Unstructured Lumigraph Rendering

Chris Buehler Michael Bosse Leonard McMillan Steven Gortler Michael Cohen
MIT Computer Graphics Group Harvard University Microsoft Research

Abstract

We describe an image based rendering approach that generalizes many image based rendering algorithms currently in use including light field rendering and view-dependent texture mapping. In particular it allows for lumigraph style rendering from a set of input cameras that are not restricted to a plane or to any specific manifold. In the case of regular and planar input camera positions, our algorithm reduces to a typical lumigraph approach. In the case of fewer cameras and good approximate geometry, our algorithm behaves like view-dependent texture mapping. Our algorithm achieves this flexibility because it is designed to meet a set of desirable goals that we describe. We demonstrate this flexibility with a variety of examples.
Keyword Image-Based Rendering

1 Introduction

Image-based rendering (IBR) has become a popular alternative to traditional three-dimensional graphics. Two effective IBR methods are view-dependent texture mapping (VDTM) [3] and the light field/lumigraph [10, 5] approaches. The light field and VDTM algorithms are in many ways quite different in their assumptions and input. Light field rendering requires a large collection of images from cameras whose centers lie on a regularly sampled two-dimensional patch, but it makes few assumptions about the geometry of the scene. In contrast, VDTM assumes a relatively accurate geometric model, but requires only a small number of textures from input cameras that can be in general position.

We suggest that, at their core, these two approaches are quite similar. Both are methods for interpolating color values for a desired ray as some combination of input rays. In VTDM this interpolation is performed using a geometric model to determine which pixel from each input image "corresponds" to the desired ray. Of these corresponding rays, those that are closest in angle to the desired ray are appropriately weighted to make the greatest contribution to the interpolated result.

Light field rendering can be similarly interpreted. For each desired ray (s, t, u, v), one searches the image database for rays that intersect near some (u, v) point on a "focal plane" and have a similar angle to the desired ray, as measured by the rays intersection on the "camera plane" (s, t). In a depth-corrected lumigraph, the focal plane is effectively replaced with an approximate geometric model, making this approach even more similar to view dependent texture mapping.

With this research, we attempt to address the following questions. Is there a generalized rendering framework that spans all of these various image-based rendering algorithms, having VDTM and lumigraph/light fields as extremes? Might such an algorithm adapt well to various numbers of input images from cameras in general configurations while also permitting various levels of geometric accuracy?

In this paper we approach the problem by first describing a set of goals that we feel any image based rendering algorithm should have. We find that no previous IBR algorithm simultaneously satisfies all of these goals. These algorithms behave quite well under appropriate assumptions on their input, but can produce unnecessarily poor renderings when these assumptions are violated.

In this paper we describe "unstructured lumigraph rendering" (ULR), an approach for IBR that generalizes both lumigraph and VDTM rendering. Our algorithm is designed specifically with our stated goals in mind. As a result, our renderer behaves well with a wide variety of inputs. These include source cameras that are not on a common (s, t) plane, and even sets of source cameras taken by walking forward into the scene.

It should be no surprise that our algorithm bears many resemblances to previously published approaches. The main contribution of our algorithm is that, unlike all previously published methods, it is designed to meet our listed goals and, thus, works well on a wide range differing inputs, from a few images with an accurate geometric model to many images with minimal geometric information.

2 Previous Work

The basic idea of view dependent texture mapping (VDTM) was put forth by Debevec et al. [3] in their Facade image-based modeling and rendering system. Facade was designed to estimate geometric models consistent with a small set of source images. As part of this system, a rendering algorithm was developed where pixels from all relevant data cameras were combined and weighted to determine a view-dependent texture for the derived models. In later work Debevec et al [4] describe a real-time VDTM algorithm. In this algorithm, each polygon in the geometric model maintains a "view map" data structure that is used to quickly determine a set of three data cameras that should be used to texture it. Most real-time VDTM algorithms use projective texture mapping [6] for efficiency.

At the other extreme, Levoy and Hanrahan [10] described the light field rendering algorithm where a large collection of images were used to render novel views of a scene. This collection of images was captured from cameras whose centers lie on a regularly sampled two-dimensional plane. Light fields otherwise make few assumptions about the geometry of the scene. Gortler et al. [5] described a similar rendering algorithm called the lumigraph. In addition, the authors of the lumigraph paper suggests many workarounds to overcome limitations of the basic approach, including a "rebinning" process to handle source images acquired from general camera positions, and a "depth-corrected" extension to allow for more accurate ray reconstructions when there is an insufficient number of source cameras.

