

Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields

Wei-Chao Chen *
University of North Carolina at Chapel Hill

Jean-Yves Bouguet †
Intel Corporation

Michael H. Chu ‡
Intel Corporation

Radek Grzeszczuk §
Intel Corporation

Abstract: A light field parameterized on the surface offers a natural and intuitive description of the view-dependent appearance of scenes with complex reflectance properties. To enable the use of surface light fields in real-time rendering we develop a compact representation suitable for an accelerated graphics pipeline. We propose to approximate the light field data by partitioning it over elementary surface primitives and factorizing each part into a small set of lower-dimensional functions. We show that our representation can be further compressed using standard image compression techniques leading to extremely compact data sets that are up to four orders of magnitude smaller than the input data. Finally, we develop an image-based rendering method, light field mapping, that can visualize surface light fields directly from this compact representation at interactive frame rates on a personal computer. We also implement a new method of approximating the light field data that produces positive only factors allowing for faster rendering using simpler graphics hardware than earlier methods. We demonstrate the results for a variety of non-trivial synthetic scenes and physical objects scanned through 3D photography.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—color, shading, shadowing, texture I.4.8 [Image Processing And Computer Vision]: Scene Analysis—color, shading I.2.10 [Artificial Intelligence]: Scene Analysis—3D/stereo scene analysis, texture

Keywords: Image-based Rendering, Texture Mapping, Compression Algorithms, Rendering Hardware.

1 Introduction

The recent proliferation of inexpensive but powerful graphics hardware and new advances in digital imaging technology are enabling novel methods for realistic modeling of the appearance of physical objects. On the one hand, we see a tendency to represent complex analytic reflectance models with their sample-based approximations that can be evaluated efficiently using new graphics hardware features [15; 16; 17]. On the other hand, we are witnessing a proliferation of image-based rendering and modeling techniques

* e-mail: ciao@cs.unc.edu

† e-mail: jean-yves.bouguet@intel.com

‡ e-mail: michael.h.chu@intel.com

§ e-mail: radek.grzeszczuk@intel.com

Copyright © 2002 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212-869-0481 or e-mail permissions@acm.org).
© 2002 ACM 1-58113-521-1/02/0007 \$5.00



Figure 1: A combination of synthetic and scanned objects rendered using light field mapping. Complex, physically realistic reflectance properties of this scene are preserved.

[26; 22; 14; 4] that attempt to represent the discrete radiance data directly in the sample-based format without resorting to the analytic models at all. These techniques are popular because they promise a simple acquisition and an accurate portrayal of the physical world. The approach presented here combines these two trends. Similar to other image-based methods, our approach produces a sample-based representation of the surface light field data. Additionally, the proposed representation can be evaluated efficiently with the support of existing graphics hardware.

1.1 Overview of Light Field Mapping Approach

A surface light field is a 4-dimensional function $f(r, s, \theta, \phi)$ that completely defines the outgoing radiance of every point on the surface of an object in every viewing direction. The first pair of parameters of this function (r, s) describes the surface location and the second pair of parameters (θ, ϕ) describes the viewing direction. A direct representation and manipulation of surface light field data is impractical because of the large size. Instead, we propose to approximate the light field function as a sum of a small number of products of lower-dimensional functions

$$f(r, s, \theta, \phi) \approx \sum_{k=1}^K g_k(r, s) h_k(\theta, \phi). \quad (1)$$

We show that it is possible to construct the approximations of this form that are both compact and accurate by taking advantage of the spatial coherence of the surface light fields. We accomplish this by partitioning the light field data across small surface primitives and building the approximations for each part independently. The partitioning is done in such a way as to ensure continuous approximations across the neighboring surface elements. Because the

functions $g_k(r, s)$ and $h_k(\theta, \phi)$ encode the light field data and are stored in a sampled form as texture maps, we call them the *light field maps*. Similarly, we refer to the process of rendering from this approximation as *light field mapping*. In the remainder of the paper we will refer to functions $g_k(r, s)$ as the *surface maps* and functions $h_k(\theta, \phi)$ as the *view maps*.

As shown in earlier work, similar types of factorizations can be used to produce high quality renderings of objects with complex surface reflectance properties at interactive frame rates [15; 17; 16]. Most previous work, however, assumes that the surface of the object has uniform reflectance properties, which greatly simplifies the problem. Our method shows for the first time how to compute such factorization for surfaces where each point has a unique reflectance property. This generalization makes light field mapping ideally suited for modeling and rendering of physical objects and scenes scanned through 3D photography.

1.2 Contributions

We make the following contributions to analysis and representation of image-based data and to hardware-accelerated rendering of image-based models.

Data Partitioning: We introduce a novel type of partitioning of the light field data that ensures a continuous approximation across individual triangles. It represents the light field data for each triangle as a linear blend of 3 light fields corresponding to the vertices of the triangle. See Section 2.1.

Approximation Through Factorization: We introduce a new type of factorization of light field data. Dense and uniform sampling of the view-dependent information, enabled through the use of view maps $h_k(\theta, \phi)$, ensures correct image synthesis from any camera location. See Section 2.2.

Positive Factorization: We introduce a new method of approximating the light field data that uses non-negative matrix factorization [20] to produce positive only factors. This method allows for faster rendering using generic graphics hardware at high approximation quality. It is significantly easier to implement than homomorphic factorization [25]. See Section 2.2.

Hardware-Accelerated Rendering: We develop a fully hardware-accelerated rendering routine that can visualize the light field data from the proposed, compact representation at highly interactive frame rates. The rendering algorithm is extremely simple to implement, since it reduces to a multi-pass texture mapping operation. Rendering quality can be improved progressively by increasing the number of rendering passes. See Section 3.

Compression: We report the highest compression of surface light fields to date. Since the light field maps are in essence collections of images that themselves exhibit redundancy, they can be further compressed using standard image compression techniques. This compression combined with the redundancy reduction achieved through the approximation results in extremely high compression ratios. Our method routinely approaches 4 orders of magnitude compression ratio and allows for hardware-accelerated rendering directly from the compressed representation. Additional compression is possible for transmission and storage purposes. See Section 4.

1.3 Related Work

Traditionally, computer graphics researchers have favored analytic reflectance models for their intuitive simplicity, compactness and flexibility [37; 7; 38; 29]. Although simple to represent, these models are often costly to compute and difficult to develop. Several approaches have been developed to approximate reflectance functions with a set of predefined basis function [34; 18; 19] but without much consideration for rendering efficiency. Inverse rendering

methods [33; 40; 9; 2] approximate scene attributes, such as lighting and surface reflectance, by fitting analytic models to sample-based radiance data. Ramamoorthi and Hanrahan [31] propose a signal processing framework for inverse rendering. Light field mapping is a related method that uses a factor analysis approach rather than a signal processing approach.

Image-based methods [26; 22; 14; 4; 24] represent the radiance data directly in the sample-based format. Compression of light field data is an important area of research. Levoy and Hanrahan [22] use VQ [13], Magnor and Girod [23] have developed a series of disparity-compensated hybrid light field codecs. Chai *et al.* [5] use spectral analysis to compute the “minimum sampling rate” for light field rendering and show that scene depth information significantly decreases this rate. Miller *et al.* [27] propose a method of rendering surface light fields from input images compressed using JPEG-like compression. Malzbender *et al.* [24] approximate a sequence of images captured under varying lighting conditions using biquadratic polynomials. The representation is very compact and can be implemented in hardware.

