Efficient Acquisition, Representation, and Rendering of Light Fields

Hartmut Schirmacher

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Eingereicht am: 10.06.2003

Tag des Kolloquiums: 16.12.2003

Betreuender Hochschullehrer / Supervisor:
Prof. Dr. Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany

Gutachter / Reviewers:
Prof. Dr. Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany
Prof. Dr. Wolfgang Straßer, Universität Tübingen, Germany

Dekan / Dean:
Prof. Dr. Philipp Slusallek, Universität des Saarlandes, Saarbrücken, Germany

Hartmut Schirmacher
c/o Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken

Kontakt / Contact:: www.hartmut-schirmacher.de

Short Abstract. In this thesis we discuss the representation of three-dimensional scenes using image data (image-based rendering), and more precisely the so-called light field approach. We start with an up-to-date survey on previous work in this young field of research. Then we propose a light field representation based on image data and additional per-pixel depth values. This enables us to reconstruct arbitrary views of the scene in an efficient way and with high quality. Furthermore, we can use the same representation to determine optimal reference views during the acquisition of a light field. We further present the so-called free form parameterization, which allows for a relatively free placement of reference views. Finally, we demonstrate a prototype of the Lumi-Shelf system, which acquires, transmits, and renders the light field of a dynamic scene at multiple frames per second.
Note: An additional two-page abstract is included in this thesis, starting on page 145.

Kurzzusammenfassung. Diese Doktorarbeit beschäftigt sich mit der Repräsentierung dreidimensionaler Szenen durch Bilddaten (engl. *image-based rendering*, deutsch *bildbasierte Bildsynthese*), speziell mit dem Ansatz des sog. Lichtfelds. Nach einem aktuellen Überblick über bisherige Arbeiten in diesem jungen Forschungsgebiet stellen wir eine Datenrepräsentation vor, die auf Bilddaten mit zusätzlichen Tiefenwerten basiert. Damit sind wir in der Lage, beliebige Ansichten der Szene effizient und in hoher Qualität zu rekonstruieren sowie die optimalen Referenz-Ansichten bei der Akquisition eines Lichtfelds zu bestimmen. Weiterhin präsentieren wir die sog. Freiform-Parametrisierung, die eine relativ freie Anordnung der Referenz-Ansichten erlaubt. Abschließend demonstrieren wir einen Prototyp des Lumishelf-Systems, welches die Aufnahme, Übertragung und Darstellung des Lichtfeldes einer dynamischen Szene mit mehreren Bildern pro Sekunde ermöglicht.
Hinweis: Eine zusätzliche zweiseitige Zusammenfassung findet sich auf Seite 147 dieser Arbeit.

Acknowledgments

First of all, I would like to express my gratitude to my supervisor, Prof. Hans-Peter Seidel, Max-Planck-Institut für Informatik, Saarbrücken. He provided me with the basis for successful research, including the opportunity to pursue a topic as interesting as this, as well an an excellent working environment. Thanks also for applying a bit of "driving force" in some of the most critical moments during the years of this work.

Furthermore, I would like to thank Prof. Wolfgang Straßer, University of Tübingen, for acting as my second reviewer and sharing his time and expertise during the final phase of this thesis.

Special thanks also go to Prof. Wolfgang Heidrich, University of British Columbia, Vancouver, who guided and supervised me during my first Ph.D. years in Erlangen and Saarbrücken, and who helped me find my own way in this topic.

A lot of different projects have shaped this thesis. I am deeply indebted and grateful to my co-workers in these projects, especially Li Ming, Max-Planck-Institut für Informatik, and Christian Vogelgsang, University of Erlangen. We had many fruitful discussions and spent days and nights writing papers and implementing projects together, usually facing nearby submission deadlines. Furthermore, I was always in the lucky position to work with a group of extremely creative and competent colleagues, who happily shared their knowledge and helped wherever there was need, be it in theory or practice, in scientific or in personal matters. I cannot mention them all here, but the warmest thanks go to (in alphabetic order) Stefan Brabec, Katja Daubert, Georg Demme, Michael Gösele, Jan Kautz, Hendrik Lensch, Marcus Magnor, Christian Rössl, Annette Scheel, Philipp Slusallek, Marc Stamminger, Pere Pau Vázquez, and Jens Vorsatz.

I am also very grateful to the students who helped implementing some of our projects, and who helped acquiring or converting the necessary data. Thanks go to Hendrik Kück, Michael Repplinger, Pascal Schüler, Jan Uschok, and Markus Weber.

A project such as this doctoral thesis, in conjunction with many other practical issues of being a research scientist in an active and demanding research group, has brought me close to my physical and psychological limits many times. This would have been impossible to manage without the continuous and never-failing support of my family and my close friends. Thanks for always believing in me and supporting me.

Finally, I shall not forget to mention the support by my financial sponsors, being the *Deutsche Forschungsgemeinschaft* (`www.dfg.de`) under the project SFB 603/C2 during my first year in Erlangen, and the *Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V.* (`www.mpg.de`) during the final three years in Saarbrücken.

Contents

CHAPTER 1

Introduction

And God said: Let there be light!

Genesis, 1.3

Humans cannot perceive things “as they are”. The matter in our world cannot communicate its presence “just so”. We need to rely on quantities that we can sense, such as light and sound. The result of light interacting with an object is what can be perceived by the human visual system. Light reflected off a surface makes us see the surface, a perfect mirror can make us believe objects to be in a different place, and light scattered in the earth’s atmosphere creates our idea of a blue sky.

As much as in our everyday life, light has always been playing an important role in religion, philosophy, physics, and many other sciences. It is also the foundational concept for much younger disciplines such as *computer graphics* and *computer vision*, which are the ones we are concerned with in this thesis. *Computer vision* is the science that analyzes a two-dimensional *image*, and tries to reconstruct a *model* of the object (or *scene*) observed in the image. This is a very fundamental process resembling (at least in its goals) human perception. The way back from the model to a human-understandable and visually pleasing presentation is gone by *computer graphics*, where photo-realistic or non-photo-realistic images are generated from a

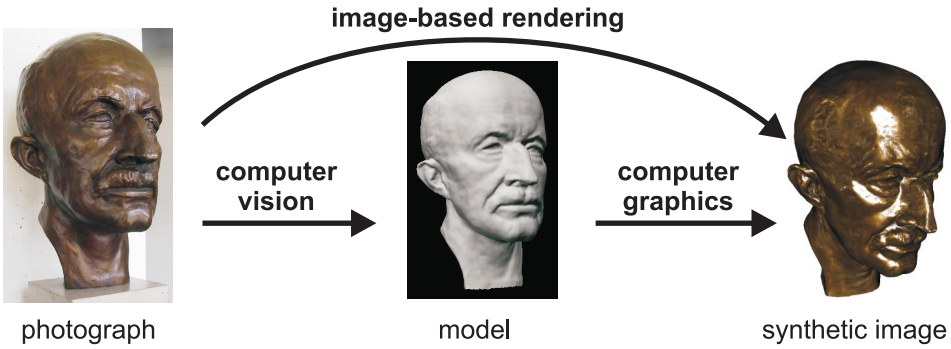

Figure 1.1: Computer vision creates models from images (modeling) – computer graphics generates images from models (rendering). The model can include geometry, material lighting, motion, and much more. Image-based rendering uses images directly for rendering, often without reconstructing an explicit model. Images and model of the Max Planck bust courtesy of Christian Rössl and Hendrik Lensch, Max-Planck-Institut für Informatik.

virtual scene model. This process is also called *rendering*. Applications of computer graphics can be found everywhere these days, especially in the media, where entire feature films are generated by rendering image sequences of virtual scenes created and animated in a computer.

In recent years, a new way of thinking has surfaced in computer graphics and computer vision, called *image-based rendering*. The shattering truth that came with this new discipline is that in some cases, one might get along very well *without* an explicit model. Why spend time and thought on modeling the geometry, material, and light sources in a scene? Why not just record images, and use these images again to generate new images?

One of the most prominent public successes of three-dimensional image-based rendering was used in the film *The Matrix* by Warner Bros. They introduced a new visual effect giving the audience the impression of "flying" around a character (a real actor, not a virtual one) who seems to be frozen in time and space. E.g. the hero or heroine would jump into the air, and suddenly the scene would freeze, the camera would move half-way around, and then the film would go on as before (cf. Figure 1.2, left).

This stunning effect is the result of a rather simple image-based rendering trick. A large number of cameras is placed on a path half-way around the character who is supposed to jump, and in the right moment and position all the cameras will take a picture simultaneously (cf. Figure 1.2, right). Playing back these recorded images, the viewer could jump between the different camera positions capturing the same moment in time from different viewpoints. Since this is not enough, computer graphics techniques are used to interpolate between adjacent camera positions, so that a "virtual" camera can move smoothly along the path predefined by the real cameras. This effect has very quickly found its way into a lot of different action movies as well as into a larger number of contemporary video clips.

Figure 1.2: Visual effects from the film *The Matrix*. Capturing a scene from many viewpoints simultaneously allows to "fly around" the scene, frozen in time. Images courtesy of Warner Bros.

This small example shows what can be achieved by capturing and manipulating light. Although the idea may seem straightforward, several years of research in this area have revealed a number of unsolved problems, especially when trying to solve more general problems with less specific and expensive acquisition hardware.

The *light field*, as mentioned in the title of this work, is a much more general concept than the camera interpolation along a path. A light field captures all the light leaving a certain region of space. So instead of placing just a number of cameras along a path, one would need to surround a certain volume ("the scene") completely by a dense set of cameras, in order to catch every single ray of light that leaves the region. In principle, a virtual viewer could then look at the recorded scene from nearly *every possible view*. In addition, we would like to allow *interactive* navigation and rendering, in contrast to just computing arbitrary views off line for generating a movie. We talk of the light field as an alternative *3D scene representation*. For many applications, viewing the scene from arbitrary positions is enough, and so the light field is a viable representation that does not require to first model the scene in a classical way.

However, as we will explain in much more detail, this task is rather involved, since the correct and fully automatic interpolation between views is not as easy as it might seem, especially if these views are far apart from each other (and nobody wants to pay for an infinite number of infinitely small cameras). Since 1996, when light fields were introduced to computer graphics, many researchers have contributed to understanding and improving the light field representation, as well as to finding efficient techniques for acquiring and rendering light fields.

All these three aspects, from acquisition over representation to rendering, are the topic of this thesis. Working on this field from 1998 to 2002, we have tried to find an alternative representation for light fields, and to exploit this representation for both efficient acquisition and rendering. Parts of that work have been published in several scientific articles [49, 112,

113, 115, 114, 76]. Here we present these different parts in a common framework, show interrelations, and discuss the different techniques from the most current point of view. The main contributions of this thesis are:

- A survey of image-based rendering, focusing on techniques for 3D scene viewing. This is one of the few comprehensive surveys of this very young and active area of research (Chapter 2).
- An in-depth discussion of previous work on light fields, Lumigraphs, and closely related topics that serve as a basis for the work in this thesis (Chapter 3).
- A new representation of light fields with additional geometric data using per-pixel depth information, and a way to refine a stored light field before rendering by using this geometric information (Chapter 4).
- An interactive rendering technique based on the above-mentioned representation, avoiding several of the problems of previous rendering techniques (Chapter 5).
- A technique for the automatic and adaptive acquisition of light fields from synthetic scenes, e.g. for interactive viewing of simulation results, again based on the new representation and rendering techniques (Chapter 6).
- A new parameterization allowing a very flexible placement of cameras around a scene, and modified rendering algorithms that exploit this new structure efficiently (Chapter 7).
- A complete prototype system that is able to capture a light field of a *dynamic* scene, reconstruct some approximate geometric information, transmit all necessary information over a network, and instantly render arbitrary views of the captured scene at the remote location (Chapter 8).

We conclude this thesis in Chapter 9 by summing up the contributions and experience made in the course of this work, and by discussing the opportunities and possible future development of light fields and image-based rendering.

CHAPTER 2

Short Tour of Image-based Rendering

> A Beginning is the time for taking the most delicate care that the balances are correct.
>
> Frank Herbert, *Dune*

Light fields are part of a rather young research field called *image-based rendering* (IBR). Although the actual light field movement in computer graphics started no earlier than in 1996, a lot of work has been done since then, and also a lot of earlier related work must be taken into account in order to position and relate to the ideas presented in this thesis. The goal of this chapter is to give an idea of image-based rendering as a whole, as well as to define the position of light fields and Lumigraphs in relation to other image-based rendering techniques, always focusing on techniques that can be used for 3D scene viewing.

After briefly introducing the idea of image-based rendering in the wider field of *3D image analysis and synthesis*, we sketch the most popular and important techniques in Sections 2.3 – 2.7, grouped by basic classes of data representation. In Section 2.8 we compare the presented techniques with respect to a number of basic criteria such as navigational freedom as well as

geometric and photometric complexity. We end the tour with a brief summary of the topic and some conclusions that might be useful to potential users of IBR techniques.

2.1 3D Image Analysis and Synthesis

Only a decade ago, *computer graphics* and *computer vision* were two separate fields of research. Recently, and mainly through the developments in *image-based modeling and rendering*, the two fields began to interfere with each other to such an extent that nowadays many researchers are talking of the *convergence* of graphics and vision [72, 96]. In much the same spirit we like to see our research positioned within *3D image analysis and synthesis* [34] that comprises and combines both fields.

Computer vision [93, 65, 133] is concerned with the *analysis* of single or multiple images or image sequences in order to extract data such as geometry, motion, or segmentation information. In general, one can say that computer vision reconstructs *model parameters* from images (see again page 10, Figure 1.1). These computational models are used to represent shape, motion, material, lighting, and many more. Coming from the "model" end of this problem, image analysis is also often called *image-based modeling*.

Traditional computer graphics [29, 35] goes in the opposite direction by *synthesizing* images from computational models. Given a model consisting of 3D geometry, texture and other material properties, as well as virtual lighting, computer graphics tries to compute either *photo-realistic* or *non-photo-realistic* images of this model from the current user's point of view.

The actual reason why these two basic approaches need to be brought together is quite simple: for many problems we would like to synthesize *output* images of a scene that first needs to be acquired by capturing and analyzing *input* images. In order to do this most effectively and efficiently, the in-between model must be well adapted to the constraints of both input and output. Under the common roof of *3D image analysis and synthesis* [34], we can summarize all research that looks at either of the two problems (analysis or synthesis), or at both problems as a single pipeline from images over some kind of model back to images. In the context of this work, we will have a look at a topic that is very much in the spirit of the complete analysis-synthesis pipeline, and is called *image-based rendering*.

2.2 Image-based Rendering

Image-based rendering algorithms have one common feature: they directly use image data for rendering. Usually, this image data is inferred from the real world by means of still photography or video capture, and is used either as the only underlying data for re-rendering the scene, or as a means to increase the visual complexity and photo-realism of a geometric model.

Image-based rendering has undergone an enormous activity boost in the last few years, mainly through the rapid development of digital imaging sensors. Affordable digital cameras

are evolving at an impressive speed due to their distribution in the consumer market. At the same time, computer graphics hardware acceleration has also focused on the efficient support of multi-texturing and similar methods, making it easy and very efficient to use image data for rendering. The rather simple idea that *photo-realism* can be achieved very easily by actually using photographs has inspired a whole lot of recent research.

Unfortunately, there is no comprehensive textbook on image-based rendering yet, and even the number of complete and up-to-date survey articles is very small. Up to now the best general references to the knowledge of the author are very brief surveys by McMillan and Gortler [96] and Lengyel [72], more detailed surveys by Sing Bing Kang [58, 59] and Heinrich Müller [98], and Section 5.4 in the textbook on 3D image analysis and synthesis edited by Girod, Greiner, and Niemann [34]. Furthermore, there is a profound description on the foundations of 3D reprojection in the Ph.D. thesis of McMillan [94], a recent review of different light field techniques in the Ph.D. thesis of Camahort [40], and some surveys that touch the same or similar fields of research, e.g. by Heidrich [47] and by Zwicker et al. [149].

In the following tour, we cannot cover all aspects of image-based rendering, especially since image-based techniques are used in a variety of different contexts in order to speed up existing algorithms. So in what follows, we focus on techniques for interactive viewing of a 3D scene using image-based rendering as the principal technology.

2.3 Using a Single Image

We start our IBR tour with the very basis of most interactive image-based rendering work, which is texture mapping. Following that we outline image-based approaches that use a single image (probably obtained by stitching together multiple original images) in order to enhance realism and/or replace geometry, namely sprites, panoramas, and environment maps.

Texture Mapping. The application of so-called texture maps [13, 6] can be considered one of the most important early image-based techniques. Although it is not a scene viewing technique itself, many other approaches are based on texture mapping, so it must be mentioned here. The most important point about using texture maps is that it drastically increases the visual complexity of the object while the geometric complexity stays the same.

A texture map is a 2D image that can be mapped onto a geometric object. The simplest way of using a texture map is to just use the image to define the (spatially varying) diffuse color of the object. The technique of texture mapping basically reduces to a *resampling* task, since the color of every output pixel must be correctly reconstructed from the corresponding texture pixels, or *texels*. A lot of important issues have been researched, e.g. ways for proper and efficient sampling and filtering. Further important steps were multi-resolution representation of textures, so-called *mip-maps* [142], as well as *projective texture mapping* [117], which allows projecting an image onto an object from an arbitrary point in space, just like using a slide projector. One of the strongest points of texture mapping is that it is fully supported by modern

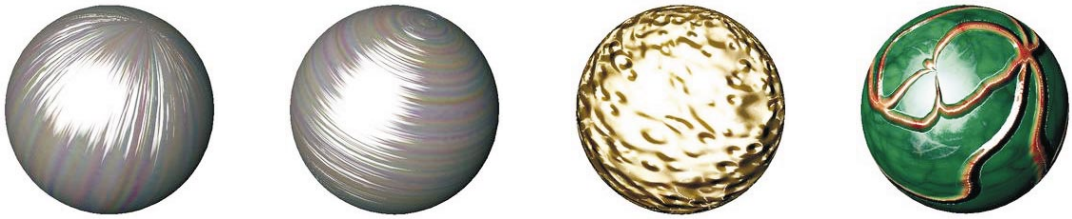

Figure 2.1: Texture mapping can be used to increase the visual complexity of an object without making it geometrically more complex. These renderings (courtesy Jan Kautz, MPI Informatik) only use a simple sphere as the scene geometry, and add reflection details or "bumps by texture mapping techniques.

graphics libraries such as OpenGL [99, 116], and it can be done completely in the graphics hardware, so it does not consume any CPU time. Figure 2.1 illustrates some examples of texture mapping.

For a good insight to the basics of texture mapping see Heckbert's survey [43] and master's thesis [44]. For further reading about more recent texture mapping techniques see Haeberli [41], Heidrich and Seidel [51] and Heidrich et al. [52].

Sprites, Billboards, Impostors. Even long before modern 3D graphics infrastructure was established, graphics and especially game programmers used so-called *sprites* (simple 2D images) in order to model single objects in a scene. In the context of walk-throughs in complex environments, the logical succession of the sprite concept is called *billboard* or *impostor*. Billboards are used to represent far-away 3D objects through a simple 2D image, and are usually projected on a rectangle orthogonal to the current viewing direction [109]. In order to reduce the geometric complexity of the view and thus guarantee interactive frame rates, a lot of improvements, extensions, and applications have been proposed [111, 122, 124, 73, 22]. In 1996, Torborg and Kajiya even proposed a multi-layer sprite system as an alternative graphics hardware architecture [131] called *Talisman*.

Panoramas. A very popular and effective way of creating a realistic 3D experience with 2D images is the use of so-called *panoramas*. A panorama is a (large) image mapped on a cylinder, sphere, or similar shape, in such way that it encloses the viewer, or at least fills as much as possible of the viewer's field of view [95, 17]. This technique is also very popular in large-screen and 3D movie theaters.

The interesting issues concerning panoramas are how they are created and how to give the user as much freedom in navigation as possible. Creation of panoramas is either done with specialized camera devices that use extreme fish-eye lenses or that take a picture of a mirrored ball (cf. Figure 2.2, left), or by *stitching* together "normal" photographs using special software.

The inherent problem of a panorama is that it is only valid from the original point of view (e.g. the user is allowed to rotate, but not to change position). As soon as the user moves to a

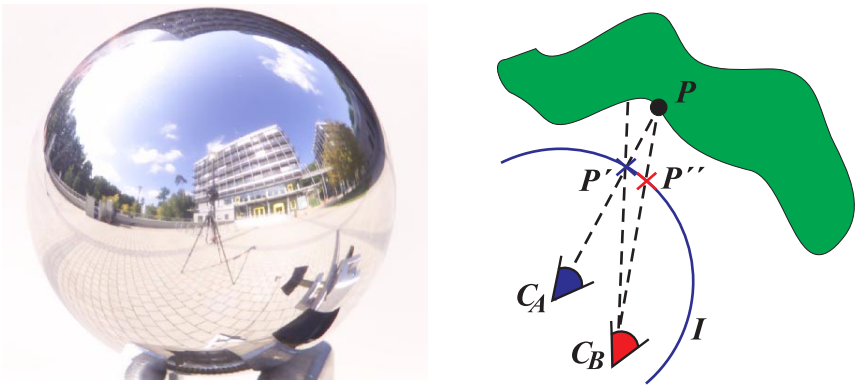

Figure 2.2: Panoramas. Left: taking a picture of a mirrored ball gives a 180 degree panorama of the scene, including an image of the camera itself (picture courtesy of Michael Gösele and Philippe Bekaert). Right: the parallax effect cannot be captured by a panorama. If the viewer changes location from C_A to C_B, the 3D point P is observed in the wrong location P' of the panoramic image I, instead of the correct P''.

different location, the expected parallax effect is missing, and the 3D impression is seriously degraded (cf. Figure 2.2, right).

The QuicktimeVR^{TM} system by Chen [17] allows the user to stitch ordinary photographs together for creating cylindrical panoramas for multiple viewpoints. Then the user can jump from point to point and rotate as well as zoom from these discrete viewpoints. Several other techniques and systems have been proposed lateron, including spherical panoramas [129]. Just recently Benosman and Kang have edited a comprehensive textbook on the topic [4].

Environment Maps. One very important special case of a panorama is the so-called environment map [6, 38]. An environment map represents the light falling into the scene from all possible incoming directions, and can be imagined as some infinitely large texture-mapped shape surrounding the scene. This rather simple (and graphics hardware-supported) approach can effectively model the background of a scene, but it can also be used to *illuminate* the objects of the scene like a real environment would do. The basic assumption for that is that the objects are very small compared to the distance from the environment.

In the case of mirroring objects, one simply reflects the viewing ray off the objects and looks up the illumination value for the resulting direction in the environment map. This rather simple technique is very effective in enhancing the realism when rendering mirroring objects [116, 42]. In order to support this step efficiently in graphics hardware, different parameterizations for environment maps have been proposed [47, 46], especially spheres, cubes, and more recently paraboloids [50]. Moreover, prefiltering techniques have been proposed in order to efficiently apply the technique to glossy objects or other advanced reflectance func-

Figure 2.3: Example of the view morphing technique. Here it is used to morph between two views of *different* persons. Note how the features (e.g. nose, eyes) are correctly transformed from one view to the other. Images courtesy of Steve Seitz and Chuck Dyer.

tions [51, 46, 60, 61]. Recently in 1999 Cabral et al. [10] proposed to use multiple environment maps and warp and interpolate between these maps in order to overcome the view-dependence inherent in classical environment map techniques.

2.4 Interpolating Between Two Views

Having sketched the basic approaches that use single images for representing parts of a scene, we now follow up with techniques that interpolate novel views from two original views. *Image morphing* is the variant that uses only image information and a number of correspondences, whereas *view interpolation* refers to a technique that makes use of per-pixel depth information. Finally, the work on *plenoptic modeling* generalizes this approach to non-planar images.

Image Morphing and View Morphing. Image morphing is a technique for generating smooth and plausible transitions between two images. One very prominent example for image morphing is the video clip for Michael Jackson's song "black and white", where different faces are being smoothly transformed into one another. The key issue in image morphing is the question how to do the blending between the two images. Simply cross-dissolving the two images does not produce a realistic impression, so some care must be taken that the important *features* in one image slowly transform into the same features in the second image. For example, the nose of one face should be transformed into the nose of the other face. This can be achieved by manually specifying some feature points, automatically reconstructing a dense mesh of correspondence points between the two images, and finally doing the interpolation by moving the mesh vertices from the source to the destination positions and interpolating pixel colors in between [143, 3, 71].

A similar technique can also be used to smoothly blend between two different camera views of the same scene, thereby giving the viewer the impression of a smoothly moving camera although the real images were taken with still cameras in different positions. However,

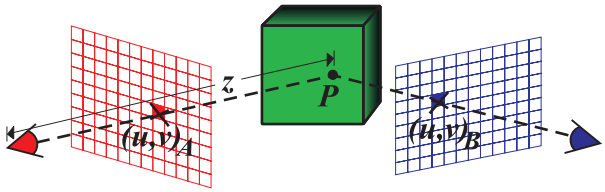

Figure 2.4: *3D reprojection*, also called *3D warping*. A 3D point P is seen through two images A and B, at positions $(u, v)_A$ and $(u, v)_B$, respectively. So a pixel in image A can be reprojected to the correct position in image B if the 3D position or the depth z of P is known.

simple image morphing does not preserve 3D shape, and so in-between views usually do not correspond to a real view taken from the same position. In 1996 Seitz and Dyer [119] have proposed the so-called *view morphing* technique to avoid this problem. The basic trick is to first *prewarp* the two images so that they are defined on coplanar image planes, then *interpolate* between the two images, and finally *postwarp* the result into the desired view. The important fact about this approach is that all these steps, especially linear interpolation of parallel images, is shape-preserving. The only negative side of this approach is that a number of correspondences between the two images must be specified manually in order to determine the transformation. Figure 2.3 shows an example of this technique.

View Interpolation. In the case of synthetic images, the next step for automatic interpolation of two images is to exploit as much information as possible, especially the camera position and orientation as well as the depth value of every pixel[1]. Using this information, the 3D position of every pixel can be determined, and thus the pixel can be correctly *reprojected* into any other view. Figure 2.4 illustrates this general principle that is also called *3D warping*, or more specifically, *forward warping*. In 1993, Chen and Williams proposed a technique called *view interpolation* [16] that uses 3D warping for doing a geometrically correct interpolation between two views. They first create a so-called *morph map* organized in a quad-tree manner that contains correspondences between pixel blocks in the two views. During rendering, they interpolate the colors for each block and render them back-to-front in order to avoid occlusion problems and without the need to use Z buffering. Finally, holes in the resulting images are filled by 2D filtering operations. The view interpolation rendering step (excluding morph map generation) can be performed at interactive frame rates, so that a collection of views can be used as a representation for interactive scene rendering.

Later on, Mark et al. [87] use a similar approach (based on McMillan and Bishop's warping technique [95]) in order to generate in-between frames for interactive rendering by image-based techniques, thereby increasing the rendering frame rate and effectively decreasing display latency. Instead of interpolation, the final color of each pixel is determined from only one of the

[1] A pixel's depth value describes the distance from the eyepoint to the 3D point corresponding to the pixel. As an alternative to the actual distance, sometimes the orthogonal distance from the image plane is used.

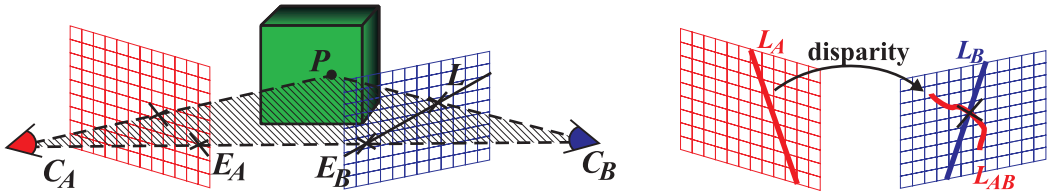

Figure 2.5: Left: A 3D point P and the two centers of projection C_A and C_B span the so-called *epipolar plane*. Intersecting the line (C_A, C_B) with the image planes yields the *epipoles* E_A and E_B. The projection of P in image B lies on the *epipolar line* L that is the intersection of the epipolar plane with the image plane. Right: the projection of a novel viewing ray (not shown) into two reference images yields two lines L_A and L_B. Disparity-mapping L_A into image B yields a curve L_{AB}. The intersection of L_{AB} with L_B determines the reference pixel position for the desired viewing ray.

corresponding reference pixels.

Already in 1994, Laveau and Faugeras [70] proposed an alternative technique for reconstructing novel views from pairs of reference images. They use the *fundamental matrix* and *epipolar geometry* concepts for expressing the relative geometry of two views [27, 26]. Figure 2.5 (left) illustrates the principle of epipolar geometry, which constrains the image of a pixel in one view to lie on an *epipolar line* in another view. Laveau and Faugeras propose an algorithm that works for both strongly and weakly calibrated cameras, meaning that only the relative transformation between cameras needs to be known. They use dense disparity maps[2] for each input image, and perform ray casting and correspondence searching during the rendering phase. Figure 2.5 (right) illustrates the basic situation: they first project the desired viewing ray into both reference views, yielding two epipolar lines L_A and L_B. They apply the disparity map to L_A, yielding a curve L_{AB} in the other image. The intersection points of this curve with L_B give the possible reference pixel positions. Laveau and Faugeras also mention how to relate more than two reference views, but they do not propose a technique for efficiently using this information for rendering (e.g. how to determine the best subset of views for reconstructing a certain pixel, or how to weight the contributions from each of the input views).

Plenoptic Modeling and 3D Warping. Bringing together ideas from both panoramas and view interpolation, in 1995 McMillan and Bishop [95] proposed a technique for interpolating novel views (planar or panoramic) from pairs of existing cylindrical panoramas. The name of the method refers to the so-called *plenoptic function* [1] that represents the spatial and directional distribution of light (see also Section 3.1 on page 36). Like for view interpolation, their technique is based on 3D forward warping.

[2]The *disparity* value describes the relative position of the corresponding pixel on the *epipolar line* in a second view. Its information is roughly equivalent to a depth value, except that the cameras do not need to be strongly calibrated.

Figure 2.6: User interface of the Aspen Movie Map system. Image courtesy of Michael Naimark and Andy Lippman, MIT Media Lab

One important contribution of McMillan's and Bishop's work is showing the existence of an *occlusion-compatible* rendering order for both planar and cylindrical projections. This means that pixels can be rendered in the right order so that occlusions are handled correctly without using a Z buffer. Like Laveau and Faugeras, McMillan and Bishop propose to find pixel correlations by searching for corresponding pixels along epipolar lines (cf. Figure 2.5), or *epipolar curves* in the cylindric case [95].

2.5 Image Collections and Ray Databases

The work on view interpolation that we have presented so far is usually restricted to a small number of original views. The next very important issue is how to find the relevant samples for reconstructing arbitrary views from a *large* collection of reference images.

This discussion leads to the very general idea of a *ray database* [96]. This name denotes approaches that store a large number of originally sampled rays from multiple images in order to generate arbitrary novel images by interpolating from these rays.

The techniques that fall under this category mainly differ in how rays are parameterized, how they are filtered for storage and for rendering, and how freely the user can navigate. In this section, we will concentrate on *pure* ray databases, whereas the next section will include techniques that exploit (and thus need) additional geometric information about the scene.

Movie Maps. In the late 1970's a group at MIT developed a system for virtual travel in the town of Aspen [77, 28]. This was done by using four vehicle-mounted video cameras pointing in different directions, and by capturing images every three meters along the path of the vehicle. The user could later navigate along that path and turn around by replaying these images from a random access video disc (see Figure 2.6). Later, the same principle has also been applied to capture whole grids of viewpoints using a helicopter, e.g. flying over the San Francisco bay area.

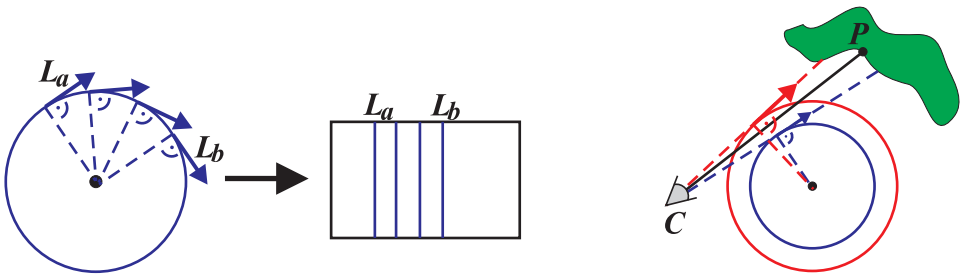

Figure 2.7: Concentric Mosaics. Left: each vertical line L_a of a single concentric mosaic image corresponds to one slit image taken from a different point on the circle and looking in the associated tangent direction. Right: reconstructing a ray from a novel viewpoint C to a point P can be done by interpolating between the tangent rays (dashed lines) of the closest two concentric mosaics.

Concentric Mosaics. In 1999, Shum et al. [123] introduced the so-called *concentric mosaics*. These mosaics consist of a large number of vertical slit images captured from viewpoints on concentric coplanar circles around a common center. Imagine a slit camera mounted on a beam rotating on a tripod, and looking in the tangent direction (i.e. orthogonal to the beam). A concentric mosaic is obtained by stitching together the slit images for one complete rotation. Multiple concentric mosaics are obtained by shifting the camera to a different position on the beam (cf. Figure 2.7). Using this setup, rays can be parameterized by three parameters: rotation angle, radius, and elevation angle above the plane.

The viewer is restricted to stay close to the ground plane and within the concentric circles covered during the capturing process. Viewing rays are reconstructed by linear interpolation from the nearest pixels in the closest two mosaics. The authors also propose two different approaches for using depth information and 3D warping to avoid vertical distortion problems.

Light Fields. The most general pure ray database up to now is the so-called *light field* that has been proposed in 1996 by Levoy and Hanrahan [74] and concurrently by Gortler et al. [37]. The general idea is to capture all rays that pass through a bounded region in 3D space. Using this restriction, a ray can be defined by the entry and exit points on the surface of that 3D region, thus requiring four dimensions (two for each intersection point) for parameterizing all possible rays passing through the region.

In the original light field work [74, 37], the authors used sets of coplanar planes for the parameterization. Such a two-plane set is called a *light slab* (see Figure 2.8, left), and the seamless combination of up to six such slabs would result in a complete parameterization of all rays passing through the region bounded by one half of the planes. The right picture in Figure 2.8 illustrates such a multi-slab light field. In the meantime, many alternative parameterizations have been proposed, e.g. using spherical, combination of spherical and planar, as well as polar coordinates [54, 12, 132]. We will go into the details of light field parameteriza-

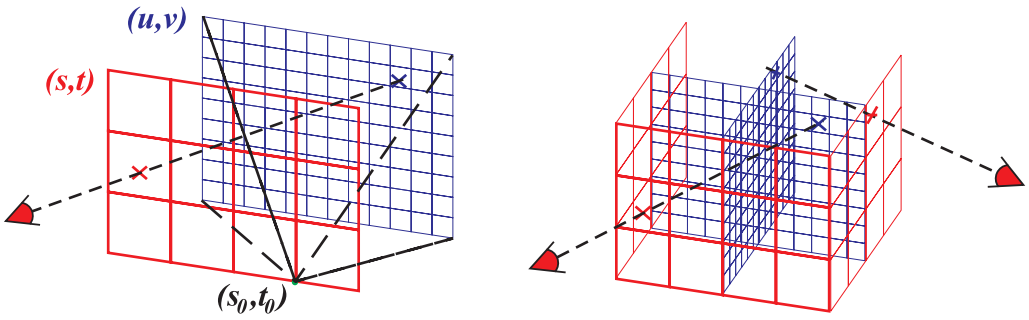

Figure 2.8: Left: One *light slab* in a two-plane-parameterized light field. Each ray passing through both planes can be characterized by the two intersection points (s, t) and (u, v). All rays through one point (s_0, t_0) on one plane yield a sheared perspective image on the other plane. Right: sketch of a multi-slab light field with three pairs of planes. Different rays may intersect with different slabs, but still each ray is characterized by one pair of intersections.

tion and discretization later-on in Section 3.2.