Many extensions, enhancements, alternatives, and variations to these basic algorithms have since been suggested. This includes techniques for rendering digitized three-dimensional models in combination with acquired images such as Pulli et al. [13] and Wood et al. [18]. Shum et al. [17] has suggested alternate lower dimensional lumigraph approximations that use only approximate depth correction. Heigl et al. [7] described an algorithm to perform IBR from an unstructured set of data cameras where the projections of the source cameras' centers were projected into the desired image plane, triangulated, and used to reconstruct the interior pixels. Isaksen et al. [9] have shown how the common "image-space" coordinate frames used in light-field rendering can be viewed as a focal plane for dynamically generating alternative ray reconstructions. A formal analysis of the trade off between number of cameras fidelity of geometry is presented in [1].

3 Goals

We begin by presenting the following list of desirable properties that we feel an ideal image-based rendering algorithm should have. We also point out that no previously published method satisfies all of these goals.

Use of geometric proxies: When geometric knowledge is present, it should be used to assist in the reconstruction of a desired ray (see figure 1). We refer to such approximate geometric information as a *proxy*. The combination of accurate geometric proxies with surface reflectance properties that are nearly Lambertian allows for high quality reconstructions from relatively few source images. The reconstruction process merely entails looking for rays from source cameras that see the "same" point. This idea is central to all VDTM algorithms. It is also the distinguishing factor in geometry-corrected lumigraphs and surface light-field algorithms. Approximate proxies, such as the focal-plane abstraction used by Isaksen [9], allow for the accurate reconstruction of rays at specific depths from standard light fields.

With a highly accurate geometric model, the visibility of any surface point relative to a particular source camera can also be determined. If a camera's view of the point is occluded by some other point on the geometric model then that camera should not be used in the reconstruction of the desired ray. When possible, image-based algorithms should compute visibility.

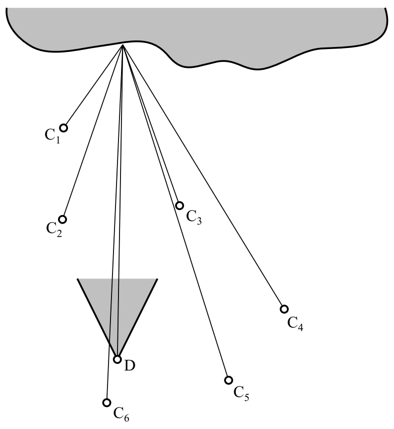

Figure 1: When available, approximate geometric information should be used to determine which source rays correspond well to a desired ray.

Epipole consistency: When a desired ray passes through the center of projection of a source camera it can be trivially reconstructed from the ray database (assuming a sufficiently high-resolution input image and the ray falls within the camera's field of view) (see Figure 2). In this case, an ideal algorithm should return a ray from source image. An algorithm with epipole consistency will reconstruct this ray correctly without any geometric information. With large numbers of source cameras, algorithms with epipole consistency can create accurate reconstructions with essentially no geometric information. Light field and lumigraph algorithms were designed specifically to maintain this property.

Surprisingly, many real-time VDTM algorithms, do not ensure this property and so will not work properly when given poor geometry. The algorithms described in [13, 2], reconstruct all of the rays in a fixed desired view using a fixed selection of three source images but, as shown by the original light-field paper, proper reconstruction of a desired image may involve using some rays from each of the source images. The algorithm described in [4], always uses three source cameras to reconstruct all of the desired pixels for an observed polygon of the geometry proxy. This departs from epipole consistency if the proxy is coarse. The algorithm of Heigl et al. [7] is an notable exception that, like a light field or lumigraph, maintains epipole consistency.

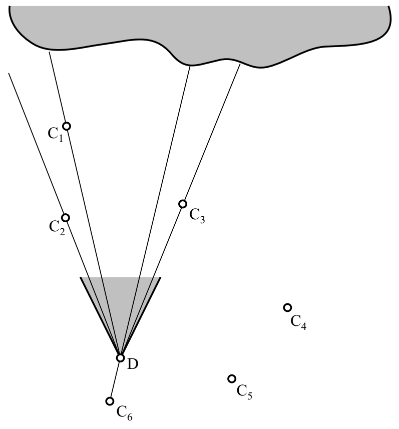

Figure 2: When a desired ray passes through a source camera center, that source camera should be emphasized most in the reconstruction.

Resolution sensitivity: In reality, image pixels are not really measures of a single ray, but instead an integral over a set of rays subtending some solid angle. This angular extent should be accounted for by the rendering algorithm (See Figure 3). In particular, if a source camera is far away from an observed surface, then its pixels represent integrals over large regions of the surface. If these ray samples are used to reconstruct a ray from a closer viewpoint, an overly blurred reconstruction will result (assuming the desired and reference rays subtend comparable solid angles). Resolution sensitivity is an important consideration when combining source rays from cameras with different fields-of-view, or when combining rays from cameras located various distances from the imaged surface. It is seldom considered in traditional light-field and lumigraph rendering, since the source cameras usually have common focal lengths and are located roughly the same distance from any reconstructed surface. However, when using unstructured input cameras, a wider variation in camera-surface distances can arise, and it is important to consider image resolution in the ray reconstruction process. Few image-based rendering approaches have dealt with this problem.

