SIGRAD 2011

Geometry Independent Surface Light Fields for Real Time Rendering of Precomputed Global Illumination

E. Miandji[†1] and J. Kronander[1 ‡] and J. Unger[1 §]

[1]Linkoping University, Sweden

Abstract

We present a framework for generating, compressing and rendering of Surface Light Field (SLF) data. Our method is based on radiance data generated using physically based rendering methods. Thus the SLF data is generated directly instead of re-sampling digital photographs. Our SLF representation decouples spatial resolution from geometric complexity. We achieve this by uniform sampling of spatial dimension of the SLF function. For compression, we use Clustered Principal Component Analysis (CPCA). The SLF matrix is first clustered to low frequency groups of points across all directions. Then we apply PCA to each cluster. The clustering ensures that the within-cluster frequency of data is low, allowing for projection using a few principal components. Finally we reconstruct the CPCA encoded data using an efficient rendering algorithm. Our reconstruction technique ensures seamless reconstruction of discrete SLF data. We applied our rendering method for fast, high quality off-line rendering and real-time illumination of static scenes. The proposed framework is not limited to complexity of materials or light sources, enabling us to render high quality images describing the full global illumination in a scene.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

The ongoing pursuit for virtual realism has incited many researchers for efficient and accurate modelling of the interaction of light between surfaces. The complexity of analytical models for spatially varying surface properties limit their usage for real-time rendering. Due to this limitation, many Image Based Rendering (IBR) techniques were introduced to directly acquire the appearance of a scene through captured images [Zha04]. A successful appearance description model, Surface Light Field (SLF), was introduced in [MRP98]. The SLF function is defined as $f(r,s,\theta,\phi)$; where r and s are parametric coordinates for addressing a point on a surface. θ and ϕ are used for representing a direction in spherical coordinates. Depending on the sampling density of this function, the data generated by this method easily exceeds the capabilities of modern hardware even for a moderately detailed scene. Therefore various compression methods has been widely used to reduce the SLF size for rendering.

In the context of radiometry in computer graphics, one can see a surface light field as a set of exitant radiance values for each point on a scene along every possible direction. The radiance can be resampled data from High Dynamic Range (HDR) images or computer generated radiance data based on physically based rendering techniques. In the case of computer generated radiance data, by placing a virtual camera anywhere in the scene we can simply look up the SLF data based on intersection point of the viewing ray with the scene and the direction of it. We utilize this observation in order to present a SLF-based framework for fast real-time rendering of static scenes with all-frequency view-dependent radiance distribution. Our method (Section 3) is divided into three stages: data generation (Section 3.1), compression (Section 3.2) and rendering (Section 3.3). We remove the resampling stage from [CBCG02] and instead compute the outgoing radiance of uniformly sampled points

† ehsmi345@student.liu.se
‡ joel.kronander@liu.se
§ jonas.unger@itn.liu.se

on each surface in the scene along different directions. We then compress the SLF data using Clustered Principal Component Analysis (CPCA), similar to [SHHS03]. The renderer can efficiently decompress and reconstruct the SLF data on the GPU (Section 4). We will present our results for a real-time and an offline renderer (Section 5)

Since our SLF approximation method is not based on surface primitives (vertices, faces and edges), having low tessellated geometries does not affect the rendering quality. Instead we use uniform sampling of surfaces which leads to decoupling of lighting from geometric complexity. For instance, a highly tessellated portion of a polygonal mesh may be a diffuse reflector having uniform radiance values. In this case, a lot of memory is dedicated for a very low frequency lighting data. Since lighting complexity is purely scene dependent and cannot be determined before rendering (specially for glossy and specular objects), we use dense uniform sampling to ensure that we do not under-sample SLF function. Then we cluster this data, taking advantage of the fact that for most scenes the within-cluster frequency is low [MSRB07], therefore allowing us to approximate them with low order of principal components.

The gap between the image quality of photo realistic offline renderers and the state of the art real-time renderers is our main motivation. Complexity of certain materials at micro-scale is beyond the capability of current hardware for real-time rendering using analytic solutions. Our proposed framework is not limited to complexity of materials or light sources. Our method supports first order lighting from any type light source, e.g. point, spot and area lights. The compression and rendering stages allow us to render all-frequency view-dependent lighting effects in real-time for static scenes. To summarize, we state our main contributions:

1. A flexible representation of SLF that decouples radiance data from geometric complexity.
2. Application of CPCA in efficient compression of SLF data while preserving view-dependent high frequency details.
3. An efficient real-time rendering method for CPCA generated data.
4. A fast power iteration method adapted for CPCA

2. Related Work

In computer graphics, the ultimate goal is to sample a dataset (3D world) and then reconstruct or equivalently render this data. There are two models for this purpose; *source description* and *appearance description* [Zha04]. The former requires mathematical models in order to describe the 3D world. Reflection models, geometric models such as polygonal meshes and light transport models are examples of source descriptors. The data for this model is computed using mathematical models and during rendering they are reconstructed. The latter is based on capturing data from a real environment using cameras or similar equipment. The plenoptic function [AB91], defined as $l^{(7)}(V_x, V_y, V_z, \theta, \phi, \lambda, t)$, is used for representing such model; the first three arguments define a point in space where a camera is placed, θ and ϕ define a direction, λ is the wavelength of the light rays towards the camera and t represents time. The goal of many Image Based Rendering (IBR) techniques is to simplify this function by applying certain assumptions for practical sampling and rendering [Zha04].