Acquisition of a real-world light field is a rather difficult topic. In a laboratory setup, one can use a robot gantry to exactly position a camera at several different viewpoints in a common plane [74]. These viewpoints then yield the sample points in one of the two slab planes, and the pixels of the captured images correspond to the sample points in the other plane. In order to capture multiple slabs like this, the object or the camera gantry need to be rotated in 90-degree steps.

Another approach was proposed and realized by Gortler et al. [37], who use a hand-held video camera in order to easily acquire a large number of images without the need for a sophisticated gantry. Then they calibrate the camera images with the help of markers in the scene, reconstruct a rough approximation of the scene geometry using silhouette information or similar volumetric techniques, and resample the image data to the desired viewpoint locations in the two-plane light field. For more detail on this representation please refer to Section 3.5. Note that the *rebinning* step (resampling the images into the desired eyepoint positions) requires knowledge about the scene geometry in order to account for parallax effects.

Rendering from two-plane light fields is very simple and efficient. In software, every desired ray can simply be interpolated from the nearest rays in the ray database. If both planes are sampled using a uniform grid, a ray would be interpolated from 4×4 nearest rays connecting the four nearest points on each plane (cf. Figure 3.4 on page 40). Gortler et al. also proposed a very efficient and simple approach for approximating quadri-linear interpolation by texture mapping, which was later extended to arbitrary tesselations of the viewpoint plane [125].

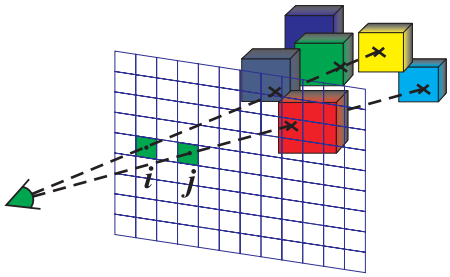

Figure 2.9: Layered depth image (LDI). Every pixel in the LDI (sketched as green squares) holds color and depth samples for all intersections of the corresponding ray with the scene. In the above example, pixel i represents three samples, and pixel j two (only the front-facing intersections are counted in this illustration).

2.6 Ray Databases with Per-Pixel Geometry

As one has probably concluded from the approaches presented so far, image data can be reused in a much more general way if it can be *reprojected* into arbitrary viewing positions. However, this is only possible if the true 3D position of the pixel is known. This is why the vast majority of modern image-based rendering algorithms rely on geometry information of some form (per-pixel depth values or disparities, binary volumes, triangle meshes). In the next two sections we outline the most prominent of these techniques. We first concentrate on those that use sampled per-pixel geometry information in the reference images, and then continue in the next section with techniques that use "global" geometry data such as a polygon model of the scene.

Layered Depth Images. In 1998 Shade et al. [121] proposed an image-based rendering approach that uses an intermediate, merged representation of views. The so-called *layered depth image (LDI)* can be seen as an image with multiple samples for every pixel. Each sample represents a different object surface intersected by the corresponding line of sight (cf. Figure 2.9), and holds the color, the depth value, and some information for rendering (splat computation), based on normal direction and distance into the LDI's center of projection. Note that this representation implies that the scene is purely diffuse, since each surface point is represented by a single color value, whereas view interpolation techniques blend between different views and thereby allow for directional change in the appearance of an object.

Already in 1996, Max proposed a technique very similar to LDIs using multiple depth layers for representing and rendering complex objects such as trees [91, 92].

An LDI is constructed by reprojecting a number of input images (together with known camera parameters) from the LDI's center of projection and sorting the samples into bins along each line of sight (e.g. merging samples that are projected into nearly the same position). Alternatively, one can construct an LDI from a synthetic scene using a modified ray tracing software that registers all intersections of a ray with scene objects (as opposed to registering only the

front-most intersection).

Rendering from an LDI is done by *splatting* [141] the LDI samples into the novel view in back-to-front order, which is possible by adapting McMillan's occlusion order algorithm [95]. Since the range of well-reconstructed viewing directions is somewhat limited for a single LDI, Lischinski and Rappoport [78] combine three orthogonal LDIs into a so-called **layered depth cube** (LDC). Oliveira and Bishop [102] combine six LDIs with a common center of projection to yield **image-based objects** (IBO). Chang et al. [15] have further extended the LDI concept to **LDI trees**, a multi-resolution representation with hierarchical space partitioning suitable for representing complete scenes.

Multiple-Centers-of-Projection Images. The MCOP image as introduced by Rademacher et al. [106] is a 2D image where each vertical pixel column of the image is taken from a slightly different viewpoint on a continuous camera path. You can imagine the acquisition of such an image as done by a slit camera (having just a vertical slit) that is moved along a certain path during the acquisition of the picture. Furthermore, the film of the slit camera would be continuously transported at the same time. This representation is somewhat similar to concentric mosaics [123], where we interpolate between multiple circular paths. Rendering from MCOP images requires per-pixel depth values and is done by 3D warping (see above). A similar but less general idea applied to cell animation back drops has also been published before under the name of *multi-perspective panoramas* [145].

Delta Trees. The *delta tree* proposed by Dally et al. [18] stores a quad-tree hierarchy of images taken on a sampling sphere around an object, and further employs a coding scheme based on the discrete cosine transform (DCT) in order to compress redundant information. Rendering is done by forward warping, and for the acquisition as well as the rendering a diffuse scene is assumed (sampling density is determined by geometric properties of the object alone, and rendering does not employ blending between adjacent views).

2.7 Ray Databases with Polygonal Geometry Models

Besides the approaches presented so far, several techniques exist that use a number of reference views along with a real polygonal model of the scene. Depending on structure and quality of the geometric data we can distinguish different approaches. View-dependent texturing was designed for architectural models and assumes a simple underlying geometry (large flat structures). Surface light fields are a very compact and high-quality representation for view-dependent effects, but require exact geometry in order to yield good results. Lumigraphs lie in between these two approaches, and can make efficient use of geometric approximations of different quality. Unstructured Lumigraphs allow efficient rendering from input images that are much less well-structured (e.g. scattered in space) than for classical Lumigraphs. Methods based on the visual hull of an object combine geometry reconstruction and rendering at interactive speed, but are somewhat limited in the shapes they can represent.

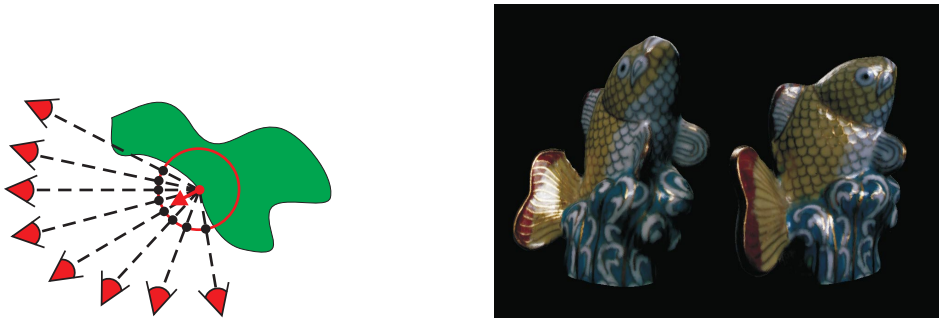

Figure 2.10: Surface light fields. Left: data from all input images is resampled into *Lumispheres* in each texel on the surface geometry (2D sketch). Discrete vector-quantized Lumispheres are then used for representing the reflected light on each point of the surface. Right: Impressive rendering quality achieved with surface light fields. Images courtesy of Daniel Wood and the University of Washington Graphics and Imaging Laboratory.

View-Dependent Texturing. One very important approach to image-based rendering is to use some approximation of the scene's geometry in the form of polygons, and then texture map these polygons depending on the current viewpoint. The first popular method along that line was probably *view-dependent texture mapping* by Debevec et al. in 1996 [21] for rendering architectural models from a combination of manually built coarse geometric models and images acquired by photography. The basic idea is use the reference images to compensate for missing detail in the geometric model and to add view-dependent effects.

In the original work, view interpolation is done in the output image on a per-pixel basis from the respective two closest views. Visibility and other related issues are computed at rendering time for each pixel, and so the technique does not allow for interactive frame rates. In 1999 an improved technique was proposed by Debevec et al. [19]. They precompute a so-called *view map* for each triangle. This view map is used to determine the three best views for every polygon and every possible viewing direction. Projective texture mapping is used for rendering. Although the texture for each polygon is blended from three different views, the technique does not ensure smooth blending across polygon edges, since every polygon is treated independently.

Surface Light Fields. While view-dependent texture mapping was mainly thought for very sparse sampling of the possible space of directions (i.e. a few architectural photographs), surface light fields represent a high-quality single object representation that is generated from a large number of photographs. The first work on parameterizing a light field directly on the object's surface was by Miller et al. in 1998 [97]. It was mainly concerned with coding and compression issues related to that representation, and was only applied to synthetic scenes.

In 2000, Wood et al. [146] presented a number of techniques for generating and rendering

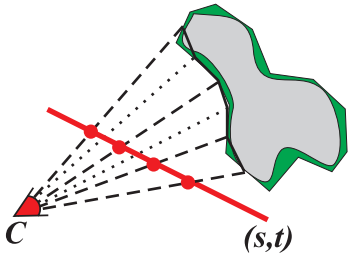

Figure 2.11: 2D sketch of adaptive depth correction in a Lumigraph slab. Sample rays (dashed) are cast from the new viewpoint C through the Lumigraph eyepoints (s, t) to intersect the geometric proxy (green polygon). Where adjacent depth values differ too much, additional sample rays (dotted) are cast adaptively.

compact surface light fields from photographs and range scan data, including geometric level-of-detail control during rendering. The underlying data structure is a collection of so-called *Lumispheres* (cf. Figure 2.10) that represent the distribution of outgoing light parameterized over the reflected direction (this is done in order to maximally exploit directional coherence). These Lumispheres are at first generated by collecting samples from all photographs for all texels on the geometric model, and resampling these into a regular grid on the directional sphere, relative to the ideal reflected light direction (in order to maximally exploit coherence). Then, every actual Lumisphere is replaced by a linear combination from a set of representative Lumispheres by principal function analysis, which is an approach that yields better results than simple vector quantization. The representation obtained in that way is very compact and gives compelling rendering results at interactive frame rates.

Lumigraphs. Surface light fields assume that the exact geometry of an object is known, and use the surface for yielding a ray parameterization that is very well adapted to the scene geometry, and thus is very compact. A Lumigraph as presented by Gortler et al. in 1996 [37] sticks more closely to the original light field idea, so the basic information for rendering is a set of images. A good view reconstruction by pure image interpolation requires very dense sampling, and thus an enormous amount of image data. If two reference views are too far apart from each other, the parallax effect will cause severe blurring and ghosting artifacts when blending them.

Therefore the Lumigraph uses a *geometric proxy* in order to *depth-correct* the interpolation process. In the efficient implementation based on texture hardware, depth correction is performed by casting rays through the vertices to be drawn, and by intersecting these rays with the geometric proxy. The projection of the resulting 3D positions of the desired pixels is then used to determine the actual texture coordinates used for interpolating from the reference views (cf. Figure 2.11). The depth correction works adaptively by subdividing display triangles, so that the actually drawn triangles all have an approximately constant depth. More detail on this topic

will be given in Section 3.5.

The concept of the geometric proxy has later been used and extended to simulate arbitrary photographic effects. By replacing the geometric proxy with a virtual focal plane, Isaksen et al. achieve realistic depth-of-field effects [55].

Unstructured Lumigraph Rendering. As we have mentioned before, light fields and Lumigraphs assume that all viewpoints fall on a common plane in order to achieve a simple parameterization and efficient rendering. This restriction is very annoying when dealing with real-world cameras, since it is very hard to position the camera exactly. So an expensive and error-prone rebinning step is required. Heigl et al. first freed the Lumigraph approach from these limitations by proposing a rendering technique very similar to Lumigraph rendering, but directly working on a sequence of images taken by a hand-held camera [53]. Their work includes camera calibration as well as geometry reconstruction (see also Koch et al. [66]), and they achieve fast rendering by re-triangulating the reconstructed viewpoints with respect to the current view, and blending between the respective three views associated with each of these viewing triangles.

Buehler et al. [9] have extended this idea in order to provide the viewer with *unrestricted* navigation capabilities in a collection of images. The so-called *unstructured Lumigraph* only shares very few of the original ideas of light fields and Lumigraphs, since no bounded volume is used anymore to restrict the viewer. Like in Heigl's work, reference viewpoints are triangulated for every output frame, and the resulting triangles are used to blend between the reference views. Furthermore, heuristics are derived to account for occlusions as well as resolution issues. In order to have enough data for interpolation, more than three nearest views can be used for interpolation. They demonstrate the versatility of their approach by applying it to various different types of scenes.

Visual Hull. Last, but not least, there are several image-based representations and rendering techniques based on the so-called *visual hull* of an object. In 1994 Laurentini [69] introduced the visual hull as the maximal volume that is consistent with a given set of *silhouettes*. Matusik et al. introduced a technique called *image-based visual hulls* in 2000 [90], and extended and improved the technique further using *polyhedral* representation in 2001 [89]. The basic idea is to use silhouette information to reconstruct a geometric proxy of the scene, and then re-texture this geometry using the input images. Robust geometry reconstruction is done by clever and efficient intersection of the extruded silhouettes in the reference images. Rendering can be done either in a per-output-pixel fashion [90] or by unstructured Lumigraph rendering [89]. A complete system is demonstrated that performs image capture with six cameras, geometry reconstruction, and rendering at interactive frame rates. The big advantage of the technique is the robustness of the silhouette-based reconstruction approach (although this requires segmentation of the image into foreground and background). The down side is that the reconstructed geometry is rather coarse (Laurentini called this the *silhouette-active surface*) since concavities that are not part of the silhouette cannot be modeled.

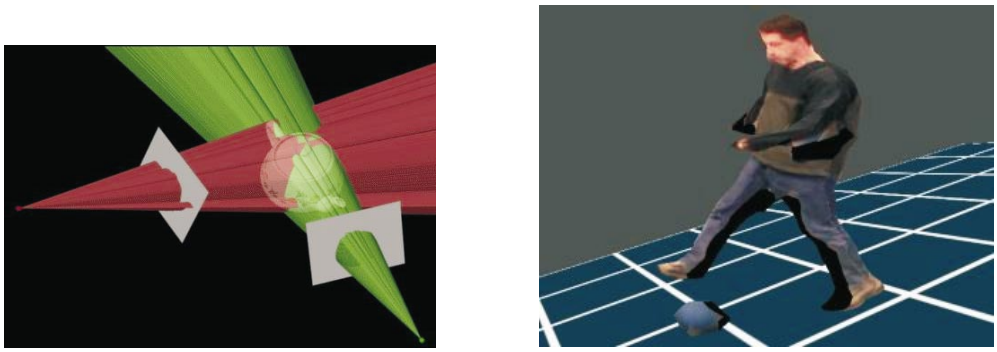

Figure 2.12: Visual hull. Left: Extruding and intersecting the object's silhouette from each reference image yields an approximation of the 3D object shape. Right: example of a person moving in front of the camera, reconstructed and rendered into a virtual environment at interactive frame rates. Pictures courtesy of W. Matusik and C. Buehler.

2.8 Comparison of IBR Techniques

In the previous sections we have sketched the most prominent approaches to reconstructing arbitrary views of a scene through image-based rendering. The choice of the "optimal" technique is strongly application-dependent, since the methods differ in various respects. In what follows, we present the most basic and important characteristics of IBR approaches, and compare the presented techniques with respect to these criteria. After discussing these points we try to compress this rather complex information into a short overview in Table 2.9.1 on page 34.

2.8.1 Freedom of Navigation

All the presented methods store discrete subsets of the light distribution in a scene, which in its most general form (apart from wavelength and time dependencies) is the five-dimensional *plenoptic function* [1]. However, almost every method restricts the viewer to move only in certain sub-spaces, or, in other words, some techniques yield satisfactory results only if the viewer stays within certain regions (or on certain paths) in space.

Billboards/Impostors only represent an object from a tight range of viewing directions and a very limited (far away) range of distances between the viewer and the object. They are neither suited for rotation of the viewer, nor for drastic changes in the viewing distance. *Panoramas* restrict the viewer to a single viewpoint, but allow rotations and zooming. Extended systems allow hopping between distinct viewpoints. *View interpolation and morphing* techniques allow the viewer to move and rotate freely within a certain region of space. However, since only two or three reference images are used for interpolation, this viewing region is very limited, and data

will be missing when the viewer moves too far off the original views. Since the ray directions stored in a *layered depth image* are the same as in a traditional 2D image, the viewer must stay close to the original viewing direction. Although the layered approach can compensate for disocclusion artifacts when the viewer is moving, distortion and under-sampling will occur if the viewing angle differs too much from the original one.

Movie maps also allow for translation and rotation of the viewer, but only on a predefined path and for few predefined viewing angles. *Concentric mosaics* restrict the viewer to stay close to the ground plane in order to avoid vertical distortion (or missing data, even if distortion correction is used). Similarly, *multiple-center-of-projection images* restrict the viewer to stay close to the recorded path. *Light fields and Lumigraphs* allow free translation, rotation, and zooming of the view, but are restricted to the outside (or inside, for outward-looking light fields) of a bounded 3D region. This means that the viewer cannot explore the inside of a scene, even if it consists of multiple objects with free space in between them. This could be overcome for some scenes by using a separate light field for every object and displaying the scene by rendering this set of non-overlapping light fields. *View-dependent texturing*, *unstructured Lumigraph rendering*, *visual hulls*, as well as *surface light fields* are defined on the objects' surface (or an approximation of it) and thus do not suffer from the above-mentioned constraints. They allow the viewer to navigate freely through a collection of arbitrarily placed images/objects, also correctly treating inter-object occlusions through the use of geometric information. *Layered depth cubes* and *LDI trees* also allow unconstrained navigation, since they can encode multiple objects along the same viewing ray, so the viewer can travel in between these objects even if they are stored in a single image-based representation.

2.8.2 Geometric Information and Complexity

Panoramas and *movie maps* only show the scene from the original viewpoints, so they do not need to take care of parallax, occlusion, and distortion. They can represent any kind of scene. Similarly, impostors can represent arbitrarily complex objects if the viewing restrictions are tightly watched (large distance from the object and no drastic changes in viewing direction). Pure *light field rendering* can also represent any kind of object, but the rendering is blurry if the scene geometry extends far from the light field image plane (an effect very similar to a limited focal depth). In that case the scene must be sampled in an extremely dense fashion (see storage efficiency below). On the other hand, light fields have the big advantage that they do not require any geometric information of the scene, so in principle they work for all kinds of scenes. The same ideas apply to *concentric mosaics*. *View morphing* can be used on any kind of geometry, but here the user must manually specify a suitable (and probably rather complex) set of feature correspondences between the two views.

Many techniques require a sampled representation of the scene geometry, usually as per-pixel depth values. Although this representation is very convenient for most kinds of geometry, there are problems in special cases such as fur and hair. The sampled depth values are usually

not *consistent* across the different views (e.g. the viewing rays hit different strings of hair), so the smooth reconstruction of the geometry by warping will fail. This applies to all warping-based techniques such as *view interpolation*, *plenoptic modeling*, *LDI*, *LDC*, *IBO*, *LDI tree*, and *delta tree*.

A similar problem can be observed for all techniques that require a polygonal scene representation, since complex fine-grained geometry like hair and fur cannot be represented by polygons in a satisfactory way. Techniques that especially suffer from this disadvantage are *view-dependent texturing*, *Lumigraphs*, *unstructured Lumigraphs*, and *surface light fields*. However, Lumigraph techniques and view-dependent texturing can somewhat compensate for the problem by using an "averaged" geometry and using a directionally very dense set of reference images. For surface light fields it is not clear how much the very sophisticated data preprocessing would suffer from incorrect geometry, since the normal directions are the basis for exploiting coherence. The *visual hull* approach is geometrically limited, since the silhouette intersection cannot reconstruct concavities that are not part of one of the silhouettes.

2.8.3 Photometric Complexity

The presented approaches also differ with respect to the photometric complexity that they can encode and represent. The most basic distinction can be made between techniques that are only suited to represent diffuse objects, and techniques that can represent appearance varying with viewing direction. All techniques that only store a single directional sample for each scene point fall in the first category, including simple *texture mapping*, *layered depth images*, *layered depth cubes*, *LDI trees*, and *delta trees*. Some other warping-based techniques employ a simple Z buffer or occlusion-compatible rendering, and thus only use a single ("front-most") sample for color reconstruction. This is also not suited for smooth reconstruction of view-dependent effects. From our list of techniques, this applies especially to *plenoptic modeling*.

Panoramas and *movie maps* display zoomed versions of the original image from the original viewpoint, so they can capture the exact appearance of the object. Since the viewer is not able to change position continuously, view-dependent effects will never appear and thus do not need to be modeled.

All other techniques sketched in this chapter use interpolation between multiple original samples for each pixel, and so directionally changing appearance can be represented in principle. However, there are two important factors that are responsible for the reconstruction quality of view-dependent effects: the smoothness of the reconstruction, especially when the viewer is moving continuously, and the directional sampling density that can actually be utilized for highly view-dependent surfaces. *View-dependent texturing*, for example, interpolates between multiple input images for each triangle, but does not ensure a smooth transition across triangle edges because each triangle is treated independently. In contrast to pure light fields, *Lumigraphs*, *unstructured Lumigraphs* and *visual hulls* usually do not employ such a dense set of reference images, and are thus bound to "smear" sharp view-dependent features. However,

in principle they can compensate this by adding more reference images, at the cost of storage explosion. *Surface light fields* use very sophisticated preprocessing in order to compile a very large number of directional samples into a compact and compressed form (so-called *Lumispheres*), so they are best suited for representing glossy and highly specular surfaces.

2.8.4 Further Issues

There are some more issues that distinguish one technique from another. One important point is **storage efficiency**. In general, for a non-diffuse scene, the more navigational freedom is required, the more samples need to be stored, and the more the dimensionality of the problem increases (e.g., from 3D for images stored along a predefined path, to 4D for a light field). This is why *light fields* and *Lumigraphs* are most storage-intensive. Compression schemes exist, but will either only save a constant factor or slow down rendering (cf. Section 3.3). *Unstructured Lumigraphs* potentially exhibit even less coherence between the reference images, so using compression is even harder with that approach. *Surface light fields* are a very compact representation of objects even with complex photometric properties because of their compact encoding of reflected light in the local coordinate system of the surface, which exploits coherence relatively well.

Another important issue is that whereas some techniques directly use the image information for rendering, other require very long **preprocessing** of the data before it can be rendered. Especially *surface light fields* require very computation-intensive processing, but also the rebinning phase for *Lumigraphs* is very costly, and similarly *layered depth images* need to be constructed by resampling and merging the original image samples. This is one strength of *panoramas*, *concentric mosaics*, *unstructured Lumigraphs*, or *warping*-based techniques, since they render directly from the original images. The *visual hull* approach has even been demonstrated as a complete system that can capture, process, and render a scene at interactive frame rates.

This leads to another important issue, which is **data acquisition**. One common problem is to acquire real-world images with cameras that match the constrained viewpoint arrangements of representations such as *light fields* or *concentric mosaics* (viewpoints exactly in a common plane or on exact circles). Specialized capture gantries exist, but are hardly desirable in general. The biggest problem is the acquisition of geometry information. The only robust and accurate way to acquire the geometry of a scene is to use a real-time range finder, e.g. based on laser technology. Reconstructing dense geometry information directly from images is still an open research problem. So from this point of view, the best choice would be one of the "purely" image-based techniques such as *panoramas*, *concentric mosaics*, and *light fields*.

2.9 Conclusions

We have presented a survey of image-based rendering algorithms for viewing a 3D scene. After sketching the most prominent approaches in this field, we have compared the actual techniques with respect to different criteria such as navigational freedom, geometric and photometric complexity, and others. Table 2.9.1 (page 34) gives a short overview of the different techniques and their constraints as well as other important properties.

To summarize, there is no “perfect” universal image-based rendering technique. Simple and memory-efficient techniques usually constrain the space of possible viewpoints or are restricted to diffuse scenes. A general and purely image-based technique such as the light field is extremely memory-consuming due to the large redundancy, caused by the parameterization that cannot adopt to the scene’s individual structure. Surface-based representations such as surface light fields are much more compact and help to achieve very high quality representations within acceptable memory bounds, but they require much preprocessing as well as a good geometric model of the object. Techniques such as Lumigraphs and unstructured Lumigraphs yield intermediate results in quality and memory consumption, since they better exploit coherence through the geometric proxy, but are by far less compact than a surface light field.

So the choice of the right technique strongly depends on the desired application. For a virtual walk around an object on a ground floor, a technique like concentric mosaics should give the best results. For jumping between different viewpoints in space and then looking around, panoramas are suited best. For representing visually complex artifacts in a virtual museum, the right choice is to use surface light fields. But all the above-mentioned techniques leave room for improvement. For many potential applications such as “instant” capture and viewing of a detailed scene (photo-realistic tele-presence) there is no optimal technique so far. Thus image-based rendering, especially in the direction of light fields, Lumigraphs, and unstructured Lumigraphs, still gives rise to a lot of interesting research activity.

Technique	Navigational Constraints	Geometric Constraints	Photometric Constraints	Other Issues
impostor	range, dir	–	–[7]	
panorama	fix viewp.	–	–[7]	
view morphing	betw. 2 views	manual[3]	–	
view interpolation	betw. 2 views	depth[4]	–	
plenoptic modeling	betw. 2 views	depth[4]	diffuse	
movie map	fix viewp.+dir	–	–	
concentric mosaic	ground plane	–	–	(-)acquisition
light field	outside[1]	–[2]	–	(-)acquisition, memory
LDI	view dir range	depth[4]	diffuse	
LDC, IBO, LDI tree	–	depth[4]	diffuse	
MCOP image	predefined path	depth[4]	–	(-)acquisition
delta tree	outside	depth[4]	diffuse	
view-dep. texture	–	proxy[5]	not smooth	
surface light field	–	polygons[6]	–	(+)memory
Lumigraph	outside[1]	proxy[5]	–[8]	(-)memory
unstruc. Lumigraph	–	proxy[5]	–[8]	(-)memory
visual hull	–	"simple" shape[9]	–[8]	(+)dynamic scene

[1)] outside or inside a certain region of space for inward/outward-looking light fields, respectively.
[2)] blurring/ghosting when object is off the imaginary image plane.
[3)] requires manual specification of feature points.
[4)] requires per-pixel depth or disparity information.
[5)] requires geometric approximation, e.g. coarse polygonal mesh.
[6)] requires polygonal representation of the scene.
[7)] different impostors would be needed for different viewing directions.
[8)] usually sampling is not dense enough to capture sharp highlights.
[9)] the silhouette-active shape cannot represent concavities that are not visible in one of the input silhouettes.

Table 2.9.1: Comparison of the presented techniques. For more detail on the mentioned criteria, please refer to Section 2.8. (-) and (+) in the last column denote positive and negative properties of the listed techniques, respectively.

CHAPTER 3

Light Fields and Lumigraphs in More Detail

> The sky above the port had the color
> of television tuned to a dead channel.
>
> William B. Gibson, *Neuromancer*

Having positioned light fields and Lumigraphs in the field of image-based rendering, we have seen that they are interesting approaches to representing a 3D scene by a collection of images and, if available, a geometric proxy for correlating the image data. Both light fields and Lumigraphs receive a lot of attention, since the principle is simple, rendering is efficient, and the representation can be exploited and extended in many ways.

In this chapter we will go through the details of light field and Lumigraph representation, rendering, and acquisition, since these components build the foundation for our own work presented in the following chapters. In Section 3.1 we start with the general idea, followed by parameterization and discretization issues in Section 3.2 and by coding and compression in 3.3. In Section 3.4 we sketch known techniques for efficient light field rendering before we extend the light field concept to Lumigraphs with additional geometric information and depth correction in Section 3.5. The so-called *unstructured* Lumigraph approach, which probably

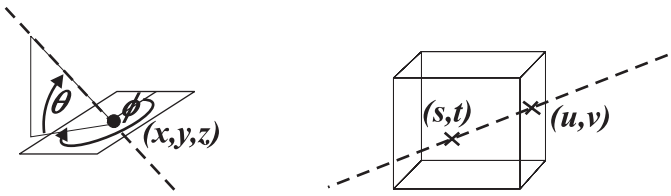

Figure 3.1: Left: Sample of a five-dimensional plenoptic function: light is expressed in terms of an exit point (X, Y, Z) and a ray direction (θ, ϕ). Right: sample of a four-dimensional light field: light is expressed in terms of the two intersections (u, v) and (s, t) of the light ray with the boundary of a bounded 3D region. This parameterization "accumulates" the light emitted from all points along that ray within the bounded region.

represents the most versatile and powerful light field-based approach so far, follows in Section 3.6. In Section 3.7 we list some popular extensions and applications of light fields, and Section 3.8 sketches the different approaches for acquiring light field data. In Section 3.9 we sum up this chapter and discuss the most prominent properties and problems of the light field approach.

3.1 Plenoptic Function vs. Light Field

The term *plenoptic function* has been coined by Adelson and Bergen in 1991 [1]. The underlying observation is very old and can, for example, be traced back to the work of Leonardo da Vinci [62]:

> The body of air is full of an infinite number of radiant pyramids caused by the objects located in it. These pyramids intersect and interweave without interfering with each other during their independent passage throughout the air in which they are infused.

Leonardo described the idea of the *distribution of light in space*, which is a function describing the radiant flux flowing from every point (x, y, z) into every direction (θ, ϕ) (see Figure 3.1). Each of these position-direction pairs forms an independent *pencil* of light. In addition to light source position and light direction, light can also travel independently at different wavelengths λ, and, in general, the light distribution varies with time t. So the complete plenoptic function P is seven-dimensional, $P = P(x, y, z, \theta, \phi, \lambda, t)$. P describes what *can be observed* by a camera or a human observer. The time dependency can be dropped for describing static scenes. Since the plenoptic function is most often evaluated separately for a number of discrete color channels like RGB (red, green, blue), wavelength dependency is also usually dropped, leaving us with a five-dimensional function.

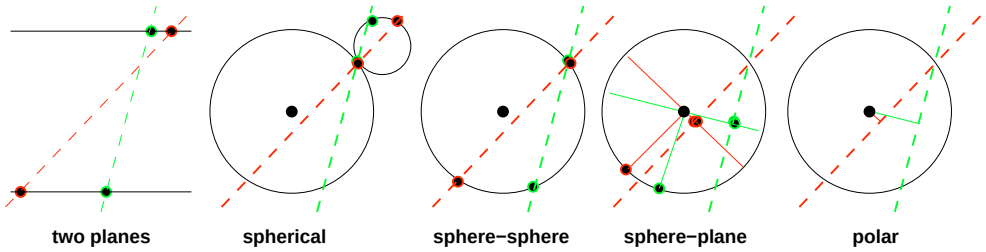

Figure 3.2: 2-D sketch of light field parameterizations, for two different rays to be parameterized (dashed lines). The solid lines correspond to planes in 3D, and the circles correspond to spheres. Please refer to Section 3.2 for the details.

Gershun [33] was the first who scientifically discussed the *light field* as a function describing brightness emitted from a certain point in a certain direction. His work was published in 1936 in Russian, and translated to English in 1939. Levoy and Hanrahan [74] as well as Gortler et al. [37] introduced the term to the computer graphics community in 1996. These latter two papers have coined the meaning of "light field" as we want to understand it in the context of this work.

A *light field* as used in [74, 37] is a discrete representation of a four-dimensional subset of the plenoptic function. The reduction from five to four dimensions is due to the fact that a light field only represents the light leaving a bounded region of 3D space, or light falling into such a region, respectively. Figure 3.1 illustrates this principle. This restriction to four dimensions implies that the observer's viewpoint must be placed outside the bounded region, in free space without any occluders, light sources, or other objects influencing the light distribution. The viewer cannot move to the inside of the bounded region, since no information is available from where the light is actually emitted[1]. We only know the amount of accumulated light passing through the boundary of the region.

One of the first questions that comes to mind is how to best parameterize the light field, and how to sample it in order to obtain an efficient discrete representation.

3.2 Parameterization and Discretization

The light field parameterization determines how all possible light rays can be addressed. Figure 3.2 shows the light field parameterizations that have been proposed in the computer graphics literature during the last years. In a *two-plane* parameterized light field [74, 37], a ray (dashed lines) is parameterized by its intersection with two parallel planes (solid lines). Such a pair of

[1] Due to clarity of the presentation, we leave out the dual case of an outward-looking light field, where the viewer is placed inside the bounded region, and where the light field represents the light coming through the boundary into the region.

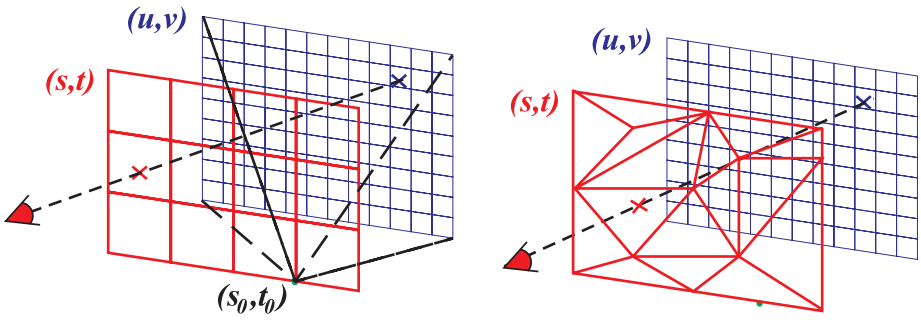

Figure 3.3: Two-plane parameterization and discretization. Left: One *light slab* in a two-plane-parameterized light field. All rays through one point (s_0, t_0) in one plane yield a sheared perspective image in the (u, v) plane. Right: Irregular 2D eyepoint mesh together with a regular 2D image for each eyepoint.

parallel planes (or all rays that pass through them) is also called a *light slab*. In order to represent all possible directions, six light slabs are combined into a single light field (cf. Figure 2.8 on page 23). Gu et al. [39] discuss some interesting theoretical properties of the two-plane parameterization and the corresponding ray space. *Spherical* light fields [54] use the intersection with a positional sphere (large circle) and with a directional sphere (small circle) placed at the positional intersection point. *Sphere-sphere* parameters [12] are determined by intersecting twice with the same sphere. *Sphere-plane* coordinates [12] consist of the intersection with a plane, and the normal direction of that plane. The plane is chosen perpendicular to the ray and passes through the center such that its normal can be represented by a position on a directional sphere. *Polar* coordinates [132] are given by finding the point on the ray closest to the center and then using the polar angles, the distance, and the rotation of the ray within the tangent plane.

Emilio Camahort's thesis [40] contains an analysis and discussion on the uniformity and invariance under typical transformations of light field representations. He proposes a variant of the sphere-plane representation called direction-and-point parameterization (DPP), and shows that this DPP is bound to produce less artifacts than other parameterizations under a number of measures for pixelization artifacts.

Discretization of the light field data is usually done by a regular (or approximately uniform) sampling in each of the four parameter dimentions. For the two-plane parameterization, this implies that we can see the light slab as a regular array of sheared perspective views (see Figure 3.3, left), with the "eyepoints" in one of the planes, and "pixels" in the other plane. This is why they are also called *eyepoint plane* and *image plane*, respectively. This strong resemblance with "classical" 2D images results in two strong advantages: captured 2D imagery can be inserted into the light field by simple resampling (shearing), and efficient rendering can be performed by drawing texture mapped rectangles (see Section 3.4).

The sphere-plane parameterization also has the interesting property that it resembles a collection of orthographic projections with the projection direction begin represented on the directional sphere. For the two-plane parameterization, the same can be achieved by setting the eyepoint plane to infinity.

The sphere-sphere and sphere-plane parameterizations have mainly been proposed in order to achieve a more uniform sampling of the complete space of rays passing through the bounded region [12]. Camahort and Fussel [11] also provide a comparison of different parameterizations with respect to uniformity and the related rendering artifacts.

