Surface Light Fields for 3D Photography

Daniel N. Wood[1] *Daniel I. Azuma*[1] *Ken Aldinger*[1]
Brian Curless[1] *Tom Duchamp*[1] *David H. Salesin*[1,2] *Werner Stuetzle*[1]

[1]University of Washington [2]Microsoft Research

Abstract

A *surface light field* is a function that assigns a color to each ray originating on a surface. Surface light fields are well suited to constructing virtual images of shiny objects under complex lighting conditions. This paper presents a framework for construction, compression, interactive rendering, and rudimentary editing of surface light fields of real objects. Generalizations of vector quantization and principal component analysis are used to construct a compressed representation of an object's surface light field from photographs and range scans. A new rendering algorithm achieves interactive rendering of images from the compressed representation, incorporating view-dependent geometric level-of-detail control. The surface light field representation can also be directly edited to yield plausible surface light fields for small changes in surface geometry and reflectance properties.

CR Categories: I.3.2. [Computer Graphics]: Picture/Image Generation–*Digitizing and scanning, Viewing algorithms*

Keywords: surface light fields, 3D photography, lumigraph, light field, function quantization, principal function analysis, view-dependent level-of-detail, image-based rendering, wavelets.

1 Introduction

Recent advances in digital cameras, 3D laser scanners and other imaging technology are enabling us to capture enormous quantities of geometric and radiance data with unprecedented ease and accuracy. These advances hold great promise for *3D photography*, the process by which both the shape and appearance of physical objects are modeled and realistically rendered. But to make 3D photography truly practical, quite a few open problems still need to be solved.

First, we need a good representation for those 3D datasets. The framework described in this paper is based on the *surface light field*, a term coined by Miller *et al.* [22]. The surface light field is a function that assigns an RGB value to every ray leaving every point on a surface. When constructed from observations made of an object, a surface light field encodes sufficient information to construct realistic images of the object from arbitrary viewpoints. Surface texture, rapid variation in specularity, and global effects like interreflection and shadowing are all correctly represented. Some of these properties can be seen in Figure 1.

However, a good representation by itself is only half the story. Because the datasets acquired by 3D photography techniques are so large, good *compression* algorithms are needed. Furthermore, we need algorithms to *render* those datasets efficiently, ideally at interactive speeds. To this end, we need to develop *level-of-detail controls* for the rendering process, with shape and appearance under independent control. Finally, just as in traditional 2D photography, accurately capturing the real world is not sufficient for many applications; a useful representation for the results of 3D photography should also be *editable*.

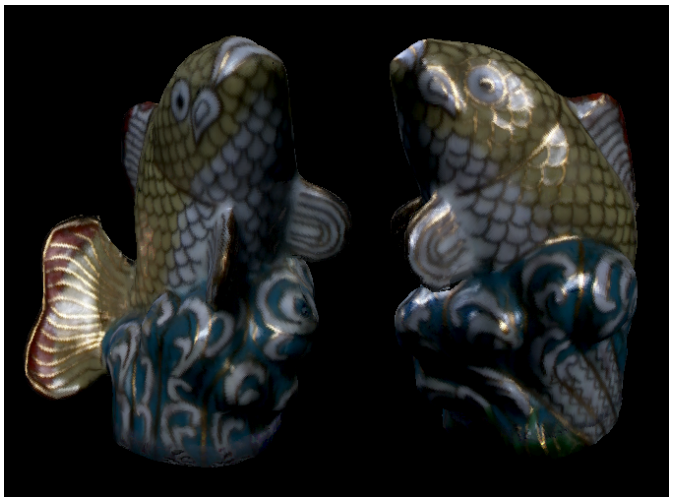


Figure 1 Images of a surface light field demonstrating detailed surface texture, rapid changes in specular properties, and interreflections. The specular variations occur, for example, in the gold paint on the tail of this porcelain fish. The tail also reflects light onto the body, as indicated by the reddish hue on the side of the fish in the left panel.

In this paper, we address each of these problems. In particular, our contributions include:

Estimation/compression. Our raw data consists of a set of 2D digital color photographs of an object together with a collection of laser range scans. To make a surface light field tractable for rendering, the data must fit into main memory. To this end we present two new algorithms that simultaneously estimate and compress the surface light field. The first is a generalization of vector quantization; the second is a generalization of principal component analysis.

Rendering. We demonstrate an algorithm that can render our surface light fields at interactive frame rates. Evaluation of the surface color takes time proportional to the occupied screen space. The amount of time required to render the underlying geometry is controlled using a new view-dependent level-of-detail algorithm for meshes with subdivision connectivity. The level of geometric approximation does not affect the sharpness of the surface texture.

Editing. Our representation of surface light fields allows editing, using 3D analogs of image processing algorithms to filter reflected light, and modifications of surface geometry. We can simulate changes in the reflectance properties of the surface, and we can generate plausible images of the object after it has been deformed or moved relative to its environment.

SIGGRAPH 2000, New Orleans, LA USA

1.1 Related work

Surface light fields fit into the broad framework of *image-based* rendering schemes. Image-based methods take a collection of photographs as input, construct a representation of the surface color or radiance, and use it to synthesize new images from arbitrary viewpoints. The methods tend to differ in the number of input images they use, the representation of the data, the degree to which they incorporate geometric information about the object into the image representation, and the compression techniques they employ. Our own approach leverages high-resolution geometry to improve image quality while affording a compact representation.

Levoy and Hanrahan [17] acquire many hundreds of images, which are resampled to lie on a regular grid in a two-plane parameterization. New images are computed by interpolation between ray samples, using essentially no geometric data. They apply vector quantization to obtain compressed representations of light fields. Gortler *et al.* [12] present a similar two-plane parameterization that they call a *lumigraph*, in which they interpolate image samples via a hierarchical *push-pull* algorithm. They use approximate surface geometry derived from photograph silhouettes (or higher-resolution geometry in the case of synthetic data) to perform a depth correction that substantially reduces ghosting and blurring artifacts. In both these methods, the representation restricts the viewpoint to lie outside of the convex hull of the object. Magnor and Girod [20, 21] develop an MPEG-like scheme for compressing two-plane light fields that produces better compression ratios than those obtained by Levoy and Hanrahan. Our approach depends on both high-resolution geometry and dense sets of images. It removes the convex hull restriction of the two-plane light field and admits a new form of compressed representation that can be rendered in real time. For comparable data sizes, our representation yields sharper images and greater compression ratios than two-plane representations.

View-dependent texture mapping [7, 8, 26] is a kind of light field that does not require resampling the input images. This approach uses geometric information to re-project each input image into the desired camera viewpoint. The re-projected input images are then blended together using weights based on the view direction primarily, and possibly other factors such as sampling rate. Because the blending in view-dependent texture mapping incorporates visibility information, this approach supports rendering within the convex hull of the object. In practice, view-dependent texture mapping has been used with fewer images and surfaces that are less specular than those demonstrated with two-plane light fields, though this is not a fundamental limitation. As noted in Debevec *et al.* [8], a surface light field can be viewed as a distillation of view-dependent texture mapping into a more efficient representation.

Miller *et al.* [22] use surface light fields to render solutions to synthetic (non-diffuse) global illumination problems. They apply JPEG-like image compression techniques to sets of texture maps. Their technique achieves compression rates for surface light fields that are comparable to those of Levoy and Hanrahan's vector quantization method. Walter *et al.* [31] also use surface light fields to approximate solutions to global illumination problems. Their representation involves basis functions derived from hardware lighting models, which provides very fast rendering, but does not support textured surfaces, nor can it adequately model complex phenomena such as rapidly varying specularity. In addition, problems exist in the 3D photography realm that do not arise with synthetic data: most importantly, neither a surface parameterization nor the radiance along arbitrary rays are known *a priori* and must instead be constructed.

