Unstructured Lumigraph Rendering

Chris Buehler Michael Bosse Leonard McMillan Steven Gortler Michael Cohen
MIT Laboratory for Computer Science Harvard University Microsoft Research

Abstract

We describe an image based rendering approach that generalizes many current image based rendering algorithms, including light field rendering and view-dependent texture mapping. In particular, it allows for lumigraph-style rendering from a set of input cameras in arbitrary configurations (i.e., 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. When presented with fewer cameras and good approximate geometry, our algorithm behaves like view-dependent texture mapping. The algorithm achieves this flexibility because it is designed to meet a set of specific 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 if any assumptions about the geometry of the scene. In contrast, VDTM assumes a relatively accurate geometric model, but requires only a small number of images from input cameras that can be in general positions. These images are then "projected" onto the geometry for rendering.

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 VDTM this interpolation is performed using a geometric model to determine which pixel from each input image "corresponds" to the desired ray in the output image. Of these corresponding rays, those that are closest in angle to the desired ray are 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 ray's 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.

Given these related IBR approaches, we attempt to address the following questions: Is there a generalized rendering framework that spans all of these 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 suggesting a set of goals that any image based rendering algorithm should have. We find that no previous IBR algorithm simultaneously satisfies all of these goals. Therefore these algorithms behave quite well under appropriate assumptions on their input, but may produce unnecessarily poor renderings when these assumptions are violated.

We then describe an algorithm for "unstructured lumigraph rendering" (ULR), that generalizes both lumigraph and VDTM rendering. Our algorithm is designed specifically with the 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 plane, such as source images taken by moving forward into a scene, a configuration that would be problematic for previous IBR approaches.

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

2 Previous Work

The basic approach to view dependent texture mapping (VDTM) is put forth by Debevec et al. [3] in their Façade image-based modeling and rendering system. Façade is 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 cameras were combined and weighted to determine a view-dependent texture for the derived geometric 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 input cameras that should be used to texture it. Like most real-time VDTM algorithms, this algorithm uses hardware supported projective texture mapping [6] for efficiency.

At the other extreme, Levoy and Hanrahan [10] describe the light field rendering algorithm, in which a large collection of images are used to render novel views of a scene. This collection of images is 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] describe a similar rendering algorithm called the lumigraph. In addition, the authors of the lumigraph paper suggest 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-correction" extension to allow for more ac-

curate ray reconstructions from an insufficient number of source cameras.

Many extensions, enhancements, alternatives, and variations to these basic algorithms have since been suggested. These include 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] suggests alternate lower dimensional lumigraph approximations that use approximate depth correction. Heigl et al. [7] describe 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] show 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 the number of cameras and the fidelity of geometry is presented in [1].

3 Goals

We begin by presenting a list of desirable properties that we feel an ideal image-based rendering algorithm should have. No previously published method satisfies all of these goals. In the following section we describe a new algorithm that attempts to meet these goals while maintaining interactive rendering rates.

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 nearly Lambertian surface properties 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 consider visibility in their reconstruction.

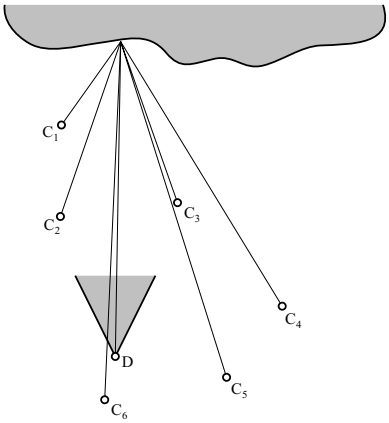


Figure 1: When available, approximate geometric information should be used to determine which source rays correspond well to a desired ray. Here C_x denotes the position of a reference camera, and D is desired novel viewpoint.