Sloan et al. [125] propose an adaptive discretization scheme for the two-plane light field. They show that it is easily possible to use irregular eyepoint meshes instead of the regular eyepoint grid (see Figure 3.3, right), and to use this mesh directly for hardware-accelerated rendering. More detail about this will follow in Section 3.4.

3.3 Coding and Compression of Light Fields

As already sketched in the previous sections, a light field is a four-dimensional function that can also be regarded as a 2D array of 2D images. The discrete representation of such a structure can be very memory-intensive even for a relatively coarse regular sampling of the function E.g., 32x32 images at 256x256 RGB pixels each results in 192 Megabytes of data. One simple method for reducing this is to use one of the well-known image compression techniques to exploit the redundancy in two of the four dimensions. However it is clear that a much better compression can be achieved when removing redundancy in all four dimensions (especially since two nearby views of the same object are very similar).

One important issue when designing a compression scheme for light fields is that the decompression algorithm must allow for random access of the stored data. If this requirement is not met, the rendering algorithm either cannot perform at interactive speed, or the whole data must be decoded before rendering. Another desirable property is to have a true multi-resolution representation of the light field, so that the rendering can properly adopt to the current viewing conditions.

Levoy and Hanrahan [74] use vector quantization (VQ) [32] plus an additional Lempel-Ziv (LZ) encoding for compression. VQ allows random access during decoding and gives a constant compression ratio of up to 24:1, whereas LZ encoding is only used to reduce disk storage since the complete light field must be LZ-decoded before VQ decoding for rendering can be applied. Heidrich et al. [48] show how to use the so-called *texture color table* OpenGL extension (available on certain SGI systems) to render vector-quantized light fields with the support of graphics hardware.

For the spherical parameterization (see Section 3.2), Ihm et al. [54] have proposed and demonstrated a two-dimensional wavelet decomposition that suffers from the same problems as all 2D compression schemes since it cannot exploit coherence in all four dimensions.

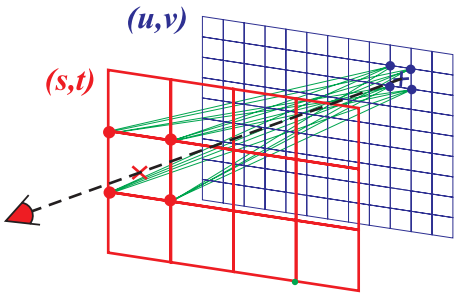

Figure 3.4: Reconstruction of light along a desired viewing ray by quadri-linear interpolation of the 16 nearest rays in the light field. These are the rays that connect the 4 nearest (s, t) sample points with the four nearest (u, v) sample points.

Kiu et al. [63] as well as Magnor and Girod [82, 84] proposed compression schemes similar to video encoding, where the images are decomposed into blocks, and the whole image data stream is represented by simple operations that predict most blocks from a smaller set of reference blocks. Magnor and Girod further proposed an encoding scheme that is based on *disparity compensation* [83, 79]. As we have seen in Section 2.4, with the use of disparity or depth information we can predict where to place the block in a different image if we know the relation of the two points of view. Block-based compression schemes are reported to yield high compression ratios of up to 1000:1, depending on the desired image quality.

Several authors [68, 104, 81, 105] have proposed true four-dimensional wavelet decomposition in order to encode and compress light fields, allowing for optimal redundancy exploitation as well as for multi-scale representation of the data. However, the approaches by Lalonde and Fournier [68] as well as by Magnor and Girod [81] are not suitable for decompression of large light fields during interactive rendering. Peter and Straßer [104, 105] were the first who achieved true interactive rendering of 4D wavelet-decomposed light fields by means of a very carefully designed coding and caching scheme. The demonstrated compression ratios are in the range of 25:1 to over 100:1, depending on the desired image quality.

For more detail about light field compression, the interested reader is referred to the work of M. Magnor [86] and I. Peter [105].

3.4 Efficient Light Field Rendering

Rendering a light field means reconstructing the light along the desired viewing rays from the "ray database". Different reconstruction filters can be applied, and for pure software-based reconstruction from a regularly sampled light field, the most popular choice is to use quadri-linear interpolation. Figure 3.4 illustrates this process, where the light traveling from the light field region to the viewer along a certain viewing ray is reconstructed by interpolating between

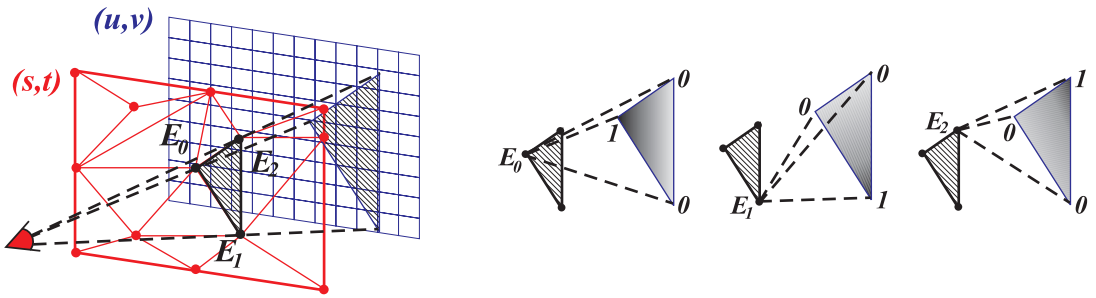

Figure 3.5: Light field rendering based on texture mapping. Each projected triangle on the (u, v) plane is drawn with three different textures, corresponding to each of the three contributing eyepoints E_0, E_1, E_2. The gray level gradients illustrate the alpha weights used to blend the three contributions. The weights are set to 1 in the eyepoint that is associated with the current texture, and 0 in the other two vertices.

the four nearest neighbors on each plane, resulting in 4x4 rays to interpolate from.

Levoy and Hanrahan [74] proposed a ray-casting-based light field rendering algorithm purely done in software which performs well on common desktop hardware. At the same time Gortler et al. [37] proposed an algorithm based on texture mapping that is very well supported by modern graphics hardware. This algorithm has been extended to irregular eyepoint meshes later on by Sloan et al. [125]. They propose to dynamically adapt these irregular meshes in order to match rendering time constraints and to adapt the set of used viewpoints (and textures) to the available resources of the graphics hardware.

Let us assume that the eyepoint plane is partitioned into a triangle mesh. Each viewing ray passing through that plane intersects one of these viewing triangles. The light along that ray should be interpolated from the three eyepoints that form the triangle, combined with the four pixel positions that are closest to the intersection of the ray with the image plane.

This can be done using a combination of texture mapping and alpha blending (cf. Figure 3.5). The triangle is drawn three times. For each of the three eyepoints E_0, E_1, E_2, the triangle is drawn with the texture corresponding to the eyepoint's image data. Furthermore, alpha blending [99] is used, and the alpha weights are set to 1 at the currently active eyepoint, and 0 at the two other vertices of the triangle. Within the triangle, the blending weights are interpolated linearly. If the triangle is drawn three times with these blending weights, a ray very close to an original eyepoint will be only reconstructed from the corresponding reference view, and the influence of this reference view will decrease linearly towards the other eyepoints, ensuring a smooth transition across the triangle edges.

Camahort [12] et al. have transferred the texture-based rendering to the sphere-plane parameterization by subdividing the directional sphere into polygonal patches that can be treated in very much the same way as the eyepoint triangles in the two-plane parameterization. Sloan

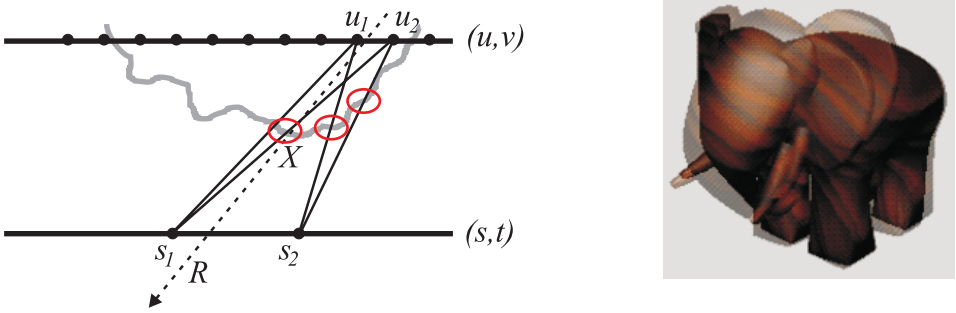

Figure 3.6: Blur problem for light field interpolation without using depth correction. The viewing ray R corresponds to the light emitted by the surface near X, but the reconstruction will interpolate from the nearest samples on the (s, t) and (u, v) planes, representing different surface locations (red circles). If the correct depth value for the ray X is known, the sample positions (texture coordinates) used for interpolation can be modified so that the sample rays represent the correct part of the scene. The picture on the right illustrates the effect of blurring and ghosting in an uncorrected light field with only 2x2 reference views.

et al. [126] have designed a rendering algorithm that can render very large Lumigraphs on parallel graphics hardware such as an SGI Onyx InfiniteReality machine. Vogelgsang and Greiner [137] give many useful hints for designing an efficient and practical implementation of texture-based light field rendering, including optimal slab geometry setup, texture slot management, and details on adaptive depth correction.

3.5 The Lumigraph: Adding Geometry Information

Although the basic light field idea sounds convincing, there is one big drawback apart from memory consumption. Looking closer at the interpolation scheme, we find that it only works correctly if the object is located in the imaginary image plane (u, v). The further the object is located off that plane, the more the interpolation will choose the wrong samples to interpolate from. Figure 3.6 illustrates this problem: for reconstructing the light along a certain viewing ray R, the correct solution would be the radiance emitted by the 3D surface near point X in the scene. However, the "nearest" samples used by the interpolation scheme represent different surface locations (red circles, e.g. the ray between s_2 and u_2), since the surface is off the (u, v) plane.

This problem leads to a very inconvenient behavior: the light field reconstructions become blurry for certain parts of the scene, and in extreme cases the objects will also spawn "ghost" versions of themselves, in such a way that each object seems to appear multiple times, as can be observed in the right picture of Figure 3.6.

In order to solve this annoying problem, Gortler et al. propose the so-called *adaptive depth correction* [37]. The basic idea is relatively simple: given an approximation of the scene geometry, one can actually cast some viewing rays, intersect the scene geometry, and use the resulting information about the scene position for depth-correcting the interpolation function. In principle, one would need to test every pixel. Since this is computationally too expensive, Gortler et al. cast rays through all eyepoints plus the centers of the eyepoint triangles. If the detected depth values for a triangle are approximately the same, they stop this procedure; if they differ by more than a certain threshold, the triangle is subdivided further, and more depth rays are shot. Once the depth values are determined adaptively this way, the resulting triangles are rendered as described before, but the texture coordinates are modified in such a way that they represent the correct scene locations for the desired viewing rays.

Assuming geometry information about the scene, Gortler et al. also propose to insert arbitrary views into a Lumigraph by a so-called *rebinning* process. They capture images using a hand-held camera, and after reconstructing the scene geometry using voxel coloring (see Section 3.8), they insert the captured views into the Lumigraph by first reprojecting the viewing rays into their corresponding bins in the Lumigraph, using the geometry information in order to account for parallax when switching from the original viewpoint to the nearest ones in the Lumigraph. Next, they use a hierarchical push-pull algorithm to first average the inserted data on coarser representation levels, and then pushing the averaged data down to the finer levels again in order to fill holes in the image data.

Following the original Lumigraph paper, other authors have proposed to use different kinds of geometric proxies, apart from a polygonal model. For example, Camahort and et al. [12] have used a hierarchical binary volume in conjunction with ray casting in order to perform depth correction efficiently. Vogelgsang and Greiner [138] proposed using per-pixel depth maps directly for depth correction. Although this involves a limited search in the depth maps for every depth ray, Vogelgsang and Greiner are able to perform interactive rendering using this technique.

Chai et al. [14] present a theoretical analysis of the tradeoff between the amount of geometric information in a Lumigraph and the reconstruction quality. They describe how to optimally place the Lumigraph image plane and how to distribute the 2^n depth "planes" in space when using n bits of depth information. However, Chai et al.'s analysis is a purely geometric study, not including any photometric effects. Since these represent one of the most important limiting factors for the visual rendering quality, further studies on light field sampling are required in order to predict optimal sampling properties.

3.6 Unstructured Lumigraphs

Heigl et al. [53] apply a variation of the Lumigraph technique for using arbitrary sequence of images directly for rendering. They capture a series of images using a hand-held video camera,

perform an automatic calibration of these images [66], and reconstruct dense depth maps from the image data (cf. Figure 3.7). For rendering a novel view, they project all camera centers into the image plane and perform a triangulation of these projected cameras. With this triangulation, they render in very much the same way as described in the original Lumigraph paper, although they do not give any performance results, since they do not provide an implementation based on texture mapping. Also, their depth correction is restricted to the original eyepoint triangles, since for shooting arbitrary depth rays (as in the case of subdivided triangles) they would need to perform a search in the depth maps, as shown later by Vogelgsang and Greiner [138].

Two years later, Buehler et al. [9] proposed a technique called *unstructured Lumigraph rendering*, which goes in a similar direction as Heigl et al.'s work. Like Heigl et al., Buehler et al. use an arbitrary set of views, and project and triangulate these eyepoints for every novel view. Furthermore, they generalize the interpolation process by introducing heuristics to account for resolution control and occlusion, and for blending between more than just the nearest three views (e.g. to fill in missing information). The result is called a *blending field* that defines the weight of each reference image for every point in the desired image plane. Instead of adaptive depth correction, they project all scene vertices into the image plane, add more vertices from a regular grid, add the projected eyepoints, and use all the resulting vertices for tesselating the image plane. The blending field value in each vertex defines the alpha blending weight of each reference image. As usual, the blending weights are interpolated linearly within the triangles. Instead of the three "nearest" reference views, Buehler et al. employ a k-nearest-neighbor scheme (usually with $k = 4$), where the distance is defined by the angular deviation between the desired ray and the ray from the reference eyepoint. The triangles are actually drawn on the approximate scene geometry, using projective texture mapping of the reference images.

The principal difference between unstructured and classical Lumigraphs is the ability to use arbitrary views directly for rendering, without the need to rebin them into a regular structure (e.g. camera snapshots or video streams). Furthermore, since the "optimal" views for reconstructing a novel view are chosen according to heuristics, and since the light is assumed to be emitted from the approximate scene geometry, the viewer is no longer restricted to a certain volume, but can move freely inside the scene.

3.7 Extensions and Applications

Having discussed the basic techniques used for representation and rendering of light fields and Lumigraphs, we briefly sketch some extensions and applications that have been proposed in the literature. The work listed here shows that the light field can be used for much more than just viewing an object as it appeared at the time of the image acquisition.

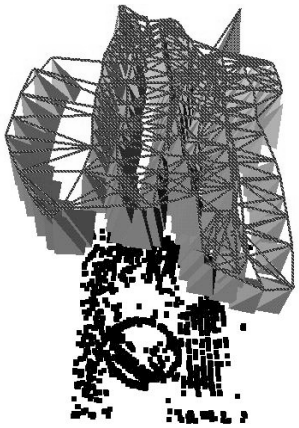

Figure 3.7: Heigl et al. [53, 66] use arbitrary real-world camera positions. The picture shows the reconstructed camera frustra for a hand-held video camera sequence. Picture courtesy of Benno Heigl, University of Erlangen-Nürnberg.

3.7.1 Dynamic Reparameterization

Isaaksen et al. [55] propose an interesting application and extension of the Lumigraph idea. Instead of trying to use a "true" geometric proxy, they exploit the same idea to simulate a *focal surface*. As already explained in Section 3.5, the light field interpolation will return the correct values for scene points that lie on the image plane, and will blur image regions that are far off that plane. This is very similar to the well-known *depth-of-field* effect in photography, where only objects close to the focal plane are actually in focus. Applying this idea, we can simply move the imaginary image plane during rendering, and assume it to be in an arbitrary position (varying distance from the point of view). The effect of this is that objects in that plane will appear in focus, whereas others will become increasingly blurred the farer they are away from the plane. For the Lumigraph depth correction technique, one uses arbitrary focal *surfaces* instead of a single plane (e.g. also multiple planes for different image parts).

In addition to just using this effect, Isaaksen et al. also propose a modified interpolation scheme that allows to influence the strength of the depth-of-field effect similar to the variable aperture of a camera. The basic idea is to enlarge the kernel of the reconstruction filter by including more further-off reference rays for reconstructing a viewing ray. In a texture-based implementation, this corresponds to blending from more than just the nearest three views. Isaaksen et al. use multi-texturing to achieve this. The more reference views are blended together, the larger the virtual aperture is, and the stronger the blurring effect for objects off the focal surface becomes apparent.

3.7.2 Canned Light Sources

Heidrich et al. [45] point to an important application of light fields for rendering and global illumination: the representation of complex luminaries by a light field, the so-called *canned light source*, and the illumination of a virtual scene using such a canned light source. They address several important issues that need to be resolved when using light fields for lighting simulation. They provide both an efficient ray tracing solution with adaptive sampling, and a solution for direct scene illumination using OpenGL, employing projective texture mapping and spotlight illumination. Their OpenGL solution accounts correctly for quadratic fall-off and the cosine weighting of incoming light, and can be combined with existing techniques for simulating hard and soft shadows using OpenGL.

3.7.3 Scene Relighting

Wong et al. [144] store a BRDF (for bidirectional distribution function) in a light field structure, rather than color values. For every ray in the database, they store 16 or 25 wavelet coefficients of the distribution function that describes how incoming light is reflected in all possible directions. Using this representation, they can illuminate a synthetic scene stored in the augmented light field using up to three light sources at near-interactive frame rates.

Lischinski and Rappoport [78] use a combination of light fields and layered depth cubes for representing and rendering highly reflective scenes. They do not only store radiance values, but also geometric information such as depth and normal in the light field. The method is computationally very expensive and cannot perform at interactive frame rates on contemporary graphics hardware.

3.7.4 Geometry Light Fields

Heidrich et al. [48] exploit the basic structure of a light field to encode scene geometry rather than exitant light. Instead of storing color values for each ray in the light field database, they store the reflection and/or refraction directions of the corresponding rays in a *geometry light field*. The basic difference to work such as that by Wong et al. (see above) is that this allows to decouple illumination computation from geometric information.

Color-coding the direction vectors and rendering them into the frame buffer results in a false-color image that encodes the incoming direction of light for each pixel in the image. This information can then be used in a second step to look up the incoming light from either an environment map or a second light field. In the first case, it is best to directly encode the environment map coordinates in the so-called geometry light field (e.g. half-way vectors for spherical environment maps). In the second case, four-dimensional light field coordinates need to be stored an rendered, and false coloring can be done using R, G, B, and alpha channels.

The second step of looking up the illumination in an environment map or light field can be done very efficiently by using the so-called OpenGL *pixel texture extension* [120] that adds

this indirection to the texture lookup stage. However, the depth of the frame buffer limits the resolution of the environment map, or illumination light field, respectively.

3.7.5 Volume Rendering

Patrick Ohly [101] proposed to use an extension of light fields for interactive volume rendering. In addition to the four dimensions that are needed to parameterize the rays in the database, he augments the light field by additional parameters like offset and steepness of an opacity mapping ramp as often used in volume rendering. Then he renders a volume from all different viewing positions and for all desired combinations of additional parameters, and stores the resulting images in the extended light fields.

As a result, the viewer can not only move, rotate, and zoom the volume, but also change the opacity mapping parameters interactively. Since the rendering is simply done by interpolation from pre-recorded images, no actual volume rendering needs to be performed online, and so the performance is reported to be much better than for some available volume rendering software. However, due to large memory requirements, the technique has only been demonstrated for very coarse samplings of the additional parameters.

3.8 Acquisition of Light Fields and Lumigraphs

Most of the sections so far have concentrated on representating and rendering light fields. However, there is another important point that must be considered when designing a light field application: the acquisition of image data and, if applicable, geometry data. These issues present problems both for synthetic and real-world data.

3.8.1 Light Fields and Lumigraphs from Synthetic Scenes

One interesting application of light fields is to use them as a representation for a global illumination solution. Illumination simulation can consume hours of computation time even for rendering a single view of a complex scene. It is therefore desirable to render multiple views in the form of a light field, so a viewer can later move around the scene interactively.

The acquisition of synthetic light fields or Lumigraphs is done by high-quality rendering of the scene for the desired viewpoints, e.g. by techniques such as ray tracing, path tracing, photon maps, radiosity, or hybrid methods. Implementations usually allow to query the per-pixel depth value together with the pixel color, or alternatively the polygonal scene geometry can be used for Lumigraph representations. If the renderer is very flexible, it might be possible to query exactly those rays that are to be stored in the light field (sheared perspective views). If not, it is easily possible to correctly set the viewpoint and viewing direction, and then resample the acquired images into the right views. Again it must be noted that resampling the data into a

different viewpoint is much more involved, since the parallax effect requires a 3D reprojection using geometric information.

The open problems during acquisition are: light field setup (location, orientation, and size of the image plane and eyepoint plane), and how to choose the best sampling resolution for both planes. The light field planes are usually specified manually, e.g. in such a way that the image planes cross in the center of the scene, and the eyepoint planes form a bounding box around the scene. Image resolution usually depends on the desired resolution during the interactive viewing application. Choosing the right eyepoints is a hard problem that has been addressed by several authors. Magnor and Heidrich [85] demonstrate how a regular eyepoint grid should be chosen, depending on the geometric properties of the scene. Chai et al. [14] also address this problem, but like Magnor and Heidrich only consider geometry. There is also a considerable body of work on so-called *viewpoint selection* or *view coverage* in other contexts of computer graphics and computer vision. For a recent overview of this work, see [135]. However, to the knowledge of the author no work so far has shown how sampling depends on photometric characteristics of the scene, e.g. on the specularity of materials.

3.8.2 Light Fields from Real-World Scenes

When capturing a real-world scene in a classical two-plane light field, this can only be done if a the camera positions can be controlled exactly, and if they match the eyepoint positions in the eyepoint plane (again, because resampling into different viewpoints would involve 3D reprojection using geometric information). Levoy and Hanrahan solve this by using a motor-controlled turntable and a robot gantry that places a camera in the desired positions on a regular grid [74]. Stanford University also started a project to build a *light field camera* that consists of a real camera matrix being able to capture a complete light slab at once [134]. Yang et al. [147] present a very interesting and much cheaper solution by mounting an array of simple lenses on a flatbed scanner. Although this is a very smart solution, the resulting image quality is not suited for high-end applications.

Another very costly solution is to use walls or even rooms completely covered by cameras. The most popular of these approaches are probably the *virtualized reality* project [56, 107, 57], and the *sea of cameras* approach [30] and similar setups used for tele-immersion projects like the *office of the future* [108]. Since normally these cameras are not placed in an exactly regular and planar structure, this would require a more flexible light field representation, as for example the unstructured Lumigraph approach.

3.8.3 Geometry Acquisition from Real-World Scenes

Besides capturing the image data, for representations such as the Lumigraph or for warping-based techniques we also need geometric information about the scene. For techniques that rely very much on the exact surface geometry (e.g. surface light fields), it seems imperative

to use a commercial 3D scanning device such as a laser range scanner or a 3D scanner based on structured light [7]. Usually these scanners are very expensive and require the object to be placed on a turntable etc., so this is not really practicable in non-controllable environments. However, projects like *Digital Michelangelo* [75] have shown what modern 3D scanning and imaging technology can achieve.

A much more fundamental goal of computer vision and, more recently, computer graphics, is to reconstruct the geometry simply from captured image data. This approach has some clear advantages since it inherently ensures consistency between image data and geometry data, and furthermore inexpensive imaging devices are becoming readily available. Nyland et al. [100] discuss in detail the advantages of having dense depth maps or equivalent geometry data available for image-based rendering.

Gortler et al. [37] have demonstrated the feasibility of such an approach in the context of Lumigraph acquisition. They place the object into a scene equipped with markers. Using the marker positions in the captured images, they can reconstruct the camera parameters such as position, direction, and focal length. Then they reconstruct a volumetric model from the set of calibrated images [128, 118, 67].

Koch et al. [66, 53] go a step further by employing a fully automatic *calibration* technique for determining the camera parameters, and by reconstructing dense depth maps from the calibrated image sequence. This latter step (depth map reconstruction) is also a hard problem that has driven a whole area of research for many years. The most prominent approaches is to reconstruct depth from image pairs by so-called *depth from stereo* [23]. More sophisticated approaches also use more than two images, e.g. three [2], or even a *camera matrix* [64].

The problem of such approaches is missing robustness. All of these methods exploit color correspondences between two (or more) images, which is bound to fail for a wide variety of scenes since diffuse materials as well as consistent lighting during the whole capture process must be assumed. Another problem is processing time, since none of the more robust and higher-quality approaches can be used at interactive frame rates, at least not without using specialized hardware, e.g. [25]. So it seems not possible to capture and render Lumigraphs of dynamic scenes instantaneously.

Vogelgsang et al. [139] propose a general framework for storing, resampling, converting, and displaying all the image and geometry data necessary for typical light field applications, allowing the integration of different geometric proxy types. Vogelgsang [136] also proposes a file format and API for storing image-based scene descriptions, also optionally including geometry data for the scene.

3.8.4 High Dynamic Range Acquisition and Rendering

Last, but not least, there is one more important issue when capturing a high-quality light field. So far, we have always talked of “color” as the representation of light in a light field, usually referring to RGB color triplets encoded in 24 bit (8 bit per channel). Of course this represen-

Figure 3.8: Sequence of images taken at various exposure times in order to reconstruct a single high dynamic range panoramic image. Pictures courtesy of Michael Gösele and Philippe Bekaert, Max-Planck-Institut für Informatik.

tation is not suited to account for the wide variety of light intensities found in the real world. However, most modern digital cameras as well as graphics displays are limited to even less than these 24 bits of color resolution.

In some cases, like illumination simulation or perception-related viewing applications, it is imperative to capture the image data with higher color resolution. For synthetic scenes, we can assume that a renderer can be implemented based on floating-point representation for color values. For real-world scenes, we must work around the limitations of the digital camera. Debevec et al. [20] have proposed a convincing way for doing this. Instead of capturing a single image from a certain point of view, they capture multiple images, using a sequence of different exposure times (see Figure 3.8). They combine the data from these various brightness levels into a single *high dynamic range* image that is able to capture all color details from darkest to brightest colors.

When displaying such high dynamic range colors on a computer display, one can go the opposite way and apply a so-called *tone mapping* operator [88] to the image. Tone mapping maps the real colors to display colors, taking into account different criteria such as the brightness distribution in the scene and the adaptive and non-linear nature of the human visual system. Most capturing techniques for static scenes can be extended by high dynamic range capture, except if the camera cannot stand still long enough to capture a series of images from the same point of view. Rendering algorithms can also be extended by an additional tone mapping step; however this can significantly reduce rendering performance. Unfortunately an in-detail discussion of tone mapping is beyond the scope of this thesis.

3.9 Discussion and Open Problems

Light fields and Lumigraphs present one of the most interesting approaches for image-based scene representation, since they can be used for scenes that are both geometrically and photometrically complex while providing very convincing rendering performance.

However, the pure light field approach is bound to be intractable for arbitrary scenes, simply because scene sampling must be extremely dense in order to yield high-quality reconstruction, presenting a hard challenge to acquisition and compression techniques.

The geometry-augmented Lumigraph approach is better off in that respect, since the additional geometric proxy decouples the required sampling density from the geometric properties of the scene to some extent. However, a similar problem arises because of the photometric complexity of the scene, which can only be captured by a denser sampling.

From these considerations it becomes clear that it would be very advantageous to have a fully adaptive representation for Lumigraphs, exploiting coherence in each of the four dimensions. This representation should be flexible enough to be used in all stages directly, from acquisition to rendering. Furthermore, the representation could help to steer the acquisition process, so that the best views for representing the scene are chosen automatically. The graphics group at ETH Zürich has recently presented such an approach based on a wavelet representation of the light field [80].

Another problem is that of adaptive depth correction. Implementations have shown that the adaptive algorithm as proposed by Gortler et al. [37] is very hard to control. Near the object boundaries the adaptive refinement will spend a lot of computation time during rendering, so the user must control the renderer using several thresholds and bounds, and the performance of the algorithm is highly dependent on the scene and the current view. The unstructured Lumigraph follows a novel approach and avoids adaptive depth correction. However, they need to use the complete geometry information for every frame, and the triangulation of the image plane is not consistent in time (triangles will flip when eyepoints and scene points move in different directions). Furthermore, they build a weighted sum of several heuristic metrics in order to compute their blending field, and it is not clear whether it is possible at all to find a set of weights that will give satisfactory results, independently of the scene and the viewpoint.

Another issue is that of polygonal geometry proxies. Although it is possible to acquire such a proxy both for real-world and synthetic scenes, it is always harder to reconstruct a consistent and high-quality polygonal model than to obtain just a sampled representation like a per-pixel depth map. Furthermore, sampled geometry seems to better fit the original spirit of the light field approach than a geometric model. Although Vogelgsang and Greiner [138] have recently demonstrated how to use depth maps for light field rendering, they need to rely on the adaptive depth correction algorithm, with the above-mentioned problems.

Last, but not least, there is much room for research on capturing and rendering of dynamic scenes. In principle light fields and Lumigraphs can use sampled image and geometry data, so these techniques should be well suited for combination with real-time scene acquisition. The recent work on polyhedral visual hull [89] goes into this direction (and employs modified unstructured Lumigraph rendering), but the purely silhouette-based geometry reconstruction is too coarse to give convincing results.

To sum up, there are many different problems that need to be addressed in order to make light fields and Lumigraphs a practicable and useful scene representation for everyday applications, touching all three phases of the process: acquisition (view selection, geometry acquisition), representation (compact adaptive data structure), and rendering (better depth correction and blending mechanisms). We will address several of these issues in this thesis and give

directions for possible improvements.

CHAPTER 4

Warping Within a Lumigraph

> It started in mud, as many things do.
>
> Tad Williams, *Otherland*

As explained in the previous chapter, rendering from a sparsely sampled light field introduces a large number of blurring and ghosting artifacts. This can be compensated by adaptive depth correction if additional geometric data of the scene is available.

Since adaptive depth correction is difficult to control (balance between speed and quality) and slows down rendering, we propose an alternative approach for cases where a constant frame rate and high image quality are more important than a compact representation in core memory. We propose to use *3D image warping* in order to generate more in-between views from the existing reference views, thereby *refining* the light field before the actual rendering step.

The warping formulae presented in this chapter are derived for the specialized case of two-plane parameterized light fields and eyepoints that fall into the light field's eyepoint plane, and are therefore highly efficient and surprisingly simple. The warping and compositing techniques described here lay the foundation for several of the later chapters, although the techniques will be modified and applied in different contexts later on.

4.1 Motivation and Outline of the Method

As we have learned during the brief history of image-based viewing in the past years, the user has to decide at some point between a "pure" image-based representation like the light field, and "geometry-augmented" representations like the Lumigraph. As explained in more detail in Section 3.5, pure light fields must employ very dense sampling of the scene in order to yield a reasonable reconstruction quality. This results in a huge amount of data that can only be compensated partially by compression techniques without trading in too much quality for compression efficiency.

In contrast, additional geometric information leads to a visible increase in reconstruction quality for lower-resolution sampling of the scene (cf. Section 3.5), since it establishes the correct correspondence between the image data and its actual position in 3D space. Therefore, it seems highly desirable to use this geometric information whenever possible.

So there are two very important questions to ask before designing an application that employs light field viewing:

1. *Can we rely on geometry information?*

2. *Can we afford depth correction?*

Of course these two questions are closely related, since geometry information is required in order to perform depth correction. However, there are cases in which depth information is available, but depth correction is too costly. For example, immersive virtual reality applications dictate limits on rendering time, and on-the-fly depth correction decreases rendering performance by a considerable factor [37, 137, 115]. Moreover, high-quality depth correction is usually performed *adaptively*, and that makes it very hard to predict the maximum time needed for rendering one frame.

So the purpose of the work described in this chapter is to transfer the use of geometry information to a preprocessing stage, e.g. during the loading and construction of the light field data, before the actual viewing. Instead of on-the-fly depth correction, we perform *Lumigraph refinement* by generating in-between views that increase the eye plane resolution. This way one can store a relatively compact low-resolution Lumigraph, and view it as a high-resolution light field that yields good reconstruction quality without performing depth correction during the viewing phase.

The methodology of Lumigraph refinement is based on warping input images and compositing the warped image data in order to obtain smooth and occlusion-correct results. To this end, we derive very efficient specialized two-plane warping formulae, and show how to perform the per-pixel depth test and blending operations.

The method can be outlined as follows. First of all, one must *choose the new eyepoints* to be inserted. This step is discussed for uniform eyepoint grids (Section 4.2.1) as well as for arbitrary eyepoint meshes (Section 4.2.2).

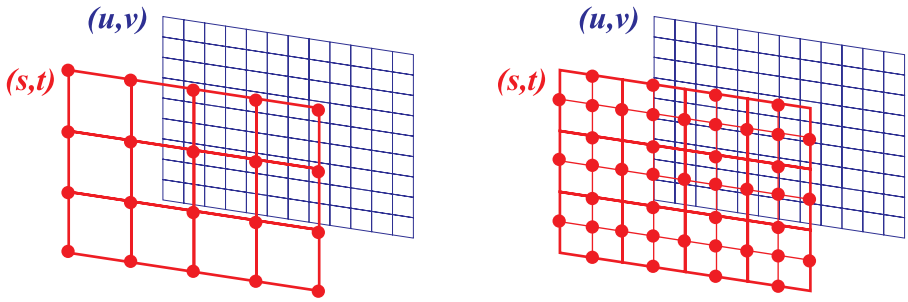

Figure 4.1: Refinement in a uniform eyepoint grid. The red circles show the eyepoints before the refinement (left), and the new eyepoints inserted by one level of uniform refinement (right).

Next, for every new eyepoint we need to *choose the source images* from which the new view/image is going to be reconstructed (Section 4.3).

All pixels in the source images for one new eyepoint are then reprojected (a.k.a. *warped*) into the novel view using specialized and very efficient warping formulae. We start with the "standard" case of warping within *sheared perspective projections* (Section 4.4.1), and also discuss some details of how to treat so-called *background pixels* (Section 4.4.2).

Finally, since multiple source images are warped into the same novel view, we need to perform a *compositing* step in order to reconstruct each final pixel value of the new image (Section 4.5). This compositing includes a *depth test* (Section 4.5.1) as well as *blending* the remaining color contributions to obtain the final pixel color (Section 4.5.2). Furthermore, we present an algorithm that performs hierarchical *hole filling* (Section 4.5.3) of the reconstructed image.

After presenting all the necessary steps in detail, we sum up the overall approach in a brief algorithmic description (Section 4.6), show results obtained with our experimental implementation (Section 4.7), and conclude this chapter with a discussion of these results and the prospects of our approach (Section 4.8).

4.2 Refinement: Inserting Intermediate Eyepoints

As explained in Chapter 3, a light field contains a set $\{I_0, I_1, \ldots, I_{N-1}\}$ of N images. Each image I_k is associated with an eyepoint (s_k, t_k) which is the center of projection used to generate the image. In the case of a two-plane-parameterized light field, all eyepoints lie within a common plane E, and all images I_k cover the same domain on the common image plane I. I and E are coplanar. The eyepoints on E are either defined on a regular uniform grid, or they are distributed arbitrarily and are organized in a triangle mesh (see again Figure 3.3 on page 38).

