Interactive Light Field Editing and Compositing

Billy Chen*  Daniel Horn*  Gernot Ziegler †  Hendrik P. A. Lensch*
*Stanford University  † MPI Informatik

Figure 1: Our system enables a user to interactively edit and composite light fields. This figure shows some of the operations that can be accomplished by the system. In (a) we create a light field by compositing multiple copies of a giraffe and flower light field. Each copy is twisted or turned differently to create a visually interesting scene. (b) is an image where objects have been selectively focused or defocused to create an artistic effect. Even though objects are defocused, occlusions between them are maintained and defocus blur blends with objects hidden behind occluders. This effect cannot be accomplished with a single image of each object. Image (c) shows light fields of refractive objects (glass balls) composited with a background light field. Because the background is a light field, the refracted image exhibits parallax and visibility change in different views. Note that we have chosen to focus on the virtual image in the refracted glass ball; consequently, the real image of the background is defocused. In (d) we approximate soft shadows cast from the giraffe light field onto the flower light field. Notice the penumbra of the shadow cast on the ground due to the light and notice that the shadow interacts with the flower believably.

Abstract

Light fields can be used to represent an object’s appearance with a high degree of realism. However, unlike their geometric counterparts, these image-based representations lack user control for editing and manipulating them. We present a system that allows a user to interactively edit and composite light fields. We introduce five basic operators, alpha-compositing, apply, multiplication, warping, and projection, which may be used in combination. To describe how multiple operators and light fields are combined together, we introduce a compositing language similar to those used for combining shader programs. The language provides a structured foundation to composite light fields. This language also serves as an interface between the user and the graphics hardware.

Keywords: light fields, image-based modeling and rendering, 3D photography

e-mail: {billyc|danielrh|lensch}@graphics.stanford.edu
†email: gziegler@mpi-sb.mpg.de

1 Introduction

A light field [Levoy and Hanrahan 1996; Gortler et al. 1996] is a four-dimensional function mapping rays in free space to radiance. Using light fields to represent scene information has become a popular technique for rendering photo-realistic images. Thus, the need arose to manipulate these datasets as flexibly as images and 3D models. In this paper, we present a system that allows a user to interactively edit and composite 4D light fields. Interactive rendering rates are achieved by exploiting programmable graphics hardware. Users can perform light field insertion, deformation and removal, refocusing, rapid-prototyping of photo-realistic scenes, or simulation of refractive effects in our framework.

To support these capabilities, we introduce five basic operators: alpha-compositing, apply, multiplication, warping, and projection. In combination, these operators enable a large number of ways to edit a light field. The operators are designed to be simple so that they can be implemented as basic functions on programmable graphics hardware. Complex operators can be created by combining several basic operators together.

To describe how this combining occurs, we introduce a compositing language, similar to those used to describe shaders [Cook 1984; Proudfoot et al. 2001] or pixel streams [Perlin 1985]. The language provides a structured foundation to composite light fields. This language also serves as an interface between the user and the graphics hardware.

We have developed a simple hardware-accelerated framework for representing light fields, their operators, and the data flow. The framework provides a high-level abstraction layer for manipulating light fields, rather than working directly at the pixel level. We use this framework to interactively edit and composite light fields.
The rest of the paper is organized as follows. First, we describe related work in the areas of compositing and image-based editing. Second, we present the basic operators and how to combine them. Third, we show how these operators can be implemented on programmable graphics hardware to enable interactive editing. Finally, we describe four useful editing operations using the basic operators and demonstrate their use on a variety of light fields.

2 Related work

The idea of compositing images was originally used in film and video production to simulate effects like placing an image of an actor in front of a different background [Fielding 1972]. In 1984, Porter and Duff introduced the alpha channel in order to composite digital images [Porter and Duff 1984]. Recently, Shum et al. extended 2D compositing to 4D light fields [Shum and Sun 2004]. They built a system that allows a user to manually segment a light field into layers. The system composites these layers together using a technique called coherence matting. Our framework offers a similar, but simpler compositing operator amongst a larger gamut of operators. Our focus is on simple operators that can be combined easily on the graphics hardware.

