# Modeling and Rendering of DPP-Based Light Fields

Miguel Escrivá, Alejandro Domingo, Francisco Abad, Roberto Vivó, Emilio Camahort
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{mescriva, adomingo, fjabad, rvivo, camahort}@dsic.upv.es

## Abstract

*Autostereoscopic displays are a subject of recent research efforts in Computer Graphics. Such displays have to be fed graphics information in order to produce spatial images. This information is typically 4D radiance data called a light field. Traditionally light-field models were based on the two-plane parameterization. In this paper, however, we present a light-field representation that is based on the direction-and-point parameterization. This parameterization has certain uniformity properties that produce better rendering results. We describe the files and data structures needed to store the representation, and we introduce a rendering algorithm that takes advantage of the uniformity properties of the direction-and-point parameterization. Our algorithm runs in real time and renders light-field models that look like their geometric counterparts.*

## 1. Introduction

Recently 3D spatial and autostereoscopic displays are receiving a lot of attention [1, 10, 12, 13, 2]. The goal is to provide multiple viewers with a 3D image of the object of interest. This usually requires rendering 3D volumetric data or 4D light-field data. We are concerned with 4D light-field representations like those used in autostereoscopic displays.

Formally, a light field represents the radiance flowing through all the points in a scene in all possible directions. For a given wavelength, we can represent a static light field as a 5D scalar function $L(x, y, z, \theta, \phi)$ that gives radiance as a function of location $(x, y, z)$ in 3D space and the direction $(\theta, \phi)$ the light is traveling.

In practice autostereoscopic devices display a 4D version of the light-field function [11, 8]. This version is typically based on the two-plane parameterization (2PP) that was originally inspired by holography and, specifically, by holographic stereograms [3, 9]. This choice of parameterization simplifies rendering by avoiding the use of cylindrical and spherical projections during the light-field reconstruction process. However, even 2PP models that rely on uniform samplings of the planes are known to introduce biases in the line sampling densities of the light field. These biases cannot be eliminated [4]. This problem is called the *disparity problem* and can only be solved by choosing a different parameterization.

In this paper we introduce a light-field representation that is based on the direction-and-point parameterization (DPP) and does not suffer from the disparity problem. Such a representation allows the user to move freely around an object without noticing any resolution changes in the model. This guarantees light-field invariance under rotations and translations. Our representation samples the light-field function by *uniformly* sampling the set of lines intersecting the object's convex hull. As an approximation to the convex hull we use a sphere tightly fit around the object.

Implementing a light-field model for Computer Graphics rendering is similar to implementing a computer model of any other function. It starts with a process that generates or captures a set of discrete samples to build the model. Samples are then organized and stored so that they can be efficiently retrieved for rendering.

A light-field implementation can be characterized by its representation, its storage scheme, and its construction and rendering algorithms. We present all of these in this paper. First we review previous work in light fields. Then we introduce our light-field representation and we detail its storage organization. In Section 4 we describe our light-field rendering algorithm. Finally, Section 5 presents results obtained with our representation. The paper finishes with some conclusions and directions for future work.

## 2 Previous Work

The first two light-field implementations were proposed by Levoy and Hanrahan [11] and Gortler et al. [8]. Both implementations are 2PP-based and discretize the light-field support by imposing rectilinear grids on both planes.

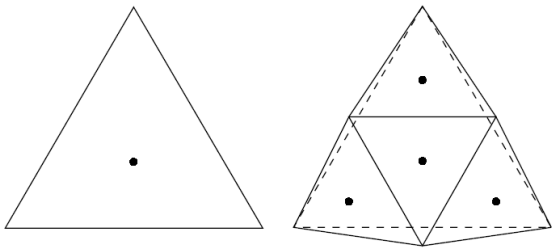


**Figure 1. Recursive tessellation of the sphere.** *Left,* **a triangle before subdivision, and** *right,* **the triangle after subdivision. Note that subtriangles' vertices have been slightly raised to meet the surface of the sphere; also, dots at the center of the triangles represent directional samples.**

The two-sphere parameterization (2SP) represents each line passing through an object by its two intersection points with a sphere tightly fit around the object [4]. Discretizations of the 2SP are based on nearly-uniform tessellations of the sphere that satisfy certain hierarchical multiresolution properties. For each ordered pair of spherical triangles a 2SP-based representation stores a light-field sample taken along the line passing through the triangles centers. The multiresolution properties of the sphere tessellation support filtering and construction of a mipmap-like structure for the light-field data.