4.2.1 Uniform Eyepoint Grid

For uniformly sampled eyepoint grids, one level of refinement naturally means to halve the distance between two neighboring eyepoints. Figure 4.1 illustrates such a sampling pattern. Between every four vertices

$$\begin{array}{cc} (s, t+S_t) & (s+S_s, t+S_t) \\ (s, t) & (s+S_s, t) \end{array}$$

five new vertices are inserted at the following positions:

$$\begin{array}{ccc} & (s+S_s/2, t+S_t) & \\ (s, t+S_t/2) & (s+S_s/2, t+S_t/2) & (s+S_s, t+S_t/2) \\ & (s+S_s/2, t) & \end{array}$$

In these formulae, s and t denote the two coordinates in the eyepoint plane, and S_s and S_t denote the respective grid step size (distance between two neighboring vertices in each dimension). Overall, one-level refinement of an $N_s \times N_t$ grid leads to a new grid with $(2N_s - 1) \times (2N_t - 1)$ vertices. The refinement scheme can be iterated to obtain even higher-resolution grids.

For each of these new vertices we need to generate an image that represents the view from the corresponding eyepoint. Since a new vertex is inserted *between* two or four existing vertices, reconstruction of the novel view is always an *interpolation* between existing views, and never an *extrapolation* into a region of eye space that has not been sampled yet. This is an important fact and partially explains the high quality of the results presented later in Section 4.7.

4.2.2 Arbitrary Eyepoint Meshes

Of course, refining a Lumigraph does not need to be done in the grid-like manner described above. If the eyepoints are organized in a triangle mesh rather than in a grid, there are two basic methods of how to refine the mesh: choosing some mesh subdivision scheme, or adding arbitrary vertices and then re-triangulating the mesh, e.g. by performing a Delaunay triangulation [103, 24].

When using subdivision, one can choose among a number of operators with different properties [148]. The user can basically choose the right scheme depending on how many new vertices/views shall be introduced in each subdivision step, or based on geometrical properties of the resulting mesh (well-shapedness of the triangles). The only constraint is that one may

not choose a scheme that changes the vertex positions, since this would imply to throw away the input images and warp them into different view positions.

Two of the probably most common examples for such subdivision schemes are listed here to give an idea how the refinement of an arbitrary mesh works:

- 4:1 subdivision: one vertex is introduced in the center of every edge in the mesh, resulting in four times the number of triangles/vertices[1] .
- 3:1 subdivision: one vertex is introduced in the center of every triangle. New edges are inserted from the new vertex to the three triangle vertices, and the old triangle edges are flipped in order to yield well-shaped triangles again. This results in three times the number of triangles/vertices.

4.3 Choosing the Source Images

Once we know the eyepoint of the new view to be inserted into the Lumigraph, we need to choose the source images from which to generate that novel view. For this decision it is very important to note one fundamental assumption underlying light field rendering:

It is assumed that the sampling of the scene is sufficiently dense to reconstruct all possible light rays from only the closest neighboring views.

Without this assumption, *every* single input view in the light field might contain some piece of information needed to reconstruct a certain novel view, and we could not restrict the reconstruction/rendering process to the eyepoints close to the desired viewing ray. This is true for all known techniques: quadri-linear light field interpolation, adaptive depth correction, and k-nearest-neighbor unstructured Lumigraph rendering. For a more detailed discussion of the relation between scene geometry and light field sampling density, the reader is referred to [14].

In the case of a regular eyepoint grid refinement, a new vertex lies either in the center of a grid cell or on the edge of a cell, between two original vertices. If the new viewpoint is in the center, the four views that constitute the cell are chosen to act as the source images. If the new viewpoint is on the edge of a cell, one can either use only the two input views that form this edge, or include information from the "missing" directions by also warping from the nearest two grid cells in the respective direction. In our experience, it has always been sufficient to use only two source images for the edge viewpoints.

[1] The linear relation between the number of vertices and the number of triangles can be estimated roughly by using Euler's formula $V - E + F = 2$, which shows the linear relation of the number of vertices V, edges E, and faces F in an arbitrary polygonal mesh of genus 0 [103]. Furthermore, an inner edge in a triangle mesh is always shared by exactly two triangles, and a triangle consists of three edges, so $E = \frac{3}{2}F$ except for the mesh boundary. Putting this together, we get $V \approx \frac{F}{2}$ for the inside of a mesh.

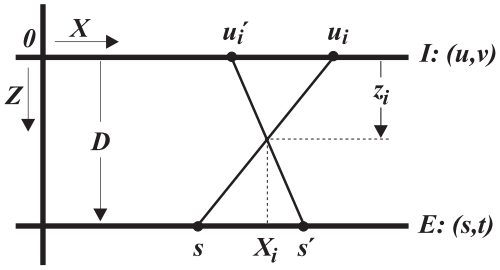

Figure 4.2: 2D sketch of the geometric situation for warping within a two-plane light slab with sheared perspective views. I denotes the image plane with (u, v) coordinates and E the eyepoint plane with (s, t) coordinates.

When refining a Delaunay mesh, each vertex in the inner part of a triangle is reconstructed from the three views that constitute the triangle, and a vertex on an edge is reconstructed from the two views/vertices on that edge, and optionally also from the remaining two vertices in the two triangles sharing that edge.

4.4 Two-Plane Lumigraph Warping

Now that we know the source images as well as the desired destination view, we can define a warping procedure that *reprojects* the pixels of the source image into the novel view. We will discuss the case of "standard" two-plane light fields using sheared perspective views, and furthermore discuss the treatment of background pixels. For more detail on light fields with orthographic projection, please refer to [49].

4.4.1 Sheared Perspective Views

Figure 4.2 shows the situation in a "standard" two-plane light slab. The image plane I and the eye plane E are parallel to each other and form a light slab. Without loss of generality, we assume that they are both coplanar with the XY plane. I is assumed at $Z = 0$, and E at $Z = D, D > 0$, so D is the distance between the two planes. The origin of the (u, v) coordinates on I is $(X, Y, Z) = (0, 0, 0)$ in 3D space, and the origin of the eye plane coordinates (s, t) on E is $(0, 0, D)$.

Now we want to reproject a pixel i at u_i in the original view s to its new location u_i' in view s'. Since we know the pixel's *depth* z_i (measured as the 3D point's Z coordinate), we can find the actual 3D point P_i that is associated with pixel u_i. Let X_i denote the u (equals s, equals X) coordinate of P_i. Then we have the following geometric relations [49]:

$$\frac{X_i - s}{D - z_i} = \frac{u_i - s}{D}, \tag{4.1}$$

$$\frac{X_i - s'}{D - z_i} = \frac{u_i' - s'}{D}. \tag{4.2}$$

Solving these for the new pixel position u_i' yields

$$\begin{aligned} u_i' &= u_i + \frac{z_i}{D - z_i} s - \frac{z_i}{D - z_i} s' \\ &= u_i - \frac{z_i}{D - z_i} \Delta s, \end{aligned} \tag{4.3}$$

where $\Delta s := (s' - s)$ is the translation of the viewpoint in s-direction (or the so-called *baseline* of the two eyepoints in that direction). Similarly, we get $v_i = v_i - z_i/(D - z_i) \cdot \Delta t$.

This means that for this specific setting the *pixel flow* $\Delta u_i := (u_i' - u_i)$ depends only on the pixel's depth z_i and on the baseline $\Delta s := (s' - s)$ connecting old and new camera position. This is much simpler than the warping equation for general camera transformations [95, 94].

In particular, it is possible to precompute and store the *two-plane disparity* $\bar{z}_i := z_i/(z_i - D)$ for each pixel i. The warping of one pixel then only requires two multiplications and two additions[2], one each for u_i' and v_i'.

When warping *within* a two-plane-parameterized light field, the pixel flow $\Delta \mathbf{u_i}$ is proportional to the pixel's two-plane disparity $\bar{\mathbf{z}}_\mathbf{i}$ and the baseline $\Delta \mathbf{s}$ between the old and the new viewpoint:

$$\Delta \mathbf{u_i} = \bar{\mathbf{z}}_\mathbf{i} \Delta \mathbf{s}$$

Please note that similar results can also be found in [37], where the geometric situation for depth correction in two-plane parameterizations is discussed.

4.4.2 Background: Scene Points at Infinity

Depending on the kind of scene data, a pixel i may not represent something in the actual scene, but rather contain the information that the ray through this pixel i does not intersect any scene geometry. In this case we assume $z_i = -\infty$. Such a pixel i is usually referred to as a *background* pixel.

If background pixels are simply understood as "holes" in the light field, they do not carry color information for the background, and can be discarded during the reprojection[3]. However,

[2] Please note that there is also some computation involved in mapping between the floating point coordinates (u, v) and the integer-numbered pixel indices in the actual source and destination images.

[3] In that case, the background is usually drawn first during rendering, followed by the scene drawn over / in front of it.

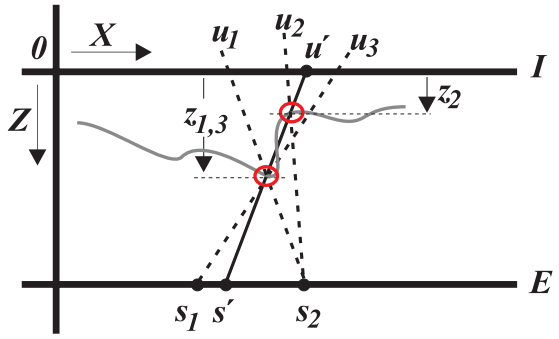

Figure 4.3: Compositing and depth test of multiple scene samples in a destination view s'. The grey curved line represents the true scene geometry that corresponds to the depth values. Samples u_1, u_2, and u_3 from two input views s_1 and s_2 will all be reprojected onto destination pixel u'. However, only u_1 and u_3 represent the front-most visible surface for view s', so these two samples will be blended together. Sample u_2 (with depth $z_2 < z_{1,3}$) is occluded and will be discarded.

if background pixels are color samples of a far-away background, we need to reproject these pixels in order not to lose this information. Since a background pixel is at infinity, the ray *direction* alone determines the background point. In a certain direction, we always see the same background fragment, independently of the viewpoint. This results in a simple "shifting" of the background pixel by the distance between the new and the old eyepoint:

$$u_i' \quad = \quad u_i + \Delta s, \tag{4.4}$$

and analogously, $v_i' = v_i + \Delta t$. Note that this approach of reprojecting the background is only valid for perspective views, and is not feasible for the sheared orthographic views discussed in the next section.

4.5 Compositing and Hole Filling

So far we have discussed how to select a set of source images for a novel viewpoint, and how to warp individual pixels from a source image view (s, t) into the desired view (s', t'). Since multiple images are used to reconstruct the novel view, more than one pixel may be reprojected to each pixel j of the novel image (given that it has the same resolution), resulting in a set C^j of *samples* for each destination pixel j:

$$C^j \quad := \quad \left\{ (L_k^j, z_k^j) \right\}, \tag{4.5}$$

where each sample consists of a color L_k^j and a depth value z_k^j. The samples represent some part of the scene seen through the destination pixel j (so they all fall on the same ray through j). However, the samples may belong to different scene points with different distances to the viewpoint, and some of the samples are probably "occluded" by others (cf. Figure 4.3). The remaining samples that belong to the same front-most point of the scene represent that point's appearance as observed from the different original views.

4.5.1 Depth Test

As motivated above, the different samples for a destination pixel may belong to different scene points, distinguished by their depth. So we need to compare the samples' depth values in order to find out which samples belong to the front-most (and thus visible) point (e.g. an occluding surface). Samples that lie behind this front-most point are simply discarded[4]. However, we need to keep *all* samples that belong to the front-most point, since the final pixel color is determined by blending these samples (see next section).

So the depth test procedure is rather straightforward: we determine the *minimal depth* (with the maximal z_k^j value $z_{\max}^j$) among all samples for one pixel j:

$$z_{\max}^j \quad := \quad \max\left\{ z_k^j \mid (L_k^j, z_k^j) \in C^j \right\}. \tag{4.6}$$

Then we discard all samples with a depth that differs from $z_{\max}^j$ by more than a certain *depth resolution* ϵ_z and store the surviving samples in the so-called *blend set* B^j:

$$B^j \quad := \quad \left\{ (L_k^j, z_k^j) \in C^j \mid z_{\max}^j - z_k^j \leq \epsilon_z \right\}. \tag{4.7}$$

This threshold ϵ_z can be chosen according to the properties of the scene and the resolution of the depth values. If the Lumigraph contains quantized depth values (e.g. linear depth coded in 8 bits), one can even use a threshold of 0, meaning that two samples on the same pixel are assumed to represent the same 3D point if their quantized depth value is exactly the same.

4.5.2 Blending

For each destination pixel j, the set B^j of samples that have "survived" the depth test is supposed to belong to the same visible point of the scene. Each sample $L_k^j \in B^j$ represents the color of this point, viewed from a different position. In standard 3D warping, the objects in

[4] Of course, this is only true for opaque samples. If the samples carry color and transparency information, only fully opaque pixels can occlude others, and the transparency coefficient must be included in the blending equation (see Section 4.5.2).

the scene are usually assumed to be Lambertian, meaning that a point is perceived to have exactly the same color from all possible viewing directions. If this was true, we could simply choose the front-most sample and use only that for reconstructing the color. This would allow for employing the hardware-supported depth buffer technique, and thus enable very fast compositing.

However, usually objects are not Lambertian, and furthermore, even for Lambertian objects it is better to blend between neighboring contributions for the sake of anti-aliasing (e.g. in order to avoid "popping" effects due to sampling problems). This blending is usually done by a weighted average of the samples:

$$\begin{aligned} L^j &= \frac{1}{W} \sum_{k=1}^{|B^j|} w_k^j L_k^j \ , \\ W &:= \sum_{k=1}^{|B^j|} w_k^j . \end{aligned} \tag{4.8}$$

The weights w_k^j for sampling can be computed in many different ways. The general idea is that a sample should be weighted according to how "close" its original sampling ray is to the viewing ray it is reprojected to. Since there seems to be no simple natural solution for defining "ray closeness", usually the weight is chosen as the inverse distance of the rays' intersections with the image plane, or as the inverse angular deviation of the original ray from the destination ray (see also Buehler et al. [9] for a discussion on this).

In our experiments concerning Lumigraph refinement, we have tested both constant weights (all samples have the same weight) and image-plane distance weights (decreasing weight with increasing distance between the two ray intersections on the image plane). We could find no visible improvement by using the distance measure, so we decided in favor of the computationally cheaper constant weights. Note that the constant weights allow for leaving out the expensive normalization term $1/W$.

Note, however, that the topic of non-constant weights will be brought up again in the next chapter, in the context of warping for arbitrary off-eye plane viewpoints.

4.5.3 Hole Filling

After warping and compositing all source pixels, we obtain the final pixel colors of the image associated with the novel viewpoint. However, due to various reasons that will be explained below, *holes* can appear in the constructed image.

Holes due to de-occlusion. When warping only a single original image to a new viewing position, parallax will inevitably uncover some areas that have previously been occluded. These

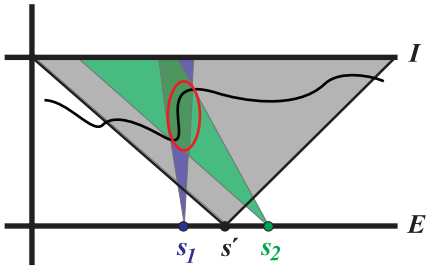

Figure 4.4: Gaps resulting from under-sampling caused by the angle under which part of a surface (circled area) is seen by the new viewpoint s'. While s_1 does not contain sufficient samples for the circled area, s_2 does.

regions show up as visibility gaps in the destination image since no pixels from the source are projected into these areas. These gaps can be seen as problems arising from extrapolation.

Luckily, the new eyepoints inserted into the Lumigraph are located within a triangle or rectangle of original eyepoints. Therefore, the new image can be formed by interpolation similar to [16] if the information from all source images is used. Furthermore, we can assume that the original Lumigraph is sampled fine enough for the environment, so that each point visible from any position on the (s,t)-plane is also seen by at least one of the nearest original eyepoints.

Holes due to under-sampling. The other kind of holes that often occurs in warped images is due to the angular difference under which a surface is seen. If the solid angle covered in the destination image is larger than in the source image, the warped pixels will not cover the whole area in the destination image, which will result in holes. These holes are caused by under-sampling the object. The solid angle increases if either the new viewpoint is closer to the object, or if it oberserves the surface under a more perpendicular angle than the old viewpoint.

In the Lumigraph setting, in which the eyepoints move perpendicularly to the viewing direction, the change of distance to an object is relatively small, so that the number of resulting gaps should be minimal. The second cause, however, is possible, as depicted in Figure 4.4.

If view s_1 was to be used as the only source image for the new viewpoint s', a large gap would occur in the destination image. However, in the setting of a Lumigraph, one of the neighboring original views, in this case s_2, usually sees the surface under a similar if not better angle than the destination image. Thus, the number of gaps from this cause typically is also smaller than in a more general setting.

Hierarchical hole filling. To fill any remaining gaps, one can employ a hierarchical approach. Instead of generating only one image at the final resolution, we generate a pyramid of images, where each level has half the resolution of the previous one. This pyramid resembles a mipmap [142] of the destination image. Warping to such a mipmap is inexpensive: let (u_i', v_i') be the warped pixel position according to Equation 4.3 for some point. Then the pixel positions

for the lower resolution images are $(u_i'/2, v_i'/2)$, $(u_i'/4, v_i'/4)$, and so forth.

After performing the compositing step for multiple levels of this pyramid, we can use the color information from the coarser levels to fill missing information in the finest level, which will be used for display. If we find a "hole" in the finest resolution (e.g. a pixel that has not been assigned a single input image sample), we can simply look if the next-coarser pyramid level contains a color for this pixel. If so, we use that color. If not, we go up another level in the pyramid and iterate the process. Algorithm 4.1 on page 69 explains this procedure in the form of pseudo code.

Boundary treatment. Using this simple approach works well for filling holes on the inside of an object. If an object is surrounded by background, this means that there are not necessarily any samples around the boundary of the object. If we simply fill up the pixels next to the object boundary with pixels from coarser pyramid levels, we effectively reduce the resolution of the object's silhouette by half.

However, the solution to this problem is rather simple: in order to avoid this "extrapolation" of the object boundary, one must apply a morphological operator on the coarser-level images in order to remove the object boundaries on these levels. More precisely, using the so-called "erosion" operator, which is well-known from image processing [36], removes all boundaries of all objects in the coarser images. After this operator has been applied, the holes near the boundary will not be filled up from coarser-level information, whereas holes in the inner part of an object will be filled as before.

4.6 The Complete Algorithm

Algorithm 4.1 on page 69 illustrates the complete Lumigraph refinement process (one level of refinement) including hole filling. After determining the set of novel views to be inserted (Section 4.2), the main loop iterates over all novel viewpoints to be inserted. For each of these novel views, the set of relevant source views is determined (Section 4.3). Each one of these source views is then warped into the resolution pyramid of the novel view (Section 4.4), and the reprojected samples are blended with the existing ones (Section 4.5) on each level of the pyramid. After all pixels for one novel view are warped and composited on all pyramid levels, the morphological filtering is done on the coarse pyramid levels, and hole filling closes the gaps in the high-resolution image with data from the low-resolution levels of the pyramid (Section 4.5.3).

4.7 Results

For the purpose of evaluating the quality and performance of the warping and refining algorithms, we have created Lumigraphs from two synthetic scenes depicted in Figure 4.5. The first scene consists of a little airplane without textures. The second scene contains an elephant

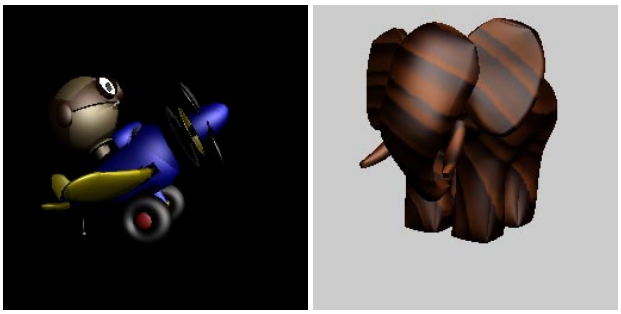

Figure 4.5: The two scenes used for comparison purposes. A simple airplane, and a textured elephant, both with a rather complex visibility.

rendered using a procedural wood shader. Both are ray-traced RenderMan objects, and the base resolution Lumigraph contains 4×4 images of size 256×256.

The visibility conditions in both scenes are rather complex. For example, if the airplane is viewed from the side, its wing and rotor blades occlude parts of the plane's body. Similarly, the elephant's trunk and tusks occlude part of the body, and its legs occlude each other.

The first criterion for the quality of our warping algorithm is the number of holes that need to be filled by the pyramid warping approach. This is best depicted by showing the depth image of a warped view where only the highest resolution layer has been used (Figure 4.6). The holes show up as small dots in the indicated regions. It can be seen that there are only very few holes, and since they only consist of isolated pixels, they are easily filled using the second level of the image pyramids. For example in Figure 4.6, only seven pixels had to be filled. Due to the increased geometric complexity, the number of holes for the elephant lies between five and twenty. Note that, although parts of the elephant's back are covered by its ears in two of the source images, the other two source images fill in this missing information.

In order to further evaluate the quality of the warped images, we compare them to both ray-traced images and images generated using quadri-linear interpolation (Figure 4.7). Due to the large parallax between the source images, the interpolated images exhibit strong blurring and ghosting effects. This is especially visible for those parts of the object that protrude far from the position of the imaginary image plane, which usually cuts through the center of the object. In our examples, the most problematic part of the scene is clearly the airplane's wing.

Although the protruding wings are rather difficult to sample from side views, the results show that our warping/compositing approach is adequate for light fields. As can be seen in Figure 4.7, the quality of the images that are generated and inserted into the Lumigraph are practically indistinguishable from the ray-traced original images.

Warping proceeds at more than 8 images/sec. on an R10000 SGI O2, while lightfield rendering using graphics hardware on the same machine is more than 3 times as fast (≈ 30 frames/sec.). Ray-tracing takes several minutes for each of the two scenes. These numbers

show that our warping as a stand-alone method is almost fast enough for interactive viewing, obtaining an image quality that is very much comparable to ray-tracing. On the other hand, the location of the viewpoints is, of course, limited to the eyepoint plane.

The final criterion for the evaluation of our approach is the improvement in image quality when quadri-linear interpolation is applied to a refined Lumigraph. A comparison is depicted in Figure 4.8. It shows images reconstructed from the 4×4 Lumigraph of the elephant, and from a refined version with 13×13 images, as well as a sequence of resolutions for the airplane (2×2, 4×4, 7×7, and 13×13 images). As can be seen, the ghosting and blurring artifacts are dramatically reduced with increasing Lumigraph resolution. Independent of Lumigraph size, these images can be rendered at full screen resolution at constant frame rates of roughly 20 frames/second on an SGI O2.

4.8 Discussion

The contribution of this chapter is two-fold. First, we have introduced an alternative representation of Lumigraphs as a collection of images with per-pixel depth information. As we will see later on, this contribution lays the foundation for the following chapters, and the representation as well as the ideas that have been presented in this chapter will give rise to a whole number of exciting new possibilities in the context of light field acquisition and rendering.

Secondly, we proposed and developed a technique for refining a Lumigraph through warping, thereby avoiding adaptive depth correction. We have derived efficient warping formulae for the case of two-plane parameterized light fields, we showed how to properly do the occlusion testing and further compositing of the warped samples, and we discussed different refinement strategies. The net result is a technique that, given a Lumigraph with a sparse set of reference images, robustly and quickly generates in-between views. The refined Lumigraph can be rendered at very high frame rates without performing any costly adaptive depth correction.

The results clearly show how that light field rendering quality increases with every refinement step. However, it is obvious that for some scenes (as for example the aeroplane) it does not seem reasonable to refine the Lumigraph until the apparent size of all artifacts drops below the size of a pixel, since this would require an immense amount of redundant image data. In the end, the user has to decide when to apply this refinement technique or an alternative approach.

In the next chapter, we will extend the presented approach and make it suitable for direct reconstruction of arbitrary views at interactive frame rates.

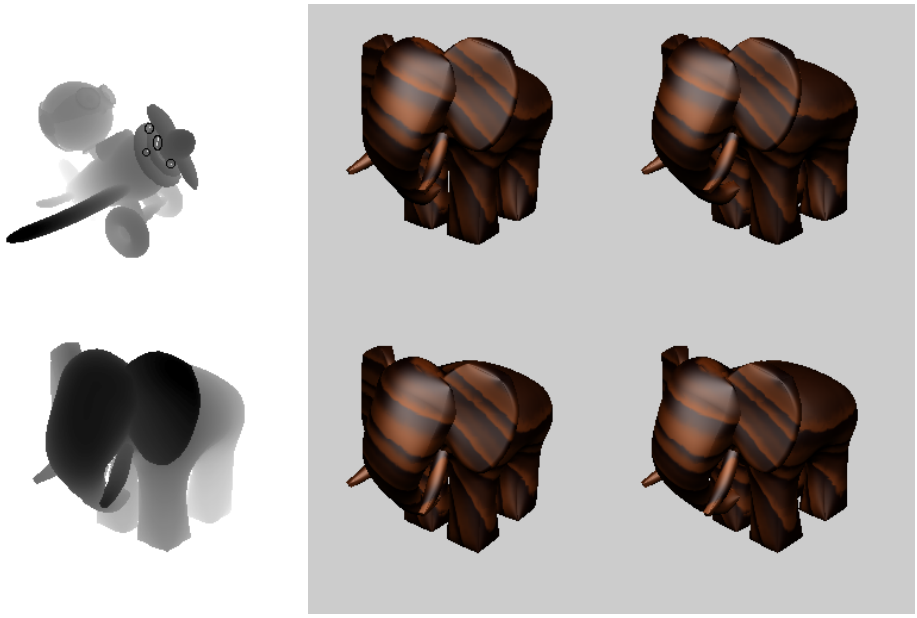

Figure 4.6: Left: gray-coded depth values of a warped and composited view before applying hole filling. Holes are marked with black circles. Center/right: source images used for warping the elephant.

Figure 4.7: Left: images generated through quadri-linear interpolation from four source images. Center: ray-traced images. Right: images generated by our warping algorithm.

Figure 4.8: Left: quadri-linear reconstruction from a 4×4 and a 13×13 Lumigraph. Center and right: quadri-linear reconstruction from 2×2, 4×4, 7×7, and 13×13 Lumigraphs. The 7×7 and 13×13 Lumigraphs have been generated from the original 4×4 light fields using our refinement algorithm.

```
/* select novel viewpoints */
S ← set of viewpoints to be inserted
foreach new viewpoint b in S
  /* select source images */
  Ĩ_b ← set of relevant source images for view b
  init image pyramid for I_b
  foreach source image a in Ĩ_b
    /* multi-level warping and compositing */
    foreach pixel i in I_a
      compute warped pixel position u'_i in view b
      compute new pixel index i' in I_b
      foreach pyramid level l from fine to coarse
        compute pixel index i'_l on this level
        depth test / blend with other samples for i'_l
  /* pyramid prefiltering */
  foreach level l except finest level in pyramid
    apply erosion operator to image on this level
  /* hole filling */
  foreach pixel i in finest level of pyramid
    if i is a hole then
      foreach level l except finest, from fine to coarse
        compute pixel index i'_l on this level
        if pixel i'_l on this level is no hole then
          set color of pixel i to that of i'_l
          break foreach loop
```

Algorithm 4.1: Pseudo code for Lumigraph refinement, putting together the different steps described in Sections 4.2 – 4.5.

CHAPTER 5

Warping Lumigraphs into Arbitrary Views

> It can hardly be a coincidence that no language on Earth has ever produced the expression "as pretty as an airport".
>
> Douglas Adams, *The Long Dark Tea-Time of the Soul*

In this chapter we reconsider the question of how to warp Lumigraph image data into a novel view, but this time the eyepoint position is *not* restricted to the Lumigraph's eyepoint plane as in the previous chapter. Rather, we generate the *output view* for an arbitrary viewer by warping and compositing regions of the input views. In contrast to the previous chapter, this algorithm is no preprocessing step, but is executed *interactively* while the viewer is navigating.

Besides the actual reprojection, the most important point of this approach is a *partitioning* of the input views, such that it is not necessary to warp *all* image data in order to reconstruct the output view. This is made possible by the underlying light field structure that defines which parts of the scene should be reconstructed from which input views.

Furthermore, the proposed algorithm exploits modern graphics hardware capabilities by generating a single texture per view (matching the resolution of the input views) and reprojecting that texture into the final view using a standard `OpenGL` texture mapping step. This keeps our algorithm independent of the output image size.

5.1 Motivation and Outline of the Method

As we have seen in the previous chapter, warping can be used as a method for reconstructing novel views from Lumigraph data. So far, we have looked only at the restricted case of reconstructing views that fall on the Lumigraph's eyepoint plane. In this chapter, we will extend these ideas and use warping as an alternative means for reconstructing arbitrary views from a Lumigraph.

Although the traditional rendering method based on texture mapping is very efficient, it becomes tricky to implement and hard to control when using adaptive depth correction. Adaptive depth correction slows down rendering, and the rendering time is dominated by the varying number of triangles used for reconstruction (cf. Section 3.5). Usually, several user-defined parameters have to be fine-tuned in order to establish the right balance between the number of artifacts and the speed of the rendering algorithm.

With the warping-based rendering method proposed here, the depth correction is performed for every input pixel, so no parameter tuning is needed and we can expect the maximum rendering quality. Moreover, the speed of our proposed algorithm scales well with the number of input images. It only requires a small amount of texture memory for rendering. As we will see, the performance of our algorithm is comparable to that of classical pure light field rendering methods.

The input for interactive Lumigraph warping is again a two-plane parameterized Lumigraph with per-pixel depth values. The user navigates interactively around the Lumigraph, and the algorithm directly reconstructs the user's view from any position and with any camera parameters the user might choose. In order to do this efficiently, interactive Lumigraph warping does the following:

- For the given output view, determine which parts of the output image should be reconstructed from which part of the input data. We want to use as few input images as possible, so that the algorithm performs well even for very large Lumigraphs. This part of the method is called *partitioning* and will be discussed in Section 5.3.

- All pixels in the determined source regions of the input images are reprojected into the novel view. Like in the previous section, the reprojection takes place within the original Lumigraph image plane. The reprojection step (a.k.a. warping) is discussed in Section 5.2.

- Compositing of the reprojected pixels into a single output texture. As before, compositing consists of a depth test and a blending operation. The result is an output texture of the same resolution as the input images. Compositing is explained in Section 5.4.

- Remapping of the output texture into the final view of the user. This step is a standard 3D graphics operation that is easily performed by drawing the image plane rectangle and applying the generated texture using OpenGL [99].

After explaining each of these steps in more detail, we sketch the pseudo code of the overall algorithm in Section 5.5, and then show results obtained with our experimental implementation in Section 5.6. At the end of the chapter, in Section 5.7, we discuss these results and point out advantages, drawbacks, and possible future improvements of the method.

5.2 Lumigraph Warping for Arbitrary Eyepoints

As defined in the previous chapter, a Lumigraph contains a set $\{I_0, I_1, \ldots, I_{N-1}\}$ of N images and corresponding depth maps. Each image I_k is projected onto the same domain on image plane I, and is associated with an eyepoint $(s,t)_k$ which is the center of projection used to generate the image. Again, all eyepoints are organized in a mesh on the eyepoint plane E. Warping a pixel i at $(u,v)_i$ from a reference image I_k into the image plane texture T means determining the corresponding pixel location $(u',v')_i$ in T. This correspondence is established by the depth value z_i of the pixel.

Figure 5.1 depicts the basic situation that is a 2D sketch of the two-plane arrangement similar to that in Figure 3.3 on page 38. The pixel u_i observed from a reference eyepoint $(s,t)_k$ corresponds to the color of a scene object at P_i with X coordinate X_i. The output camera C observes the same point at pixel position u_i' in the texture image T on the image plane I. D defines the distance between the two planes of the slab, and z_i is the depth value of pixel i, measured from the image plane towards the eyepoint plane.

From the triangles in Figure 5.1 we can derive these two basic relations:

$$\frac{u_i' - X_i}{z_i} = \frac{u_i' - s_C}{z_C}, \tag{5.1}$$

$$\frac{s_k - X_i}{D - z_i} = \frac{s_k - u_i}{D}. \tag{5.2}$$

Solving the two equations for X_i, substituting one into the other, and solving again for u_i' gives the following:

$$u_i' = \frac{u_i z_C D + z_i z_C s_k - z_i z_C u_i - z_i D s_C}{D(z_C - z_i)}. \tag{5.3}$$

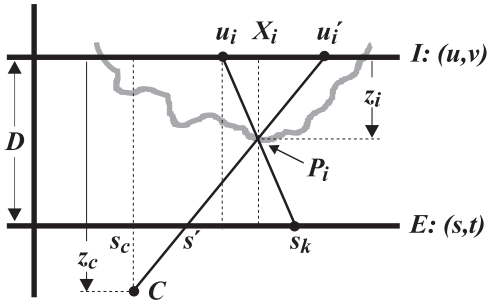

Figure 5.1: Schematic situation on the (s,t) and (u,v) planes when warping a pixel u_i from a reference view $(s,t)_k$ to its new location u_i' for an arbitrary off-plane viewpoint C. P_i denotes the scene point corresponding to the pixel. The depth z_i of a pixel is measured as illustrated.

Equation 5.3 computes the view-corrected coordinate u_i' of a pixel i, given its coordinate u_i in the reference image, its depth z_i, the reference eyepoint s_k, and the depth z_C and plane coordinate s_C of the output camera C. Since some of these variables are constant through all pixels for a pair $(C,(s,t)_k)$, it makes sense to isolate some terms:

$$\begin{aligned} u_i' &= \frac{1}{D(z_C - z_i)}\left[A_c u_i - z_i(z_C u_i + B_k)\right], \\ A_c &:= z_C D, \\ B_k &:= z_C s_k - D s_C. \end{aligned} \tag{5.4}$$

For each output image, A_c is constant, and B_k varies with the reference eyepoint $(s,t)_k$. The same equations also apply to the second pixel coordinate v_i' when substituting v for u and t for s.

Just to make sure that these results are consistent with those in the previous chapter, we assume the output camera C to be identical with some eyepoint s' on the eyepoint plane E. In this case we can substitute $s_C = s'$ and $z_C = D$ in the above equations, leading to

$$\begin{aligned} u_i' &= \frac{1}{D(D - z_i)}\left[D^2 u_i + z_i D s_k - z_i D u_i - z_i D s'\right] \\ &= \frac{u_i D(D - z_i) + z_i D(s_k - s')}{D(D - z_i)} \\ &= u_i - \frac{z_i}{D - z_i}(s' - s_k). \end{aligned} \tag{5.5}$$

As expected, Equation 5.4 reduces to Equation 4.3 on page 59 for this special case.

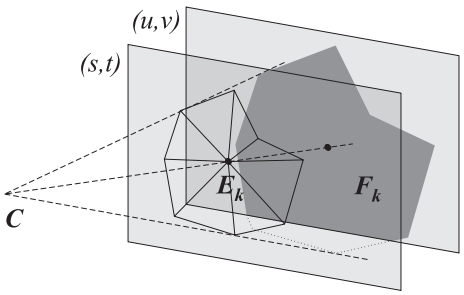

Figure 5.2: Texture partitioning: The reference image associated with eyepoint E_k (with coordinates $(s,t)_k$) has to contribute to the region F_k, which results in projecting the triangle fan around E_k from the current viewpoint C and intersecting it with the image plane domain.

5.3 Inverse Warping through Partitioning

Now that we know how to warp single pixels, the remaining task is to determine which pixels from which reference images we need to warp in order to obtain an output image of high quality at minimal cost.

In order to support the general case of a non-uniform viewpoint plane, we use a Delaunay triangulation of the eyepoints $\{(s,t)_k\}$ in the (s,t) camera plane (cf. Sections 3.2, 3.4, 4.2.2). This means that the radiance values observed through each eyepoint triangle $((s,t)_a, (s,t)_b, (s,t)_c)$ will be interpolated from pixels in the three associated images I_a, I_b, I_c.