Unstructured input: It is also 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 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 apply a preprocessing step, called rebinning, that resamples the input images from virtual source cameras situated on a regular grid. This rebinning process adds an additional reconstruction and sampling step to lumigraph creation. This extra step tends to degrade the overall quality of the representation. This can be demonstrated by noting that a rebinned lumigraph cannot, in general, reproduce its input images. The surface light field algorithm suffers from essentially the same resampling problem.

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 the 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 are designed specifically to maintain this property.

Surprisingly, many real-time VDTM algorithms do not ensure this property, even approximately, and therefore, 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 on a 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.

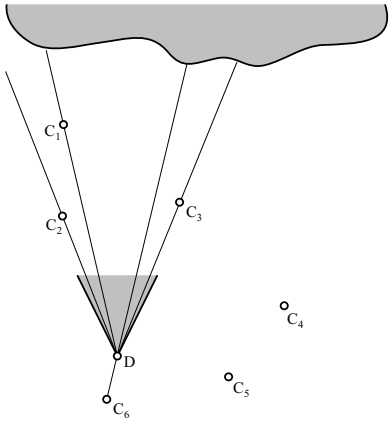


Figure 2: When a desired ray passes through a source camera center, that source camera should be emphasized most in the reconstruction. Here this case occurs for cameras C_1, C_2, C_3, and C_6.

Minimal angular deviation: In general, the choice of which input images are used to reconstruct a desired ray should be based on a natural and consistent measure of closeness (See Figure 3). In particular, source image rays with similar angles to the desired ray should be used when possible.

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 3, the "closest" ray on the (s,t) plane is not necessarily the closest one measured in angle.

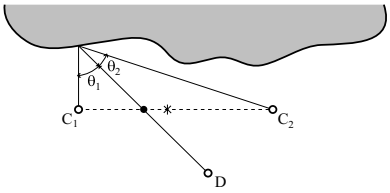


Figure 3: Angle deviation is a natural measure of ray difference. Interestingly, as shown in this case, the two plane parameterization gives a different ordering of "closeness." Source camera C_2's ray is closer in angle to the desired ray, but the ray intersects the camera (s,t) plane closer to C_1.

Continuity: When one requests a ray with infinitesimal small distance from a previous ray intersecting a 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 contribution due to any particular camera should fall 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 provide spatial continuity, even at high tessellation rates of the proxy. Nearby points on the proxy can have very different triangulations of the "source camera view map" resulting in very different reconstructions. While this objective is subtle, it is nonetheless important, since lack of such continuity can introduce noticeable artifacts.

Resolution sensitivity: In reality, image pixels are not really measures of a single ray, but instead an integral over a set of rays subtending a small solid angle. This angular extent should ideally be accounted for by the rendering algorithm (See Figure 4). For example, 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 focal lengths, or when combining rays from cameras with varying distance and obliqueness relative to 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-to-surface distances can arise, and it is important to consider image resolution in the ray reconstruction process. To date, no image-based rendering approaches have dealt with this problem.

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 5). This is not the case for unstructured rendering algorithms that use desired-image-space measurements of "ray closeness" [7]. As shown in Figure 5, two desired cameras that share a desired ray will have a different "closest" cameras, therefore giving different reconstructions.

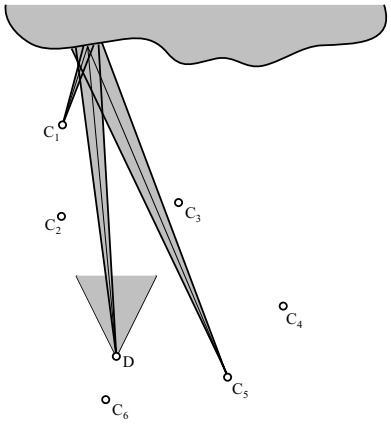


Figure 4: When cameras have different views of the proxy, their resolution differs. Here cameras C_1 and C_5 see the same proxy point with different resolutions.