Surface light fields was first introduced in [MRP98] as an IBR technique for visualizing results of precomputed global illumination. They formulated this representation for closed parametric surfaces but used polygonal surfaces for practical sampling of surfaces. Spatial samples were placed on vertices and interpolated over the triangle. For directional samples they subdivide a polygon if it is more that eight pixels in screen space. By representing the SLF data as an array of images, block coding techniques were utilized for compression. Our method is similar to [MRP98] regarding the utilization of precomputed global illumination results, but differs in data generation, compression and rendering.

Chen et. al. [CBCG02] introduced a new approximation of SLF by using vertex-centered partitioning. Each part was projected into lower dimensional functions using PCA or NMF, resulting in a surface map and a view map. They tile and store surface and view maps in textures and compress the results further using Vector Quantization (VQ) and standard hardware accelerated texture compression methods. Unlike [MRP98], Chen et. al. used 3D photography for acquiring a set of images. This technique is based on using geometric models to assist re-sampling of captured images in order to evaluate the SLF function. Utilizing hardware accelerated interpolation between SLF partitions, they could achieve real-time performance. Similarly, in [WBD03] a bi-triangle or edge-based partitioning was introduced.

Lambert et. al. [LDH07] propose a sampling criterion in order to optimize the smoothness of outgoing radiance in the angular domain of SLF. This criterion eliminates the use of actual surface in the SLF definition, replacing the surface with a parametrization of SLF function. A seminal work in SLF rendering was introduced in [WAA*00]. They propose a framework for construction, compression, rendering and editing SLF data acquired through 3D photography. The compression was performed using a generalizations of VQ and PCA. The framework can achieve interactive performance on the CPU using a new view-dependent level-of-detail algorithm. The editing supports linear modification of surface geometry, changes in reflectance properties and transformation relative to the environment.

The study of compression methods is not limited to IBR literature. A machine learning compression method [KL97, TB99] was employed in [SHHS03] for compressing the precomputed radiance transfer (PRT) data. Referred

to as CPCA, it is a combination of VQ and PCA. The signal is partitioned into clusters and transformed to an affine subspace. Compressing radiance transfer data is an active research field. In [MSRB07], an insightful study was performed to determine how light transport dimensionality increases with the cluster size. They show that the number of basis functions for glossy reflections is augmented linearly relative to cluster size. This study resulted in determining the optimal patch size for all-frequency relighting of 1024×1024 images. Ruiters and Klein applied tensor approximation to compress Bidirectional Texture Functions (BTF) with a factor of 3 to 4 better than PCA [RK09]. Tensor approximation was also used for compression of PRT data [TS06].

A thorough review of PRT can be found in [KSL05, Ram09]. Compared to these methods, our approach supports first order lighting from any type of light source such as point, spot and area lights. Although recent research such as [KAMJ05] add local lighting support to PRT, our method has this advantage inherently. Additionally, it can be used for all-frequency view-dependent effects that cannot be rendered faithfully with PRT because of projection on SH basis functions.

3. Method

In this section we present a detailed overview of our SLF compression and real-time rendering method. The technique can be outlined by the following steps:

1. We start by uniformly sampling the surface of a mesh in texture space, creating a set of points. For each point, we generate a number of directions on the unit sphere centered at the point. Then we evaluate outgoing radiance.
2. The data is clustered based on the difference between the radiance of points in all directions. We apply PCA on each cluster and store the results to disk
3. During rendering and for each ray, we fetch and decompress the radiance data corresponding to intersection point and the direction of ray

In the following subsections (3.1, 3.2 and 3.3) we will discuss each of the three main steps in more detail.

3.1. Data Generation

Sampling the SLF function requires discretization of spatial and directional parameters. We choose the two dimensional texture space for parameterizing the spatial coordinates. This is shown in Figure 1 (a) and (b). Given the number of samples for each coordinate of texture space, we use a rasterizer to determine world space points on a surface corresponding to sampled texture coordinates. The rasterizer uses barycentric coordinates to determine the world space coordinate of a point inside a triangle by interpolating vertex positions based on texture coordinates.