The work on sample-based approximations for some of the analytic models listed above includes [15; 17; 21]. Kautz and McCool [16] propose a method for hardware assisted rendering of arbitrary BRDFs through their decomposition into a sum of 2D separable functions that is based on an idea proposed by Fournier [12]. Our application is fundamentally different from this work, since it allows each surface point to have different reflectance properties and is therefore ideally suited for modeling and rendering of objects and scenes scanned through 3D photography. Homomorphic factorization of McCool *et al.* [25] generates a BRDF factorization with positive factors only, which are easier and faster to render on the current graphics hardware, and deals with scattered data without a separate resampling and interpolation algorithm. We present a novel method for factorization of light field data that also produces positive only factors using non-negative matrix factorization [20]. Lensch *et al.* [21] reconstruct a spatially varying BRDF from a sparse set of images. First, the object is split into cluster with different BRDF properties, then a set of basis BRDFs is generated for each cluster, finally the original samples are reprojected into the space spanned by the basis BRDFs. Data factorization in this case is extremely time consuming and rendering is not real-time.

The work on surface light fields includes view-dependent texture mapping [10; 11; 30]. Additionally, Wood *et al.* [39] use a generalization of VQ and PCA (principal component analysis) [1] to compress surface light fields and propose a new rendering algorithm that displays compressed light fields at interactive frame rates. Their 2-pass rendering algorithm interpolates the surface normals using Gouraud shading producing incorrect results for large triangles. To alleviate the problem they introduce a view-dependent geometry refinement at the cost of increasing the complexity of the renderer.

Nishino *et al.* [28] propose the eigen-texture method that also performs factorization on image data. Their basic theoretical approach is similar to ours however it applies only to a 3D subset of the 4D light field and cannot be directly extended to the full 4D data. Additionally, our approach uses a vertex-based partitioning which ensures a continuous reconstruction, we explore a positive factorization technique, and we develop a fully hardware-accelerated rendering of surface light fields directly from our representation.

2 Approximation through Factorization

We start the description of the approximation method with a novel partitioning scheme that ensures continuous approximations throughout the surface of the model.

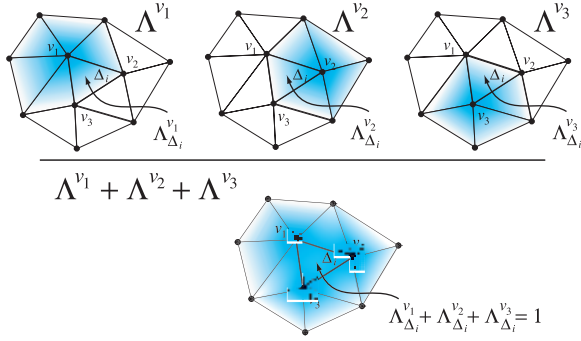


Figure 2: The finite support of the hat functions Λ^{v_j} around vertex v_j , $j = 1, 2, 3$. $\Lambda_{\Delta_i}^{v_j}$ denotes the portion of Λ^{v_j} that corresponds to triangle Δ_i . Functions Λ^{v_1} , Λ^{v_2} and Λ^{v_3} add up to one inside Δ_i .

2.1 Surface Light Field Partitioning

We assume that the geometry of the models is represented as a triangular mesh. An obvious partitioning of the light field function is to split it between individual triangles. Unfortunately, an approximation of surface light field partitioned this way results in visible discontinuities at the edges of the triangles. We demonstrate this effect in the video accompanying the paper. To eliminate the discontinuities across triangle boundaries we propose to partition surface light field data around every vertex. We refer to the surface light field unit corresponding to each vertex as the *vertex light field* and for vertex v_j denote it as $f^{v_j}(r, s, \theta, \phi)$. Partitioning is computed by weighting the surface light field function

$$f^{v_j}(r, s, \theta, \phi) = \Lambda^{v_j}(r, s) f(r, s, \theta, \phi) \quad (2)$$

where Λ^{v_j} is the barycentric weight of each point in the ring of triangles relative to vertex v_j . Because of their shape, the weighting functions are often referred to as the *hat functions*. The top row of Figure 2 shows hat functions Λ^{v_1} , Λ^{v_2} , Λ^{v_3} for three vertices v_1 , v_2 , v_3 of triangle Δ_i . As shown at the bottom of Figure 2, these 3 hat functions add up to unity inside triangle Δ_i . Therefore, Equation (2) defines a valid partitioning of the surface light field because the sum of all vertex light fields equals to the original light field data.

In the final step of vertex-centered partitioning we reparameterize each vertex light field to the local coordinate system of the vertex. To this end, we define the viewing direction angles θ and ϕ as the azimuth and elevation angles of the viewing direction vector in the local reference frame of the vertex. We define the vertex reference frame in such a way that its z -axis is parallel to the surface normal at the vertex. We use the same letters to denote the local parameters of the vertex light field in order to simplify the notation.

2.2 Vertex Light Field Approximation

Vertex-centered partitioning of light field data allows us to approximate each vertex light field independently while maintaining continuity across the whole model. Analogous to Equation (1), we approximate each vertex light field as

$$f^{v_j}(r, s, \theta, \phi) \approx \sum_{k=1}^K g_k^{v_j}(r, s) h_k^{v_j}(\theta, \phi). \quad (3)$$

We present two methods of approximating vertex light fields: PCA (principal component analysis) [1] and NMF (non-negative matrix factorization) [20]. Both of these methods approximate the light field data as a linear combination of a small number of basis images but each one produces an approximation with a different set of

features. For example, each PCA-based basis image gives a global approximation of the data. This means that we can use a subset of them to produce a consistent approximation, since adding successive basis images simply improves the accuracy preceding basis images. NMF, on the other hand, produces basis images that form a parts-based representation and therefore all of them need to be used to produce a consistent approximation of the input data. PCA-based approximation is therefore more *progressive* than NMF-based approximation. However, PCA allows the approximation factors to be of arbitrary sign. This makes rendering more difficult and requires more specialized graphics hardware. See Section 3.1 for details. NMF does not allow negative values in the factors resulting in an approximation that is much easier and faster to render on generic graphics hardware.

We will use the matrix factorization framework to describe how PCA and NMF construct the light field data approximations. We first discretize the vertex light field function $f^{v_j}(r, s, \theta, \phi)$ into a 4-dimensional grid $f^{v_j}[r_p, s_p, \theta_q, \phi_q]$, where index $p = 1, \dots, M$ refers to the discrete values $[r_p, s_p]$ describing the surface location within the triangle ring of vertex v_j , and index $q = 1, \dots, N$ refers to the discrete values $[\theta_q, \phi_q]$ of the two viewing angles. We then rearrange the discretized vertex light field into the matrix

$$\mathbf{F}^{v_j} = \begin{bmatrix} f^{v_j}[r_1, s_1, \theta_1, \phi_1] & \cdots & f^{v_j}[r_1, s_1, \theta_N, \phi_N] \\ \vdots & \ddots & \vdots \\ f^{v_j}[r_M, s_M, \theta_1, \phi_1] & \cdots & f^{v_j}[r_M, s_M, \theta_N, \phi_N] \end{bmatrix}, \quad (4)$$

where M is the total number of surface samples inside the triangle ring and N is the total number of views for each surface sample. We refer to matrix \mathbf{F}^{v_j} as the *vertex light field matrix*. Each column of this matrix represents the appearance of the vertex ring under a different viewing direction. A detailed discussion of resampling the irregular input light field data can be found in Section 5.

Both PCA and NMF construct approximate factorizations of the form

$$\tilde{\mathbf{F}}^{v_j} = \sum_{k=1}^K \mathbf{u}_k \mathbf{v}_k^T \quad (5)$$

where \mathbf{u}_k is a vectorized representation of discrete surface map $g_k^{v_j}[r_p, s_p]$ and \mathbf{v}_k is a vectorized representation of discrete view map $h_k^{v_j}[\theta_q, \phi_q]$. The differences between the two methods arise from the constraints imposed on the matrix factors \mathbf{u}_k and \mathbf{v}_k . PCA constrains \mathbf{u}_k 's to be orthonormal and \mathbf{v}_k 's to be orthogonal to each other. NMF, on the other hand, does not allow negative entries in the matrix factors \mathbf{u}_k and \mathbf{v}_k . Singular value decomposition can be used to compute PCA factorization. Since only the first few summation terms of the factorization are needed, one can use an efficient algorithm that computes partial factorization. See Appendix for details on how to compute the factorizations.