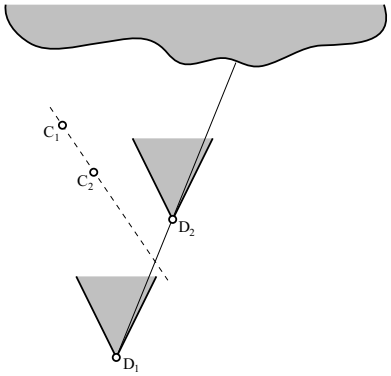


Figure 5: When ray angle is measured in the desired view, one can get different reconstructions for the same ray. The algorithm of Heigl et al. would determine C_2 to be the closest camera for D_1, and C_1 to be the closest camera for D_2. The switch in reconstructions occurs when the desired camera passes the dotted line.

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. In designing a new algorithm to meet our desired goals we have also strived to ensure that the result is still computed efficiently.

Table 1 summarizes the goals of what we would consider an ideal rendering method. It also compares our Unstructured Lumigraph Rendering (ULR) method to other published methods.

4 Unstructured Lumigraph Rendering

We present a lumigraph rendering technique that directly renders views from an unstructured collection of input images. The input to our Unstructured Lumigraph Rendering (ULR) algorithm is a collection of source images along with their associated camera pose estimates as well as an approximate geometric proxy for the scene.

4.1 Camera Blending Field

Our real-time rendering algorithm works by first 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 is based on our stated

Goals	lh96	gor96	deb96	pul97	deb98	pigh98	hei99	wood00	ULR
Use of Geometric Proxy	n	y	y	y	y	y	y	y	y
Epipole 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 the algorithms lh96 [10], gor96 [5], deb96 [3], pul97 [13], deb98 [4], pigh98 [12], hei99 [7], wood00 [18], and ULR according to our desired goals.

goals, and includes factors related to the angular difference between the desired ray and those available in the given image set, estimates of undersampling, and field-of-view [13, 12]. Given the blending field, each pixel of the desired image is then reconstructed by a weighted average of the corresponding pixels in each weighted input image.

We begin by discussing how cameras are weighted based on angle similarity. Then, we generalize our approach for other considerations such as resolution and field-of-view.

A given desired ray r_d, intersects the surface proxy at some front-most point p. We consider the rays r_i from p to the centers C_i of each source camera i. For each source camera we define the angular penalty, $\text{penalty}_{ang}(i)$, as the angular difference between r_i and r_d. (see Figure 6).

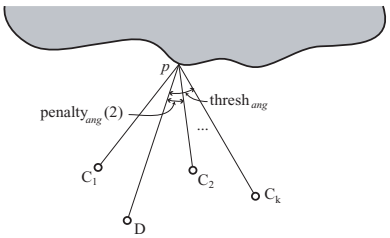


Figure 6: The angle of the k^{th} farthest camera is used as an angle threshold.

When $\text{penalty}_{ang}(i)$ is zero we would like the blending weight used for camera i, $w_{ang}(i)$, to be at a maximum. To best satisfy epipole consistency, this maximum weight should be infinite relative to the weights of all other cameras. It is unclear, however, when $w_{ang}(i)$ for a particular camera should drop to zero.

For example, one way to define a smooth blending weight would be to set a global threshold thresh_{ang}. Then, the weight $w_{ang}(i)$ could decrease from w_{max} to zero as $\text{penalty}_{ang}(i)$ increases from zero to thresh_{ang}. 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 thresh_{ang}. But using a large thresh_{ang}would blend too many cameras at pixels where there are many angularly close cameras, giving an unnecessarily blurred result.

One way to solve this of problem is to use a k-nearest neighbor interpolation approach. That is, we consider only the k cameras with smallest $\text{penalty}_{ang}(\cdot)$s when reconstructing a desired ray. All other cameras are assigned a weight of zero. In this approach, we must take care that a particular camera's $w_{ang}(i)$ falls to zero as it leaves the set of closest k. We accomplish this by defining an adaptive thresh_{ang}. We define thresh_{ang}locally to be the k^{th} largest value of $\text{penalty}_{ang}(\cdot)$ in the set of k-nearest cameras. We then compute a weight function that has maximum value w_{max} at zero and has value zero at thresh_{ang}.