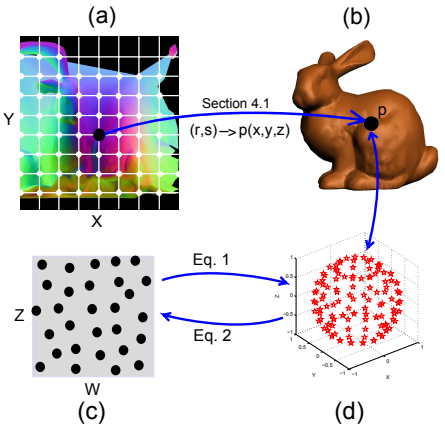

Figure 1: *A schematic illustration of data generation stage. (a) is texture space of the Stanford bunny model discretized with resolution X and Y, (b) illustrates a world space point on the mesh corresponding to a sample (r,s). The parameter space of directions sampled uniformly with resolution Z and W is shown in (c). And (d) illustrates a set of directions on the unit sphere.*

For sampling the unit sphere we need a two dimensional space of variables on the interval $[0,1]$ with respect to solid angle [PH04]. We define samples in this space and map them to 3D directions on the unit sphere. During rendering, the inverse mapping should be applied for SLF addressing. The forward and inverse mapping are shown in Figure 1 (c) and (d). For this purpose, we use the Latitude-Longitude formula in [RHD*10]. The forward mapping is expressed as:

$$\begin{aligned}(\theta,\phi) &= (\pi(\xi_1 - 1), \pi v),\\ (D_x, D_y, D_z) &= (sin\phi sin\theta, cos\phi, -sin\phi cos\theta),\end{aligned} \tag{1}$$

where $\xi_1 \in [0,2]$ and $\xi_2 \in [0,1]$ are uniform variables with constant spacing. ξ_1 and ξ_2 are mapped to azimuth and elevation angles, respectively. The bigger interval for ξ_1 is handled explicitly by multiplying a set of uniform variables on the interval $[0,1]$ by 2. Consequently, for backward mapping we have:

$$(\xi_1, \xi_2) = (1 + \frac{1}{\pi} atan2(D_x, -D_z), \frac{1}{\pi} arccos D_y) \tag{2}$$

Also note that we sample a sphere instead of a hemisphere. This eliminates the need for converting every ray to local coordinate frame of a point during rendering with the drawback of having larger data. Since the compression method approximates all directions with a few principal components ($k << n$), this choice of sampling increases the rendering performance and does have a small impact on size of the data.

Having the position map and a set of outgoing directions for each point, we evaluate the outgoing radiance. We unwrap the discretized spatial (r and s) and directional dimensions (ϕ and θ) defining a matrix, F, where each row corre-

sponds to a surface point and each column represents a direction. In other words, each row contains radiance for a point along all directions; and each column represents radiance along a specific direction for all surface points. Although many representation of our 4D data are possible (such as tensors), this type of representation facilitates our compression stage where we cluster this matrix and apply PCA on each cluster. To dicretize wavelength λ, we create three matrices F_R, F_G and F_B each storing radiance data for a color component. Compression is applied to each matrix separately. In the remainder of this paper we ignore wavelength dimension and denote the SLF matrix as F.

3.2. Compression

In this section, we discuss the compression of the generated SLF. The requirements for compression are as follows:

1. *High compression ratio*: Because of the large amount of data stored in the SLF data structure, an algorithm with high compression ratio is vital for fitting this data in the system memory and ultimately the GPU memory.
2. *Faithful decoding and reconstruction of data*: A scene may include diffuse surfaces (low frequency radiance variation along the surface) along with highly specular surfaces or even caustics (which exhibit very high frequency radiance variations). The SLF contains high frequency data in both spatial and angular domain. This puts high requirements on a compression method that can reconstruct high frequency data faithfully.
3. *Random access*: During the rendering stage it is required that the data be accessed randomly. This means that the decompression method should be able to locally decode the data and return the requested radiance value in an effective manner.

The widely used PCA method exhibits high compression ratio and enables random access to data; yet it cannot reproduce high frequency angular variations of radiance present in specular and glossy surfaces. For diffuse surfaces it can reproduce smooth surfaces with only one principal component, satisfying all three requirements to a great extent.

To this end, we use Clustered Principal Component Analysis (CPCA). A fast and versatile algorithm adapted from machine learning literature [KL97, TB99] to the field of computer graphics by Sloan et al. [SHHS03]. CPCA is a two stage method of clustering followed by a PCA for each cluster. When there is only one cluster, it is equivalent to PCA. This method has a high compression ratio and will provide random access of compressed data in real-time and on the GPU. The clustering part of CPCA will compensate for the reconstruction error of the compressed all-frequency SLF data by creating clusters of low frequency radiance data; allowing us to recover view-dependent details present in specular and glossy surfaces. The quality of reconstruction is highly dependent on the number of principal components and clusters; which increases the flexibility of the algorithm.

We calculate the mean of each row of F and store it in a *mean map*, denoted as μ. Then we calculate the matrix of residuals, G, via

$$G = [x_{p1} - \mu_{p1}, x_{p2} - \mu_{p2}, \ldots, x_{p_m} - \mu_{p_m}]^T, \qquad (3)$$

The normalizing stage can also be done after clustering by subtracting the mean from each cluster. We cluster the SLF matrix using the K-Means method. Rows of G are treated as data items and columns as variables. The result of clustering is a set of cluster IDs with a size equal to the number of points. The outcome will be a set of matrices G_i of size $m_i \times n$ where $m_i \ll m$ is the number of points belonging to cluster i that have similar radiance distribution along all sampled directions. Note that m_i is not constant among all clusters. Each cluster may have different number of points.