Destination region partitioning. In other words, each reference image I_k should contribute to all those triangles containing $(s,t)_k$ as a vertex. One such triangle *fan* F_k is depicted in Figure 5.2. After projecting the fan onto the image plane and clipping it against the valid image domain, we obtain the *destination region*, which is the texture region that I_k has to contribute to. As one can see, determining these destination regions results in a three-time *partitioning* of the image plane domain, where each point in the domain belongs to exactly three destination regions.

Source region computation. Since the pixels will be forward-projected, we also need to know the *source* region R_k in image I_k from which all pixels have to be projected so that the whole destination region will be covered. The source and destination regions are not identical in general because the pixels "flow" through the image. In order to compute a conservative bound, the region has to be extended by the maximum flow that can occur when warping *into* the destination region (which does not need to be the same as when warping *from* the region).

Looking again at Figure 5.1, we can re-formulate the warping equations in terms of the "virtual" viewpoint s' on the eyepoint plane (cf. Figure 5.1 on page 74). This corresponds to

Equation 4.3 presented in Section 4.4 (page 59) and leads to:

$$u_i' - u_i \quad = \quad \frac{z_i}{z_i - D}(s' - s_k). \tag{5.6}$$

We see that the pixel flow $(u_i' - u_i)$ is simply the product of the *virtual camera movement* or *baseline* $\Delta s = (s' - s_k)$ on the viewpoint plane, and of the source pixel's *two-plane disparity* $\bar{z}_i := z_i/(z_i - D)$ that denotes the flow of pixel i for a unit camera movement on the eyepoint plane.

The easiest way to find a conservative bound for the absolute value of the pixel flow is to separately bound the two factors $\bar{z}_i$ and Δs. The first factor can be bound by simply computing the maximal absolute disparity value $\bar{z}_{\max} := \max\{|\bar{z}_i|\}$ separately for each image, and using the maximum of these values among the images which contribute to the fan.

The second factor can be bound by finding the maximal distance $|s' - s|$, where $s = s_k$ is the center of the fan, and s' is any point in the fan: So we can find the bound by computing the length of the maximal "radial" edge within the fan (giving us some kind of maximal fan radius).

The maximal flow $f_k^{\max}$ is now obtained as the product $\bar{z}_{\max} \cdot d_k^F$:

$$f_k^{\max} \quad = \quad \max\{|\bar{z}_i|\} \cdot \max\{||(s,t)_k - (s,t)_j|| \; ; \; (s,t)_j \text{ in fan}\}. \tag{5.7}$$

One can tighten the bound by partitioning the reference images into smaller blocks B (e.g. 16x16 pixels each) and storing the maximal disparity $\bar{z}_B$ for each block in each image. In order to determine $f_k^{\max}$, one must check for each block B whether pixels can flow into the target region F_k. This is the case if

$$\min\{||X - (s,t)_k|| \; ; X \in B\} - d_k^F \leq \bar{z}_B \cdot d_k^F, \tag{5.8}$$

meaning that, starting from the closest point in the block, the maximal flow from within the block is large enough to reach the boundary of the target region.

Region growing. In order to define a conservative warping source region, we extend ("grow") the destination region around $(s,t)_k$ by this distance $f_k^{\max}$ in each direction. It is clear that this boundary is far from optimal, so that more pixels will be warped than are actually needed. On the other hand, it guarantees that all the required pixels will be reprojected. Instead of growing the region orthogonal to each edge, we can simply grow the bounding box of the region, or any other bounding shape.

As mentioned before, the destination regions represent a true partition of the image domain. Ideally, if we knew for every pixel in the destination region which original pixel(s) will be projected onto it, the number of pixels to be reprojected would not depend at all on the number of eyepoints/images. This would mean that the computational effort to generate an output image would remain constant, regardless of the number of images in the Lumigraph.

The source regions obtained by region growing results in an overhead that depends on the pixel flow bounds. This means that we will warp more pixels than are actually needed in the ideal case. Fortunately, the pixel flow boundary (and thus, the size of the overhead) is tied to the triangle fan size, which decreases with an increasing number of eyepoints in the Lumigraph. So we can expect the region growing overhead to remain roughly constant. We will see in Section 5.6 that this is actually the case.

5.4 Compositing

In addition to finding the relevant samples and reprojecting them according to the current view, we have to interpolate each texel's final color from all the visible samples that have been projected into the texel's domain. As we have already described in Section 4.5, this involves first a depth test and then a color interpolation.

While in the previous chapter compositing was merely a preprocessing step in the pipeline for generating an image (remember that it was just used to add some in-between images into the light field), here it will directly define the final pixel color, except for one last texture resampling step done during the OpenGL display. This means that the blending function (defining the sample weights) must be smooth across all eyepoint triangle edges. If this requirement is not met, the edges will become visible due to blending artifacts.

In order to guarantee a smooth reconstruction of colors across triangle boundaries, the interpolation weight for a sample from reference image I_k should be 1 at the triangle vertex corresponding to the eyepoint $(s,t)_k$, and 0 on the boundary of fan F_k. If we choose linear weights that meet these conditions, this corresponds to barycentric triangle coordinates with respect to $(s,t)_k$ [140].

Instead of implementing the compositing operations in terms of the blend set B^j as described in Section 4.5.2, we use a simple algorithm that does not require to store multiple samples per output pixel. For every output pixel j we store a current color L_j and depth z_j, as well as the sum W_j of the interpolation weights so far. Now another sample (L'_j, z'_j) is projected into the same location. If the new sample is definitely occluded by the existing sample (e.g. $z'_j < z_j - \epsilon_z$), it is discarded[1].

In any other case, we compute the blending weight w'_j of the new sample. If the new sample occludes the existing sample ($z'_j > z_j + \epsilon_z$), the old color and depth are discarded and the pixel is set to the new sample

$$\begin{aligned} L_j &\leftarrow L'_j, \\ z_j &\leftarrow z'_j, \\ W_j &\leftarrow w'_j. \end{aligned}$$

[1] Remember that z coordinates are increasing towards the viewer.

If none of these two conditions apply, we interpolate using the new sample's weight w'_j and the sum of weights W_j of all other samples that have contributed to the color L_j so far:

$$\begin{aligned} L_j &\leftarrow \frac{1}{W_j + w'_j}\,(W_j L_j + w'_j L'_j), \\ W_j &\leftarrow W_j + w'_j \end{aligned}$$

Hole filling can be performed in exactly the same way as described in Section 4.5.3 using an image pyramid. Note that all the special conditions discussed in that section still apply to the new situation here, since the viewing rays can still be interpolated from the "surrounding" views, and the sampling rate does not increase if the viewer does not move behind the eyepoint plane.

5.5 Performance and Complexity of the Algorithm

The rendering algorithm (see Algorithm 5.1) combines the parts that have been explained in the previous sections. It iterates through all eyepoints $(s,t)_k$ of the current slab, computes the corresponding fan F_k, and extends it to obtain the warping source region R_k (Section 5.3). All pixels in this region are then warped into the image plane texture (Section 5.2), and blended with the other valid samples (Section 5.4). After doing this for all eyepoints, the resulting textured polygon is drawn in the slab's image plane using graphics hardware.

The computational complexity of the algorithm is dominated by the inner loop that iterates over all pixels in the source region of every image. So the sum of all pixels in all source regions determines the number of pixels that must be visited. As you can see in the pseudo code in Algorithm 5.1, visited background pixels (with $z_i = -\infty$) are skipped immediately, so that the foreground pixels in the source regions dominate the actual computational complexity.

As described in Section 5.3, the union of all *destination* regions is three times the number of pixels in one source image (given that all source images and the output texture have the same resolution), or less due to clipping. This means that without region growing, the worst-case complexity only depends on image resolution. The region growing algorithm extends the regions such that the union of all *source* regions is bigger than that of the destination regions, and its size depends on the scene's depth structure. As we will see in the following section, the region growing overhead is practically independent of the number of eyepoints in the Lumigraph, and is an acceptably small fraction of the overall computational cost.

```
/* interactive Lumigraph warping */
C ← output camera for this frame
for all front-facing slabs S do
  init buffers {(L_j, z_j)}
  for all eyepoints (s,t)_k ∈ S do
    /* partitioning */
    R_k ← triangle fan around (s,t)_k
    f_k^max ← max. pixel flow into R_k
    grow R_k by f_k^max
    /* warping + blending */
    for all pixels u_i in R_k
      if z_i ≠ −∞ then
        u_i' ← warp(u_i, z_i, (s,t)_k, C)
        j ← pixel_index(u_i')
        compositing of (L_j, z_j) with (L_i, z_i)
  /* texture display */
  specify {L_j} as texture
  draw textured image plane polygon
end
```

Algorithm 5.1: Pseudo code of the basic rendering algorithm, putting together the parts explained in Sections 5.2 – 5.4.

5.6 Results

In order to validate our approach, we show some results produced with our experimental implementation. First, we analyze the image fidelity and performance for different numbers of reference images and different image plane resolutions. Next, we discuss the issue of quantizing the depth values used for warping. Last, we compare the new algorithm with implementations of light field interpolation and depth-corrected Lumigraph rendering techniques.

5.6.1 Rendering Quality

Using the adaptive acquisition approach proposed in [112] (see also next chapter), we have generated images of a ray-traced elephant for building Lumigraphs of the same scene with different numbers of eyepoints between 4 and 164. We have rendered the same views of the elephant for an increasing number of viewpoints (4, 25, 50, 100, 164). Figure 5.3 shows a view computed with our new method from a 2x2-image Lumigraph at 256x256 image plane pixels.

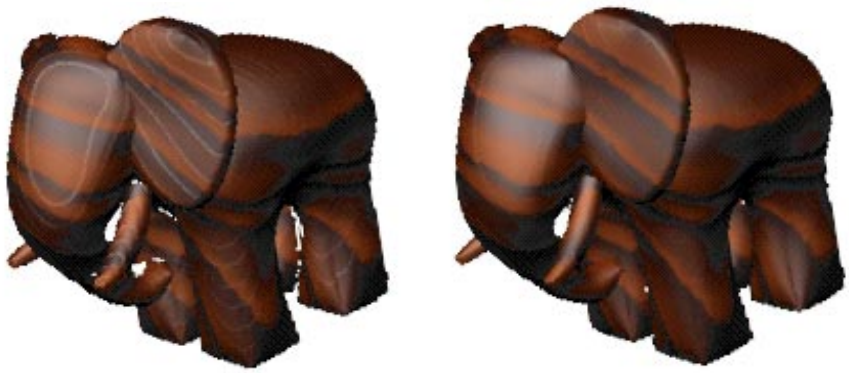

Figure 5.3: Left: view of an elephant Lumigraph generated from only 2x2 images. One can see that the elephant appears very sharp, but there are visible disocclusion and highlight artifacts near the trunk and the hind legs. Right: the same view, but generated from a 25-image Lumigraph. The disocclusion artifacts are gone, and highlight artifacts are barely visible.

Despite the small amount of input images, the resulting view appears very sharp. Due to very sparse scene sampling, disocclusion artifacts near the legs and the trunk of the elephant are clearly visible. The glossy highlights on the surface also generate some artifacts and appear as color seams.

We also show the same view generated from a Lumigraph containing 25 images. This version of the Lumigraph already provides enough information to avoid the disocclusion problems as well as most of the glossy artifacts when using our algorithm. Only a few artifacts can be detected from some viewing positions, mostly small disocclusions caused by the trunk or the legs. From most of the valid viewpoints, this 25-image Lumigraph cannot be distinguished visually from versions rendered using many more reference images.

If the viewer moves behind the eyepoint plane and gets very close to the image plane, the camera can be very close to the actual scene object, and under-sampling artifacts appear (cf. Figure 5.4). Since the gaps are always very small (mostly single pixels), they could be removed quite easily by a simple gap filling approach such as the one described in Section 4.5.3. However, by definition a light field only represents the light *leaving* its bounded 3D domain, and the viewer should actually not be allowed to move inside the two planes. So one can decide between constraining the user's navigation or filling gaps. Instead of moving closer to the image plane, the viewer can change the field-of-view angle. This results in "zooming in" on the object without generating any under-sampling gaps. The only artifact in that case is that the image will become blurry as soon as the output resolution is higher than that of the corresponding image plane domain.

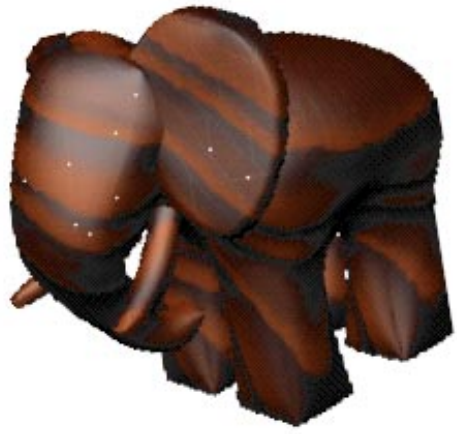

Figure 5.4: Zoomed-in view of a 25-image elephant Lumigraph, viewed from behind the eye-point plane. Gaps (white dots) with the maximal width of 1 pixel can be observed.

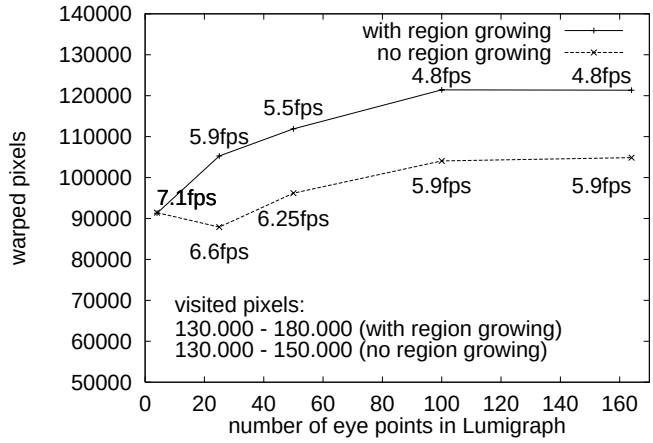

Figure 5.5: Warped pixels / frame rate plotted against the number of eyepoints in the 'elephant' Lumigraph. For the lower curve, warping source regions have not been extended at all. The upper curve was obtained using the global disparity maximum (cf. Section 5.3). Rendering times are given for a 270 MHz SGI O2 (R12K). On a 300 MHz Octane, the rate goes up by a factor of 1.5–1.9.

5.6.2 Performance Issues

The rendering performance of the algorithm depends on several factors. The resolution of the image plane texture plus the region growing overhead determine the upper bound for the number of pixels to be processed. However, since the algorithm discards background pixels immediately, it is obvious that the main computation time is spent on the actually warped foreground pixels (cf. Algorithm 5.1). As discussed in Section 5.3, the number of processed pixels should only exhibit a weak dependency on the number of reference images.

Figure 5.5 shows the number of foreground pixels as well as the frame rate for the "elephant" Lumigraph, for different numbers of reference images. The image plane resolution is 256x256 pixels, the maximum disparity is $\bar{z} = 0.3$. It can be observed that the rendering time increases sub-linearly with the number of eyepoints, and the frame rate is proportional to the number of warped pixels. This confirms our discussion about the algorithm's computational complexity (Section 5.3 as well as Section 5.5). The difference of the two plotted curves nicely visualizes the constant overhead added by region growing.

Figure 5.7 shows a different artificial scene captured in a 36-image Lumigraph with 512x512 image plane pixels. The rendering time for this Lumigraph increases as expected due to the four times higher image plane resolution, but since the number of foreground pixels is comparable to that in the elephant scene (relatively much black background in the scene), we still obtain 4.8 frames/s for 95.000 warped pixels with region growing, and 5.5 fps for 70.000 pixels without region growing on an O2 (max. disparity is 0.5). The average number of visited pixels (including background) per frame is 212.000, as opposed to 140.000 for the elephant Lumigraph.

We viewed the images at full screen resolution. As expected, the this does not affect the rendering time since the final texture mapping step is done in hardware.

5.6.3 Depth Quantization

In all examples in this paper, the depth values have been uniformly quantized in the range $[-D : D]$ before using them for warping. Choosing a quantization of 8 bit does not affect the quality or performance of the algorithm at all, and it removes the need for a blending depth threshold ϵ_z as explained in Section 4.5.1. In our experiments, we found that even with 6 bit depth resolution, the images appear quite sharp and clean. Only the most prominent edges in the scene become a bit blurry (e.g. the elephant's ear). When using less than 6 bit, the artifacts are similar to, but not as strong as, those of pure light field interpolation. Following the principles discussed by Chai et al. [14], distributing the depth planes in non-linear manner would help to exploit the depth information even better.

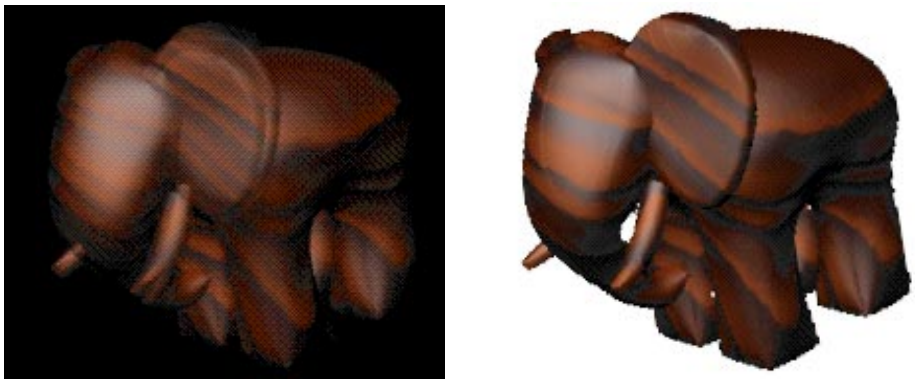

Figure 5.6: Left: view generated by light field interpolation from a 50-image Lumigraph. Despite the large number of images, there are very strong blurring and ghosting artifacts. Right: similar view, generated from 25 images using our technique.

5.6.4 Comparison with Previous Approaches

Figure 5.6 depicts the same view as Figure 5.3, but computed through interpolation from a light field without depth information [74]. Even though we used twice the number of images than for Figure 5.3, very strong blurring and ghosting artifacts appear. These are even visible for the 164-eyepoint light field.

In Figure 5.7 we compare the reconstruction of a different scene using our algorithm to a reconstruction along the lines of the original work on depth-corrected Lumigraph rendering [37]. The algorithm casts depth rays towards a coarse geometric model of the scene. If the three depth values for an eyepoint triangle differ, the triangle is subdivided adaptively, and more depth rays are used. If the algorithm does not detect any depth discontinuities using the initial depth rays, the adaptive scheme fails. This leads to blurred edges similar to those in Figure 5.7. Also, if a discontinuity is detected, quite a large number of depth rays (up to one per pixel along an edge) has to be cast in order to adapt the triangles to the edge. The computational cost of the approach depends on depth continuity in the scene as well as on the number of geometric primitives used for approximating the scene. More experiments with the original Lumigraph algorithm would be necessary in order to analyze these aspects in detail. From the tests with the implementation at hand, we learned that the algorithm becomes non-interactive as soon as more than 1000 triangles are used.

5.7 Discussion

With the work presented in this chapter we have established warping as an alternative rendering technique for Lumigraphs. Besides the actual reprojection of pixels for two-plane-

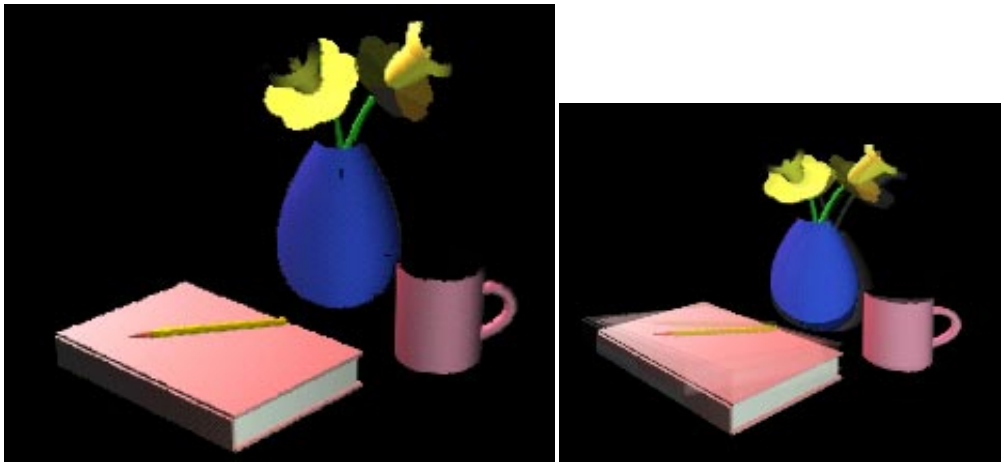

Figure 5.7: Left: Lumigraph view of a simple scene, computed by warping from 6x6 images of 512x512 pixels resolution. Right: Rendering of the same scene using the Lumigraph rendering algorithm with adaptive depth correction [37]. Since the depth rays miss some edges in the scene, there is still a large amount of blurring.

parameterized light fields, we have introduced a partitioning and region growing scheme that solves the inverse warping problem in an efficient way on a per-triangle-fan basis. Another benefit of the partitioning is that the computational cost for rendering only increases sub-linearly with the number of reference images in the Lumigraph, so warping can be used efficiently for rendering Lumigraphs of arbitrary size. We have also examined the influence of depth quantization on the rendering quality, and found that even with a rather coarse depth resolution (6 bit linear quantization) we achieve competitive results, which indicates that this technique could also be used for applications that involve noisy real-world data.

We believe that Lumigraph warping has several advantages over the traditional rendering with adaptive depth correction. First, in contrast to depth correction it runs “out of the box”, without the need for parameter tuning in order to yield a good trade-off between performance and quality. Second, adaptive depth correction fails whenever a depth discontinuity is not detected, whereas warping exploits the geometry information completely. Third, Lumigraph warping is suitable for applications that require stable frame rates (e.g. for virtual reality), whereas depth correction has the problem of highly view-dependent computational cost due to its adaptive nature.

There is still some room for improvement of the method. First of all, it would be nice to avoid under-sampling artifacts that occur if the viewer enters “forbidden terrain” and moves behind the eyepoint plane[2]. This can be done quite easily by adapting (coarsening) the reso-

[2]Please remember that the light field only represents the light distribution leaving that plane, and not the light

lution of the image plane texture if the user does so. This could even lead to a more general multi-resolution approach that uses mip-maps of the reference images and adapts the texture resolution to the current viewpoint and screen resolution. Second, a further tightening of the region growing bounds based on more localized depth information should be examined in order to achieve further speedup of the method.

distribution of the points between the two planes.

CHAPTER 6

Adaptive Lumigraph Acquisition

> And similarly solely by the faculty of judgment which rests in my mind, I comprehend that which I believed I saw with my eyes.
>
> René Descartes, *Meditations of First Philosophy*

In this chapter we are concerned with acquiring Lumigraphs from synthetic scenes. Since a Lumigraph consists of a number of images and geometry data (e.g. per-pixel depth), acquisition can be done by rendering a number of views of the synthetic scene, and using these rendered images as the Lumigraph input data.

However, high-quality rendering of synthetic scenes (e.g. including global illumination computations) consumes a large amount of time, and so the goal of this work is to find a method that only acquires those images that are actually needed to faithfully reconstruct the appearance of the scene.

This is solved by an adaptive acquisition algorithm starting with an initially under-sampled Lumigraph, and predicting the reconstruction quality from this Lumigraph by estimating warp-

ing and blending errors for eyepoints not yet covered. We use a modified warping technique similar to that described in the previous chapters to simulate the reconstruction process and thereby predict which additional views increase reconstruction quality the most.

6.1 Motivation and Outline of the Method

Automatic acquisition of light fields and Lumigraphs is an important area of research both for real-world and for synthetic light fields (cf. Section 3.8). One of the most important points is *view planning*, meaning the selection of the best set of viewpoints in order to faithfully represent the scene. Most of the work that has been done so far has been concentrating on geometric aspects, making sure that all faces of all objects are sampled properly from at least one of the views.

However, this is clearly not enough in our context, since one important goal of light field rendering is to represent photometrically complex objects. It is clear that an automatic acquisition algorithm must take into account aspects such as visual quality of the reconstructed images and view-dependent changes in surface appearance due to non-diffuse materials.

Our solution to the problem is to use a kind of *a priori* error measure that detects if there are sources of error when reconstructing the scene from the current set of views. For a certain novel view, we use a modified warping algorithm in order to predict how strong the contributions from the different source images differ in each reconstructed pixel, and we also detect "holes" in the reconstruction, indicating that the scene geometry has not been sampled adequately.

Thereby we can predict most of the potential reconstruction errors and acquire those views that fill in the missing data to compensate for these errors. By simply indicating the desired number of images in the Lumigraph (or the tolerated maximum error), our method can construct the, in our sense, optimal Lumigraph from arbitrary synthetic scenes without further knowledge about lighting or material.

The algorithm proceeds as follows. We start with a Lumigraph that contains only a few initial views of the scene and defines the light field geometry (image plane / eyepoint plane), and construct an eyepoint plane triangulation. Then we select a number of candidate points (midpoints of all edges). For each candidate eyepoint, we compute the reconstruction error by using a modified warping algorithm (Section 6.2). This error measure estimates the potential geometric errors (holes or under-sampled areas), as well as photometric errors (under-sampling of view-dependent appearance effects). We assume that the *acquisition benefit* of an eyepoint is proportional to the reconstruction error, since the reconstruction quality of the Lumigraph will not increase much if we acquire a view that can also be reconstructed well from the already acquired image data (small error).

Since the eyepoint with the largest reconstruction error is supposed to yield the biggest acquisition benefit, we acquire the image data for that view. After the actual acquisition step, we update the eyepoint triangulation and recompute the affected error estimates in the neigh-

borhood of the new eyepoint. We iterate the process until the desired number of images or a user-defined error threshold is reached.

In the following section, we first explain in detail the heart of our method, which is the estimation of the reconstruction error. Then in Section 6.3 we discuss the most important aspects of the adaptive acquisition process, such as the selection of candidate points, the update of triangulation and error estimates, and the convergence behavior of the method. After sketching the overall algorithm in Section 6.4, we show results obtained with our experimental implementation in Section 6.5, and conclude in Section 6.6 with a discussion of these results and of potential future improvements.

6.2 Estimating the Reconstruction Error

In order to estimate the error or potential benefit associated with any candidate viewpoint, we warp and composite the new image from the nearest reference views, as described in Chapters 4 and 5. As before, a pixel i from every selected reference frame is warped to its new location in the destination image using Equation 4.3 (page 59). For every destination pixel j with coordinates $(u,v)_j$, we obtain the set $C^j = \left\{(L_k^j, z_k^j)\right\}$ containing the color and depth values of all samples from the relevant source images that map to $(u,v)_j$ (cf. Section 4.5).

We again determine the front-most sample $k_{\max}$ with the maximal Z value $z^j_{\max}$ from C^j. We construct the *blend set* B^j which is the subset of C^j containing all samples within an ϵ_z-interval around $z^j_{\max}$. The final color L^j of pixel j is determined by blending from all pixels in B^j using the weighted sum

$$L^j \quad = \quad \frac{1}{\sum w_k} \sum_{k=0}^{|B^j|-1} w_k L_k^j, \tag{6.1}$$

with w_k being the constant or barycentric weight for sample k, as described before in Sections 4.5.2 and 5.4. Please note that hierarchical hole filling is not part of the considerations in this chapter, since holes are missing information, and we are concerned with avoiding holes rather than hiding them lateron.

6.2.1 Sources of Reconstruction Error

With the above-mentioned data for every destination pixel at hand (color and depth of all non-occluded samples), we can look at possible sources of reconstruction error, and define an error measure that takes all these effects into account.

Color deviation and photometric under-sampling. In order to estimate the *blending error* $\mathcal{E}_j^{(B)}$ associated with destination pixel j, we compute the weighted L_2 error which accounts

for the deviations from the final pixel color:

$$\mathcal{E}_j^{(B)} = \frac{1}{\sum w_k} \sqrt{\sum_{k=0}^{|B^j|-1} \Big[w_k \| L_k^j - L^j \| \Big]^2}. \tag{6.2}$$

Blending errors can occur due to different reasons: they account for view-dependent appearance changes of the material (e.g. highlights moving with the viewing angle), or they can indicate that the texture of the object has higher spatial frequencies than represented in the final image. This latter case has to be treated separately, since the error measure would be useless and the algorithm would probably not converge. In principle, proper low-pass filtering is an issue that has to be solved already by the renderer used for acquiring the input images.

Disocclusions and geometric under-sampling. Other sources of error in the reconstruction of a pixel j in the destination image are *holes* and *single maps*. In the case of holes, no source pixel maps to the destination pixel ($|B^j| = 0$), and in the case of single maps only one source pixel is warped onto the destination location ($|B^j| = 1$). This can both occur due to a strong change in viewing angle (under-sampling), or due to disocclusion of scene regions that are not visible from the original views.

In both situations, we do not have the necessary information (e.g., at least two samples for the same pixel) to derive a meaningful color deviation estimate. Since we must assume that an arbitrary reconstruction error could result from such a pixel, we let holes and single maps contribute some constant error to the overall error estimate (see Equations 6.4 and 6.5 below).

Additional background treatment. If Lumigraphs are used for generating views all around an object, it is reasonable to assume that the reference views contain the complete object. In other words, no part of the object silhouette lies outside any reference view. This assumption can be exploited to tighten the error bound and eliminate an unnecessary part of the still conservative estimate. To this end, we back-project holes (see below) in the destination image and see if their back-projected location is outside of any reference view. If this is the case, the pixel cannot be part of the object, and we can safely discard it.

In order to do so, we just need to reversely apply the "background" case in Equation 4.4 (page 60) to the hole pixel in the destination image. A pixel j with destination coordinates $(u, v)_{j'}$ and a depth value of $z_{j'}$ corresponds to a pixel outside the domain $dom(I)$ of a reference image I if

$$z_{j'} = -\infty \quad \wedge \quad \left[\begin{pmatrix} u_{j'} \\ v_{j'} \end{pmatrix} - \begin{pmatrix} \Delta s(I) \\ \Delta t(I) \end{pmatrix} \right] \notin dom(I), \tag{6.3}$$

where $(\Delta s, \Delta t)$ is the camera shift (baseline) on the eyepoint plane. Under the above-mentioned assumption of completely contained object silhouettes in every input view, we define the predicate $out(j)$ to be true if Equation 6.3 holds for any of the reference images used for warping. For other light field types, e.g. outward-looking Lumigraphs, it is typically false.

Now we can define the contribution of a hole pixel and of a single map pixel to the overall error. Holes induce a non-zero error contribution if they do not back-project to the outside, and single mapped pixels contribute if they do not belong to the background. Using this, we define the *hole error* and the *single map error* for a pixel j as follows:

$$\mathcal{E}_j^{(H)} = \begin{cases} w_H & |B^j| = 0 \ \wedge \ \neg out(j) \\ 0 & else \end{cases}, \tag{6.4}$$

$$\mathcal{E}_j^{(S)} = \begin{cases} w_S & |B^j| = 1 \ \wedge \ z_j > -\infty \\ 0 & else \end{cases}. \tag{6.5}$$

It is reasonable to choose the weights w_H and w_S from the same range as the norm of the color values, because a single pixel should not contribute more to the error than the maximal possible color deviation. Section 6.5 will give some hints about choosing appropriate error weights.

Undetectable effects. Although the cases discussed so far cover the most important effects, there are still situations where reconstruction error can in principle not be predicted. If the set of input images represents a directionally very sparse sampling of a certain surface, and if the surface material has very sharp (high-frequency) photometric features such as a very small highlight, this feature might be missed completely. If for example this highlight is not contained in any input image, the algorithm has no way at all of predicting it.

6.2.2 Overall Error Measurement

Now that we can determine the total per-pixel error by taking into account all three kinds of error (blending color difference, holes, and single maps), we can sum up this error using the L_2 norm in order to obtain some scalar error estimate for the whole view:

$$\mathcal{E} = \frac{1}{N} \sqrt{\sum_{j=0}^{N-1} \left[\mathcal{E}_j^{(B)} + \mathcal{E}_j^{(H)} + \mathcal{E}_j^{(S)} \right]^2}, \tag{6.6}$$

where N is the number of pixels in the destination image. The resulting total error $\mathcal{E}$ lies in the same interval as the per-pixel error, with smaller values indicating better reconstruction quality.

Figure 6.7 visualizes the typical error distribution of some candidate viewpoint. It distinguishes between true holes (light red) and outside-mapping holes (darker red) as well as foreground and background single maps (light and darker green). The gray scale regions show blending errors, with darker gray referring to higher error.

6.2.3 Additional Constraints: Triangulation Quality

As already mentioned in the outline of this method (Section 6.1), the acquisition of views leads to incrementally updating the triangle mesh that contains the eyepoints on the Lumigraph's eyepoint plane.

Like in most cases when employing an adaptive triangulation, we want the mesh to obey some geometric quality conditions. For example, there should be no sharp angles which lead to degenerated triangles, and we want to prevent triangles from becoming too small compared to the rectangle on the viewing plane. Both conditions are generally unwanted since they would have the effect that a view will get interpolated from several very similar points. In order to prevent such degeneration, triangle quality is incorporated into the weight computation. An edge e will be assigned a benefit of 0 if either the opposite angle or the area of either one of the triangles sharing e are too small.

The resulting triangulation should also be well conditioned in the sense that every vertex is connected to its nearest neighbors via direct edges. The best way to achieve this is to update the triangulation incrementally after inserting a new vertex in order to assert that the triangulation is locally Delaunay (for more information about Delaunay triangulations, see any book on computational geometry, e.g. [24]). However, the update of the triangulation involves flipping of edges, which can induce computating many new edge weights as well as updating old weights. This increases the acquisition overhead (see results section).

6.3 Adaptive Acquisition Algorithm

In order to refine the set of viewpoints incrementally, we partition the viewpoint domain into a triangular mesh with the currently acquired views being assigned to the triangle vertices. For example, we start with four vertices/views representing the four corners of a rectangle in the viewpoint plane and create a mesh with two triangles. Figure 6.1 illustrates such an initial mesh, although any other mesh with an arbitrary number of eyepoints can be chosen. Note, however, that the outer vertices of the mesh define the convex hull of the eyepoint domain.

Each new vertex in the mesh implies the acquisition of a new image with the viewpoint as the center of a sheared perspective projection. Due to the cost of acquiring such an image, it seems reasonable to consider only refinement strategies which insert a single new vertex into the mesh. One such strategy is to split an edge by inserting a new vertex at the center of that edge. The vertex will be connected to the start and end vertex of the original edge, and with the third vertices in the far side of the triangles sharing that original edge. For edges at the border of the mesh, there is only one such triangle. For inner edges, there are exactly two. Figure 6.1 illustrates the topological changes when splitting an inner edge and inserting a new vertex at the edge's midpoint.

6.3.1 Edge Weights for Eye Point Candidate Selection

Each edge e of the triangulation is assigned a weight B_e corresponding to a *splitting benefit* value, which is given by the warping error (Equation 6.6) in its midpoint position. B_e indicates the benefit of inserting a new viewpoint in the center of e. In order to compute that benefit,

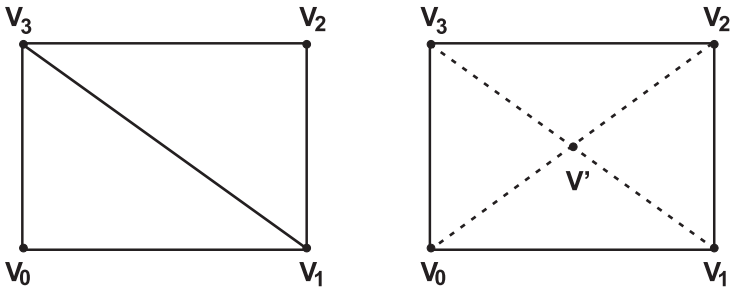

Figure 6.1: A simple initial viewpoint mesh consisting of four corner vertices $V_0 \ldots V_3$ and five edges (solid lines). V' is a candidate viewpoint for the diagonal edge (V_1, V_3). When inserting V', four new edges (dashed lines) would be created, and the new view would be warped from the images at $V_0 \ldots V_3$.

