Light Field Mapping: Efficient Representation of Surface Light Fields

Wei-Chao Chen

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175
Research Advisor: Henry Fuchs
ABSTRACT

Recent developments in image-based modeling and rendering provide significant advantages over traditional image synthesis processes, including improved realism, simple representation and automatic content creation. Representations such as Plenoptic Modeling, Light Field, and the Lumigraph are well suited for storing view-dependent radiance information for static scenes and objects. Unfortunately, these representations have much higher storage requirement than traditional approaches, and the acquisition process demands very dense sampling of radiance data. With the assist of geometric information, the sampling density of image-based representations can be greatly reduced, and the radiance data can potentially be represented more compactly. One such parameterization, called Surface Light Field, offers natural and intuitive description of the complex radiance data. However, issues including encoding and rendering efficiency present significant challenges to its practical application.

We present a method for efficient representation and interactive visualization of surface light fields. We propose to partition the radiance data over elementary surface primitives and to approximate each partitioned data by a small set of lower-dimensional discrete functions. By utilizing graphics hardware features, the proposed rendering algorithm decodes directly from this compact representation at interactive frame rates on a personal computer. Since the approximations are represented as texture maps, we refer to the proposed method as Light Field Mapping. The approximations 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 uncompressed light field data. We demonstrate the proposed representation through a variety of non-trivial physical and synthetic scenes and objects scanned through acquisition systems designed for capturing both small and large-scale scenes.
INTRODUCTION

In the field of computer graphics research, the quest for photo-realistic image synthesis has traditionally focused on the study of light transport mechanisms. Starting with analytical models of both the surface geometry and reflectance properties, these algorithms render images by using a combination of physical simulations and heuristics. Following this paradigm, we have witnessed a tremendous improvement in image quality, rendering efficiency and scene complexity over the past three decades. On the other hand, despite the continual improvement of modeling tools and techniques, model creation remains labor-intensive, and it has become the primary bottleneck in the traditional paradigm.

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 [1–3]. On the other hand, we are witnessing the emergence of Image-Based Rendering and Modeling (IBRM) techniques [4–7] 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.

Light Field Mapping Overview

A combination of synthetic and physical objects rendered using Light Field Mapping is shown in figure 1. A surface light field [8–10] is a 4-dimensional function \( f(r,s,\theta,\phi) \) that completely defines the radiance of every point on the scene surface geometry in every viewing direction. The first pair of parameters of this function \((r,s)\) describes the surface location and the second pair of parameters \((\theta,\phi)\) describes the viewing direction. In practice, a surface light field function is normally stored in sampled form, where as the geometry information are normally represented as surface mesh. Figure 2 illustrates the surface light field parameterization.

Because of its large size, a direct representation and manipulation of the light field data is impractical. We propose to approximate the discrete 4-dimensional surface light field function \( f(\cdot) \) as a sum of products of lower-dimensional functions.
We demonstrate that it is possible to construct approximations of this form that are both compact and accurate by taking advantage of the spatial coherence of the surface light fields. This is accomplished by partitioning the surface light field data across small surface primitives and building the approximations for each part independently. The proposed partitioning also ensures continuous approximations across the neighboring surface elements. By taking advantage of existing hardware support for texture mapping and composition, we can visualize surface light fields directly from the proposed representation at highly interactive frame rates. Because the discrete functions $g_k$ and $h_k$ 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 [10].
We make the following contributions to analysis and representation of image-based data and to hardware-accelerated rendering of image-based models.

- Partitioning of IBRM radiance samples on geometry for high-quality compression—We propose a novel type of light field data partitioning which allows efficient compression without introducing discontinuity artifacts.
- Efficient and high-quality compression algorithms for the radiance samples—We propose a class of compression algorithms that use simple linear approximations and are entirely data-driven. These algorithms work very well for intricate real-life scenes.
- A simple rendering algorithm suitable for real-time visualization—The rendering algorithm decompresses on-the-fly and renders using commodity graphics hardware at highly interactive rates.

In a later section, we introduce the proposed partitioning and approximation framework. Then we propose efficient rendering algorithms that allow on-the-fly decompression in graphics hardware. This representation are stored as images and can be further compressed using image processing algorithms, and several algorithms we have experimented with are presented. A description and implementation of the acquisition system used in this research is provided.
RELATED WORK

Reflection Models. Much conventional graphics research represents the surface reflectance properties as a model in the form of 4-dimensional Bidirectional Reflectance Distribution Function (BRDF) $r(u_i, f_i, u_o, f_o)$. A BRDF defines the ratio of the outgoing radiance at direction $(u_o, f_o)$ to the incoming irradiance from direction $(u_i, f_i)$. Many earlier methods represent the BRDF analytically. These methods can be further classified into two categories, namely empirical models [11–15] and physically-based models [16–19].

A recent research trend is to approximate BRDF by lower-dimensional sampled functions to facilitate hardware-accelerated rendering. Heidrich et al. [1] investigated and concluded many analytical BRDFs can be separated into products of 2-dimensional functions. Kautz and McCool [2] propose hardware-assisted rendering of arbitrary BRDFs through their decomposition into a sum of 2D separable functions. The homomorphic factorization of McCool et al. [20] generates a BRDF factorization with positive factors only, which are easier and faster to render on the current graphics hardware. Although the homeomorphic factorization framework can potentially support progressive encoding, the authors only presented an algorithm that limits the factorization to three factors. The research presented in this thesis is in part inspired by these research, although our application is fundamentally different. These methods are limited to sample-based representation of shift-invariant reflectance models. Our research focuses on complex, real world surfaces that may have different reflectance properties on each point of the surfaces. We also present a novel method for factorization of light field data that produces only positive factors [21]. The proposed method is significantly easier to implement and more accurate than the homomorphic factorization in [20].