## 3 Our Light-Field Representation

We build a light-field representation of a target 3D object enclosed in a tightly fit bounding ball. Our goal is to sample the light field for the lines that intersect the object's convex hull. We use the object's bounding ball as an easy-to-implement approximation of its convex hull.

This DPP implementation relies on a (nearly) uniform discretization of the set of all directions in 3D Cartesian space, a 2D space. As each point on a sphere's surface corresponds to a single direction, we can obtain such a discretization by subdividing the surface of the sphere into (nearly) equilateral, (nearly) identical spherical polygons.

A perfectly uniform tessellation produces $D$ spherical triangles each of area $4\pi/D$. Common uniform tessellations are those based on the platonic solids. The platonic solid with the most faces, the icosahedron, has 20 faces. To obtain finer tessellations we use recursive subdivision [6, 5, 7] (see Figure 1). Each edge of each triangle in the original icosahedron is divided into two equally long segments at its center point. That point is then projected out onto the sphere's surface to define a new vertex of the tessellation. All vertices are then connected by new edges that define four new subtriangles for each original triangle.

We can apply this subdivision process multiple times. Every time we obtain a new set of spherical subtriangles with nearly 1/4th the area of the triangles in the previous subdivision step.

We associate to each spherical triangle a planar triangle $T_k$ with the same vertices. Each triangle defines a pencil of directions $\Omega_k$ that starts at the center of the sphere and passes through the triangle. Each pencil $\Omega_k$ is approximated by a single directional sample $\vec{\omega_k}$ that starts at the center of the sphere and passes through the center of the triangle $T_k$. When a triangle is subdivided, only three new directional samples are created. The center subtriangle *inherits* the directional sample of its parent triangle, even though it may not pass exactly through its center.

### 3.1 Building a Light-Field Model

We now describe how to construct a model of an object. A simple DPP-based light field can be constructed from a synthetic model as follows. We center the target object at the origin and scale it to fit inside of the unit sphere. We choose the number of directional samples $D$ and build the sample set $\{\omega_k\}_{k=1}^{D}$ using subdivision of the icosahedron. For each direction $\vec{\omega_k}$ we render a parallel projection of the object onto a projection plane $P_k$ centered at the origin. The resulting image is stored as image $L_k$ of the representation. A depth map $D_k$ can also be obtained from depth information. All the images are stored in an image array.

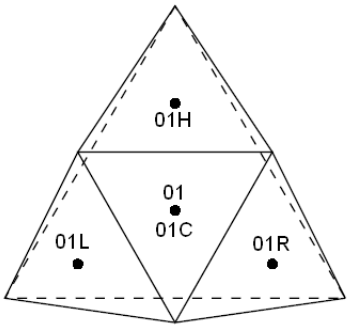
We use orthographic projections along the directions $\vec{\omega_k}$. The $y$ axes of the projections are chosen so that they are orthogonal to the $\vec{\omega_k}$'s and point upwards on the planes spun by each $\vec{\omega_k}$ and the world's $y$-coordinate axis.

### 3.2 Storing the Light Field

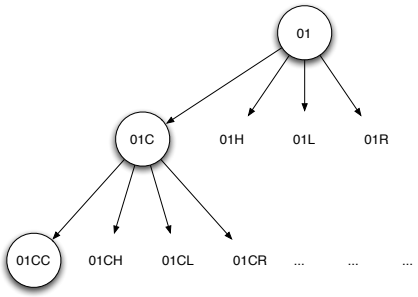
The coarsest level of the geodesic approximation, level 0, has 20 triangles. Each subsequent level of the approximation has 4 times more triangles than the previous one. Such a tessellation can be stored as a hierarchy of triangles or, equivalently, pencils with different resolutions.