Partitioning of light field data into vertex light fields ensures that in the resulting approximation each triangle shares its view maps with the neighboring triangles. Therefore, even though we decompose each vertex light field independently, we obtain an approximation that is continuous across triangles regardless of the number of approximation terms K . Let $g^{v_j}[r_p, s_p]$ be the surface map and $h^{v_j}[\theta_q, \phi_q]$ be the view map corresponding to one approximation term of vertex light field $f^{v_j}[r_p, s_p, \theta_q, \phi_q]$. Let $\tilde{f}^{\Delta_i}[r_p, s_p, \theta_q, \phi_q]$ denote the corresponding approximation term of the light field data for triangle Δ_i . The following equality holds

$$\tilde{f}^{\Delta_i}[r_p, s_p, \theta_q, \phi_q] = \sum_{j=1}^3 g_{\Delta_i}^{v_j}[r_p, s_p] h^{v_j}[\theta_q, \phi_q] \quad (6)$$

where index j runs over the three vertices of triangle Δ_i and $g_{\Delta_i}^{v_j}[r, s]$ denotes the portion the surface map corresponding to the triangle

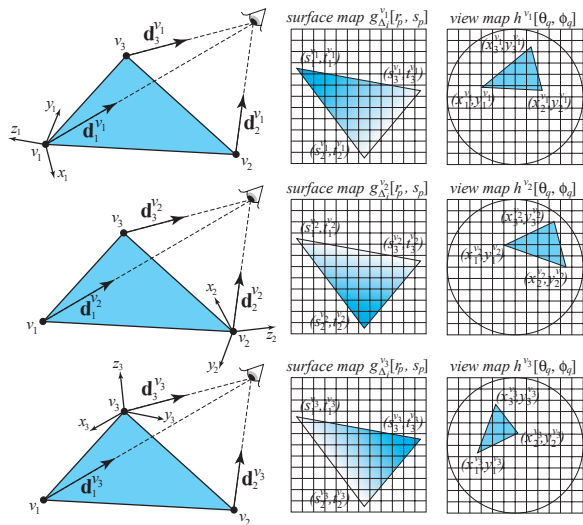


Figure 3: Light field maps for one approximation term of one triangle. Vertex reference frames are shown in the left column.

Δ_i . Note that this equality holds for all approximation terms. Next, we show how this property can be used to develop a very simple and efficient rendering algorithm.

3 Rendering Algorithm

Our rendering algorithm takes advantage of the property of vertex-centered partitioning, described by Equation (6), which says that the light field for each triangle can be expressed independently as a sum of its 3 vertex light fields. This allows us to write a very efficient rendering routine that repeats the same sequence of operations for each mesh triangle. Additionally, since each light field approximation term is also evaluated in exactly the same way, we only need to explain how to evaluate one approximation term of one triangle.

Figure 3 shows six light field maps used in Equation (6) to compute one approximation term of light field for triangle Δ_i . The middle column shows surface maps $g_{\Delta_i}^{v_j}[r_p, s_p]$. The pixels covered by the shaded triangle correspond to the points inside triangle Δ_i where we sampled the light field function. We will describe the texture coordinates of these points as (s, t) . Note that due to partitioning the pixels of the surface maps are weighted, as indicated in the figure by gradient shading, but this does not alter the rendering algorithm in any way.

The right column shows view maps $h^{v_j}[\theta_q, \phi_q]$. In each image, the pixels inside the circle correspond to the orthographic projection of the hemisphere of viewing directions, expressed in the local coordinate system xyz of vertex v_j , onto the plane xy shifted and scaled to the range $(0, 1)$. We will describe the texture coordinates of these points as (x, y) . This projection allows a simple texture coordinate computation

$$x = (\mathbf{d} \cdot \mathbf{x} + 1)/2, \quad y = (\mathbf{d} \cdot \mathbf{y} + 1)/2 \quad (7)$$

where \mathbf{d} represents the normalized local viewing direction and vectors \mathbf{x} and \mathbf{y} correspond to the axes of the local reference frame. Other transformations from 3D directions to 2D maps are possible [15] but we found the one described here efficient and accurate.

Rendering of light field approximations is done using texture mapping. Based on where the camera is located, the rendering algorithm needs to access a different 2D subset of the 4D light field function. This is done by recomputing the view map coordinates $(x_i^{v_j}, y_i^{v_j})$ every time the camera moves. To this end, we apply Equations (7) to vectors $\mathbf{d}_i^{v_j}$, which represent the viewing directions to

vertex v_i expressed in the reference frame of vertex v_j . This results in 3 texture fragments shown in the right column of Figure 3. Note that the texture coordinates are different for each fragment because we are using a different reference frame to compute them. Since the surface map texture coordinates $(s_i^{v_j}, t_i^{v_j})$ do not depend on the viewing angle, they do not need to be recomputed when the camera moves.

Evaluating one complete approximation term is equivalent to multiplying pixel-by-pixel the image projections of the surface map and view map texture fragment pairs for the 3 vertex light fields of the triangle, each shown in a separate row of Figure 3, and adding the results together. The multiple term approximation of each triangle light field is computed by simply adding the results of each approximation term.

3.1 Hardware Accelerated Implementation

In this section we discuss efficient implementations of light field mapping using specific hardware features such as multitexturing and extended color range. Recall that one of the fundamental operations of the proposed rendering algorithm was the pixel-by-pixel multiplication, or modulation, of the surface map fragment by the corresponding view map fragment. Multitexturing hardware support enables us to compute the modulation of multiple texture fragments very efficiently in one rendering pass. Consequently, for the NMF-based approximation of light field data, which produces strictly positive light field maps, we need at most 3 rendering passes to render each light field approximation term using multitexturing graphics hardware that supports 2 texture sources. Without multitexturing hardware support we can implement the rendering algorithms described above using an accumulation buffer, though significantly less efficiently.

For the PCA-based approximation, which in general will produce light field maps that contain negative values, rendering can benefit from graphics hardware that permits a change to the limits of the color range from the traditional $[0, 1]$ range to $[min, max]$, e.g., $[-1, 1]$. NVIDIA cards support extended color range through the register combiners extension [35] but they still clamp the negative output to zero. This means that we can render PCA-based approximations using register combiners at the expense of doubling the rendering passes as follows. Let M be the result of modulation of two texture fragments A and B using register combiners. Let M_+ be the result of clamped modulation of fragments A and B and let M_- be the result of modulating $-A$ and B the same way. We can compute M by subtracting the outputs of the two modulations $M = M_+ - M_-$. Even when the graphics hardware does not support any extended pixel range we can still render PCA-based approximations, though the number of rendering passes required to evaluate one approximation term increases to 4.

PCA-based matrix factorization requires that we subtract the *mean view* from the light field matrix before performing the decomposition. It changes the rendering routine only slightly—we first texture map the triangle using its mean view and then add the approximation terms of the modified light field matrix exactly as it was done before. One of the advantages of extracting the mean view and rendering it separately is that in many cases this view represents an approximation to the diffuse component of the surface material and, as such, is interesting to visualize independently from the view-dependent component of the light field data.