Compositing is one way to edit images. Many researchers have investigated other editing operations for either a single image or up to a handful of images. Oh et al. built a system that takes a single image as input and allows the user to augment it with depth information [Oh et al. 2001]. Subsequent editing operations, like cloning and filtering, can be performed with respect to scene depth. Barsky has used a similar image representation to simulate how a scene would be observed through a human optical system [Barsky 2004].

To handle multiple images of a scene, Seitz and Kutulakos developed a system that builds a voxel-representation [Seitz and Kutulakos 1998]. In this system, 2D editing operations, like scissoring or morphing, are propagated to the other images using the voxel representation. In our system, we use sufficiently sampled light fields as an alternative representation of the scene and our operators manipulate each light field individually.

For editing light fields, several researchers have built systems that take light fields as input, perform a specific editing task and return a new light field. Two such systems allow for warping light fields [Zhang et al. 2002; Chen et al. 2005]. In the former, two input light fields are combined to produce a morphed version. In the latter, an animator can interactively deform an object represented by a light field. Our system offers a general editing framework where warping is one of many operations.

The idea of combining operators to produce novel ones is inspired by early work on flexible shading models [Cook 1984]. Cook uses a shading tree to represent a wide range of shading characteristics. In our paper, we describe a language that allows light fields to be combined.

3 Basic light field operators

A light field is a four-dimensional function that takes a ray as input, and returns the radiance along that ray. We augment the radiance at each ray with a scalar alpha quantity that represents the opacity of the light field at that ray and call this the RGBA color.

Given this representation of a light field, we now define five basic operators that enable light field compositing and editing. The first four operators composite and manipulate light fields. The last operator computes 2D projections of light fields.

3.1 Basic operators

Notations and Conventions

Each of the following operators takes one or two light fields as input and returns a single light field. The exception is the projection operator, which returns a 2D image. When operating upon colors, we perform the same operation component-wise on each channel unless noted. To obtain the scalar opacity value from a color $c$ we write $c[a]$.

Alpha compositing

The alpha compositing operator enables correct depth compositing of multiple light fields in a scene. This operator can perform all twelve of Porter and Duff's [1984] compositing operations on 4D light fields\(^1\). Similar to their formulation, we also assume colors are premultiplied by the alpha channel.

Given a ray, $r$, we define the compositing operator $C$ as

$$C_{F_A,F_B}(L^A, L^B)(r) = L^A(r) \cdot F_A(L^B(r)) + L^B(r) \cdot F_B(L^A(r))$$

where $F_A(c)$ and $F_B(c)$ are functions that take in a color and return a scalar.

As a shorthand, when $F_A(c)$ or $F_B(c)$ are defined the following way:

$$F(c) = 0$$

$$F(c) = 1$$

$$F(c) = c[a]$$

$$F(c) = c[1 - a]$$

we give these functions the names 0, 1, $\alpha$, and $1 - \alpha$ respectively.

Thus the traditional over, addition and out operators can be represented as follows:

$$L^A \text{ over } L^B = C_{1, 1 - \alpha}(L^A, L^B)$$

$$L^A + L^B = C_{1, 1}(L^A, L^B)$$

$$L^A \text{ out } L^B = C_{1 - \alpha, 0}(L^A, L^B)$$

Other Binary Operators: apply and multiplication

While we can use the compositing operator above to define addition by setting blending fractions $F_A$ and $F_B$ to 1, compositing does not support arbitrary binary operations on colors. Thus we define the new operator apply, which takes two light fields and a binary function that maps colors to a color. Apply then returns a new light field where the binary function has been applied to colors of the corresponding rays in both light fields.

$$A_F(L^A, L^B)(r) = F(L^A(r), L^B(r))$$

We can then define light field multiplication by passing a binary function, color\_multiply, to apply. color\_multiply is a function that takes two colors and performs a component-wise multiplication in each color channel, including alpha.

$$L^A \times L^B = A_{\text{color\_multiply}}(L^A, L^B)$$

\(^1\)Our compositing operators can also be applied to 2D images since an image can be expressed as a simple 4D light field.
**Warping**

The warping operator deforms a light field function $L$ by transforming the directions of input rays. The warping operator takes two inputs, the original light field $L$ and any warping function $F$ that maps input rays to output rays. The warping operator is defined as:

$$ W(L, F)(r) = L(F(r)) $$

**Projection**

The task of the projection operator is to map a 4D light field to a 2D image for viewing.