We define an encoding for the triangles and the images in the hierarchy (see Figure 2(b)). Level 0 of a light-field model stores 20 images with ids between 01 and 20. The images correspond to triangles that can be tessellated to obtain 4 new subtriangles. Subtriangles are encoded with letters *C*, *H*, *L* and *R*. Figure 2(a) shows how the letters are assigned to the subtriangles:

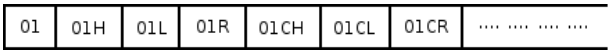
- **C** - Represents the center subtriangle (note that this subtriangle *inherits* the directional sample of its parent triangle).




(a) Encoding of the subtriangles of the original icosahedron triangle labeled *01*.




(b) The hierarchical structure of the subdivision: *01*, *01C* and *01CC* represent (roughly) the same direction, so they share the same image.




(c) Storage of the images both in memory and disk: the tree in (b) collapses to an image array, where *01C* and *01CC* do not appear because they share the image of *01*.

**Figure 2. The hierarchical structure, encoding and storage of the light field's image data.**

- **H** - Subtriangle located right above or below the center triangle (the *h*orizontal triangle).
- **L** - Subtriangle to the left of the center triangle.
- **R** - subtriangle to the right of the center triangle.

Our choice of discrete directions is adequate for multiple reasons. If we place the icosahedron so that two opposite vertices coincide with the north and south poles of the unit sphere, we entirely avoid the singularities that occur at $\phi = \pm\pi/2$. The subdivision process can only choose a directional sample at $\phi = \pm\pi/2$ when $D$ reaches infinity. Our experiments show that the assumption that the discretization is uniform, even though it is only close to uniform, has no noticeable effects on the light-field rendering and processing algorithms. And it substantially simplifies the light-field representation and the design of the algorithms.

We use two strategies to store a light field's image data.

1. We store the data in 20 files, one for each level-0 directional sample. Within each file, images are stored in breadth-first order, the order used to generate the set of directional samples (see Figure 2(c)). We use the TIFF image format to store the images, since it supports multiple images per file and a variety of lossless and lossy compression schemes.
2. We store each image in a separate file.

Associated to each light-field model we also store two other files, a model file (Figure 3) and a direction file (Figure 4). The model file contains the parameters of the model:

- The directional and positional resolutions of the light field.
- The position and orientation of the model.
- The number of images per directional sample (for 4D light fields we only support one image per sample).
- The location of the $P_k$ planes of the representation.
- The image and depth-map filename conventions.

The direction file contains information specific to each direction, like the pencil's ID, resolution of the image data and the position and orientation of the $P_k$ planes (see Figure 4).

Light-field models are notorious for requiring large amounts of storage, and ours is not an exception. For example a light-field model with a 256×256 positional resolution and 20480 directional samples requires roughly 3.7GB without compression. If we compress the images using a lossless algorithm we can get a 10:1 compression rate (assuming that most of the images contain a certain amount of background pixels). With lossy (JPEG) compression we can achieve as much as 40:1 rates depending on the amount of background pixels.

In any case, all this image data is difficult to fit in main memory. So we use out-of-core storage techniques to mantain just a working set of the images in memory. Our system uses a texture least-recently-used (LRU) cache that manages the light-field images so that only new images need to be loaded from disk at any time.

