The Lumigraph

Steven J. Gortler
Microsoft Research

Radek Grzeszczuk
University of Toronto[*]

Richard Szeliski
Microsoft Research

Michael F. Cohen
Microsoft Research

Abstract

This paper discusses a new method for capturing the complete appearance of both synthetic and real world objects and scenes, representing this information, and then using this representation to render images of the object from new camera positions. Unlike the shape capture process traditionally used in computer vision and the rendering process traditionally used in computer graphics, our approach does not rely on geometric representations. Instead we sample and reconstruct a 4D function, which we call a Lumigraph. The Lumigraph is a subset of the complete plenoptic function that describes the flow of light at all positions in all directions. With the Lumigraph, new images of the object can be generated very quickly, independent of the geometric or illumination complexity of the scene or object. The paper discusses a complete working system including the capture of samples, the construction of the Lumigraph, and the subsequent rendering of images from this new representation.

1 Introduction

The process of creating a virtual environment or object in computer graphics begins with modeling the geometric and surface attributes of the objects in the environment along with any lights. An image of the environment is subsequently rendered from the vantage point of a virtual camera. Great effort has been expended to develop computer aided design systems that allow the specification of complex geometry and material attributes. Similarly, a great deal of work has been undertaken to produce systems that simulate the propagation of light through virtual environments to create realistic images.

Despite these efforts, it has remained difficult or impossible to recreate much of the complex geometry and subtle lighting effects found in the real world. The modeling problem can potentially be bypassed by capturing the geometry and material properties of objects directly from the real world. This approach typically involves some combination of cameras, structured light, range finders, and mechanical sensing devices such as 3D digitizers. When successful, the results can be fed into a rendering program to create images of real objects and scenes. Unfortunately, these systems are still unable to completely capture small details in geometry and material properties. Existing rendering methods also continue to be limited in their capability to faithfully reproduce real world illumination, even if given accurate geometric models.

Quicktime VR [6] was one of the first systems to suggest that the traditional modeling/rendering process can be skipped. Instead, a series of captured environment maps allow a user to *look around* a scene from fixed points in space. One can also flip through different views of an object to create the illusion of a 3D model. Chen and Williams [7] and Werner et al [30] have investigated smooth interpolation between images by modeling the motion of pixels (i.e., the *optical flow*) as one moves from one camera position to another. In Plenoptic Modeling [19], McMillan and Bishop discuss finding the disparity of each pixel in stereo pairs of cylindrical images. Given the disparity (roughly equivalent to depth information), they can then move pixels to create images from new vantage points. Similar work using stereo pairs of planar images is discussed in [14].

This paper extends the work begun with Quicktime VR and Plenoptic Modeling by further developing the idea of capturing the complete flow of light in a region of the environment. Such a flow is described by a *plenoptic function*[1]. The plenoptic function is a five dimensional quantity describing the flow of light at every 3D spatial position (x, y, z) for every 2D direction (θ, ϕ). In this paper, we discuss computational methods for capturing and representing a plenoptic function, and for using such a representation to render images of the environment from any arbitrary viewpoint.

Unlike Chen and Williams' view interpolation [7] and McMillan and Bishop's plenoptic modeling [19], our approach does not rely explicitly on any optical flow information. Such information is often difficult to obtain in practice, particularly in environments with complex visibility relationships or specular surfaces. We do, however, use approximate geometric information to improve the quality of the reconstruction at lower sampling densities. Previous flow based methods implicitly rely on diffuse surface reflectance, allowing them to use a pixel from a single image to represent the appearance of a single geometric location from a variety of viewpoints. In contrast, our approach regularly samples the full plenoptic function and thus makes no assumptions about reflectance properties.

If we consider only the subset of light leaving a bounded object (or equivalently entering a bounded empty region of space), the fact that radiance along any ray remains constant[1] allows us to reduce the domain of interest of the plenoptic function to four dimensions. This paper first discusses the representation of this 4D function which we call a Lumigraph. We then discuss a system for sampling the plenoptic function with an inexpensive hand-held camera, and "developing" the captured light into a Lumigraph. Finally this paper describes how to use texture mapping hardware to quickly reconstruct images from any viewpoint with a virtual camera model. The Lumigraph representation is applicable to synthetic objects as well, allowing us to encode the complete appearance of a complex model and to rerender the object at speeds independent of the model complexity. We provide results on synthetic and real sequences and discuss work that is currently underway to make the system more efficient.

[*] Work performed while visiting Microsoft Research.

[1] We are assuming the medium (i.e., the air) to be transparent.

2 Representation

2.1 From 5D to 4D

The plenoptic function is a function of 5 variables representing position and direction [2]. If we assume the air to be transparent then the radiance along a ray through empty space remains constant. If we furthermore limit our interest to the light leaving the convex hull of a bounded object, then we only need to represent the value of the plenoptic function along some surface that surrounds the object. A cube was chosen for its computational simplicity (see Figure 1). At any point in space, one can determine the radiance along any ray in any direction, by tracing backwards along that ray through empty space to the surface of the cube. Thus, the plenoptic function due to the object can be reduced to 4 dimensions [3].

The idea of restricting the plenoptic function to some surrounding surface has been used before. In full-parallax holographic stereograms [3], the appearance of an object is captured by moving a camera along some surface (usually a plane) capturing a 2D array of photographs. This array is then transferred to a single holographic image, which can display the appearance of the 3D object. The work reported in this paper takes many of its concepts from holographic stereograms.

Global illumination researchers have used the "surface restricted plenoptic function" to efficiently simulate light-transfer between regions of an environment containing complicated geometric objects. The plenoptic function is represented on the surface of a cube surrounding some region; that information is all that is needed to simulate the light transfer from that region of space to all other regions [17]. In the context of illumination engineering, this idea has been used to model and represent the illumination due to physical luminaires. Ashdown [2] describes a gantry for moving a camera along a sphere surrounding a luminaire of interest. The captured information can then be used to represent the light source in global illumination simulations. Ashdown traces this idea of the surface-restricted plenoptic function back to Levin [15].

A limited version of the work reported here has been described by Katayama et al. [11]. In their system, a camera is moved along a track, capturing a 1D array of images of some object. This information is then used to generate new images of the object from other points in space. Because they only capture the plenoptic function along a line, they only obtain horizontal parallax, and distortion is introduced as soon as the new virtual camera leaves the line. Finally, in work concurrent to our own, Levoy and Hanrahan [16] represent a 4D function that allows for undistorted, full parallax views of the object from anywhere in space.

2.2 Parameterization of the 4D Lumigraph

There are many potential ways to parameterize the four dimensions of the Lumigraph. We adopt a parameterization similar to that used in digital holographic stereograms [9] and also used by Levoy and Hanrahan [16]. We begin with a cube to organize a Lumigraph and, without loss of generality, only consider for discussion a single square face of the cube (the full Lumigraph is constructed from six such faces).