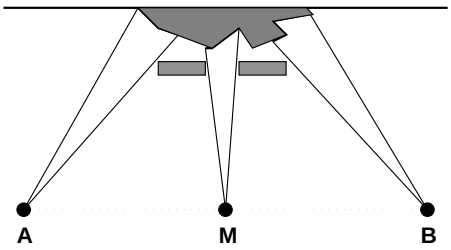

Figure 6.2: If the edge (A, B) is split, the new viewpoint M observes parts of the scene which have been occluded when viewed from A and B. This can result in higher error values for the new half edges (A, M) and (M, B) as for the full edge (A, B).

we need to determine all direct neighbors of the new potential viewpoint. The set of potential neighbors contains all vertices which would be directly connected to the new vertex if the edge e were to be split. We can infer from Figure 6.1 that these vertices include the start and end vertex of e, plus the remaining vertices of the one (in case of a border edge) or two triangles sharing e.

After inserting the new vertex V', we must compute or update the weights of all edges in all triangles containing V', because V' is a potential neighbor of the midpoints of all those edges and thus contributes to all associated error values. In other words, the new vertex V' must be considered as one of the warping sources for computing any edge weight in the triangle fan around V'. For example, the insertion of V' in Figure 6.1 would induce recomputing the weights for all remaining and new edges (eight edges in total).

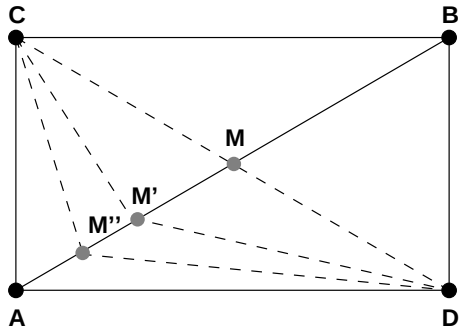

Figure 6.3: Suppose the error of viewpoints on the edge (A, B) is dominated by C and D. If we insert M, the error on (A, M) and (M, B) can still be nearly as high as before. This could cause an infinite sequence of splits creating $M, M', M'', \ldots$

6.3.2 On the Convergence of the Method

When an edge (A, B) is split and two new edges (A, M) and (M, B) are created, one would like the weights of the new half edges to be smaller than that of the old split edge. Usually, this proves to be true, since the part of the error which is dominated by the differences between A and B decreases (e.g. holes and single map regions become smaller, and the blending error decreases since the angular difference from which points in the scene are observed gets decreases by a factor of two). However, one can think of a few circumstances which can prevent the error from decreasing monotonically:

- New holes or single maps can occur when a viewpoint can see regions which have been completely occluded before (see Figure 6.2). This can cause a temporary increase of the weight on the split edge, but after the new half edges are split, the occluded regions become smaller and the error decreases. Since the scene details that can be observed are limited to the size of a (u, v) pixel, the increase of error must terminate at some point.

- The error can be dominated by the other one or two neighbors orthogonal to the splitting direction (C and D in Figure 6.3). If the error due to the difference between C and D does not get significantly smaller when inserting new points $M, M', \ldots$, this could generate an infinite sequence of splits of the same edge. On the other hand, if the difference between C and D is big, the weights of (C, M) and (D, M) must also be significantly big. Note that this kind of error domination cannot be caused by blending error, since the blending colors are weighted with their barycentric weights. So in the case sketched in Figure 6.3 the shorter edges would become more and more important, while the C/D edges would lose their influence on the overall blending error.

Although it does not seem very likely that one of these cases will actually prevent the algorithm from converging, the additional geometric constraints (Section 6.2.3) would come into play at

that point and stop unnecessary splitting chains.

6.4 Overall Algorithm

The overall algorithm for adaptively creating Lumigraphs is sketched in Algorithm 6.1. First an initial triangulation is created, and the initial input views are acquired. Then the benefit weights B_e are computed (Section 6.2) for the center of every edge e in this initial triangulation.

The main loop first selects the edge with the maximal acquisition benefit B, inserts a new vertex at the center of this edge, and acquires the image corresponding to that view. Then, the triangulation is updated incrementally, and the affected edge weights are recomputed (cf. Section 6.3).

The algorithm sketched here ends after the desired number of images is acquired. Instead of terminating the program after a fixed number of splits, one could also end the main loop after the maximal weight has dropped below some maximal error bound ϵ. For many applications, however, one would like to specify the number of viewpoints used in a Lumigraph, e.g. due to limited resources of the machine.

6.5 Results

We have built an experimental implementation of the adaptive Lumigraph acquisition algorithm as sketched in Section 6.4 and used it for computing adaptive Lumigraphs of several test scenes. In the experiments presented here, we have used the following parameters:

- hole weight $w_H = 0.1 \times$ maximal color value
- single map weight $w_S = 0.1 \times w_H$
- do not allow splits if any angle drops below 15 degrees
- do not allow a triangle to become smaller than $1/10000$th of the viewpoint domain

The actual acquisition was implemented by calling the `Vision` rendering system that has been developed at the University of Erlangen [127]. `Vision` is a very powerful and general image synthesis framework that supports high-end (and thus expensive) geometry and material representations as well as sophisticated rendering methods.

Figure 6.5 shows the four extreme views of the “elephant” scene that we already know from previous chapters. These four views are used as the images associated with the initial vertices as sketched in Figure 6.1.

The image resolution is 256×256 pixels, and ray tracing of a single view with an external rendering package takes approximately four minutes. In contrast, warping and error estimation

```
N ← number of desired viewpoints
create initial vertices Ṽ
for each vertex V ∈ Ṽ do
  acquire image I_V
create initial edges Ẽ
for each edge e ∈ Ẽ do
  compute benefit B_e
while |Ṽ| < N do {
  select e ∈ Ẽ with B_e = max{B_e' | e' ∈ Ẽ}
  split edge e
  V ← newly created vertex
  Ẽ_n ← {all edges of all triangles around V}
  acquire image I_V
  if [needed] then {
    flip non-Delaunay edges in Ṽ
    Ẽ_n ← Ẽ_n ∪ {all edges of all changed triangles}
  }
  for each edge e' ∈ Ẽ_n do
    compute benefit B_e'
}
```

Algorithm 6.1: Simplified pseudo code for the adaptive Lumigraph acquisition algorithm explained in Section 6.1 – 6.4.

for a view from four reference views consumes less than one second on a Silicon Graphics O2 195MHz R10k machine (without utilizing highly optimized code).

Figure 6.7 displays the viewpoint plane meshes generated after 40, 100, and 160 iterations of the algorithm. Right from the beginning, the lower region of the viewpoint plane has been sampled more densely due to the greater changes in color (highlights) and visibility (leg occlusions) when viewing the elephant from below. After 160 iterations, the time consumed for the Lumigraph acquisition was approximately ten and a half hours for ray tracing and 17 minutes for error estimation after split operations. Instead of the proposed incremental Delaunay triangulation, we performed a global Delaunay retriangulation of the mesh every 20 iterations, which required the recomputation of all weights eight times. This took another 30 minutes, but the cost could be significantly reduced by implementing efficient incremental update of the Delaunay triangulation. In total, the overhead for the adaptive acquisition lies somewhere between 2.5% – 7.5% of the rendering time even for this relatively simple scene. For more

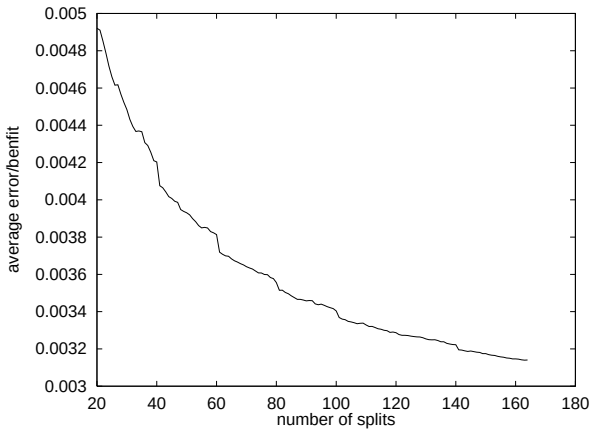

Figure 6.4: Average error/benefit value after each iteration for the “elephant” scene. The error decreases continuously smoothly during the whole acquisition phase.

complex scenes, the overhead should be negligible.

In order to analyze the contribution of holes, single maps, and blending error to the total error bound, we have visualized the different error contributions for each viewpoint before its insertion. Figure 6.7 shows the color-coded error distributions for each worst potential viewpoint (the next viewpoint to be inserted) after 40, 100, and 160 iterations, respectively. Please note that the colors of the error images have been enhanced in order to make the blending error more visible. As one can see, the initially large hole and single map regions (red and green, respectively) become smaller and smaller, although new small holes may pop up like in the bottom-most error image. The blending error also decreases drastically as the viewpoints fall closer together.

In order to validate the convergence of our approach towards a good Lumigraph reconstruction quality, we have plotted the average error/benefit value against the iteration number in Figure 6.4. The error decreases after almost every split operation, and the curve seems to converge. In accordance with these results, Figure 6.6 shows an image which has been generated by the Lumigraph refinement technique (cf. Chapter 4) for the next potential viewpoint after 160 iterations. The image quality is very high, and the image cannot be distinguished visually from the ray traced image.

6.6 Discussion

We have proposed a method for adaptively acquiring images for Lumigraphs from synthetic scenes. The acquisition is steered by a warping-based a priori error estimator which conservatively accounts for all shading errors and visibility events that can be derived from the given

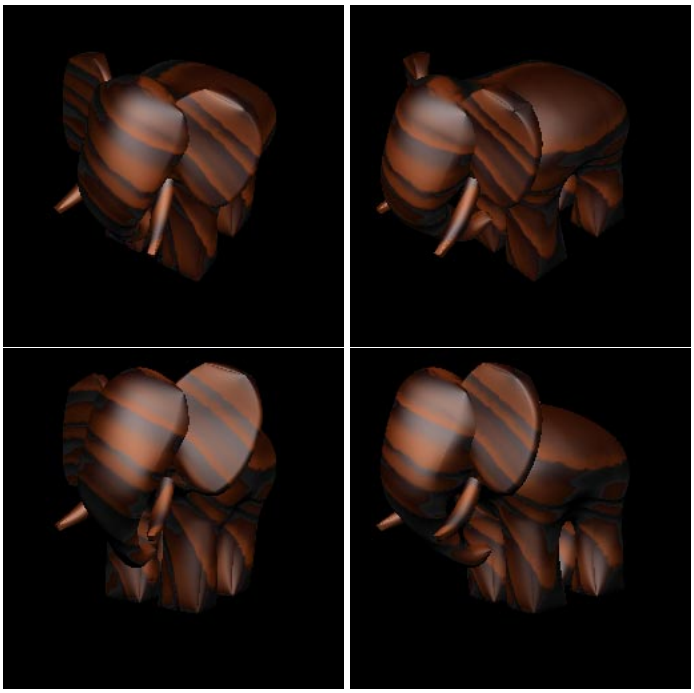

Figure 6.5: The four extreme views of the “elephant” scene. The differences between the views are the changing highlights on the ear and the hind legs, as well as several self occlusion effects.

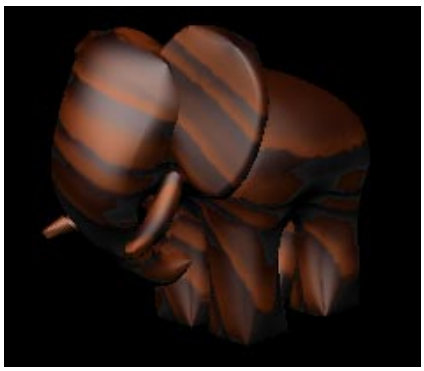

Figure 6.6: Morphed elephant corresponding to the next viewpoint to be inserted after 160 iterations (cf. last row in Figure 6.7). By eye, this image cannot be distinguished from the ray-traced original.

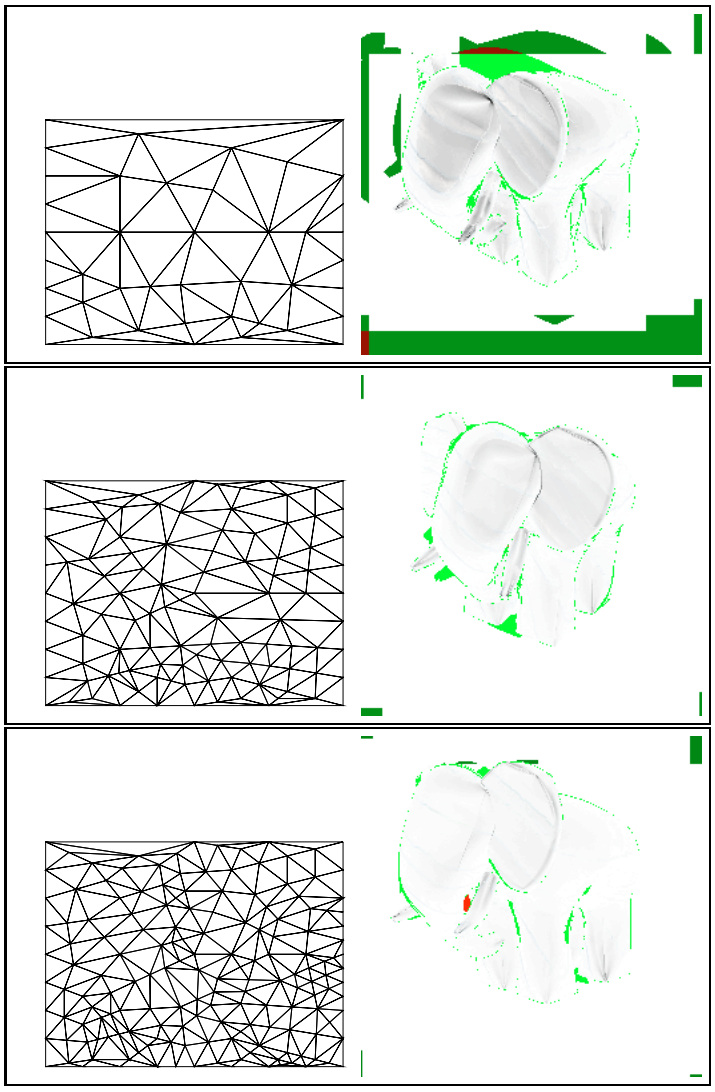

Figure 6.7: Left column: viewpoint plane meshes for the “elephant” scene after 40, 100, and 160 iterations (top down). Right column: corresponding error images for the next potential viewpoint. Darker colors denote bigger error, holes are red, single maps green. Darker red/green visualizes holes/single maps which do not generate any error because they map to the outside of a reference view (cf. Section 6.2.1).

set of viewpoints, without requiring any underlying material or lighting model of the scene. Our method enables the Lumigraph creator to specify the desired number of viewpoints or an error threshold, and then an optimal Lumigraph is automatically constructed in the sense of reconstruction quality when using Lumigraph warping.

The implemented algorithms have proven to be successful in that they generate adaptive viewpoint meshes which correspond to the color and visibility structure of the scene. The overhead for the adaptive method compared to pure image acquisition is relatively small and can be disregarded for complex scenes which take a long time to render. The error bound converges well in our examples, and the reconstruction quality of the generated Lumigraphs is very high. From subjective observations it can be said that the reconstruction quality seems to be evenly distributed in the viewpoint domain.

One limitation of our method is the fact that small highlights and small occluded areas cannot be predicted if they are not visible from any of the current viewpoints. Those faetures will then simply not appear in any of the reconstructed views. This problem can only be solved by a posteriori error bounds (which may not be affordable for scenes that take a long time to render), or by exploiting more information from the ray tracer like surface and reflectance data.

Another drawback of our method is that our "greedy" approach only takes into consideration how a *single* view will improve reconstruction quality. In order to yield truely optimal quality, it would be better to perform a more *global* optimization that finds the *set* of views that improves the quality best. But since every additional view changes the error metric for all surrounding views, it seems hard to solve this efficiently.

Another opportunity for improvement is the treatment of object boundaries. These depth discontinuities are usually undersampled because samples occlude each other (see also the color-coded weighting in Section 7.5, page 114). These undersampled boundaries will always keep the error measure from dropping to zero. So it would be very helpful to avoid this "false" increase of error, e.g. by using a morphological operator (erosion, like in Section 4.5.3) on the error image in order to wipe out all error contributions on the object boundaries.

In order to speed up acquisition by a large factor, one could acquire only those pixels of a novel view which cannot be reconstructed correctly by warping. This would save a large amount of rendering time. Furthermore, the results of this work could be applied to the acquisition of Lumigraphs from video streams. The a priori error estimator can be used to pick an optimal sequence of frames from a large stream of images.

Finally, our warping-based error predictor could be used very well for compression of light fields, since it can be used to identify redundant image data. For example, it could be used as an alternative predictor for disparity-compensated coding along the lines of Magnor et al. [83].

CHAPTER 7

Free-Form Light Fields

> Freedom is, therefore, only an idea of reason whose objective reality is in itself questionable.
>
> Immanuel Kant, *Groundwork of the Metaphysic of Morals*

The previous chapters presented different techniques that basically relied on the two-plane light field parameterization, which is probably the most popular one used so far. However, the very strict constraints on extrinsic and intrinsic camera parameters are inconvenient for many real-world applications. In this chapter we propose a *free-form* light field parameterization that employs a very flexible combination of freely placed cameras on an arbitrary convex free-form surface, but that still allows for very efficient hardware-supported rendering. In contrast to the recently introduced approach of *unstructured* light field rendering [53, 9], the convex free-form surface provides a consistent triangulation of the camera views and avoids heuristics as required by the completely unstructured approach.

In the context of fast hardware-supported rendering, we also discuss and solve the problem of partial occlusion, occurring when one or more of the "closest" input views do not see parts of

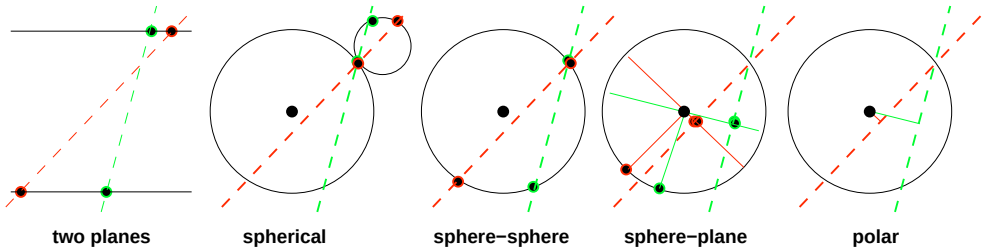

Figure 7.1: 2-D sketch of light field parameterizations, for two different rays to be parameterized (dashed lines). Also compare discussion in Section 3.2.

the scene. In warping-based rendering as discussed in previous chapters, the occlusion problem was solved rather trivially by depth testing in the compositing phase.

Free-form light field rendering uses the original images directly and involves a single hardware-filtered resampling step through texture mapping. We discuss both polygon-based and warping-based rendering techniques. With polygon-based rendering, original image quality and resolution are optimally exploited, and we demonstrate that rendering can be done very efficiently at rates comparable to those known from traditional two-plane-parameterized light field rendering.

7.1 Motivation and Overview

As we have shown in the previous chapters, the two-plane light field parameterization is simple and convenient, and allows for designing very efficient rendering algorithms. Alternative parameterizations have been proposed by several authors (cf. Figure 7.1) for different reasons such as coherence/compression, sampling uniformity, and for seamless coverage of all possible viewing directions with just a single light field entity instead of multiple slabs [54, 12, 11, 132]. Furthermore, in cases where the *exact* scene geometry is available in the form of polygon meshes, and when preprocessing time is not an issue, it has turned out to be very convenient to use the surface parameterization directly for light field parameterization [97, 146]. Please refer to Section 3.2 for more details.

One practical and important problem is light field acquisition from real-world scenes, since it is not easy to set up an array of cameras in such way that they match the parameterization, except by using robot gantries [74] or custom-built camera arrays [147, 134]. If the geometry of the recording unit does not match the desired parameterization, the corresponding image data has to be resampled (“rebinned”), which can be very time-consuming, and decreases image quality.

One solution to this problem is so-called *unstructured* light field rendering [53, 9]. In this approach an arbitrary set of cameras is used, and their centers of projection are projected

into the current view and triangulated for every single frame during rendering. In [9] different heuristic metrics are used to weight the contribution of each eyepoint to the final image, based on directional deviation, occlusion, image resolution, and similar criteria. Choosing and weighting these criteria is a rather sophisticated task, and the user has to fine-tune a whole number of parameters in order to get optimal rendering for a certain scene.

With this work, we follow a slightly different path and try to find the most flexible light field representation that does not require frame-to-frame retriangulation or multiple heuristic image weighting. We discuss what is required to maintain a consistent and simple reconstruction from the three nearest views in the light field, very much following the original ideas and assumptions of light field rendering.

The two main contributions in this chapter are the proposal and discussion of the actual *parameterization* (Section 7.2), and *efficient rendering* from that parameterization, both using a polygon-based (Section 7.3) or warping-based (Section 7.4) technique.

We first introduce the free-form parameterization in the continuous case (Section 7.2.1) and then apply it to the discrete case (Section 7.2.2). Then we discuss in more detail the reasons for restricting the eyepoints of the reference views to a convex surface (Section 7.2.3), and close the discussion of our parameterization showing similarities and differences with respect to other parameterizations.

After explaining the actual parameterization, we first propose an efficient polygon-based rendering algorithm (Section 7.3) that is based on the original Lumigraph rendering algorithm with adaptive depth correction [37]. However, we project the reference textures directly onto the approximated scene geometry, similar to Heigl et al. [53]. We also discuss how to do the adaptive depth correction efficiently with both triangle meshes or per-pixel data as the geometric proxy (Section 7.3.2). Finally, we explain the problem of partial occlusion and show how to perform occlusion correction most efficiently (Section 7.3.3).

In Section 7.5 we show a number of different results obtained with our implementation of free-form light fields (polygon-based rendering), and compare the quality and performance of our technique with that of classical two-plane parameterized rendering. In Section 7.6 we close the chapter with a discussion of the results and possible future extensions of our method.

7.2 Convex Free-Form Parameterization for Light Fields

Our proposed free-form light field parameterization is a simple and convenient generalization of the two-plane parameterization, and also shares some ideas with sphere-to-plane light fields.

The most important idea behind the free-form parameterization is that it allow a number of arbitrary cameras to be placed all around an object in order to represent its appearance from all possible directions. This set of input (or reference) images is used directly as light field data, without any costly and imprecise resampling before the actual rendering.

Like any sphere-like parameterization (spherical, sphere-sphere, sphere-plane), free-form

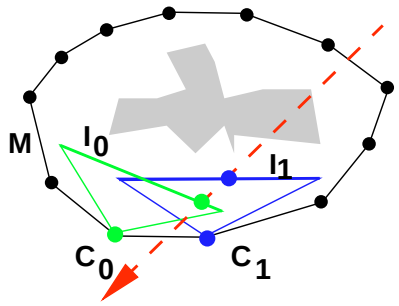

Figure 7.2: Finding the nearest samples for a given ray involves intersecting the ray with the camera mesh M as well as with the images planes I_0, I_1 associated with the nearest camera vertices C_0, C_1 (in 3D there are three nearest C_k).

light fields allow seamless representation and viewing of *all* views around an object. However, the reference eyepoints do not need to lie exactly on a sphere, but are only constrained to lie on an arbitrary convex free-form surface. The surface may be open or closed, so a free-form light field can represent partial as well as complete light fields of a scene.

7.2.1 Continuous Parameterization

Inspired by the acquisition process using real cameras, a free-form light field is defined by a *convex* polygon mesh M called *camera mesh*. The camera mesh consists of N *camera vertices* $C_k, k \in \{0 \dots N-1\}$, each representing a camera's center of projection. Associated with each camera vertex C_k is an *image plane* I_k. All samples through one camera vertex C_k form a perspective image on the associated image plane I_k (cf. Figure 7.2). The cameras must be set up in such way that every camera sees the *full silhouette* of the scene (no part of the scene is outside the camera's viewing frustum).

Imagine a continuous light field with a densely populated camera mesh (i.e. every surface point on the mesh is a camera vertex). In this case an arbitrary ray passing through the light field can be thought of as parameterized by its first intersection (s, t) with the camera mesh and its intersection (u, v) with the associated image plane $I_{(s,t)}$. Like in the two-plane parameterization, a ray is parameterized by a 4-tuple (s, t, u, v), and again we can imagine (s, t) to be an eyepoint, and (u, v) to be a pixel. The only difference so far is that the pixels do not need to lie in the same image plane.

7.2.2 Application to Discrete Light Fields

In practice, the light field contains only a finite number of camera vertices and pixels, and we assume that the camera surface is a triangular mesh consisting of eyepoints as vertices.

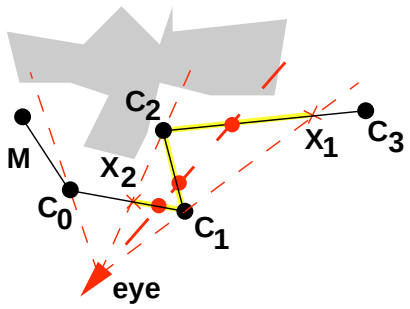

Figure 7.3: If the camera mesh is concave, a ray may intersect the mesh multiple times upon entering the scene. The contributions from (X_2, C_1), (C_1, C_2), and (C_2, X_1) overlap.

In order to reconstruct the light along a certain ray (s, t, u, v), we must find and interpolate the "nearest" light field samples. This is done by first finding the three camera vertices C_0, C_1, C_2 of the triangle that encloses the intersection point (s, t). For each of these camera vertices C_k we can determine the intersection $(u, v)_k$ of the desired ray with the corresponding image plane I_k. Since that image-plane intersection will in general not fall exactly in the center of a reference pixel, we use the nearest four pixels (in the regular pixel grid). So in the end the light along a ray is reconstructed from 3×4 reference samples.

7.2.3 Why a Convex Shape?

Figure 7.3 shows why we assume the camera mesh to be convex. If this constraint is violated, a ray may intersect multiple times with the camera mesh, so that the contributions from the three patches (X_2, C_1), (C_1, C_2), and (C_2, X_1) overlap when seen from the eye. For software rendering, this could be taken into account by a modified dynamic interpolation computation. In order to yield a smooth transition between the contributions from C_1 and C_3, the algorithm would need to "blend out" the contribution from C_3 at X_1, which requires the computation of C_1's projection onto (C_2, C_3). So non-convex camera meshes require higher computational effort, either for projecting the camera vertices into the viewer's image plane and performing a retriangulation for every frame as in Heigl et al. [53], or for normalizing an arbitrary number of blending weights, an operation which is not available for hardware-supported interpolation. As we show in the next section, alpha blending can be used for convex meshes because the interpolation weights can be precomputed for every camera patch.

The full-silhouette constraint is needed to make sure that each of the "nearest" three cameras in the mesh actually contains data for the desired image patch, which is one prerequisite for the fundamental assumption stated in Section 4.3 on page 57 (sampling is dense enough so that each ray can be reconstructed from only the closest views). But still there may be cases where one of the nearest views has no information about a certain point in the scene because

of self-occlusion, which will be handled in Section 7.3.

7.2.4 Relation to Other Parameterizations

It is obvious that the free-form parameterization is a generalization of the two-plane approach. In order to yield the equivalent of one two-plane slab (compare Figure 3.3 on page 38), use a plane as eyepoint mesh M, define an image plane I parallel to M, define an image domain $dom(I)$ on I, and set up all cameras to use a sheared perspective projection to map exactly to $dom(I)$.

Similarly, one can create a complete two-plane light field consisting of six slabs by defining M to be a cube, and for each of the cube faces defining one image plane as for a single slab. The only difference to a "real" multi-slab light field is the treatment of the cube edges. In a two-plane light field, each slab has its own image plane, and the camera vertices on the edges of the cube are defined twice, once for each slab they belong to. Since these camera vertices are exactly the same for both slabs sharing the cube edge, light field blending can be performed smoothly and seamlessly across that edge. In the free-form parameterization, however, this vertex doubling is not necessary. Each edge vertex can have an arbitrary image plane. When actually "emulating" a multi-slab light field, we assign to each edge vertex the image plane that gives the best sampling of the scene.

It should be noted here that the free-form parameterization cannot mimic a spherical, sphere-sphere, or polar parameterization (cf. Figure 7.1), since in these cases the images are not defined on planes. The sphere-plane parameterization is different in that respect [12]. All rays with the same direction share a single image plane, so the sphere-plane parameterization is very well suitable for the representation of orthogonal views (each eye vertex C_k represents one direction to the center of the sphere, and stores the image on the plane I_k perpendicular to that plane). For these views, the "nearest" camera vertices are not defined by the distance to the intersection of the desired ray with the eye mesh, but by the difference in direction between the desired viewing ray and the eye vertices' direction vectors.

This means that for orthogonal views, the selection of the nearest neighbors is different, since the eye vertices represent directions rather than spatial locations. Once we have found these "nearest" eye directions (C_0, C_1, C_2), we can proceed as before by intersecting the desired viewing ray with the image planes (I_0, I_1, I_2). So with this slight change it is possible to incorporate orthogonal views and mimic the sphere-plane parameterization.

Last, but not least, the free-form light field shares ideas with the approach of unstructured light field rendering [53, 9]. That approach does not impose restrictions on the cameras used, since the eyepoints are treated as "scattered data". For every new frame to be rendered, the original eyepoints are projected into the current view and then tessellated in order to yield the adjacency information needed for smooth blending. Since there are also no restrictions like our full-silhouette constraint (cf. Section 7.2.1), some of the cameras might be close-up views or partial views of the scene, and so more than the three nearest neighbors might be necessary

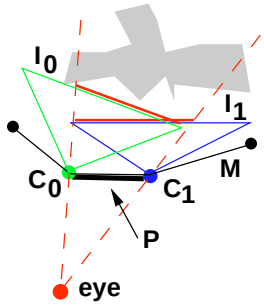

Figure 7.4: Polygon-based rendering. Seen from the eye, the whole patch P should be interpolated from its viewpoints C_0, C_1, so we project P onto I_0 and I_1 and draw the resulting patches with the corresponding alpha-blended textures.

in order to reconstruct a certain ray. Cameras can represent the same information in different resolutions. Defining a proper weighting of the different contributions is not an easy task. The free-form parameterization avoids this, at the cost of restricting the viewing space.

7.3 Polygon-based Rendering

As we will show, free-form light fields can be rendered very efficiently using either an approach based on textured polygons or warping. In this section we explain in more detail the polygon-based approach.

7.3.1 View Selection and Blending

Texture-based rendering of free-form light fields can be done in much the same way as in the original Lumigraph framework [37], and as proposed later in Heigl et al. [53] in the context of unstructured light fields. We observe that for all rays passing through one *camera patch* $P = (C_0, C_1, C_2)$ on M, we want to render P's projection onto each of the three image planes I_0, I_1, I_2 (cf. Figure 7.4). Each of these projections corresponds to the image on P as viewed from one of the cameras C_k. Similar to previous work, we draw these three texture-mapped patches. The important difference is that we do not use a common image plane, but draw each contribution on its own image plane. Viewed from the desired eyepoint, it makes no difference whether or not these patches lie in a common plane (as in the two-plane case). As we will see in the next section, the patches are actually drawn on the approximated scene geometry if scene geometry information is used and depth correction is performed. In that case, the same polygon is used for all three fragments, and the texture is properly projected from the reference views on that polygon.

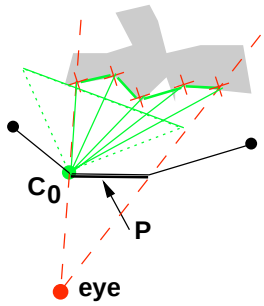

Figure 7.5: Adaptive depth correction for the image associated with vertex C_0 and patch P. After subdivision, small depth-corrected texture triangles are drawn that faithfully represent the scene geometry for the current view.

In order to use only those camera patches that actually see the front-facing side of the scene, we perform a simple back-facing test, either by comparing the camera patch normal, or the camera's optical axis to the viewing direction.

In addition to bilinear texture filtering that is performed to interpolate between the pixels of each patch, alpha values are used to perform the final interpolation between the patches. For the texture associated with camera C_k, we use an alpha value of 1 for the projection of C_k onto I_k, and alpha values of 0 in the other two vertices of the patch.

7.3.2 Adaptive Depth Correction

If the sampling of the light field is not very dense (e.g. if we only have captured a few views of the scene), the classical light field rendering approach leads to serious blurring and ghosting artifacts. This can be compensated for by performing *depth correction*, if in addition to the image data some approximation of the scene geometry is available. We shoot rays through the camera vertices and find the depth values of the ray's first intersection with the scene geometry either by actually performing intersection tests [37], reading back the OpenGL Z buffer [137], searching in a binary volume [12], or searching in a per-pixel depth map [138]. With depth information for each vertex, we move the vertices to their true 3-D location in order to take advantage of correct perspective interpolation as featured in concurrent graphics hardware (see Figure 7.5).

In addition to the three vertices, we also generate a depth ray for the center of the triangle. If all four depth values are approximately the same, we draw the triangle as it is. If the values differ by more than a certain depth threshold, or if one, two, or three of the four rays do not intersect the scene at all, we subdivide the triangle and perform depth correction recursively[1].

[1]The depth value for camera vertices can be determined by a simple depth map lookup. However, for arbitrary other points (e.g. during subdivision), a depth map search has to be performed in order to find the corresponding

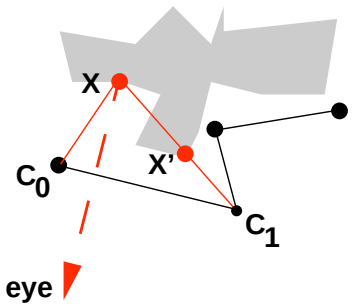

Figure 7.6: Occlusion. Although C_0 and C_1 are the two closest views for the desired reconstruction, C_1 sees X' rather than the desired point X due to scene self-occlusion.

Furthermore, we stop at a minimum triangle size (specified in the viewer's image space), and enforce an initial subdivision by means of a maximal triangle size. This way the image plane triangulation adapts locally to the scene geometry as far as needed for the reconstruction of the current view, and the three cameras that define the subdivided patch finally use the same depth corrected image plane mesh (the polyline between the small crosses in Figure 7.5). In order to avoid artifacts at patch boundaries, we also perform a T-vertex removal by further splitting edges that are subdivided in one patch, but not in the neighboring patch.

7.3.3 Occlusion Correction

In addition to depth-correcting the texture coordinates, one must also determine if the desired scene parts are actually visible from the input views that have been chosen for the reconstruction. An example of this problem is illustrated in Figure 7.6. Although C_0 and C_1 are the closest views for reconstructing the desired scene part (e.g. point X), using C_1's image data would blend a completely different part of the scene with the desired one, resulting in a wrong reconstruction. The solution to this problem is rather straightforward. During depth correction we already determine the depth of each texture vertex (along the ray from the eyepoint). Now we can check if a ray from this point (e.g. X in our example) to each of the cameras intersects the scene geometry. In the case of per-pixel depth maps, we simply check if the 3D location of the vertex in the different input views is the same as that of the desired point by reprojecting and comparing the corresponding pixels with their respective depth values. With polygonal geometry or a binary volume representation, an additional ray casting step is needed to detect an intersection with the scene geometry.

If a vertex is occluded from a camera view C_k, we set the blending weight for the texture at this vertex to 0, and adjust the two other weights in the corresponding triangle in such a way

point (inverse mapping). In contrast to Heigl et al. [53] who did not address this problem, we solve this problem along the lines of Vogelgsang et al. [138].

that they still sum up to 1 (so the patch will not appear darker), and the remaining weights maintain their original ratio (so their relative importance is not changed). For example, let the three weights be x, y, z, and the z vertex must be removed due to occlusion. In that case we add $z \cdot x/(x+y)$ to weight x, add $z \cdot y/(x+y)$ to weight y, and set z to 0. If a vertex is occluded in two views, the remaining view receives a weight of 1. If a vertex is occluded in all three views, we set all three weights to 0.