Unstructured input: It is very desirable for an image-based rendering algorithm to accept input images from cameras in general position. The original light-field method assumes that the cameras are arranged at evenly spaced positions on a single plane. This limits the applicability of this method since it requires a special capture gantry that is both expensive and is difficult to use in many settings [11].

The lumigraph paper describes an acquisition system that uses a hand-held video camera to acquire input images [5]. They then apply a preprocessing step, called rebinning, that resamples the input images from virtual source cameras situated on a regular grid. The rebinning process adds in an additional layer of image quality degradation; a rebinned lumigraph cannot, in general, reproduce its input images. The surface light-field algorithm suffers from the same problem.

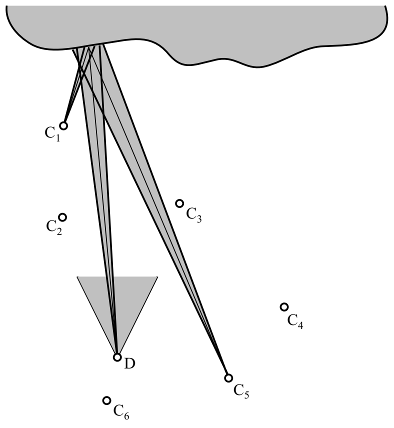

Figure 3: When cameras have different distances from the proxy, their resolution differs. Care should be taken not to obtain an overly blurry reconstruction

Equivalent ray consistency: Through any empty region of space, the ray along a given line-of-sight should be reconstructed consistently, regardless of the viewpoint position (unless dictated by other goals such as resolution sensitivity or visibility) (See Figure 4). This is not the case for unstructured rendering algorithms that use desired-image-space measurements of "ray closeness" [7]. As shown in figure 4, two desired cameras that share a desired ray will have a different "closest" cameras, therefore giving different reconstructions.

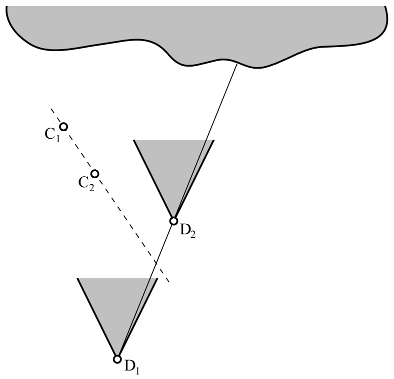

Figure 4: When ray angle is measured in the desired view, one can get different reconstructions for the same ray. The algorithm of Heigl et al would determine C_2 to be the closest camera for D_1, and C_1 to be the closest camera for D_2.

Continuity: When one requests a ray with vanishingly small distance from a previous ray but intersects the same or nearby point on the proxy, the reconstructed ray should have a color value that is correspondingly close to the previously reconstructed color. Reconstruction continuity is important to avoid both temporal and spatial artifacts. For example, the weight used for a camera should drop to zero as one approaches the boundary of its field of view [3], or as one approaches a part of a surface that is not seen by a camera due to visibility [14].

The VDTM algorithm of [4], which uses a triangulation of the directions to source cameras to pick the "closest three" does not guarantee spatial continuity, even at high tesselation rates of the proxy. Nearby points on the proxy can have different triangulations of the "source camera view map" giving very different reconstructions.

While this objective is subtle, it is nonetheless important, since lack of this continuity introduces significant artifacts.

Minimal angular deviation: In general, the choice of which data cameras are used to reconstruct a desired ray should be based on a natural and consistent measure of closeness (See Figure 5). In particular, we use source image rays with the most similar angle to the desired ray.

Interestingly, the light-field and lumigraph rendering algorithms that select rays based on how close the ray passes to a source camera manifold do not quite agree with this measure. As shown in figure 5, the "closest" ray on the (s, t) plane is not the closest one measured in angle.

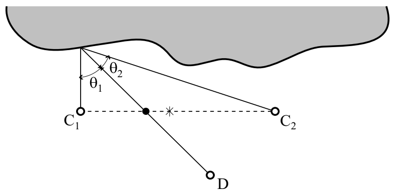

Figure 5: Angle deviation is a natural measure of ray difference. Interestingly, as shown in this case, the two plane parameterization gives a different ordering of "closeness" Source camera C_2's ray is closer in angle to the desired ray, but the ray intersects the camera (s,t) plane closer to C_1.