To accommodate for spatial variance on the surface, there are also research efforts to extend the BRDF beyond 4 dimensions. For example, a Bidirectional Texture Function (BTF) is a 6-dimensional function that provides a BRDF for each 2-dimensional surface point [22–24]. Instead of explicitly sampling the BTF, Lensch et al. [25] took a different approach by taking a sparse set of photographs around an object and reconstructing a set of basis BRDFs from these photographs. Although this algorithm requires only a small set of photographs, it does not work well for complicated surfaces with many different BRDFs.

Image-Based Representations. Image-based methods synthesize novel images directly from input photographs. Some of the IBRM representations contain only samples from the acquisition process, such as the plenoptic modeling [4] and related image warping research [26–28], the
light field rendering [5], and the lumigraph [6]. Others use traditional surface primitives to store geometric information, such as View-Dependent Texture Mapping (VDTM) [29–31, 7]. Figure 3 categorize these techniques according to the amount of geometric and radiance data required for each representation.

Chai et al. [32] pointed out that geometry information can be used to trade off radiance information. Therefore, some of the representations are more redundant than others, and can be potentially represented more compactly. Much research has been done on compression of light field data. Levoy and Hanrahan [5] use the Vector Quantization (VQ) technique [33] for compression of light field data. Magnor, Eisert and Girod have developed a series of disparity-compensated light field codecs [34–36]. Most of the research, however, do not lead to hardware-accelerated rendering algorithms.

**VDTM and The Surface Light Field.** View-Dependent Texture Mapping (VDTM) extends classic texture mapping algorithms by stores multiple images per surface primitive. During rendering, the algorithm texture-maps the surfaces using images taken from similar viewing directions. The primary advantages of VDTM techniques are that the required amount of radiance data in a VDTM is generally much lower than purely sample-based IBRM representations. However, as the number of input images increases, the amount of radiance data grows proportionally, and therefore VDTM algorithms does not scale to the number of input images.
Nishino et al. [37] proposed the eigen-texture method that compresses generalized VDTM data using Principal Component Analysis (PCA) algorithm [38]. They achieve approximately 20:1 compression ratio. Our proposed Light Field Mapping technique is sometimes confused with the eigen-texture method. The original formulation of the eigen-texture method only allows synthesis of input images and novel views on the path connected by a pair of images. This technique is thus not a general VDTM representation. Unlike for Light Field Mapping method, there are no reported real-time rendering algorithms for the eigen-texture representation.

The surface light field parameterization [8,9] is very similar to a VDTM parameterization. However, VDTM defines the viewing parameters $\theta, \phi$ on a per surface primitive basis, whereas surface light field parameterization treats each surface point differently. VDTM is therefore an approximation of surface light field. Miller et al. in [8] proposed a method of rendering surface light fields from input images compressed using JPEG-like compression. Wood et al. [9] uses a generalization of VQ and PCA to compress surface light field and proposes a two-pass rendering algorithm that displays compressed light fields at interactive frame rates. Compared to this technique, we reported approximately two orders of magnitude faster rendering speed using Light Field Mapping approach in [39].

SURFACE LIGHT FIELDS APPROXIMATION

This section describes our method for approximating the radiance data. We first present a novel partitioning method that allows each partition to be approximated independently without introducing discontinuity artifacts. Then, we describe our approximation framework based on matrix factorization and decomposition algorithms. These approximations can be decoded and visualized very efficiently using the proposed Light Field Mapping rendering algorithms in a later section.

Surface Light Field Partitioning

A IBRM sample database is generally very large, and in practice, we can only handle a local scope of data during preprocessing. To enable efficient compression, the samples within a local scope should be highly coherent. For surface light field parameterization, surface primitives naturally define the unit of scope for our purposes. The units together form a partitioning of the sample database, and if we can process each part independently without introducing artifacts, we can parallelize both the approximation and decoding algorithms. Based on these observations, an effective surface light field partitioning scheme should divide surface light field data. Furthermore, independent approximation of each part should not introduce artifacts.
Since the geometry of our models is represented as a triangular mesh, an obvious partitioning of the light field function $f(r,s,\theta,\phi)$ is to split it between individual triangles

$$f^{\Delta_t}(r, s, \theta, \phi) = \Pi^{\Delta_t}(r, s)f(r, s, \theta, \phi),$$

where $\Pi^{\Delta_t}(r, s)$ is a step function that is equal to one within the triangle $\Delta_t$ and zero elsewhere. Because the partitioning breaks the original surface light field function on the triangle boundaries, we refer to this approach as triangle-centered partitioning. Unfortunately, when each function is approximated independently, the approximation process results in visible discontinuities at the edges of the triangles.