Denoting a cluster's normalized SLF matrix as G_i, we write the Singular Value Decomposition (SVD) of it as $G_i = U_i D_i V_i^T$; where U_i is a $m_i \times k$ matrix of left singular vectors, V_i is a $n \times k$ matrix including right singular vectors and D_i is a diagonal matrix of singular values sorted in decreasing order. When $k < N$ we achieve a least-squares optimal linear approximation $\hat{G}_i = U_i D_i V_i^T$ of G_i where the approximation error, [SHHS03], is:

$$\sum_{j=1}^{m_i} \|x_{p_j} - \hat{x_{p_j}}\|^2 = \sum_{j=1}^{m_i} \|x_{p_j} - \mu_{p_j}\|^2 - \sum_{j=1}^{k} (D_i)_j^2 \qquad (4)$$

This decomposition can be thought of as selecting a set of orthonormal eigenvectors representing a subset of directions and linearly approximating every other direction by calculating a weighted sum of eigenvectors. The weights are rows of $U_i D_i$ and principal components are rows of V_i. Since we are interested in the first few approximation terms ($k \ll n$), we use Power iteration method [CBCG02]. This method is well-suited for our technique for the following reasons:

- This method can iteratively calculate first k terms instead of performing a full SVD. This saves a lot of computation time
- The approximation is improved by adding more terms without the requirement for recalculating previous terms.
- The memory footprint is very low. A careful implementation of Algorithm 1 (see Appendix) will require extra memory allocation of size $n^2 + m + 2n$ when $m \gg n$. This is constant for all approximation terms.

In this method, left and right eigenvectors of the covariance matrix is calculated for each term, and iterated for all approximation terms k. The traditional power iteration method presented in [CBCG02] assumes that $m_i \gg n$, therefore the covariance matrix, $G_i^T G_i$, is of size $n \times n$. In the clustered PCA method [CPCA], it is very likely that each cluster will contain fewer points than directions. This will increase the PCA computation time dramatically since we have to apply PCA on each cluster. Instead, we modify power iteration according to [SHHS03], handling situations when a

cluster has more variables rather than data items. The modified power iteration algorithm is presented in the Appendix.

We denote U_i as the surface map of cluster i and V_i as view map of cluster i. The number of points within clusters add to total number of points for a shape. Therefore we create one surface map U for each shape. On the other hand, each cluster has a unique view map V_i. In order to find the corresponding view map, we add a cluster index for each point in the surface map. Defining the size operator as Γ, the total memory consumption (γ) for this representation is

$$\begin{aligned}\gamma &= \Gamma(U)+\Gamma(V)+\Gamma(\mu)+\Gamma(\chi),\\ &= [\zeta mk]+\sum_{i=1}^{\eta}[\zeta nk]+[\zeta m]+[m],\end{aligned} \tag{5}$$

where χ is a vector containing cluster indices, ζ is number of color components and η is number of clusters.

3.3. Rendering

The rendering method presented here can be easily implemented for both CPU-based (offline) and GPU-based (real-time) renderers. In the offline case, a ray caster is used for shooting rays from the camera into the scene and extracting the corresponding radiance value of the screen sample being processed. The same procedure is applied to the real time renderer but with the difference that we fetch and reconstruct data in a pixel shader based on the view vector. The data is stored in volume textures for cache efficient access.

In order to get radiance of a point along a viewing ray, we need to reconstruct the SLF matrix, F, which is a discretization of the SLF function, $f(r,s,\theta,\phi)$:

$$f(r,s,\theta,\phi) \approx F[r,s,u,v] = \left(\sum_{i=1}^{\eta} U_i V_i^T\right)+\mu, \tag{6}$$

where U_i corresponds to the surface map of cluster i and V_i is the view map of cluster i. Here $[r,s]$ and $[u,v]$ are parametric space coordinates of a point and direction which are mapped to address the SLF matrix.

One advantage of this representation is that we can directly compute an element in F without the need for complete reconstruction of it. We define α and β to be a row in surface map and view map, respectively. In this way, α represents a surface map row index of a point (r,s) and β represents a view map row index of a direction (θ,ϕ). Then, we can compute the SLF matrix element $F[r,s,u,v]$ as

$$F[r,s,u,v] = \left(\sum_{j=1}^{k} U_i[\alpha,j]V_i[\beta,j]^T\right)+\mu[\alpha], \tag{7}$$

where $U_i[\alpha,j]$ is an element in the surface map of cluster i at row α and column j; similarly, $V_i[\beta,j]$ is an element in view map of cluster i at row β and column j. The summation in Equation (7) is an inner product between a row in U_i and V_i. Note that in practice we have one surface map U and several view maps V_i, $1 < i < \eta$. Hence, $U_i[:,j]$ with $j = 1\ldots k$, represents all the principal vectors inside U that have cluster ID equal to i. Additionally, Equation 7 allows us to render a shape in one pass. It is not required to render each cluster separately (Section 4). As a result, the need for super-clustering is removed. This technique was introduced in [SHHS03] to reduce the overdraw of a triangles with vertices belonging to different clusters.