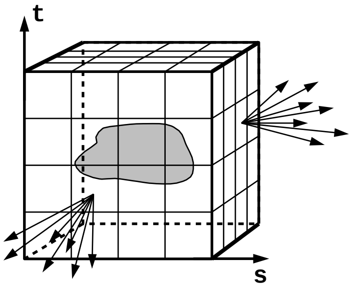

Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

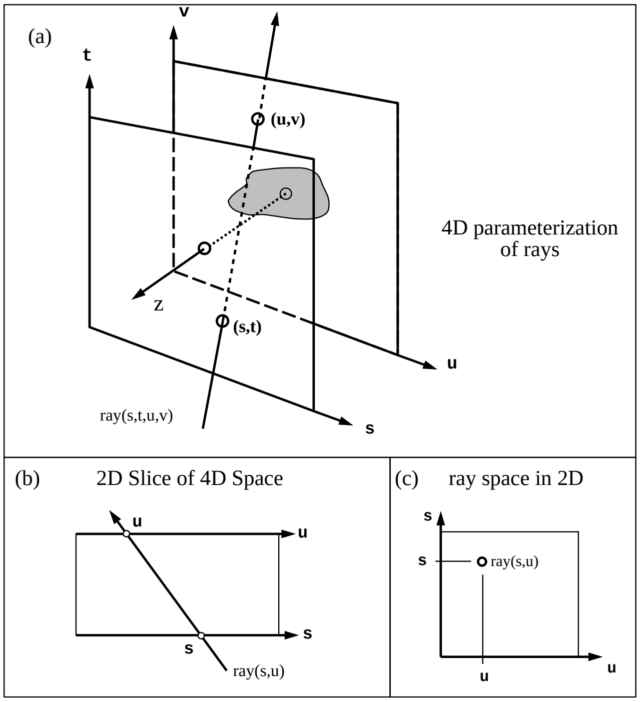

Figure 2: Parameterization of the Lumigraph

We choose a simple parameterization of the cube face with orthogonal axes running parallel to the sides labeled s and t (see Figure 1). Direction is parameterized using a second plane parallel to the st plane with axes labeled u and v (Figure 2). Any point in the 4D Lumigraph is thus identified by its four coordinates (s, t, u, v), the coordinates of a ray piercing the first plane at (s, t) and intersecting the second plane at (u, v) (see Ray(s, t, u, v) in Figure 2). We place the origin at the center of the uv plane, with the z axis normal to the plane. The st plane is located at $z = 1$. The full Lumigraph consists of six such pairs of planes with normals along the x, $-x$, y, $-y$, z, and $-z$ directions.

It will be instructive at times to consider two 2D analogs to the 4D Lumigraph. Figure 2(b) shows a 2D slice of the 4D Lumigraph that indicates the u and s axes. Figure 2(c) shows the same arrangement in *2D ray coordinates* in which rays are mapped to points (e.g., ray(s, u)) and points are mapped to lines.[4]

Figure 3 shows the relationship between this parameterization of the Lumigraph and a pixel in some arbitrary image. Given a Lu-

[2] We only consider a snapshot of the function, thus time is eliminated. Without loss of generality, we also consider only a monochromatic function (in practice 3 discrete color channels), eliminating the need to consider wavelength. We furthermore ignore issues of dynamic range and thus limit ourselves to scalar values lying in some finite range.

[3] In an analogous fashion one can reconstruct the complete plenoptic function inside an empty convex region by representing it only on the surface bounding the empty region. At any point inside the region, one can find the light entering from any direction by finding that direction's intersection with the region boundary.

[4] More precisely, a line in ray space represents the set of rays through a point in space.

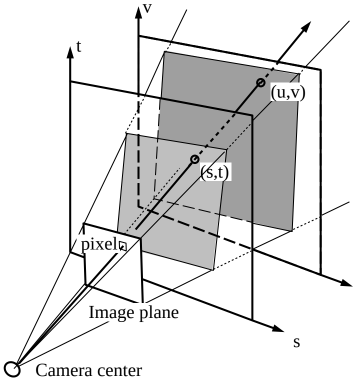

Figure 3: Relationship between Lumigraph and a pixel in an arbitrary image

migraph, L, one can generate an arbitrary new image coloring each pixel with the appropriate value $L(s, t, u, v)$. Conversely given some arbitrary image and the position and orientation of the camera, each pixel can be considered a sample of the Lumigraph value at (s, t, u, v) to be used to construct the Lumigraph.

There are many advantages of the two parallel plane parameterization. Given the geometric description of a ray, it is computationally simple to compute its coordinates; one merely finds its intersection with two planes. Moreover, reconstruction from this parameterization can be done rapidly using the texture mapping operations built into hardware on modern workstations (see section 3.6.2). Finally, in this parameterization, as one moves an eyepoint along the st plane in a straight line, the projection on the uv plane of points on the geometric object track along parallel straight lines. This makes it computationally efficient to compute the apparent motion of a geometric point (i.e., the *optical flow*), and to apply depth correction to the Lumigraph.

2.3 Discretization of the 4D Parameterization

So far, the Lumigraph has been discussed as an unknown, continuous, four dimensional function within a hypercubical domain in s, t, u, v and scalar range. To map such an object into a computational framework requires a discrete representation. In other words, we must choose some finite dimensional function space within which the function resides. To do so, we choose a discrete subdivision in each of the (s, t, u, v) dimensions and associate a coefficient and a basis function (reconstruction kernel) with each 4D grid point.

Choosing M subdivisions in the s and t dimensions and N subdivisions in u and v results in a grid of points on the st and uv planes (Figure 4). An st grid point is indexed with (i, j) and is located at (s_i, t_j). A uv grid point is indexed with (p, q) and is located at (u_p, v_q). A 4D grid point is indexed (i, j, p, q). The data value (in fact an RGB triple) at this grid point is referred to as $x_{i,j,p,q}$

2.3.1 Choice of Basis

We associate with each grid point a basis function $B_{i,j,p,q}$ so that the continuous Lumigraph is reconstructed as the linear sum

$$\tilde{L}(s,t,u,v) = \sum_{i=0}^{M}\sum_{j=0}^{M}\sum_{p=0}^{N}\sum_{q=0}^{N} x_{i,j,p,q}B_{i,j,p,q}(s,t,u,v)$$

where $\tilde{L}$ is a finite dimensional Lumigraph that exists in the space defined by the choice of basis.

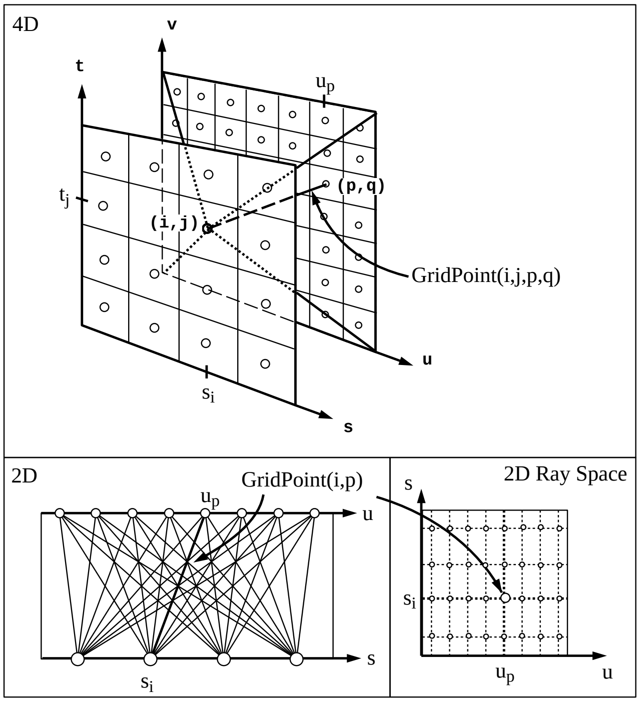

Figure 4: Discretization of the Lumigraph

For example, if we select constant basis functions (i.e., a 4D *box* with value 1 in the 4D region closest to the associated grid point and zero elsewhere), then the Lumigraph is piecewise constant, and takes on the value of the coefficient of the nearest grid point.