The blending weight that we use in our real-time system is

$$w_{ang}(i) = 1 - \frac{\text{penalty}_{ang}(i)}{\text{thresh}_{ang}}.$$

This weight function has a maximum of 1 and falls off linearly to zero at thresh_{ang}, and so consequently it does not exactly satisfy epipole consistency. Epipole consistency can be enforced by multiplying $w_{ang}(i)$ by $1/\text{penalty}_{ang}(i)$ (or by other ways) at the cost of more computation.

We then normalize the blending weights to sum to unity,

$$\tilde{w}_{ang}(i) = \frac{w_{ang}(i)}{\sum_{j=1}^{k} w_{ang}(j)}.$$

This weighting is well defined as long as all k closest cameras are not equidistant. For a given camera i, $\tilde{w}_{ang}(i)$ is a smooth function as one varies the desired ray along a continuous proxy surface.

In addition to angular difference, we also wish to penalize cameras using metrics based on resolution and field-of-view. Using these various penalties, we define the combined penalty function as

$$\begin{aligned} \text{penalty}_{comb}(i) &= \alpha\ \text{penalty}_{ang}(i) + \beta\ \text{penalty}_{res}(i) \\ &+ \gamma\ \text{penalty}_{fov}(i) \end{aligned}$$

where the constants α, β, and γ control the relative importance of the different penalties. A constant can be set to zero to ignore a penalty. We can then define $\tilde{w}_{comb}(i)$ using the k-nearest neighbor interpolation strategy described above.

Resolution Penalty Given the projection matrices of the reference cameras, the proxy point p, and the normal at p, we can predict the degree of resolution mismatch by using the Jacobian of the planar homography relating the desired view to a reference camera. This calculation subsumes most factors resulting in resolution mismatches, including distance, surface obliqueness, focal length, and output resolution.

For efficiency, we approximate this computation by considering only the distances from the input cameras to the imaged point p. In addition, we generally are only concerned with source rays r_i that significantly *undersample* the observed proxy point p. Of course, oversampling can also lead to problems (e.g., aliasing), but proper use of mip-mapping can avoid the need to penalize images for oversampling. Thus, the simplified resolution penalty function that we use is

$$\text{penalty}_{res}(i) = \max(0, \|p - C_i\| - \|p - D\|),$$

where D is the center of the desired camera.

Field-of-View Penalty We do not want to use rays that fall outside the field-of-view of a source camera. We can include this consideration using the penalty function:

$$\text{penalty}_{fov}(i) = \begin{cases} 0 & \text{if } r_i \text{ within field-of-view} \\ \infty & \text{otherwise} \end{cases},$$

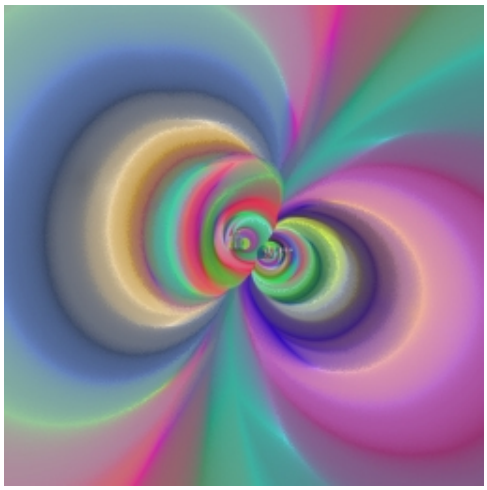


Figure 7: A visualized camera blending field. This example is from the "hallway" dataset described in the results section. The virtual camera is looking down the hallway.

which simply rejects all cameras that do not see the proxy point. In order to maintain continuity, we adjust this penalty function so that it smoothly increases toward ∞ as r_i approaches the border of image i.