## 4 Rendering Algorithm

The DPP rendering algorithm is an adapted version of the Lumigraph algorithm [8]. Given the viewing parameters, it starts by placing an imaginary sphere centered at

```
SnapshotModel {
Name                phot
Path                ./DATA/phot.ssm
Center              0 0 0
Radius              1
Azimuth             0
Elevation           90
SpinAngle           0

Geode {
    Type        UNIFORM
    Levels      4
    Pencils     5120
    Snapshots   6820
    }
}
SnapshotsPerPencil 1

DMapFormat          None
DMapType            NONE
DMapResolution      0 0
ImageFormat         PNG
ImageColorModel     RGB
ImageResolution     256 256
JPEGQuality         075
DMapCompression     DMC_NONE
```

**Figure 3. Model file example.**

the eye position. The sphere is tessellated exactly like the sphere representing the set of directional samples of the light-field model. The rendering algorithm determines which pencils of directions intersect the viewing frustum. For each of those pencils, it then renders an image on the portion of the frustum intersected by the pencils.

Since pencils of directions form a hierarchy, the rendering algorithm is a breadth-first search. If a parent pencil intersects the frustum, then its children are traversed. For each pencil, intersection is determined as follows.

- Directional vectors are obtained for each of the vertices of a given pencil triangle $T_k$.
- All vertices are tested for intersection with the viewing frustum.
- If any of them intersects the window, then $T_k$ is visible and the search continues with the subtriangles of $T_k$.

We leave the reprojection and display steps of the algorithm to the rendering hardware. We render each visible triangle $T_k$ with a texture map containing a portion of $L_k$. That portion is determined by the geometric relationship between the viewer's position and the light-field model as given by its center and orientation. We determine the correct texture coordinates by casting three rays, one through each of the vertices of $T_k$, starting at the viewer's position. We intersect the rays with the plane $P_k$ of the light-field representation. $P_k$ is given in world coordinates and depends on the light-field model's position and orientation. Once we have the intersection points with $P_k$, we compute their $(u, v)$ coordinates using a transformation based on the geometric parameters of the light-field model. Lastly, we use a texture map transform to properly warp the visible portion of $L_k$ onto $T_k$. The triangle is then texture-mapped and rendered using standard graphics hardware.

This rendering algorithm displays the radiance data associated to each triangle using a constant reconstruction kernel. No interpolation is performed using the images (texture maps) associated to neighboring triangles. Another version of the algorithm performs quadralinear interpolation, where

```
01          256 256 -0.039 0.29 1 -0.591
01c         256 256 -0.039 0.29 1 -0.591
01cc        256 256 -0.039 0.29 1 -0.591
.....
10lc1c      256 256 -0.044 0.088 1 -0.428
10lc1h      256 256 -0.045 0.121 1 -0.442
.....
```

**Figure 4. Direction file example.**

| Directional Samples | Time | Size |
|---|---|---|
| First dataset | | |
| 20480 | $7h31m$ | 178MB |
| 5120 | $1h50m$ | 44MB |
| 1280 | $25m$ | 11MB |
| Second dataset | | |
| 20480 | $1h27m$ | 208MB |
| 5120 | $20m$ | 52MB |
| 1280 | $5m$ | 13MB |

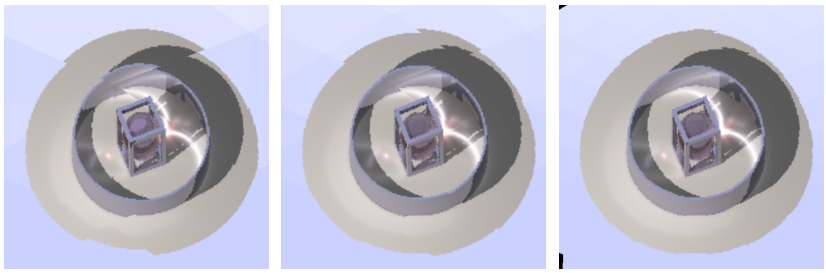
**Table 1. Construction times and storage required by our datasets.**

linear interpolation is applied to each of the four dimensions of the light field [4]. In the spatial domain interpolation is achieved by using hardware assisted mip-mapping. In the directional domain interpolation is done by alpha-blending neighboring triangles.