4 Compression of Light Field Maps

Approximation through matrix factorization described in Section 2.2 can be thought of as a compression method that removes local redundancy present in the light field data. The compression ratio

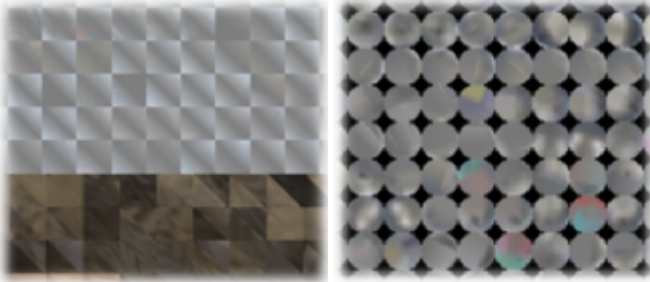


Figure 4: Surface maps (left) and view maps (right) computed using PCA-based approximation for the bust model. The lower portion of the left image represents the extracted mean textures. All other light field maps are stored in $[-1, 1]$ range, where -1 is mapped to black.

of this method is closely related to the size of the surface primitives used for partitioning. On the one hand, a dense mesh with many vertices will have unnecessarily large number of view maps. On the other hand, a coarse mesh requires more approximation terms for the same approximation quality. See Section 6.1 for the analysis of the relationship of the compression ratio and the approximation quality to the triangle size. Currently, we choose the size of triangles empirically so that we obtain about two orders of magnitude compression ratio through approximation while maintaining high quality without using many approximation terms. Section 6.2 gives a detailed analysis of compression ratios obtained through the approximation.

Figure 4 shows, for the bust model, surface maps (left) and view maps (right) computed using PCA-based approximation. It is easy to see that the light field maps are still redundant. First, the individual maps are similar to each other, suggesting global redundancy of the data. Second, some of the light field maps have very little information content and can be compressed further using a variety of existing image compression techniques.

Figure 5 gives an overview of the different levels of compression we apply to the light field data. For optimal run-time performance, compressed light field maps need to fit entirely in the texture memory cache and be decompressed on-the-fly during rendering. This process should only introduce minimal run-time memory overhead. In the following paragraphs we discuss several techniques that satisfy these criteria. Other image compression techniques can be used to further reduce the offline storage size, as shown in Figure 5, but are not discussed in the paper.

Data redundancy *across* individual light field maps can be reduced effectively using VQ [13]. Our implementation represents each triangle surface map $g_{\Delta_i}^v[r_p, s_p]$ and each view map $h^v[\theta_q, \phi_q]$ as a vector. The algorithm groups these vectors based on their size and generates a separate codebook for every group. We initialize the codebooks using either pairwise nearest neighbor or split algorithm. The codebooks are improved by the generalized Lloyd algorithm utilizing square Euclidean distance as the cost function. We then store the resulting codebooks as images. The rendering algorithm from VQ-compressed images does not change in any way—it simply indexes a different set of images.

We use either a user-specified compression ratio or the average distortion to drive the VQ compression. With the distortion-driven algorithm, the light field maps corresponding to the higher approximation terms exhibit more redundancy and thus are often compressed into a smaller codebook. In practice, light field maps can be compressed using VQ by an order of magnitude without significant loss of quality.

Data redundancy *within* individual light field maps can be reduced efficiently using block-based algorithms. One such method, called S3TCTM, is often supported on commodity graphics cards

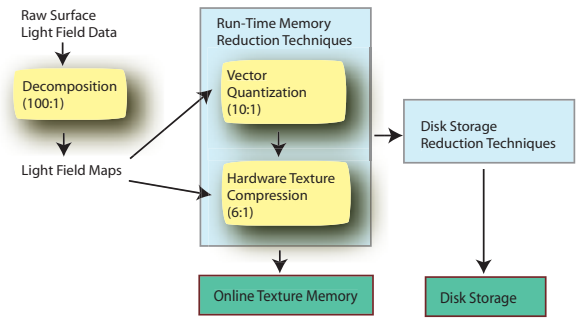


Figure 5: Compression Overview. The number under each technique describes its approximate compression ratio.

today. It offers compression ratios between 6:1 and 8:1 and can be cascaded with VQ for further size reduction. Limited by hardware implementation cost, these algorithms are not very sophisticated in nature. For example, the S3TC algorithm divides the textures into 4-by-4 texel blocks and performs color interpolation within each block. Since the algorithm uses blocks that are smaller than most light field maps, when compared to VQ, it generates noisier images but it preserves the specularities better.

For PCA-based approximation, we observe that the light field maps associated with higher approximation terms contain lower spatial frequency. To test if we can reduce the resolution of these light field maps without significant impact on the rendering quality, we implemented a simple method that subsamples the image resolution uniformly within each term. The results, although not reported in the paper, proved effective. A more sophisticated approach would apply selective resolution reduction to each light field map, or simply reduce light field matrix resolution during resampling.

5 Implementation Details

This section describes the following three implementation related issues: the acquisition of the geometry and the radiance data of physical objects, resampling of the radiance data, and tiling of the light field maps.

5.1 Data Acquisition

Figure 6 gives an illustration of the overall data acquisition process. A total of N_I ($200 < N_I < 400$) images are captured with a handheld digital camera (Fig. 6a). Figure 6b shows one sample image. Observe that the object is placed on a platform designed for the purpose of automatic registration. The color circles are first automatically detected on the images using a simple color segmentation scheme. This provides an initial guess for the position of the grid corners that are then accurately localized using a corner finder. The precise corner locations are then used to compute the 3D position of the camera relative to the object. This may be done, given that the camera has been pre-calibrated. The outcome of this process is a set of N_I images captured from known vantage points in 3D space.

The object geometry is computed using a structured lighting system consisting of a projector and a camera. The two devices are visible in Figure 6a. Figure 6c shows an example camera image acquired during scanning. The projector is used to project a translating striped pattern onto the object. A similar temporal analysis employed by Curless *et al.* [8] and Bouguet *et al.* [3] is used for accurate range sensing. In order to facilitate scanning, the object is painted with white removable paint, a technique especially useful when dealing with dark, highly specular or semi-transparent objects. We take between 10 and 20 scans to completely cover the

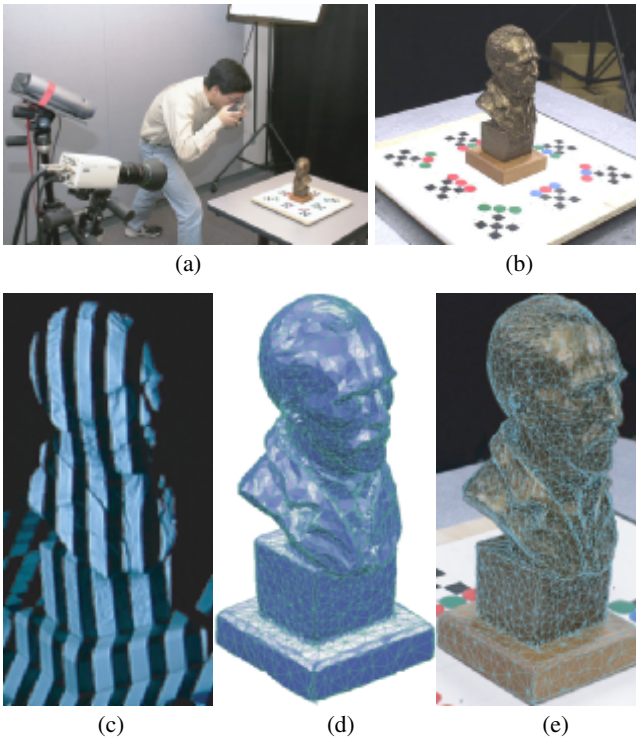


Figure 6: Data acquisition. (a) The user is capturing approximately 300 images of the object under a fixed lighting condition using a hand-held digital camera. (b) One sample image. The color circles guide the automatic detection of the grid corners that are used for computing the 3D location of the camera. (c) The painted object being scanned using the structured lighting system. (d) The complete 3D triangular mesh consisting of 7228 triangles constructed from 20 scans. (e) The projection of the triangular mesh onto image (b).

surface geometry of the object. Between two consecutive scans, the object is rotated in front of the camera and projector by about 20 degrees. For that purpose, the calibration platform has been designed to rotate about its central axis. The individual scans are automatically registered together in the object reference frame using the same grid corners previously used for image registration. The resulting cloud of points (approx. 500,000 points) is then fed into mesh editing software to build the final triangular surface mesh shown in Figure 6d. Since we use the same calibration platform for both image and geometry acquisition, the resulting triangular mesh is naturally registered to the camera images. Figure 6e shows the projection of the mesh onto the camera image displayed in Figure 6b illustrating the precise alignment of the geometry and the images. The image reprojection error is less than one pixel.

