about my light fields (Chapter 9). Finally, I conclude with future work (Chapter 10) and various code samples for reproducing some key figures in the thesis (Appendix A).

Chapter 2

Background and Previous Work

This thesis examines the sampling and reconstruction methods used in light field rendering. Previous researchers have looked at methods to create light fields, as well as ways to deal with the high sampling rates required for light field representation. In this chapter, I discuss the major issues that arise when rendering from a ray database and what previous researchers have done to avoid these problems.

In recent years, researchers have been looking at ways to create images through the process of reconstruction. Chen and Williams were one of the first teams to look at image-synthesis-by-image-reconstruction; their system created new views by depth-based morphing between neighboring images [4]. However, this type of system does not allow the user to move far from where the images were taken. McMillan and Bishop advanced image-based rendering by posing the reconstruction process as querying a database of rays [15]. They suggested a 5-D plenoptic function to store and query rays of light that pass through a scene. Because the images were thought of as individual bundles of ray, they could move through a scene with more flexibility then previously available. However, because the system was 5-D, many more rays than necessary were represented in regions of open space.

Soon after, Gortler, Grzeszczuk, Szeliski, and Cohen published "The Lumigraph" [7] and Levoy and Hanrahan published "Light Field Rendering" [14]. These papers similarly reduced the 5-D plenoptic function to a special case of 4-D. In the 4-D case, the user can freely move around unobstructed space, as long as she doesn't pass in

front of any objects. This reduction of dimensions greatly increases the usability and storage of the ray database. In addition, the 4-D ray database can be captured simply by moving a camera along a 2-D manifold.

Researchers have been looking at the ray database model of image-based rendering for some time, as it is a quite effective way to represent a scene. A continuous representation of a ray database is sufficient for generating any desired ray. In such a system, every ray is reconstructed exactly from the ray database with a simple query. However, continuous databases are impractical or unattainable for all but the most trivial cases. In practice, one must work with finite representations in the form of discretely-sampled ray databases. In a finite representation, every ray is not present in the database, so the queried ray must be approximated or reconstructed from samples in the database.

As with any sampling of a continuous signal, the issues of choosing an appropriate initial sampling density and defining a method for reconstructing the continuous signal are crucial factors in effectively representing the original signal. In the context of light fields and lumigraphs, researchers have explored various parameterizations and methods to facilitate better sampling and rendering. Camahort, Lerios, and Fussell used a more uniform sampling, based on rays that pass through a uniformly subdivided sphere, for multi-resolution light field rendering [2]. For systems with limited texture memories, Sloan, Cohen, and Gortler developed a rendering method for high frame rates that capitalizes on caching and rendering from dynamic non-rectangular sampling on the entrance plane [21]. For synthetic scenes, Halle looked at rendering methods that capitalize on the redundant nature of light field data [10]. Given a minimum depth and a maximum depth, Chai, Tong, Chan, and Shum derived the minimum sampling rate for alias-free reconstruction with and without depth information [3]. Holographic stereograms are closely related to light fields; Halle explored minimum sampling rates for a 3-D light field (where the exit surface is two dimensional, but the entrance surface is only one dimensional) in the context of stereograms [11]. Shum and He presented a cylindrical parameterization to decrease the dimensionality of the light field, giving up vertical parallax and the ability to

translate into the scene [20].

The choice of a ray database parameterization also affects the reconstruction methods that are used in synthesizing desired views. It is well understood for images that even with properly-sampled dataset, a poor reconstruction filter can introduce post-aliasing artifacts into the result.

The standard two-plane parameterization of a ray database has a substantial impact on the choice of reconstruction filters. In the original light field system, a ray is parameterized by a predetermined entrance plane and exit plane (also referred to as the *st* and *uv* planes using lumigraph terminology). Figure 2-1 shows a typical sparse sampling on the *st* plane and three possible exit planes, uv_1, uv_2, and uv_3. To reconstruct a desired ray r which intersects the entrance plane at (s, t) and the exit plane at (u, v), a renderer combines samples with nearby (s, t) and (u, v) values. However, only a standard light field parameterized using exit plane uv_2 gives a satisfactory reconstruction. This is because the plane uv_2 well approximates the geometry of the scene, while uv_1 and uv_3 do not.

The original light field system addresses this reconstruction problem by aperture filtering the ray database. This effectively blurs information in the scene that is not near the focal surface; it is quite similar to depth-of-field in a traditional camera with a wide aperture. In the vocabulary of signal processing, aperture filtering band-limits the ray database with a low-pass prefilter. This removes high-frequency data that is not reconstructed from a given sampling without aliasing. In Figure 2-1, aperture filtering on a ray database parameterized by either the exit plane uv_1 or uv_3 would store only a blurred version of the scene. Thus, any synthesized view of the scene appears defocused, "introducing blurriness due to depth of field."[14] No reconstruction filter is able to reconstruct the object adequately, because the original data has been band-limited. This method breaks down in scenes which are not sufficiently approximated by a single, fixed exit plane. In order to produce light fields that capture the full depth range of a deep scene without noticeable defocusing, the original light field system would require impractically large sampling rates.

Additionally, it can be difficult to create an aperture prefiltered ray database. The

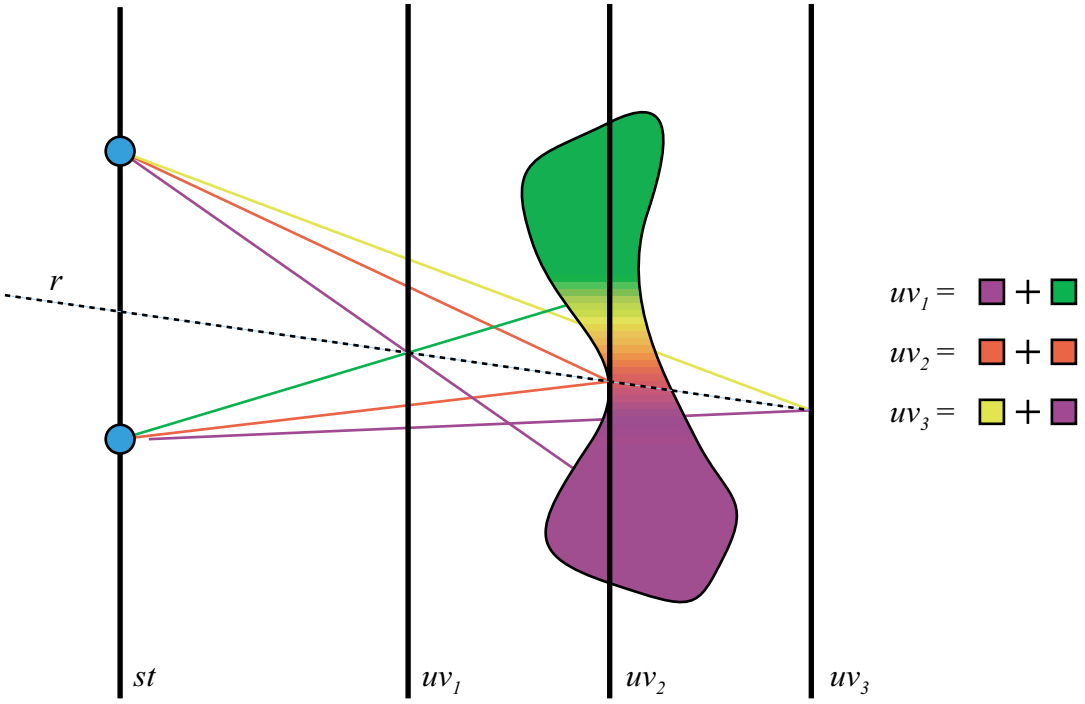


Figure 2-1: The parameterization of the exit plane, or uv plane, affects the reconstruction of a desired ray r. Here, the light field would be best parameterized using the uv_2 exit plane.

process of aperture prefiltering requires the high frequency data to be thrown away. To create a prefiltered ray database of a synthetic scene, a distributed ray-tracer [5] with a finite aperture renderer could be used to render only the spatially varying low frequency data for each ray. Also, a standard ray tracer could be used, and many rays could be averaged together to create a single ray. This averaging process would be the low-pass filter. In the ray-tracing case, one is actually rendering a high-resolution light field and then throwing away much of the data. Thus, this is effectively creating a lossy compression scheme for high-resolution light fields. If one wanted to aperture prefilter a light field captured with a camera system, a similar approach would likely be taken. Or, one could open the aperture so wide that the aperture size was equal to camera spacing. The system would have to take high resolution samples on the entrance plane: in other words, one would have to take a "very dense spacing of views. [Then, one] can approximate the required anti-aliasing by averaging together some number of adjacent views, thereby creating a *synthetic aperture*."[14] One is in effect using a form of lossy compression. The major drawback is that one has to acquire much more data than one is actually able to use. In fact, Levoy and Hanrahan did "not currently do" aperture prefiltering on their photographic light fields because of the difficulties in capturing the extra data and the limitation due to the small aperture sizes available for real cameras. Finally, if the camera had a very large physical aperture of diameter d, the light field entrance plane could be sampled every d units. However, lenses with large diameters can be very expensive, hard to obtain, and will likely have optical distortions.

To avoid aperture prefiltering, the lumigraph system is able to reconstruct deep scenes stored at practical sampling rates by using depth-correction. In this process, the exit plane intersection coordinates (u, v) of each desired ray r are mapped to new coordinates (u', v') to produce an improved ray reconstruction. This mapping requires an approximate depth per ray, which can be efficiently stored as a polygonal model of the scene. If the geometry correctly approximates the scene, the reconstructed images always appear in focus. The approximate geometry requirement imposes some constraints on the types of light fields that can be captured. Geometry

is readily available for synthetic light fields, but acquiring geometry is difficult for photographically-acquired ray databases.

Acquiring geometry can be difficult process. In the Lumigraph paper, the authors describe a method that captures a light field with a hand-held video camera [7]. First, the authors carefully calibrated the video camera for intrinsic parameters. Then, the object to capture was placed on a special stage with markers. The camera was moved around while filming the scene, and the markers were used to estimate the pose. A foreground/background segmentation removed the markers and stage from the scene; the silhouettes from the foreground and the pose estimation from each frame were used to create a volumetric model of the object. Finally, because the camera was not necessarily moved on a plane, the data was rebinned into a two-parallel plane parameterization before rendering. Using this method, the authors were only able to capture a small number of small object-centered light fields.

Both the light field and lumigraph systems are fixed-focus systems. That is, they always produce the same result for a given geometric ray r. This is unlike a physical lens system which exhibits different behaviors depending on the focus setting and aperture size. In addition to proper reconstruction of a novel view, I would like to produce photographic effects such as variable focus and depth-of-field at interactive rendering rates. Systems have been built to render these types of lens effects using light fields, but this work was designed only for synthetic scenes where an entire light field is rendered for each desired image [12].

In this chapter, I have discussed previous work in light field rendering and explained the problems that often arise in systems that render from a ray database. Typically, light fields require very high sampling rates, and techniques must be developed when capturing, sampling, and reconstructing novel images from such a database. I have discussed what previous researchers have developed for this problem.

Chapter 3

Focal Surface Parameterization

This chapter describes the mathematical framework of a dynamically-reparameterized-light-field-rendering-system [13]. First, I describe a formulation for the structures that make up the parameterization. Then, I show how the parameterization is used to reconstruct a ray at run time.

My dynamically reparameterized light field (DRLF) parameterization of ray databases is analogous to a two-dimensional array of pinhole cameras treated as a single optical system with a synthetic aperture. Each constituent pinhole camera captures a focused image, and the camera array acts as a discrete aperture in the image formation process. By using an arbitrary focal surface, one establishes correspondences between the rays from different pinhole cameras.

In the two-parallel-plane ray database parameterization there is an entrance plane, with parameters (s, t) and an exit plane with parameters (u, v), where each ray r is uniquely determined by the 4-tuple (s, t, u, v).

The DRLF parameterization is best described in terms of a camera surface, a 2-D array of data cameras and images, and a focal surface (see Figure 3-1). The camera surface C, parameterized by coordinates (s, t), is identical in function to the entrance plane of the standard parameterization. Each data camera $D_{s,t}$ represents a captured image taken from a given grid point (s, t) on the camera surface. Each $D_{s,t}$ has a unique orientation and internal calibration, although I typically capture the light field using a common orientation and intrinsic parameters for the data cameras.

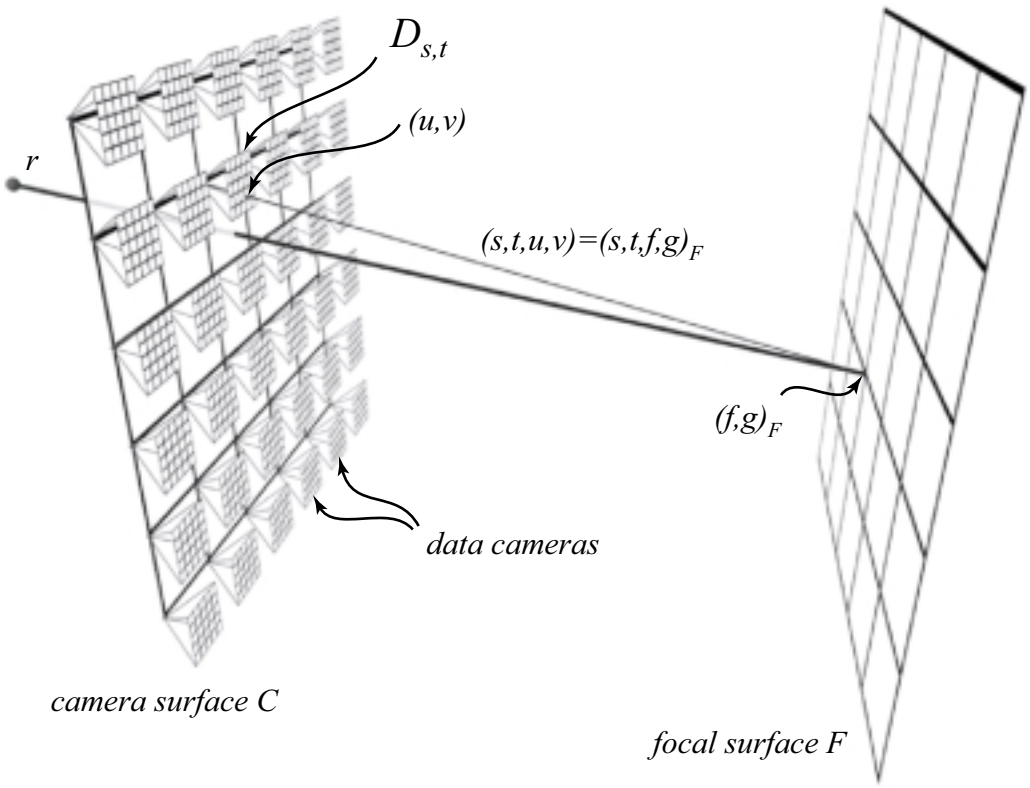


Figure 3-1: My parameterization uses a camera surface C, a collection of data cameras $D_{s,t}$, and a dynamic focal surface F. Each ray (s, t, u, v) intersects the focal surface F at $(f, g)_F$ and is therefore named $(s, t, f, g)_F$.

Each pixel is indexed in the data camera images using image coordinates (u, v), and each pixel (u, v) of a data camera $D_{s,t}$ is indexed as a ray $r = (s, t, u, v)$. Samples in the ray database exist for values of (s, t) where there is a data camera $D_{s,t}$. The focal surface F is a dynamic two-dimensional manifold parameterized by coordinates $(f, g)_F$. Because the focal surface may change dynamically, I subscript the coordinates to describe which focal surface is being referred to. Each ray (s, t, u, v) also intersects the focal surface F, and thus has an alternate naming $(s, t, f, g)_F$.

Each data camera $D_{s,t}$ also requires a mapping $\mathbf{M}_{s,t}^{F \to D} : (f, g)_F \to (u, v)$. This mapping tells which data camera ray intersects the focal surface F at $(f, g)_F$. In other words, if $(f, g)_F$ was an image-able point on F, then the image of this point in camera $D_{s',t'}$ would lie at (u', v'), as in Figure 3-2. Given the projection mapping $\mathbf{P}_{s,t} : (X, Y, Z) \to (u, v)$ that describes how three-dimensional points are mapped to pixels in the data camera $D_{s,t}$, and the mapping $\mathbf{T}_F : (f, g)_F \to (X, Y, Z)$ that maps points $(f, g)_F$ on the focal surface F to three-dimensional points in space, the mapping $\mathbf{M}_{s,t}^{F \to D}$ is easily determined, $\mathbf{M}_{s,t}^{F \to D} = \mathbf{P}_{s,t}\mathbf{T}_F$. Since the focal surface is defined at run time, $\mathbf{T}_F$ is also known. Likewise, $\mathbf{P}_{s,t}$ is known for synthetic light fields. For captured light fields, $\mathbf{P}_{s,t}$ is either be assumed or calibrated using readily available camera calibration techniques [24, 26]. Since the data cameras do not move or change their calibration, $\mathbf{P}_{s,t}$ is constant for each data camera $D_{s,t}$. For a dynamic focal surface, one modifies the mapping $\mathbf{T}_F$, which changes the placement of the focal surface. A static $\mathbf{T}_F$ with a focal surface that conforms to the scene geometry gives a depth-correction identical to the lumigraph [7].