7.4 Warping-based Rendering

As we have seen in the previous chapters, warping is an alternative light field rendering technique that is especially suitable if the input data includes per-pixel geometry information. In this section we briefly describe the major differences when using a warping-based approach for free-form parameterizations.

7.4.1 Reprojection Between Arbitrary Planes

When using warping for reconstructing views from a free-form light field, the first question is what should be the target plane for the reprojection. Since it does not seem to make sense to define an artificial common image plane for "collecting" and compositing all contributions, there seem to be two "natural" choices:

1. Use approximated scene geometry patches as for the polygon-based rendering, and do the final reprojection into the current view using texture mapping (similar to the two-step reprojection used in Chapter 5).

2. Directly reproject into the viewer's image plane.

At first glance, the second solution seems to be simpler and more efficient. The only difference between the two approaches is that the resolution variation of all the reprojected image fragments may be much larger in the viewer's image plane than in the "local" planes where only three nearby views (that will usually have similar resolutions) are fused. Techniques how to deal with the resolution variation will be discussed in the next section.

Since the actual reprojection is between two arbitrary planes (the relative transformation of which changes with the viewer), a general 3D pixel reprojection must be used, e.g. along the lines of McMillan [94].

7.4.2 Partitioning

Another important question for free-form light field warping is how to do the partitioning (cf. Section 5.3) in order to avoid the reprojection of pixels that do not contribute to the desired regions.

The region growing mechanism presented in Section 5.3 relies on the two-plane disparity which is meaningless in the context of arbitraily oriented planes, and one would need to examine whether it is possible and reasonable to derive a similar mechanism based on *general disparity* as introduced by McMillan.

Alternatively, one can use the same adaptive depth correction algorithm as presented for the polygon-based rendering (Section 7.3), and find the right source patches by recursive subdivision of the patches that seem to cover a large depth interval. The only basic mechanism needed for that is efficient querying of a depth value for a given target ray/pixel. In the context of per-pixel depth map (note that depth is given per source pixel, not per destination pixel), this can be implemented along the lines of [138].

7.5 Results

In order to validate our approach, we have implemented the described polygon-based technique and performed some experiments with real-world and synthetic light field data. All tests were performed on standard PC hardware with a 1GHz Athlon processor and GeForce2 GTS or GeForce2 MX 32 graphics. If not stated otherwise, all input images have a resolution of 256x256 pixels, and geometry information is given as per-pixel depth maps.

7.5.1 Image Fidelity

Figure 7.7 compares two-plane (left) to free-form (right) rendering, both Lumigraphs being generated from the same set of 33 photographs. Usually the two approaches yield roughly the same image quality. In this example the two-plane resampling step significantly degrades the image quality, and a more sophisticated warping/reconstruction software is necessary in order to maintain maximum image fidelity. Figure 7.8 shows a view from a different, 169-image free-form Lumigraph (“Petruschka”). Again the visual quality is very high, and the object boundaries appear sharp and clear.

Figure 7.9 shows a view of a synthetic statue, rendered from a complete (all-around) free-form Lumigraph consisting of 107 images. Here one can observe the effect of depth and occlusion correction on image quality (leftmost: no correction; second-left: correction enabled). We also visualize the depth-correction grid as well as the color-coded blending weights. As expected, the weights are discontinuous where occlusion in one or more input views occurs.

7.5.2 Performance Issues

The performance of free-form light field rendering is comparable to that of two-plane parameterized rendering with irregular eyepoint meshes. The basic difference is that for free-form light fields, every eye mesh vertex (or the corresponding depth-corrected 3-D point) has to be

projected into three different planes instead of one. Without depth correction, this does not increase rendering time significantly, since the number of eyepoints is fairly small. In our current implementation, all the light fields shown here can be rendered at up to 100 frames per second without depth correction.

When depth correction is switched on, lots of eye mesh vertices are introduced by the adaptive subdivision scheme, and so the free-form overhead is more noticable. For the "coke" sequence (Figure 7.7), both two-plane and free-form Lumigraphs of the demonstrated quality can be rendered at roughly 30 frames per second. For the "Petruschka" scene (Figure 7.8), the free-form frame rates vary between 14 (780 rendering triangles) and 1.7 (5000 triangles). The corresponding frame rates for two-plane rendering are a factor of 1.2 – 2.5 times higher. The statue scene (Figure 7.9) can be rendered at speets from 60 (500 triangles) to 4.6 (3500 triangles) frames/second, again with a performance gain of 1.5 – 2.1 for two-plane rendering. 7–10% of the total free-form rendering time is spent for occlusion correction.

7.6 Discussion

We have introduced the concept of *free-form light fields* that combine the efficiency and simplicity of traditional two-plane light fields with the flexibility and ease-of-use of an almost arbitrary camera arrangement and seamless "all-around" viewing. We have shown that a convex eyepoint surface is the most general concept that still allows for a static eyepoint triangulation, and that it is a generalization of the two-plane parameterization. We have also presented efficient rendering algorithms, both polygon-based and warping-based, and solved the problem of partial occlusion, which is especially important in order to yield crisp object boundaries.

We have implemented and tested the polygon-based renderer and compared its quality and performance to a highly optimized implementation of two-plane-parameterized polygon-based rendering. We have shown that the additional cost for the free-form parameterization is relatively small, and that the reconstruction quality is comparable or even higher.

We believe that free-form light fields are by far the most convenient approach for representing and rendering inward-looking light fields of objects, especially from real-world data. They allow for almost free camera arrangements, and all intrinsic and extrinsic camera parameters such as position, orientation, focal length, and resolution may vary between the different images. Hardware-supported texture filtering is exploited efficiently to resample this data into the desired view. Furthermore, implementation of free-form light fields is no more complicated than that of other light field techniques.

One of the most interesting points is maybe the relation to the "unstructured light field" approach. As already discussed in Section 7.2.4, rendering an unrestricted light field is much more involved since no constraints are imposed on the cameras used, and a lot of effects (and more contributions for every ray to be reconstructed) have to be taken into consideration. The unstructured approach allows the user to navigate through a set of "scattered" views, while the

Figure 7.7: Light field view of the “coke” scene, reconstructed from 33 real images. Left: two-plane parameterization Right: free-form parameterization. The two-plane Lumigraph has been resampled into 30 plane-aligned images in a pre-process step, whereas the free-form Lumigraph on the right directly uses the original camera frustra.

Figure 7.8: “Petruschka” scene, reconstructed from 169 synthetic images using the free-form Lumigraph rendering approach.

free-form light field concentrates on the representation of the appearance of a single object, following up on the original idea of a light field that was based on representing a bounded region of space from the outside (or vice versa. everything outside that region from the inside). The future will show which of the two approaches is more suitable for what kind of application.

One of the most useful extensions of our approach would be to find a way of integrating partial and close-up views without sacrificing the simplicity of our solution, and to find a general framework for using multi-resolution techniques to represent the object on multiple scales. Another important direction of work to make this technique complete is the automatic construction of a suitable eyepoint mesh, which addresses topics such as mesh generation, view selection, and acquisition planning.

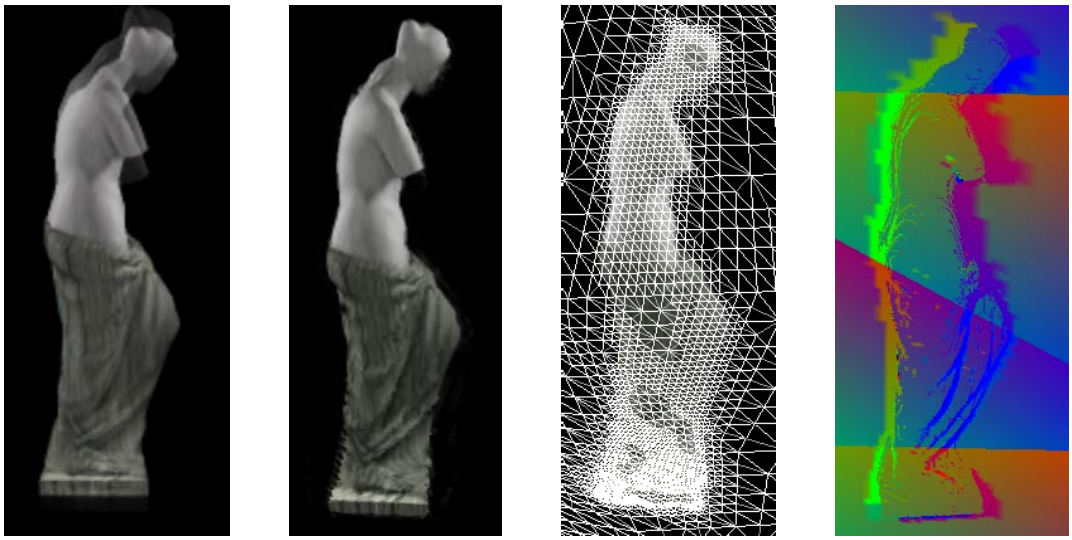

Figure 7.9: View from a complete (all-around) free-form Lumigraph of a statue (107 synthetic images). Left to right: without depth correction (pure light field approach); with adaptive depth correction and occlusion correction; depth-correction grid (image plane triangles); occlusion-corrected blending weights (color coded). Note the typical ghosting in the upper part of the uncorrected reconstruction (left image), and the discontinuous occlusion-corrected weights along the sharp object boundaries (rightmost image).

CHAPTER 8

Instant Lumigraphs from Dynamic Scenes

> You cannot step into the same river twice, for fresh waters are ever flowing in upon you.
>
> Heraclitus

In the previous chapters we have advocated for a light field representation including per-pixel depth and warping, and we have shown that this approach yields a good and simple representation for efficient and high-quality rendering of static Lumigraphs. Furthermore, we have introduced free-form light fields, a generalization of the two-plane geometry that allows the flexible use of real-world cameras without intermediate resampling steps.

In this chapter, we put together many ideas from these techniques and representations, and we present a distributed image-based rendering framework that handles all processing stages from image capture to compositing and rendering *on the fly*, and thereby allows for instantaneous processing of *time-varying* scenes at interactive frame rates. The proposed architecture is scalable in the number of sensors, since a region-of-interest mechanism derived from the partitioning approach in the previous chapters keeps the total bandwidth requirements nearly

independent of the number of sensors. Our framework integrates all kinds of sensors that are able to deliver color and depth information.

Furthermore, we demonstrate an early version of a complete tele-immersion system called the *Lumi-Shelf*, which is based on the proposed framework and combines a set of consumer quality video cameras and standard PCs with a hierarchical correspondence-based depth-from-stereo software and the proposed Instant Lumigraph rendering back-end.

8.1 Motivation and Overview

Efficient and flexible image-based scene representation is still one of the most important challenges in computer graphics, especially for the representation of *time-varying* scenes and *immediate* processing of multi-image data for redisplay. These are requirements for applications such as interactive 3-D movies, augmented reality, and tele-immersion.

Since *purely* image-based techniques such as light field rendering rely on a large number of very densely spaced cameras, they require custom light field camera hardware [134]. Other techniques need only a few input images, but some additional geometric information for reconstructing views from the scene. For non-synthetic scenes, this geometric information has to be inferred from the input images, or must be acquired by additional sensors.

Thanks to years of computer vision research, bi- and trifocal camera sensor systems are currently emerging on the market and will soon provide us with hybrid image and geometry data (per-pixel depth) at video frame rates. Several of these sensors have to be combined in order to represent a scene, and the data from these different sensors must be fused in order to reconstruct arbitrary views from this representation.

One important problem is that efficient multi-image rendering algorithms require computationally expensive intermediate data representations that cannot be computed on the fly. The most prominent examples for this kind of scene representations are “classical” Lumigraphs, layered depth images, and surface light fields. They all share the idea of resampling and processing the image and geometry input data to allow for very efficient and high-quality rendering of the represented scene, and this resampling and processing is usually very expensive.

On the other hand, a simple and efficient approach as 3D warping is not sufficient in the presence of multiple input images, since it is often not desirable to transmit and reproject *all* image data from *all* cameras just to compute a single output image for the current view. So something like the partitioning scheme introduced in Section 5.3 must be employed in order to find and transmit only the regions of interest from every input image.

In this chapter, we present a method that solves these problems by putting together many pieces from the previous chapters of this thesis, adding a depth reconstruction technique, and arranging all these components in the right way in a processing network that optimally distributes the computational load and minimizes communication between the network nodes. In contrast to any previous work, our approach allows 3D rendering of time-varying, non-diffuse

scenes from an arbitrary number of cameras, at interactive frame rates.

In the next sections, we will construct a framework that splits up the different tasks involved in the process in a reasonable way, and that defines the data flow between these tasks. We also present a prototype for rendering Instant Lumigraphs from real-world scenery, and show a number of different results obtained with that prototype.

We start with a system overview (Section 8.2) that explains what the different components are and how they work together. Then in Section 8.3 we present the basic data structure called a *single-slab free-form Lumigraph*, which is half-way between two-plane and free-form light fields with per-pixel depth. Next, we explain the two remaining parts of the framework: acquisition (Section 8.4) and rendering (Section 8.5). Finally, we introduce an experimental prototype of an Instant Lumigraph called the *Lumi-Shelf* (Section 8.7), which uses off-the-shelf video cameras and a software-based depth-from-stereo technique for reconstructing geometric information. We present results obtained with ths prototype as well as with other kind of data (Section 8.8). We conclude with Section 8.9, where we discuss the results and point out future directions of research in the area of Lumigraph-based tele-immersion systems.

8.2 Instant Lumigraph System Overview

The Lumigraph processing framework consists of two main parts, the *acquisition* or *sensor* part and the *rendering* or *display* part. The sensor part includes the image capture as well as depth reconstruction (Section 8.4), whereas the rendering part performs the sample selection (region-of-interest computation), sample reprojection, and compositing (Section 8.5).

Depending on the computational demands or physical location of the sensors and the computational resources of the display host, Instant Lumigraph processing can be parallelized and distributed on several inter-networked nodes. The most natural way to do this is sketched in Figure 8.1. The data of several sensors is processed by a *sensor node*, which transmits the desired data to a *display node*.

The display node is informed about the desired view, and exploits this information for determining a so-called *region of interest* (ROI) for every sensor. This region of interest is an estimate of the subset of image data needed from the sensor, and helps reducing the transmission load significantly. Additionally, the sensor might use the ROI information to reduce the internal bus load or computation cost (e.g. perform depth reconstruction only on the desired scan line segments).

Moreover, the sum of data in all regions of interest increases sub-linearly with the number of sensors, so that this scheme scales very well to a large number of sensors, with or without network connection.

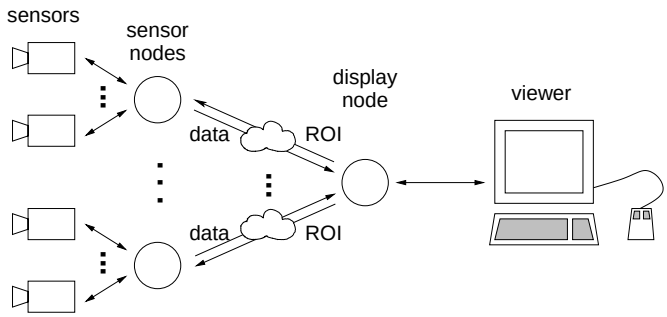

Figure 8.1: Distributed Lumigraph processing. The display host is connected to a number of sensor hosts. Each sensor host processes an arbitrary number of sensors. The display host updates the region-of-interest information for each sensor, and the sensors send the corresponding image regions.

8.3 The Single-Slab Free-Form Lumigraph

In order to capture a time-varying scene, we usually observe it with a set of calibrated video cameras. Given that we have some means of additionally reconstructing a dense depth map for each of these cameras, this leads us directly to the idea of using a free-form light field representation with per-pixel depth information. Choosing this representation, we only need to make sure that the cameras are set up in such way that their optical centers lie on a convex surface, and then we can apply the whole machinery presented in Chapter 7 in order to identify the interesting regions and render the images efficiently.

However, since the work in this chapter was developed partially in parallel to the work on free-form light fields, and because of several reasons that have to do with efficient rendering and region-of-interest detection (see Section 8.5), we have chosen an approach that lies halfway between a light slab representation and a free-form light field. Like in the free-form setup, cameras (or sensors) S_k lie on a convex free-form surface and are relatively free in all of their intrinsic and extrinsic parameters. But similar to the two-plane parameterization, we define a common image plane I that will be used as the target for the reprojection. In order to avoid dealing with resolution issues, we restrict the cameras to keep roughly the same image resolution when projected to I. Since this is only possible if the optical axes of the cameras are all roughly orthogonal to the image plane, the whole setup is somewhat similar to the two-plane light slab, and we call it a *single-slab* free-form parameterization (cf. Figure 8.2). Like before, every sensor S_k should see the complete object silhouette.

For the set of camera eyepoints, we can easily construct a *camera surface* M by projecting the camera points onto the image plane, triangulating the projected points, and lifting that triangulation back into 3-space. Now we have a complete single-slab Lumigraph, defined by the image plane and the set of cameras with their intrinsic and extrinsic parameters.

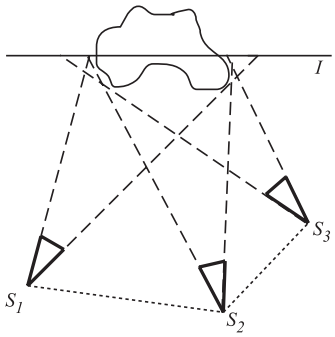

Figure 8.2: Single-slab free-form light field. Each sensor with optical center S_k must see the object's silhouette completely, and neighboring cameras should have roughly the same resolution on the common virtual image plane I.

8.4 Acquisition and the Sensor Nodes

In the Instant Lumigraph framework, a sensor is treated as a black box that delivers a 2-D image containing dense color and depth information. This data may or may not be varying over time.

Still images with depth can originate from many sources. For real-world images, depth values can be reconstructed by techniques like voxel coloring [118][1], while for synthetic images depth is usually available directly through the rendering engine (e.g. ray tracer, OpenGL, etc.).

Practical examples for time-varying sensors are stereo camera pairs, cameras synchronized with range finders, and many more. With the same framework, we could also support hardware like a camera matrix (cf. Section 3.8), if its output is split again into individual images.

In the context of our framework, the sensors can be used optimally if they support a region of interest (ROI). This means that a ROI is specified right before the acquisition, and the sensor reconstructs color and depth data only for the desired region. For example, a stereo camera pair could take into account the ROI by only processing the scan lines that go through the ROI, and by restricting their correspondence search range within the scan line to the ROI, extended by the algorithm-dependent search radius.

Another example are modern CMOS cameras, which support the readout of only a certain ROI directly in hardware. This can help speeding up the image capture, and reduces the bandwidth from the camera to the sensor host. This is especially important if multiple cameras are driven by a single sensor host.

[1] Please note that techniques such as voxel coloring are not suited for on-the-fly processing (cf. Section 3.8), but have to be executed in a preprocessing step.

8.5 Rendering and the Display Node

Rendering arbitrary views from the single-slab free-form Lumigraph can be done in very much the same way as for the two-plane Lumigraphs presented in Chapter 5. Again we reproject the desired pixels from the input images into the common image plane I, taking into account the current viewpoint of our virtual camera. The important difference is that the image planes of the input cameras are not parallel to I, so the specialized formulae in Section 5.2 do not apply here. One needs to use the general plane-to-plane 3D warping techniques as for example presented by McMillan [94] (cf. Section 2.4). Our restriction to a fix common image plane has three advantages for rendering:

1. Since the transformation between input images and virtual image plane are fix, we can precompute and store most of the warping coefficients for every input pixel.

2. Because of our assumption of similar resolution of the input images on the virtual image plane, we do not need to deal with sophisticated (and time-consuming) reconstruction filters.

3. We can measure the pixel flow in the virtual image plane and use this flow information for the succeeding frames (with a time-varying plane, the flow information cannot be reused).

It is obvious that our concept of a single-slab light field gives us several advantages concerning the simple and efficient implementation of the Instant Lumigraph prototype.

As in the chapters before, our warping-based rendering algorithm performs a partitioning on the virtual image plane I, determines the interesting source region (ROI) for every input view, reprojects all samples from that region from every input view, and composites the samples into a single texture on I. Finally, I is drawn as a texture-mapped rectangle using OpenGL.

The resolution of the image plane texture is found by projecting pixels from all input images onto I and determining the corresponding resolution. Then, we use half of the average of this resolution for our image plane texture. Obviously, this approach only works well if the projected resolution of the input images does not vary too strongly on I. In addition to this automatic resolution determination, we introduce a fine-tuning parameter that allows the user to scale the image plane resolution.

The reprojection itself is done in a straightforward way, as a full 3D warping operation. However, since the transformation between the input images and I is constant, we precompute and store most coefficients for that warping for every input pixel. Alternatively, one could use an efficient incremental warping implementation.

During reprojection, multiple input samples will be projected to the same texture cell on I. We perform an occlusion test and compositing operation as explained before in Section 4.5 and Section 5.4.

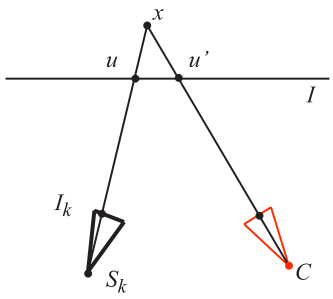

Figure 8.3: Pixel reprojection and optical flow on the virtual image plane. The pixel at position u is reprojected from the sensor's image plane I_k to the Lumigraph image plane I. By knowing the pixel's depth, we can reconstruct the 3-D point X, and then project from there onto I. The difference $(u' - u)$ on I is called the *optical flow* of that pixel for the view C.

8.6 ROI Partitioning

For reconstructing an arbitrary view C from the Lumigraph, it is imperative to determine which sensors should contribute to which part of the reconstructed image. These *regions of interest* (ROI) define which data needs to be transmitted from the sensor nodes to the display node, and the ROI supports speeding up (reducing) the processing on the sensor node.

The ROI concept corresponds exactly to the idea of the source regions presented in Section 5.3. As already discussed there, the partitioning is done by projecting the eyepoint mesh onto the virtual image plane I and growing the resulting triangle fans around each camera vertex by some factor that is the product of the triangle fan size and an upper bound of the pixel flow on I into the desired target region.

In the case of static two-plane Lumigraphs (Chapter 5), the optical flow and ROI was conservatively estimated using global bounds on the scene's depth. However, since the depth range varies with time, and the sensor might not provide an inexpensive way of querying this bound, we choose a very simple and convenient approach. For a sensor S_k we initially extend the target fan F_k by some large offset R_0, and use this as the ROI. During the reconstruction of the first view, we warp all the pixels from the ROI into our desired view. Now we determine the maximal optical flow R_k that occurred during this warping phase, and use R_k for the reconstruction of the next view. We use a minimum value for R in order to prevent the actual flow from getting away undetected. If we assume the changes in the scene to be smooth, this method is sufficient for detecting all necessary sensor samples. If some application requires a truly conservative estimate for the ROI, the maximal pixel flow on I must be derived from the depth value bounds (or disparity bounds) of the current sensor data or by adaptive subdivision, as sketched in Section 7.4.2. However, adaptive subdivision is not very well suited since the adaptive and recursive nature of the algorithm would induce some communication overhead

(many small queries from the display node to the sensor nodes and back).

8.7 The Lumi-Shelf

As a low-cost prototype for a complete real-world Instant Lumigraph system, we built a device we call the *Lumi-Shelf*, since it is basically a bookshelf with two rows of cameras (see Figure 8.7). We use six consumer quality Firewire video cameras aligned in two rows. We partition the cameras into stereo pairs, and every stereo pair is connected to one sensor PC. Each sensor PC does the following:

1. Rectify the two images using a lookup table. The result of rectification is that the scan lines of the two rectified images are aligned to the baseline between the two cameras [31].

2. Match corresponding pixels in the two images. Using a custom hierarchical block-based depth-from-stereo algorithm [23, 5], pixels in the left image are searched in the right image, within a certain search range in the same scan line.

3. Check stereo constraints such as left-right consistency in order to filter out the most obvious false matches.

4. Compute each pixel's depth from the detected disparity value.

5. Transmit color and depth information for the ROI to the display node.

In addition to the above-mentioned steps, the sensor part can make more efficient use of the ROI information by concentrating the search on the ROI, and one could use segmentation techniques like background subtraction [130, 8] in order to further speed up the matching process and to improve the matching quality near the boundaries. Our current implementation of the hierarchical scheme tries to trade quality for speed, so that currently the algorithm yields a dense range map of rather low quality at near-interactive rates (0.5-3 frames/second) for a resolution of 320×240 pixels.

As already stated in Section 8.1, we hope that camera systems with additional range sensors will soon emerge on the consumer market, so that it is not necessary to rely on pure software-based solutions. Simple depth-from-stereo matching has several disadvantages, such as the assumption of diffuse objects and the inability to reconstruct large texture-less regions.

8.8 Results

We have implemented the proposed system and tested it on off-the-shelf dual-Pentium III Linux PCs running at 800 MHz and equipped with nVIDIA GeForce graphics cards. We used the system to render all combinations of static vs. time-varying and synthetic vs. real-world data.

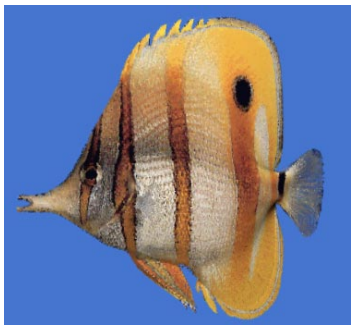

Figure 8.4: A synthetic fish, reconstructed from nine images with 512×486 pixels each, at $1.5 - 2.5$ fps.

Figure 8.5: Synthetic view of a real-world wooden elk reconstructed using a voxel coloring technique. The images consist of 760×502 pixels, and the display runs at 1 fps.

Figure 8.6: Some images from an "immersive movie". A talking beetle animation has been rendered and recorded from nine different viewpoints in a resolution of 512×486. The viewer can move freely while the Lumigraph movie is running over the network.

Figure 8.7: Left: Our Lumi-Shelf, consisting of a bookshelf, six consumer-quality Firewire video cameras connected to three Linux PCs, and one additional PC for rendering. Right: Some shots from an interactive Lumi-Shelf session, where the scene in front of the shelf can be rendered at near-interactive rates.

8.8.1 Static Lumigraphs / Rendering Performance

The single-processor version of the Lumigraph renderer (no sensor processing and network transfer) typically processes and displays images of 256×256 pixels at 9 frames/sec. Figure 8.4 shows the Lumigraph reconstruction of a synthetic fish that has been rendered from nine viewpoints using 3DStudioMax in a resolution of 512×486 pixels. Figure 8.5 shows a reconstruction of a real-world wooden elk. The depth values for the object have been reconstructed using a voxel coloring technique in a preprocessing step.

8.8.2 Lumigraphs from Animations

For demonstrating the performance of the technique on time-varying imagery with high-quality depth values, we rendered an animation sequence of a talking beetle using the 3DStudioMax software in conjunction with a plug-in for storing depth values along with the color information. The 70 frames of the animation were rendered from nine different viewpoints with an image resolution of 512×486 pixels.

We use a playback program that reads the image and depth files from file one by one and sends them over a network connection to a destination machine. We distributed the nine players plus the display process on three PCs connected via a 100MBit Ethernet LAN. The result (cf. Figure 8.6) is an “immersive movie” playing at roughly 6–8 frames/second, where the user can move freely within the scene while the movie is playing.

Of course for real movie applications one would have to consider more efficient and compact encoding and decoding of the image data. If this encoding also allows to extract only the ROI from the movie stream, it might well be possible to run all playback processes and the display process on the same machine while maintaining interactive rates.

8.8.3 Experiments with the Lumi-Shelf

Finally, we have experimented with our Lumi-Shelf prototype for demonstrating the processing of live real-world video data. As shown in the snapshots in Figure 8.7, it is possible to render arbitrary views of the real-world scene at frame rates that are limited mostly by the software depth-from-stereo implementation. The quality of the resulting views is much worse than that of the rendered animations, due to the noise in the color images as well as due to large numbers of false matches in the depth reconstruction.

The problem with our implementation is that we would need to spend even more computation time in order to identify more false matches. Another problem is that simple depth-from-stereo approaches are in principle not capable of unambiguously finding all correspondences. For correct depth reconstruction, techniques with more than two views must be used, such as multi-baseline stereo [110], camera matrix stereo [64], or a combination of shape-from-silhouettes [90, 89] and depth-from-stereo. Actually, first approaches to this idea have been presented recently as a follow-up project of this work [76].

However, in the resulting images one can clearly see the structure of the scene, and a viewer can move interactively around a person waving its hand or playing with a ball. By computing two separate views for both eyes, the system could easily be extended to stereo vision, without significantly increasing the overall computation time. In that case much of the reconstructed and transmitted data would also be used for the second very similar view, and only the actual rendering would be performed twice.

8.9 Discussion

We have developed the Instant Lumigraph framework, an architecture for immediate Lumigraph processing for time-varying scenes. This architecture is based on the single-slab free-form Lumigraph, a special variant of the free-form light field introduced in the previous chapter. Our framework is capable of working with any kind of sensor that captures images and depth, and it performs well even with a large number of sensors. We have proposed a distributed computation scheme and have taken care of the bandwidth requirements by a region-of-interest mechanism that is similar to the partitioning scheme presented in Chapter 5.

Furthermore, we have built up and experimented with the Lumi-Shelf, an early prototype of a complete tele-immersion system based on the proposed framework. Our experiments have shown that our approach is promising, even though the reconstruction of depth values needs to be improved in order to get satisfactory results for interactive real-world viewing.

The most important point about our work is that the presented framework is open for any kind of scene and sensor. Since the underlying representation is a Lumigraph with per-pixel depth information, the rendering only relies on a sampled representation of color and geometry and does not impose any constraints on the scene geometry or material (e.g. diffuse surfaces). Of course this is partially compensated by the sensors that are used for geometry reconstruc-

tion, since approaches such as correspondence matching assume diffuse surfaces in order to cross-correlate features in image pairs. However, with the framework presented here one can use and exploit all kinds of sensors that are and will be developed (e.g. laser range scanners synchronized with cameras). With the "perfect" sensors at hand, the Instant Lumigraph will enable interactive immersive viewing of time-varying, geometrically and photometrically complex remote scenes.

In the future, it will be important to incorporate the results of the previous chapter on free-form light fields, and thereby allow full all-around viewing of a scene. To this end, it is necessary to overcome the problems discussed in Section 8.5, which should be feasible by adding resolution-dependent reconstruction techniques and by a very efficient incremental warping implementation. Furthermore, it seems very promising to combine our technique with some kind of efficient multi-view video compression in order to reduce memory and network transfer costs. One could further think of extending the technique to time-varying camera parameters, e.g. for moving or zooming cameras.

CHAPTER 9

Discussion and Conclusions

All that we see or seem
is but a dream within a dream.

Edgar Allan Poe, from the poem *A Dream Within A Dream*

In this thesis we have presented a number of different contributions in the area of light field representation. First, we have taken the reader on a tour of image-based rendering and compared the different approaches with respect to different important criteria. After further presenting the known work on light fields in more detail, we have introduced and discussed an alternative Lumigraph representation based on images with per-pixel depth information. Although previously this structure was only used in other contexts of IBR (e.g. view interpolation, layered depth images), we have shown how to use per-pixel depth and 3D warping efficiently for rendering structured Lumigraphs, and have found a way to restrict computation to only the required regions of the reference images through a partitioning scheme inspired by the texture-based Lumigraph rendering algorithm. The big advantage of our proposed rendering technique, in contrast to the adaptive depth correction approach, is that it generates maximally sharp images at a high and constant frame rate, and does not require any scene-dependent fine-tuning.

We have further exploited 3D warping for predicting reconstruction error in Lumigraph data, and used this principle to drive an adaptive acquisition process that can be used for the automatic generation of Lumigraphs from synthetic scenes, e.g. for efficient display of simulation results. This technique is one of the very few that takes into account both geometric and photometric properties of the scene in order to predict further views necessary to sample the scene adequately. The results also show the versatility of the warping-based approach, which can be extended in a much more flexible way than highly specialized algorithms relying on the hardware texture processing pipeline.

Additionally, we have introduced the concept of a convex *free form parameterization*, which defines the most flexible parameterization for a *structured* light field (meaning that it comes with a view-independent triangulation of eyepoints). At the same time, Buehler et al. [9] have introduced *unstructured* Lumigraph rendering as described in Chapter 3. The structured approach allows a slightly more efficient rendering and a theoretically more consistent reconstruction, whereas the unstructured approach brings greater flexibility at the cost of more heuristic parameters that need to be fine-tuned to achieve good reconstruction quality.

Last, but not least, we have demonstrated a complete prototype of the *Lumi-shelf*, a system for instantaneous Lumigraph capture, transmission, and rendering. Together with the polyhedral visual hull approach by Matusik et al. [89], the system is the first that can handle Lumigraphs of dynamic scenes. While Matusik et al. use unstructured Lumigraph rendering for the display part, we employ free-form Lumigraph rendering. The major difference is the way and level of real-time geometry acquisition. While we try to reconstruct dense depth maps from pairs of input views, they reconstruct a very coarse polyhedral model from silhouette information. While our approach suffers from the missing robustness of stereo reconstruction, their approach gives a relatively robust, but very coarse approximation of the geometry. Again, it is an open question for the future how the strengths of these two approaches can be combined.

Pixels vs. Polygons? The comparison with other approaches such as image-based visual hull and polygon-based Lumigraph rendering leads to a more general discussion that touches the foundations of image-based rendering and computer graphics as a whole: where is the right tradeoff between image data and geometric information, and what is the right way of representing both in a consistent, compact, and efficient manner? Researchers are speaking of the "image-geometry continuum" and are raising challenges of "pixels" vs. "polygons". However, a useful and practical comparison is very hard to do since many different aspects of acquisition, representation, and rendering must be taken into account for each specific application.

Nevertheless, some experience can be reported, illuminating at least some of these aspects. The principle advantage of a purely image-based approach is that it can represent a scene without any knowledge of its contents, i.e. without prior construction of a model. However, images are a sampled representation, and thus the fundamental principles of sampling theory must be applied, e.g. the sampling density must be adapted to the characteristics of the scene, which brings us back to its geometric and photometric properties. Although they do not need to be modeled explicitly, at least their characteristics need to be known in advance, e.g. the

material with the sharpest reflection characteristics, and the size of the smallest geometric feature in the scene. Using these "extreme" characteristics as an estimation for the whole scene will ensure a sampling rate that is sufficiently high. However this approach is bound to be intractable in terms of memory and computation time. So the only viable way of faithfully representing a complex scene is by adapting the representation locally to the scene, and to do this in all dimensions of the representation (see also Section 3.3, coding and compression). This further requires the scene's features to be analyzed to some extent, somewhat similar to our adaptive acquisition approach.

Once this has been achieved, there is still the question of how to represent the different modalities of the data. Should we store a depth value with each pixel, or rather generate a global geometry model? This question does not seem so important, since in principle one representation can be converted into the other. However, if we *start* with sampled data, it might be a good choice to leave it in this form instead of reducing and approximating it further. On the other hand, in practice the sampling rate is usually not sufficient, or the cameras are not ideally calibrated, and so the sampled representation is not consistent. This can give rise to a number of problems, e.g. blurring and ghosting, or pixelization and popping effects.

So if we can afford it in terms of reconstruction effort, a polygonal model seems to be the more reliable choice at second glance. However, it has always been stated that the biggest advantage of image-based representations is their independence of the scene‘s geometric complexity. So most hybrid image-based rendering algorithms only assume a simple polygonal approximation of the scene. It is obvious that in some cases this approach is bound to fail, e.g. if this geometric proxy and the real geometry differ too much. Furthermore, it is not trivial to construct a consistent and truthful polygonal model from a set of range images, or even by merging multiple partial models. In the context of 3D scanning, this step usually involves manual labor that cannot be replaced by fully automatic algorithms so far.