To eliminate the discontinuities across triangle boundaries, we propose to partition surface light field data around each vertex. The part of surface light field corresponding to each vertex is referred to as the vertex light field and for vertex $v_j$ it is denoted as $f^{v_j}(r, s, \theta, \phi)$. This partitioning is computed by multiplying weighting to the surface light field function

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

where $\Lambda^{v_j}$ is the barycentric weight of each point in the ring of triangles centered around vertex $v_j$. The value of $\Lambda^{v_j}$ is equal to 1 on vertex $v_j$, and it decreases linearly toward zero at the boundary. Because of their shape, the weighting functions are often referred to as the hat functions. In Figure 4, the top row shows hat functions $\Lambda^{v_1}, \Lambda^{v_2}, \Lambda^{v_3}$ for three vertices $v_1, v_2, v_3$ of

![Figure 4. The finite support of the hat functions $\Lambda^{v_j}$ around vertex $v_j$, $j=1,2,3$. $\Lambda^{v_j}_{\Delta_t}$ denotes the portion of $\Lambda^{v_j}$ that corresponds to triangle $\Delta_t$. Functions $\Lambda^{v_1}, \Lambda^{v_2}$ and $\Lambda^{v_3}$ add up to one inside $\Delta_t$.](image-url)
triangle $\triangle$. The bottom row of the same figure shows that these three hat functions add up to unity inside triangle $\triangle$. Therefore, Equation (3) defines a valid surface light field partitioning, because the original surface light field can be reconstructed by simply summing up individual vertex light fields.

The final step of vertex-centered partitioning reparameterizes each vertex light field to the local vertex reference frame, as shown in Figure 5. A vertex reference frame is defined such that its $z$-axis is parallel to the normal at the vertex. The reparameterized $(\theta, \phi)$ are simply the polar and azimuth angles of viewing directions in this frame. The vertex light field functions together with their corresponding local coordinates allow us to reconstruct the original data unambiguously. In the rest of the section, when we refer to a vertex light field function, we assume it to be expressed in the local coordinate, and for simplicity we use the same notation for both local and global parameters.

**Vertex Light Field Approximation**

As stated, vertex-centered partitioning of light field data allows us to approximate each partition independently without introducing discontinuity artifacts. We propose to approximate vertex light field as

$$f_{v_j}^v(r, s, \theta, \phi) \approx \sum_{k=1}^{K} g_{v_j}^v(r, s) h_{v_j}^v(\theta, \phi) \cdot \Delta$$

The 2D functions $g_{v_j}^v$ contain only the surface parameters, and we refer to them as *surface maps*. Similarly, we refer to $h_{v_j}^v$ as *view maps*. The above ap-
Light Field Mapping: Efficient Representation of Surface Light Fields

proximation can effectively compress the function \( f_v \) if we only need a few approximation terms \( K \) to achieve high quality of approximation. We leverage existing matrix factorization algorithms to calculate the above approximations numerically. Before we discuss details of these algorithms, we describe how the vertex light field approximation problem can be transformed into a 2D matrix factorization problem.

For practical purposes, we assume that the vertex light field is stored in discrete format \( f_v [r_p, s_p, \theta_v, \phi_v] \), where index \( p = 1, \ldots, M \) refers to the discrete values \([r_p, s_p]\) describing the surface location within triangle ring of vertex \( v_j \), and index \( q = 1, \ldots, N \) refers to the discrete values \([\theta_v, \phi_v]\) of the viewing angles. We may rearrange the discrete vertex light field into a 2D matrix

\[
F_v = \begin{bmatrix}
  f_v[r_1, s_1, \theta_1, \phi_1] & \cdots & f_v[r_1, s_1, \theta_N, \phi_N] \\
  \vdots & \ddots & \vdots \\
  f_v[r_M, s_M, \theta_1, \phi_1] & \cdots & f_v[r_M, s_M, \theta_N, \phi_N]
\end{bmatrix},
\]

where \( M \) is the total number of surface locations and \( N \) is the total number of views at each surface location. We refer to matrix \( F_v \) as the vertex light field matrix. In practice, to obtain vertex light field matrices from input images, we need to resample the input samples or photographs. Our 4D resampling algorithm are described later.

Matrix factorization algorithms construct approximate factorizations of the form

\[
\tilde{F}_v = \sum_{k=1}^{K} u_k v_k^T,
\]

where \( u_k \) is a vectorized representation of discrete surface map \( g_v[r_p, s_p] \) and \( v_k \) is a vectorized representation of discrete view map \( h_v[\theta_v, \phi_v] \). The matrix \( F_v \) contains \( M \times N \) samples, whereas its approximation contains \( K \times (M + N) \). If \( K \ll \min(M, N) \), the size of approximation become much smaller than the size of original matrix \( F_v \).

Matrix Approximation Algorithms

In Equation 6, we observed that matrix factorization algorithms can be used to approximate the light field matrix. Although the light field matrices are generally quite large, computing full matrix factorization may be very time consuming. However, because we only need approximate factorization of the matrix, with proper implementation, the efficiency of these algorithms can be drastically improved.
We experimented with two algorithms to calculate the approximations: Principal Component Analysis (PCA) [38] and Non-Negative Matrix Factorization (NMF) [21]. Both of these algorithms compute matrix factorization in a form similar to Equation 6, and they have been used in a wide range of applications such as data compression and unsupervised learning. The differences between the two algorithms arise from the constraints imposed on the approximation factors $u_k$ and $v_k$. PCA enforces the factors $u_k$ and $v_k$ to be orthogonal vectors and keeps the factorization progressive; that is, once an order $K$ factorization is computed, the first $(K-1)$ pairs of vectors provide the best order $(K-1)$ approximation. NMF, on the other hand, enforces all entries in vectors $u_k$ and $v_k$ to be positive. Unlike PCA, NMF produces a non-progressive factorization. In other words, a new approximation has to be recomputed when a different order $K$ is chosen.