Real time: It is desirable that the rendering algorithm run at interactive rates. Most of the image-based algorithms that we considered here achieve this goal.

Table 1 summarizes the goals what we would consider an ideal rendering method. It also compares ULR to other published methods.

4 Algorithm

We present a lumigraph rendering technique that directly renders views from an unstructured collection of input images. The input to our algorithm is a collection of source images along with their associated camera pose estimates.

4.1 Camera Blending Field

Our real-time rendering algorithm works by evaluating a "camera blending field" at a set of vertices in the desired image plane and interpolating this field over the whole image. This blending field describes how each source camera is weighted to reconstruct a given pixel; the calculation of this field should be based on our stated goals, and includes factors related to ray angular difference, estimates of undersampling, and field of view [13, 12].

We begin by discussing how angle similarity is best utilized. A given desired ray r_d, intersects the surface proxy at some frontmost point p. We consider the rays r_i that connect p to the centers c_i of each source camera i. For each source camera we denote the angular difference between r_i and r_d as $\text{angDiff}(i)$ (see figure 6).

One way to define a smooth blending weight "angBlend" based on our measured angDiff would be to set a constant threshold, "angThresh". angBlend could then decrease from one to zero as angDiff increases from zero to angThresh.

This approach proves unsatisfactory when using unstructured input data. In order to account for desired pixels where there are no angularly close cameras we would need to set a large angThresh.

	lh96	gor96	deb96	pul97	deb98	pigh98	hei99	wood00	ULR
Use of Geometric Proxy	n	y	y	y	y	y	y	y	y
Epipolar Consistency	y	y	y	n	n	n	y	y	y
Resolution Sensitivity	n	n	n	n	n	n	n	n	y
Unstructured Input	n	resamp	y	y	y	y	y	resamp	y
Equivalent Ray Consistency	y	y	y	y	y	y	n	y	y
Continuity	y	y	y	y	n	y	y	y	y
Minimal Angular Deviation	n	n	y	n	y	y	n	y	y
Real-Time	y	y	n	y	y	y	y	y	y

Table 1: Comparison of algorithms according to our desired goals.

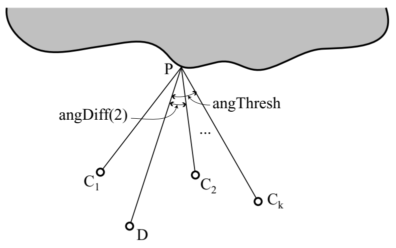

Figure 6: The angle of the kth angularly furthest camera is used as an angle threshold.

But using a large angThresh would blend unnecessary cameras at desired pixels where there are many angularly close cameras. This would result in an unnecessary loss of view dependence.

An adaptive way to compute an angBlend would be to always use the k source cameras with smallest angDiff. In this case we must take care that a particular camera's angBlend falls to zero as it leaves the set of "closest k".

We can accomplish this by combining the ideas of "closest k" and the use of an angular threshold. We define angThresh locally to be the kth largest value of angDiff searching over the source cameras and compute the blend as

$$\mathrm{angBlend}(i) = \max(0, 1 - \frac{\mathrm{angDiff}(i)}{\mathrm{angThresh}})$$

In order to have the blending weights sum to unity we normalize as

$$\mathrm{normalizedAngBlend}(i) = \frac{\mathrm{angBlend}(i)}{\sum_{j=1}^{k} \mathrm{angBlend(j)}}$$

This is well defined as long as not all k closest cameras are equidistant. For a given camera i, normalizedAngBlend(i) is a smooth function as one varies the desired ray along a continuous proxy surface.

Resolution To reconstruct a given desired ray r_d, we do not want to use source rays r_i that significantly undersample the observed proxy point p. Given the positions of the cameras, their fields of view, and p's normal, we could compute an accurate prediction of the degree of undersampling. For simplicity we perform this computation as

$$\mathrm{resDiff}(i) = \max(0, \parallel p - c_i \parallel - \parallel p - d \parallel)$$

where d is the center of the desired camera.

Given the two difference measurements angDiff and resDiff, we compute a combined difference measure as the weighted combination:

$$\mathrm{angResDiff}(i) = \alpha\ \mathrm{angDiff}(i) + \beta\ \mathrm{resDiff}(i)$$

Using this new distance measure, we can compute the k closest cameras, define a "angResThresh" threshold and compute "angResBlend(i)"

$$\mathrm{angResBlend}(i) = \max(0, 1 - \frac{\mathrm{angResDiff}(i)}{\mathrm{angResThresh}})$$

Field of view and visibility In our rendering, we do not want to use rays that fall outside the field of view of the source camera. To incorporate this, when searching for the k closest cameras as measured by angResDiff, we only look at cameras where r_i falls within its field of view. To incorporate this factor smoothly, we define fovBlend(i) to be a "feathering" that goes to zero as r_i approaches the edge of the fov of c_i.