Nishino *et al.* [23, 24] generate surface light fields of real objects, though their images are relatively dense in only one rotational direction. Geometric information is represented by a coarse triangular mesh. They construct a set of texture maps for each triangle by projecting each image onto the mesh. Compression is achieved by performing a principal component analysis on each set of textures. (Interestingly, the vectors in their analysis are formed by holding a direction fixed and letting surface location vary. This is the opposite of our analysis in Section 4.6, where, to form a vector, we fix a surface location and let direction vary.) Their approach successfully models objects with simple geometric structure and smoothly varying specularity. However, it has not been demonstrated on objects that exhibit both high geometric complexity and rapid BRDF variation, nor does it provide real-time rendering.

Inverse rendering is an alternative to generating a surface light field. The goal of these techniques is to estimate the surface BRDF from images and geometric data. Previous work on inverse rendering [28, 33] has assumed that the BRDF is piecewise linear with respect to a coarse triangulation of the surface. Our techniques require no such assumptions, and, of course, inverse rendering does not solve the re-rendering problem—a non-interactive global illumination algorithm is required to produce photorealistic results. Recent work has extended interactive rendering techniques to a wider range of lighting models and environments. Cabral *et al.* [3] describe a technique for using radiance environment maps to render objects under arbitrary lighting conditions and with any isotropic BRDF. Heidrich *et al.* [13] use texture mapping hardware for the same purpose but allow a different class of BRDFs. However, these two methods do not handle global effects like shadows or interreflection.

1.2 Overview

We have developed algorithms for acquiring light field data of real objects, and for estimating, compressing, rendering, and editing their surface light fields. We have tested these algorithms on two objects, a small ceramic fish with a shiny surface and detailed texture, and a marble elephant with more complex geometry and less pronounced specular highlights.

The following sections describe these new algorithms in detail. We begin by describing our representation of surface light fields (Section 2). Next, we discuss our data acquisition process (Section 3). We then describe our algorithms for estimating and compressing surface light fields and compare the quality of these methods to two-plane light fields of similar size (Section 4). Finally, we discuss our algorithms for rendering and editing surface light fields (Sections 5 and 6), and present ideas for future research (Section 7).

2 Representation

Roughly speaking, a surface light field is a function that associates a color to every ray originating from a surface. Our algorithm for constructing images from a surface light field relies on a good parameterization of an object's surface mesh M. The methods of either Eck *et al.* [9] or Lee *et al.* [16] yield a parameterization

$$\varphi : K_0 \rightarrow M \subset \mathbb{R}^3, \tag{1}$$

whose domain K_0 is a triangular mesh with a small number of faces, called a *base mesh*. We use a variant of the algorithm of Lee *et al.* to parameterize our scanned geometry.

The parameterization allows us to represent the surface light field as a function

$$L : K_0 \times S^2 \rightarrow RGB, \tag{2}$$

where S^2 denotes the sphere of unit vectors in $\mathbb{R}^3$. Radiance is represented by points in $\mathbb{R}^3$ corresponding to RGB triples. If u is a point on the base mesh and ω is an outward pointing direction at the surface point $\varphi(u)$, then $L(u,\omega)$ is the RGB value of the

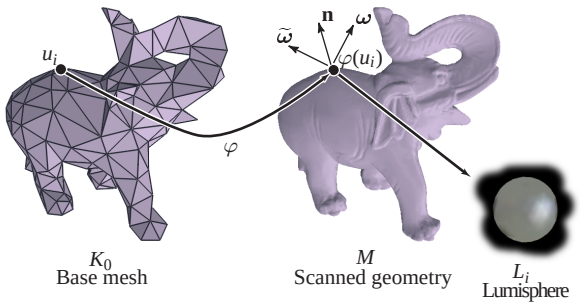


Figure 2 Representation of the surface light field. Points u on the base mesh, K_0, are mapped to the geometric surface, M, by φ. The lumisphere, L_i at the grid point u_i, represents the radiance leaving surface point $\varphi(u_i)$. Directions are denoted by $\boldsymbol{\omega}$, or $\widetilde{\boldsymbol{\omega}}$ after reflection through the surface normal $\mathbf{n}$ as described in Section 4.4.

light ray starting at $\varphi(u)$ and traveling in direction $\boldsymbol{\omega}$. Although $L(u,\boldsymbol{\omega})$ has no physical interpretation when $\boldsymbol{\omega}$ is inward pointing, our compression, rendering, and editing techniques rely on L being defined over the entire direction sphere.

We make the simplifying assumption that $L(u,\boldsymbol{\omega})$ is piecewise linear in $\boldsymbol{\omega}$. To make this more precise we have to define what we mean by a piecewise-linear function on S^2. It is not difficult to verify that the map

$$h(\boldsymbol{\omega}) \equiv \frac{(\sin^{-1}\omega_x,\ \sin^{-1}\omega_y,\ \sin^{-1}\omega_z)}{|\sin^{-1}\omega_x| + |\sin^{-1}\omega_y| + |\sin^{-1}\omega_z|} \tag{3}$$

is a homeomorphism between S^2 and the regular octahedron with vertices $(\pm1,\pm1,\pm1)$. We use h because it introduces less distortion than radial projection and yet can be evaluated quickly using a lookup table for $\sin^{-1}$.

Composition with h induces a bijection between functions on the octahedron and functions on the sphere. We say that a function $F(\boldsymbol{\omega})$ is *piecewise linear* if it is piecewise linear with respect to an s-times-subdivided octahedron, *i.e.*, the mesh resulting from s four-to-one subdivisions of the octahedron. We call a piecewise-linear RGB-valued function a *lumisphere*, and we let C^s_{PL} denote the vector space of all lumispheres.

With these definitions, the surface light field L can be represented by a function, whose domain is K_0 and whose range is C^s_{PL}, that sends a point u on K_0 to the lumisphere $L(u,\cdot)$. This definition can be described compactly in mathematical notation as follows:

$$K_0 \rightarrow C^s_{PL} \ :\ u \mapsto L(u,\cdot) \tag{4}$$

We have chosen subdivision level $s = 3$ in all our examples. In this case the space of lumispheres has dimension $3 \times 258 = 774$. We arrived at this value experimentally. Setting $s = 2$ results in noticeable degradation in the image quality, while $s = 4$ gives little improvement at the expense of higher dimension.

It is useful to think of a surface light field as a lumisphere-valued texture map, which assigns a lumisphere instead of a single color to each texel. There is one rectangular texture map for each triangle in K_0. The K_0 triangle is mapped to the lower-left corner of its rectangle, and the upper right corner is unused. (For compactness we store pairs of texture maps interleaved in memory.) As in conventional texture mapping, each texture map is divided into square texels, and these texels define a partition of each face of K_0 into cells. The surface light field L is thus piecewise-constant with respect to this partition of K_0. Let u_i denote the center of the i-th cell. Cell dimensions (correspondng to the texture map resolution) are chosen so that the images $\varphi(u_i)$ and $\varphi(u_j)$ of any two adjacent grid points u_i and u_j are separated by at most one pixel in the image plane of each camera. We denote the lumisphere at the grid point u_i by L_i—that is, $L_i(\boldsymbol{\omega}) \equiv L(u_i,\boldsymbol{\omega})$.

Figure 2 illustrates key aspects of our notation.

3 Data acquisition

Acquiring the raw data to build a surface light field for a real object requires four steps: (1) range scanning the object, (2) building a mesh to represent its geometry, (3) capturing a collection of images of the object, and (4) registering the images to the mesh. Because the techniques presented in this paper do not depend on the specifics of our acquisition process, we present only a brief summary here of the procedure that we have used successfully.