**PCA Algorithm.** The PCA factorization is based on computing the partial Singular Value Decomposition (SVD) of a matrix. The SVD of a $M \times N$ matrix $F$ has the following form

$$F = \sum_{i=1}^{\min(M,N)} u_i s_i v_i^T,$$

where the column vectors $u_i$ and $v_i$ are the orthonormal left- and right-singular vectors of the matrix $F$, respectively. The singular values $s_i$ are ranked in non-ascending order so that $s_i \leq s_j$ for all $i < j$. Therefore, a partial sum of Equation 7 also gives the RMS optimal approximation to matrix $F$ for the given dimensionality.

Because our goal is to approximate a matrix, it is unnecessary to perform the time-consuming SVD. In order to efficiently compute a $K$-term approximation of a matrix $F$, we can instead compute the eigenvectors corresponding to the $K$ largest eigenvalues of the covariance matrix $F^T F$. The power iteration algorithm is well suited to achieve this goal [40].

PCA algorithm requires the elements in matrix $F$ to have zero mean. To satisfy this precondition, we subtract the average column vectors from the matrix $F$ before performing PCA. The extracted vectors contain only surface parameters and can be treated as traditional diffuse texture maps during the rendering process.

**NMF algorithm.** We apply the iteratively algorithm presented by Lee et al. [21] to compute NMF approximation. Unlike PCA algorithm, all approximation vectors $u_k$ and $v_k$, $k = 1 \cdots K$ are updated simultaneously in every iteration, and we denote the matrix form of these vectors as $U$ and $V$ respectively. To improve approximation precision, we subtract the minimum
column vector from matrix \( F \) before performing NMF. These minimum vectors are also treated as diffuse texture maps in the rendering process.

**RENDERING ALGORITHMS**

Earlier we described the process of partitioning and approximating surface light field data. The process generates sets of images collectively referred to as light field maps. In this Section we propose algorithms that use light field maps to achieve real-time rendering and on-the-fly decompression by taking advantage of graphics hardware features.

**Rendering by Texture-Mapping**

Let \( g^v_j[r_p,s_p] \) be the surface map and \( h^v_k[\theta_q,\phi_q] \) be the view map corresponding to the \( k \)-th approximation term of vertex light field \( f[v_j,\theta_q,\phi_q] \). The approximation of the light field data for triangle \( \Delta_t \) can be written as

\[
\bar{f}^\Delta_t[r_p,s_p,\theta_q,\phi_q] = \sum_{k=1}^{K} \left( \sum_{j=1}^{3} (g^v_j[r_p,s_p]_{\Delta_t}) (h^v_k[\theta_q,\phi_q]) \right), \tag{8}
\]

where index \( j \) runs over the three vertices of triangle \( \Delta_t \), and \( g^v_i[r,s]_{\Delta} \) denotes the portion of the surface map \( g^v_i \) corresponding to triangle \( \Delta \). Equation (8) suggests that even though the approximation is done in a vertex-centered fashion, an approximation term 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. We now describe the rendering algorithm for one approximation term for one triangle. For simplicity we also drop the index \( k \) in functions \( g^v_i \) and \( h^v_k \). We may devise a straightforward rendering algorithm as follows. For each target pixel in the triangle, we calculate the parameters \( (r,s,\theta,\phi) \) for this pixel, and evaluate Equation (8) to decode the approximation. This rendering algorithm is straightforward but it fails to exploit the inherent parallelism in the light field maps format.

To speed up the rendering process, we may utilize texture mapping features in contemporary graphics hardware. Texture coordinates within surface primitives are normally linearly interpolated from the vertex texture coordinates. Therefore, we need to transform the parameters of view maps into a space compatible with hardware texture coordinates interpolation. Assume vector \( d \) is the normalized viewing direction, and that vectors \( x \) and \( y \) correspond to the axes of the local reference frame. We may then
calculate the texture coordinate \((x,y)\) by the orthographic projection of \(d\) onto the plane defined by vectors \(x\) and \(y\)

\[
x_p = s_x(d \cdot x) + x_c, \quad y_p = s_y(d \cdot y) + y_c
\]  

(9)

where the scale-and-bias parameters \((s_x, s_y, x_c, y_c)\) represent the size and relative location of the view map on the texture map. This projection, as shown in Figure 6, is normally referred to as an XY-map. This projection offers a reasonable approximation on the interpolation quality and it is quite efficient to compute. Other transformations, such as the hyperbolical maps described in Heidrich et al. [1], can also be used for the viewing projection of light field maps.

Figure 7 illustrates the 3 light field maps pairs used to compute one approximation term of a light field for triangle \(\Delta_l\). The shaded portions indicate the parts of the light field maps that are used to decode appearance from a certain viewing direction. The middle column shows surface maps \(g^{\theta_s}[r_p, s_p]_{\Delta_l}\) and the right column shows view maps \(h^{\theta_s}[\theta_\rho, \phi_\eta]\). The samples on the view maps are parameterized with the XY-map of the local reference frame attached to the vertex. Based on where the camera is located, the rendering algorithm calculates the texture coordinates \((x^{\theta_s}_l, y^{\theta_s}_l)\) for each view map. To this end, we apply Equations (9) to the viewing direction vector \(d\) in the local reference frame \((x_l, y_l, z_l)\) of vertex \(v_l\) to calculate the texture coordinate \((x_l, y_l)\) on the XY-maps. This results in 3 texture fragments shown in the right column of Figure 7. Note that the texture coordinates are different for each view map fragment because we use different reference frames to compute them. The surface map texture coordinates do not depend on the viewing angle and they remain static throughout the rendering process.
Evaluating one complete approximation term then simply proceeds as follows. We texture map each pair of surface map and view map texture fragments and multiply the results pixel-by-pixel. The product is then placed into the accumulation buffer. Multiple term approximation of each triangle light field is computed by running the same algorithm multiple times using their corresponding light field maps.