With an accurate proxy, we would in fact compute visibility between p and C_i and only consider source rays that potentially see p as in [4]. In our setting we use proxies with unit depth complexity, so we have not needed to implement visibility computation. A visibility penalty function would assign ∞ to completely invisible points and small values to visible points. Care should be taken to smoothly transition from visible to invisible regions [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. The camera 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 evaluate the camera blending field at a sparse set of points in the image plane, triangulate these points, and interpolate the camera blending field over the rest of the image (see Figure 9). This approach assumes that the camera blending field is sufficiently smooth to be accurately recovered from the samples. The pseudocode for the algorithm and descriptions of the main procedures appear below:

```
Clear frame buffer to zero
Select camera blending field sample locations
Triangulate blending field samples
for each blending field sample location j do
   for each input image i do
      Evaluate blending weight i for sample location j
   end for
   Renormalize and store k closest weights at j
end for
for each input image i do
   Set current texture to texture i
   Set current texture matrix to matrix i
   Draw triangles with blending weights in alpha channel
end for
```

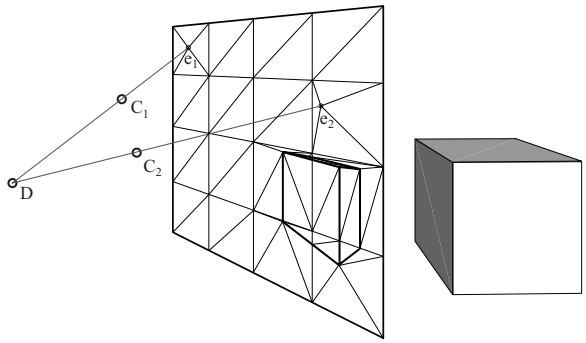


Figure 8: Our real-time renderer uses the projection of the proxy, the projection of the source camera centers, and a regular grid to triangulate the image plane.

Selecting Blending Field Samples We sample the camera blending field at a sparse set of locations in the image plane. These locations, which correspond to desired viewing rays, are chosen according to simple rules.

First, we project all of the vertices of the geometric proxy into the desired view and use these points as sample locations. To enhance epipole consistency, we next add a sample at the projection of every source camera in the desired view. Finally, we include a regular grid of samples on the desired image plane to obtain a sufficiently dense set of samples needed to capture the interesting spatial variation of the camera blending weights.

Triangulating Samples We next construct a *constrained* Delaunay triangulation of the blending field samples (see Figure 8).

First, we add the edges of the geometric proxy as constraints on the triangulation. This constraint prevents triangles from spanning two different surfaces on the proxy. Next, we add the edges of the regular grid as constraints on the triangulation. These constraints help keep the triangulation from flipping as the desired camera is moved.

Given this set of vertices and constraint edges, we create a constrained Delaunay triangulation of the image plane using Shewchuk's software [16]. The code automatically inserts new vertices at all edge-edge crossings.

Evaluating Blending Weights At each vertex of the triangulation, we compute and store the set of cameras with non-zero blending weights and their associated blending weights. Recall that at a vertex, these weights always sum to one.

Multiple sets of weights may need to be stored at each sample location if the sampling ray intersects the proxy multiple times. Triangles adjacent to these samples may need to be rendered multiple times on different proxy planes.

Drawing Triangles We render the desired image as a set of projectively mapped triangles as follows. Suppose that there are a total of m unique cameras ($k \leq m \leq 3k$, where k is the neighborhood size) 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 to projectively texture the source camera's data onto the rendered proxy triangle. For sampling rays that intersect the proxy multiple times, the triangles associated with those samples are rendered once for each planar surface that they intersect, with the z-buffer resolving visibility.