To reconstruct a ray r from the ray database, I use a generalized depth-correction. One first finds the intersections of r with C and F. This gives the 4-D ray coordinates $(s_0, t_0, f, g)_F$ as in Figure 3-2. Using cameras near (s_0, t_0), say $D_{s',t'}$ and $D_{s'',t''}$, one applies $\mathbf{M}_{s',t'}^{F \to D}$ and $\mathbf{M}_{s'',t''}^{F \to D}$ on $(f, g)_F$, giving (u', v') and (u'', v''), respectively. This gives two rays (s', t', u'', v'') and (s'', t'', u'', v'') which are stored as the pixel (u', v') in the data camera $D_{s',t'}$ and (u'', v'') in the data camera $D_{s'',t''}$. One then applies a filter to combine the values for these two rays. In the diagram, two rays are used, although in practice, one can use more rays with appropriate filter weights.

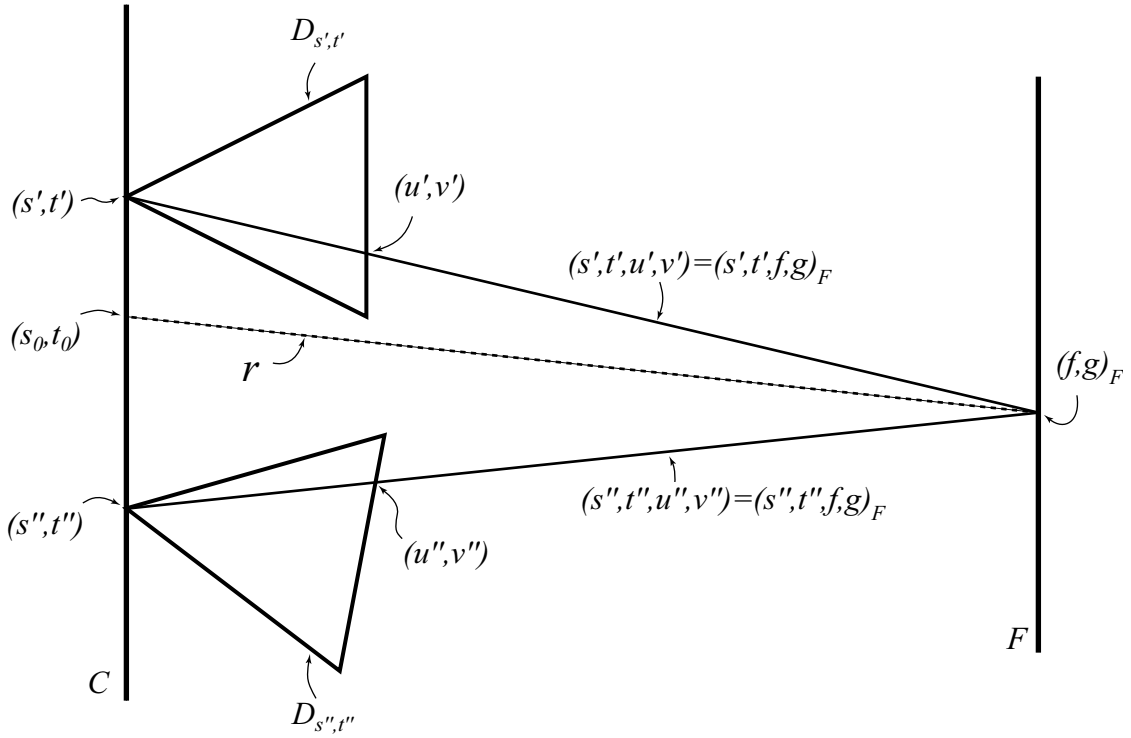


Figure 3-2: Given a ray $(s, t, f, g)_F$, one finds the rays (s', t', u', v') and (s'', t'', u'', v'') in the data cameras which intersect F at the same point $(f, g)_F$.

Chapter 4

Variable Aperture and Focus

A dynamic parameterization can be used to efficiently create images that simulate variable focus and variable depth-of-field. The user creates focused images of moderately sampled scenes with large depth variation and moderate sampling rates without requiring or extracting geometric information. In addition, this new parameterization gives the user significant control when creating novel images.

This chapter describes the effects that the variable aperture (Section 4.1) and variable focal surface (Section 4.2) have on the image synthesis. In addition, I explore some approaches that I took to create a system that has multiple planes of focus (Section 4.3).

4.1 Variable Aperture

In a traditional camera, the aperture controls the amount of light that enters the optical system. It also influences the depth-of-field present in the images. With smaller apertures, more of the scene appears in focus; larger apertures produce images with a narrow range of focus. In my system synthetic apertures are simulated not to affect exposure, but to control the amount of depth-of-field present in an image.

A depth-of-field-effect can be created by combining rays from several cameras on the camera surface. In Figure 4-1, the two rays r' and r'' are to be reconstructed. In this example, the extent of the synthetic apertures A' and A'' is four data cameras.

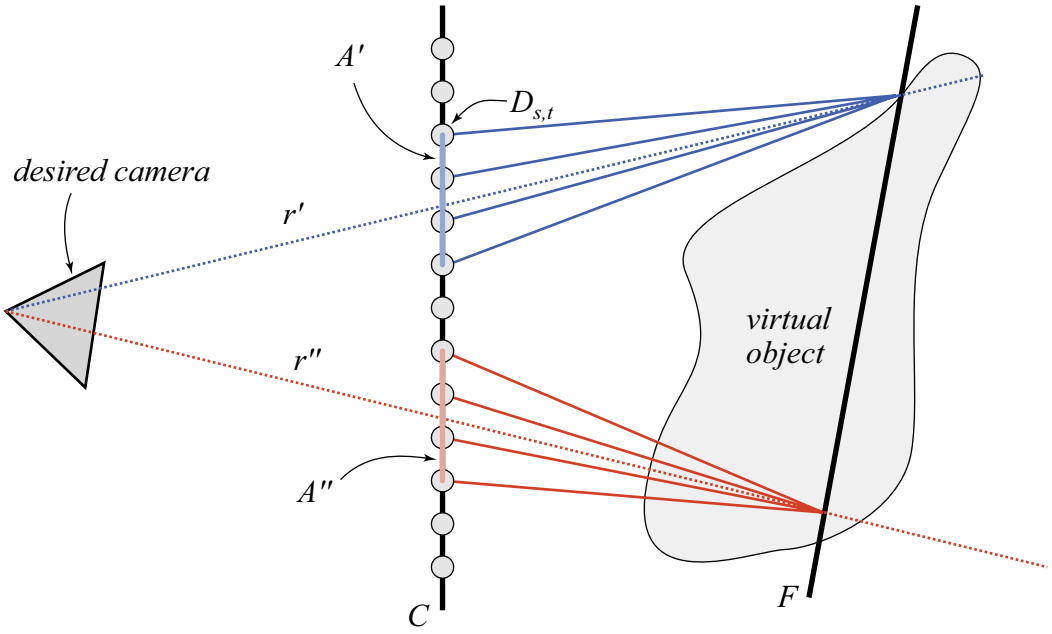


Figure 4-1: The synthetic aperture system places the aperture at the center of each desired ray. Thus the ray r' uses the aperture A' while r'' uses A''.

The synthetic apertures are centered at the intersections of r' and r'' with the camera surface C. Then, the ray database samples are recalled by applying $\mathbf{M}_{s,t}^{F \to D}$ for all (s,t) such that $D_{s,t}$ lies within the aperture. These samples are combined to create a single reconstructed ray.

Note that r' intersects F near the surface of the virtual object, whereas r'' does not. The synthetic aperture reconstruction causes r' to appear in focus, while r'' does not. The size of the synthetic aperture affects the amount of depth of field.

It is important to note that this model is not necessarily equivalent to an aperture attached to the desired camera. For example, if the desired camera is rotated, the effective aperture remains parallel to the camera surface. Modeling the aperture on the camera surface instead of the desired camera makes the ray reconstruction more efficient and still produces an effect similar to depth-of-field (See Figure 4-2). A more realistic and complete lens model is given in [12], although this is significantly less efficient to render and impractical for captured light fields.

My system does not equally weight the queried samples that fall within the syn-

Figure 4-2: By changing the aperture function, the user controls the amount of depth of field.

thetic aperture. Using a dynamic filter that controls the weighting, I improve the frequency response of the reconstruction. Figure 4-3 illustrates various attempts to reconstruct the pink dotted ray $r = (s_0, t_0, f, g)_F$. I use a two-dimensional function $w(x, y)$ to describe the point-spread function of the synthetic aperture. Typically w has a maximum at $w(0, 0)$ and is bounded by a square of width δ. The filter is defined such that $w(x, y) = 0$ whenever $x \leq -\delta/2$, $x \geq \delta/2$, $y \leq -\delta/2$, or $y \geq \delta/2$. The filters should also be designed so that the sum of sample weights add up to 1. That is, $\sum_{i=\lfloor -\delta/2 \rfloor}^{\lceil \delta/2 \rceil} \sum_{j=\lfloor -\delta/2 \rfloor}^{\lceil \delta/2 \rceil} w(x+i, y+j) = 1$ for all (x, y).

The aperture weighting on the ray $r = (s_0, t_0, f, g)_F$ is determined as follows. The center of each aperture filter is translated to the point (s_0, t_0). Then, for each camera $D_{s,t}$ that is inside the aperture, I construct a ray $(s, t, f, g)_F$ and then calculate (s, t, u, v) using the appropriate mapping $\mathbf{M}_{s,t}^{F \to D}$. Then each ray (s, t, u, v) is weighted by $w(s - s_0, t - t_0)$ and all weighted rays within the aperture are summed together. One could also use the aperture function $w(x, y)$ as a basis function at each sample to reconstruct the continuous light field, although this is not computationally efficient (this type of reasoning is covered in [7]).

The dynamically reparameterized light field system can use arbitrarily large synthetic apertures. The size of the aperture is only limited to the extent to which there are cameras on the camera surface. With sufficiently large apertures, one can "see through objects," as in Figure 4-5. One problem with making large apertures occurs when the aperture function falls outside the populated region of the camera surface. When this occurs, the weighted samples do not add up to one. This creates a vignetting effect where the image darkens when using samples near the edges of the camera surface. For example, compare Figure 4-4a with no vignetting to Figure 4-4b. In Figure 4-4b, the desired camera was near the edge of the light field, so the penguin appears dimmed. This can be solved by either adding more data cameras to the camera surface or by reweighting the samples on a pixel by pixel basis so the weights always add up to one.

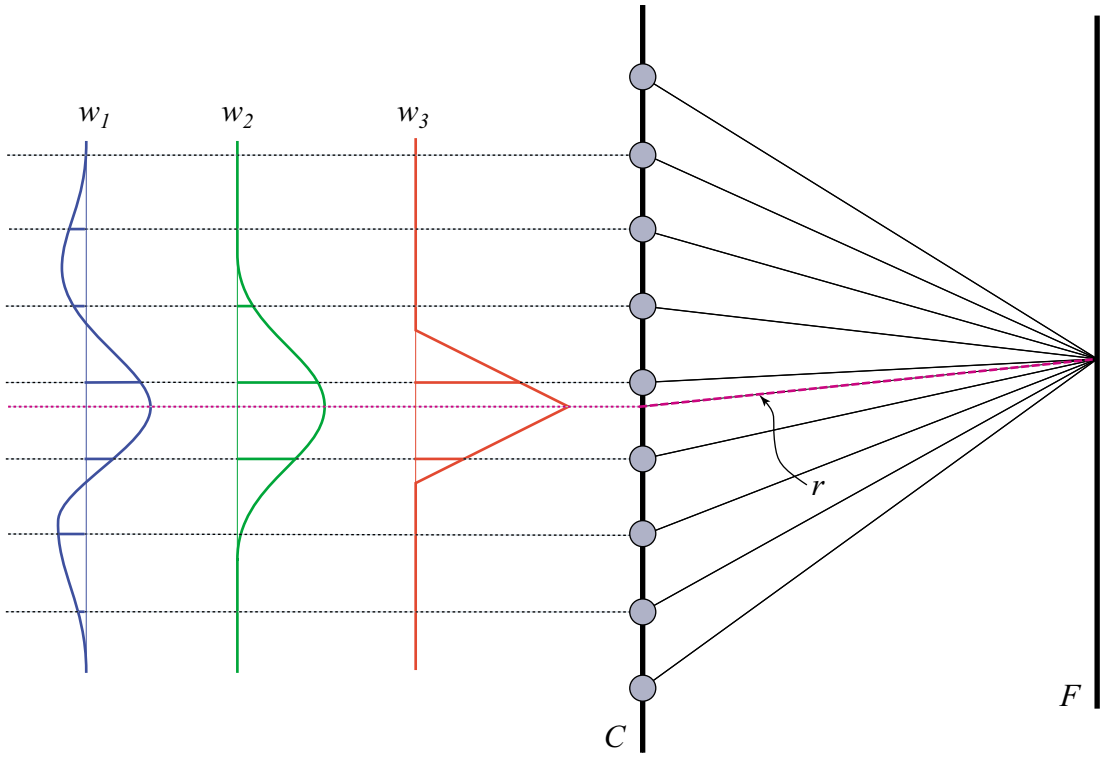
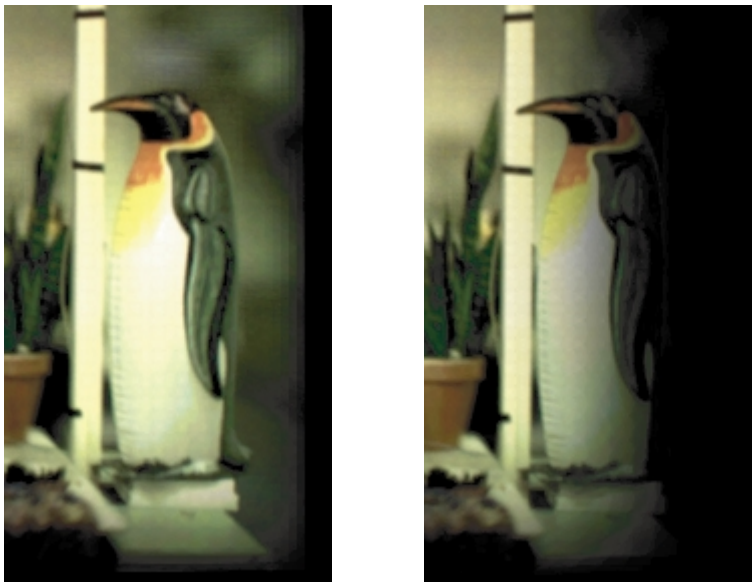


Figure 4-3: By changing the shape of the aperture filter, the user controls the amount of depth-of-field. In this figure, filter w_1 reconstructs r by combining 6 rays, w_2 combines 4 rays, and w_3 combines 2.



(a) (b)

Figure 4-4: A vignetting effect can occur when using large apertures near the edge of a light field.

Figure 4-5: By using a very large aperture, one can "see through objects.".

4.2 Variable Focus

Photographers using cameras do not only change the depth-of-field, but they vary what is in focus. Using dynamic parameterization, it is possible to create the same effect at run-time by modifying the focal surface. As before, a ray r is defined by its intersections with the camera surface C and focal surface F and are written $(s_0, t_0, f, g)_F$. The mapping $\mathbf{M}_{s,t}^{F \to D}$ tells which ray (s, t, u, v) in the data camera $D_{s,t}$ approximates $(s_0, t_0, f, g)_F$.

When the focal surface is changed to F', the same ray r now intersects a different focal surface at a different point $(f', g')_{F'}$. This gives a new coordinate $(s_0, t_0, f', g')_{F'}$ for the ray r. The new mapping $\mathbf{M}_{s,t}^{F' \to D}$ gives a pixel (u', v') for each camera $D_{s,t}$ within the aperture.

In Figure 4-8, there are three focal surfaces, F_1, F_2, and F_3. Note that any single ray r is reconstructed using different sample pixels, depending on which focal surface is used. For example, if the focal surface F_n is used, then the rays (s', t', u'_n, v'_n) and (s'', t'', u''_n, v''_n) are used in the reconstruction.

Note that this selection is a dynamic operation. In the light field and lumigraph systems, the querying the ray r would always resolve to the same reconstructed sample. As shown in Figure 4-6, one can effectively control which part of the scene is in focus by simply moving the focal surface. If the camera surface is too sparsely sampled, then the out-of-focus objects appear aliased, as with the object in the lower image of Figure 4-6. This aliasing is analyzed in Section 6. The focal surface does not have to be perpendicular to the viewing direction, as one can see in Figure 4-7.