**Utilizing Hardware Features for Efficient Rendering**

The light field maps decoding process is simple and amendable for hardware implementation. In this section we discuss efficient rendering algorithms using specific hardware features, such as multitexturing, extended color range, and vertex shaders.

Multitexturing Support. This hardware support enables us to compute the modulation, or multiplication, of multiple texture fragments very efficiently in one rendering pass. Consequently, for the NMF-based approximations, which contain strictly positive light field maps, we need 3 rendering passes to render each approximation term with multitexturing graphics hardware.
that supports 2 texture sources. Each rendering pass decodes the approximation from one of the three vertices $v_j$ in Equation 8.

Extended Color Range. For the PCA-based approximation, which in general produces 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 $[0,1]$. Without extended range support, we may need up to four rendering passes for each full-range modulation [2]. Recently more hardware platforms are supporting extended color range [41], but the output results are normally clamped to positive values. We may use this feature to evaluate full-range modulation in two rendering passes as follows. Let $M$ be the result of modulation of two texture fragments $A$ and $B$. Let $M_+$ and $M_-$ be the clamped modulation of fragments $(A,B)$ and $(-A,B)$ respectively. We can compute $M$ by subtracting the outputs of the two modulations $M = M_+ - M_-.$

**Improving Memory Bus Efficiency**

In modern graphics systems, the amount of texture memory are generally less than the total system memory, and for larger light field maps, texture swapping between system and texture memory may be required. In order to reduce the swapping overhead, individual light field maps can be tiled or mosaicked together into larger texture maps, or texture atlases. The rendering routine goes through each texture atlas and reconstructs approximations associated with this texture. To ensure optimal texture swapping, we devised the tiling routine in two steps. First, surface geometry of the model is divided into several groups. Then, light field maps corresponding to the same group are tiled together to generate texture atlases.

Model Segmentation. When the texture memory in the graphics subsystem is abundant, one simply tiles all surface maps and view maps into two large texture atlases. In practice, however, not only may the size of texture memory not be sufficient, the maximum size of a texture map is also limited. Therefore, we may divide the surface geometry into several groups of triangles and collect the view maps and surface maps within each group into texture atlases. Our current implementation segments the model into multiple pieces by running a breadth-first search algorithm on surface triangles. Each search generates a connected group of triangles, and the search stops when all triangles connected to the root node are visited, or when the number of triangles or vertices exceeds a user-defined size.

Texture Atlas Generation. After model segmentation, surface maps and view maps can be tiled into texture atlases. To simplify the problem, our current implementation generates fixed size view maps for the whole model, and we only allow a predefined set of surface map sizes during the resampling pro-
cess. Same-size light field maps from the same approximation term are then tiled together into one texture atlas. Since one triangle requires three surface maps per approximation term, these maps are tiled in the same texture.

Assume that the surface geometry is divided into \( p \) groups. We denote the view map atlas for term \( k \), group \( i \) as \( V^k_i \). Let \( [S^k_1, S^k_2, \ldots, S^k_{iqi}] \) be the list of surface map atlases in group \( i \). When we render the scene in the texture atlas centric ordering, each view map and surface map is loaded only once, and is thus optimal in terms of texture swapping cost.

**COMPRESSION OF LIGHT FIELD MAPS**

Approximation through matrix factorization described in earlier can be thought of as a compression method that removes local redundancy in the vertex light field function. The compression ratio of this method is closely related to the size of the surface primitives used for partitioning. Currently, we choose the size of triangles empirically to obtain about two orders of magnitude compression ratio through approximation while maintaining high approximation quality without using many approximation terms.

Despite the efficiency of light field maps representation, they are still redundant. First, 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. For optimal run-time performance, compressed light field maps need to be decompressed on-the-fly during rendering. In this section we discuss several techniques that satisfy these criteria. Figure 8 gives an overview of the different types of compression.

![Compression Overview](image)

**Figure 8.** Compression Overview. The number under each technique describes its approximate compression ratio.
algorithms we have experimented with. Other image compression techniques can be used to further reduce the off-line storage size, but are not discussed here.

Global Data Redundancy. Data redundancy across individual light field maps can be reduced effectively using VQ [33]. Our implementation represents each triangle surface map $g_{ij}(r_p,s_p)$ and each view map $h_{ij}(u_{pq},v_{pq})$ 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 an user-specified compression ratio or the average distortion to drive the VQ compression algorithms. 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 by an order of magnitude using VQ without significant loss of quality. In the current implementation, VQ is applied after all light field maps are computed. However, since the light field maps in the PCA algorithm are computed incrementally, we could potentially apply VQ after each iteration of the approximation algorithm and then factor the resulting error into the next approximation term.

Local Data Redundancy. Data redundancy within individual light field maps can be reduced efficiently using block-based algorithms. One such method, called S3TC™, is often supported on commodity graphics cards 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 within each block it calculates and stores two representative colors. Each texel in the original block is then replaced by the linear interpolation of the representative colors. Since this algorithm uses blocks that are smaller than most light field maps, when compared to VQ, it generates noisier images but it preserves the specularities and sharp highlights better.