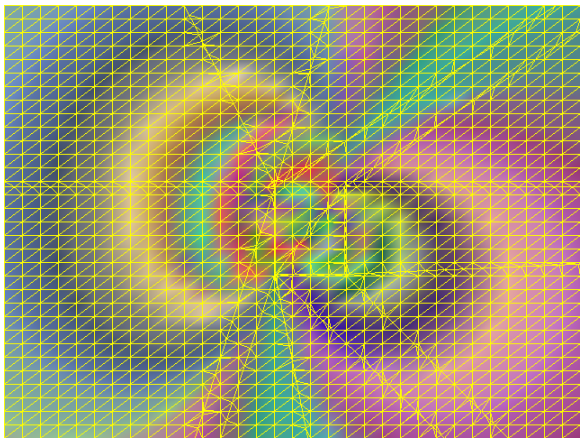


Figure 9: A visualized sampled color blending field from the real-time renderer. Camera weights are computed at each vertex of the triangulation. The sampling grid is 32×32 samples.

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 the real-time ULR algorithm. In all cases, the size k of the camera neighborhood is 4, $\alpha = 1$, $\beta = 0$, and $\gamma = 1$ unless stated otherwise. A 16×16 size grid is used for sampling the camera blending field.

Pond The pond dataset (Figure 11a) is constructed from a two second (60 frame) video sequence captured with a digital hand-held video camera. The camera is calibrated to recover the focal length and radial distortion parameters of the lens. The cameras' positions are recovered using structure-from-motion techniques.

In this simple example, 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 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 roughly 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 near the height at which 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 such as these. 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 added aspect: motion in a lumigraph. To create this "motion 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 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 phase of the period, 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 3D digitizing 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 geometric 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.

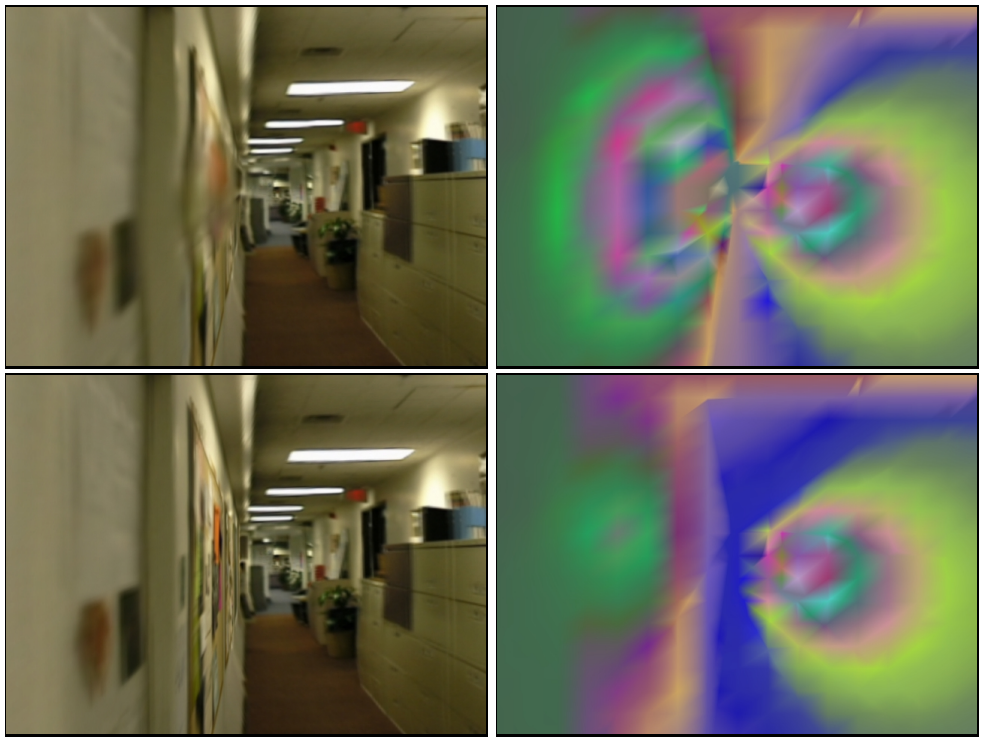
Hallway The Hallway dataset (Figure 11f) is constructed from a video sequence in which the camera moves forward into the scene. The camera is mounted on an instrumented robot that records its position as it moves. This forward camera motion is not handled well by previous image-based rendering techniques, but it is processed by the ULR 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 accommodation (β, which depends on the global scene scale, was 0.05). 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 and Future Work