Range scanning. We took a number of range scans of each object using a Cyberware Model 15 scanner. Glossy objects like the fish and elephant are not ideal candidates for laser scanning. To improve laser returns, we coated them with a removable powder. The fish was built from 36 scans, and the elephant from 49.

Reconstructing the geometry. The scans were registered using a small number of hand-selected point correspondences to initialize a global iterated closest-points algorithm [2, 10]. The registered scans were merged into a single triangle mesh using the volumetric method described by Curless and Levoy [6]. The final meshes representing the surfaces of the fish and elephant contain 129,664 triangles and 311,376 triangles, respectively.

Acquiring the photographs. We used a camera attached to a spherical gantry arm to capture photographs from poses spaced roughly evenly over the sphere. The camera positions were known relative to one another, but not relative to the objects being photographed. We took 638 photographs of the fish and 388 photographs of the elephant, together with photographs of a calibration pattern, which we used to determine the intrinsic camera parameters using Tsai's method [30]. During acquisition, the camera and gantry arm occasionally cast shadows onto the object. Because we wanted to capture the object under fixed lighting conditions, we manually removed photographs taken under those circumstances.

Registering the photographs to the geometry. We registered the set of photographs to the reconstructed mesh with user assistance. By hand-selecting correspondences between points on the mesh and points on a small subset of the photographs, we generated a list of 3D point-to-ray correspondences. We then registered the photographs to the geometry using an iterated closest-points algorithm.

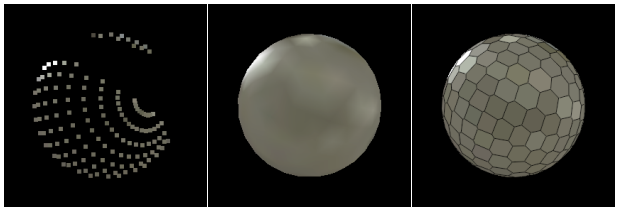
4 Estimation and compression

Once we have acquired the raw image and geometric data, we must estimate a surface light field that approximates that input. This section describes three estimation techniques; the latter two directly create compressed representations.

4.1 Assembling data lumispheres

The first step in the estimation process is the construction of a useful intermediate representation, consisting of a *data lumisphere* for each grid point in the surface light field. A data lumisphere is a set of samples from a full lumisphere, each consisting of a color and a direction corresponding to an observation of a grid point. We use $\mathcal{L}_i$ to denote the data lumisphere associated with point u_i on the base mesh. Assembling data lumispheres is a resampling problem that we solve separately for each grid point on the base mesh K_0.

Consider a fixed grid point u_i, and let c_{ij} denote the RGB value of the point in the j-th photograph defined by the ray from $\varphi(u_i)$ to the



(a) (b) (c)

Figure 3 Lumispheres from a point under the elephant's trunk. (The surface normal points directly out of the page.) The swath of missing points were occluded by the trunk. (a) Data lumisphere. (b) Faired piecewise-linear lumisphere. (c) Faired lumisphere with vertices shown as constant-colored Voronoi regions (used for illustration only).

location of the j-th camera. The value c_{ij} is computed by bilinear interpolation in the j-th photograph. Some or all of the c_{ij} might be invalid because the point $\varphi(u_i)$ may not be visible from the j-th camera position. If $\varphi(u_i)$ is visible, we find the direction vector $\boldsymbol{\omega}_{ij}$ from $\varphi(u_i)$ to the location of camera j and add the pair $(c_{ij}, \boldsymbol{\omega}_{ij})$ to the data lumisphere $\mathcal{L}_i$ for grid point u_i. Figure 3(a) shows the data lumisphere for a point on our elephant.

To determine if $\varphi(u_i)$ is occluded with respect to the j-th camera, we render, from the perspective of that camera, both the original mesh M and additional geometry that conservatively bounds the platform on which the object rests. (Because the platform obscures parts of the objects in some photographs, we add geometry representing the platform to ensure that we do not project the platform onto the object.) The depth buffer gives us a *depth image*, which we compare to the depth of each point $\varphi(u_i)$ to determine if it is visible.

4.2 Pointwise fairing

Our first estimation algorithm, *pointwise fairing*, constructs a piecewise-linear lumisphere from each data lumisphere independently at each surface point. If the data covered the entire direction sphere, we could estimate L_i using the standard least-squares procedure of setting L_i to be the lumisphere in C^s_{PL} that best approximates the data lumisphere:

$$L_i = \operatorname*{argmin}_{F \in C^s_{PL}} E_{\text{dist}}(F, \mathcal{L}_i) \tag{5}$$

The argmin notation evaluates to the value of its subscript that minimizes the expression to which it is applied. Here, F is a lumisphere, and $E_{\text{dist}}(F, \mathcal{L}_i)$ measures how well F approximates $\mathcal{L}_i$:

$$E_{\text{dist}}(F, \mathcal{L}_i) \equiv \frac{1}{|\mathcal{L}_i|} \sum_{j \in \text{visible cameras}} |F(\boldsymbol{\omega}_{ij}) - c_{ij}|^2 \tag{6}$$

where $|\mathcal{L}_i|$ is the number of observed color values in the data lumisphere $\mathcal{L}_i$.

But the physical light field at any point on the surface is only defined on the hemisphere of outward pointing directions. Moreover, due to self-occlusion and constraints on the camera poses, the data samples often do not cover the entire direction hemisphere (see Figure 3). The fitting problem (Equation (6)) is under-determined, and it is therefore necessary to regularize it by adding a fairing term. We use a discrete approximation to the thin-plate energy:

$$E_{\text{thin}}(F) \equiv \frac{N_s}{4\pi} \sum_k |\Delta_{PL} F(\boldsymbol{\omega}_k)|^2 \,. \tag{7}$$

The sum ranges over the vertices of the s-times-subdivided octahedron (with N_s vertices, each corresponding to a direction $\boldsymbol{\omega}_k$), and Δ_{PL} denotes the umbrella Laplacian [29]. The regularized error function is then

$$E_\lambda(F, \mathcal{L}_i) \equiv E_{\text{dist}}(F, \mathcal{L}_i) + \lambda E_{\text{thin}}(F) \,. \tag{8}$$

We use conjugate gradients to find the lumisphere F that minimizes Equation (8). Figures 3(b) and 3(c) show the faired lumisphere generated from the data lumisphere in Figure 3(a). The fairing term dampens the directional variation in the fitted lumisphere. It has little physical significance, and our data is relatively free of noise; we therefore choose λ small so that E_{dist} dominates. Note that our fairing procedure assigns values to $L(u, \boldsymbol{\omega})$ at all directions $\boldsymbol{\omega} \in S^2$, including directions far away from any observations, and even directions pointing into the object.

Figure 4 illustrates the effects and the limitations of pointwise fairing. Figure 4(a) shows one of the actual photographs of the fish, and Figure 4(b) shows the same view of the uncompressed light field generated from all the photographs. The light field rendered in Figure 4(c) was generated after all photographs from viewpoints inside a cone of radius 10° about the viewing direction were removed. There is little degradation. In Figure 4(d) the radius of the cone was increased to 20°. Clearly the gap in directions has become too large for pointwise fairing to accurately approximate the actual surface light field.

4.3 Compression overview

Because C^s_{PL} is a high-dimensional space, a complete pointwise-faired surface light field may be very large. To generate a more compact surface light field, we will represent each lumisphere as a weighted sum of a small number of prototype lumispheres using two distinct methods, one analogous to vector quantization, and the other analagous to principal component analysis. Each lumisphere L_i can then be replaced by an index (as in vector quantization) or a set of coefficients (as in principal component analysis) indicating contributions from the prototypes.