5.2 Data Resampling

Before we start resampling, we need to determine the visible cameras for each mesh triangle. A triangle is considered visible only if it is completely unoccluded. Repeating this process for all N_f images results in a list of visible views for each triangle. The visible triangle views correspond to a set of texture patches of irregular size captured from various viewing directions. We now explain the two-step process of resampling the irregularly sized views into a 4-dimensional grid.

Recall that the approximation algorithms described in Section 2.2 assume light field data in the form of matrices. This requires that we normalize each texture patch to have the same shape and size as the others. The size of the normalized patch is chosen to

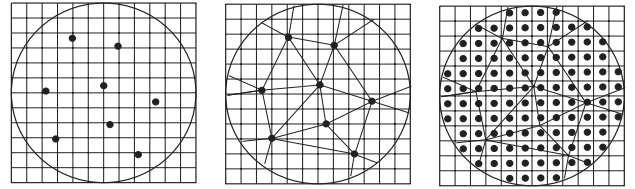


Figure 7: Resampling of views. Projection of original views (left), Delaunay triangulation of projected views (center), uniform grid of views computed by blending the original set of views (right).

be equal to the size of the largest view. Resampling is done using bilinear interpolation of the pixels in the original views.

At this stage, we have a uniform number of samples for each triangle view but the sampling of views is still irregular. Having a regular grid of viewing directions in the light field matrix is very important for two reasons. As was shown in [6], factorization of the fully resampled matrix results in a more precise approximation and encodes the view-dependent information allowing us to synthesize a correct image for any camera location using the rendering algorithm that was described in Section 3. We will use Figure 7 to explain the view resampling step. Let vectors $\mathbf{d}_{\Delta_i}^{v_j}$ be the viewing directions for the visible views of a given triangle expressed in the reference frame of vertex v_j . We start by projecting these directions onto the xy plane of the reference frame of vertex v_j using Equations (7). The result of this operation is a set of texture coordinates (Figure 7, left). Next we perform the Delaunay triangulation of these coordinates (Figure 7, middle) and compute the regular grid of views by blending the original triangle views using the weighting factors obtained from the triangulation (Figure 7, right).

We assume that the viewing direction is constant across a triangle, since the distance of the camera from the object during light field capture is generally quite large compared to the size of the triangles. The typical resolution of the view grid is $32 \times 32 = 1024$, but it can vary from triangle to triangle and between the approximation terms as explained in Section 4.

5.3 Tiling of Light Field Maps

To avoid excessive texture swapping and improve rendering efficiency, we tile individual light field maps together to create collection of light field maps. To simplify the problem, we allow a predefined set of sizes for the light field maps during the resampling process, and then tile same-size light field maps together, as shown in Figure 4. Since one triangle requires three surface maps per approximation term, we tile all these maps in the same collection. We split the geometry of the model into p segments so that the tiled view maps for each segment do not exceed maximum texture size allowed. We then produce one view map collection per segment: $[V_1, V_2, \dots, V_p]$. Let $[S_1^i, S_2^i, \dots, S_{q_i}^i]$ be the list of surface map collections for vertex map collection V_i . For each approximation term, the rendering algorithm is as follows

```

for  $i = 1, \dots, p$  do
  load view map collection  $V_i$  into texture unit 1
  for  $j = 1, \dots, q_i$  do
    load surface map collection  $S_j^i$  into texture unit 2
    render all triangles with surface maps in collection  $S_j^i$ 
  end for
end for

```

6 Results

We have acquired the surface light fields and computed the light field map representations for four physical objects with diverse and

Models	Triangles	Number of Images	Raw Image Size	Resampled Data Size
Bust	6531	339	289 MB	5.1 GB
Dancer	6093	370	213 MB	2.9 GB
Star	4960	282	268 MB	5.2 GB
Turtle	3830	277	233 MB	3.8 GB

Table 1: The sizes of the experimental data sets. The second to last column shows the total size of renormalized triangles. The last column shows the size of the resampled surface light field data used for the decomposition.

complex reflection properties, including the metallicly-painted clay *bust* in Figure 13, *dancer* in Figure 12, semi-transparent and anisotropically reflective glass *star* in Figure 14, and the furry toy *turtle* in Figure 11. We also constructed a synthetic museum room as shown in Figure 1. The scene has approximately 60K polygons. Using 1-term approximation, we can render it at 18 frames per second using the GeForce™ 3 graphics card on a 2GHz Pentium™ 4 PC.

6.1 Acquisition and Resampling

Table 1 provides information about the data sizes for the scanned models used in our experiments. We use 24-bit RGB images in all the experiments. The raw image size in this table represents the total amount of radiance data after the visibility computation and triangle normalization process as described in Section 5. When resampling the viewing directions we assume a resolution of 32×32 . Note that, traditionally, research on light field compression reports results based on the size of the resampled light field data [22; 23]. The compression ratios reported in the next section are computed based on the size of resampled data.

Figure 8 illustrates the relationship of the compression ratio and the approximation quality to the triangle size for the bust object. As seen from the figure, smaller triangles provide better approximation quality but lower compression ratio. It is also interesting that for the same target error larger triangles always give a better compression ratio. Note also that smaller triangles are slower to render for each approximation term. Depending on the target hardware platform, user may choose to reduce the number of triangles while increasing the number of required passes, or vice versa, to achieve desired frame rate and compression ratio. For the following experiments, based on the goal of 100:1 compression obtained for the 3 term approximation with no compression of light field maps, we choose mesh resolution that gives us approximately 314 samples per triangle for all our models.

6.2 Approximation and Compression

We measure the quality of the surface light field approximation using RMS error computed between the resampled light field matrix and the matrix reconstructed from the approximation. Figure 9 shows PSNR (Peak Signal-to-Noise Ratio) for PCA-based and NMF-based approximations. The results show that both techniques produce high quality approximations using few approximation terms. Note that the scale in Figure 9 is logarithmic and that all approximations that use more than 3 terms have the RMS error of less than 2.0. This result provides a quantitative proof of the effectiveness of light field approximation through matrix factorization.

Between the two algorithms, PCA produces better quality than NMF, however, the difference is visually almost indistinguishable. Note that the frame buffer quantization that occurs during each texture blending operation is not considered in Figure 9. This quantization error implies that an approximation beyond 48dB would not improve the quality of hardware-accelerated renderer.

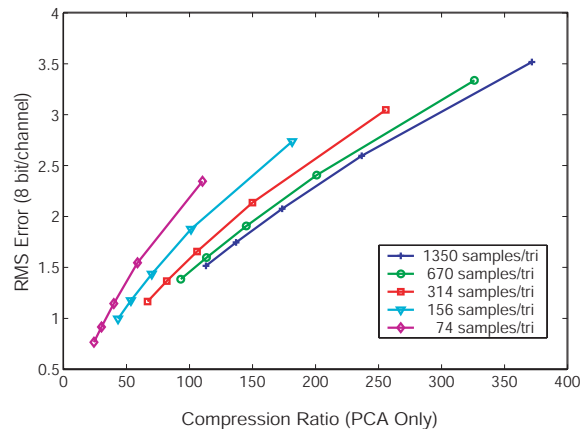


Figure 8: Relationship of the compression ratio and the approximation quality to the triangle size for the bust object. The points on each curve correspond to the different number of approximation terms for a given mesh resolution. Coarser mesh has more samples per triangle.

Table 2 lists the size and compression ratio of the light field data obtained through the light field data approximation and the additional compression of the light field maps. The size of geometric data falls below 10KB for all objects listed in the table when compressed using topological surgery [36] and therefore is negligible compared to the size of light field maps. By combining VQ with S3TC hardware texture compression, we achieve a *run-time* compression ratio of over 5000:1 for a 3-term approximation. For interactive purposes, 1-term approximation is often sufficient and thus the resulting compression ratio approaches 4 orders of magnitude.