Many scenes can not be entirely focused with a single focal plane. As in Figure 4-8, the focal surfaces do not have to be planar. One could create a focal surface out of a parameterized surface patch that passes through key points in a scene, a polygonal model, or a depth map. Analogous to depth-corrected lumigraphs, this would insure that all visible surfaces are in focus. But, in reality, these depth maps would be hard and/or expensive to obtain with captured data sets. However, using a system similar to the dynamically reparameterized light field, a non-planar focal

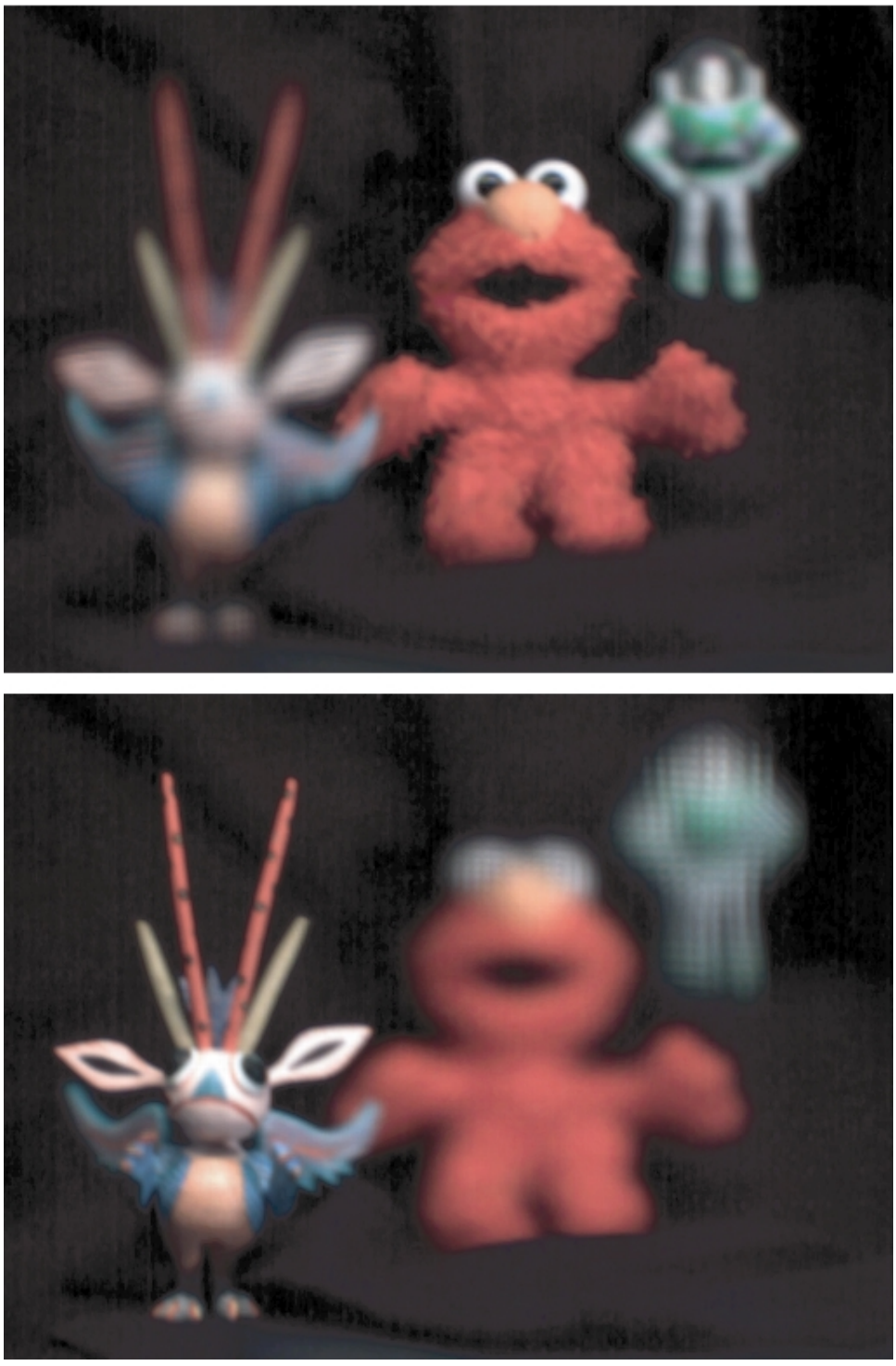


Figure 4-6: By varying the focal surface, one controls what appears in focus.

Figure 4-7: I have placed a focal plane that is not parallel to the image plane. In this case, the plane passes through part of the tree, the front rock, and the leftmost rock on the island. The plane of focus is seen intersecting with the water in a line.

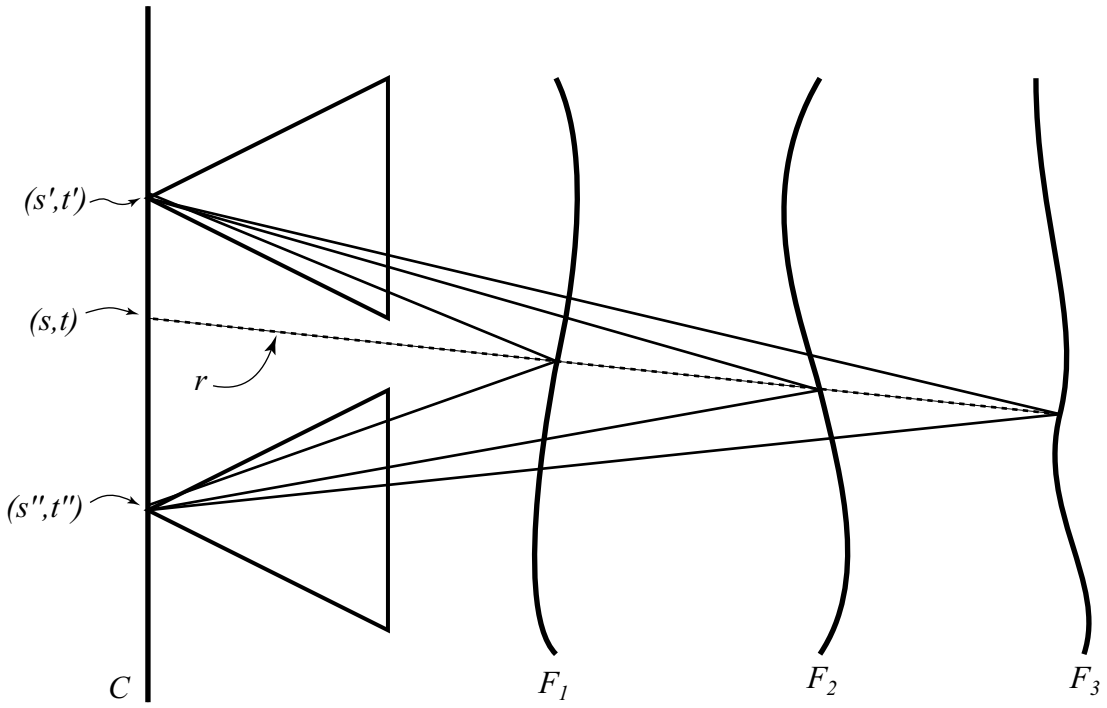


Figure 4-8: By changing the focal surfaces, I dynamically control which samples in each data camera contribute to the reconstructed ray.

surface could be modified dynamically until a satisfactory image is produced.

4.3 Multiple Focal Surfaces and Apertures

In general, we would like to have more than just the points near a single surface in clear focus. One solution is to use multiple focal surfaces. The approach is not available to real cameras. In a real lens system, only one continuous plane is in focus at one time. However, since the system is not confined by physical optics, it can have two or more distinct regions that are in focus. For example, in Figure 4-9, the red bull in front and the monitors in back are in focus, yet the objects in between, such as the yellow fish and the blue animal, are out of focus. Using a DRLF parameterization, one chooses an aperture and focal surface on a per region or per pixel basis. Multiple apertures would be useful to help reduce vignetting near the edges of the camera surfaces. If the aperture passes near the edge of the camera surface, then one could

Figure 4-9: I use two focal surfaces to make the front and back objects in focus, while those in the middle are blurry.

reduce its size so that it remains inside the boundary.

Using a real camera, this is done by first taking a set of pictures with different planes of focus, and then taking the best parts of each image and compositing them together as a post-process [18]. I now present a method that uses a ray-tracer-based dynamically-reparameterized-light-field-renderer to produce a similar result by expanding the ray-database-query-method.

Since a ray r intersects each focal surface F_i at a unique $(f, g)_{F_i}$, some scoring scheme is needed to pick which focal surface to use. One approach is to pick the focal surface which makes the picture look most focused, which means the system needs to pick the focal surface which is closest to the actual object being looked at. By augmenting each focal surface with some scoring $\sigma(f, g)_F$, which is the likelihood

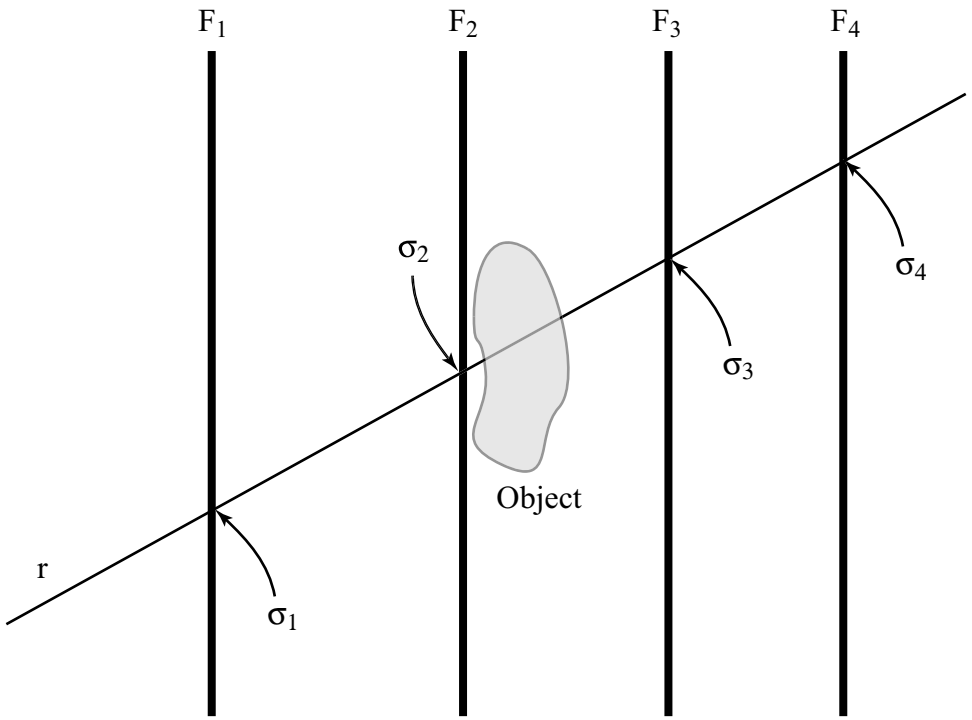


Figure 4-10: To find the best focal plane, I calculate a score at each intersection and take the focal plane with the best score. If the scoring system is good, the best score is the one nearest the surface one is looking at.

a visible object is near $(f, g)_F$, one can calculate σ for each focal surface, and have the system pick the focal point with the best score σ. In Figure 4-10, σ_2 would have the best score since it is closest to the object. Note that an individual score $\sigma(f, g)_F$ is independent of the view direction; however, the set of scores compared for a particular ray r is dependent on the view direction. Therefore, although the scoring data is developed without view dependence, view dependent information is still extracted from it.

Given light-fields, but no information about the geometry of the scene, it is possible to estimate these scores from the image data alone. I have chosen to avoid computer vision techniques that require deriving correspondences to discover the actual geometry of the scene, as these algorithms have difficulties dealing with ambiguities that are not relevant to generating synthetic images. For example, flat regions without texture can be troublesome to a correspondence-based vision system, and for

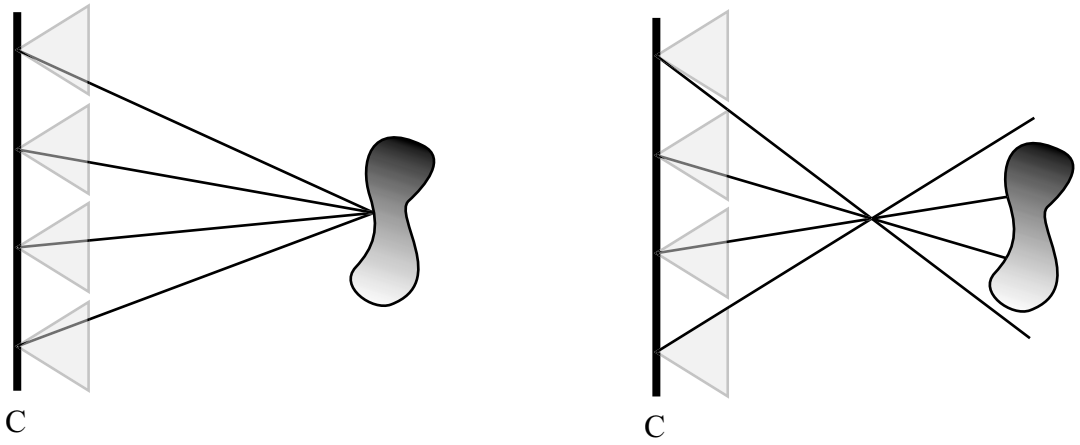


Figure 4-11: Creating a radiance function from a light field. If the radiance function is near an object, then the function is smooth. If the radiance function is not near an object, it varies greatly.

these regions it is hard to find an exact depth. However, when making images from a light field, picking the wrong depth near the same flat region would not matter, because the region would still look in focus. Therefore, a scoring system that takes advantage of this extra freedom is desired. And, because only the original images are available as input, a scoring system where scores are easily created from these input images is required.

One approach is to choose the locations on the focal surfaces that approximate radiance functions. Whereas one usually uses light-fields to construct images, it is also possible to use them to generate radiance functions. The collection of rays in the ray database that intersect at $(f, g)_F$ is the discretized radiance function of the point $(f, g)_F$. If the point lies on an object and is not occluded by other objects, the radiance function is smooth, as in the left side of Figure 4-11. However, if the point is not near an actual object, then the radiance function is not smooth, as in the right side of Figure 4-11.

To measure smoothness, one looks for a lack of high frequencies in the radiance function. High frequencies in a radiance function identify 1) a point on a extremely specular surface, 2) an occlusions in the space between a point and a camera, or

3) a point in empty space. Thus, by identifying the regions where the there are no high frequencies in the radiance function, we know the point must be near a surface. Because of their high frequency content, we might miss areas that are actually on a surface but have an occluder in the way.

Figure 4-12 shows a few examples of what these radiance functions look like. Each radiance image in column 4-12a and 4-12c has a resolution equal to sampling rate of the camera surface. Thus, since these light fields each have 16x16 cameras on the camera surface, the radiance images have a resolution of 16x16. Traveling down the column shows different radiance images formed at different depths along a single ray. The ray to be traversed is represented by the black circle, so that pixel stays constant. The third image from the top is the best depth estimate for the radiance image because it varies smoothly. To the left of each radiance image is a color cubes showing the distribution of color in each radiance image immediately to its left. The third distribution image from the top has the lowest variance, and is therefore the best choice.

Because calculating the radiance function is a slow process, the scores are found at discrete points on each focal surface as a preprocessing step. This allows the use of expensive algorithms to setup the scores on the focal surfaces. Then, when rendering, the prerecorded scores for each focal surface intersection are accessed and compared.

To find the best focal surface for a ray, an algorithm must first obtain intersections and scores for each focal surface. Since this is linear in the number of focal surfaces, I would like to keep the number of focal surfaces small. However, the radiance functions are highly local, and small changes in the focal surface position gives large changes in the score. Nevertheless, the accuracy needed in placing the focal surfaces is not very high. That is, often it is unnecessary to find the exact surface that makes the object in perfect focus; we just need to find a surface that is close to the object. Therefore, the scores are calculated by sampling the radiance functions for a large number of planes, and then 'squashing' the scores down into a smaller set of planes using some function $\lambda(\sigma_i, \ldots, \sigma_{i+m})$. For example, in Figure 4-13, the radiance scores for 16 planes are calculated. Then, using some combining function λ, the scores from these 16 planes

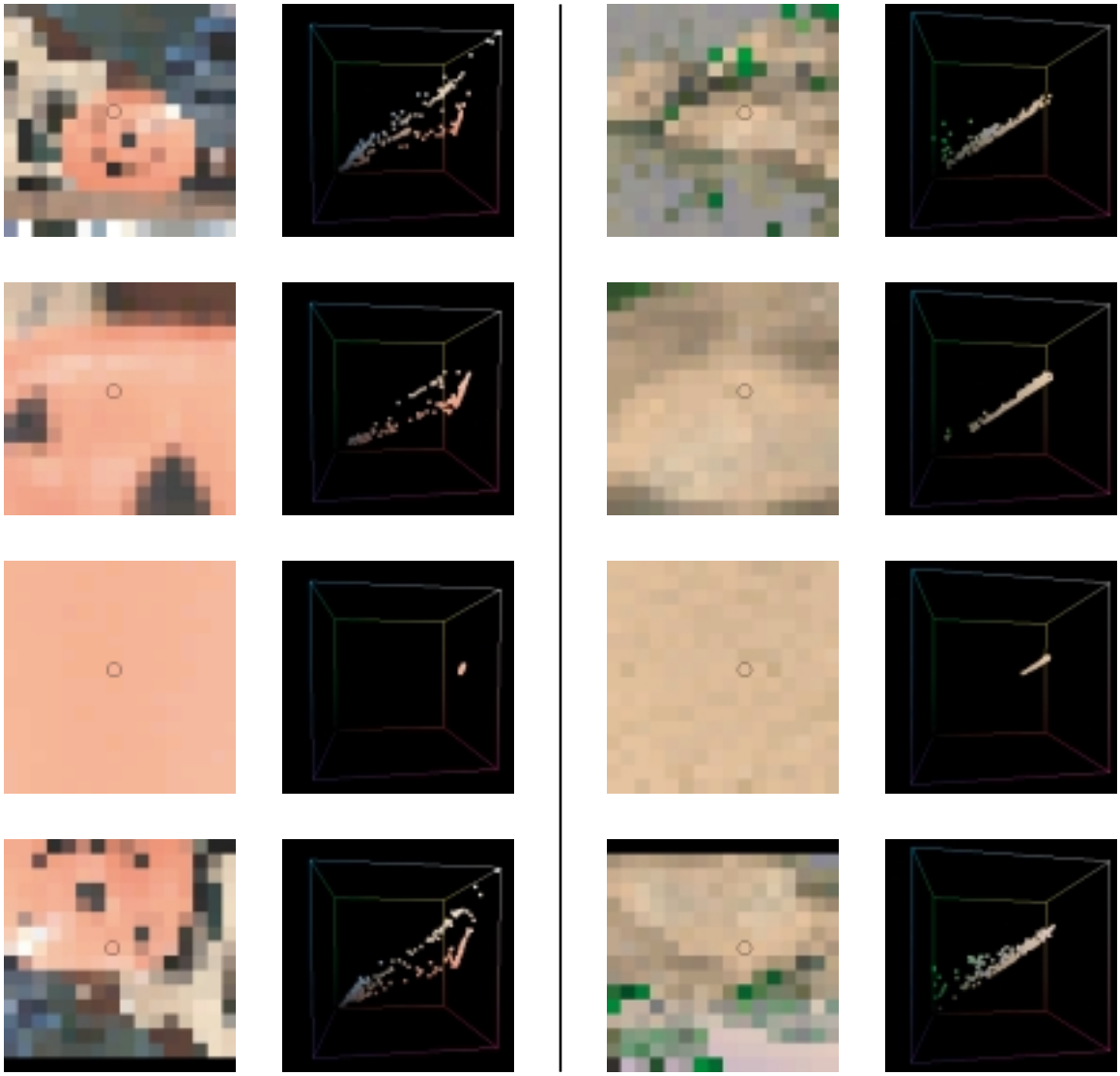


Figure 4-12: **(a,c)** Radiance Images at different depths along a single ray. The circled pixel stays constant in each image, because that ray is constant. The third image from the top is the best depth for the radiance image. **(b,d)** Color cubes showing the distribution of color in each radiance image immediately to its left. The third distribution image from the top has the lowest variance, and is therefore the best choice.