By multiplying angResBlend(i) with fovBlend(i), we obtain

$$\mathrm{angResFovBlend}(i) = \mathrm{angResBlend}(i)\mathrm{fovBlend}(i)$$

Normalizing over all the i gives us the final

$$\mathrm{normalizedAngResFovBlend}(i) = \frac{\mathrm{angResFovBlend}(i)}{\sum_{j=1}^{k} \mathrm{angResFovBlend(j)}}$$

With an accurate proxy, we would in fact compute visibility between p and c_i and only consider source rays that potentially see p [4]. In our setting we have proxies with unit depth complexity, so we have not needed to implement visibility computation. One complication is how to incorporate visibility and smoothness together into one metric. A proper feathering approach would use an algorithm like that described in [12, 14].

In figure 7 we visualize a camera blending field by applying this computation at each desired pixel. In this visualization, each source camera is assigned a color. These colors are blended at each pixel to show how they combine to define the blending field.

4.2 Real time rendering

The basic strategy of our real time renderer is to compute the camera blending only at a discrete set of points in the image plane, triangulate these points, and interpolate the camera blending over the image (see figure 9).

To obtain a real time rendering algorithm we take advantage of the fact that pixel correspondence over a planar region of the proxy is projective. Our rendering can then use projective texture mapping to map the source pixels onto the desired image.

Our strategy is to define a triangulation of the image plane using the following steps (see figure 8).

- Only a single planar region of the proxy must be observed through each triangle in the image plane. This will allow us to use texture mapping hardware to assist our rendering. With this property in mind, project all of the vertices and edges of the proxy into the desired image plane. The edges are used to

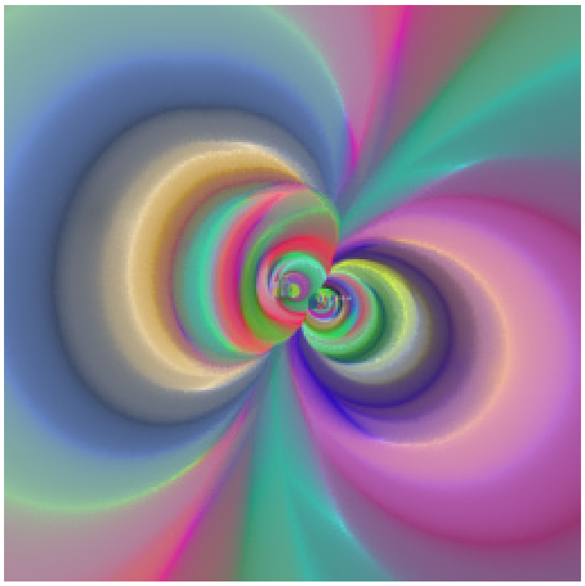

Figure 7: A visualized color blending field. Camera weights are computed at each pixel. This example is from the "hallway" dataset shown in the results section.

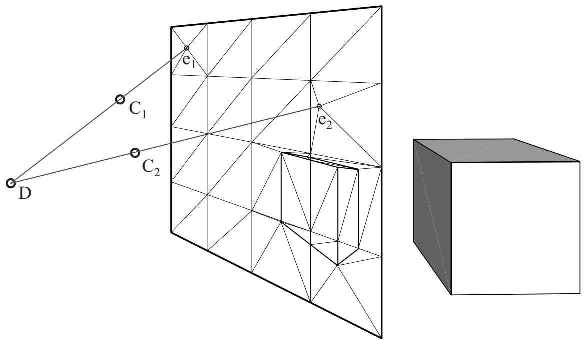

Figure 8: Our real time renderer uses the projection of the proxy, the projection of the source camera centers and a regular grid to triangulate the image plane.

constrain the triangulation. [1] New vertices are inserted at all edge-edge crossings.

- To maintain epipole consistency, we include a vertex at the desired image plane projection of each source camera's center.

- To obtain a sufficiently dense vertex set, needed to capture the interesting spatial variation of the camera blending weights, we include a regular grid of vertices on the desired image plane. The edges of this regular grid are also added as constraints in the triangulation. This static structure of edges helps keep the computed triangulation from having triangle "flips" as the desired camera is moved and the other vertices move relative to each other.

- Given this set of vertices and edges, we create a constrained Delaunay triangulation of the image plane using the constrained Delaunay code of [16].

- At each vertex of the triangulation, we compute and store a set of cameras and their associated blending weights. Recall that at a vertex, these weights sum to one.

- Over the face of a triangle we interpolate these blending weights linearly.