A naive application of vector quantization or principal component analysis might treat as input the pointwise-faired lumispheres viewed as vectors in the space C^s_{PL}. Observe, however, that the RGB values for at least half of each lumisphere—corresponding to directions pointing into the object—are mostly fiction generated by the fairing process. If we were to apply vector quantization or principal component analysis to the pointwise-faired lumispheres, these fabricated values would have the same influence as values in directions where we actually have data. This is clearly undesirable.

A more principled compression approach would use only observed data. The data, however, is an irregular and incomplete sampling of each lumisphere. We have therefore developed two new estimation/compression methods, *function quantization* and *principal function analysis*, which are similar in spirit to vector quantization and principal component analysis, but are driven by irregularly spaced data and avoid the intermediate pointwise-fairing step.

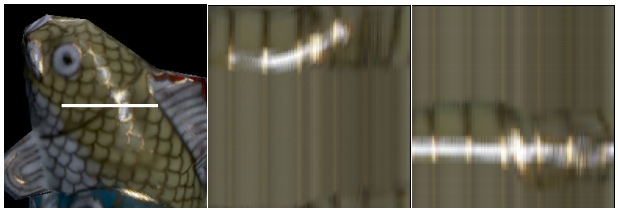
Before discussing our compression algorithms, we present two transformations of the surface light field that increase spatial coherence among lumispheres, thereby making them more compressible.

4.4 Median removal and reflection

The first transformation is *median removal*. Let m_i denote the RGB value obtained by computing the median color of data lumisphere $\mathcal{L}_i$ (separately for each color channel). We use the median rather than the mean because it is robust against outliers and more accurately represents the bulk of the data. The collection of median values can be viewed as a texture map over the surface, roughly encoding the diffuse component of the surface light field. We store this "diffuse" texture map separately and then encode the residual

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

Figure 4 Analysis of estimation with missing data. (a) Photograph taken by a selected camera. (b) Faired surface light field using all photographs. (c) Faired surface light field after first removing from the input data all photographs in a cone of radius 10° about the direction shown. (d) Faired surface light field with a cone of radius 20° removed. (e) Compressed surface light field with principal function analysis of order 3 after first removing the same cone of radius 20°. Note that the compressed surface light field reproduces the specularity of the input better than the pointwise-faired version when a significant portion of the input data is removed.



(a) (b) (c)

Figure 5 Increasing lumisphere coherence via reflection reparameterization. (a) Surface light field. (b) Transect of $L(u,\omega)$. (c) Transect of $\widetilde{L}(u,\widetilde{\omega})$. Horizontal axis shows position of u along white line across fish (a). Vertical axis shows position of ω (b) or $\widetilde{\omega}$ (c) on a user-selected great circle. Note that in the right panel the specular highlights are much better aligned.

surface light field after subtracting the diffuse component. This serves two purposes. First, if we compress only the residual surface light field, any diffuse texture will be exactly preserved. Second, the residual will be more compressible if the specular behavior of the surface is simpler than the diffuse (*e.g.*, an object with diffuse texture and a glossy coat.) Median removal before compression is analogous to mean-removed vector quantization [11].

The second transformation, *reflection*, is a reparametrization of the lumispheres. Let $\mathbf{n}$ be the unit surface normal at a surface. Then for a direction $\omega \in S^2$, let $\widetilde{\omega}$ be the reflection of ω about the normal $\mathbf{n}$ (transformed quantities will always be denoted with a tilde '˜'):

$$\widetilde{\omega} \equiv 2\,(\mathbf{n}\cdot\omega)\,\mathbf{n} - \omega. \tag{9}$$

Similarly, the reflected (and median-removed) surface light field $\widetilde{L}$ is defined at each grid point by:

$$\widetilde{L}_i(\widetilde{\omega}) \equiv L_i(\omega) - m_i\,. \tag{10}$$

Where, by Equation (9), ω is $\widetilde{\omega}$ reflected around the surface normal, $\mathbf{n}_i$, at the i-th grid point. Obviously, $\widetilde{L}$ (plus the diffuse texture map) contains the same information as L. To see why we expect the reflected reparameterization to increase spatial coherence, consider the three elements that determine the lumisphere L_i at a point: the incoming radiance, the BRDF and the normal. First, assume that the incoming radiance at two points u_i and u_j is the same; this is approximately true for points that are nearby relative to the sources of light illuminating them. Second, assume that the BRDF is reflective. A *reflective BRDF* [3] is one that reflects the incoming radiance through the surface normal and then convolves with a "direction-invariant" filter (*i.e.*, a space-invariant filter, where space is restricted to the surface of the sphere of directions S^2). As observed by Rusinkiewicz [27], many BRDFs are approximately reflective. If these two assumptions hold, the reflected lumispheres $\widetilde{L}_i$ and $\widetilde{L}_j$ will be the same even if the normals $\mathbf{n}_i$ and $\mathbf{n}_j$ are different. For an example, consider the case of a perfect mirror surface and an environment that is infinitely far away. Ignoring non-local effects such as occlusions and interreflections, all of the reparametrized lumispheres will agree on their overlap because they contain parts of the same environment map. If the surface had some roughness, then the lumispheres would be blurred, reflected images of the environment, but they would still roughly agree on the overlap. Figure 5 illustrates the effect of reparameterization for the fish, whose environment consists of several small light sources.

We always estimate and store median-removed and reflected lumispheres; however, the transformations have no effect on the pointwise-fairing algorithm.

4.5 Function quantization

Function quantization is a generalization of vector quantization to the case of irregularly sampled data. The goal is to construct a *codebook* comprised of a collection of prototype lumispheres (*codewords*) $\{P_0,\ldots,P_n\}$ and a map assigning to each grid point $u_i \in K_0$ a codeword index k_i, and thereby a codeword P_{k_i}. For a given n, the codebook and map should minimize the combined energy over all data lumispheres, *i.e.*, $\sum_i E_\lambda(P_{k_i},\widetilde{\mathcal{L}}_i)$. This formulation is different from vector quantization in that the inputs (data lumispheres) are not vectors.

Function quantization starts with an initial codebook consisting of a single lumisphere and a small *training set* of randomly selected grid points. It proceeds by alternating between *codebook fitting* and *codeword splitting*, until the codebook reaches a user-specified size.

Codebook fitting is accomplished via Lloyd iteration [11], *i.e.*, by repeatedly applying the following two steps:

1. ***Projection:*** For each grid point u_i in the training set, find the index k_i of the closest codeword:

$$k_i = \operatorname*{argmin}_k E_\lambda(P_k,\widetilde{\mathcal{L}}_i). \tag{11}$$

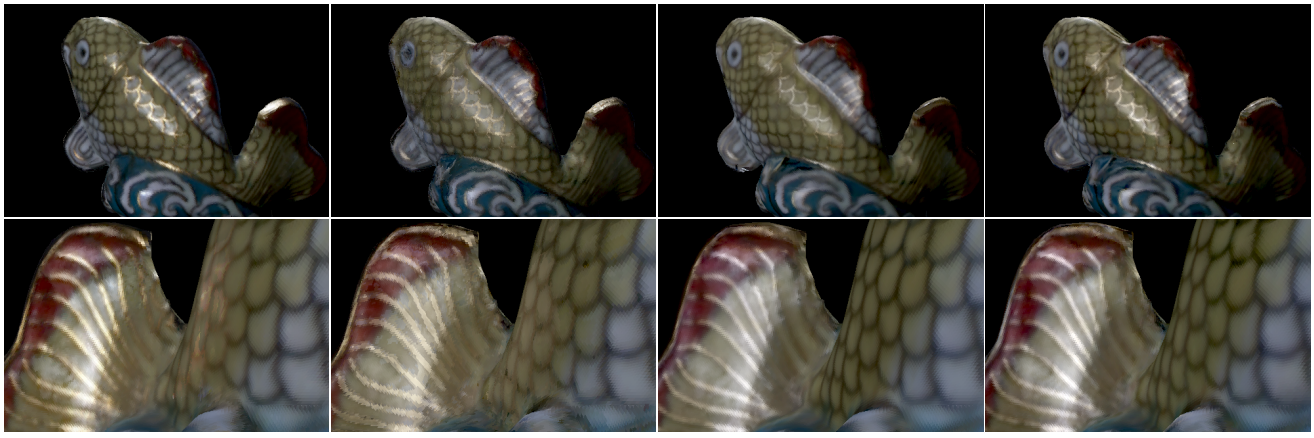
   This partitions the training set into clusters of grid points that project to the same codeword.