The projection operator selects rays from the input light field and organizes them into an image. In our system we use a canonical pinhole camera model centered at $(0,0,0)$ and facing towards negative z with a field of view of 90 degrees and aspect ratio of 1. Given that $\mathbb{L}$ is the space of all 4D light field functions, and $I$ is the space of all images, then:

$$ P: \mathbb{L} \rightarrow I $$

Additionally, we provide an abstraction over our canonical $P$ that takes in the location of an arbitrary pinhole and the extent of an image plane and creates a projection from that new camera location. It accomplishes this by applying a warping function to the input light field to orient it to our underlying canonical projection operation. In our paper, we use $P_{h,c,r,u}$ to represent this abstraction. It denotes taking a 4D-to-2D pinhole projection at location $h$ on an image plane parameterized by its center, right, and up vectors $(c,r,u)$.

### 3.2 A light field compositing language

In order to describe how operators interact, we introduce a compositing language that combines operators into an expression similar to code in a shading language. For example, the following expression:

$$ P_{h,c,r,u}(L_{\text{windowframe}} \circ (L_{\text{glass}} \times (L_{\text{flower}} \circ L_{\text{table}}))) $$

describes a pinhole image of a light field created by alpha-compositing a light field of a flower with a table and viewing it through a window frame containing tinted glass. The pinhole camera is located at $h$ with an image plane whose center, right, and up vectors are $c$, $r$, and $u$, respectively.

### 4 Implementation on the GPU architecture

In order to keep the compositing process interactive, we make extensive use of the compute power and bandwidth available in modern programmable graphics hardware. Our framework provides a general interface for loading and manipulating 4D light fields on the GPU.

The framework consists of two parts: the data representation for light fields, and the functions that represent light field operators. To represent a light field, we use the two-plane parameterization [Levoy and Hanrahan 1996]. Hence, the light field is an array of images, where $(u,v)$ describes the 2D location of a pinhole camera in the $UV$-plane. The $(s,t)$ coordinates describe the location of a pixel in the $ST$-plane. All the camera images are aligned to the $ST$-plane. A ray is parameterized by two intersection points with the $UV$- and $ST$-plane. Because the light field is discretized, a color $L(r)$ and its alpha are interpolated from nearby samples using quadrilinear interpolation.

A light field using this representation is stored as a 3D texture on the graphics hardware. The $UV$ axes are vectorized into the $z$ axis in the 3D texture. The $ST$ axes map directly to each $xy$-planar slice in the 3D texture. In other words, the $z$ axis selects a camera in $(u,v)$ coordinates, and the $xy$-plane is the image plane in $(s,t)$ coordinates. The 3D texture is compressed using S3 texture compression [Iourcha et al.]. This extension provides a 4:1 compression ratio for RGBA data. The compression is key to being able to store multiple light fields in texture memory. Multiple light fields are each stored as separate 3D textures, each occupying one of the 16 available texture units. Multiple operator expressions can be associated to a single texture unit, allowing editing of multiple instances of a single light field without using additional texture units or memory.

The basic light field operators are represented as functions in a fragment program written in GLSL. To edit and composite light fields during runtime, the user specifies a GLSL expression using the defined basic functions. The user-definable functions for apply and composite are written directly in GLSL as functions mapping one or two colors to another color or scalar. The resulting fragment program is passed the virtual view, $P_{v,c,r,u}$ and the $UV$- and $ST$-plane locations for each light field. This fragment program is compiled and applied to the textures.

When the fragment program begins, the virtual view’s image plane $(c,r,u)$ is discretized into display pixels. As the fragment program is executed per display pixel, the pixel is converted into a ray using the location of the virtual view. This ray is passed into the user-supplied compositing expression to determine its RGBA color. A 2D image is formed when the fragment program has executed for all the display pixels.

### 5 Creating novel editing operators

We now describe four light field editing operators that are combinations of the basic operators described in Section 3. These operators are deformation, synthetic aperture focusing, refraction, and shadow casting.

#### 5.1 Light field deformation

Light field deformation consists of supplying a warping function for each light field and compositing the resulting light fields together. Chen et al. used free form deformation to specify the warping function [Chen et al. 2005]. In our system the user specifies a ray-to-ray warp as a function in the fragment shader. Figure 1a illustrates one example of a composition where two captured light fields were used to create a scene where a giraffe is surrounded by a grove of flowers.