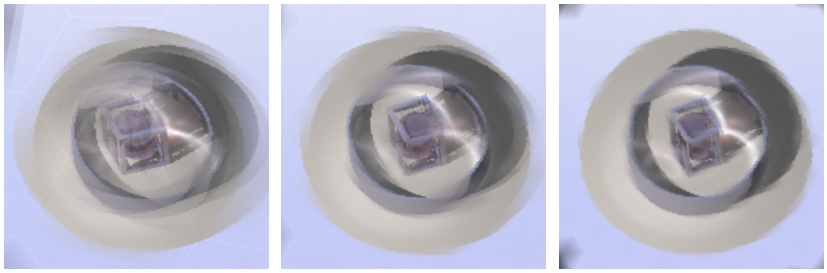
The complexity of the rendering algorithm depends on the number of directional samples that fall inside the viewing frustum. The wider the field of view, the more triangles need to be rendered.

## 5 Results

We tested our system's implementation by building and rendering different light-field models constructed from



(a) 1280 directional samples. (b) 5120 directional samples. (c) 20480 directional samples.



(d) 1280 directional samples. (e) 5120 directional samples. (f) 20480 directional samples.

**Figure 5. These light-field models with different directional resolutions were obtained by ray tracing a geometric model with multiple light sources, reflection, refraction, and caustics. The spatial resolution of each light-field image is 256×256 pixels (rendering resolution is 512×512 pixels). The top row was rendered using constant reconstruction and the bottom row using quadralinear interpolation.**

traditional geometric models. To obtain the light-field models we used the Persistence of Vision Raytracer (http://www.povray.org). The two models shown in this paper were built using a 3GHz Pentium 4 machine. Table 1 summarizes the time and memory requirements of both models. Memory requirements can be reduced by using 4D compression schemes instead of the brute force approach we use.

The light-field models were rendered using an OpenGL-based (http://www.opengl.org) renderer. For image storage and manipulation we used the OpenIL library (http://openil.sourceforge.net). Device-dependent graphics were implemented using the SDL library (http://www.libsdl.org). The images shown in Figures 5 and 6 were rendered using our rendering algorithm. Images took less than 10 msecs to render. Rendering time depends on the light field's directional resolution and the type of reconstruction filter: constant or quadralinear.

Constant reconstruction runs faster at the expense of producing rendering artifacts that look like seams. Quadralinear interpolation requires rendering more triangles and thus runs slower. It usually produces better results by smoothing out the seams. However, it may still produce ghosting like other interpolation methods, especially when the number of directional samples is low.

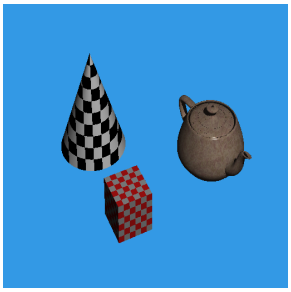
## Conclusions

We have presented a representation for 4D light-field modeling and rendering. Our representation is based on the direction-and-point parameterization. We show how to generate the light field's radiance data and store it in memory and disk. We describe the data structures and files used by the representation. We also explain how our rendering algorithm works.

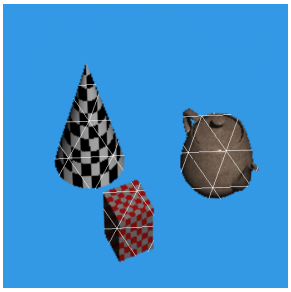
The rendering algorithm allows the display of light fields with different spatial and directional resolutions. Additionally it supports two types of reconstruction kernels, constant and quadralinear. Rendering large light-field models requires using out-of-core techniques to handle memory usage problems.

Our system has the advantage that it uses a parameterization that is uniform and thus guarantees light-field invariance under rotations and translations. This allows the user to move freely around a model without noticing any resolution changes. Our results demonstrate that DPP-based light-field models are competitive with geometry-based models.