2. ***Optimization:*** For each cluster, find the best piecewise-linear lumisphere:

$$P_k = \operatorname*{argmin}_{F\in C^s_{PL}} \sum_{i\in\text{cluster }k} E_\lambda(F,\widetilde{\mathcal{L}}_i)\,, \tag{12}$$

   where the summation is over all of the data lumispheres $\widetilde{\mathcal{L}}_i$ in the k-th cluster.

We perform the optimization steps using conjugate gradients. The iteration terminates when the decrease in error between successive codebooks falls below a user-defined threshold. Then, if the



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

Figure 6 Comparison of different estimation techniques applied to the fish. (a) Pointwise faired surface light field. (b) Function quantization with 1024 codewords. (c) Principal function analysis with subspace dimension 2. (d) Principal function analysis with subspace dimension 5.

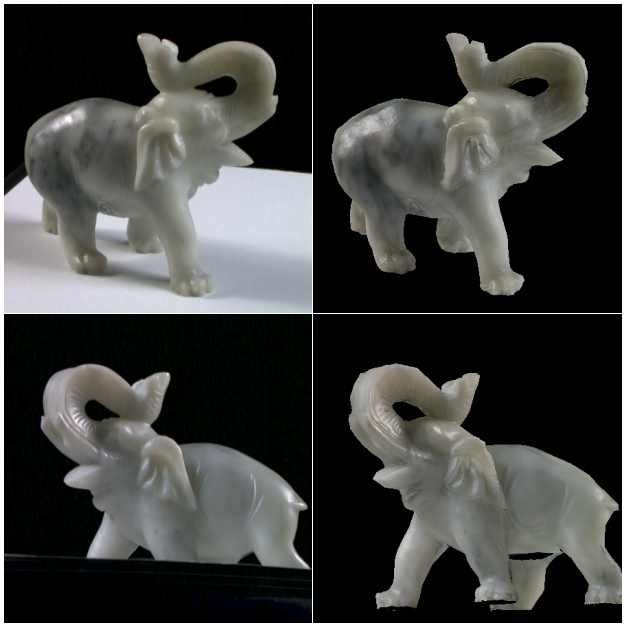


Figure 7 Comparison of compressed elephant surface light field with input photographs. Left: Elephant photographs. Right: Elephant surface light field (5.3 megabytes encoded with principal function analysis, subspace dimension $q = 2$). Note that the image on the bottom right shows a part of the elephant that was occluded in the corresponding photograph. Also note that some points on the very bottom of the elephant were not seen by any camera (using our conservative approximation of the platform) and are black.

codebook is smaller than desired, codeword splitting doubles the codebook size by cloning each codeword and adding a small perturbation to the clone. After a codebook of the desired size has been found, codewords are assigned to all grid points by projecting all the corresponding data lumispheres (not just those in the training sample) onto the codebook.

4.6 Principal function analysis

Principal function analysis, based on principal component analysis, is an alternative to function quantization. For a given set of data vectors and a given approximation dimension q, principal component analysis finds the q-dimensional affine subspace that best approximates the data vectors in the least squares sense. As in the case of function quantization, we must generalize this approach to the case of irregularly sampled data.

Our goal, then, is to find the q-dimensional subspace $V \subset C^s_{PL}$ that best approximates all of the data lumispheres in the training set. Each lumisphere $\widetilde{L}_i$ is represented by the point $F \in V$ that minimizes $E_\lambda(F, \widetilde{\mathcal{L}}_i)$. We call F the *projection of* $\widetilde{\mathcal{L}}_i$ *onto* V, or $\pi_V(\widetilde{\mathcal{L}}_i)$. Overloading E_λ, we view it as a function of q-dimensional subspaces of C^s_{PL}; it measures how well a subspace approximates the data lumispheres in the training set, *i.e.*,

$$E_\lambda(V) \equiv \frac{1}{T} \sum_i E_\lambda(\pi_V(\widetilde{\mathcal{L}}_i), \widetilde{\mathcal{L}}_i)\,. \tag{13}$$

The summation is over all grid point indices in the training set, and T is the size of the training set.

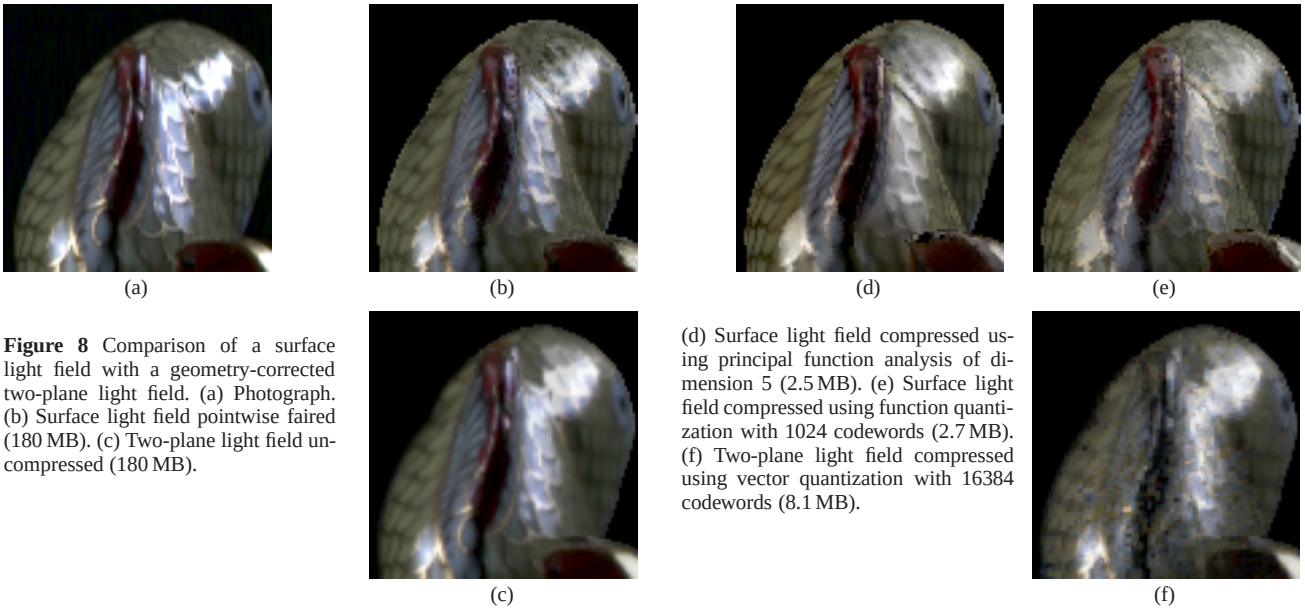
While principal component analysis reduces to an eigenvalue problem, we have not succeeded in finding a corresponding formulation for minimizing the functional defined in equation (13). We have therefore taken a different approach.

Each q-dimensional affine subspace of C^s_{PL} can be expressed as the affine span of $q + 1$ prototype functions, and E_λ can be regarded as a functional on the space of $(q + 1)$-tuples of prototypes. Since E_λ depends only on the affine span of the prototypes, minimizing E_λ will not uniquely determine the prototypes.

