

Chapter 13

Image-based rendering

13.1	View interpolation	545
13.1.1	View-dependent texture maps	547
13.1.2	<i>Application: Photo Tourism</i>	548
13.2	Layered depth images	549
13.2.1	Impostors, sprites, and layers	549
13.3	Light fields and Lumigraphs	551
13.3.1	Unstructured Lumigraph	554
13.3.2	Surface light fields	555
13.3.3	<i>Application: Concentric mosaics</i>	556
13.4	Environment mattes	556
13.4.1	Higher-dimensional light fields	558
13.4.2	The modeling to rendering continuum	559
13.5	Video-based rendering	560
13.5.1	Video-based animation	560
13.5.2	Video textures	561
13.5.3	<i>Application: Animating pictures</i>	564
13.5.4	3D Video	564
13.5.5	<i>Application: Video-based walkthroughs</i>	566
13.6	Additional reading	569
13.7	Exercises	570

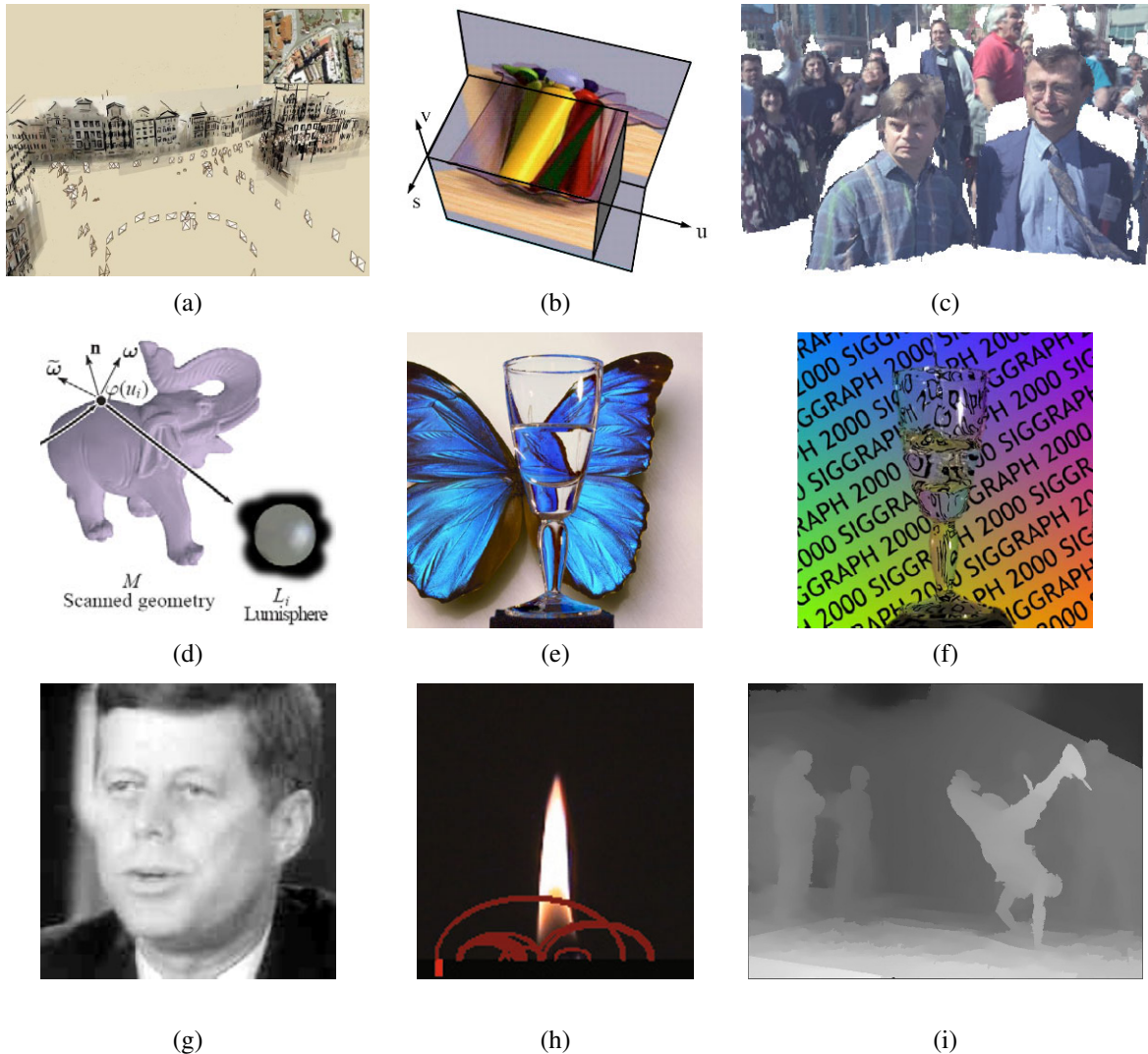


Figure 13.1 Image-based and video-based rendering: (a) a 3D view of a Photo Tourism reconstruction (Snavely, Seitz, and Szeliski 2006) © 2006 ACM; (b) a slice through a 4D light field (Gortler, Grzeszczuk, Szeliski *et al.* 1996) © 1996 ACM; (c) sprites with depth (Shade, Gortler, He *et al.* 1998) © 1998 ACM; (d) surface light field (Wood, Azuma, Aldinger *et al.* 2000) © 2000 ACM; (e) environment matte in front of a novel background (Zongker, Werner, Curless *et al.* 1999) © 1999 ACM; (f) real-time video environment matte (Chuang, Zongker, Hindorff *et al.* 2000) © 2000 ACM; (g) Video Rewrite used to re-animate old video (Bregler, Covell, and Slaney 1997) © 1997 ACM; (h) video texture of a candle flame (Schödl, Szeliski, Salesin *et al.* 2000) © 2000 ACM; (i) video view interpolation (Zitnick, Kang, Uyttendaele *et al.* 2004) © 2004 ACM.

Over the last two decades, image-based rendering has emerged as one of the most exciting applications of computer vision (Kang, Li, Tong *et al.* 2006; Shum, Chan, and Kang 2007). In image-based rendering, 3D reconstruction techniques from computer vision are combined with computer graphics rendering techniques that use multiple views of a scene to create interactive photo-realistic experiences, such as the Photo Tourism system shown in Figure 13.1a. Commercial versions of such systems include immersive street-level navigation in on-line mapping systems¹ and the creation of 3D Photosynths² from large collections of casually acquired photographs.