To create this scene, let us begin with the captured datasets. Figure 2a is an image from a captured light field of a toy flower, after matting the flower from the background. The flower was acquired in front of a known background and we used a Bayesian matting technique [Chuang et al. 2001] to extract mattes for all images of the light field. The mattes are stored in the alpha component of the light field.

Let us call this light field $L_{\text{flower}}$ and place it at the origin of a coordinate system where the $xz$ plane is the ground plane and the $y$ axis is the up direction in the image. We now define a twisting function on this light field. Each ray in freespace is parameterized by two points, its intersection with the $UV$- and $ST$-planes of...
F creates a twisting effect on the light field, and we call this function $F_{twist}$. Therefore, to create an image with a shallow depth of field, we sum from the basic operators to perform focusing within a light field.

The pinhole cameras have centers that lie in the lens aperture. Their fields of view are shown as black points within the lens. Their fields of view are shown as black lines. The dashed lines show the distortion of each field of view through the lens. The image plane of each pinhole camera coincides with the focal plane.

$\Sigma_{i} \Pi_{P_{L_{1}}(L)}$ (7)

Where $S$ is the set of discrete pinhole positions within the lens aperture, and $P_{L_{1}}(L)$ is a projection operator that renders a view of the light field from pinhole position $i$ onto an image plane with center $c$, right vector $r$ and up vector $u$. We call this operator the focusing operator. Figure 6b illustrates focusing within a light field.

This focusing operator creates an image where objects at a single plane in the scene are in focus. Sometimes a user may want to focus on objects at multiple depths. We call this image a multi-focal plane image because the final image has more than one depth of focus. This has uses in sports photography and movies.

In a multi-focal plane image, each object, represented by a separate light field, has its own focal plane. Therefore, for each object we can tweak the amount of defocus it has by simply moving its associated focal plane. In our formulation, all the light fields are viewed through a common lens aperture.

To create a multi-focal plane image, we step through each sample point on the lens as was done for a single-plane of focus. However, for each sample point we create multiple pinhole images, one for each light field. The image plane for each pinhole projection is aligned to the focal plane for that light field and a 2D image is rendered from that light field. The pinhole images are then composited using the over operator to form a single image per sample point. We then sum all sample point images to form the multi-focal plane image.

For two light fields with two focal planes, the expression is:

$$\Sigma_{i \in S} \Pi_{P_{L_{1}}(L_{1})} \text{ over } P_{L_{2}}(L_{2})$$

where $c_{1}$ and $c_{2}$ are the locations of the centers of the focal planes for lightfield $L_{1}$ and $L_{2}$ respectively. Figures 6c and d illustrate the multi-focal plane image.

5.3 Refraction

We now define an operator that simulates objects with transmissive properties. This allows the object to refract other light fields in the scene. Figure 1c illustrates an example of two glass balls refracting two background light fields.
5.4 Shadows

To create sharp shadows, we introduce a virtual point light source outside the virtual light field. Notice that the virtual image is flipped, which is the correct effect for a solid glass ball. The artifacts in the virtual image are due to low sampling of the warping function \( F^{\text{refraction}} \), this could be remedied by using a higher sampling or by defining an analytic description of the ray warp. In (b), we add a reflection light field which contains a specular highlight. Both the highlight and the refraction move with the observer.

A refractive object warps the incoming rays to different outgoing directions. This warp can be stored as a function or it can be pre-computed and stored as a 4D lookup table [Heidrich et al. 1999; Yu et al. 2005]. We call this warping function \( F^{\text{refraction}} \). To create the refraction effect, we simply warp any light field behind the object with \( F^{\text{refraction}} \).

For example, Figure 4 shows how a calibration light field \( L_{\text{calibration}} \) is refracted through a glass ball light field. The compositing expression is:

\[
W(L_{\text{calibration}}, F^{\text{refraction}})
\]

\( F^{\text{refraction}} \) maps incident rays through the refracted sphere. We pre-compute \( F^{\text{refraction}} \) by raytracing through a synthetic sphere and storing the ray-to-ray mapping as a 4D lookup table.