To address the uniqueness problem, we consider a new functional:

$$E_{\lambda,\mu}(P_0, \ldots, P_q) \equiv E_\lambda(V) + \mu \sum_{k \in 0,\ldots,q} \|P_k - P_{mean}\|^2 \tag{14}$$

where $P_0, \ldots, P_q$ are the prototypes defining V, and P_{mean} is their mean, and where the projection $\pi_V(\widetilde{\mathcal{L}}_i)$ of a data lumisphere $\widetilde{\mathcal{L}}_i$ is restricted to lie inside the convex hull of the prototypes. (The squared norm of a lumisphere, $\|F\|^2$, is the sum of the squared norms of the vertex values divided by the number of vertices.) This additional spring energy term penalizes widely-spaced prototypes. Minimizing it is a non-linear optimization problem, which we solve via conjugate gradients. After the subspace has been determined by selection of the prototypes, we assign barycentric coordinates to all grid points by projecting all corresponding data lumispheres (again, not just those in the training sample) onto the subspace.



(a) (b) (d) (e)

(c) (f)

Figure 8 Comparison of a surface light field with a geometry-corrected two-plane light field. (a) Photograph. (b) Surface light field pointwise faired (180 MB). (c) Two-plane light field uncompressed (180 MB). (d) Surface light field compressed using principal function analysis of dimension 5 (2.5 MB). (e) Surface light field compressed using function quantization with 1024 codewords (2.7 MB). (f) Two-plane light field compressed using vector quantization with 16384 codewords (8.1 MB).

4.7 Compression results

We tested the various estimation and compression algorithms on the surface light fields of both the fish and elephant. Figure 6 compares results of the different methods. Figure 6(a) shows two views of the uncompressed (pointwise-faired) fish, the entire model (top) and a closeup of the tail fin (bottom). This data set contains 176 MB of color data, plus 0.7 MB for geometry and normals. Figure 6(b) demonstrates function quantization with 1024 codewords, resulting in a color data size of 2.7 MB. Figures 6(c) and (d) illustrate principal function analysis with subspace dimensions 2 and 5, resulting in color data sizes of 1.7 MB and 2.3 MB, respectively. Note that the 2-dimensional principal function analysis example, with its total file size of 2.4 MB (1.7 MB color + 0.7 MB geometry), results in more than 70:1 compression.

Overall, principal function analysis leads to smoother images than function quantization; function quantization introduces artifacts such as jagged edges on the fish's tail. However, function quantization is more accurate, better preserving the color of highlights and effects such as interreflections that are lost during principal function analysis.

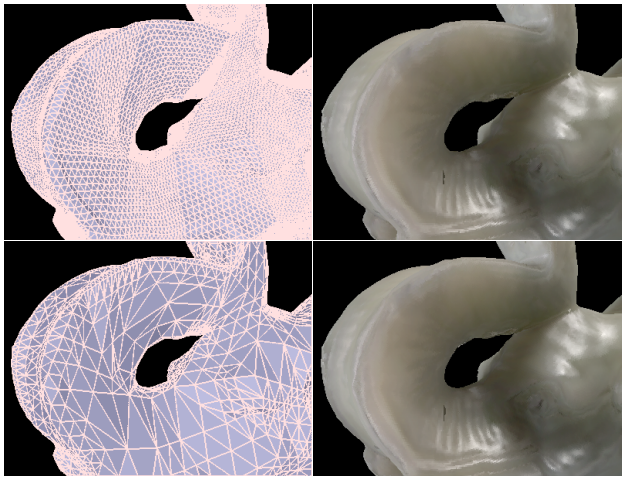
Not surprisingly, increasing the dimension of the subspace in principal function analysis improves the quality of the results; *e.g.*, dimension 5 produces highlights substantially sharper and brighter than dimension 2. Rendering time, however, is asymptotically linear in the dimension q. Currently, other costs dominate when the dimension is low, and in our examples, dimensions 2 and 5 can be rendered at roughly the same speed. By contrast, the rendering time for a function-quantized surface light field is independent of codebook size (ignoring the effect of the memory hierarchy). The complementary strengths of function quantization and principal function analysis suggest a hybrid approach (see Section 7).

We achieved similar compression results with the elephant. A pointwise-faired elephant requires 409 MB of color data and 1.6 MB of geometric data. Applying principal function analysis with a 2-dimensional subspace compresses the elephant's color data to 3.7 MB.

Figure 7 compares synthesized images of the elephant with the photographs. The compressed surface light field captures most of the features of the input data, but the highlights are less bright. In addition to the lower dimension of the subspace, the lower fidelity may be a result of the fact that the scanned geometry of the elephant appeared to be of lower quality than that of the fish. Errors in the geometry, particularly the normals, adversely affect the quality of the compression. Note also that, even though the bottom reconstruction includes the feet of the elephant, which were not visible in the corresponding photograph, our compression algorithm succeeds in inferring plausible shading and highlights for that part of the model. The compressed representation is essentially a learned model of lumispheres; the unseen portions of data lumispheres are filled in by finding the closest lumisphere in the model. Figures 4(d) and (e) also show that principal function analysis can produce more realistic highlights than pointwise fairing given incomplete data lumispheres.

We have done an informal comparison of image quality between a surface light field and a two-plane light field. We constructed a two-plane light field of the fish with six slabs arranged along the faces of a cube. The resolution of the light field, 400^2 for the far plane and 8^2 for the near plane, was chosen to approximately match the corresponding resolutions of the surface light field: the far-plane resolution matches the input photograph resolution, and the near-plane resolution approximately matches the surface light field's directional resolution. The resulting raw data size is about 180 MB, the same size as our pointwise-faired (*i.e.*, uncompressed) surface light field. The input images were resampled into the two-plane parameterization offline using the scanned fish geometry and view-dependent texture mapping. We then compressed the data using the vector quantization technique (and software) of Levoy and Hanrahan [17], using their default settings: a codebook of 16384 $2\times2\times2\times2\times3$ codewords (*i.e.*, 2×2 camera positions, 2×2 image pixels and 3 color channels). All of the renderings of two-plane light fields use the geometry correction technique of Gortler *et al.* [12].

Figure 8 compares images generated from uncompressed and compressed surface light fields with corresponding images generated from the two-plane light field. The uncompressed data sets give reproductions of similar quality, although the two-plane light field's quadralinear interpolation has different filtering characteristics. When compressed, the surface light field produces more compelling reproductions even though the compressed two-plane light field data (8.1 MB + geometry) is more than 3 times the size of the compressed surface light field (2.5 MB + geometry).



	Model	Faces	False color (secs/frame)	Light field (secs/frame)
top	uniform	102,400	0.36	0.59
bottom	LOD	5823	0.07	0.31

Figure 9 View-dependent level-of-detail. Left: Geometry visualization. Right: Surface light field. Top: Uniform subdivision, $r = 4$. Bottom: View-dependent level-of-detail with error terms chosen to match the uniform subdivision. Shown in the table, very bottom, are rendering times, first for false color only (the step that uses geometry), and second for the entire surface light field rendering algorithm.

The near-plane resolution of the two-plane light field we constructed, though comparable in angular resolution to our surface light field, is lower than those demonstrated by Gortler *et al.* and Levoy and Hanrahan. We have observed that lowering this resolution results in artifacts such as erroneous interpolation among rays that strike quite different surface points due to occlusion. Azuma [1] discusses this effect and other difficulties inherent in reduction of the near-plane resolution.

5 Rendering

In this section we present an interactive surface light rendering algorithm. Our implementation runs entirely in software and achieves interactive rates on PC-class hardware without 3D acceleration.