Similarly, a quadralinear basis function has a value of 1 at the grid point and drops off to 0 at all neighboring grid points. The value of $\tilde{L}(s, t, u, v)$ is thus interpolated from the 16 grid points forming the hypercube in which the point resides.

We have chosen to use the quadralinear basis for its computational simplicity and the C^0 continuity it imposes on $\tilde{L}$. However, because this basis is not band limited by the Nyquist frequency, and thus the corresponding finite dimensional function space is not shift invariant [24], the grid structure will be slightly noticeable in our results.

2.3.2 Projection into the Chosen Basis

Given a continuous Lumigraph, L, and a choice of basis for the finite dimensional Lumigraph, $\tilde{L}$, we still need to define a *projection* of L into $\tilde{L}$ (i.e., we need to find the coefficients x that result in an $\tilde{L}$ which is by some metric *closest* to L). If we choose the L^2 distance metric, then the projection is defined by integrating L against the *duals* of the basis functions [8], given by the inner products,

$$x_{i,j,p,q} = < L, \tilde{B}_{i,j,p,q} > \qquad (1)$$

In the case of the box basis, $B = \tilde{B}$. The duals of the quadralinear basis functions are more complex, but these basis functions sufficiently approximate their own duals for our purposes.

One can interpret this projection as point sampling L after it has been low pass filtered with the kernel $\tilde{B}$. This interpretation is pursued in the context of holographic stereograms by Halle [9]. One can also interpret this projection as the result of placing a physical or synthetic "skewed" camera at grid point (s_i, t_j) with an aperture corresponding to the bilinear basis and with a pixel centered at

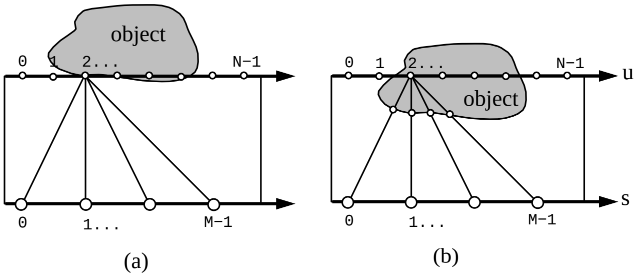

Figure 5: Choice of resolution on the uv plane

(u_p, v_q) antialiased with a bilinear filter. This analogy is pursued in [16].

In Figure 16 we show images generated from Lumigraphs. The geometric scene consisted of a partial cube with the pink face in front, yellow face in back, and the brown face on the floor. These Lumigraphs were generated using two different quadrature methods to approximate equation 1, and using two different sets of basis functions, constant and quadralinear. In (a) and (c) only one sample was used to compute each Lumigraph coefficient. In these examples severe ghosting artifacts can be seen. In (b) and (d) numerical integration over the support of $\tilde{B}$ in st was computed for each coefficient. It is clear that best results are obtained using quadralinear basis function, with a full quadrature method.

2.3.3 Resolution

An important decision is how to set the resolutions, M and N, that best balance efficiency and the quality of the images reconstructed from the Lumigraph. The choices for M and N are influenced by the fact that we expect the visible surfaces of the object to lie closer to the uv plane than the st plane. In this case, N, the resolution of the uv plane, is closely related to the final image resolution and thus a choice for N close to final image resolution works best (we consider a range of resolutions from 128 to 512).

One can gain some intuition for the choice of M by observing the 2D subset of the Lumigraph from a single grid point on the uv plane (see $u = 2$ in Figure 5(a)). If the surface of the object lies exactly on the uv plane at a gridpoint, then all rays leaving that point represent samples of the radiance function at a single position on the object's surface. Even when the object's surface deviates from the uv plane as in Figure 5(b), we can still expect the function across the st plane to remain smooth and thus a low resolution is sufficient. Thus a significantly lower resolution for M than N can be expected to yield good results. In our implementation we use values of M ranging from 16 to 64.

2.3.4 Use of Geometric Information

Assuming the radiance function of the object is well behaved, knowledge about the geometry of the object gives us information about the coherence of the associated Lumigraph function, and can be used to help define the shape of our basis functions.

Consider the ray (s, u) in a two-dimensional Lumigraph (Figure 6). The closest grid point to this ray is (s_{i+1}, u_p). However, gridpoints (s_{i+1}, u_{p-1}) and (s_i, u_{p+1}) are likely to contain values closer to the true value at (s, u) since these grid points represent rays that intersect the object nearby the intersection with (s, u). This suggests adapting the shape of the basis functions.

Suppose we know the depth value z at which ray (s, u) first intersects a surface of the object. Then for a given s_i, one can compute a corresponding u' for a ray (s_i, u') that intersects the same geometric location on the object as the original ray (s, u)[5]. Let the depth z be 0 at the uv plane and 1 at the st plane. The intersections can then be found by examining the similar triangles in Figure 6,

$$u' = u + (s - s_i)\frac{z}{1-z} \tag{2}$$

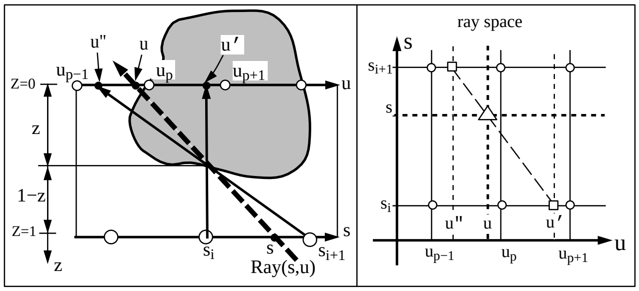

Figure 6: Depth correction of rays

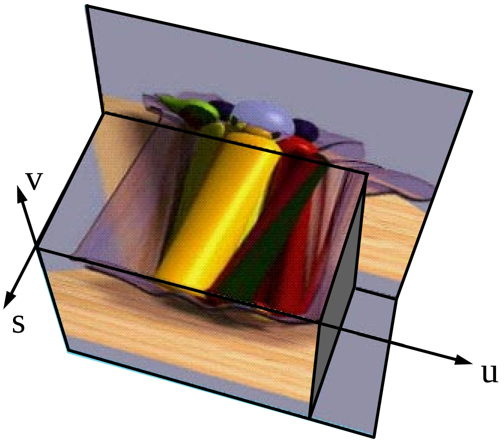

Figure 7: An (s, u, v) slice of a Lumigraph

It is instructive to view the same situation as in Figure 6(a), plotted in *ray space* (Figure 6(b)). In this figure, the triangle is the ray (s, u), and the circles indicate the nearby gridpoints in the discrete Lumigraph. The diagonal line passing through (s, u) indicates the *optical flow* (in this case, horizontal motion in 2D) of the intersection point on the object as one moves back and forth in s. The intersection of this line with s_i and s_{i+1} occurs at u' and u'' respectively.

Figure 7 shows an (s, u) slice through a three-dimensional (s, u, v) subspace of the Lumigraph for the ray-traced fruitbowl used in Figure 19. The flow of pixel motion is along straight lines in this space, but more than one motion may be present if the scene includes transparency. The slope of the flow lines corresponds to the depth of the point on the object tracing out the line. Notice how the function is coherent along these flow lines [4].

We expect the Lumigraph to be smooth along the optical flow lines, and thus it would be beneficial to have the basis functions adapt their shape correspondingly. The remapping of u and v values to u' and v' performs this reshaping. The idea of shaping the support of basis functions to closely match the structure of the function being approximated is used extensively in finite element methods. For example, in the Radiosity method for image synthesis, the mesh of elements is adapted to fit knowledge about the illumination function.