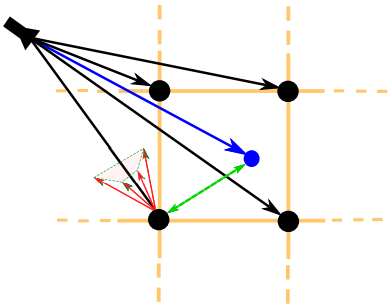

Figure 2: *Schematic view of SLF reconstruction during rendering*

To apply Equation (7) to a ray tracer or a GPU based scanline renderer, we need to convert a point $p(x,y,z)$ and a direction $d(x',y',z')$ to surface and view map row indices (α and β). To address surface map, we fetch texture coordinates of p (the intersection point). Then we find four nearest points in the surface map. Due to uniform sampling, this can be easily done by clamping or rounding $\iota_1 X$ and $\iota_2 Y$, where ι_1 and ι_2 are interpolated texture coordinates of point p; X and Y are sampling resolution of texture space. The final surface map value for p is calculated by bi-linear interpolation of four neighboring points. As described in Figure 2, the blue arrow and circle correspond to the main view vector and its intersection point with the surface, respectively. The black arrows and circles are four nearest neighbors to the intersection point. The weights are calculated as the inverse of the Euclidean distance between (ι_1,ι_2) for main intersection point and the four nearest points. This is shown for a point in Figure 2 as the two-sided dashed green line.

To address the view map, we apply a similar procedure. We use the same neighboring points in order to calculate four direction vectors. For the main direction and each of the four additional direction vectors, we first calculate parameter space coordinates of them by applying Equation (2), getting ξ_1 and ξ_2. Then, each (ξ_1,ξ_2) coordinate set is discretized as $(\xi_1 Z,\xi_2 W)$, again resulting in four view map indices for each. This is shown as red arrows for one of the nearest points in Figure 2. For simplicity of the illustration, we did not include red arrows for other neighboring points. In order to interpolate view map values of a point, the weights are calculated as the difference between the main direction vector and the four nearest direction vectors (red arrows in Figure 2). This is expressed as $w_i = e^{d.d_i}$, for $i = 0\ldots 3$; where $|.|$ represents a dot product and d_i are nearest direction vectors.

The final view map value of the main direction is a weighted average with weights calculated before for spatial interpolation. This type of interpolation leads to seamless reconstruction of the SLF function (Section 5).

4. Implementation

The data generation stage was implemented as a renderer (C++ class) in PBRT [PH04]. We chose to implement a renderer rather than a surface integrator in order to experiment with different surface integrators. The renderer's input is a set of shapes flagged as SLF and non-SLF. For a SLF shape, we generate SLF matrix, compress it and then store both compressed and non-compressed data to disk. If a shape is flagged as non-SLF, we ignore it during data generation. Data generation parameters are divided in two parts. Parameters such as sphere sampling and compression methods that are unique for all shapes are provided by the renderer. On the other hand, $[X,Y,Z,W,k,\eta]$ are provided per shape. Therefore one can define various settings based on material complexity of a shape. We will use the same notation when presenting our rendering results. The renderer first computes the position map, a two dimensional array of points (Section 3), followed by a bucket of outgoing directions for each point. The point and the bucket of directions are given to a thread for computing outgoing radiance values using a surface integrator. Afterwards, the data is stored in F_R, F_G and F_B at a row corresponding to the point; then the thread terminates. This is iterated XY/q times, where q is the number of available processors.

The K-Means was performed by an optimized multiprocessor implementation, provided by Wei-keng Liao (http://users.eecs.northwestern.edu/~wkliao/). We implemented the modified power iteration (Algorithm 1) using the Eigen 3 library (http://eigen.tuxfamily.org) for matrix and vector algebra. The error tolerance, ε, was set to 1e-12 and the maximum number of iterations, c, to 1000. The parameter c ensures finite loops and since we handle numerical fluctuations by monitoring the error, it is guaranteed that $\mathbf{v}_p$ for $p = c$, will have the least error.

We implemented a PBRT based renderer for offline rendering and a GPU based renderer using DirectX. For offline rendering, we do not use a surface integrator since the incoming radiance to the camera can be directly acquired from compressed SLF data. Providing a scene containing SLF and non-SLF shapes, we compute radiance for rays that intersect SLF shapes. To evaluate outgoing radiance for non-SLF shapes, we evoke the default integrator, e.g. path tracing. Our implementation does not support light transport between SLF and non-SLF shapes although they can exist in the same scene.