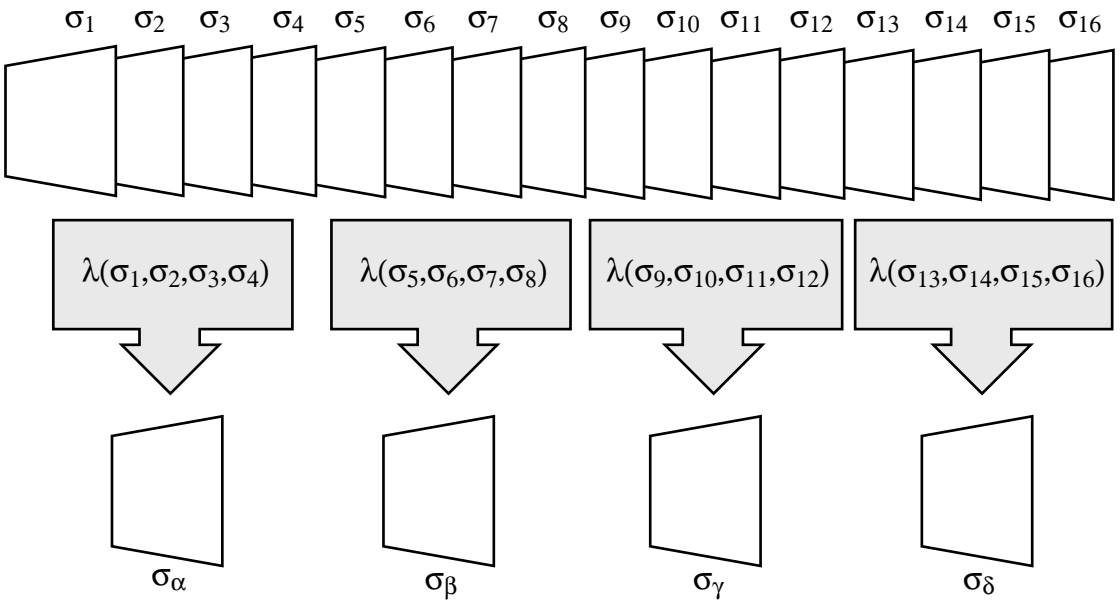
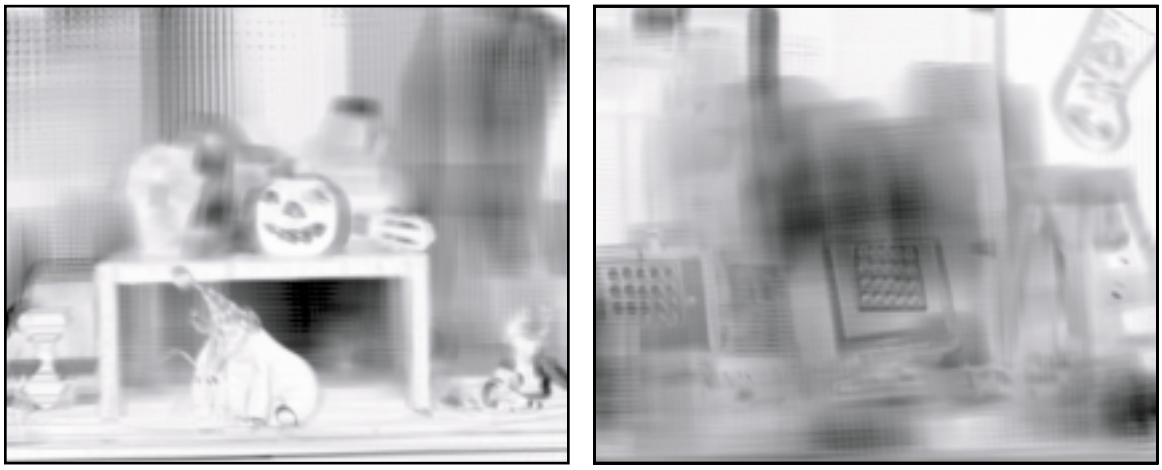


Figure 4-13: I compute scores on many focal surfaces, and then combine them to a smaller set of focal surfaces, so the run-time algorithm has less scores to compare.

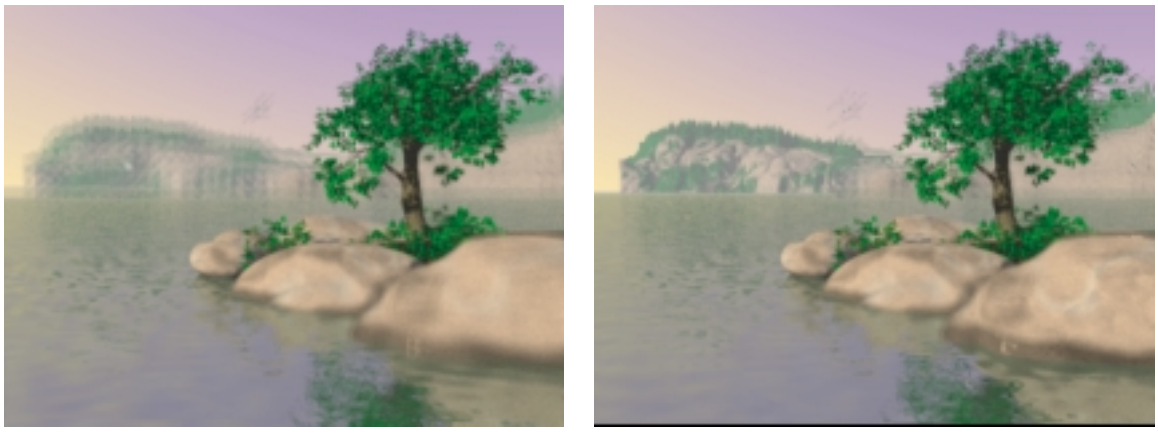
are combined into new scores on four planes. These four planes and their scores are used as focal surfaces at run time. I chose to use the maximizing function, that is, $\lambda(\sigma_i, \ldots, \sigma_{i+m}) = \max(\sigma_i, \ldots, \sigma_{i+m})$. Other non-linear or linear weighting functions might provide better results.

Figure 4-14 shows two visualizations of the scores on the front and back focal planes used to create the image in Figure 4-9. The closer to white, the better the score σ, which means objects are likely to be located near that plane. Figure 4-15b is another example that was created using 8 focal planes combined down to 4.



(a) (b)

Figure 4-14: Visualizations of the σ scores used on the front (a) and back (b) focal planes for the picture in Figure 4-9. The closer to white, the better the score σ, which means objects are likely to be located near that plane.



(a) (b)

Figure 4-15: **(a)** Using the standard light field parameterization, only one fixed plane is in focus. Using the smallest aperture available, this would be the best picture I could create. **(b)** By using 4 focal planes (originally 8 focal planes with scores compressed down to 4), I clearly do better than the image in (a). Especially note the hills in the background and the rock in the foreground.

Chapter 5

Ray-Space Analysis of Dynamic Reparameterization

It is instructive to consider the effects of dynamic reparameterization on light fields when viewed from ray space [14, 7] and, in particular, within epipolar plane images (EPIs) [1]. The chapter discusses how dynamic reparameterization transforms the ray space representations of a light field (Section 5.1). In addition, I explore the ray space diagram of a single point feature (Section 5.2). Finally, I present and discuss actual imagery of ray space diagrams created from photographic light fields (Section 5.3).

5.1 Interpreting the Epipolar Image

It is well-known that 3-D structures of an observed scene are directly related to features within particular light field slices. These light field slices, called epipolar plane images (EPIs) [1], are defined by planes through a line of cameras. The two-parallel plane ($2PP$) parameterization is particularly suitable for analysis under this regime as shown by [8]. A dynamically reparameterized light field system can also be analyzed in ray space, especially when the focal surface is planar and parallel to the camera surface. In this analysis, I consider a 2-D subspace of rays corresponding to fixed values of t and g on a dynamic focal plane F. When the focal surface is parallel

to the camera surface, the sf slice is identical to an EPI.

A dynamically reparameterized 2-D light field of four point features is shown in Figure 5-1a. The dotted point is a point at infinity. A light field parameterized with the focal plane F_1 has a sf_1 ray-space slice similar to Figure 5-1b. Each point feature in geometric space corresponds to a linear feature in ray space, where the slope of the line indicates the relative depth of the point. Vertical features in the slice represent points on the focal plane; features with positive slopes represent points that are further away and negative slopes represent points that are closer. Points infinitely far away have a slope of 1 (for example, the dashed line). Although not shown in the figure, variation in color along the line in ray space represents the view-dependent radiance of the point. If the same set of rays is reparameterized using a new focal plane F_2 that is parallel to the original F_1 plane, the sf_2 slice shown in Figure 5-1c results. These two slices are related by a shear transformation along the dashed line. If the focal plane is oriented such that it is not parallel with the camera surface, as with F_3, then the sf slice is transformed non-linearly, as shown in Figure 5-1d. However, each horizontal line of constant s in Figure 5-1d is a linearly transformed (i.e. scaled and shifted) version of the corresponding horizontal line of constant s in Figure 5-1b. In summary, dynamic reparameterization of light fields amounts to a simple transformation of ray space. When the focal surface remains perpendicular to the camera surface but its position is changing, this results in a shear transformation of ray space.

Changing the focal-plane position thus affects which ray-space features are axis-aligned. This allows the use of a separable, axis-aligned reconstruction filters in conjunction with the focal plane to select which features are properly reconstructed. Equivalently, one interprets focal plane changes as aligning the reconstruction filter to a particular feature slope, while keeping the ray space parameterization constant.

Under the interpretation that a focal plane shears ray space and keeps reconstruction filters axis aligned, the aperture filtering methods described in Chapter 4 amount to varying the extent of the reconstruction filters along the s dimension. In Figure 5-1e, the dashed horizontal lines depict the s extent of three different aperture

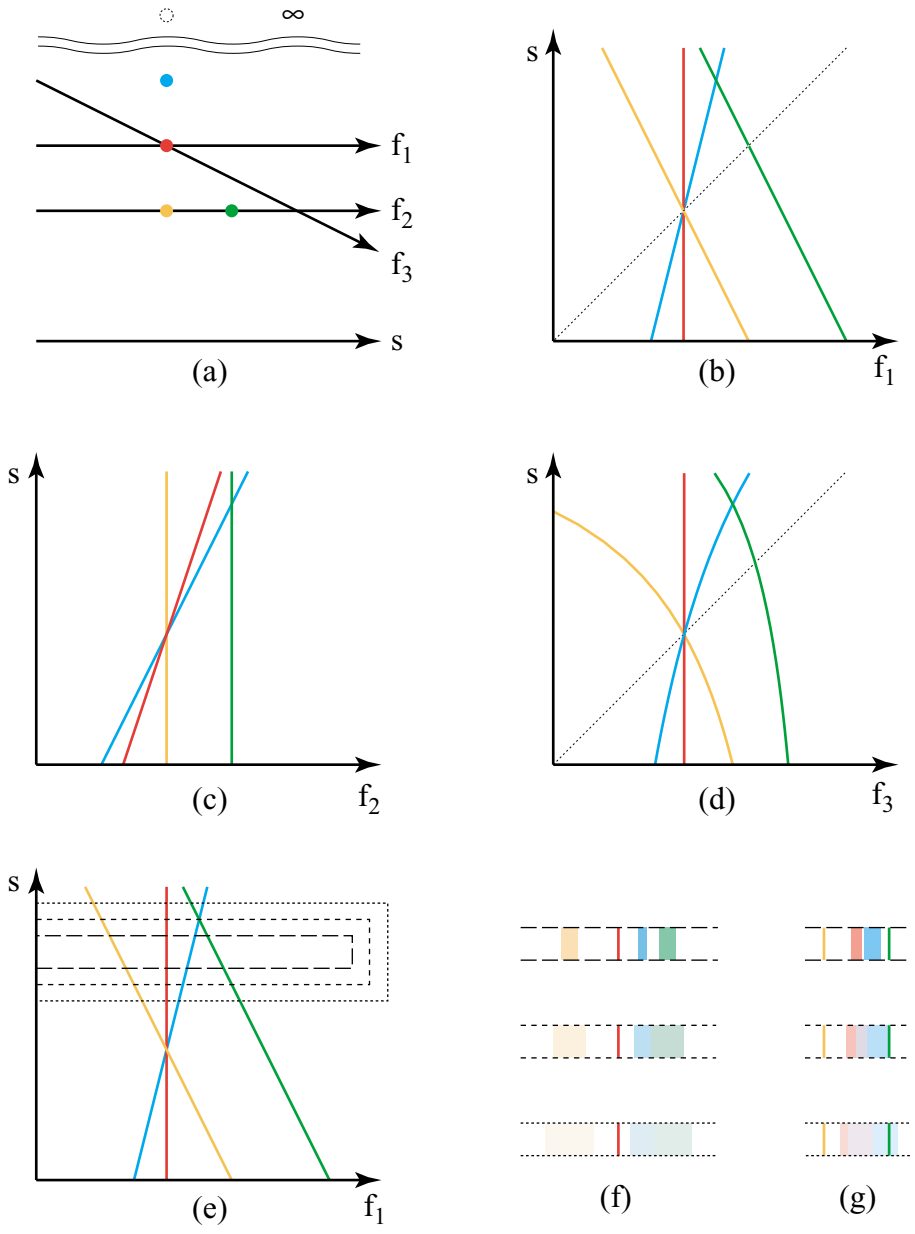


Figure 5-1: **(a)** A light field of four points, with 3 different focal planes. **(b,c,d)** sf slices using the three focal planes, F_1, F_2 and F_3. **(e)** Three aperture filters drawn on the ray space of (b) **(f)** Line images constructed using the aperture filters of (e). **(g)** Line images constructed using the aperture filters of (e), but the ray space diagram of (c)

filters (I assume they are infinitely thin in the f_1 dimension). Figure 5-1f shows three line images constructed from the EPI of Figure 5-1e. As expected, the line image constructed from the widest aperture filter has the most depth-of-field. Varying the extent of the aperture filter has the effect of "blurring" features located far from the focal plane while features located near on the focal plane are relatively sharp. In addition, the filter reduces the amount of view-dependent radiance for features aligned with the filter. When the ray space is sheared to produce the parameterization of Figure 5-1c and the same three filters are used the images of Figure 5-1g are produced.

5.2 Constructing the Epipolar Image

Arbitrary ray-space images (similar to Figure 5-1) of a single point can be constructed as follows. Beginning with the diagram of Figure 5-2. Let $\mathbf{P}$ be a point feature that lies at a position $(\mathbf{P_x}, \mathbf{P_z})$ in the 2-D world coordinate system. The focal surface $\mathbf{F}$ is defined by a point $\mathbf{F_0}$ and a unit direction vector $\mathbf{F_d}$. Likewise, the camera surface $\mathbf{C}$ is defined by a point $\mathbf{S_0}$ and a unit direction vector $\mathbf{S_d}$. I would like to find a relation between s and f so that I can plot the curve for $\mathbf{P}$ in the EPI domain.

The camera surface is described using the function

$$\mathbf{S_0} + s\mathbf{S_d} = \mathbf{S_s}$$

The focal surface is described using the function

$$\mathbf{F_0} + f\mathbf{F_d} = \mathbf{F_f}$$

Thus, given varying values of s, $\mathbf{S_s}$ plots out the camera surface in world coordinates. Likewise, given varying values of f, $\mathbf{F_f}$ plots out the focal surface in world coordinates.