[5] Assuming there has been no change in visibility.

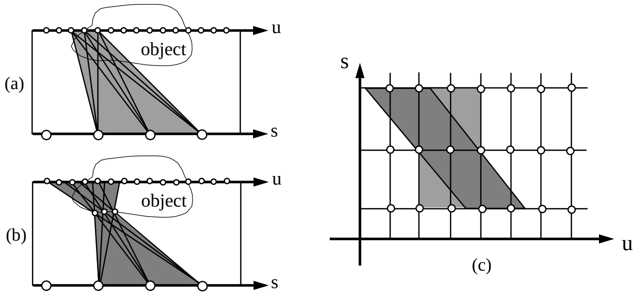

Figure 8: (a) Support of an uncorrected basis function. (b) Support of a depth corrected basis function. (c) Support of both basis functions in ray space.

The new basis function $B'_{i,j,p,q}(s,t,u,v)$ is defined by first finding u' and v' using equation 2 and then evaluating B, that is

$$B'_{i,j,p,q}(s,t,u,v) = B_{i,j,p,q}(s,t,u',v')$$

Although the shape of the new *depth corrected* basis is complicated, $\tilde{L}(s,t,u,v)$ is still a linear sum of coefficients and the weights of the contributing basis functions still sum to unity. However, the basis is no longer representable as a tensor product of simple boxes or hats as before. Figure 8 shows the support of an uncorrected (light gray) and a depth corrected (dark gray) basis function in 2D geometric space and in 2D ray space. Notice how the support of the depth corrected basis intersects the surface of the object across a narrower area compared to the uncorrected basis.

We use depth corrected quadralinear basis functions in our system. The value of $\tilde{L}(s,t,u,v)$ in the corrected quadralinear basis is computed using the following calculation:

```
QuadralinearDepthCorrect(s,t,u,v,z)
    $\mathrm{Result} = 0$
    $h_{st} = s_1 - s_0$ /* grid spacing */
    $h_{uv} = u_1 - u_0$
    for each of the four $(s_i, t_j)$ surrounding $(s, t)$
        $u' = u + (s - s_i) * z/(1 - z)$
        $v' = v + (t - t_j) * z/(1 - z)$
        $\mathrm{temp} = 0$
        for each of the four $(u_p, v_q)$ surrounding $(u', v')$
            $\mathrm{iterpWeight}_{uv} =$
                $(h_{uv} - \mid u_p - u' \mid) * (h_{uv} - \mid v_q - v' \mid)/h_{uv}^2$
            $\mathrm{temp} \mathrel{+}= \mathrm{interpWeight}_{uv} * L(s_i, t_j, u_p, v_q)$
        $\mathrm{interpWeight}_{st} =$
            $(h_{st} - \mid s_i - s \mid) * (h_{st} - \mid t_j - t \mid)/h_{st}^2$
        $\mathrm{Result} \mathrel{+}= \mathrm{interpWeight}_{st} * \mathrm{temp}$
    return Result
```

Figure 17 shows images generated from a Lumigraph using uncorrected and depth corrected basis functions. The depth correction was done using a 162 polygon model to approximate the original 70,000 polygons. The approximation was generated using a mesh simplification program [10]. These images show how depth correction reduces the artifacts present in the images.

3 The Lumigraph System

This section discusses many of the practical implementation issues related to creating a Lumigraph and generating images from it. Figure 9 shows a block diagram of the system. The process begins with capturing images with a hand-held camera. From known markers in the image, the camera's position and orientation (its *pose*) is estimated. This provides enough information to create an approximate geometric object for use in the depth correction of (u, v) values. More importantly, each pixel in each image acts as a sample of the plenoptic function and is used to estimate the coefficients of the discrete Lumigraph (i.e., to *develop* the Lumigraph). Alternatively, the Lumigraph of a synthetic object can be generated directly by integrating a set of rays cast in a rendering system. We only briefly touch on compression issues. Finally, given an arbitrary virtual camera, new images of the object are quickly rendered.

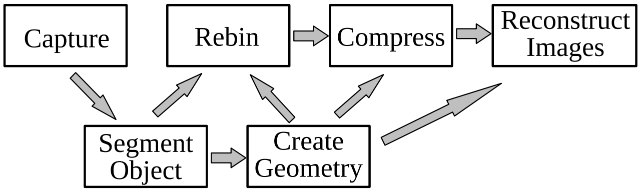

Figure 9: The Lumigraph system

3.1 Capture for Synthetic Scenes

Creating a Lumigraph of a synthetic scene is straightforward. A single sample per Lumigraph coefficient can be captured for each gridpoint (i, j) by placing the center of a virtual pin hole camera at (s_i, t_j) looking down the z axis, and defining the imaging frustum using the uv square as the film location. Rendering an image using this skewed perspective camera produces the Lumigraph coefficients. The pixel values in this image, indexed (p, q), are used as the Lumingraph coefficients $x_{i,j,p,q}$. To perform the integration against the kernel $\tilde{B}$, multiple rays per coefficient can be averaged by jittering the camera and pixel locations, weighting each image using $\tilde{B}$. For ray traced renderings, we have used the ray tracing program provided with the Generative Modeling package[25].

3.2 Capture for Real Scenes

Computing the Lumigraph for a real object requires the acquisition of object images from a large number of viewpoints. One way in which this can be accomplished is to use a special motion control platform to place the real camera at positions and orientations coincident with the (s_i, t_j) gridpoints [16]. While this is a reasonable solution, we are interested in acquiring the images with a regular hand-held camera. This results in a simpler and cheaper system, and may extend the range of applicability to larger scenes and objects.

To achieve this goal, we must first calibrate the camera to determine the mapping between directions and image coordinates. Next, we must identify special calibration markers in each image and compute the camera's pose from these markers. To enable depth-corrected interpolation of the Lumigraph, we also wish to recover a rough geometric model of the object. To do this, we convert each input image into a silhouette using a blue-screen technique, and then build a volumetric model from these binary images.

3.2.1 Camera Calibration and Pose Estimation

Camera calibration and pose estimation can be thought of as two parts of a single process: determining a mapping between screen pixels and rays in the world. The parameters associated with this process naturally divide into two sets: extrinsic parameters, which define the camera's pose (a rigid rotation and translation), and intrinsic parameters, which define a mapping of 3D camera coordinates onto the screen. This latter mapping not only includes a perspective (pinhole) projection from the 3D coordinates to undistorted

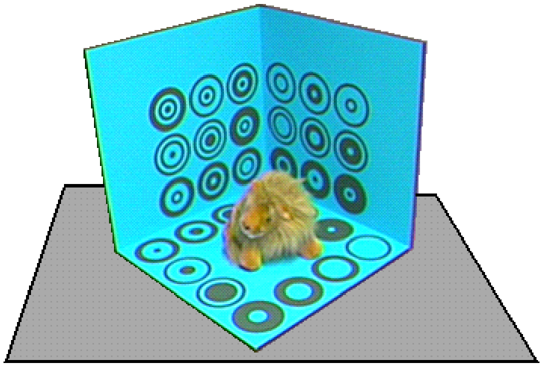

Figure 10: The capture stage

image coordinates, but also a radial distortion transformation and a final translation and scaling into screen coordinates [29, 31].