For real-time rendering, we store surface and view maps in 3D textures. Each slice of this texture contains an approximation term for a surface or view map. We can also store each approximation term in separate textures. But due to hardware limitations, this will limit the number of PCA terms, k, to a small value. Another advantage of using a volume texture is that we can change the number of PCA terms in real-time; that is, specifying a smaller value than k in Equation 7 or equivalently sampling fewer slices from the volume texture. As mentioned earlier, the compression stage generates one surface map and η view maps, where η is the number of clusters. Therefore, the size of the 3D surface map will be $X \times Y \times k$. For the 3D view map the size is $Z \times W \times \lceil\sqrt{\eta}\rceil$.

We tile individual view maps for each cluster in a sequential order. The same pattern is used for additional slices. There are two ways for addressing individual view maps inside the 3D view map during rendering. We can create a 2D texture, cluster map, of size $\lceil\sqrt{\eta}\rceil^2$ with two components where each element points to the top-left corner of a view map across all slices. Correspondingly, we can calculate the address in the pixel shader given η, Z and W. The first method is less expensive considering the fast texture lookup in modern hardware; it is also more straightforward to implement.

We implemented the real-time renderer utilizing DirectX 9.0. The surface, view and mean maps are 128-bit, 4-channel floating point textures. Cluster IDs are encoded in alpha channel of the mean map while the value for $\lceil\sqrt{\eta}\rceil$ is provided as a global parameter. Consequently, after fetching a mean map texel and extracting its alpha value, we can sample the cluster map. Returned values are texture addresses for top-left corner of a tile in view map. Having this base address, we fetch the exact texel inside a tile by adding the scaled ξ_1 and ξ_2 values in Equation 2. Because of HDR textures, the real time renderer requires a tone mapping operation as the final stage.

5. Results

Rendering results of our method is shown in Figure 3. The scene consists of a diffuse Cornell box, a diffuse torus and two specular spheres. For specular surfaces, the roughness parameter is set to 0.02. The mesh for the pink specular sphere is distorted intentionally. The left wall is textured with a relatively high frequency texture. As it can be seen in Figure 3, our uniform sampling method can reconstruct this texture with minimum aliasing or noise, despite the fact that it has four vertices only. Previous methods based on surface primitive partitioning of SLF fail on cases like this. Data generation and compression parameters for each shape is included in Table 1. In addition, we used a box filter of width 5 for interpolating radiance along super sampled sphere directions (Section 3.1). Sphere samples were generated using low-discrepancy sampler in PBRT. For the reference image we used the photon mapping integrator with 400000 photons and 64 final gathering samples. The resolution was set to 1024×1024 and we used 256 samples per pixel. Our CPU based renderer was configured accordingly but with the difference that we used only 8 samples per pixel. This value

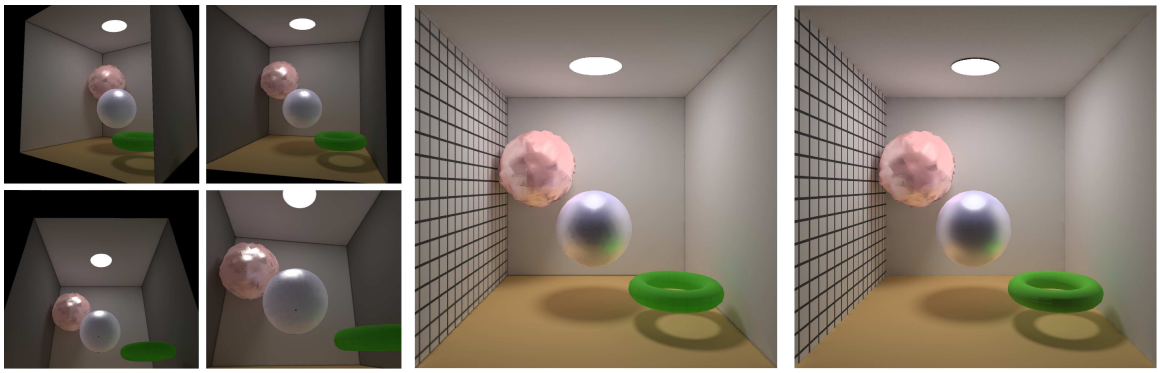

Figure 3: *Rendering results for our GPU based renderer (a), the CPU based renderer (b) and the reference image (c)*

Table 1: *Per-shape parameters and timing results for three stages of our method.*

Shape	X	Y	Z	W	k	η	Data Gen.	Comp.	CPU Rend.	GPU Rend.	Ref. Rend.
Walls (average)	256	256	32	16	1	1	144 min	1.7 sec	-	-	-
Torus	256	256	32	16	1	1	58 min	1.7 sec	-	-	-
Blue sphere	256	256	64	32	8	256	946 min	24 min	-	-	-
Pink Sphere	256	256	64	32	8	256	832 min	27 min	-	-	-
The scene	-	-	-	-	-	-	1980 min	51 min	16.7 sec	120 FPS	202 min

was enough since our rendering method internally interpolates radiance in spatial and angular dimensions. The image resolution for GPU renderer was set to 1920×1080.