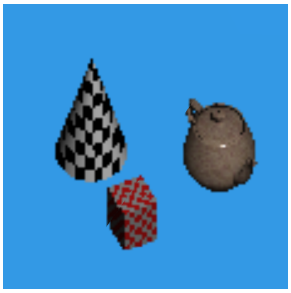
We are currently working on improving our representation and its rendering algorithm. First, we want to augment the light-field data with depth information (the representation already supports it). Using depth information we can render higher quality images using lower directional reso-



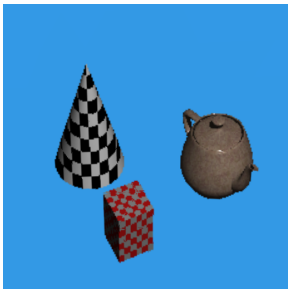
(a) The original geometric model rendered with POV.



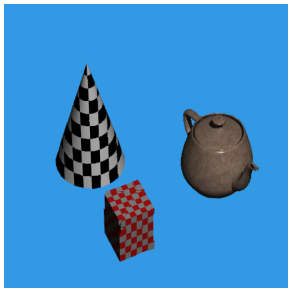
(b) Light-field model rendered showing the texture-mapped triangles that fall inside the viewing frustum.



(c) Light-field model with 128×128 spatial samples.



(d) Light-field model with 256×256 spatial samples.



(e) Light-field model with 512×512 spatial samples.

**Figure 6. This model contains a cone, a box and the Utah teapot rendered with a ray tracer. The light-field versions contain 20480 directional samples.**

lutions. This reduces the amount of storage required by the light field's radiance data.

We are also improving our representation and rendering algorithm to handle multiresolution. Our directional sampling algorithm already supports multiresolution by hierarchically subdividing the sphere. Spatial multiresolution can be implemented using texture mipmaps. Multiresolution can be used to build non-uniform models and to adaptively control a model's rendering frame-rate.

## Acknowledgements

This work was partially supported by grant TIN2005-08863-C03-01 of the Spanish Ministry of Education and Science and STREP project IST-004363 of the 6th Framework Program of the European Union.

## References

[1] S. Allan. 3-Deep. *IEEE Spectrum*, pages 22–27, April 2005.

[2] T. Balogh, T. Forgcs, O. Balet, E. Bouvier, F. Bettio, E. Gobbetti, and G. Zanetti. A scalable holographic display for interactive graphics applications. *IEEE VR 2005 Workshop on Emerging Display Technologies*, March 2005.

[3] S. A. Benton. Survey of holographic stereograms. In *Processing and Display of Three-Dimensional Data*, volume 367, pages 15–19, 1982.

[4] E. Camahort, A. Lerios, and D. Fussell. Uniformly sampled light fields. In *Proc. Eurographics Rendering Workshop '98)*, pages 117–130, 1998.

[5] G. Dutton. Locational properties of quaternary triangular meshes. In *Fourth International Symposium on Spatial Data Handling*, pages 901–910, Zurich, Switzerland, 1990.

[6] G. Fekete. Rendering and managing spherical data with sphere quadtrees. In *Visualization90*, pages 176–186, Los Alamitos, California, 1990. IEEE Computer Society Press.

[7] J. S. Gondek, G. W. Meyer, and J. G. Newman. Wavelength dependent reflectance functions. In *Proc. SIGGRAPH '94*, pages 213–220, 1994.

[8] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proc. SIGGRAPH '96*, pages 43–54, 1996.

[9] K. Haines and D. Haines. Computer graphics for holography. *IEEE Computer Graphics and Applications*, pages 37–46, January 1992.

[10] A. Isaksen, L. McMillan, and S. J. Gortler. Dynamically reparameterized light fields. In *Proc. SIGGRAPH '00*, pages 297–306, 2000.

[11] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. SIGGRAPH '96*, pages 31–42, 1996.

[12] K. Perlin, S. Paxia, and J. S. Kollin. An autostereoscopic display. In *Proc. SIGGRAPH '00*, pages 319–326, 2000.

[13] R. Yang, S. Chen, X. Huang, S. Li, L. Wang, and C. Jaynes. Towards the light field display. *IEEE VR 2005 Workshop on Emerging Display Technologies*, March 2005.