We use a camera with a fixed lens, thus the intrinsic parameters remain constant throughout the process and need to be estimated only once, before the data acquisition begins. Extrinsic parameters, however, change constantly and need to be recomputed for each new video frame. Fortunately, given the intrinsic parameters, this can be done efficiently and accurately with many fewer calibration points. To compute the intrinsic and extrinsic parameters, we employ an algorithm originally developed by Tsai [29] and extended by Willson [31].

A specially designed stage provides the source of calibration data (see Figure 10). The stage has two walls fixed together at a right angle and a base that can be detached from the walls and rotated in 90 degree increments. An object placed on such a movable base can be viewed from all directions in the upper hemisphere. The stage background is painted cyan for later blue-screen processing. Thirty markers, each of which consists of several concentric rings in a darker shade of cyan, are distributed along the sides and base. This number is sufficiently high to allow for a very precise intrinsic camera calibration. During the extrinsic camera calibration, only 8 or more markers need be visible to reliably compute a pose.

Locating markers in each image is accomplished by first converting the image into a binary (i.e., black or white) image. A *double thresholding* operator divides all image pixels into three groups separated by intensity thresholds T_1 and T_2. Pixels with an intensity below T_1 are considered black, pixels with an intensity above T_2 are considered white. Pixels with an intensity between T_1 and T_2 are considered black only if they have a black neighbor, otherwise they are considered white. The binary thresholded image is then searched for *connected components* [23]. Sets of connected components with similar centers of gravity are the likely candidates for the markers. Finally, the ratio of radii in each marker is used to uniquely identify the marker. To help the user correctly sample the viewing space, a real-time visual feedback displays the current and past locations of the camera in the view space (Figure 11). Marker tracking, pose estimation, feedback display, and frame recording takes approximately 1/2 second per frame on an SGI Indy.

3.3 3D Shape Approximation

The recovery of 3D shape information from natural imagery has long been a focus of computer vision research. Many of these techniques assume a particularly simple shape model, for example, a polyhedral scene where all edges are visible. Other techniques, such as stereo matching, produce sparse or incomplete depth estimates. To produce complete, closed 3D models, several approaches have been tried. One family of techniques builds 3D volumetric models directly from silhouettes of the object being viewed [21]. Another approach is to fit a deformable 3D model to sparse stereo data. Despite over 20 years of research, the reliable extraction of accurate 3D geometric information from imagery (without the use of active illumination and positioning hardware) remains elusive.

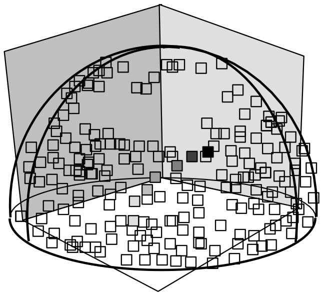

Figure 11: The user interface for the image capture stage displays the current and previous camera positions on a viewing sphere. The goal of the user is to “paint” the sphere.

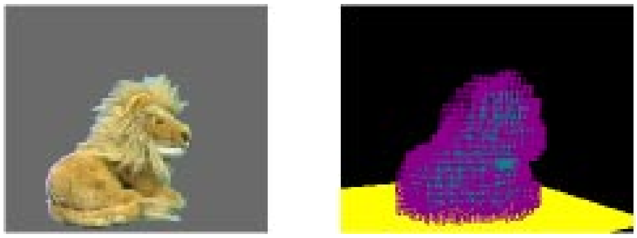

Figure 12: Segmented image plus volume construction

Fortunately, a rough estimate of the shape of the object is enough to greatly aid in the capture and reconstruction of images from a Lumigraph. We employ the octree construction algorithm described in [26] for this process. Each input image is first segmented into a binary object/background image using a blue-screen technique [12] (Figure 12). An octree representation of a cube that completely encloses the object is initialized. Then for each segmented image, each voxel at a coarse level of the octree is projected onto the image plane and tested against the silhouette of the object. If a voxel falls outside of the silhouette, it is removed from the tree. If it falls on the boundary, it is marked for subdivision into eight smaller cubes. After a small number of images are processed, all marked cubes subdivide. The algorithm proceeds for a preset number of subdivisions, typically 4. The resulting 3D model consists of a collection of voxels describing a volume which is known to contain the object[6] (Figure 12). The external polygons are collected and the resulting polyhedron is then smoothed using Taubin’s polyhedral smoothing algorithm [27].

3.4 Rebinning

As described in Equation 1, the coefficient associated with the basis function $B_{i,j,p,q}$ is defined as the integral of the continuous Lumigraph function multiplied by some kernel function $\tilde{B}$. This can be written as

$$x_{i,j,p,q} = \int L(s,t,u,v)\, \tilde{B}_{i,j,p,q}(s,t,u,v)\, ds\, dt\, du\, dv \qquad (3)$$

In practice this integral must be evaluated using a finite number of samples of the function L. Each pixel in the input video stream coming from the hand-held camera represents a single sample

[6] Technically, the volume is a superset of the *visual hull* of the object [13].

$L(s_k, t_k, u_k, v_k)$, of the Lumigraph function. As a result, the sample points in the domain cannot be pre-specified or controlled. In addition, there is no guarantee that the incoming samples are evenly spaced.

Constructing a Lumigraph from these samples is similar to the problem of multidimensional scattered data approximation. In the Lumigraph setting, the problem is difficult for many reasons. Because the samples are not evenly spaced, one cannot apply standard Fourier-based sampling theory. Because the number of sample points may be large ($\approx 10^8$) and because we are working in a 4 dimensional space, it is too expensive to solve systems of equations (as is done when solving thin-plate problems [28, 18]) or to build spatial data structures (such as Delauny triangulations).

In addition to the number of sample points, the distribution of the data samples have two qualities that make the problem particularly difficult. First, the sampling density can be quite sparse, with large gaps in many regions. Second, the sampling density is typically very non-uniform.

The first of these problems has been addressed in a two dimensional scattered data approximation algorithm described by Burt [5]. In his algorithm, a hierarchical set of lower resolution data sets is created using an image pyramid. Each of these lower resolutions represents a "blurred" version of the input data; at lower resolutions, the gaps in the data become smaller. This low resolution data is then used to fill in the gaps at higher resolutions.

The second of these problems, the non-uniformity of the sampling density, has been addressed by Mitchell [20]. He solves the problem of obtaining the value of a pixel that has been super-sampled with a non-uniform density. In this problem, when averaging the sample values, one does not want the result to be overly influenced by the regions sampled most densely. His algorithm avoids this by computing average values in a number of smaller regions. The final value of the pixel is then computed by averaging together the values of these strata. This average is not weighted by the number of samples falling in each of the strata. Thus, the non-uniformity of the samples does not bias the answer.

For our problem, we have developed a new hierarchical algorithm that combines concepts from both of these algorithms. Like Burt, our method uses a pyramid algorithm to fill in gaps, and like Mitchell, we ensure that the non-uniformity of the data does not bias the "blurring" step.

For ease of notation, the algorithm is described in 1D, and will use only one index i. A hierarchical set of basis functions is used, with the highest resolution labeled 0 and with lower resolutions having higher indices. Associated with each coefficient x_i^r at resolution r is a weight w_i^r. These weights determine how the coefficients at different resolution levels are eventually combined. The use of these weights is the distinguishing feature of our algorithm.

The algorithm proceeds in three phases. In the first phase, called *splat*, the sample data is used to approximate the integral of Equation 3, obtaining coefficients x_i^0 and weights w_i^0. In regions where there is little or no nearby sample data, the weights are small or zero. In the second phase, called *pull*, coefficients are computed for basis functions at a hierarchical set of lower resolution grids by combining the coefficient values from the higher resolution grids. In the lower resolution grids, the gaps (regions where the weights are low) become smaller (see figure 13). In the third phase, called *push*, information from the each lower resolution grid is combined with the next higher resolution grid, filling in the gaps while not unduly blurring the higher resolution information already computed.