6.3 Rendering

Figure 10 compares the rendering performance of PCA-based and NMF-based approximations. NMF-based rendering is 50% faster than PCA-based for the same number of approximation terms. The performance disparity will be larger if the target platform only supports positive texture values. The rendering performance of our algorithm is not very sensitive to the size of light field map data—doubling the image size reduces the frame rate by less than 20%. Rendering from compressed and uncompressed light fields is equally fast if image sets in both cases fit into the texture memory.

Figures 11-14 compare the rendering quality of our routines against the input images and report the corresponding errors. The errors reported in the figures are computed based on the difference between the input image and the rendered image using both APE (average pixel error) and PSNR for the *foreground* pixels only. Currently we discard partially visible triangles in the resampling process, which also contributes to the error. In the future, we plan to address the problem of partially occluded triangles by looking at factor analysis algorithms that use data statistics to fill in missing information [32].

7 Conclusion and Future Work

We have developed a new representation of surface light fields and demonstrated its effectiveness on both synthetic and real data. Using our approach, surface light fields can be compressed several thousand times and efficiently rendered at interactive speed on modern graphics hardware directly from their compressed representation. Simplicity and compactness of the resulting representation leads to a straightforward and fully hardware-accelerated rendering algorithm. Additionally, we present a new type of light field

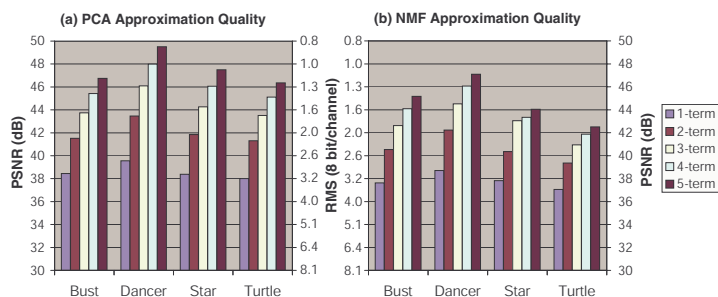


Figure 9: Approximation quality for different models and different number of decomposition terms. PSNR and RMS are based on the weighted average of the approximation errors for all light field matrices.

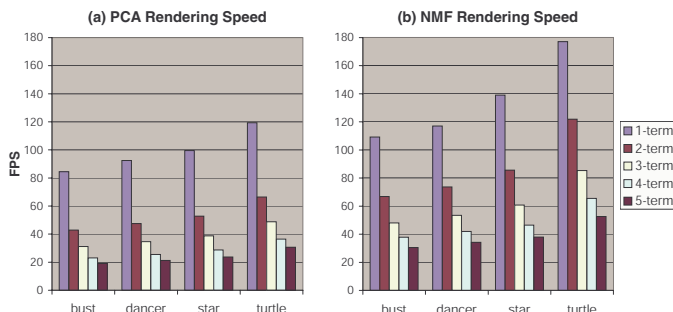


Figure 10: Rendering performance using NVIDIA GeForce 3 graphics card on a 2GHz Pentium 4 PC displayed at 1024×768 window with objects occupying approximately 1/3 of the window.

data factorization that produces positive only factors. This method allows faster rendering using commodity graphics hardware. Furthermore, the paper contains a detailed explanation of the data acquisition and preprocessing steps, providing a description of the complete modeling and rendering pipeline. Finally, our PCA-based approximation technique is particularly useful for network transport and interactive visualization of 3D photography data because it naturally implies progressive transmission of radiance data.

One of the limitations of a surface light field is that it models only the outgoing radiance of a scene. Consequently it is not possible to use surface light fields to render dynamic scenes where lighting changes and objects move. In the future, we are planning to work on extending the approach presented in this paper to relighting and animation of image-based models. If successful, these results would prove that image-based modeling and rendering is a practical and an appealing paradigm for enhancing the photorealism of interactive 3D graphics.

Acknowledgments

We thank Srihari Sukumaran for helping with the early version of the resampling and tiling algorithm. We are indebted to Omid Moghadam for his support in making light field mapping part of the MPEG4 standard. We thank Gary Bishop, Bernd Girod and Pat Hanrahan for critical reads of this paper. Timely proofreading by Gordon Stoll is greatly appreciated. We thank Henry Fuchs, Anselmo Lastra, Ara Nefian, Lars Nyland and Herman Towles for helpful suggestions and discussions and Gary Bradski, Bob Liang and Justin Rattner for encouraging this work. This research is in part supported by NSF Cooperative Agreement “Science and Technology Center for Computer Graphics and Scientific Visualization”, “National Teleimmersion Initiative” funded by Advanced Networks and Services, and Link Foundation Fellowships.

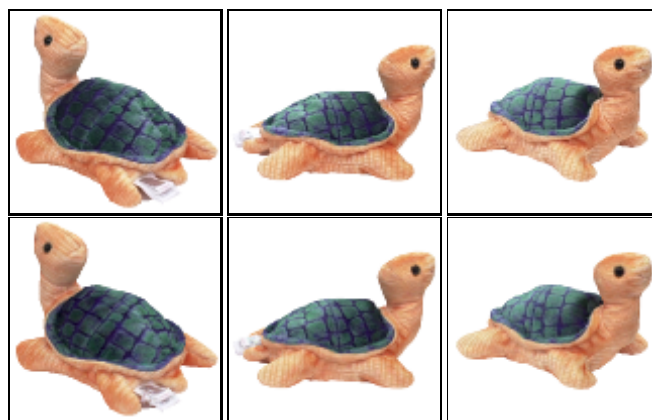


Figure 11: The figure shows a comparison for the turtle model between the input images shown at the top row and the images synthesized from the 1-term PCA approximation compressed using both VQ and S3TC shown at the bottom row. APE = 9.5, PSNR = 25.5 dB, the compression ratio is 8202:1, and the size of compressed light field maps is 468 KB.

References

- [1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.
- [2] S. Boivin and A. Gagalowicz. Image-Based Rendering of Diffuse, Specular and Glossy Surfaces From a Single Image. *Proceedings of SIGGRAPH 2001*, pages 107–116, August 2001.
- [3] J.-Y. Bouguet and P. Perona. 3D Photography Using Shadows in Dual-Space Geometry. *International Journal of Computer Vision*, 35(2):129–149, December 1999.
- [4] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen. Unstructured Lumigraph Rendering. *Proceedings of SIGGRAPH 2001*, pages 425–432, August 2001.
- [5] J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum. Plenoptic Sampling. *Proceedings of SIGGRAPH 2000*, pages 307–318, July 2000.
- [6] W.-C. Chen, R. Grzeszczuk, and J.-Y. Bouguet. Light Field Mapping: Hardware-accelerated Visualization of Surface Light Fields. Published as part of “Acquisition and Visualization of Surface Light Fields,” SIGGRAPH 2001 Course Notes for Course 46, pages 410–416, August 2001.
- [7] R. L. Cook and K. E. Torrance. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.
- [8] B. Curless and M. Levoy. Better Optical Triangulation through Spacetime Analysis. *Proc. 5th Int. Conf. Computer Vision, Boston, USA*, pages 987–993, 1995.
- [9] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the Reflectance Field of a Human Face. *Proceedings of SIGGRAPH 2000*, pages 145–156, July 2000.
- [10] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. *Proceedings of SIGGRAPH 96*, pages 11–20, August 1996.
- [11] 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.
- [12] A. Fournier. Separating Reflection Functions for Linear Radiosity. *Eurographics Rendering Workshop 1995*, pages 296–305, June 1995.
- [13] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
- [14] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. *Proceedings of SIGGRAPH 96*, pages 43–54, August 1996.
- [15] W. Heidrich and H.-P. Seidel. Realistic, Hardware-Accelerated Shading and Lighting. *Proceedings of SIGGRAPH 99*, pages 171–178, August 1999.
- [16] J. Kautz and M. D. McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. *Eurographics Rendering Workshop 1999*, June 1999.
- [17] J. Kautz and H.-P. Seidel. Towards Interactive Bump Mapping with Anisotropic Shift-Variant BRDFs. *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 51–58, August 2000.
- [18] J.J. Koenderink and A.J. van Doorn. Bidirectional Reflection Distribution Function Expressed in Terms of Surface Scattering Modes. In *European Conference on Computer Vision*, pages II:28–39, 1996.
- [19] E. P. F. Lafortune, S.-C. Foo, K. E. Torrance, and D. P. Greenberg. Non-Linear