For sake of construction, let s be the domain and f be the range. Thus, we wish to express values of f in terms of s. Given a value of s, find some point $\mathbf{S_s}$. Then construct a ray which passes from the point $\mathbf{S_s}$ on the camera surface through the point $\mathbf{P}$ that intersects the focal surface at some point $\mathbf{F_f}$. This ray is shown as the

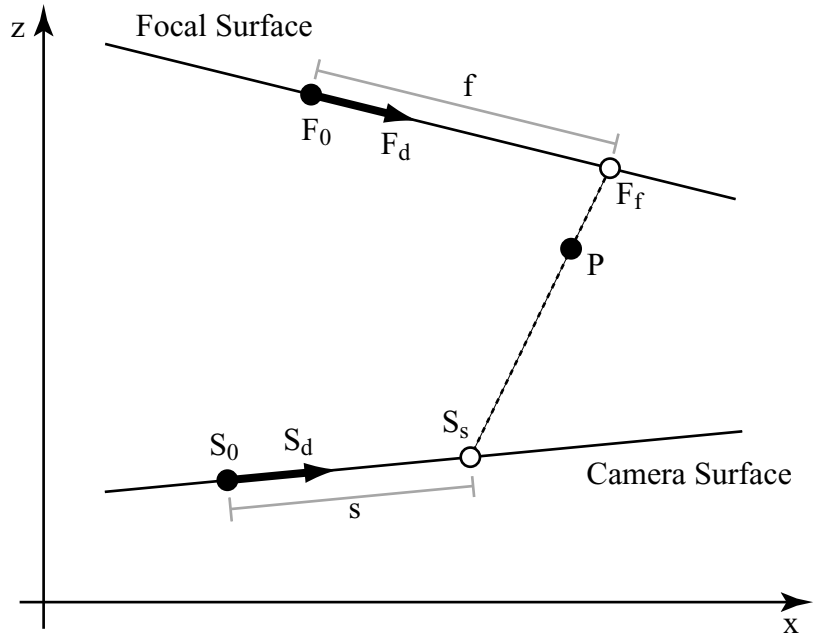


Figure 5-2: Calculating the (s, f) pair for a point P in space. The focal surface is defined by a point F_0 and a direction F_d. Likewise, the camera surface is defined by a point S_0 and a direction S_d.

dotted line in Figure 5-2. The equation for this ray is

$$\mathbf{S_s} + \alpha(\mathbf{P} - \mathbf{S_s}) = \mathbf{F_f}$$

This gives the final relation between s and f as

$$\mathbf{S_0} + s\mathbf{S_d} + \alpha(\mathbf{P} - \mathbf{S_0} - s\mathbf{S_d}) = \mathbf{F_0} + f\mathbf{F_d}$$

Because $\mathbf{S_0}$, $\mathbf{S_d}$, $\mathbf{F_0}$, and $\mathbf{F_d}$ are direction or position vectors, the above equation is a system of two equations, with three unknown scalars, s, f, and α. This lets me solve for f in terms of s, and likewise I solve for s in terms of f. The two equations in the system are

$$\begin{aligned} \alpha P_x + S_{0x}(1-\alpha) + sS_{dx}(1-\alpha) &= F_{0x} + fF_{dx} \\ \alpha P_y + S_{0y}(1-\alpha) + sS_{dy}(1-\alpha) &= F_{0y} + fF_{dy} \end{aligned}$$

Solving the above system of equations for f in terms of s gives:

$$f = \frac{F_{0x}(P_y - S_{0y} - sS_{dy}) - F_{0y}(P_x - S_{0x} - sS_{dx}) + P_x(S_{0y} + sS_{dy}) - P_y(S_{0x} + sS_{dx})}{F_{dy}(P_x - S_{0x} - sS_{dx}) - F_{dx}(P_y - S_{0y} - sS_{dy})} \tag{5.1}$$

Solving for s in terms of f gives:

$$s = \frac{(F_{0x} + fF_{dx})(P_y - S_{0y}) + (F_{0y} + fF_{dy})(S_{0x} - P_x) + P_xS_{0y} - P_yS_{0x}}{F_{0x}S_{dy} - F_{0y}S_{dx} + fF_{dx}S_{dy} - fF_{dy}S_{dx} - P_xS_{dy} + P_yS_{dx}} \tag{5.2}$$

If we take the common case where $\mathbf{F_d}$ and $\mathbf{S_d}$ are parallel and both point along the s-axis, then $F_{dy} = S_{dy} = 0$, $F_{dx} = S_{dx} = 1$ and we can reduce equation (5.1) to the linear form:

$$f = \frac{F_{0x}(P_y - S_{0y}) - F_{0y}(P_x - S_{0x} - s) + P_xS_{0y} - P_y(S_{0x} + s)}{-(P_y - S_{0y})} \tag{5.3}$$

5.3 Real Epipolar Imagery

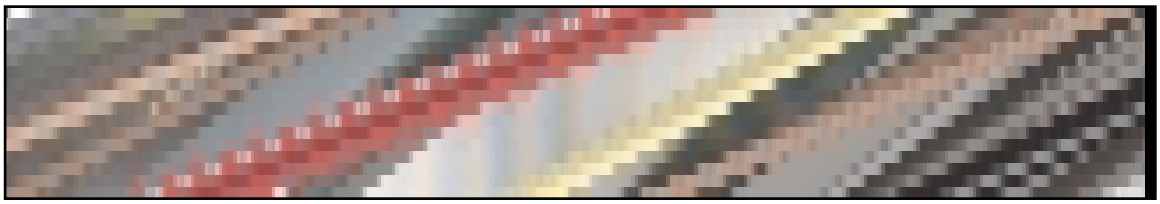
Looking at epipolar images made from real imagery is also instructive. By taking a horizontal line of rectified images from a ray database (varying s, keeping t constant) it is possible to construct such an EPI. Three of these images are shown in Figure 5-3a. The green, blue, and red lines represent values of constant v which are used to create the three EPIs of Figures 5-3b-d.

In Figure 5-4, I show the effect of shearing the epipolar images using the example EPI imagery from Figure 5-3d. In Figure 5-4a, I reproduce Figure 5-3d. In this EPI, the focal plane lies at infinity. Thus, features infinitely far away would appear as vertical lines in the EPI. Because the scene had a finite depth of a few meters, there are no vertical lines in the EPI. When the focal plane is moved to pass through features in the scene, the EPIs are effectively sheared so that other features appear vertical. Figures 5-4b and 5-4c show the shears that would make the red and yellow features in focus when using an vertically aligned reconstuction filter.

(a)

(b)

(c)



(d)

Figure 5-3: **(a)** Three sample images taken from a line of images of a ray database (constant t, varying s). The green, blue and red lines represent the constant lines of v for making epipolar images. **(b,c,d)** Epipolar images represented by the green, blue, and red lines.

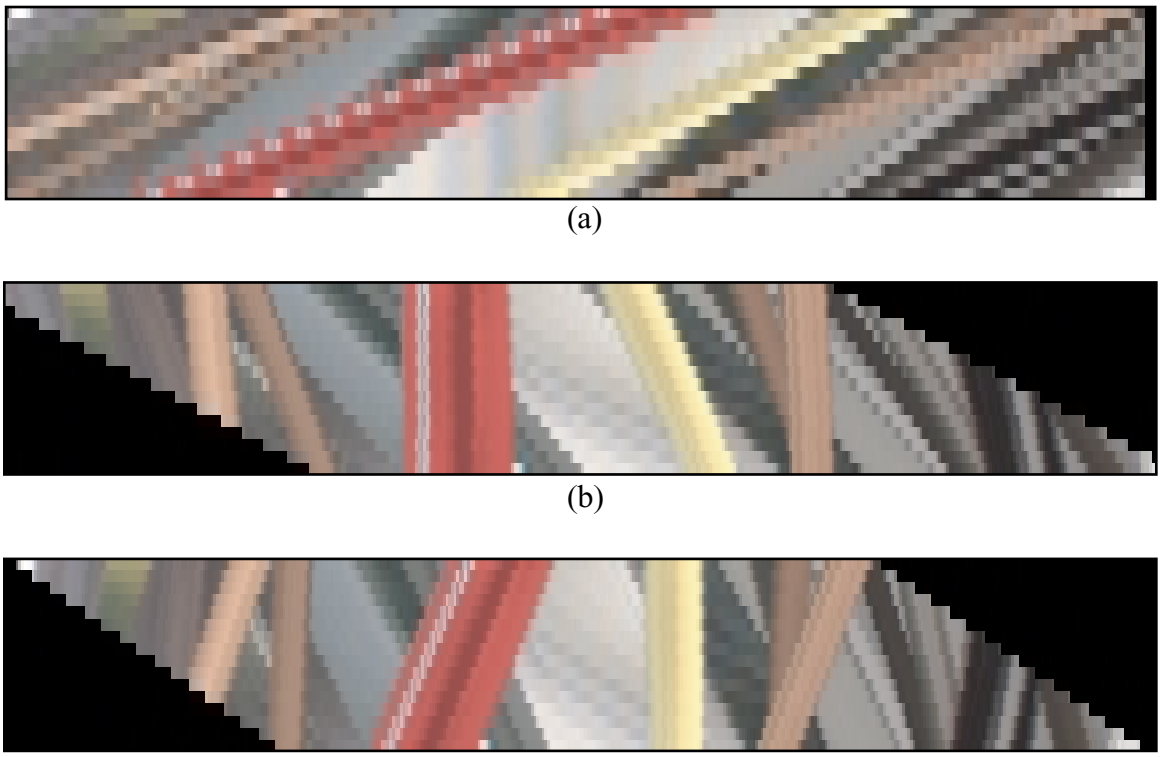


Figure 5-4: **(a)** The EPI of Figure 5-3d; the focal plane is placed at infinity. **(b)** Same EPI, sheared so that the focal plane passes through the red feature. **(c)** Same EPI, sheared so that the focal plane passes through the yellow feature.

Chapter 6

Fourier Domain Analysis of Dynamic Reparameterization

This chapter discusses the relationship between dynamic reparameterization and aliasing. When discussing aliasing, it is useful to look at the frequency domain transform of the system. I will discuss how to apply Fourier analysis to light fields. Then, I describe how my reparameterization compares with other approaches to limit aliasing within a light field.

Ray space transformations have other effects on the reconstruction process. Since shears arbitrarily modify the relative sampling frequencies between dimensions in ray space, they present considerable difficulties when attempting to band-limit the source signal. Furthermore, any attempt to band-limit the sampled function based on any particular parameterization severely limits the fidelity of the reconstructed signals from the light field.

In this section, I analyze the frequency-domain dual of a dynamically reparameterized light field. Whereas in Chapter 5 I interpreted dynamic reparameterization as ray space shearing, in this chapter I interpret the ray space as fixed (using dimensions s and u) and instead shear the reconstruction filters.

Consider an 'ideal' continuous light field of a single linear feature with a Gaussian cross section in the u direction located slightly off the u plane as shown in the EPI of Figure 6-1a. In the frequency domain, this su slice has the power spectrum shown

in white Figure 6-1b (ignore the red box for now). Any sampling of this light field generates copies of this spectrum as shown in Figure 6-1c (ignore the blue box for now). Typical light fields have a higher sampling density on the data camera images than on the camera surface, and this example reflects this convention by repeating the spectrum at different rates in each dimension. Attempting to reconstruct this signal with a separable reconstruction filter under the original parameterization as shown by the blue box in Figure 6-1c results in an image that exhibits considerable post-aliasing, because of the high-frequency leakage from the other copies. This quality degradation shows up as ghosting in the reconstructed image, where multiple copies of a feature can be faintly seen. Note that this ghosting is a form of post-aliasing; the original sampling process has not lost any information.

One method for remedying this problem is to apply an aggressive band-limiting prefilter to the continuous signal before sampling. This approach is approximated by the aperture-filtering step described in [14]. When the resulting band-limited light field is sampled using the prefilter of Figure 6-1a (shown as the red box), the power spectrum shown in Figure 6-1d results. This signal can be reconstructed exactly with an ideal separable reconstruction filter as indicated by the blue box in Figure 6-1d. However, the resulting EPI, shown in Figure 6-1e, contains only the low-frequency portion of the original signal energy, giving a blurry image.

Dynamic reparameterization allows many equally valid reconstructions of the light field. The shear transformation of ray-space effectively allows for the application of reconstruction filters that would be non-separable in the original parameterization. Thus, using dynamic reparameterization, the spectrum of the single point can be recovered exactly using the filter indicated by the blue box in Figure 6-1f.

Issues are more complicated in the case when multiple point features are represented in the light field, as shown in the *su* slice in Figure 6-2a. The power spectrum of this signal is shown in Figure 6-2b. After sampling, multiple copies of the original signal's spectrum interact, causing a form of pre-aliasing that cannot be undone by processing. Dynamic reparameterizations allows for a single feature from the spectrum to be extracted, as shown by the two different filters overlaid on Figure 6-2c.

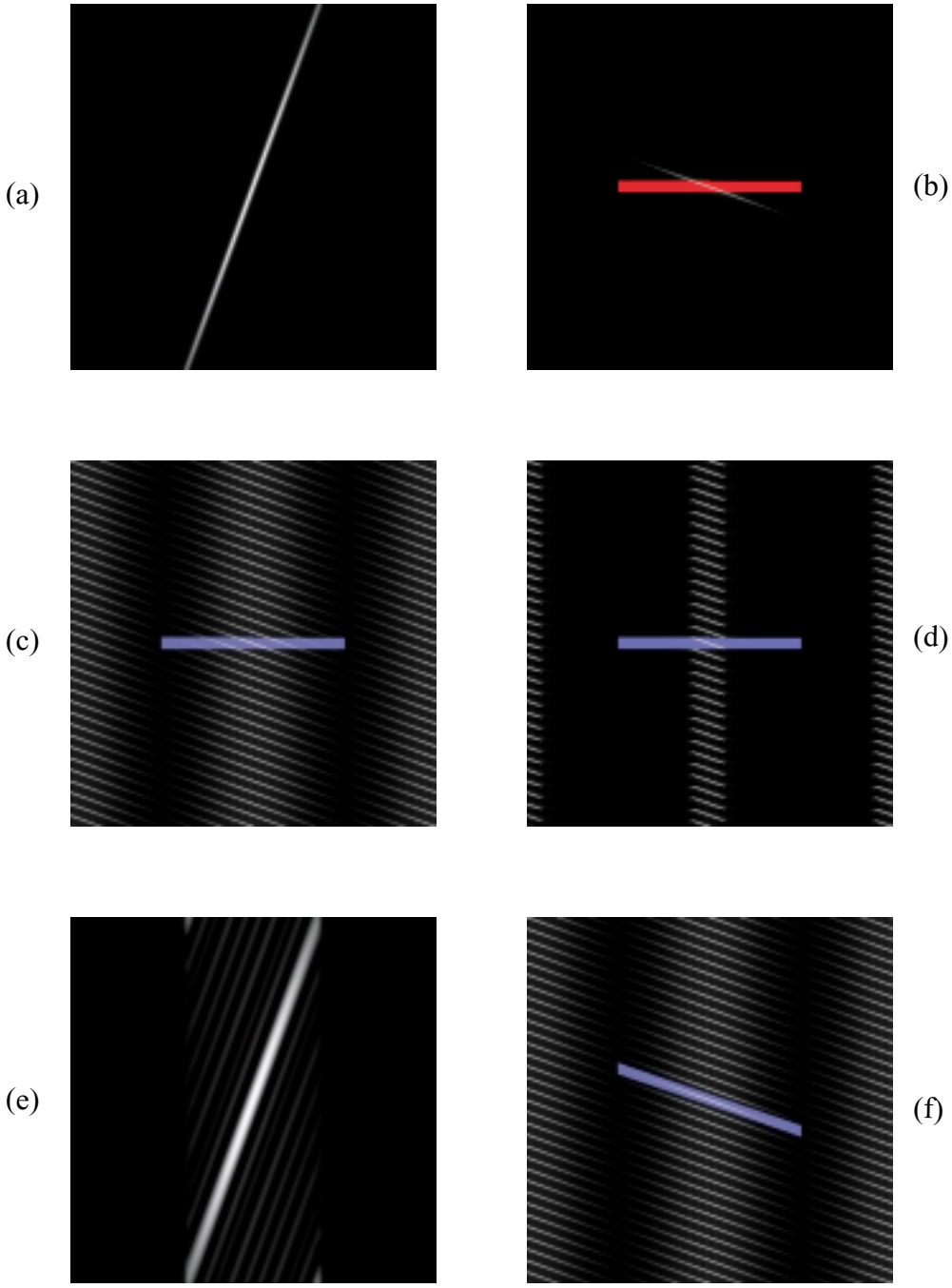


Figure 6-1: **(a)** *su* slice of a single feature. **(b)** Frequency domain power spectrum of (a). Aperture prefilter drawn in red. **(c)** Power spectrum after typical sampling. Traditional reconstruction filter shown in blue. **(d)** Power spectrum of sampled data after prefilter of (b). Traditional reconstruction filter shown in blue. **(e)** In space domain, the result of (d) is a blurred version of (a). **(f)** By using alternative reconstruction function filter, I accurately reconstruct (a).

However, some residual energy from the other points is also captured, and appears in the reconstructed image as ghosting (see Figure 6-2e, where the blue filter was used in reconstruction).