3.4.1 Splatting

In the splatting phase, coefficients are computed by performing Monte-Carlo integration using the following weighted average estimator:

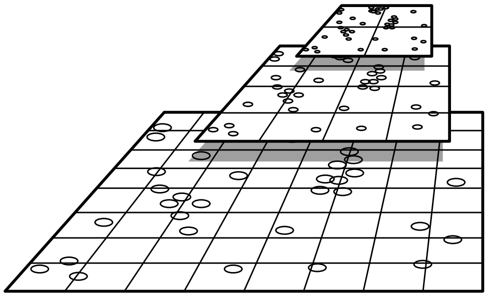

Figure 13: 2D pull-push. At lower resolutions the gaps are smaller.

$$\begin{array}{rcl} w_i^0 & = & \sum_k \tilde{B}_i(s_k) \\ x_i^0 & = & \frac{1}{w_i^0} \sum_k \tilde{B}_i(s_k)\, L(s_k) \end{array} \qquad (4)$$

where s_k denotes the domain location of sample k. If w_i^0 is 0, then the x_i^0 is undefined. If the $\tilde{B}_i$ have compact support, then each sample influences only a constant number of coefficients. Therefore, this step runs in time linear in the number of samples.

If the sample points s_k are chosen from a uniform distribution, this estimator converges to the correct value of the integral in Equation (3), and for n sample points has a variance of approximately $\frac{1}{n}\int \left(\tilde{B}_i(s)\, L(s) - x_i\, \tilde{B}_i(s)\right)^2 ds$. This variance is similar to that obtained using importance sampling, which is often much smaller than the crude Monte Carlo estimator. For a full analysis of this estimator, see [22].

3.4.2 Pull

In the *pull* phase, lower resolution approximations of the function are derived using a set of wider kernels. These wider kernels are defined by linearly summing together the higher resolution kernels ($\tilde{B}_i^{r+1} = \sum_k \tilde{h}_{k-2i} \tilde{B}_k^r$) using some discrete sequence $\tilde{h}$. For linear "hat" functions, $\tilde{h}[-1..1]$ is $\{\frac{1}{2}, 1, \frac{1}{2}\}$

The lower resolution coefficients are computed by combining the higher resolution coefficients using $\tilde{h}$. One way to do this would be to compute

$$\begin{array}{rcl} w_i^{r+1} & = & \sum_k \tilde{h}_{k-2i}\, w_k^r \\ x_i^{r+1} & = & \frac{1}{w_i^{r+1}} \sum_k \tilde{h}_{k-2i}\, w_k^r\, x_k^r \end{array} \qquad (5)$$

It is easy to see that this formula, which corresponds to the method used by Burt, computes the same result as would the original estimator (Equation (4)) applied to the wider kernels. Once again, this estimator works if the sampling density is uniform. Unfortunately, when looking on a gross scale, it is imprudent to assume that the data is sampled uniformly. For example, the user may have held the camera in some particular region for a long time. This non-uniformity can greatly bias the estimator.

Our solution to this problem is to apply Mitchell's reasoning to this context, replacing Equation (5) with:

$$\begin{array}{rcl} w_i^{r+1} & = & \sum_k \tilde{h}_{k-2i}\, \min(w_k^r, 1) \\ x_i^{r+1} & = & \frac{1}{w_i^{r+1}} \sum_k \tilde{h}_{k-2i}\, \min(w_k^r, 1)\, x_k^r \end{array}$$

The value 1 represents full saturation[7], and the $\min$ operator is used to place an upper bound on the degree that one coefficient in a highly

[7] Using the value 1 introduces no loss of generality if the normalization of $\tilde{h}$ is not fixed.

sampled region, can influence the total sum [8].

The pull stage runs in time linear in the number of basis function summed over all of the resolutions. Because each lower resolution has half the density of basis functions, this stage runs in time linear in the number of basis functions at resolution 0.

3.4.3 Push

During the push stage, the lower resolution approximation is used to fill in the regions in the higher resolution that have low weight [9]. If a higher resolution coefficient has a high associated confidence (i.e., has weight greater than one), we fully disregard the lower resolution information there. If the higher resolution coefficient does not have sufficient weight, we blend in the information from the lower resolution.

To blend this information, the low resolution approximation of the function must be expressed in the higher resolution basis. This is done by upsampling and convolving with a sequence$h$, that satisfies $B_i^{r+1} = \sum_k h_{k-2i} B_k^r$.

We first compute temporary values

$$\begin{array}{lcl} tw_i^r &=& \sum_k h_{i-2k}\ \min(w_k^{r+1},1) \\ tx_i^r &=& \frac{1}{tw_i^r}\ \sum_k h_{i-2k}\ \min(w_k^{r+1},1)\, x_k^{r+1} \end{array}$$

These temporary values are now ready to be blended with the values x and w values already at level r.

$$\begin{array}{lcl} x_i^r &=& tx_i^r\,(1-w_i^r) + w_i^r\, x_i^r \\ w_i^r &=& tw_i^r\,(1-w_i^r) + w_i^r \end{array}$$

This is analogous to the blending performed in image compositing.

3.4.4 Use of Geometric Information

This three phase algorithm must be adapted slightly when using the depth corrected basis functions B'. During the splat phase, each sample ray $L(s_k, t_k, u_k, v_k)$ must have its u and v values remapped as explained in Section 2.3.4. Also, during the push and pull phases, instead of simply combining coefficients using basis functions with neighboring indices, depth corrected indices are used.

3.4.5 2D Results

The validity of the algorithm was tested by first applying it to a 2D image. Figure 18 (a) shows a set of scattered samples from the well known mandrill image. The samples were chosen by picking 256 random line segments and sampling the mandrill very densely along these lines [10]. Image (b) shows the resulting image after the *pull/push* algorithm has been applied. Image (c) and (d) show the same process but with only 100 sample lines. The success of our algorithm on both 2D image functions and 4D Lumigraph functions leads us to believe that it may have many other uses.

3.5 Compression

A straightforward sampling of the Lumigraph requires a large amount of storage. For the examples shown in section 4, we use, for a single face, a 32×32 sampling in (s,t) space and 256×256 (u,v) images. To store the six faces of our viewing cube with 24-bits per pixel requires $32^2 \cdot 256^2 \cdot 6 \cdot 3 = 1.125$GB of storage.

Fortunately, there is a large amount of coherence between (s,t,u,v) samples. One could apply a transform code to the 4D array, such as a wavelet transform or block DCT. Given geometric information, we can expect to do even better by considering the 4D array as a 2D array of images. We can then *predict* new (u,v) images from adjacent images, (i.e., images at adjacent (s,t) locations). Intraframe compression issues are identical to compressing single images (a simple JPEG compression yields about a 20:1 savings). Interframe compression can take advantage of increased information over other compression methods such as MPEG. Since we know that the object is static and know the camera motion between adjacent images, we can predict the motion of pixels. In addition, we can leverage the fact that we have a 2D array of images rather than a single linear video stream.

Although we have not completed a full analysis of compression issues, our preliminary experiments suggest that a 200:1 compression ratio should be achievable with almost no degradation. This reduces the storage requirements to under 6MB. Obviously, further improvements can be expected using a more sophisticated prediction and encoding scheme.