- Approximation of Reflectance Functions. *Proceedings of SIGGRAPH 97*, pages 117–126, August 1997.
- [20] D. D. Lee and H. S. Seung. Learning the Parts of Objects by Non-Negative Matrix Factorization. *Nature*, 401:788–791, 1999.
- [21] Hendrik P. A. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-Based Reconstruction of Spatially Varying Materials. In *Twelveth Eurographics Rendering Workshop 2001*, pages 104–115. Eurographics, June 2001.
- [22] M. Levoy and P. Hanrahan. Light Field Rendering. *Proceedings of SIGGRAPH 96*, pages 31–42, August 1996.
- [23] 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.
- [24] T. Malzbender, D. Gelb, and H. Wolters. Polynomial Texture Maps. *Proceedings of SIGGRAPH 2001*, pages 519–528, August 2001.
- [25] Michael D. McCool, Jason Ang, and Anis Ahmad. Homomorphic Factorization of BRDFs for High-Performance Rendering. *Proceedings of SIGGRAPH 2001*, pages 171–178, August 2001.
- [26] L. McMillan and G. Bishop. Plenoptic Modeling: An Image-Based Rendering System. *Proceedings of SIGGRAPH 95*, pages 39–46, August 1995.
- [27] 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.
- [28] K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-Texture Method: Appearance Compression Based on 3D Model. In *Proceedings of the IEEE Computer Science Conference on Computer Vision and Pattern Recognition (CVPR-99)*, pages 618–624, 1999.
- [29] P. Poulin and A. Fournier. A Model for Anisotropic Reflection. *Proceedings of SIGGRAPH 90*, 24(4):273–282, August 1990.
- [30] 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.
- [31] R. Ramamoorthi and P. Hanrahan. A Signal-Processing Framework for Inverse Rendering. *Proceedings of SIGGRAPH 2001*, pages 117–128, August 2001.
- [32] Sam Roweis. EM algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.
- [33] Y. Sato, M. D. Wheeler, and K. Ikeuchi. Object Shape and Reflectance Modeling from Observation. *Proceedings of SIGGRAPH 97*, pages 379–388, August 1997.
- [34] P. Schröder and W. Sweldens. Spherical Wavelets: Efficiently Representing Functions on the Sphere. *Proceedings of SIGGRAPH 95*, pages 161–172, August 1995.
- [35] J. Spitzer. Texture Compositing With Register Combiners. Game Developers Conference, April 2000.
- [36] G. Taubin and J. Rossignac. Geometric Compression Through Topological Surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.
- [37] K. E. Torrance and E. M. Sparrow. Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces. *Journal of Optical Society of America*, 56(7), 1966.
- [38] G. J. Ward. Measuring and Modeling Anisotropic Reflection. *Proceedings of SIGGRAPH 92*, 26(2):265–272, July 1992.
- [39] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle. Surface Light Fields for 3D Photography. *Proceedings of SIGGRAPH 2000*, pages 287–296, July 2000.
- [40] Y. Yu, P. E. Debevec, J. Malik, and T. Hawkins. Inverse Global Illumination: Recovering Reflectance Models of Real Scenes From Photographs. *Proceedings of SIGGRAPH 99*, pages 215–224, August 1999.

Appendix: Matrix Factorization

Let \mathbf{F} be a matrix of size $M \times N$. The goal of matrix factorization leading to Equation (5) is to compute an $M \times K$ matrix $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_K]$ and an $N \times K$ matrix $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_K]$ such that the matrix

$$\tilde{\mathbf{F}} = \mathbf{UV}^T \quad (8)$$

best approximates \mathbf{F} in the least squares sense. The PCA and NMF factorizations are two algorithms achieving this goal while enforcing different constraints on the vectors \mathbf{u}_k and \mathbf{v}_k . PCA enforces the vectors \mathbf{u}_k and \mathbf{v}_k to be orthogonal and keeps the factorization progressive—that is, once the order K factorization is computed, the $K-1$ first pairs of vectors provide the best approximation at the order $K-1$. NMF, on the other hand, enforces vectors \mathbf{u}_k and \mathbf{v}_k

to have all positive entries. Unlike PCA, NMF produces a non-progressive factorization. In other words, if a different approximation order K is chosen, the overall matrices \mathbf{U} and \mathbf{V} have to be recomputed. The next two sections describe the implementation.

PCA Algorithm

The PCA factorization is based on computing the partial singular value decomposition of matrix \mathbf{F} . Since only the first $K \ll N$ eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_K$ of matrix $\mathbf{A} = \mathbf{F}^T \mathbf{F}$ need to be computed, the power iteration algorithm is well suited to achieve this goal. The pseudo-code for this algorithm looks as follows:

```

for  $p = 1, \dots, K$  do
   $\mathbf{F}_p \leftarrow \mathbf{F} - \sum_{k=1}^{p-1} \mathbf{u}_k \mathbf{v}_k^T$ 
   $\mathbf{A}_p \leftarrow \mathbf{F}_p^T \mathbf{F}_p$ 
  Initialize  $\mathbf{v}_p \leftarrow$  random  $N \times 1$  non-zero vector
  repeat
     $\mathbf{v}_p \leftarrow \mathbf{A}_p \mathbf{v}_p$ 
     $\mathbf{v}_p \leftarrow \mathbf{v}_p / \|\mathbf{v}_p\|$ 
  until  $\mathbf{v}_p$  converges
   $\lambda_p \leftarrow \sqrt{\mathbf{A}_p \mathbf{v}_p}$ 
   $\mathbf{u}_p \leftarrow \mathbf{F}_p \mathbf{v}_p / \lambda_p$ 
   $\max_u \leftarrow$  max of the absolute values of all the entries of  $\mathbf{u}_p$ 
   $\max_v \leftarrow$  max of the absolute values of all the entries of  $\mathbf{v}_p$ 
   $\alpha \leftarrow \sqrt{\max_u \lambda_p / \max_v}$ 
   $\mathbf{u}_p \leftarrow \lambda_p \mathbf{u}_p / \alpha$ 
   $\mathbf{v}_p \leftarrow \alpha \mathbf{v}_p$ 
  Quantize the entries of  $\mathbf{u}_p$  and  $\mathbf{v}_p$  for texture storage
end for

```

The coefficient α introduced in the code helps splitting the multiplicative factor λ_p among the vectors \mathbf{u}_p and \mathbf{v}_p so as to minimize the maximum quantization error.