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. The technique generalizes lumigraph and VDTM rendering algorithms. The real-time implementation has all the benefits of structured lumigraph rendering, including



(a) (b)

Figure 10: Operation of the ULR 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. Beside each image is the corresponding sampled camera blending field.

speed and photorealistic quality, while allowing for the use of geometric proxies, unstructured input cameras, and variations in resolution and field-of-view.

Many of our choices for blending functions and penalty functions are motivated by the desire for real-time rendering. More work needs to be done to determine the best possible functions for these tasks. In particular, a more sophisticated resolution penalty function is needed, as well as a more principled way to combine multiple, disparate penalties.

Further, nothing prevents our current implementation from sampling the blending field non-regularly. An interesting optimization would be to adaptively sample the blending field to better capture subtle variations and to eliminate visible grid artifacts.

Finally, not all the desired properties are created equal. It is clear that some are more important than others (e.g., equivalent ray consistency seems less important), and it would be useful to quantify these relationships for use in future algorithms.

References

[1] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. *SIGGRAPH 00*, pages 307–318.

[2] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. *1997 Symposium on Interactive 3D Graphics*, pages 25–34.

[3] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs. *SIGGRAPH 96*, pages 11–20.

[4] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. *Eurographics Rendering Workshop 1998*.

[5] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *SIGGRAPH 96*, pages 43–54.

[6] P. Heckbert and H. Moreton. Interpolation for polygon texture mapping and shading. *State of the Art in Computer Graphics: Visualization and Modeling*, 1991.

[7] B. Heigl, R. Koch, M. Pollefeys, J. Denzler, and L. Van Gool. Plenoptic modeling and rendering from image sequences taken by hand-held camera. *Proc. DAGM 99*, pages 94–101.

[8] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. *SIGGRAPH 97*, pages 225–232.

[9] A Isaksen, L. McMillan, and S. Gortler. Dynamically reparameterized light fields. *SIGGRAPH '00*, pages 297–306.

[10] M. Levoy and P. Hanrahan. Light field rendering. *SIGGRAPH 96*, pages 31–42.

[11] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. *SIGGRAPH 2000*, pages 131–144.

[12] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. *SIGGRAPH 98*, pages 75–84.

[13] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. *Eurographics Rendering Workshop 1997*, pages 23–34.