Our performance results for three stages of the algorithm are illustrated in Table 1. The rendering parameters are the same to those mentioned earlier and the data generation stage uses the same photon mapping parameters. As illustrated, for diffuse surfaces we used the least amount of clusters and PCA terms ($k = 1$ and $\eta = 1$). For specular shapes, we set $k = 8$ and $\eta = 512$. Although we can increase k and reduce η while getting the same image quality, according to Equation 7 it will result in severe performance lost. Additionally, increasing k will affect the size of view map and surface map (Equation 5) while η only has impact on the view map. Comparing the rendering time for the reference renderer with our CPU renderer, we can conclude that our renderer can be used for fast realistic rendering of novel views of static scenes. We tested our results using a PC with a quad-core Intel Xeon processor and a NVIDIA GeForce 8800 Ultra. Using a PC with more cores will improve data generation and compression performance linearly due to the fact that all the stages utilize parallel processing.

6. Conclusions and Future Work

We presented a framework for creating, compressing and rendering SLF data that can be used for viewing static scenes with global illumination effects. Our SLF representation decouples radiance data from geometric complexity by using uniform sampling of spatial dimension. We also showed that the application of CPCA on SLF data can lead to relatively high compression ratios while preserving view-dependent high frequency details. Additionally, we presented an efficient rendering algorithm that can be used for real-time or fast off-line rendering of scenes with complex materials. Although we focused on computer-generated radiance data, the compression and rendering algorithm can be simply applied to re-sampled data of captured digital images.

Our future work is mainly concentrated on compression, rendering and interpolation. We seek to analyse various compression techniques on SLF data. Of course this is dependent on the representation of discretized SLF function. Whether we see it as a tensor or matrix, different compression techniques can be applied. Wavelet analysis, Tensor approximation [RK09, TS06] and sparse representations [RK09, BE08] have been successfully applied for compression. Using our presented representation, we can compress the surface map and the set of view maps further by utilizing aforementioned techniques. Having a smaller compressed data with minimal loss allows us to increase the size of uncompressed SLF matrix by sampling more densely, leading to better image quality with little rendering overhead.

References

[AB91] ADELSON E. H., BERGEN J. R.: The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, Landy M. S., Movshon A. J., (Eds.). MIT Press, Cambridge, MA, 1991, pp. 3–20. 2

[BE08] BRYT O., ELAD M.: Compression of facial images using the k-svd algorithm. *J. Vis. Comun. Image Represent. 19* (May 2008), 270–282. 7

[CBCG02] CHEN W.-C., BOUGUET J.-Y., CHU M. H., GRZESZCZUK R.: Light field mapping: efficient representation and hardware rendering of surface light fields. *ACM Trans. Graph. 21* (July 2002), 447–456. 1, 2, 4

[KAMJ05] KRISTENSEN A. W., AKENINE-MÖLLER T., JENSEN H. W.: Precomputed local radiance transfer for real-time lighting design. *ACM Trans. Graph. 24* (July 2005), 1208–1215. 3

[KL97] KAMBHATLA N., LEEN T. K.: Dimension reduction by local principal component analysis. *Neural Comput. 9* (October 1997), 1493–1516. 2, 4

[KSL05] KAUTZ J., SLOAN P.-P., LEHTINEN J.: Precomputed radiance transfer: theory and practice. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM. 3

[LDH07] LAMBERT P., DESCHENES J.-D., HEBERT P.: A sampling criterion for optimizing a surface light field. In *3-D Digital Imaging and Modeling, 2007. 3DIM '07. Sixth International Conference on* (aug. 2007), pp. 47 –54. 2

[MRP98] MILLER G. S. P., RUBIN S. M., PONCELEON D. B.: Lazy decompression of surface light fields for precomputed global illumination. In *Rendering Techniques* (1998), Drettakis G., Max N. L., (Eds.), Springer, pp. 281–292. 1, 2

[MSRB07] MAHAJAN D., SHLIZERMAN I. K., RAMAMOORTHI R., BELHUMEUR P.: A theory of locally low dimensional light transport. *ACM Trans. Graph. 26* (July 2007). 2, 3

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 3, 6

[Ram09] RAMAMOORTHI R.: Precomputation-based rendering. *Found. Trends. Comput. Graph. Vis. 3* (April 2009), 281–369. 3

[RHD*10] REINHARD E., HEIDRICH W., DEBEVEC P., PATTANAIK S., WARD G., MYSZKOWSKI K.: *High Dynamic Range Imaging, Second Edition: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 3

[RK09] RUITERS R., KLEIN R.: Btf compression via sparse tensor decomposition. *Computer Graphics Forum 28*, 4 (jul. 2009), 1181–1188. 3, 7

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22* (July 2003), 382–391. 2, 4, 5

[TB99] TIPPING M. E., BISHOP C. M.: Mixtures of probabilistic principal component analyzers. *Neural Comput. 11* (February 1999), 443–482. 2, 4