ACQUISITION OF SURFACE LIGHT FIELDS

In this Section, we describe the surface light fields acquisition systems used in our research. We first describe an accurate acquisition system for
small-scale scenes, and move on to discuss another system designed to capture larger environments.

**Small-Scale Acquisition**

Figure 9 illustrates the small-scale acquisition system used in our research [10]. This system scans a 1 ft$^3$ volume accurately. It employs a registration platform for automatic registration between images and range scans. The object is placed on the platform and remains static to the platform throughout the acquisition process. The system scans geometry and radiance as two separate steps, and data from both steps are registered together using the coordinate system defined on the platform.

The first acquisition stage acquires radiance data with a hand-held camera, as shown in Figure 9(a). The internal parameters of the camera are calculated in a separate calibration stage. For each object, we capture between 200 to 400 images, covering the upper-hemisphere of the platform.
Figure 9(b) shows one sample image captured with this process. The color circles on the platform used to provides an initial estimate for the position of all the grid corners on the platform. The initial estimates are then localized using a corner finder to calculate the accurate camera pose relative to the platform. The outcome of this process is a set of \( N_i \) images captured from known vantage points in 3D space.

The 3D geometry of the object is scanned using a structured lighting system consisting of a projector and a video camera, as shown in Figure 9(a). In this phase, we paint the objects with white removable paint in order to improve the accuracy of the scanned geometry. Figure 9(c) shows an example camera image acquired during scanning. The projected stripped patterns observed by the camera are used to triangulate the 3D position of the object surfaces. Because each scan only covers parts of the object, we take between 10 and 20 around the object to completely cover the object surface. The individual scans are naturally registered together on the registration platform coordinate system. The resulting point cloud, containing approximately 500,000 points in this case, are fed into a mesh editing software [42] to reconstruct the final triangular surface mesh shown in Figure 9(d). Figure 9(e) shows the projection of the mesh onto the camera image displayed in Figure 9(b). The error of mesh reprojection is less than one out of two thousand pixels on the object silhouette.

Large-Scale Acquisition

The requirements for large-scale scene acquisition are very different from the small-scale ones. For example, the range scan system need to be capable of scanning a much larger volume. Also, it is often not desirable and practical to introduce calibration targets into the scene. Our large-scale surface light field acquisition system consists of a commercial laser rangefinder, the DeltaSphere\textsuperscript{TM}[43], for geometry acquisition, as shown in Figure 10(a), and a hand-held digital camera for radiance acquisition. The laser rangefinder acquires depth maps on a spherical coordinate system, and each scan produces approximately 8 million depth samples. Figure 9(c) shows one of the depth maps. Each range scan takes approximately 20 minutes. For this scene, we took a total of 7 panoramic depth scans and approximately 100 pictures at different positions in the environment. Figure 10(b) shows one example photograph after removing nonlinear distortions.

We use a commercial software package called Polyworks\textsuperscript{TM} to bring all scans into the same reference frame [44]. Polyworks implements an Iterative Closest Point (ICP) algorithm for point-cloud registration. After depth maps are registered, we triangulate each depth map separately and merge individual triangular mesh into a unified geometry with Polyworks.
We recover the camera poses using manual 2D-3D correspondences. For each image, a user first chooses a depth map, and then selects 6 or more pairs of corresponding points on both the depth map and the image. This allows the calculation of the external parameters with respect to the coordinate system of the depth map. Then, the transformations between individual depth maps and the global coordinate system obtained through the ICP algorithm during the geometry reconstruction stage is used to recover the global camera pose of the image. This process relies on an accurate ICP registration, which also requires accurate rangefinder calibration. Currently inaccuracies in rangefinder calibration account for most of the overall error. However, as shown in Figure 19, artifacts due to inaccurate registration can largely be resolved by the view-dependent nature of surface light fields representation.

**RADIANCE DATA RESAMPLING**

The approximation algorithms proposed earlier take densely sampled surface light field functions as input. In practice, however, data from the acquisition systems are scattered samples of the actual surface light field functions. Now we discuss methods and issues in preparing these data for the approximation algorithms. In particular, we assume that the input data consist of a triangular geometric mesh together with a set of camera images registered to the mesh. Alternatively, we may bypass the resampling process altogether at a cost of lower approximation quality, as described in [39].

The input data at this stage consist of a triangular mesh and a set of images taken with known camera internal and external parameters. The
The goal of resampling is to construct a surface light field function \( f[r_p s_p, \theta_q \phi_q] \) that best represents the input data. The problem of surface light field resampling is, in general, a 4D data reconstruction problem. However, if the reprojection sizes of the triangles are relatively small compared to their distances to the camera, for a triangle the samples from the same image can be regarded as having identical viewing directions \((\theta, \phi)\). Under this assumption, the resampling process can be approximated by a two-stage algorithm that first resamples on \([r_p s_p]\) and then on \([\theta_q \phi_q]\). We refer to the first stage as the **surface normalization** stage and the second stage as the **view interpolation** stage. We focus our discussion on one vertex light field function \( f^v[r_p s_p, \theta_p \phi_p] \).

**Surface Normalization.** Before normalization, we need to determine the visible cameras for the target vertex \( v_j \). Repeating the visibility calculation for all \( N \) camera images yields a list of \( N_j \) visible vertex views. We denote the viewing directions of the visible views as \([u_{vj}, f_{vj}]\), where the index \( v' = 1, \ldots, N_j \). The visible vertex views correspond to a set of texture patches of irregular size captured from various viewing directions. The algorithm then normalizes each texture patch to have the same shape and size as the others by using bilinear interpolation of the pixels in the original views. In order to preserve image sampling rate, the size of the normalized patch is chosen to be proportional to the size of the largest projected view. Then, each vertex view is multiplied with the hat function \( L^v_j \) described in equation 3.