One method for reducing this artifact is to increase the size of the synthetic aperture. In the frequency domain, this reduces the width of the reconstruction filters as shown in Figure 6-2d. Using this approach, one can reduce, in the limit, the contribution of spurious features to a small fraction of the total extracted signal energy. The part that cannot be extracted is the result of the pre-aliasing. By choosing sufficiently wide reconstruction apertures (or narrow in the frequency domain), the effect of the pre-aliasing can be made imperceptible (below the quantization threshold). Figure 6-2f is reconstructed by using a wider aperture than that in Figure 6-2e. Note that the aliasing in Figure 6-2f has less energy and is more spread out than in Figure 6-2e. In addition, the well-constructed feature has lost some view dependence, because it has also been filtered along its long dimension.

This leads to a general trade-off that must be considered when working with moderately sampled light fields. I either 1) apply prefiltering at the cost of limiting the range of images that can be synthesized from the light field and endure the blurring and attenuation artifacts that are unavoidable in deep scenes or 2) accept some aliasing artifacts in exchange for greater flexibility in image generation. The visibility of aliasing artifacts can be effectively limited by selecting appropriate apertures for a given desired image.

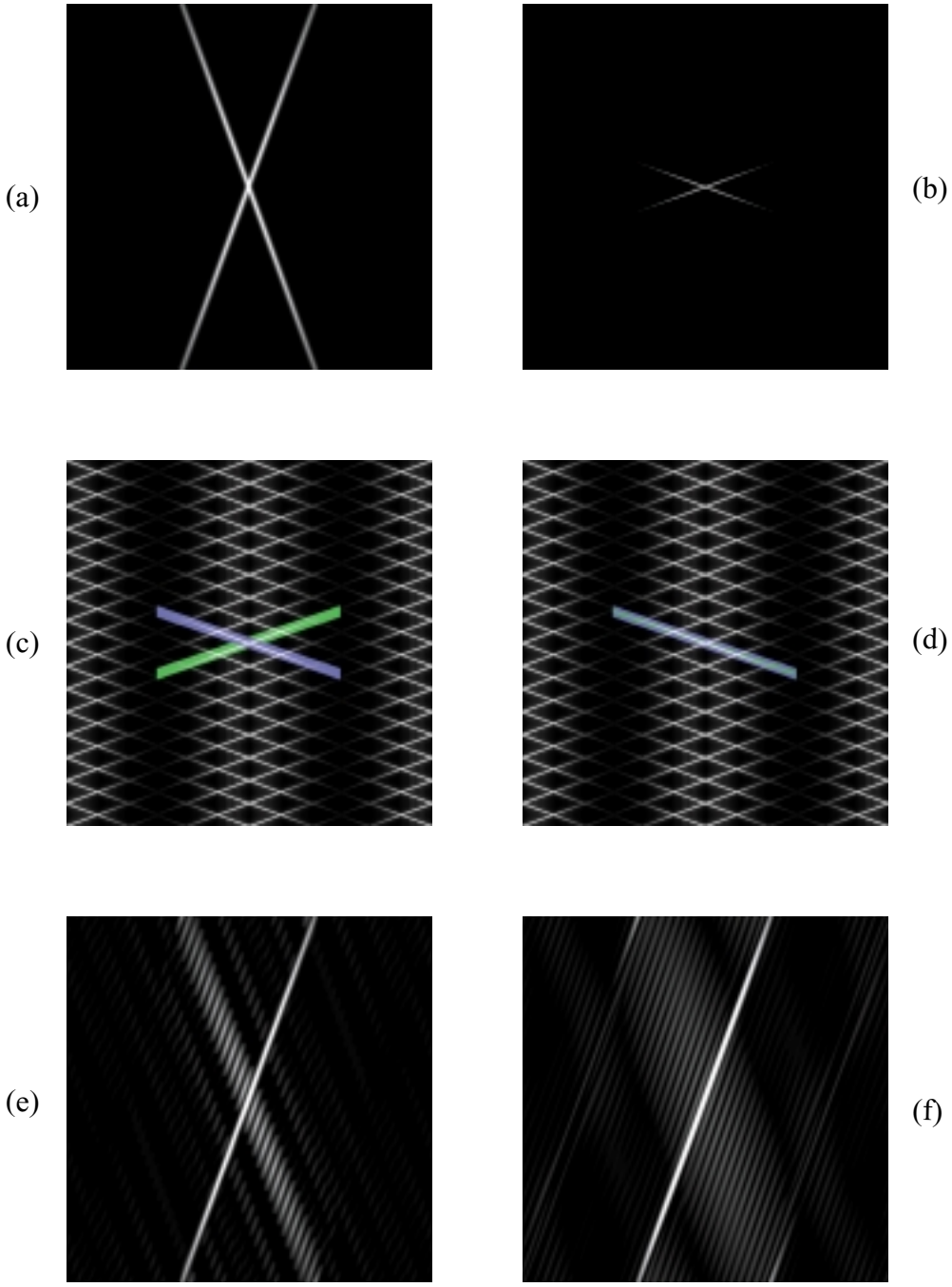


Figure 6-2: **(a)** *su* slice of two features. **(b)** Frequency domain power spectrum of the features. **(c)** Frequency domain power spectrum, colored boxes represent possible reconstruction filters. **(d)** Wide aperture reconstruction corresponds to thinner green box. **(e)** Result of small aperture reconstruction. **(f)** Result of large aperture reconstruction.

Chapter 7

Rendering

This chapter describes three methods for efficiently rendering dynamically reparameterized light fields (Sections 7.1, 7.2, and 7.3). In addition, I describe how to use the focal surface as an navigation aid in Section 7.4; this shows how focus is a powerful cue for selecting a particular depth in the scene.

As in the lumigraph and light field systems, I construct a desired image by querying rays from the ray database. Given arbitrary cameras, camera surfaces, and focal surfaces, one can ray-trace the desired image. If the desired camera, data cameras, camera surface, and focal surface are all planar, then a texture mapping approach can be used similar to that proposed by the lumigraph system. I extend the texture mapping method using multi-texturing for rendering with arbitrary non-negative aperture filters, including bilinear and higher order filters.

7.1 Standard Ray Tracing

I first describe a ray tracing method for synthesizing an image from a dynamically reparameterized light field. Given a bundle of rays from the center of projection and through each pixel of a desired image. We desire to calculate the geometry of each of these rays, and then calculate the color value that each one is assigned when reconstructing using the dynamically reparameterized light field.

The intersection techniques are those used in standard ray tracing. In the following

description, a ray $r = (s, t, u, v)$ has a color $c(r) = c(s, t, u, v)$. Likewise, a pixel (x, y) in the desired image has a color $c(x, y)$. Let K be the desired pinhole camera with a center of projection o and pixels (x, y) on its image plane. Let $w(x, y)$ be the aperture weighting function, where δ is the width of the aperture.

```
Initialize the frame buffer to black
For each pixel (x, y) in desired camera K
    r := the ray through o and (x, y)
    Intersect r with C to get (s', t')
    Intersect r with F to get (f, g)_F
    R_C := a polygon on C defined by {(s' ± δ/2, t' ± δ/2)}
    For each data camera D_{s,t} within R_C
        (u, v) := M_{s,t}^{F→D}(f, g)_f
        weight := w(s' − s, t' − t)
        c(x, y) := c(x, y) + weight * c(s, t, u, v)
```

7.2 Memory Coherent Ray Tracing

Next, I will describe a memory coherent ray tracing method that can take advantage of standard texture mapping hardware. Instead of rendering pixel by pixel in the desired image, in this approach one renders the contribution of each data camera sequentially. This causes each pixel in the desired image to be overwritten many times, but means the system only has to page in each desired camera image once. I use the same notation as in the previous section.

```
Initialize the frame buffer to black
For each data camera D_{s,t}
    R_C := a polygon on C defined by {(s ± δ/2, t ± δ/2)}
    R_K := projection of R_C onto the desired image plane
    For each pixel (x, y) within R_K
        r := the ray through o and (x, y)
        Intersect r with C to get (s', t')
        Intersect r with F to get (f, g)_F
        (u, v) := M_{s,t}^{F→D}(f, g)_f
        weight := w(s' − s, t' − t)
        c(x, y) := c(x, y) + weight * c(s, t, u, v)
```

7.3 Texture Mapping

Although the ray tracing method is simple to understand and easy to implement, there are more efficient methods for rendering when the camera surface, image surface, and focal surface are planar. I have extended the lumigraph texture mapping approach [7] to support dynamic reparameterization. This technique renders the contribution of each data camera $D_{s,t}$ using multi-texturing and an accumulation buffer [9]. This method works with arbitrary non-negative aperture weighting functions.

Multi-texturing, supported by Microsoft Direct3D 7's texture stages [16], allows a single polygon to have multiple textures and multiple projective texture coordinates. At each pixel, two sets of texture coordinates are calculated, and then two texels are accessed. The two texels are then multiplied, and this result is stored in the frame buffer. The frame buffer is written using the Direct3D alpha mode "source + destination," which makes the frame buffer act as a 8-bit, full-color accumulation buffer.

My rendering technique is illustrated in Figure 7-1. For each camera $D_{s,t}$, a rectangular polygon $R^C_{s,t}$ is created on the camera surface with coordinates $\{(s \pm$

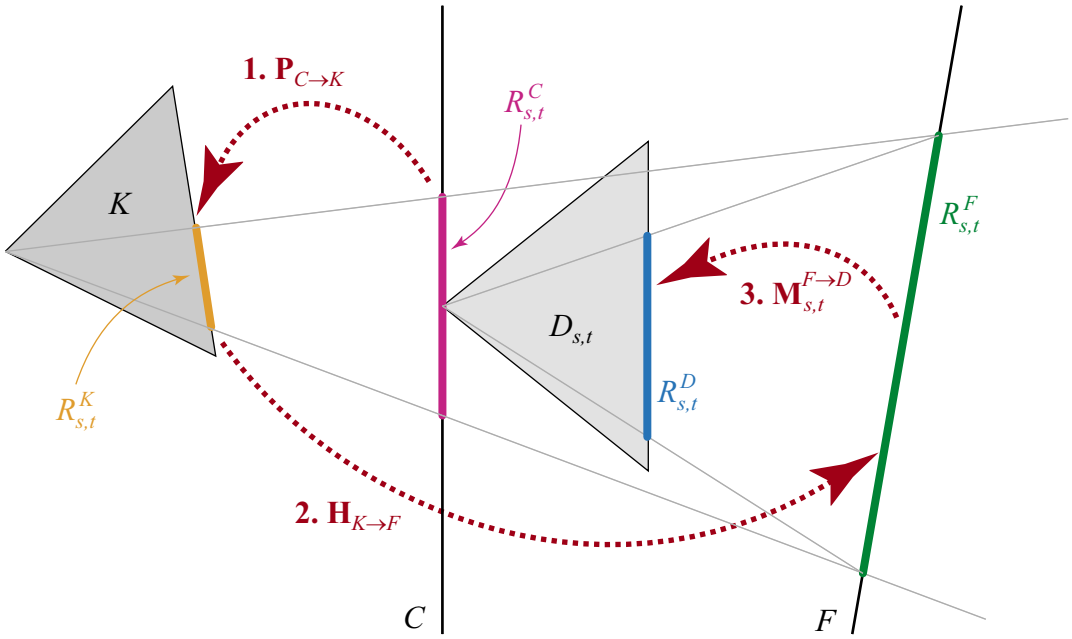


Figure 7-1: Projection matrices and planar homographies allow me to render the image using texture mapping on standard PC rasterizing hardware.

$\delta/2, t \pm \delta/2)\}$. I then project this polygon on to the desired camera K's image plane using a projection matrix $\mathbf{P}_{C\to K}$, giving a polygon $R^K_{s,t}$. This polygon $R^K_{s,t}$ represents the region of the image plane which uses samples from data camera $D_{s,t}$. That is, only pixels inside polygon $R^K_{s,t}$ use texture from data camera $D_{s,t}$.

The polygon $R^K_{s,t}$ is then projected onto the focal plane F using a planar homography $\mathbf{H}_{K\to F}$, a 3x3 matrix which changes one projective 2-D basis to another. This projection is done from the desired camera K's point of view. The resulting polygon $R^F_{s,t}$ lies on the focal plane F. Finally, the mapping $\mathbf{M}^{F\to D}_{s,t}$ is used to calculate the (u, v) pixel values for the polygon. This gives a polygon $R^D_{s,t}$, which represents the (u, v) texture coordinates for polygon $R^F_{s,t}$.

Many of these operations can be composed into a single matrix, which takes polygon $R^C_{s,t}$ directly to texture coordinates $R^D_{s,t}$. This matrix $\mathbf{M}^{C\to D}_{s,t}$ can be written as $\mathbf{M}^{C\to D}_{s,t} = \mathbf{M}^{F\to D}_{s,t}\mathbf{H}_{K\to F}\mathbf{P}_{C\to K}$.

This process gives the correct rays (s, t, u, v), but the appropriate weights from the aperture filter are still required. Because we draw a polygon with the shape of

the aperture filter, one must simply modulate the texture $D_{s,t}$ with the aperture filter texture A. For $D_{s,t}$ the projective texture coordinates $R^D_{s,t}$ are used; for the aperture filter A, the texture coordinates $\{(\pm\delta/2, \pm\delta/2)\}$ are used.

```
Initialize the frame buffer to black
For each data camera D_{s,t}
    R^C_{s,t} := polygon on C defined by {(s ± δ/2, t ± δ/2)}
    R^D_{s,t} := M^{F→D}_{s,t} H_{K→F} P_{C→K} R^C_{s,t}
    Render R^C_{s,t} using...
        texture D_{s,t}
        projective texture coordinates R^D_{s,t}
        modulated by aperture texture A
    Accumulate rendered polygon into frame buffer
```

Using this method, it is possible to render dynamically reparameterized light fields in real-time using readily available PC graphics cards that support multi-texturing. Frame rate decreases linearly with the number of data cameras that fit inside the aperture functions, so narrow apertures render faster. Vignetting, where the edge of the reconstructed images fade to black, occurs near the edges of the camera surface when using wide filters. An example of vignetting can be seen in Figure 4-4.

7.4 Using the Focal Surface as a User Interface

In typical light field representations, there is no explicit depth information. This makes it difficult to navigate about an object using a keyboard or mouse. For example, it can be hard to rotate the camera about an object when the system doesn't know where the object is located in space. Camera centered navigation is considerably simpler to use. I have found the focal surface can be used to help navigate about an object in the light field. When the focal surface is moved so that a particular pixel p belonging to that object is in focus, one can find the 3-D position P of p using the

equation $P = \mathbf{H}_{K \to P} p$. Once the effective 3-D position of the object is known, it is simple to rotate (or some other transformation) relative to that point.

Chapter 8

Autostereoscopic Light Fields

My flexible reparameterization framework allows for other useful reorganizations of light fields. One interesting reparameterization permits direct viewing of a light field. The directly-viewed light field is similar to an integral or lenticular photograph. In integral photography, a large array of small lenslets, usually packed in a hexagonal grid, is used to capture and display an autostereoscopic image [17, 23]. Traditionally, two identical lens arrays are used to capture and display the image: this requires difficult calibration and complicated optical transfer techniques [25]. Furthermore, the viewing range of the resulting integral photograph mimics the configuration of the capture system. Holographic stereograms [11] also can present directly-viewed light fields, although the equipment and precision required to create holograph stereograms can be prohibitive. Using dynamic reparameterizations, it is possible to capture a scene using light field capture hardware, reparameterize it, and create novel 3-D displays that can be viewed with few restrictions. This makes it much easier to create integral photographs: a light field is much easier to collect and a variety of integral photographs can be created from the same light field.

In an integral photograph, a single lenslet acts as a view-dependent pixel, as seen in Figure 8-1. For each lenslet, the focal length of the lens is equal to the thickness of the lens array. A reparameterized light field image is placed behind the lens array, such that a subset of the ray database lies behind each lenslet. When the lenslet is viewed from a particular direction, the entire lenslet takes on the color of a single

point in the image. To predict which color is seen from a particular direction, I use a paraxial lens approximation [22]. I draw a line parallel to the viewing direction which passes through the principle point of the lenslet. This line intersects the image behind the lenslet at some point; this point determines the view-dependent color. If the viewing direction is too steep, then the intersection point might fall under a neighboring lenslet. This causes a repeating "zoning" pattern which can be eliminated by limiting the viewing range or by embedding blockers in the lens array [17].

Since each lenslet acts as a view-dependent pixel, the entire lens array acts as a view-dependent, autostereoscopic image. The complete lens array system is shown in Figure 8-4. Underneath each lenslet is a view of the object from a virtual camera located at each lenslet's principal point.

To create the autostereoscopic image from a dynamically reparameterized light field, I position a model of the lens array into the light field scene. This is analogous to positioning a desired camera to take a standard image. Then an array of tiny sub-images is created, each the size of a lenslet. Each sub-image is generated using a dynamically-reparameterized-light-field-system, with the focal surface passing through the object of interest. Each sub-image is taken from the principal point of a lenslet, with the image plane parallel to the flat face of the lens array. The sub-images are then composited together to create a large image, as in Figure 8-3, which can be placed under the lens array. When viewed with the lens array, one would see an autostereoscopic version of the scene shown in Figure 8-2.

The placement and orientation of the lens array determines if the viewed light field appears in front or behind the display. If the lens array is placed in front of the object, then the object appears behind the display. Because the lens array image is rendered from a light field and not directly from an integral camera, I place the lens array image behind the captured object, and the object appears to float in front of the display.