3.6 Reconstruction of Images

Given a desired camera (position, orientation, resolution), the reconstruction phase colors each pixel of the output image with the color that this camera would create if it were pointed at the real object.

3.6.1 Ray Tracing

Given a Lumigraph, one may generate a new image from an arbitrary camera pixel by pixel, ray by ray. For each ray, the corresponding (s,t,u,v) coordinates are computed, the nearby grid points are located, and their values are properly interpolated using the chosen basis functions (see Figure 3).

In order to use the depth corrected basis functions given an approximate object, we transform the (u,v) coordinates to the depth corrected (u',v') before interpolation. This depth correction of the (u,v) values can be carried out with the aid of graphics hardware. The polygonal approximation of the object is drawn from the point of view and with the same resolution as the desired image. Each vertex is assigned a *red, green, blue* value corresponding to its (x,y,z) coordinate resulting in a "depth" image. The corrected depth value is found by examining the blue value in the corresponding pixel of the depth image for the $\pm z$-faces of the Lumigraph cube (or the red or green values for other faces). This information is used to find u' and v' with Equation 2.

3.6.2 Texture mapping

The expense of tracing a ray for each pixel can be avoided by reconstructing images using texture mapping operations. The st plane itself is tiled with texture mapped polygons with the textures defined by slices of the Lumigraph: $\mathrm{tex}_{i,j}(u_p, v_q) = x_{i,j,p,q}$. In other words, we have one texture associated with each st gridpoint.

Constant Basis

Consider the case of constant basis functions. Suppose we wish to render an image from the desired camera shown in Figure 14. The set of rays passing through the shaded square on the st plane have (s,t) coordinates closest to the grid point (i,j). Suppose that the uv plane is filled with $\mathrm{tex}_{i,j}$. Then, when using constant basis functions, the shaded region in the desired camera's film plane should be filled with the corresponding pixels in the shaded region of the uv plane. This computation can be accomplished by placing a virtual camera at the desired location, drawing a square polygon on the

[8] This is actually less extreme that Mitchell's original algorithm. In this context, his algorithm would set all non-zero weights to 1.

[9] Variance measures could be used instead of weight as a measure of confidence in this phase.

[10] We chose this type of sampling pattern because it mimics in many ways the structure of the Lumigraph samples taken from a hand-held camera. In that case each input video image is a dense sampling of the 4D Lumigraph along a 2D plane.

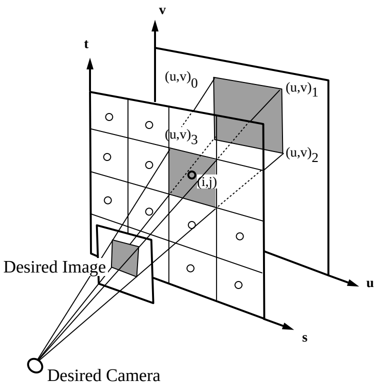

Figure 14: Texture mapping a portion of the *st* plane

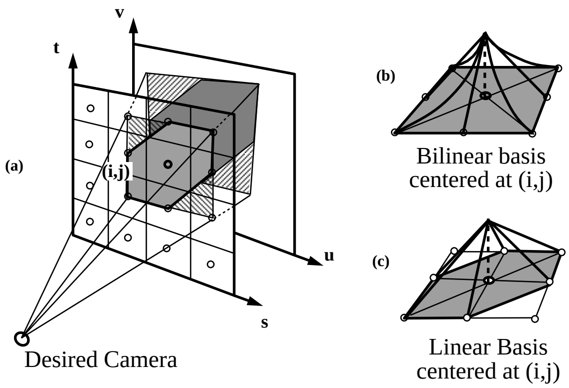
v
t
(a)
(i,j)
u
s
Desired Camera
(b)
Bilinear basis centered at (i,j)
(c)
Linear Basis centered at (i,j)

Figure 15: Quadralinear vs. linear-bilinear

st plane, and texture mapping it using the four texture coordinates $(u,v)_0$, $(u,v)_1$, $(u,v)_2$, and $(u,v)_3$ to index into $\mathrm{tex}_{i,j}$.

Repeating this process for each grid point on the st plane and viewing the result from the desired camera results in a complete reconstruction of the desired image. Thus, if one has an $M \times M$ resolution for the st plane, one needs to draw at most M^2 texture mapped squares, requiring on average, only one ray intersection for each square since the vertices are shared. Since many of the M^2 squares on the st plane are invisible from the desired camera, typically only a small fraction of these squares need to be rendered. The rendering cost is independent of the resolution of the final image.

Intuitively, you can think of the st plane as a piece of holographic film. As your eye moves back and forth you see different things at the same point in st since each point holds a complete image.

Quadralinear Basis

The reconstruction of images from a quadralinear basis Lumigraph can also be performed using a combination of texture mapping and alpha blending. In the quadralinear basis, the support of the basis function at i, j covers a larger square on the st plane than does the box basis (see Figure 15(a)). Although the regions do not overlap in the constant basis, they do in the quadralinear basis. For a given pixel in the desired image, values from 16 4D grid points contribute to the final value.

The quadralinear interpolation of these 16 values can be carried out as a sequence of bilinear interpolations, first in uv and then in st. A bilinear basis function is shown in Figure 15(b) centered at grid point (i,j). A similar basis would lie over each grid point in uv and every grid point in st.

Texture mapping hardware on an SGI workstation can automatically carry out the bilinear interpolation of the texture in uv. Unfortunately, there is no hardware support for the st bilinear interpolation. We could approximate the bilinear pyramid with a linear pyramid by drawing the four triangles shown on the floor of the basis function in Figure 15(b). By assigning α values to each vertex ($\alpha = 1$ at the center, and $\alpha = 0$ at the outer four vertices) and using alpha blending, the final image approximates the full quadralinear interpolation with a linear-bilinear one. Unfortunately, such a set of basis functions do not sum to unity which causes serious artifacts.

A different pyramid of triangles can be built that does sum to unity and thus avoids these artifacts. Figure 15(c) shows a hexagonal region associated with grid point (i,j) and an associated linear basis function. We draw the six triangles of the hexagon with $\alpha = 1$ at the center and $\alpha = 0$ at the outside six vertices[11]. The linear interpolation of α values together with the bilinear interpolation of the texture map results in a linear-bilinear interpolation. In practice we have found it to be indistinguishable from the full quadralinear interpolation. This process requires at most $6\,M^2$ texture mapped, α-blended triangles to be drawn.

Depth Correction

As before, the (u,v) coordinates of the vertices of the texture mapped triangles can be depth corrected. At interior pixels, the depth correction is only approximate. This is not valid when there are large depth changes within the bounds of the triangle. Therefore, we adaptively subdivide the triangles into four smaller ones by connecting the midpoints of the sides until they are (a) smaller than a minimum screen size or (b) have a sufficiently small variation in depth at the three corners and center. The α values at intermediate vertices are the average of the vertices of the parent triangles.

4 Results

We have implemented the complete system described in this paper and have created Lumigraphs of both synthetic and actual objects. For synthetic objects, Lumigraphs can be created either from polygon rendered or ray traced images. Computing all of the necessary images is a lengthy process often taking weeks of processing time.

For real objects, the capture is performed with an inexpensive, single chip Panasonic analog video camera. The capture phase takes less than one hour. The captured data is then "developed" into a Lumigraph. This off-line processing, which includes segmenting the image from its background, creating an approximate volumetric representation, and rebinning the samples, takes less than one day of processing on an SGI Indy workstation.