- We render the desired image as a set of projectively mapped triangles as follows. Suppose that there are a total of m unique cameras with nonzero blending weights at the three vertices of a triangle. Then this triangle is rendered m times, using the texture from each of the m cameras. When a triangle is rendered using one of the source camera's texture, each of its three vertices is assigned an alpha value equal to its weight at that vertex. The texture matrix is set up to properly projectively texture the source camera's data onto the rendered proxy triangle. [2]

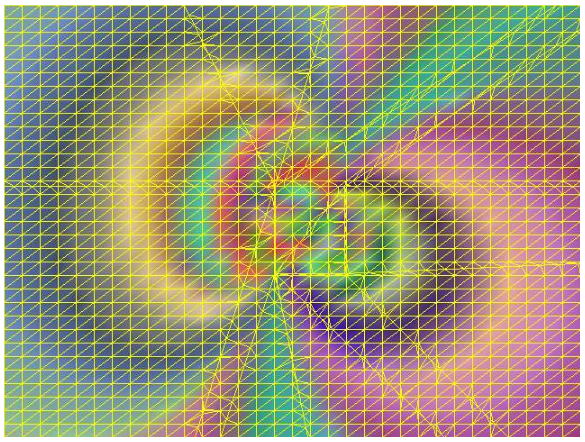

Figure 9: A visualized color blending field from the real time renderer. Camera weights are computed at each vertex of the triangulation.

It is interesting to note that if rays are shot only at the projected epipoles, then one gets a rendering algorithm similar to that of [7].

5 Results

We have collected a wide variety of data sets to test the ULR algorithm. In the following, we describe how the data sets are created and show some renderings from our real-time ULR algorithm. These examples are also shown in the accompanying video.

Pond The pond dataset (Figure 11a) is constructed from a two second (60 frame) video sequence that we captured with a digital hand-held video camera. The camera is calibrated to recover the focal length and radial distortion parameters of the lens. After running a feature tracker on the sequence, the camera's positions are recovered using structure-from-motion techniques borrowed from computer vision.

We use a single plane for the geometric proxy. The position of the plane is computed based on the positions of the cameras and the positions of the three-dimensional structure points that are

[1]In our system we project all of the vertices and edges regardless of visibility. This conservative algorithm can create more than the necessary number of regions. More efficient approaches are certainly possible.

[2]To properly do the projective texturing, we must know which plane is observed through some particular triangle. There are many possible ways to do this (such as using frame-buffer reading). In our system, we typically have proxies of low depth complexity, so we actually render each triangle ml times, where l is the depth complexity observed through the triangle, and let the z-buffer toss out all but the frontmost rendering.

computed during the vision processing. Specifically, the plane is oriented (roughly) parallel to the camera image planes and placed at the average $1/z$ distance [1] from the cameras.

Since the cameras are arranged along a linear path, and the proxy is a single plane, the pond dataset exhibits parallax in only one dimension. However, the effect is convincing for simulating views at about the height that the video camera was held.

Robot The Robot dataset (Figure 11b) was constructed in the same manner as the pond dataset. In fact, it is quite simple to build unstructured lumigraphs from short video sequences. The robot sequence exhibits view-dependent highlights and reflections on its leg and on the tabletop.

Helicopter The Helicopter dataset (Figure 11c) uses the ULR algorithm to achieve an interesting hack: motion in a lumigraph. To create this lumigraph, we exploit the fact that the motion in the scene is periodic.

The lumigraph is constructed from a *continuous* 30 second video sequence in which the camera is moved back and forth repeatedly over the scene. The video frames are then calibrated spatially using the structure-from-motion technique described above. The frames are also calibrated temporally by measuring accurately the period of the helicopter. Assuming the framerate of the camera is constant, we can assign each video frame a timestamp expressed in terms of the period of the helicopter. Again, the geometric proxy is a plane.

During rendering, a separate unstructured lumigraph is constructed and rendered on-the-fly for each time instant. Since very few images occur at precisely the same time, the unstructured lumigraph is constructed over a time window. The current time-dependent rendering program (an early version of the ULR algorithm) ignores the timestamps of the images when sampling camera weights. However, it would be straightforward to blend cameras in and out temporally as the time window moves.

Knick-knacks The Knick-knacks dataset (Figure 11d) exhibits camera motion in both the vertical and horizontal directions. In this case, the camera positions are determined using a Faro digitizing arm. The camera is synchronized with and attached to the Faro arm. When the user takes a picture, the location and orientation of the camera is automatically recorded. Again the proxy is a plane, which we position interactively by “focusing” [9] on the red car in the foreground.

Car While the previous datasets primarily occupy the light field end of the image-based spectrum, the Car dataset (11e) demonstrates the VDTM aspects of our algorithm. This dataset consists of only 36 images and a 500 face polygonal geometric proxy. The images are arranged in 10 degree increments along a circle around the car. The images are from an “Exterior Surround Video” (similar to a QuicktimeVR object) database found on the carpoint.msn.com website.