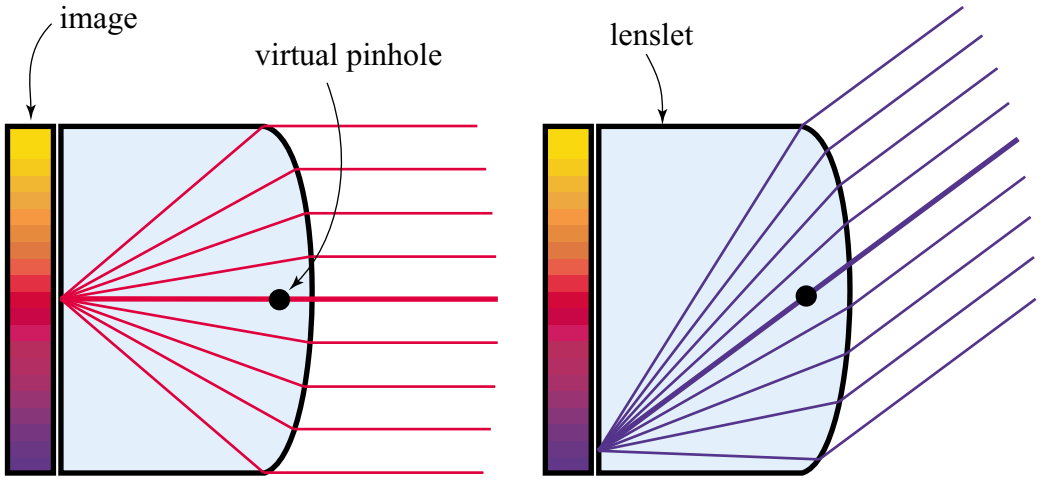


Figure 8-1: Each lenslet in the lens array acts as a view-dependent pixel.

Figure 8-2: A autostereoscopic reparameterized version of this scene is shown in Figure 8-3.

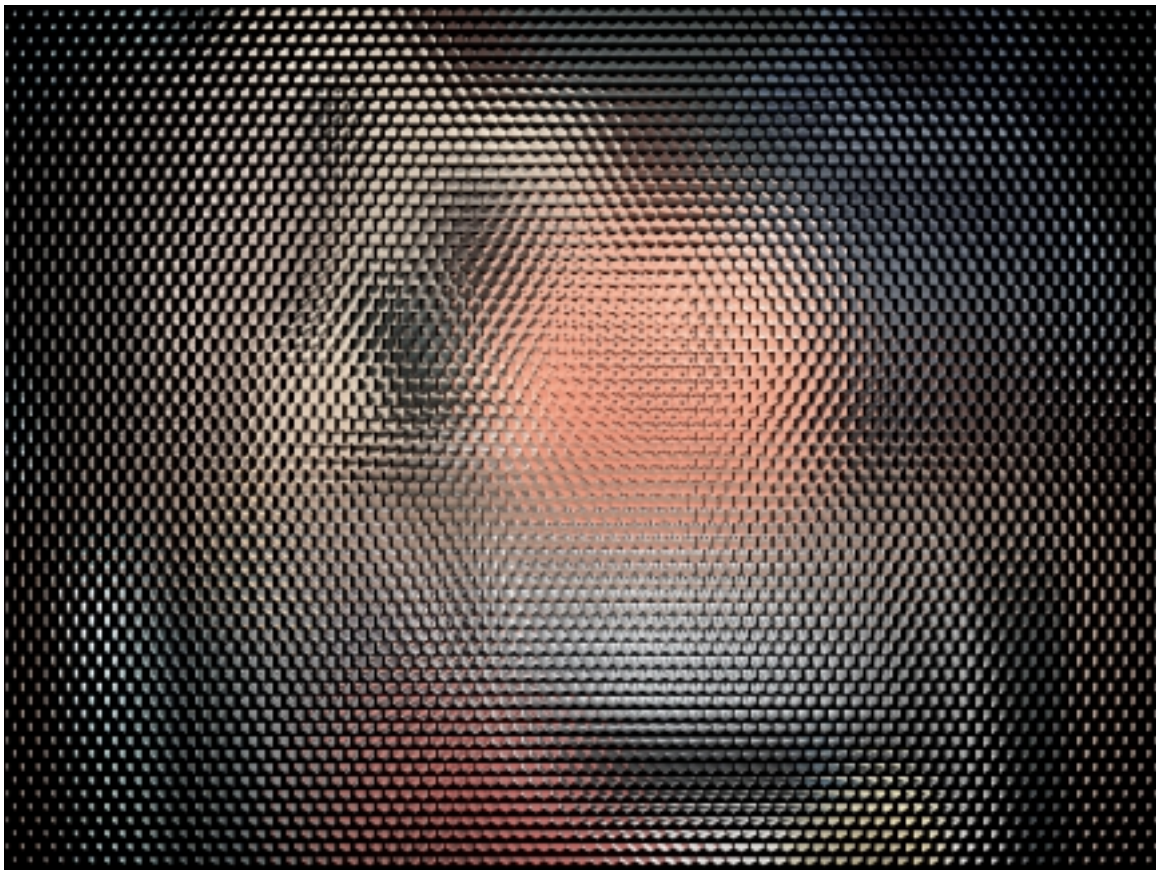


Figure 8-3: An autostereoscopic image that can be viewed with a hexagonal lens array.

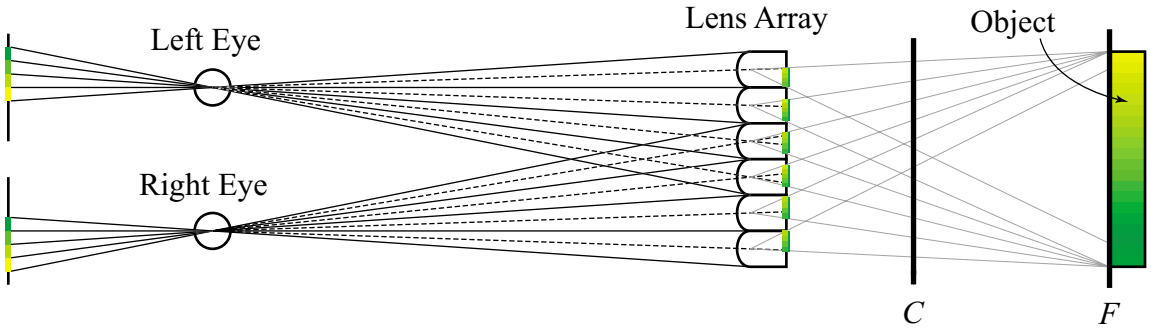


Figure 8-4: A light field can be reparameterized into a integral photograph.

Chapter 9

Results

In this chapter I will describe how we captured our light fields (Section 9.1), how we calibrated out cameras (Section 9.2), and some particulars about our renderers (Section 9.3).

9.1 Capture

The light field data sets shown in this thesis were created as follows. The tree data set, with 256 input images, was rendered in Povray 3.1, a public-domain ray-tracer. It is composed of 256 (16x16) images with resolutions of 320x240. For each set, I captured either 256 (16x16) or 1024 (32x32) pictures.

I have built a light field capture system using a digital camera mounted to a robotic motion platform. The captured data sets were acquired with an Electrim EDC1000E CCD camera (654 x 496 with gRBG Bayer pattern) with a fixed-focal length 16mm lens mounted on an X-Y motion platform from Arrick Robotics (30" x 30" displacement). The 16mm lens was specially designed to reduce geometric distortion. I used a 2x2 color interpolation filter to create a full color image from the Bayer pattern image. Finally, I resampled the raw 256 654 x 496 images down to 327 x 248 before using them as input to the renderer, to produce better color resolution.

The EDC1000E is quite noisy, with a large percentage of hot pixels. To reduce the effect of the hot pixels, I captured 10 dark images and then averaged them to produce

an average dark image. This dark image was subtracted from each subsequent image taken with the camera during the light field capture. Then, I also captured a flat field image for each capture session. To make the flat field image, I covered the lens with a diffusing white paper, and into the lens I shined a bulb of the same type used to illuminate the scene. This captured an image that I could use to calibrate the color dyes of the sensor, as well as the value between each pixel. Thus, I took this flat field image and assigned it an arbitrary color value. Then, each pixel in every subsequent image was scaled by the arbitrary color value divided by the value stored in the flat field image for that pixel.

9.2 Calibration

To calibrate the camera, I originally used a Faro Arm (a sub-millimeter accurate contact digitizer) to measure the 3-D spatial coordinates (x, y, z) of the centers of a calibration pattern on two perpendicular planes filling the camera's field of view. Then, I found the centroids of these dots in images and fed the 24 5-tuples (x, y, z, i, j) into the Tsai-Lenz camera calibration algorithm [24] which reported focal length, CCD sensor element aspect ratio, principle point, and extrinsic rotational orientation. I ignored radial lens distortion, which was reported by calibration as less than 1 pixel per 1000 pixels.

I have found that a strict calibration step is not necessary. A method has been developed which allows my data to be rectified along the two translation axes. Given that the initial images are approximately aligned to only a few degrees off-axis, only a two-axis image rectification is required. This alignment is guaranteed by careful orientation of the camera with the X-Y platform. The method relies on finding the epipolar planes induced by the horizontal and vertical camera motion. The epipoles are then rectified by a rotation to the line at infinity. The focal length is then estimated by either measuring the scene and using similar triangles, or by modifying the focal length using the real-time viewer until the images look undistorted. In my experiments, it is fairly easy to recover the focal length using this method. Because

the light fields are defined up to a scale factor, I assume that the camera spacing is a unit length, and then the focal length is defined in these units. The principal point is assumed to be the center of each image.

9.3 Rendering

I have developed two renderers, a real-time renderer and an off-line renderer. The real-time renderer uses planar homographies to efficiently render by using readily available PC rasterizers for Direct3D 7. The off-line renderer is more flexible and permits non-planar focal and camera surfaces, as well as per pixel focal surfaces and apertures. For the autostereoscopic images, I print at 300dpi on a Tektronics Phaser 440 dye sublimation printer and use a Fresnel Technologies #300 Hex Lens Array, with approximately 134 lenses per square inch [6].

Figure 9-1 shows the real-time user interface. One can change many variables of the system at run time, including camera rotation and position, focal plane location, focal length of the camera, and aperture size. The user navigates perpendicular to the view direction by dragging with the left mouse button. Holding down shift while dragging vertically allows the user to translate into and out of the scene, along the view direction. The dragging with the right button allows the user to rotate the camera, either about the object that the user clicked on when beginning the drag or about the eye (the user sets this in the "Rotate About..." frame). The user can save screen shots, as well as use the current camera position to feed as input to the autostereoscopic renderer.

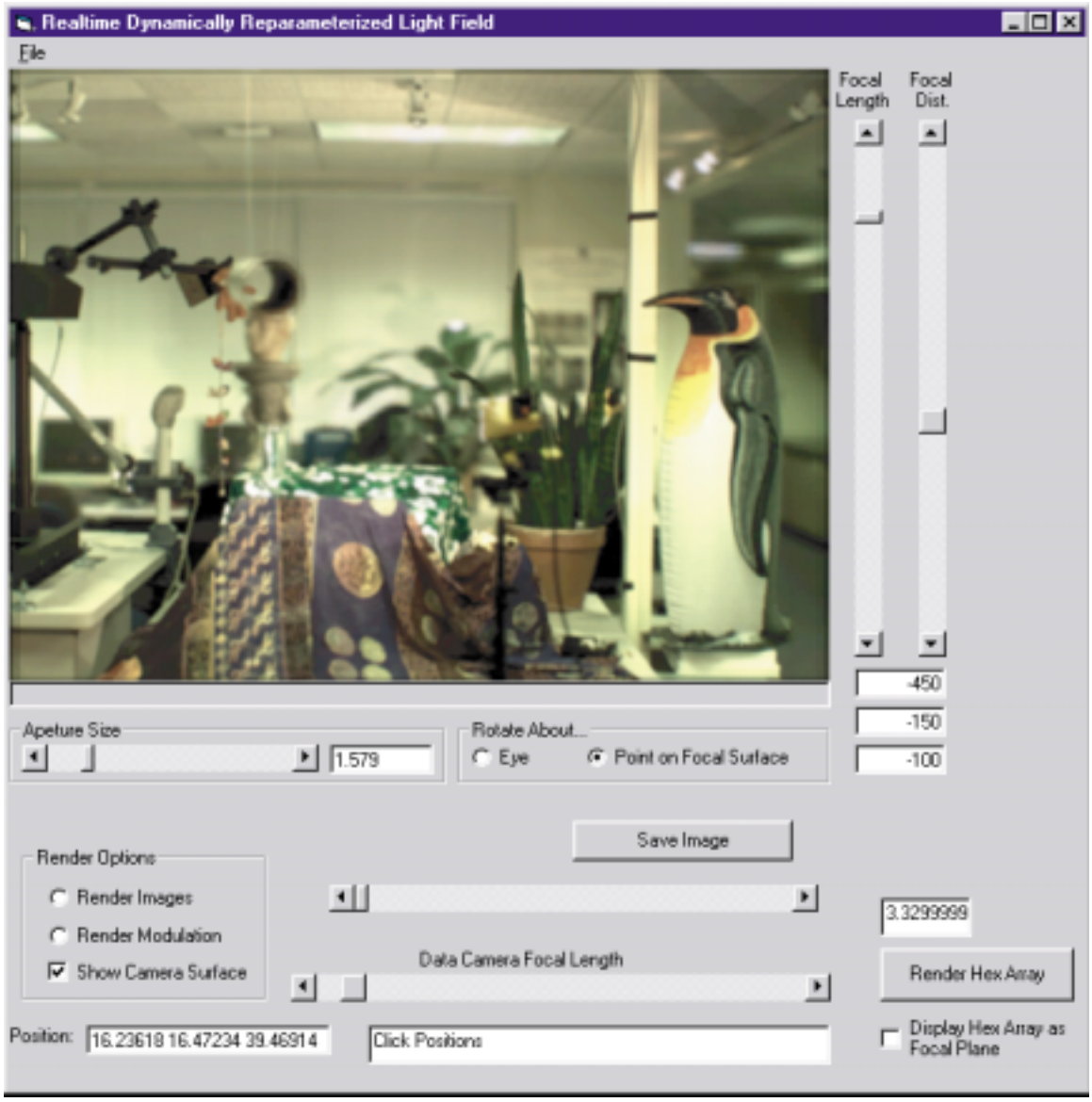


Figure 9-1: The real-time-dynamically-reparameterized-light-field-renderer allows the user to change focal plane distance, focal length, camera position, camera rotation, aperture size, and data camera focal length.

Chapter 10

Future Work and Conclusion

In this thesis, I have described a new parameterization that further develops the light field and lumigraph image-based rendering methods and extends their utility. I have presented a dynamic parameterization that permits 1) interactive rendering of moderately sampled light fields with significant, unknown depth variation and 2) low-cost, passive autostereoscopic viewing. Using a dynamic reparameterization, these techniques can be used to interactively render photographic effects such as variable focus and depth-of-field within a light field. The dynamic parameterization works independently of scene geometry and does not require actual or approximate geometry of the scene for focusing. I explored the frequency domain and ray-space aspects of dynamic reparameterization, and present an interactive rendering technique that takes advantage of today's commodity rendering hardware.

Previous light field implementations have addressed focusing problems by 1) using scenes that were roughly planar, 2) using aperture filtering to blur the input data, or 3) using approximate geometry for depth-correction. Unfortunately, most scenes can not be confined to a single plane, aperture filtering can not be undone or controlled at run time, and approximate surfaces can be difficult to obtain. I have presented a new parameterization that enables dynamic, run-time control of the sample reconstruction. This allows the user to modify focus and depth-of-field dynamically. This new parameterization allows light fields to capture data sets with depth, and helps bring the graphics community closer to truly photorealistic virtual reality. In addi-

tion, I have presented a strategy for creating directly viewable light fields. These passive, autostereoscopic light fields can be viewed without head-mounted hardware by multiple viewers simultaneously.

There is much future work to be done. I would like to develop an algorithm for optimally selecting a focal plane, perhaps using auto-focus techniques as in consumer, hand-held cameras. Currently, a human must place the focal plane manually. In addition, I believe there is promise in using these techniques for passive depth-from-focus or depth-from-defocus vision algorithms [19]. In the left column of Figure 10-1, I have created two images with different focal surfaces and a large aperture. I then apply a gradient magnitude filter to these images, which gives the output to the right. These edge images describe where in-focus, high-frequency energy exists. The out-of-focus regions have little high-frequency energy, whereas the regions in focus do. Since only structure very near the focal planes are in focus, I know that the in-focus regions identify regions where there is structure. Thus, if I convolve a high-pass filter over these narrow depth of field images and then identify the regions with high-frequency energy, I find the regions in space where structure exists. Then, I use the plane (or set of planes) which gives rise to the image with the most high-frequency energy: this plane is the auto-focus plane. Likewise, I could take the n-best planes for a multiple focal plane rendering (see Section 4.3).

Objects with little high-frequencies, even when they are in focus, such as flat regions with little texture, are not be detected by this process. However, objects with little high-frequency look as good if they are out of focus as if they are in focus. Whereas using computer vision techniques to find depth from a set of images would have to further analyze these ambiguous regions, I do not have to delve further since several values are good enough: I simply take the best one that I find.

I would also like to experiment with depth-from-defocus by comparing two images with slightly different apertures. Because my system allows the user to quickly create images with variable depth of field and focus, experimentation in this area should be quite possible.