Once the Lumigraph has been created, arbitrary new images of the object or scene can be generated. One may generate these new images on a ray by ray basis, which takes a few seconds per frame at 450×450 resolution. If one has hardware texture mapping available, then one may use the acceleration algorithm described in Section 3.6.2. This texture mapping algorithm is able to create multiple frames per second from the Lumigraph on an SGI Reality Engine. The rendering speed is almost independent of the desired resolution of the output images. The computational bottleneck is moving the data from main memory to the smaller texture cache.

Figure 19 shows images of a synthetic fruit bowl, an actual fruit bowl, and a stuffed lion, generated from Lumigraphs. No geometric information was used in the Lumigraph of the synthetic fruit bowl. For the actual fruit bowl and the stuffed lion, we have used the approximate geometry that was computed using the silhouette information. These images can be generated in a fraction of a second, independent of scene complexity. The complexity of both the geometry and the lighting effects present in these images would be difficult to achieve using traditional computer graphics techniques.

[11] The alpha blending mode is set to perform a simple summation.

5 Conclusion

In this paper we have described a rendering framework based on the plenoptic function emanating from a static object or scene. Our method makes no assumptions about the reflective properties of the surfaces in the scene. Moreover, this representation does not require us to derive any geometric knowledge about the scene such as depth. However, this method does allow us to include any geometric knowledge we may compute, to improve the efficiency of the representation and improve the quality of the results. We compute the approximate geometry using silhouette information.

We have developed a system for capturing plenoptic data using a hand-held camera, and converting this data into a Lumigraph using a novel rebinning algorithm. Finally, we have developed an algorithm for generating new images from the Lumigraph quickly using the power of texture mapping hardware.

In the examples shown in this paper, we have not captured the complete plenoptic function surrounding an object. We have limited ourselves to only one face of a surrounding cube. There should be no conceptual obstacles to extending this work to complete captures using all six cube faces.

There is much future work to be done on this topic. It will be important to develop powerful compression methods so that Lumigraphs can be efficiently stored and transmitted. We believe that the large degree of coherence in the Lumigraph will make a high rate of compression achievable. Future research also includes improving the accuracy of our system to reduce the amount of artifacts in the images created by the Lumigraph. With these extensions we believe the Lumigraph will be an attractive alternative to traditional methods for efficiently storing and rendering realistic 3D objects and scenes.

Acknowledgments

The authors would like to acknowledge the help and advice we received in the conception, implementation and writing of this paper. Thanks to Marc Levoy and Pat Hanrahan for discussions on issues related to the 5D to 4D simplification, the two-plane parameterization and the camera based aperture analog. Jim Kajiya and Tony DeRose provided a terrific sounding board throughout this project. The ray tracer used for the synthetic fruit bowl was written by John Snyder. The mesh simplification code used for the bunny was written by Hugues Hoppe. Portions of the camera capture code were implemented by Matthew Turk. Jim Blinn, Hugues Hoppe, Andrew Glassner and Jutta Joesch provided excellent editing suggestions. Erynn Ryan is deeply thanked for her creative crisis management. Finally, we wish to thank the anonymous reviewers who pointed us toward a number of significant references we had missed.

References

[1] Adelson, E. H., and Bergen, J. R. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, Landy and Movshon, Eds. MIT Press, Cambridge, Massachusetts, 1991, ch. 1.

[2] Ashdown, I. Near-field photometry: A new approach. *Journal of the Illumination Engineering Society 22*, 1 (1993), 163–180.

[3] Benton, S. A. Survey of holographic stereograms. *Proceedings of the SPIE 391* (1982), 15–22.

[4] Bolles, R. C., Baker, H. H., and Marimont, D. H. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision 1* (1987), 7–55.

[5] Burt, P. J. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition* (June 1988), IEEE Computer Society Press, pp. 144–152.

[6] Chen, S. E. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics, Annual Conference Series, 1995*, pp. 29–38.

[7] Chen, S. E., and Williams, L. View interpolation for image synthesis. In *Computer Graphics, Annual Conference Series, 1993*, pp. 279–288.

[8] Chui, C. K. *An Introduction to Wavelets*. Academic Press Inc., 1992.

[9] Halle, M. W. Holographic stereograms as discrete imaging systems. *Practical Holography VIII (SPIE) 2176* (1994), 73–84.

[10] Hoppe, H. Progressive meshes. In *Computer Graphics, Annual Conference Series, 1996*.

[11] Katayama, A., Tanaka, K., Oshino, T., and Tamura, H. A viewpoint independent stereoscopic display using interpolation of multi-viewpoint images. *Steroscopic displays and virtal reality sytems II (SPIE) 2409* (1995), 11–20.

[12] Klinker, G. J. *A Physical Approach to Color Image Understanding*. A K Peters, Wellesley, Massachusetts, 1993.

[13] Laurentini, A. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 2 (February 1994), 150–162.

[14] Laveau, S., and Faugeras, O. 3-D scene representation as a collection of images and fundamental matrices. Tech. Rep. 2205, INRIA-Sophia Antipolis, February 1994.

[15] Levin, R. E. Photometric characteristics of light-controlling apparatus. *Illuminating Engineering 66*, 4 (1971), 205–215.

[16] Levoy, M., and Hanrahan, P. Light-field rendering. In *Computer Graphics, Annual Conference Series, 1996*.

[17] Lewis, R. R., and Fournier, A. Light-driven global illumination with a wavelet representation of light transport. UBC CS Technical Reports 95-28, University of British Columbia, 1995.

[18] Litwinowicz, P., and Williams, L. Animating images with drawings. In *Computer Graphics, Annual Conference Series, 1994*, pp. 409–412.

[19] McMillan, L., and Bishop, G. Plenoptic modeling: An image-based rendering system. In *Computer Graphics, Annual Conference Series, 1995*, pp. 39–46.

[20] Mitchell, D. P. Generating antialiased images at low sampling densities. *Computer Graphics 21*, 4 (1987), 65–72.

[21] Potmesil, M. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing 40* (1987), 1–29.

[22] Powell, M. J. D., and Swann, J. Weighted uniform sampling - a monte carlo technique for reducing variance. *J. Inst. Maths Applics 2* (1966), 228–236.

[23] Rosenfeld, A., and Kak, A. C. *Digital Picture Processing*. Academic Press, New York, New York, 1976.

[24] Simoncelli, E. P., Freeman, W. T., Adelson, E. H., and Heeger, D. J. Shiftable multiscale transforms. *IEEE Transactions on Information Theory 38* (1992), 587–607.

[25] Snyder, J. M., and Kajiya, J. T. Generative modeling: A symbolic system for geometric modeling. *Computer Graphics 26*, 2 (1992), 369–379.

[26] Szeliski, R. Rapid octree construction from image sequences. *CVGIP: Image Understanding 58*, 1 (July 1993), 23–32.

[27] Taubin, G. A signal processing approach to fair surface design. In *Computer Graphics, Annual Conference Series, 1995*, pp. 351–358.

[28] Terzopoulos, D. Regularization of inverse visual problems involving discontinuities. *IEEE PAMI 8*, 4 (July 1986), 413–424.

[29] Tsai, R. Y. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation RA-3*, 4 (August 1987), 323–344.

[30] Werner, T., Hersch, R. D., and Hlavac, V. Rendering real-world objects using view interpolation. In *Fifth International Conference on Computer Vision (ICCV'95)* (Cambridge, Massachusetts, June 1995), pp. 957–962.

[31] Willson, R. G. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Carnegie Mellon University, 1994.