NMF algorithm

The NMF factorization achieves the same matrix decomposition while enforcing all entries of the two matrices \mathbf{U} and \mathbf{V} to be positive. We propose to apply the algorithm presented by Lee *et al.* [20] to compute \mathbf{U} and \mathbf{V} iteratively:

```

Initialize  $\mathbf{U} \leftarrow$  random  $M \times K$  matrix of all strictly positive entries
Initialize  $\mathbf{V} \leftarrow$  random  $N \times K$  matrix of all strictly positive entries
repeat
   $\mathbf{U}_{n1} = [u_{n1}(i, j)] \leftarrow \mathbf{FV}$ 
   $\mathbf{U}_{n2} = [u_{n2}(i, j)] \leftarrow \mathbf{UV}^T \mathbf{V}$ 
   $\mathbf{U}_n = [u_n(i, j)] \leftarrow [u_{n1}(i, j) / u_{n2}(i, j)]$ 
   $\mathbf{V}_{n1} = [v_{n1}(i, j)] \leftarrow \mathbf{F}^T \mathbf{U}$ 
   $\mathbf{V}_{n2} = [v_{n2}(i, j)] \leftarrow \mathbf{VU}^T \mathbf{U}$ 
   $\mathbf{V}_n = [v_n(i, j)] \leftarrow [v_{n1}(i, j) / v_{n2}(i, j)]$ 
   $\mathbf{U} = [u(i, j)] \leftarrow [u(i, j) * u_n(i, j)]$ 
   $\mathbf{V} = [v(i, j)] \leftarrow [v(i, j) * v_n(i, j)]$ 
   $\mathbf{U} = [u(i, j)] \leftarrow [u(i, j) / \sum_{r=1}^M u(r, j)]$ 
until  $\mathbf{U}$  and  $\mathbf{V}$  converge

```

Models	Light Field Maps (3-term)	Compression of Light Field Maps		
		VQ	S3TC	VQ+S3TC
Bust	47.7 MB (106:1)	5.7 MB (885:1)	8.0 MB (636:1)	951 KB (5475:1)
Dancer	35.7 MB (82:1)	5.4 MB (542:1)	5.9 MB (490:0)	904 KB (3303:1)
Star	42.3 MB (122:1)	7.2 MB (716:1)	7.0 MB (737:1)	1.2 MB (4301:1)
Turtle	31.7 MB (121:1)	4.5 MB (847:1)	5.3 MB (726:1)	755 KB (5084:1)

Table 2: The size and compression ratio of the radiance data obtained through the light field map approximation and additional compression of the surface light field maps.

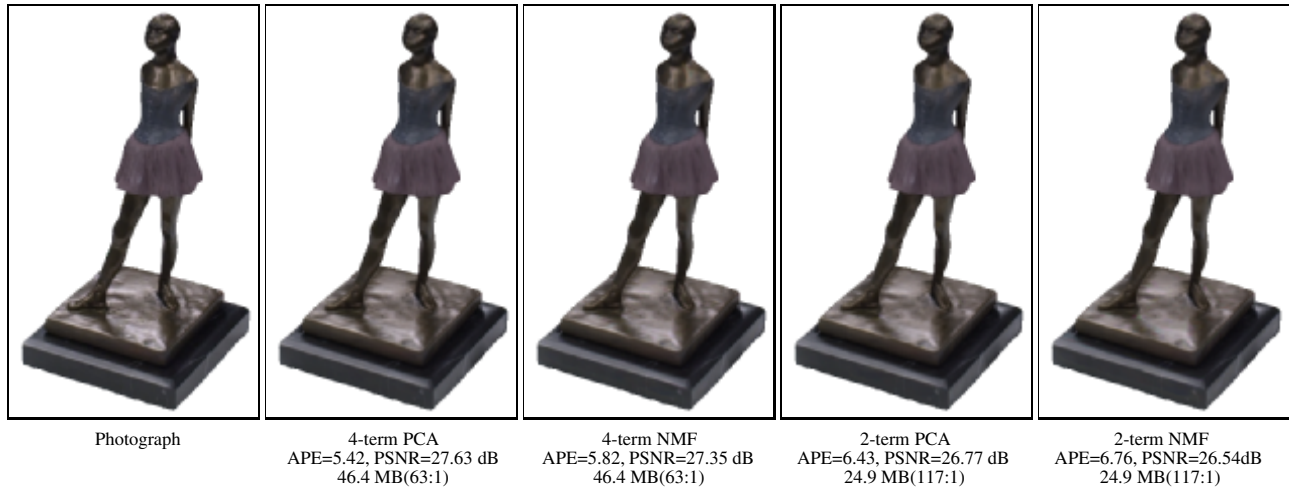


Figure 12: Comparison between PCA and NMF approximation methods. Using the same number of terms, PCA light field maps produce less error, but are slower to render than NMF.

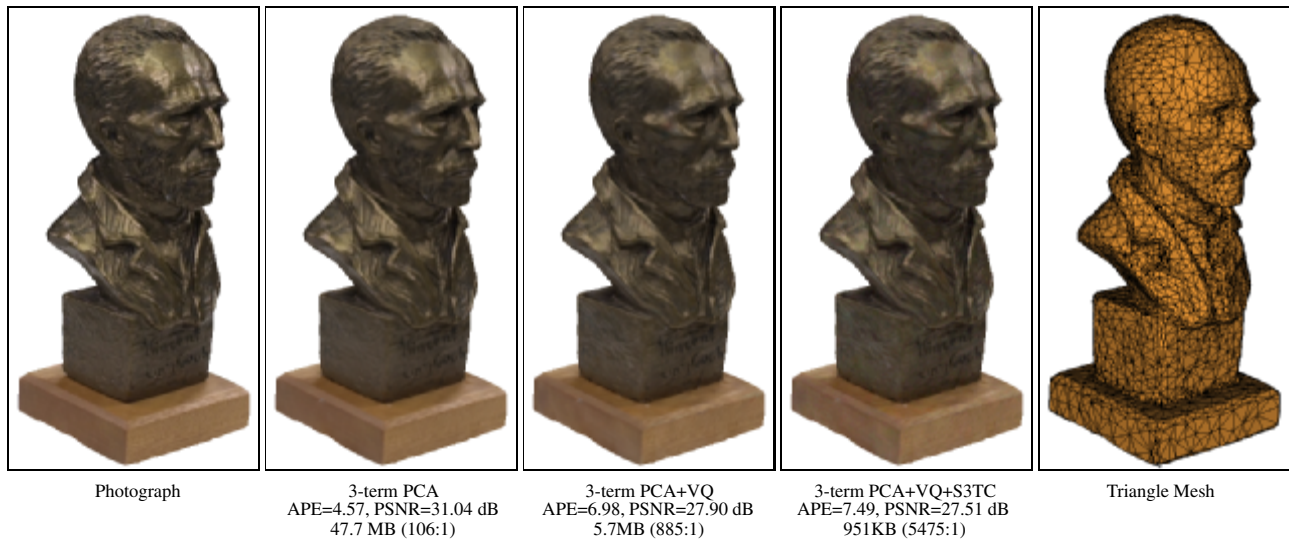


Figure 13: Comparison between different light field map compression algorithms using the bust model. VQ tends to diminish the highlight while S3TC preserves highlights better at the expense of color quality.

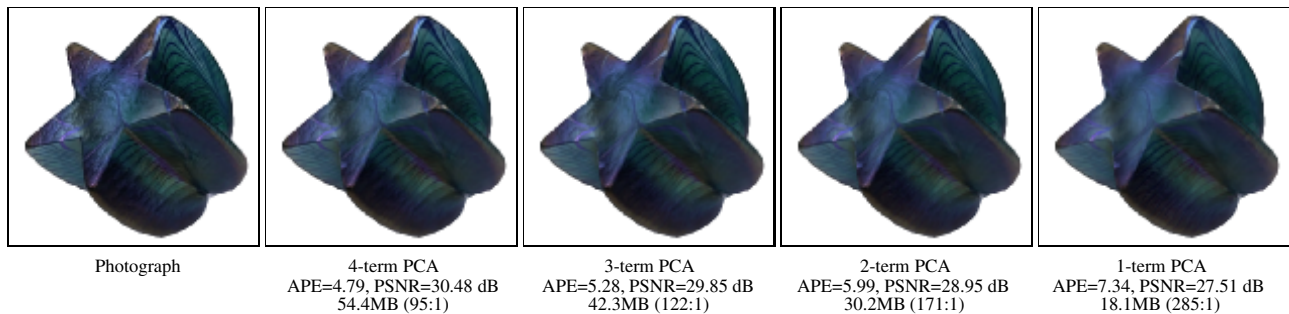


Figure 14: The figure demonstrates the progressive nature of PCA approximation. The same star model is rendered using different number of approximation terms.