5.1 Basic algorithm

Rendering a surface light field from an arbitrary viewpoint is conceptually straightforward. Each pixel in the image plane of the camera defines an incoming ray in some direction $\boldsymbol{\omega}$. Suppose the ray intersects the mesh at a point $\varphi(u_i)$, corresponding to a point $u_i \in K_0$. Then the RGB value of the pixel is $L(u_i, \boldsymbol{\omega})$. Since we actually encode the reparameterized surface light field $\widetilde{L}(u_i, \widetilde{\boldsymbol{\omega}})$ at each point, we must reflect the viewing ray about the normal before looking up the RGB value. To facilitate this process, we compute and store a *normal map* $\mathbf{n}(u)$ over the surface, so that we can quickly determine the normal $\mathbf{n}(u_i)$ at a grid point.

We render the surface light field in two passes. In the first pass we determine, for each pixel of the virtual camera, the point u_i corresponding to the surface point $\varphi(u_i)$ seen at that pixel, encoded as a face ID and barycentric coordinates. We do this efficiently by rendering the mesh in false color with Gouraud shading, using two of the framebuffer's four color channels to encode the index of the base mesh face, and the remaining two to encode the barycentric coordinates within the face.

In the second pass, we scan the frame buffer. For each pixel in the virtual camera we incrementally compute the direction $\boldsymbol{\omega}$ of the incoming (viewing) ray using a single 3-vector addition at each pixel. We compute $\widetilde{\boldsymbol{\omega}}$ by reflecting through the surface normal $\mathbf{n}(u_i)$ at $\varphi(u_i)$. Finally, we evaluate $\widetilde{L}(u_i, \widetilde{\boldsymbol{\omega}})$ by looking up the lumisphere associated with u_i and evaluating the piecewise-linear function in direction space. These operations can be done quickly with just a few floating-point operations.

5.2 View-dependent refinement of geometry

One feature of the surface light field representation is the decoupling of the surface geometry from the light field. For best results, we can render the surface geometry at the highest resolution during the first pass of the rendering algorithm, but this can be costly. Alternatively, we can render a simplified mesh (*e.g.*, the embedding of the base mesh triangles in $\mathbb{R}^3$) and still achieve a compelling result because surface light fields, like bump-mapped lighting, suggest more geometric detail than is actually present. However, this simplified mesh introduces some distortion; moreover, the coarse silhouettes are often objectionable. Instead, we have explored a middle ground between those two extremes: *view-dependent refinement* of the subdivision-connectivity surface mesh.

Most current methods for real-time, view-dependent simplification of geometry, such as those presented by Hoppe [15] and Xia and Varshney [32], employ progressive mesh representations and adapt the level of detail using edge collapses and vertex splits. For a texture-mapped surface, however, these operations can cause considerable parametric distortion, especially near the boundaries of parameter domains, placing significant constraints on the simplification [5]. Therefore, we restrict the mesh used for rendering to have four-to-one subdivision connectivity [18], and refine the mesh by adding and removing lazy wavelets [4]. This allows us to modify the geometric detail almost independently of the parameterization.

We approximate the map $\varphi : K_0 \rightarrow M \subset \mathbb{R}^3$ by a piecewise-linear map $\varphi_r : K_r \rightarrow \mathbb{R}^3$ on the simplicial complex K_r obtained by applying r four-to-one subdivisions to the base complex K_0 and setting $\varphi_r(v) = \varphi(v)$ for each vertex v of K_r. The subdivision level r is a user-defined parameter ($r = 4$ in Figure 9). We then compute the lazy-wavelet expansion of φ_r, expressing it as a sum of hat functions. Adapting the mesh can now be formulated as finding a partial sum of those hat functions, satisfying a set of view-dependent properties.

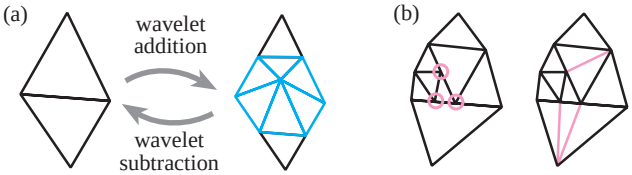


Figure 10 (a) Lazy wavelet addition and subtraction. The support of the added hat function is shown in blue. (b) T-vertices (circled in red) are eliminated by adding edges.

The retriangulation procedure is an incremental algorithm that exploits frame-to-frame coherence, similar to algorithms described by Hoppe [15] and Xia and Varshney [32]. To compute the approximation for a frame, we begin with the approximation computed for the previous frame and modify it by applying the *lazy-wavelet addition* and *lazy-wavelet subtraction* operations, illustrated in Figure 10(a), according to view-dependent criteria. To reduce the appearance of "popping," we spread the visual effect of each operation over time by geomorphing [14]. In a second quick pass over the mesh, we add temporary edges to eliminate cracks caused by "T-vertices," as shown in Figure 10(b).

Our criteria for wavelet addition and subtraction are the same three view-dependent refinement criteria described by Hoppe [15]: (1) removing wavelets that are completely backfacing, (2) removing wavelets lying completely outside the view frustum, and (3) maintaining a screen-space error bound. To accelerate computation of screen-space error, we construct, in preprocessing, a bounding volume around the set of geometric error vectors associated with a wavelet addition. We have found that an ellipsoid aligned to the surface normal generally provides a tighter bound than the shape used by Hoppe, while not adding significantly to the cost of projecting the error volume. Because coarse silhouettes tend to be more noticeable than interior distortion, we use a smaller error tolerance near the silhouette [19]. Finally, to reduce the number of wavelet addition and subtraction operations that must be considered, we enforce one additional property: A hat function at level $\ell \leq r$, centered at an edge of $K_{\ell-1}$, may appear in the sum only if the hat functions centered at the endpoints of the edge appear in the sum.

The results of view-dependent level-of-detail are illustrated in Figure 9, showing a close-up of the elephant's trunk. While achieving high accuracy, the top renderings using uniform subdivision render fairly slowly due to the large number of triangles. The bottom renderings, using the view-dependent level-of-detail algorithm with error threshholds set to match the fine geometry renderings, are obtained with far fewer triangles yielding moderately improved frame rates with little visual difference.

The close-up views shown in Figure 9 benefit greatly from the view frustum test, which causes a considerable fraction of the model to be coarsened. In the other common case, where the entire model is visible, using view-dependent level-of-detail does not give as significant of a performance benefit, but it does no worse than a static model. Of course, if the model is very distant, the level-of-detail algorithm will generate a very coarse approximation.

6 Editing

Just as the decoupling of surface geometry and the light field allows us to refine the geometry independently, we are now able to perform editing operations that are not commonly possible in an image-based rendering context. In this section we describe three such operations: lumisphere editing, rotating the object relative to its environment, and deforming the geometry.

By performing simple image processing directly on the lumispheres, we can simulate changes in surface properties, such as sharpening of specular highlights. We demonstrate this particular operation in Figures 11(a) and (b), where the highlights in the original rendering (a) have been brightened and sharpened (b). We achieve this effect by applying Perlin's bias function [25] to the values of every lumisphere. For compressed surface light fields, we can quickly approximate this by adjusting the prototype lumispheres. (For principal function analysis, this is only an approximation because the bias function is non-linear.)

The other two editing operations we illustrate, rotation of geometry relative to its environment and general deformation, fit into one conceptual framework: a transformation is applied to define a new surface. The new surface can be represented by a modified embedding of the base mesh $\varphi' : K_0 \rightarrow \mathbb{R}^3$. (Rotation is just a special case of general deformation.)

Our goal then is to compute the corresponding surface light field $L'(u, \omega)$, and our solution is operationally very simple. We compute the new surface normal field $\mathbf{n}'(u)$ and then set $L'(u, \omega) = \widetilde{L}(u, \widetilde{\omega}')$, where $\widetilde{\omega}'$ is the reflection of ω through the new normal.