The most popular example demonstrating the limits of all current algorithms is that of fur and hair. Just interpolating views of such an object will give very blurry results (in the best case). Reconstructing a polygonal model of a hairy beast from scan data is impossible (except if ignoring the hair altogether). Using sampled per-pixel depth values would be possible (if the scanner is able to capture these fine structures at all), but would lead to an inconsistent reconstruction (one sample from a certain direction hits an individual string of hair, another sample looking from a different direction does not).

Applications-specific solutions. Despite these problems of data representation, one fundamental truth remains: the best solutions are usually application-specific. For a few problems an optimal general solution can be found. But there is a continuum of different solutions with their advantages and disadvantages, and each specific application comes with a number of constraints that can be exploited to yield a novel and better solution. So one very important point for the young area of image-based rendering is to look for specific applications which will drive this field further and help to identify the actual strengths of the image-based approach. Potential examples for this might be virtual exhibitions, virtual cultural heritage projects, immersive

telecommunication and tele-presence, games that use augmented reality, and, of course, further visual effects for the media, as the one sketched in the very beginning of this thesis.

Facing difficult problems and challenging questions, we believe that image-based rendering, and especially light field representation, is a very exciting and fruitful area of research. As we have seen, many interesting projects have already been implemented, and certainly image-based rendering will continue to play a fundamental role in transporting or enhancing realism in digital imagery.

References

[1] E.H. Adelson and J.R. Bergen. Chapter 1 (the plenoptic function and the elements of early vision). In *Computational Models of Visual Processing*. MIT Press, Cambridge, MA, 1991.

[2] Shai Avidan and Amnon Shashua. Novel view synthesis by cascading trilinear tensors. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):293–306, October - December 1998. ISSN 1077-2626.

[3] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Computer Graphics (Proc. SIGGRAPH 1992)*, pages 35–42. ACM Press / Addison Wesley, 1992. ISBN 0-201-51585-7.

[4] Ryad Benosman and Sing Bing Kang. *Panoramic Vision: Sensors, Theory, and Applications*. Springer, 2001.

[5] M. Bierling. Displacement estimation by hierarchical blockmatching. In *SPIE Visual Communications and Image Processing, Vol. 1001*, pages 942–951. 1988.

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

[7] J.-Y. Bouguet, B. Curless, Paul Debevec, Marc Levoy, Shree Nayar, and Steve Seitz. Siggraph 2000 course on 3D photography. `www-2.cs.cmu.edu/~seitz/course/3DPhoto.html`, 2000.

[8] Al Bovik, editor. *Handbook of Image and Video Processing*. Academic Press, 2000.

[9] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In *Computer Graphics (Proc. SIGGRAPH 2001)*, pages 425–432. ACM Press / Addison Wesley, 2001.

[10] Brian Cabral, Marc Olano, and Philip Nemec. Reflection space image based rendering. In Alyn Rockwood, editor, *Computer Graphics (Proc. SIGGRAPH 1999)*, pages 165–170. ACM Press / Addison Wesley, 1999.

[11] E. Camahort and D. Fussell. A geometric study of light field representations. Technical Report TR99-35, Department of Computer Sciences, University of Texas, 1999.

[12] E. Camahort, A. Lerios, and D. Fussell. Uniformly sampled light fields. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering)*, pages 117–130. Springer, 1998.

[13] E. Catmull. A subdivision algorithm for computer display of curved surfaces. Technical Report UTEC-CSc-74-133, Salt Lake City, 1974.

[14] J. Chai, X. Tong, S. Chan, and H. Shum. Plenoptic sampling. In Kurt Akeley, editor, *Proc. SIGGRAPH 2000*, pages 307–318, 2000.

[15] C.-F. Chang, G. Bishop, and A. Lastra. LDI tree: A hierarchical representation for image-based rendering. In Alyn Rockwood, editor, *Computer Graphics (Proc. SIGGRAPH 1999)*, pages 291–298. ACM Press / Addison Wesley, 1999.

[16] S. E. Chen and L. Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (Proc. SIGGRAPH 1993)*, pages 279–288. ACM Press / Addison Wesley, 1993.

[17] Shenchang Eric Chen. QuickTime VR — an image-based approach to virtual environment navigation. In Robert Cook, editor, *Computer Graphics (Proc. SIGGRAPH 1995)*, pages 29–38. ACM Press / Addison Wesley, 1995.

[18] William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. The delta tree: An object-centered approach to image-based rendering. Technical Report AIM-1604, MIT, 1996.

[19] P. Debevec, G. Borshukov, and Y. Yu. Efficient view-dependent image-based rendering with projective texture mapping. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering)*. Springer, 1998.

[20] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *Computer Graphics (Proc. SIGGRAPH 1997)*, pages 369–378. ACM Press / Addison Wesley, 1997.

[21] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, pages 11–20. ACM Press / Addison Wesley, 1996.

[22] Xavier Décoret, Gernot Schaufler, François Sillion, and Julie Dorsey. Multi-layered impostors for accelerated rendering. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum 18(3) (Proc. EUROGRAPHICS 1999)*. Blackwell, 1999.

[23] U.R. Dhond and J.K. Aggarwal. Structure from stereo-a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, 1989.

[24] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[25] T. Kanade et al. Development of a video rate stereo machine. In *Proc. International Robotics and Systems Conference (IROS-95)*, Pittsburgh, PA, 1995.

[26] Olivier Faugeras and Luc Robert. What can two images tell us about a third one? *The International Journal of Computer Vision*, 18(1):5–20, 1996.

[27] Olivier D. Faugeras, Quang-Tuan Luong, and Stephen J. Maybank. Camera self-calibration: Theory and experiments. In *European Conference on Computer Vision*, pages 321–334, 1992.

[28] Cris A. Fitch. Background on the aspen movie map. `www.orbit6.com/crisf/aspen.htm`.

[29] James D. Foley, Andries can Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 2nd ed. edition, 1990.

[30] Henry Fuchs, Gary Bishop, Kevin Arthur, Leonard McMillan, Ruzena Bajcsy, Sang Lee, Hany Farid, and Takeo Kanade. Virtual space teleconferencing using a sea of cameras. In *Proc. First International Symposium on Medical Robotics and Computer Assisted Surgery*, Pittsburgh, PA, 1994.

[31] A. Fusiello, E. Trucco, and A. Verri. Rectification with unconstrained stereo geometry. In A. F. Clark, editor, *Proc. British Machine Vision Conference*, pages 400–409. BMVA Press, 1997.

[32] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[33] A. Gershun. The Light Field. *Journal of Mathematics and Physics*, 18:51–151, 1939. Original title “Svetovoe Pole”, published 1936 in Moscow.

[34] B. Girod, G. Greiner, and H. Niemann (eds.). *Principles of 3D Image Analysis and Synthesis*. Kluwer Academic Publishers, ISBN 0-7923-7850-4, 2000.

[35] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufman, 1995.

[36] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[37] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, pages 43–54. ACM Press / Addison Wesley, 1996.

[38] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[39] Xianfeng Gu, Steven J. Gortler, and Michael F. Cohen. Polyhedral geometry and the two-plane parameterization. In J. Dorsey and P. Slusallek, editors, *Rendering Techniques '97 (Proc. 8th Eurographics Workshop on Rendering)*, pages 1–12. Springer, 1997.

[40] Emilio Camahort Gurrea. *4D Light Field Modeling and Rendering*. Phd thesis, The University of Texas at Austin, 2001.

[41] Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. In *Rendering Techniques '93 (Proc. 4th Eurographics Workshop on Rendering)*, pages 259–266. Springer, June 1993.

[42] P. Hanrahan and J. Lawson. A language for shading and lighting calculations. In *Computer Graphics (Proc. SIGGRAPH 1990)*, pages 289–298. ACM Press / Addison Wesley, 1990.

[43] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, 1986. revised from Graphics Interface '86 version.

[44] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, CS Dept, UC Berkeley, May 1989.

[45] W. Heidrich, J. Kautz, P. Slusallek, and H.-P. Seidel. Canned lightsources. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering)*, pages 293–300. Springer, 1998.

[46] Wolfgang Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen, Computer Graphics Group, 1999.

[47] Wolfgang Heidrich. Interactive display of global illumination solutions for non-diffuse environments – a survey. In D. Duke and R. Scopigno, editors, *Computer Graphics Forum 20(3) (Proc. EUROGRAPHICS 2001)*, pages 225–244. Blackwell, 2001.

[48] Wolfgang Heidrich, Hendrik Lensch, and Hans-Peter Seidel. Light field-based reflections and refractions. In D. Lischinski and G.W. Larson, editors, *Rendering Techniques '99 (Proc. 10th Eurographics Workshop on Rendering)*. Springer, 1999.

[49] Wolfgang Heidrich, Hartmut Schirmacher, Hendrik Kück, and Hans-Peter Seidel. A warping-based refinement of Lumigraphs. In Vaclav Skala, editor, *Proc. 7th Int'l Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG '99)*, pages 102–109, Plzen, Chech Republic, February 1999. University of West Bohemia.

[50] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 39–46, Lisbon, Portugal, August 1998. ACM SIGGRAPH / Eurographics / ACM Press.

[51] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In Alyn Rockwood, editor, *Computer Graphics (Proc. SIGGRAPH 1999)*, pages 171–178. ACM Press / Addison Wesley, 1999.

[52] Wolfgang Heidrich, Rüdiger Westermann, Hans-Peter Seidel, and Thomas Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 127–134. ACM SIGGRAPH, April 1999.

[53] B. Heigl, M. Pollefeys, J. Denzler, and L. van Gool. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Proc. 21. Symposium für Mustererkennung (DAGM '99)*, pages 94–101, 1999.

[54] I. Ihm, S. Park, and R.K. Lee. Rendering of spherical light fields. In *Proc. Pacific Graphics '97*, 1997.

[55] A. Isaksen, L. McMillan, and S.J. Gortler. Dynamically reparameterized light fields. In Kurt Akeley, editor, *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 297–306. ACM Press / Addison Wesley, 2000.

[56] T. Kanade, P. Narayanan, and P. Rander. Virtualized reality: Concepts and early results. In *Proc. IEEE Workshop on Representation of Visual Scenes*, 1995.

[57] Takeo Kanade, Hideo Saito, and Sundar Vedula. The 3D room: Digitizing time-varying 3D events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.

[58] Sing Bing Kang. A survey of image-based rendering techniques. In *Videometrics VI (Proc. SPIE International Symposium on Electronic Imaging: Science and Technology)*, volume 3641, pages 2–16, San Jose, CA, January 1999. also DEC Cambridge Research Lab Technical Report CRL 97/4, 1997.

[59] Sing Bing Kang, Richard Szeliski, and P. Anandan. The geometry-image representation tradeoff for rendering. In *Proc. International Conference on Image Processing*, Vancouver, Canada, September 2000.

[60] Jan Kautz and Michael D. McCool. Approximation of glossy reflection with prefiltered environment maps. In *Graphics Interface 2000*, pages 119–126, May 2000.

[61] Jan Kautz, Pere-Pau Vazquez, Wolfgang Heidrich, and Hans-Peter Seidel. A unified approach to prefiltered environment maps. In B. Peroche and H. Rushmeier, editors,

Rendering Techniques '00 (Proc. 11th Eurographics Workshop on Rendering), pages 185–196. Springer, 2000.

[62] M. Kemp. *Leonardo on Painting.* Yale University Press, 1989. New Haven.

[63] M. Kiu, X. Du, R. Moorhead, D. Banks, and R. Machiraju. Two dimensional sequence compression using mpeg. In *SPIE Vol. 3309, SPIE/IS&T Electronic Imaging '97*, San Jose, CA, 1998.

[64] Yuichi Ohta Kiyohide Satoh, Itaru Kitahara. 3D image display with motion parallax by camera matrix stereo. In *Proc. 3rd Intl. Conf. on Multimedia Computing and Systems (ICMCS'96)*, 1996.

[65] Reinhard Kletter, Karsten Schlüns, and Andreas Koschan. *Computer Vision – Three Dimensional Data from Images*. Springer, 1998. ISBN 981-3083-71-9.

[66] Reinhard Koch, Marc Pollefeys, Benno Heigl, Luc J. Van Gool, and Heinrich Niemann. Calibration of hand-held camera sequences for plenoptic modeling. In *ICCV (1)*, pages 585–591, 1999.

[67] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[68] Paul Lalonde and Alain Fournier. Interactive rendering of wavelet projected light fields. In *Graphics Interface '99*, pages 107–114, June 1999. ISBN 1-55860-632-7.

[69] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[70] Stéphane Laveau and Olivier Faugeras. Representing three-dimensional data as a collection of images and fundamental matrices for image synthesis. In *Proc. ICPR94*, pages 689–691, 1994. also available as INRIA Reseach Report 2205.

[71] Seung-Yong Lee, Kyung-Yong Chwa, and Sung Yong Shin. Image metamorphosis using snakes and free-form deformations. In Robert Cook, editor, *Computer Graphics (Proc. SIGGRAPH 1995)*, pages 439–448. ACM Press / Addison Wesley, 1995.

[72] J. Lengyel. The convergence of graphics and vision. *IEEE Computer*, July 1998.

[73] Jed Lengyel and John Snyder. Rendering with coherent layers. In Turner Whitted, editor, *Computer Graphics (Proc. SIGGRAPH 1997)*, pages 233–242. ACM Press / Addison Wesley, 1997. ISBN 0-89791-896-7.

[74] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, pages 31–42. ACM Press / Addison Wesley, 1996.

[75] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In Kurt Akeley, editor, *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 131–144. ACM Press / Addison Wesley, 2000.

[76] Ming Li, Hartmut Schirmacher, Marcus Magnor, and Hans-Peter Seidel. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Proc. IEEE 2002 Workshop on Multimedia Signal Processing*, St. Thomas, US Virgin Islands, December 2002.

[77] A. Lippman. Movie-maps: An application of the optical videodisc to computer graphics. In *Computer Graphics (Proc. SIGGRAPH 1980)*, pages 32–42. ACM Press / Addison Wesley, 1980.

[78] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering)*, pages 301–314. Springer, 1998.

[79] M.E. Lukacs. Predictive coding of multi-viewpoint image sets. *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'86)*, Tokio, Japan, pages 521–524, October 1986.

[80] Reto Lütolf, Bernd Schiele, and Markus Gross. The light field oracle. In *Proc. Pacific Graphics 2002*, Tsinghua University, Beijing, China, October 9–11 2002.

[81] M. Magnor, A. Endmann, and B. Girod. Progressive compression and rendering of light fields. *Proc. Vision, Modeling, and Visualization (VMV 2000)*, Saarbrücken, Germany, pages 199–203, November 2000.

[82] M. Magnor and B. Girod. Adaptive block-based light field coding. *Proc. International Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging (IWSNHC3DI'99)*, Santorini, Greece, pages 140–143, September 1999.

[83] M. Magnor and B. Girod. Hierarchical coding of light fields with disparity maps. *Proc. IEEE International Conference on Image Processing (ICIP'99)*, Kobe, Japan, 3:334–338, October 1999.

[84] M. Magnor and B. Girod. Data compression for light field rendering. *IEEE Trans. Circuits and Systems for Video Technology*, 10(3):338–343, April 2000.

[85] M. Magnor and W. Heidrich. Image-based rendering. In B. Girod, G. Greiner, and H. Niemann, editors, *Principles of 3D Image Analysis and Synthesis*, pages 232–241. Kluwer Academic Publishers, ISBN 0-7923-7850-4, 2000.

[86] Marcus Magnor. *Geometry-Adaptive Multi-View Coding Techniques for Image-based Rendering*. Shaker Verlag, Aachen, Germany, ISBN 3-8265-8315-9, 2001. Ph.D. Thesis, University Erlangen-Nuremberg, Germany, Nov. 2000.

[87] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3D warping. In *Symposium on Interactive 3D Graphics*, pages 7–16;180, 1997.

[88] Kresimir Matkovic, Laszlo Neumann, and Werner Purgathofer. A survey of tone mapping techniques. Technical Report TR-186-2-97-12, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 1997.

[89] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In S.J. Gortler, editor, *Rendering Techniques '01 (Proc. 12th Eurographics Workshop on Rendering)*, pages 115–126. Springer, 2001.

[90] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 369–374. ACM Press / Addison Wesley, 2000.

[91] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In X. Pueyo and P. Schröder, editors, *Rendering Techniques '96 (Proc. 7th Eurographics Workshop on Rendering)*, pages 165–174. Springer, 1996.

[92] Nelson Max, Oliver Deussen, and Brett Keating. Hierarchical image-based rendering using texture mapping hardware. In D. Lischinski and G.W. Larson, editors, *Rendering Techniques '99 (Proc. 10th Eurographics Workshop on Rendering)*, pages 57–62. Springer, 1999.

[93] Bruce. A. Maxwell. Teaching computer vision to computer scientists: issues and a comparative textbook review. *International Journal of Pattern Recognitiron and Artificial Intelligence*, 12(8):1035–1051, August 1998.

[94] Leonard McMillan. *An Image-based Approach to Three-Dimensional Computer Graphics*. Ph.d. thesis, University of North Carolina at Chapel Hill, 1997. also available as UNC Technical Report TR97-013, 1997.

[95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In Robert Cook, editor, *Computer Graphics (Proc. SIGGRAPH 1995)*, pages 39–46. ACM Press / Addison Wesley, 1995.

[96] Leonard McMillan and Steven Gortler. Image-based rendering: A new interface between computer vision and computer graphics. *ACM SIGGRAPH*, 33(4), 1999.

[97] G.S.P. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proc. 9th Eurographics Workshop on Rendering)*, pages 281–292. Springer, 1998.

[98] Heinrich Müller. Image-based rendering: A survey. In S. P. Mudur and Dinesh Shikhare, editors, *ICVC99 - International Conference on Visual Computing*, pages 136–143. Fontasey Typesetters Pvt. Ltd., 1999.

[99] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, Reading MA, 1993.

[100] Lars Nyland, David McAllister, Voicu Popescu, Chris McCue, Anselmo Lastra, Paul Rademacher, Manuel Oliveira, Gary Bishop, Gopi Meenakshisundaram, Matt Cutts, and Henry Fuchs. The impact of dense range data on computer graphics. In *Proc. Multi-View Modeling and Analysis Workshop (MVIEW99), Part of CVPR99*, June 23-26, 1999.

[101] Patrick Ohly. Extended light fields and their application to volume rendering. `irafs1.ira.uka.de/~patrick/lightfields/`, 1999.

[102] Manuel M. Oliveira and Gary Bishop. Image-based objects. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 191–198. ACM SIGGRAPH, April 1999. ISBN 1-58113-082-1.

[103] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1993. ISBN 0-521-44034-3.

[104] Ingmar Peter and Wolfgang Straßer. The wavelet stream: Progressive transmission of compressed light field data. In *IEEE Visualization 1999 Late Breaking Hot Topics*, pages 69–72. IEEE Computer Society, October 1999.

[105] Ingmar Peter and Wolfgang Straßer. The wavelet stream: Interactive multi resolution light field rendering. In S.J. Gortler, editor, *Rendering Techniques '01 (Proc. 12th Eurographics Workshop on Rendering)*. Springer, 2001.

[106] Paul Rademacher and Gary Bishop. Multiple-center-of-projection images. In Michael Cohen, editor, *Computer Graphics (Proc. SIGGRAPH 1998)*, pages 199–206. ACM Press / Addison Wesley, 1998. ISBN 0-89791-999-8.

[107] Peter W. Rander, PJ Narayanan, and Takeo Kanade. Virtualized reality: Constructing time-varying virtual worlds from real world events. In *IEEE Visualization '97*, pages 277–284. IEEE, November 1997.

[108] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In Michael Cohen, editor, *Computer Graphics (Proc. SIGGRAPH 1998)*, pages 179–188. ACM Press / Addison Wesley, 1998.

[109] John Rohlf and James Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In Andrew S. Glassner, editor, *Computer Graphics (Proc. SIGGRAPH 1994)*, pages 381–394. ACM Press / Addison Wesley, 1994.

[110] H. Saito, S. Baba, M. Kimura, S. Vedula, and T. Kanade. Appearance-based virtual view generation of temporally-varying events from multi-camera images in the 3D room. In *Proc. 2nd International Conference on 3-D Digital Imaging and Modeling*, pages 516 – 525, October 1999.

[111] Gernot Schaufler. Dynamically generated impostors. In D.W. Fellner, editor, *GI Workshop Modeling - Virtual Worlds - Distributed Graphics*, pages 129–135. infix, 1995.

[112] Hartmut Schirmacher, Wolfgang Heidrich, and Hans-Peter Seidel. Adaptive acquisition of Lumigraphs from synthetic scenes. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum 18(3) (Proc. EUROGRAPHICS 1999)*, pages C151–C160. Blackwell, 1999.

[113] Hartmut Schirmacher, Wolfgang Heidrich, and Hans-Peter Seidel. High-quality interactive Lumigraph rendering through warping. In Sidney Fels and Pierre Poulin, editors, *Proc. Graphics Interface 2000*, Montreal, Canada, May 2000. CHCCS, CHCCS.

[114] Hartmut Schirmacher, Li Ming, and Hans-Peter Seidel. On-the-fly processing of generalized Lumigraphs. In D. Duke and R. Scopigno, editors, *Computer Graphics Forum 20(3) (Proc. EUROGRAPHICS 2001)*, pages C165–C173;C543. Blackwell, 2001.

[115] Hartmut Schirmacher, Christian Vogelgsang, Hans-Peter Seidel, and Günther Greiner. Efficient free-form light field rendering. In *Proc. Vision, Modeling, and Visualization (VMV '01)*. infix, 2001.

[116] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 1.2), 1998.

[117] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (Proc. SIGGRAPH 1992)*, pages 249–252. ACM Press / Addison Wesley, 1992. original reference for projective texture mapping.

[118] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. CVPR*, 1997.

[119] Steven M. Seitz and Charles R. Dyer. View morphing. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, volume 30, pages 21–30. ACM Press / Addison Wesley, 1996.

[120] SGI. Pixel texture extension. Specification document, available from `www.opengl.org`, 1996.

[121] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In Michael Cohen, editor, *Computer Graphics (Proc. SIGGRAPH 1998)*, pages 231–242. ACM Press / Addison Wesley, 1998.

[122] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony D. DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, pages 75–82. ACM Press / Addison Wesley, 1996.

[123] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In Alyn Rockwood, editor, *Computer Graphics (Proc. SIGGRAPH 1999)*, pages 299–306. ACM Press / Addison Wesley, 1999.

[124] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum 16(3) (Proc. EUROGRAPHICS 1997)*, pages 207–218. Blackwell, 1997.

[125] P.-P. Sloan, M.F. Cohen, and S.J. Gortler. Time critical Lumigraph rendering. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 17–24. ACM SIGGRAPH, 1997.

[126] P.-P. Sloan and C. Hansen. Parallel Lumigraph reconstruction. In *Proc. Symposium on Parallel Visualization and Graphics*, pages 7–15. IEEE, 1999.

[127] Philipp Slusallek. *Vision: An Architecture for Physically-Based Rendering*. Ph.d. thesis, University of Erlangen, 1995.

[128] Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 1(58):23–32, 1993.

[129] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In Turner Whitted, editor, *Computer Graphics (Proc. SIGGRAPH 1997)*, pages 251–258. ACM Press / Addison Wesley, 1997.

[130] A. Murat Tekalp, editor. *Digital Video Processing*. Prentice Hall, 1995.

[131] Jay Torborg and Jim Kajiya. Talisman: Commodity Real-time 3D graphics for the PC. In Holly Rushmeier, editor, *Computer Graphics (Proc. SIGGRAPH 1996)*, pages 353–364. ACM Press / Addison Wesley, 1996.

[132] G. Tsang, S. Ghali, E.L. Fiume, and A.N. Venetsanopoulos. A novel parameterization of the light field. In *Image and Multidimensional Digital Signal Processing '98 (Proc. 10th IMDSP Workshop)*. infix, 1998.

[133] Carnegie Mellon University. Computer vision homepage. `www-2.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html`.

[134] Stanford University. Light field camera project. `www.graphics.stanford.edu/projects/lightfield/`.

[135] Pere-Pau Vazquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint selection using viewpoint entropy. In *Proc. Vision, Modelling and Visualization (VMV01)*, Stuttgart, Germany, 2001.

[136] C. Vogelgsang. lgf – an image-based scene description file format. Technical Report 10, IMMD 9, Universität Erlangen-Nürnberg, 1999.

[137] C. Vogelgsang and G. Greiner. Hardware accelerated light field rendering. Technical Report 11, IMMD 9, Universität Erlangen-Nürnberg, 1999.

[138] C. Vogelgsang and G. Greiner. Adaptive lumigraph rendering with depth maps. Technical Report 3, IMMD 9, Universität Erlangen-Nürnberg, 2000.

[139] C. Vogelgsang, B. Heigl, G. Greiner, and H. Niemann. Automatic image-based scene model acquisition and visualization. In *Proc. Vision, Modeling, and Visualization (VMV 2000)*, Saarbrücken, Germany, November 2000.

[140] Eric W. Weisstein. Barycentric coordinates. `mathworld.wolfram.com/BarycentricCoordinates.html`.

[141] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics (Proc. SIGGRAPH 1990)*, pages 367–376. ACM Press / Addison Wesley, 1990.

[142] L. Williams. Pyramidal parametrics. In *Computer Graphics (Proc. SIGGRAPH 1983)*, pages 1–11. ACM Press / Addison Wesley, 1983.

[143] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1990.

[144] Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, and Wai-Yin Ng. Image-based rendering with controllable illumination. In J. Dorsey and P. Slusallek, editors, *Rendering Techniques '97 (Proc. 8th Eurographics Workshop on Rendering)*, pages 13–22. Springer, 1997.

[145] Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer, and David H. Salesin. Multiperspective panoramas for cel animation. In Turner Whitted, editor, *Computer Graphics (Proc. SIGGRAPH 1997)*, pages 243–250. ACM Press / Addison Wesley, 1997.

[146] D.N. Wood, D.I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In Kurt Akeley, editor, *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 287–296. ACM Press / Addison Wesley, 2000.

[147] J.C. Yang and L. McMillan. Light fields on the cheap. SIGGRAPH 2000 technical sketch.

[148] Denis Zorin, Peter Schröder, Tone DeRose, and Leif Kobbelt. Siggraph 2000 course: Subdivision for modeling and animation. `http://mrl.nyu.edu/publications/subdiv-course2000`.

[149] Matthias Zwicker, Markus H. Gross, and Hanspeter Pfister. A survey and classification of real time rendering methods. Technical Report 2000-09, Mitsubishi Electric Research Laboriatories, 2000.

Abstract

Light fields belong to the research field of image-based rendering. The light field has been introduced to computer graphics in 1996 and is an approach to represent a three-dimensional scene by a number of images, where each image is a different view of the same scene. By using specific interpolation methods, arbitrary views of the scene can be reconstructed from these reference views at interactive frame rates.

This work starts out with a comprehensive survey on previous work in the young field of image-based rendering. We classify the methods according to principal approaches, briefly sketch each method, and compare them using different important criteria. We discuss the pros and cons of the different approaches as well as the most important open questions. Then we describe in more detail the previous work on light fields, building the foundation for our own work.

First we propose a light-field based scene representation that uses image data as well as some approximate geometry information in the form of depth values. Using two different techniques we show how to exploit this hybrid representation for reconstructing arbitrary views in high quality from a relatively small number of reference views. In order to do so, we develop specialized variants of the 3D warping formulae for reprojecting a pixel from a 2D image into another image using the associated depth value. We exploit the special setup of the images in a light field in order to make the warping algorithms simpler and more efficient. Due to a partitioning scheme inspired by texture-based light field rendering, we ensure that only those parts of the image data that potentially contribute to the final view will actually be reprojected. In contrast to other light field rendering algorithms, our technique maximally exploits the stored geometric information and does not require any fine tuning of algorithmic parameters to each individual scene.

Besides the rendering of light fields, warping-based approaches can also be used for the acquisition process. We develop an adaptive technique for generating light fields from synthetic scenes. Using warping, we can use the current set of reference images for estimating a priori which potential new view will contribute most information to the current light field. This estimate is based on color deviation of corresponding pixels and also takes into account pixels that are under-sampled, e.g. due to occlusion or large angular deviation in the different views. Using this estimation, the adaptive acquisition algorithm can determine which potential new view would contain the biggest error, and can compensate for this by adding the corresponding reference view to the light field.

When dealing with light fields of real scenes it can be very beneficial to use the cameras' original image data of the scene directly for the reconstruction of novel views. For classical two-plane light field rendering, this is in general not possible since the cameras' optical centers are required to lie in a common plane. Alternative methods require to resample and reproject the data into the desired representation in a costly preprocessing step. This is why we present a new free-form parameterization for light fields, which allows to position the cameras on an

arbitrary convex surface. The free-form parameterization is the most flexible parameterization that still allows for computing a consistent topological ordering of the reference views, and thus leads to simple, robust, and very efficient rendering techniques.

Finally we present the prototype of a system called Lumi-Shelf. It combines the techniques that we have developed and is capable of acquiring a light field of a dynamic scene using multiple video cameras, transmitting the necessary image data via a network, and rendering arbitrary views of the captured scene instantaneously. The system uses a light field parameterization similar to the free-form parameterization, which allows for direct processing of the video images. Depth values are reconstructed from pairs of reference views by means of a depth-from-stereo algorithm. The above-mentioned light field partitioning scheme helps to identify the parts of the image data that are required to reconstruct the desired view. Only that part of the data (image and depth) is transmitted over the network. Finally the data is reprojected and combined in order to yield the desired image. We show the flexibility of our approach by demonstrating our system with static and dynamic, synthetic as well as real scenes.

Last, we briefly summarize our different contributions and discuss the results of our work, especially to the end of finding suitable hybrid representations for image and geometry data.

Zusammenfassung

[Deutsche Übersetzung des Titles: *Effiziente Akquisition, Repräsention und Bildsynthese von Lichtfeldern*]

Lichtfelder fallen in das Forschungsgebiet des sogenannten Image-based Rendering (deutsch: bildbasierte Bildsynthese). Das Lichtfeld hielt 1996 Einzug in die Computergrafik und bezeichnet einen Ansatz, bei dem eine dreidimensionale Szene durch eine Anzahl von Bildern repräsentiert wird, die verschiedene Ansichten derselben Szene darstellen. Mit speziellen Interpolationsverfahren können aus diesen Referenzbildern beliebige andere Ansichten der Szene rekonstruiert werden, und dies mit einer Geschwindigkeit, die Interaktion mit der Szene zuläßt.

Diese Arbeit beginnt mit einem aktuellen Überblick der bisherigen Arbeiten in dem jungen Gebiet des Image-based Rendering. Wir klassifizieren die Verfahren nach verschiedenen Ansätzen, stellen sie im Überblick vor und vergleichen sie anhand wichtiger Kriterien. Eine zusammenfassende Diskussion erläutert noch einmal die Vor- und Nachteile der verschiedenen Verfahren sowie die wichtigsten offenen Fragen. Anschließend beschreiben wir weitere Details der verschiedenen vorangegangenen Arbeiten auf dem Gebiet der Lichtfelder, die die Grundlage für die eigenen Arbeiten bilden.

Zunächst schlagen wir eine Lichtfeld-basierte Szenenrepräsentation vor, die neben den reinen Bilddaten auch approximative Geometriedaten in der Form von Tiefenwerten verwendet. Wir zeigen anhand zweier verschiedener Techniken, wie wir dank der hybriden Darstellung auch mit relativ wenigen Referenzbildern sehr gute Rekonstruktionen beliebiger Ansichten erzielen. Hierzu entwickeln wir spezialisierte Varianten des 3D-Warping-Ansatzes, mit welchem ein 2D-Bildpunkt durch den zugehörigen Tiefenwert in ein beliebiges anderes Bild hinein projeziert werden kann. Dabei nutzen wir die spezielle Anordnung der Bilder in einem Lichtfeld aus, um die Warping-Algorithmen zu vereinfachen und zu beschleunigen. Ein von texturbasiertem Lichtfeld-Rendering inspiriertes Partitionierungsschema sorgt dafür, daß lediglich diejenigen Bilddaten reprojeziert werden müssen, die auch potentiell zu der gewünschten Ansicht der Szene beitragen. Im Gegensatz zu anderen Lichtfeld-Darstellungsverfahren nutzt die von uns vorgeschlagene Methode die vorhandene Geometrie-Information optimal aus und erfordert keinerlei Feinanpassung algorithmischer Parameter an die jeweilige Szene.

Neben der Darstellung von Lichtfeldern können Warping-basierte Verfahren auch bei der Akquisition eingesetzt werden. Wir entwickeln ein adaptives Verfahren zur Generierung von Lichtfeldern von synthetischen Szenen. Mittels Warping wird aus den bereits akquirierten Referenzbildern der Szene a priori abgeschätzt, welche potentielle neue Referenzansicht die meiste fehlende Information zu dem aktuellen Lichtfeld beitragen würde. Diese Abschätzung basiert auf der Farbabweichung korrespondierender Bildpunkte und bezieht auch Bildpunkte ein, die in den Referenzbildern unterrepräsentiert sind, wie z.B. im Fall von Verdeckungen oder großen Winkeländerungen. Durch diese Abschätzung kann das adaptive Akquisitionsverfahren

ermitteln, wo in dem aktuellen Lichtfeld bei der Darstellung neuer (nicht im Lichtfeld gespeicherter) Ansichten die größten Fehler auftreten würden, und kann dies durch Hinzufügen der entsprechenden Originalansichten kompensieren.

Beim Umgang mit Lichtfeldern von realen Szenen ist es vorteilhaft, wenn die Original-Bilddaten der Kameras möglichst direkt zur Rekonstruktion beliebiger Ansichten verwendet werden können. Bei einer klassichen Lichtfeld-Parametrisierung mittels paralleler Ebenen ist dies i.a. jedoch nicht möglich, da die optischen Zentren aller Kameras in einer gemeinsamen Ebene liegen müssen. In alternativen Verfahren müssen die Bilddaten in einem aufwendigen Vorverarbeitungsschritt neu abgetastet und in die gewünschte Darstellung reprojeziert werden. Deshalb stellen wir die sogenannte Freiform-Parametrisierung für Lichtfelder vor, welche es erlaubt, die Kameras auf einer beliebigen konvexen Fläche zu positionieren. Die Freiform-Parametrisierung ist die flexibelste Parametrisierung, die es noch erlaubt, eine konsistente topologische Einordnung der Referenzansichten zu berechnen, und führt so zu einfachen, zuverlässigen und sehr effizienten Rendering-Verfahren.

Schließlich stellen wir den Prototyp eines Systems namens Lumi-Shelf vor, der die entwickelten Ansätze vereint und es ermöglicht, ein Lichtfeld einer dynamischen Szene mittels mehrerer Videokameras zu akquirieren, die notwendigen Bilddaten über ein Netzwerk zu übertragen, und eine beliebige Ansicht der Szene darzustellen. Das System verwendet eine Freiform-ähnliche Parametrisierung des Lichtfeldes, die es ermöglicht, die Bilder der Videokameras direkt zu verarbeiten. Mittels eines Depth-from-Stereo-Algorithmus werden aus je zwei Ansichten Tiefenwerte für die Bildpunkte rekonstruiert. Durch das bereits erwähnte Lichtfeld-Partitionierungsschema ist es möglich, die für die Darstellung der gewünschten Ansicht notwendigen Bilddaten zu identifizieren und nur diese (sowie die zugehörigen Tiefenwerte) über das Netzwerk zu übertragen. Im Rechner des Betrachters schließlich werden die übertragenen Bildpunkte rückprojeziert und kombiniert, um das gewünschte Bild zu erzeugen. Wir zeigen die Flexibilität unseres Ansatzes, indem wir das System für statische und dynamische, synthetische und reale Szenen demonstrieren.

Abschließend fassen wir unsere verschiedenen Beiträge noch einmal zusammen und diskutieren die im Rahmen der Arbeit ermittelten Ergebnisse, vor allem im Hinblick auf die Wahl einer geeigneten hybriden Repräsentation für Bild- und Geometriedaten.