**View Interpolation.** At this stage, each triangle view contains a uniform number of samples, but the sampling of views is still irregular. We denote the input function at this stage as \( t^v[r_p s_p, u_{vj}, f_{vj}] \). Given the input function, the goal is to reconstruct the vertex light field function \( f^v[r_p s_p, \theta_p \phi_p] \). To do this, we construct an interpolation function \( Q(u,f) \) that returns a 3×2 matrix whose first and second columns define the indices to the original views and the interpolation weights respectively. The components of \( Q(u,f) \) have the following properties

\[
\forall \{\theta, \phi| -\pi \leq \theta \leq \pi, -0.5\pi \leq \phi \leq 0.5\pi\},
\]

\[
\begin{align*}
Q(0,\phi)_{k1} & \in \{1, \ldots, N_j\}, \\
\sum_{k=1}^{3} Q(0,\phi)_{k2} & = 1.
\end{align*}
\]
This allows us to perform view interpolation as follows

\[
\begin{bmatrix}
i_1 & w_1 \\
i_2 & w_2 \\
i_3 & w_3
\end{bmatrix} \leftarrow Q^j(\theta_q, \phi_q)
\]

\[
f^v_i[r_p, s_p, \theta_q, \phi_q] = \sum_{k=1}^{3} w_k \tau^v_i[r_p, s_p, \theta^f_{ik}, \phi^f_{ik}],
\]

Figure 11 describes the definition of the interpolation function \(Q\). First, the viewing directions from the visible views \([\theta_v, \phi_v]\) are projected onto the \(xy\) plane using XY-map projection. The result is a set of texture coordinates, as shown in Figure 11(a). These coordinates are used to generate a Delaunay triangulation as shown in Figure 11(b). We then define the interpolation function as follows. For each point on the XY-map, the indices \(i\) are the three vertices of the triangle surrounding it, and the weights \(w\) are the barycentric coordinates of this point within this triangle. The resampled directions \([\theta_p, \phi_p]\) are the regular grid shown in Figure 11(c). If we collapse 2D indices \([r_p, s_p], [\theta_q, \phi_q]\), and \([\theta_v, \phi_v]\) into one dimension respectively similar to Equation 5, we can rewrite Equation 11 using 2D matrices as follows

\[
F^v_i = I^v_i^j W^v_i^j.
\]

Since each column of matrix \(W^v_i\) contains at most 3 non-zero rows, the above computation can be computed efficiently using sparse matrix multiplication routines.
RESULTS

We have acquired objects with diverse and complex reflection properties. The Van Gogh bust shown in Figure 17 is approximately one foot tall. The simplified surface geometry does not contain details such as chisel marks, but our experiments show that these details are modelled quite well by using images only. The dancer shown in Figure 16 has a metallic look except on the blouse and the skirt of the model. The topology of this object introduces interesting effects such as self shadowing. The star shown in Figure 18 is approximately 1/2 feet tall and made out of glass covered with twirled engravings and thin layers of paint. Depending on the viewing angle, it is either semi-transparent or anisotropically reflective. Such reflectance properties are difficult to model analytically. The toy turtle shown in Figure 15 is covered with a velvet-like material that is normally difficult to represent using traditional techniques.

Light Field Mapping is inherently scalable to larger scenes, because each vertex light field function and process each function independently. Figure 1 shows a synthetic surface light field composed with scanned objects. Images from different view points are rendered using a commercial renderer 3D Studio Max™. These images are then treated as input photographs for the LFM data processing pipeline. The resulting scene can be rendered at interactive rate, or at approximately one thousand times faster than the commercial renderer used to generate input images. Figure 19 shows the 3–term PCA approximation of an office scene scanned using our large-scale acquisition system. This scene can also be rendered at interactive rates on a PC.

PARTITIONING AND APPROXIMATION

We experimented with both vertex-centered and triangle-centered partitioning methods, as illustrated in Figure 12. In this experiment, we use rendered synthetic images to exclude artifacts caused by acquisition error. Clearly, triangle-centered approximations produce discontinuity artifacts across triangle boundaries. After adding more approximation terms, such artifact becomes less obvious but still visible. This problem cannot be corrected by adding more approximation terms. On the other hand, vertex-centered partitioning does not exhibit similar artifact even when only one approximation term is used.

To measure the quality of surface light field approximations, we calculate the RMS difference between the original and approximated light field matrices. We also calculate Peak Signal-To-Noise Ratio (PSNR), a commonly used image quality metric directly related to RMS error by
PSNR = 20 \log_{10} \left( \frac{I_{\text{max}}}{\text{RMS}} \right), \quad (13)

where $I_{\text{max}}$ represents the maximum pixel intensity. Figure 13 shows the approximation quality for both PCA and NMF algorithms. For each object, a PSNR is calculated over all of its vertex light field matrices. In this figure, we use 24–bit RGB pixels and therefore $I_{\text{max}} = 2^8 - 1 = 255$. As shown in the figure, both techniques provide high quality approximations using very small number of terms. Between the two algorithms, PCA produces better quality than NMF. However, the difference is visually almost indistinguishable.