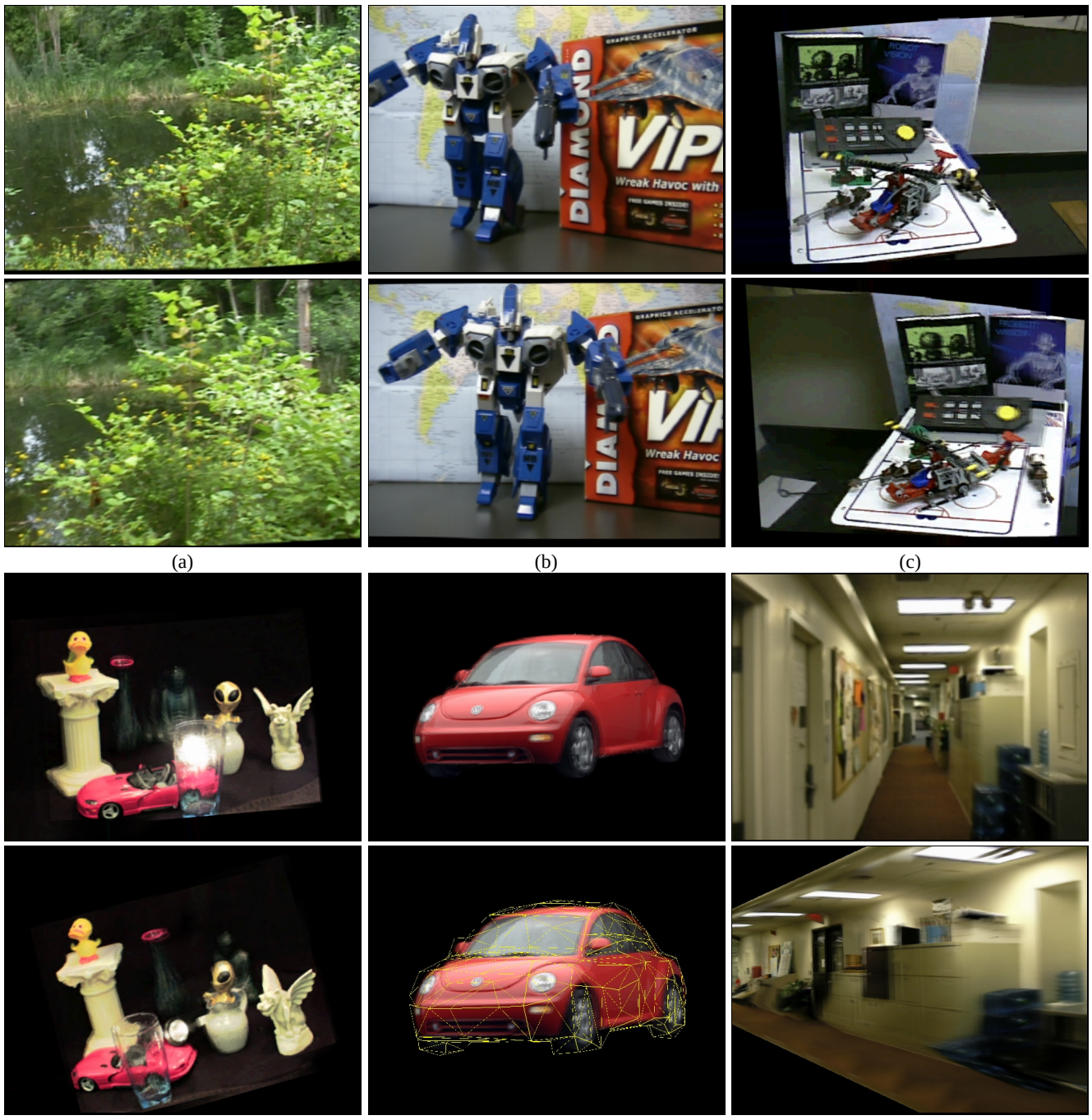
[14] Ramesh Raskar, Michael S. Brown, Ruigang Yang, Wei-Chao Chen, Greg Welch, Herman Towles, Brent Seales, and Henry Fuchs. Multi-projector displays using camera-based registration. *IEEE Visualization '99*, pages 161–168.

[15] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. *SIGGRAPH 2000*, pages 327–334.

[16] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. *First Workshop on Applied Computational Geometry*, pages 124–133, 1996.

[17] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. *SIGGRAPH 99*, pages 299–306.

[18] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. *SIGGRAPH 2000*, pages 287–296.



(a) (b) (c)

(d) (e) (f)

Figure 11: Renderings from the real-time unstructured lumigraph renderer. (a) and (b) show two virtual views of 60-image lumigraphs taken with a hand-held video camera. (c) shows two virtual views from a 1000-image moving lumigraph. (d) shows two virtual views of a 200-image lumigraph taken with a tracked camera. Note the active light source in the scene. (e) shows a 36-image lumigraph and its associated geometric proxy. (Original car images copyright © eVox Productions. Used with permission.) (f) shows two virtual views of a 200-image lumigraph. One virtual view is looking down the hallway, much like the input images, and one view is outside the hallway.