Notice that Figure 4a does not appear realistic because it is missing the reflection component. To remedy this problem, we can also warp the background light field by a \( F^{\text{reflection}} \) warp, which maps input rays to reflected rays (as opposed to refracted ones). In this case, the reflection is a specular highlight. The final compositing expression is:

\[
A(W(L_{\text{calibration}}, F^{\text{refraction}}), W(L_{\text{highlight}}, F^{\text{reflection}}))
\]

Figure 4b shows an image from this composited light field.

6 Results

First we demonstrate how the deformation operator is useful for editing light fields. Figure 5 shows an image from a light field taken of two people. Unfortunately, the people were not looking straight-ahead during the acquisition. To correct this problem, we define a new function \( F^{\text{look}} \) which linearly interpolates two \( F^{\text{twist}} \) operators as defined in Section 5.1. One \( F^{\text{twist}} \) operator is specified for each head, and is weighted by proximity to that head.

One view of the resulting deformed light field is shown in Figure 5b. Notice that the people are now looking straight at the camera. The resulting image shows changes in visibility (cf. the previously hidden ear of the left figure), which would be impossible to achieve with an image editing approach.

Second, we show the results of having multi-focal planes through a light field. This operator lets a user emphasize multiple depths in a composited scene by focusing on them. This is useful when there are several interesting objects at different depths, but unwanted objects between them. Figures 6c and d illustrate these multi-focal light fields. The toy giraffe in the middle is defocused. The defocus blur correctly mixes with the colors in the background light field.

Third, we demonstrate how the refraction operator can be used to simulate refraction through a light field of a glass ball. Figure 7 shows images of glass ball light field in front of several background ones. Because the background are also light fields, the refracted image exhibits the proper parallax due to depth, which can be seen in the accompanying video. The refracted image is also upside-down, which is consistent with the optical properties of a glass sphere.

In addition, our system allows the refraction and the focusing operator to be combined. This allows a user to focus in scenes with refractive objects. In Figure 7a, the user focuses on the \( virtual \) image of the Buddha light field. Notice that that the background is defocused. In Figure 7b, the user refocuses on the real image of the Buddha, and the virtual image and specular highlight are defocused.
Finally, we illustrate the use of the shadow operator to cast shadows from light fields. Figures 1d and 8 illustrate this operator. Notice that the toy giraffe’s shadow projects a believable silhouette onto the ground. This shadow is also correctly shaped when it interacts with the background flower light field. We simulate soft shadows by inserting multiple point light sources into the scene.

7 Conclusions and future work

We have presented a system that allows a user to edit and composite light fields at interactive rates. The system is useful for editing light fields, creating novel imaging effects, or creating complex optical and illumination effects for games.

Instead of developing specific operators suited for a single task, we developed a framework that is based on a few basic operators and a compositing language allowing these operators to be combined. We have shown that these combinations can create interesting operations.

We are discovering that these operators can be used in a variety of contexts. For example our warping operator can be used to describe the aberrometry data of a human eye, produced from a Shack-Hartmann device [Platt and Shack 1971; Barsky et al. 2002]. The warped rays would sample from an acquired light field, presenting photorealistic images of scenes as seen through a human optical system.

Finally, we are investigating the use of our system to edit other 4D datasets, like 4D incident illumination [Sen et al. 2005; Chen and Lensch 2005], or general 8D reflectance fields. The ability to composite 8D reflectance fields enables captured objects in a scene to exhibit realistic inter-reflections, global illumination, and visibility changes. Such effects are currently very difficult to produce but may be possible with further extensions to our system.

References


Figure 7: Simulating light field refraction and focusing. Image (a) shows a view from a light field created from two refractive spheres in front of a flower and Buddha light field. The sphere and Buddha light fields were rendered synthetically with 1024 cameras, at 256 x 256 resolution. The image is created using the focusing operator described in Section 5.2 to focus on the virtual image of the flower in the refractive sphere. This makes the background appear blurry. Notice that the virtual image is correctly distorted by the sphere, it is upside-down and warped. In image (b) the focal plane is shifted to the background, and the virtual images are defocused.

Figure 8: Simulating shadows. In image (a), sharp shadows are being cast from the giraffe and flower light field. In image (b), we create multiple virtual light sources and sum their masks to create soft shadows. Notice that the soft shadow interacts correctly with the flower light fields. The image is also focused on the foreground, so that flower and shadows are blurred.