Figures 11 and 12 demonstrate the geometric edits. Figure 11(a) shows the original elephant; (c) and (d) show the elephant rotated

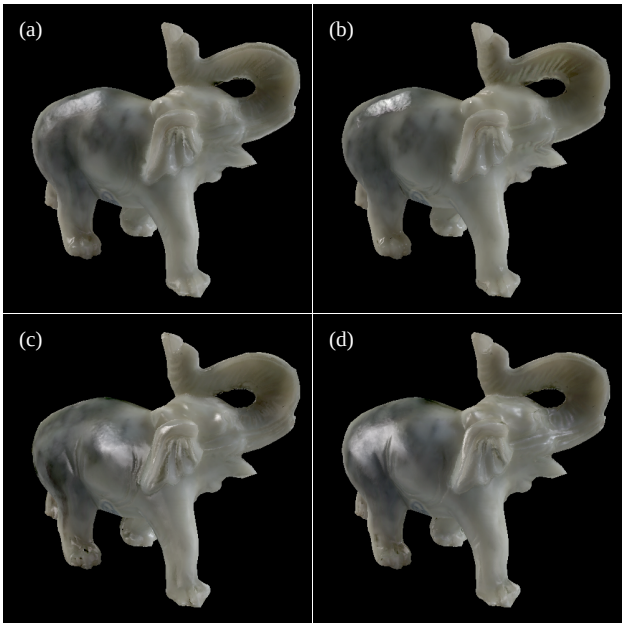


Figure 11 Editing operations applied to the elephant. (a) Original elephant. (b) Sharper and brighter highlights. (c) Environment rotated. (d) Environment rotated to another position.

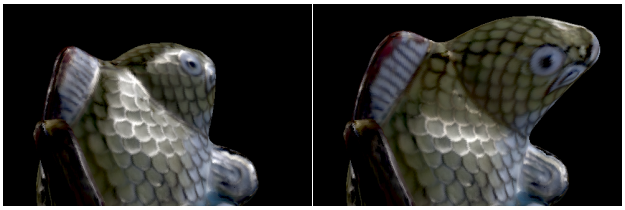


Figure 12 A volumetric deformation applied to the fish. (Original on left.)

relative to its environment. Figure 12(a) shows the fish as it was originally; Figure 12(b) shows it after deformation, with its head bent to the side.

Our method for computing the new surface light field L' from the original L is justified if the environment is infinitely far away, if there is no occlusion, shadowing or interreflection, and if the BRDFs for all surface points are reflective. These are the same assumptions that motivate our reflection transformation described in Section 4.4. Even if all of these requirements are met, there is an additional problem. For any grid point $u_i \in K_0$, the camera directions represented in the data lumisphere fall inside a hemisphere. After editing, however, there will in general be viewing directions that require values of $L'(u_i, \omega)$ for directions outside this hemisphere. In fact, if we rotate the object by 180 degrees, we will need values exactly on the opposite hemisphere. Operationally, however, inferring these values is not a problem. The estimation techniques guarantee that lumispheres are well-defined everywhere, albeit not necessarily realistic.

7 Future work

We envision a number of areas for future work:

Combining function quantization and principal function analysis. Our two compression methods can be considered extrema of a spectrum: Function quantization fits the data by a collection of 0-dimensional spaces, whereas principal function analysis uses a

single higher-dimensional space. We could do both: fit a collection of higher dimensional spaces. That approach might work well if the data lumispheres lie on a low-dimensional curved manifold in lumisphere space.

Wavelet representation of a surface light field. Constructing a wavelet expansion of the surface light field $L(u, \omega)$ might result in better compression than function quantization or principal function analysis, and would support progressive transmission and performance-tuned rendering [4].

Hole filling using texture synthesis. We have no method for assigning lumispheres to surface points not visible in any of the cameras, like those on the bottom of the elephant in Figure 7. A texture synthesis algorithm, suitably extended to operate on lumispheres instead of colors and with textures defined on general surfaces instead of the plane, could be used to fill these holes.

Acknowledgements

We would like to thank Marc Levoy for the use of the Stanford spherical gantry and other equipment. This work was supported by an NSF grant (DMS-9803226) and an Osberg Family Fellowship, as well as industrial gifts from Intel, Microsoft, and Pixar.

References

[1] D. I. Azuma. Interactive Rendering of Surface Light Fields. Technical Report UW-CSE-2000-04-01, Department of Computer Science and Engineering, University of Washington, April 2000.

[2] P. J. Besl and N. D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

[3] B. Cabral, M. Olano, and P. Nemec. Reflection Space Image Based Rendering. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 165–170, August 1999.

[4] A. Certain, J. Popović, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive Multiresolution Surface Viewing. In *SIGGRAPH 96 Conference Proceedings*, Computer Graphics Annual Conference Series, pages 91–98, August 1996.

[5] J. Cohen, M. Olano, and D. Manocha. Appearance-Preserving Simplification. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 115–122. ACM SIGGRAPH, July 1998.

[6] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 303–312, August 1996.

[7] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20, August 1996.

[8] P. E. Debevec, Y. Yu, and G. D. Borshukov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. *Eurographics Rendering Workshop 1998*, pages 105–116, June 1998.

[9] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182, August 1995.

[10] H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau. Registration of Multiple Range Views for Automatic 3-D Model Building. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 581–586, June 1994.

[11] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[12] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54, August 1996.

[13] W. Heidrich and H.-P. Seidel. Realistic, Hardware-Accelerated Shading and Lighting. *Proceedings of SIGGRAPH 99*, pages 171–178, August 1999.

[14] H. Hoppe. Progressive Meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108, August 1996.

[15] H. Hoppe. View-Dependent Refinement of Progressive Meshes. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198, August 1997.

[16] A. W. F. Lee, W. Sweldens, P. Schroeder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 95–104, July 1998.

[17] M. Levoy and P. Hanrahan. Light Field Rendering. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42, August 1996.

[18] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[19] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208, August 1997.

[20] M. Magnor and B. Girod. Adaptive Block-Based Light Field Coding. *Proc. 3rd International Workshop on Synthetic and Natural Hybrid Coding and Three-Dimensional Imaging*, pages 140–143, September 1999.

[21] M. Magnor and B. Girod. Hierarchical Coding of Light Fields with Disparity Maps. *Proc. IEEE International Conference on Image Processing*, pages 334–338, October 1999.

[22] G. S. P. Miller, S. Rubin, and D. Ponceleon. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. *Eurographics Rendering Workshop 1998*, pages 281–292, June 1998.

[23] K. Nishino, Y. Sato, and K. Ikeuchi. Appearance compression and synthesis based on 3D model for mixed reality. In *Proceedings of IEEE ICCV'99*, pages 38 – 45, September 1999.

[24] K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-Texture Method: Appearance Compression based on 3D Model. *Proc. of Computer Vision and Pattern Recognition*, 1:618–624, June 1999.

[25] K. Perlin and E. M. Hoffert. Hypertexture. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):253–262, July 1989.

[26] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data. *Eurographics Rendering Workshop 1997*, pages 23–34, June 1997.

[27] S. M. Rusinkiewicz. A New Change of Variables for Efficient BRDF Representation. In *Eurographics Rendering Workshop 1998*, pages 11–22. Eurographics, June 1998.

[28] Y. Sato, M. D. Wheeler, and K. Ikeuchi. Object Shape and Reflectance Modeling from Observation. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 379–388, August 1997.

[29] G. Taubin. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, August 1995.

[30] R. Y. Tsai. An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.

[31] B. Walter, G. Alppay, E. P. F. Lafortune, S. Fernandez, and D. P. Greenberg. Fitting Virtual Lights For Non-Diffuse Walkthroughs. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 45–48, August 1997.

[32] J. C. Xia and A. Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In *IEEE Visualization '96*. IEEE, October 1996.

[33] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse Global Illumination: Recovering Reflectance Models of Real Scenes From Photographs. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 215–224, August 1999.