The original images have no calibration information. Instead, we simply assume that the cameras are on a perfect circle looking inward. Using this assumption, we construct a rough visual hull model of the car. We simultaneously adjust the camera focal lengths to give the best reconstruction. We simplify the model to 500 faces while maintaining the hull property according to the procedure in [15]. Note that the geometry proxy is significantly larger than the actual car, and it also has noticeable polygonal silhouettes. However, when rendered using the ULR algorithm, the rough shape of the proxy is largely hidden. In particular, the silhouettes of the rendered car are determined by the images and not the proxy, resulting in a smooth contour.

Note that in these renderings, the camera blending field is only sampled at the vertices of the proxy. This somewhat sparse sampling gives reasonable results when the complexity of the proxy is high relative to the number of cameras.

Hallway The Hallway dataset (Figure 11f) is constructed from a video sequence in which the camera moves forward into the scene. The camera is mounted on an instrumented robot that records its position as it moves. This forward camera motion is not commonly used in lumigraph-style image-based rendering techniques, but it is handled by our algorithm with no special considerations.

The proxy for this scene is a six sided rectangular tunnel that is roughly aligned with the hallway walls [8]. None of the cabinets, doors, or other features are explicitly modeled. However, virtual navigation of the hallway gives the impression that the hallway is populated with actual three-dimensional objects.

The Hallway dataset also demonstrates the need for resolution consideration. In Figure 10a, we show the types of blurring artifacts that can occur if resolution is ignored. In Figure 10b, we show the result of using our simple resolution heuristic. Low resolution images are penalized, and the wall of the hallway appears much sharper, with a possible loss of view-dependence where the proxy is poor. Below each rendering in Figure 10 appears the corresponding camera blending field. Note that 10b uses fewer images on the left hand side of the image, which is where the original rendering had most problems with excessive blurring. In this case, the removed cameras are too far behind the viewer.

6 Conclusion

We have presented a new image-based rendering technique for rendering convincing new images from unstructured collections of input images. We have demonstrated that the algorithm can be executed efficiently in real-time.

Our technique is a generalization of the lumigraph and VDTM rendering algorithms. We allow for unstructured sets of cameras as well as variable information about scene geometry. Our real-time implementation has all the benefits of real-time structured lumigraph rendering, including speed and photorealistic quality, and it also allows the use of geometric proxies, unstructured input cameras, and variations in resolution and field-of-view.

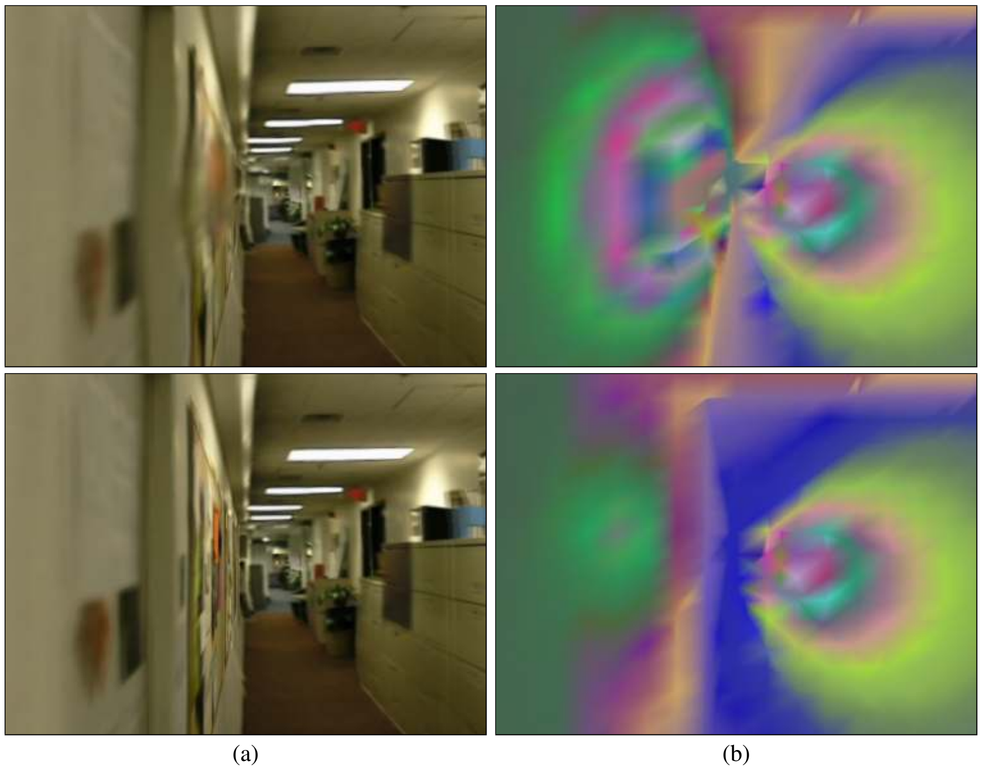

Figure 10: Operation of our scheme for handling resolution issues: (a) shows the hallway scene with no consideration of resolution and (b) shows the same viewpoint rendered with consideration of resolution. Below each image is the corresponding camera blending field.