[TS06] TSAI Y.-T., SHIH Z.-C.: All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph. 25* (July 2006), 967–976. 3, 7

[WAA*00] WOOD D. N., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D. H., STUETZLE W.: Surface light fields for 3d photography. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 287–296. 2

[WBD03] WENDONG W., BAOCAI Y., DDEHUI K.: Non-uniform light field compression: a geometry-based method for image processing. In *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on* (dec. 2003), vol. 2, pp. 1058 – 1061 Vol.2. 2

[Zha04] ZHANG C.: A survey on image-based renderingâĂŤrepresentation, sampling and compression. *Signal Processing: Image Communication 19*, 1 (jan. 2004), 1–28. 1, 2

Appendix: Modified Power Iteration

Let F be a $m \times n$ matrix that represents SLF data of a cluster. Computing SVD of F yields $F = UDV^T$. Our goal to compute a $m \times k$ matrix UD and a $n \times k$ matrix V in a way that $\hat{F} = UV^T$ best approximates original matrix F. The power iteration method computes the first k eigenvectors of the matrix $A = F^T F$, the covariance matrix of size $n \times n$. When $m < n$, we compute the $m \times m$ matrix FF^T. Now the eigenvectors are F's left singular vectors and the right singular vectors can be computed as $V^T = U^T D^{-1} F$.

Algorithm 1: Calculate $\hat{F} = UV^T$ where F is $m \times n$, U is $m \times k$ and V is $n \times k$

Require: $m > 0, n > 0, k > 0$, c is maximum number of iterations for v_p or u_p convergence and ε is the error tolerance
if $M \geq N$ **then**
  for $p = 1 \rightarrow k$ **do**
    $A_p \leftarrow F^T F$
    $\hat{\mathrm{v}}_p \leftarrow$ random $N \times 1$ non-zero values
    $\hat{\mathrm{v}}_p \leftarrow \hat{\mathrm{v}}_p / \|\hat{\mathrm{v}}_p\|$
    for $z = 1 \rightarrow c$ **do**
      $\mathrm{v}_p \leftarrow A_p \hat{\mathrm{v}}_p$
      $\lambda_p \leftarrow \|\mathrm{v}_p\|$
      $\sigma \leftarrow \|\mathrm{v}_p - \hat{\mathrm{v}}_p\|$
      if $\sigma < \varepsilon$ or $\sigma > \hat{\sigma}$ **then**
        break
      end if
      $\hat{\sigma} \leftarrow \sigma$
      $\hat{\mathrm{v}}_p \leftarrow \mathrm{v}_p$
    end for
    $\mathrm{u}_p \leftarrow F\mathrm{v}_p / \lambda_p$
    $max_u \leftarrow max(abs(\mathrm{u}_p))$ and $max_v \leftarrow max(abs(\mathrm{v}_p))$
    $\kappa \leftarrow \sqrt{max_u \lambda_p / max_v}$
    $\mathrm{u}_p \leftarrow \lambda_p \mathrm{u}_p / \kappa$
    $\mathrm{v}_p \leftarrow \kappa \mathrm{v}_p$
    $F \leftarrow F - \mathrm{u}_p \mathrm{v}_p^T$
    $U(:,p) \leftarrow \mathrm{u}_p$ *and* $V(:,p) \leftarrow \mathrm{v}_p$
  end for
else $\{N < M\}$
  for $p = 1 \rightarrow k$ **do**
    $A_p \leftarrow FF^T$
    $\hat{\mathrm{u}}_p \leftarrow$ random $M \times 1$ non-zero values
    $\hat{\mathrm{u}}_p \leftarrow \hat{\mathrm{u}}_p / \|\hat{\mathrm{u}}_p\|$
    for $z = 1 \rightarrow c$ **do**
      $\mathrm{u}_p \leftarrow A_p \hat{\mathrm{u}}_p$
      $\lambda_p \leftarrow \|\mathrm{u}_p\|$
      $\sigma \leftarrow \|\mathrm{u}_p - \hat{\mathrm{u}}_p\|$
      if $\sigma < \varepsilon$ or $\sigma > \hat{\sigma}$ **then**
        break
      end if
      $\hat{\sigma} \leftarrow \sigma$
      $\hat{\mathrm{u}}_p \leftarrow \mathrm{u}_p$
    end for
    $\mathrm{v}_p \leftarrow \mathrm{u}_p^T F / \lambda$
    $max_u \leftarrow max(abs(\mathrm{u}_p))$ and $max_v \leftarrow max(abs(\mathrm{v}_p))$
    $\kappa \leftarrow \sqrt{max_u \lambda_p / max_v}$
    $\mathrm{v}_p \leftarrow \lambda_p \mathrm{v}_p / \kappa$
    $\mathrm{u}_p \leftarrow \kappa \mathrm{u}_p$
    $F \leftarrow F - \mathrm{u}_p \mathrm{v}_p$
    $U(:,p) \leftarrow \mathrm{u}_p$ and $V(:,p) \leftarrow \mathrm{v}_p$
  end for
end if