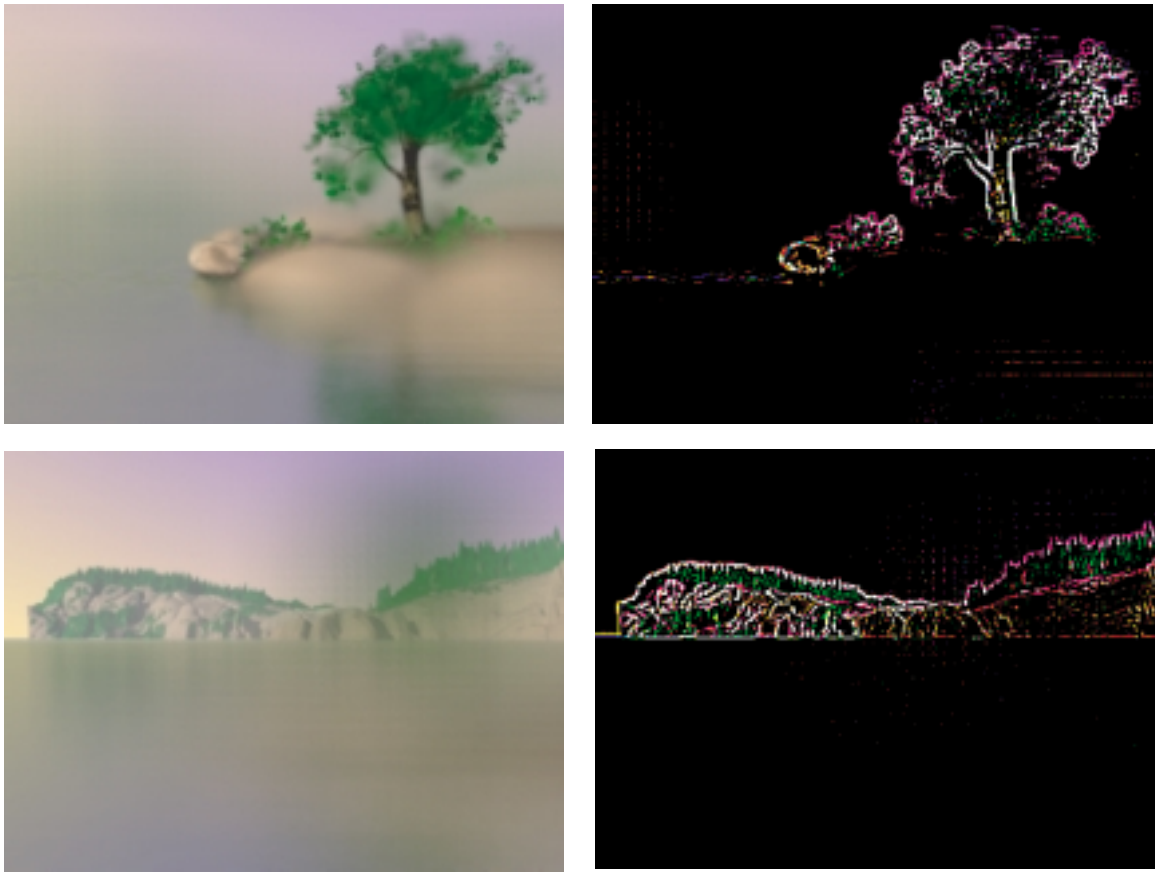


Figure 10-1: I believe my techniques can be used in a depth-from-focus or depth-from-defocus vision system. By applying a gradient magnitude filter on an image created with a wide aperture, I can detect in-focus regions.

Appendix A

MATLAB code for Frequency-Domain Analysis

This section provides the MATLAB code to produce the frequency domain analysis figures of Chapter 6. The code assumes that the EPI domain and frequency domain is of resolution 512 by 512. Because this is graphics code that draws to MATLAB bitmap arrays, the origin is in the upper left corner of the images. Also, I have used x and y axis coordinates, instead of s and f coordinates. The x axis is identical to the s axis, but the y axis points in the opposite direction as the f axis.

A.1 Making the EPI

```
function z = MakeEpi(y0, x0, a0)

% MakeEpi takes in 3 arrays of length n, which represents n EPI
% features.  For EPI feature i, the feature is centered about
% (x0(i),y0(i)) and has an angle a0(i)

% size of the EPI
height = 512;
width = 512;

z = zeros(height, width);  % initialize the image buffer
```

```
% note that we are working in a computer graphics coordinate system,
% where the origin is in the upper left.  The Positive X axis points
% to the right, and positive Y axis points downward

% We can write equations for epi lines as (F*x + G*y + H = 0)
% If we don't wish to represent horizontal lines,
% we can divide by f to get (x + (G/F)*y + H/F = 0)
% This we can rewrite as (x + A*y + B = 0)

% Given a point (x0,y0) and an angle a0, we can solve for A and B as
% follows:
% (eq1:) x0    + A*y0           + B = 0
% (eq2:) x0+cos(a0) + A*(y0-sin(a0)) + B = 0
% Solving for A and B gives us
% A = cot(a0)
% B = -cot(a0)*y0-x0

A = cot(a0);
B = -cot(a0).*y0-x0;

% We also need the line perpendicular to the one above, that also
% passes through (x0,y0).  We can find this by using a new angle
% b0 = a0 + pi/2.  Note that we will not allow this line to be
% vertical.  This gives us a line equation (C*x + y + D = 0)
% (eq1:) C*x0  + y0  + D = 0
% (eq2:) C*(x0+cos(a0+pi/2)) + y0-sin(a0+pi/2) + D = 0
% Solving for C and D gives us
% C = tan(a0+pi/2)
% D = -y0 - x0*tan(a0+pi/2)

C = tan(a0+pi/2);
D = -y0-x0.*tan(a0+pi/2);

% A Gaussian distribution with the mean centered at 0 has the form
% P(x) = 1/(s*sqrt(2*pi)) * exp(-x^2/(2*s^2))

% for our Epi features, we will represent it at the multiplication
% of two perpendicular gaussian distributions.  One gaussian will
% spread out across the line C*x+y+D=0 with a large standard deviation.
% This represents the slowly changing view dependence of the feature.
% Perpendicular to this, the other gaussian will spread out across the
% line x+A*y+B=0 with a small standard deviation.  This represents the
% boundaries of the point feature, so the epi feature is narrow in this
% direction.
```

```
sAB = 2;
sCD = 200;

kAB = 1/(sAB*sqrt(2*pi)); % scale factor for the AB gaussian
kCD = 1/(sCD*sqrt(2*pi)); % scale factor for the CD gaussian
eAB = 1/(2*sAB^2); % multiplier for the exponent in the AB gaussian
eCD = 1/(2*sCD^2); % multiplier for the exponent in the CD gaussian

for j=1:height
   for i=1:width
      tAB = i+A*j+B;
      tCD = C*i+j+D;
      z(j,i) = sum((kAB*exp(-tAB.^2*eAB)) .* (kCD*exp(-tCD.^2*eCD)));
   end
end
```

A.2 Viewing the EPI

```
function ShowEpi(z,n)
% ShowEpi will draw the EPI stored in the array z.  This function
% scales the intensity array before drawing it so that it displays
% correctly.  The parameter n is the number of EPI features in the
% EPI; this helps figure out how to correctly scale the EPI;

t = ones(size(z))/n;
s = min(t,z/max(max(z)));
imshow(real(s*n));
axis equal;
```

A.3 Making the Filter

```
function F = MakeFilter( angle, y_width, x_width )
% MakeFilter will create the ideal low-pass filter for reconstruction
% in the frequency domain.  The parameter angle gives the shearing angle
% of the original feature in EPI space to create the filter for.  y_width
% is the width of the filter in the y dimension, and x_width is the width
% of the filter in the x dimension

w = 512; % size of the EPI, frequency domain, and thus the filter
```

```
h = 512;

F = zeros(h,w);  % initialize the filter to all 0

% angle is in the EPI space, so rotate it by 90
% degrees to get it into the frequency domain
% coordinate system
a = pi/2 + angle;

% create corners of the rectangular filter
% the filter is sheared by 'a'
Pt1 = [-y_width/2-(sin(a)*x_width/2) -x_width/2];
Pt2 = [-y_width/2+(sin(a)*x_width/2) x_width/2];
Pt3 = [y_width/2+(sin(a)*x_width/2)  x_width/2];
Pt4 = [y_width/2-(sin(a)*x_width/2)  -x_width/2];

% assign those corners to the Polygon structure
PolyX = [Pt1(2) Pt2(2) Pt3(2) Pt4(2)];
PolyY = [Pt1(1) Pt2(1) Pt3(1) Pt4(1)];

% translate to the center
PolyX = PolyX+w/2;
PolyY = PolyY+h/2;

% fill in the bitmap with filter values
% if inside the polygon, assign a 1
% if outside the polygon, assign a 0
% if on the polygon, assign a 0.5
[X,Y] = meshgrid(1:w,1:h);
F = inpolygon(X,Y,PolyX,PolyY);
```

A.4 Filtering the Frequency Domain

```
function ZF = FTFilter(Z,F)

ZF = Z.*fftshift(F);
```

A.5 Downsampling the EPI

```
function out = DownsampleEpi(in, nh, nw)
%in is the array to downsample
%nh is number of times to downsample the height dimension
%nw is number of times to downsample the width dimension
%size(out) == size(in), but will have downsampled samples set to 0

[in_h in_w] = size(in);
th = in_h/nh; % there will be th samples in the height dimension
tw = in_w/nw; % there will be tw samples in the width dimension

T = zeros(nh,nw);
T(1,1) = 1;
TT = repmat(T,th,tw);
out = TT.*in;
```

A.6 Viewing the Frequency Domain Image

```
function ShowFT(Z,n)
% ShowFT will show the frequency domain image in a figure window
% Z is the frequency domain image, and n is the number of EPI
% features in the image.  This function takes the magnitude of
% each complex element in the image, and rescales it so that
% it displays correctly

R = abs(fftshift(Z));
m = max(max(R));
imshow(R,[.01*m (1/n)*m]);
```

A.7 Viewing the Frequency Domain Image and Filter

```
function ShowFilterAndFT(Z,f,n,FilterColor)
% ShowFT will show the frequency domain image in a figure window
% along with the filter, in blue.
% Z is the frequency domain image, and n is the number of EPI
% features in the image.  This function takes the magnitude of
% each complex element in the image, and rescales it so that
% it displays correctly.
% FilterColor is the [r g b] color of the filter

R = abs(fftshift(Z));
m = max(max(R));
imshow(R,[.01*m (1/n)*m]);

t = ones(size(Z))/n;
R = abs(fftshift(Z));
s = n*min(t,R/max(max(R)));
[h, w, d] = size(f);

FCred = ones(h,w,d);
FCgre = ones(h,w,d);
FCblu = ones(h,w,d);
for i=1:d
   FCred(:,:,i) = FCred(:,:,i)*FilterColor(i,1);
FCgre(:,:,i) = FCgre(:,:,i)*FilterColor(i,2);
   FCblu(:,:,i) = FCblu(:,:,i)*FilterColor(i,3);
end


C = zeros(h,w,3); % make a full color array
S = repmat(s,[1 1 d]);
C(:,:,1) = min(1,sum(f.*FCred.*(1-S)+S,3));
C(:,:,2) = min(1,sum(f.*FCgre.*(1-S)+S,3));
C(:,:,3) = min(1,sum(f.*FCblu.*(1-S)+S,3));
imshow(C);
axis equal;
```

A.8 Creating the Components of the Figure 6-1

```
function ShowFilterAndFT(Z,f,n,FilterColor)
% ShowFT will show the frequency domain image in a figure window
% along with the filter, in blue.
% Z is the frequency domain image, and n is the number of EPI
% features in the image.  This function takes the magnitude of
% each complex element in the image, and rescales it so that
% it displays correctly.
% FilterColor is the [r g b] color of the filter

R = abs(fftshift(Z));
m = max(max(R));
imshow(R,[.01*m (1/n)*m]);

t = ones(size(Z))/n;
R = abs(fftshift(Z));
s = n*min(t,R/max(max(R)));
[h, w, d] = size(f);

FCred = ones(h,w,d);
FCgre = ones(h,w,d);
FCblu = ones(h,w,d);
for i=1:d
   FCred(:,:,i) = FCred(:,:,i)*FilterColor(i,1);
FCgre(:,:,i) = FCgre(:,:,i)*FilterColor(i,2);
   FCblu(:,:,i) = FCblu(:,:,i)*FilterColor(i,3);
end


C = zeros(h,w,3); % make a full color array
S = repmat(s,[1 1 d]);
C(:,:,1) = min(1,sum(f.*FCred.*(1-S)+S,3));
C(:,:,2) = min(1,sum(f.*FCgre.*(1-S)+S,3));
C(:,:,3) = min(1,sum(f.*FCblu.*(1-S)+S,3));
imshow(C);
axis equal;
```

A.9 Creating the Components of Figure 6-2

```
function ShowFilterAndFT(Z,f,n,FilterColor)
% ShowFT will show the frequency domain image in a figure window
% along with the filter, in blue.
% Z is the frequency domain image, and n is the number of EPI
% features in the image.  This function takes the magnitude of
% each complex element in the image, and rescales it so that
% it displays correctly.
% FilterColor is the [r g b] color of the filter

R = abs(fftshift(Z));
m = max(max(R));
imshow(R,[.01*m (1/n)*m]);

t = ones(size(Z))/n;
R = abs(fftshift(Z));
s = n*min(t,R/max(max(R)));
[h, w, d] = size(f);

FCred = ones(h,w,d);
FCgre = ones(h,w,d);
FCblu = ones(h,w,d);
for i=1:d
   FCred(:,:,i) = FCred(:,:,i)*FilterColor(i,1);
FCgre(:,:,i) = FCgre(:,:,i)*FilterColor(i,2);
   FCblu(:,:,i) = FCblu(:,:,i)*FilterColor(i,3);
end


C = zeros(h,w,3); % make a full color array
S = repmat(s,[1 1 d]);
C(:,:,1) = min(1,sum(f.*FCred.*(1-S)+S,3));
C(:,:,2) = min(1,sum(f.*FCgre.*(1-S)+S,3));
C(:,:,3) = min(1,sum(f.*FCblu.*(1-S)+S,3));
imshow(C);
axis equal;
```

Bibliography

[1] R.C. Bolles, H.H. Baker, and D.H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *IJCV*, 1(1):7–56, 1987.

[2] Emilio Camahort, Apostolos Lerios, and Donald Fussell. Uniformly sampled light fields. *Eurographics Rendering Workshop 1998*, pages 117–130, 1998.

[3] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. *Proceedings of SIGGRAPH 2000*, pages 307–318, July 2000. ISBN 1-58113-208-5.

[4] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. *Proceedings of SIGGRAPH 93*, pages 279–288, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):137–145, July 1984. Held in Minneapolis, Minnesota.

[6] Fresnel Technologies. *#300 Hex Lens Array*, 1999. http://www.fresneltech.com.

[7] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *SIGGRAPH 96*, pages 43–54.

[8] Xianfeng Gu, Steven J. Gortler, and Michael F. Cohen. Polyhedral geometry and the two-plane parameterization. *Eurographics Rendering Workshop 1997*, pages 1–12, June 1997.

[9] Paul E. Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. *SIGGRAPH 90*, 24(4):309–318.

[10] Michael Halle. Multiple viewpoint rendering. *Proceedings of SIGGRAPH 98*, pages 243–254, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[11] Michael W. Halle. The generalized holographic stereogram. page 134, February 1991.

[12] Wolfgang Heidrich, Philipp Slusalek, and Hans-Peter Seidel. An image-based model for realistic lens systems in interactive computer graphics. *Graphics Interface '97*, pages 68–75.

[13] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. *Proceedings of SIGGRAPH 2000*, pages 297–306, July 2000. ISBN 1-58113-208-5.

[14] Marc Levoy and Pat Hanrahan. Light field rendering. *SIGGRAPH 96*, pages 31–42.

[15] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. *Proceedings of SIGGRAPH 95*, pages 39–46, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

[16] Microsoft Corporation. *Microsoft DirectX 7.0*, 1999. http://www.microsoft.com/directx.

[17] Takanori Okoshi. *Three-Dimensional Imaging Techniques*. Academic Press, Inc., New York, 1976.

[18] Paul Haeberli. A multifocus method for controlling depth of field. Technical report, October 1994. http://www.sgi.com/grafica/depth/index.html.

[19] A.P. Pentland. A new sense for depth of field. *PAMI*, 9(4):523–531, July 1987.

[20] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. *Proceedings of SIGGRAPH 99*, pages 299–306, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[21] Peter-Pike Sloan, Michael F. Cohen, and Steven J. Gortler. Time critical lumigraph rendering. *1997 Symposium on Interactive 3D Graphics*, pages 17–24, April 1997. ISBN 0-89791-884-3.

[22] Warren J. Smith. *Practical Optical System Layout*. McGraw-Hill, 1997.

[23] R. F. Stevens and N. Davies. Lens arrays and photography. *Journal of Photographic Science*, 39(5):199–208, 1991.

[24] R.Y. Tsai. An efficient and accurate camera calibration technique for 3-d machine vision. In *CVPR*, pages 364–374, 1986.

[25] L. Yang, M. McCormick, and N. Davies. Discussion of the optics of a new 3d imaging system. *Applied Optics*, 27(21):4529–4534, 1988.

[26] Z.Y. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.