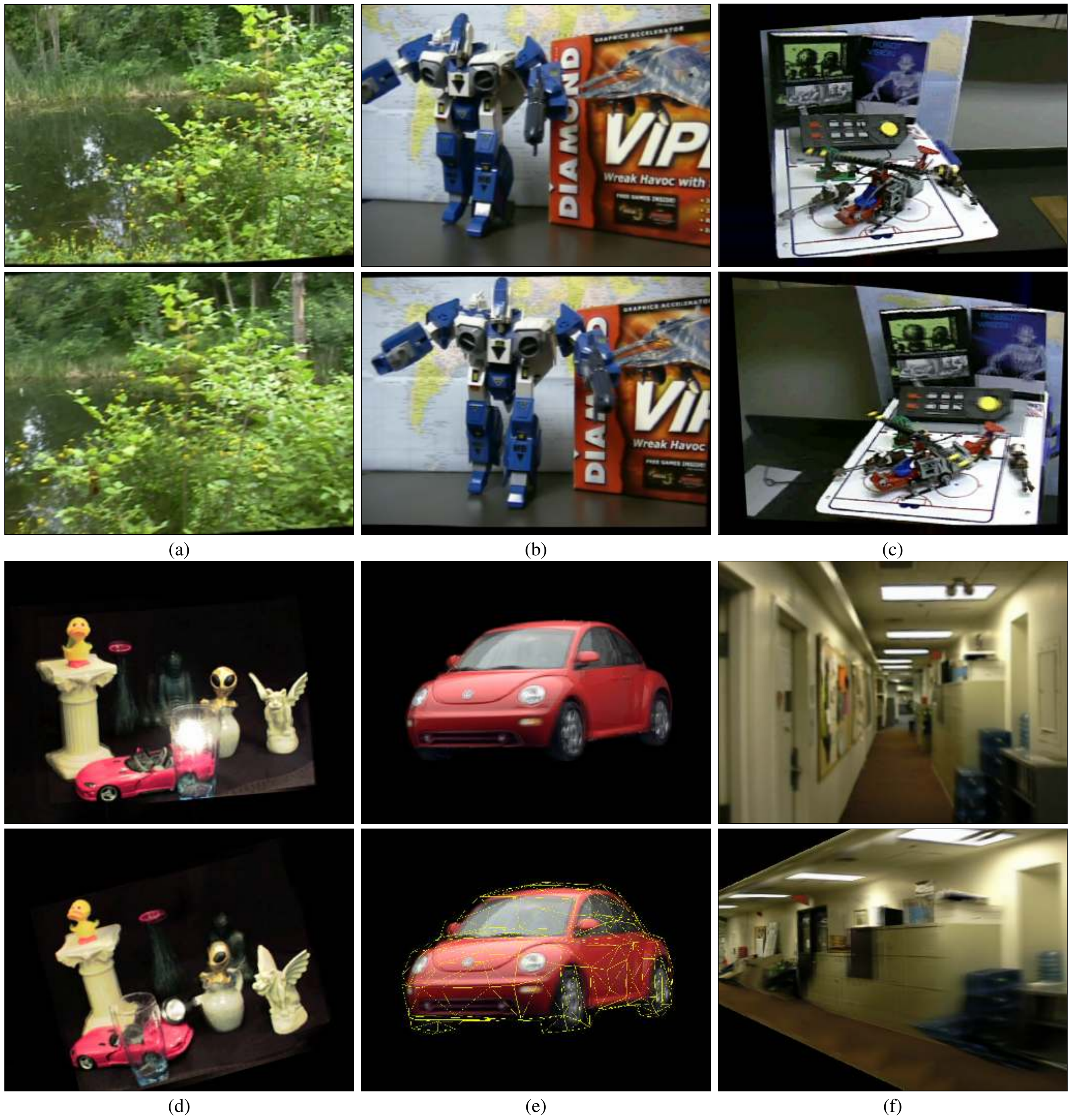

Figure 11: Renderings from the real-time unstructured lumigraph renderer. (a) Pond,(b) Robot, (c) Helicopter, (d) Knick-knacks, (e) Car, (f) Hallway