**Rendering**

Figure 14 compares the rendering performance of PCA-based and NMF-based approximations. On this platform, rendering a full-range
multitexturing modulation requires 2 rendering passes. Excluding the diffuse layer, we need 6K rendering passes for a K-term PCA approximation, whereas only 3K passes is required for a K-term NMF approximation. We observe that NMF-based rendering is 50% faster than PCA-based for the same number of approximation terms. The performance disparity would be larger if the target platform supported only positive texture values. The rendering performance 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 field maps are equally fast if image sets in both cases fit into the texture cache.

Compression

Table 1 provides information on the data sizes of the models used in the experiments. We use 24-bit RGB images in all the experiments. The input image size in this table represents the total size of input images from the acquisition system. The effective image size represents the size of all foreground pixels of the input images. The resampled data size represents the size of actual surface light field function used for approximation. In this

![Figure 14. 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.](image)

<table>
<thead>
<tr>
<th>Models</th>
<th>Number of Triangles</th>
<th>Input Image Size</th>
<th>Effective Image Size</th>
<th>Resampled Data Size</th>
<th>Sampling Density (θ, φ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bust</td>
<td>6531</td>
<td>339</td>
<td>2.5GB</td>
<td>289 MB</td>
<td>32×32</td>
</tr>
<tr>
<td>Dancer</td>
<td>6093</td>
<td>370</td>
<td>2.7 GB</td>
<td>213 MB</td>
<td>32×32</td>
</tr>
<tr>
<td>Star</td>
<td>4960</td>
<td>282</td>
<td>2.1 GB</td>
<td>268 MB</td>
<td>32×32</td>
</tr>
<tr>
<td>Turtle</td>
<td>3830</td>
<td>277</td>
<td>1.7 GB</td>
<td>233 MB</td>
<td>32×32</td>
</tr>
</tbody>
</table>
process, viewing directions are resampled on a 32×32 pixel grid. The resulting resampled data size is approximately twice as large as the input images. Traditionally, research on light field compression reports results based on the size of the resampled light field function [5,34], and therefore we calculate the compression ratio also based on the resampled data.

Table 2 lists the size and compression ratio of the light field data obtained through light field maps approximation and additional compressions of the light field maps. For all the objects, the size of the geometric data falls below 10KB when compressed using topological surgery [45] and therefore is negligible compared to the size of light field maps. By combining VQ with S3TC hardware texture compression, our method achieves 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.

Figures 15–18 compare the rendering quality of our routines against the input images and report the corresponding image errors. The errors re-

<table>
<thead>
<tr>
<th>Models</th>
<th>Light Field Maps (3–term)</th>
<th>Compression of Light Field Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VQ</td>
<td>S3TC</td>
</tr>
<tr>
<td>Bust</td>
<td>47.7 MB (106:1)</td>
<td>5.7 MB (885:1)</td>
</tr>
<tr>
<td>Dancer</td>
<td>35.7 MB (82:1)</td>
<td>5.4 MB (542:1)</td>
</tr>
<tr>
<td>Star</td>
<td>42.3 MB (122:1)</td>
<td>7.2 MB (716:1)</td>
</tr>
<tr>
<td>Turtle</td>
<td>31.7 MB (121:1)</td>
<td>4.5 MB (847:1)</td>
</tr>
</tbody>
</table>

Figure 15. 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.
Figure 16. 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 17. 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.

Reported in the figures are computed based on the differences between the input images and the rendered images using both Average Pixel Error (APE) and PSNR for the foreground pixels only. The image errors are larger than the approximation error in Figure 13 due to the resampling process. In this process, if samples from the input images do not coincide with the viewing parameter grid, the original samples is lost and we can not reconstruct input images from the resampled surface light field functions.
<table>
<thead>
<tr>
<th>Approximation Level</th>
<th>APE</th>
<th>PSNR (dB)</th>
<th>Memory (MB) (Compression Ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Photograph</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-term PCA</td>
<td>4.52</td>
<td>30.48</td>
<td>466.5 (78:1)</td>
</tr>
<tr>
<td>4-term PCA</td>
<td>4.86</td>
<td>30.06</td>
<td>54.4 (95:1)</td>
</tr>
<tr>
<td>3-term PCA</td>
<td>5.34</td>
<td>29.49</td>
<td>42.3 (122:1)</td>
</tr>
<tr>
<td>2-term PCA</td>
<td>6.07</td>
<td>28.64</td>
<td>30.2 (171:1)</td>
</tr>
<tr>
<td>1-term PCA</td>
<td>7.42</td>
<td>27.27</td>
<td>18.1 (285:1)</td>
</tr>
</tbody>
</table>

Figure 18. The figure demonstrates the progressive nature of PCA approximation. The same star model is rendered using different number of approximation terms.

Figure 19. A real office scene acquired using our large-scale acquisition system and rendered with 3-term PCA light field maps approximation. The physical dimensions of the office are approximately 15ft(W)×10ft(D)×8ft(H).
CONCLUSIONS 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 data factorization that produces positive only factors. This method allows faster rendering using commodity graphics hardware. Furthermore, this article 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.

Our current implementations of light field maps compression algorithms only demonstrated part of a variety of possible algorithms, and this issue certainly deserves more attention. One of the limitations of the surface light field is that it parameterizes only the outgoing radiance of the data. We are planning to work on extending our work 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.

ACKNOWLEDGEMENTS

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 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. A significant portion of this article appeared in [10] and we thank Radek Grzeszczuk and Jean-Yves Bouguet for their significant contributions.

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 a Link Foundation Fellowship.
REFERENCES