In this chapter, we explore a variety of image-based rendering techniques, such as those illustrated in Figure 13.1. We begin with *view interpolation* (Section 13.1), which creates a seamless transition between a pair of reference images using one or more pre-computed depth maps. Closely related to this idea are *view-dependent texture maps* (Section 13.1.1), which blend multiple texture maps on a 3D model's surface. The representations used for both the color imagery and the 3D geometry in view interpolation include a number of clever variants such as *layered depth images* (Section 13.2) and *sprites with depth* (Section 13.2.1).

We continue our exploration of image-based rendering with the *light field* and *Lumigraph* four-dimensional representations of a scene's appearance (Section 13.3), which can be used to render the scene from any arbitrary viewpoint. Variants on these representations include the *unstructured Lumigraph* (Section 13.3.1), *surface light fields* (Section 13.3.2), *concentric mosaics* (Section 13.3.3), and *environment mattes* (Section 13.4).

The last part of this chapter explores the topic of *video-based rendering*, which uses one or more videos in order to create novel video-based experiences (Section 13.5). The topics we cover include video-based facial animation (Section 13.5.1), as well as *video textures* (Section 13.5.2), in which short video clips can be seamlessly looped to create dynamic real-time video-based renderings of a scene. We close with a discussion of *3D videos* created from multiple video streams (Section 13.5.4), as well as *video-based walkthroughs* of environments (Section 13.5.5), which have found widespread application in immersive outdoor mapping and driving direction systems.

13.1 View interpolation

While the term *image-based rendering* first appeared in the papers by Chen (1995) and McMillan and Bishop (1995), the work on *view interpolation* by Chen and Williams (1993) is considered as the seminal paper in the field. In view interpolation, pairs of rendered color images are combined with their pre-computed depth maps to generate interpolated views that mimic what a virtual camera would see in between the two reference views.

View interpolation combines two ideas that were previously used in computer vision and computer graphics. The first is the idea of pairing a recovered depth map with the reference image used in its computation and then using the resulting texture-mapped 3D model to generate novel views (Figure 11.1). The second is the idea of *morphing* (Section 3.6.3) (Figure 3.53), where correspondences between pairs of images are used to warp each reference image to an in-between location while simultaneously cross-dissolving between the two warped images.

¹ <http://maps.bing.com> and <http://maps.google.com>.

² <http://photosynth.net>.

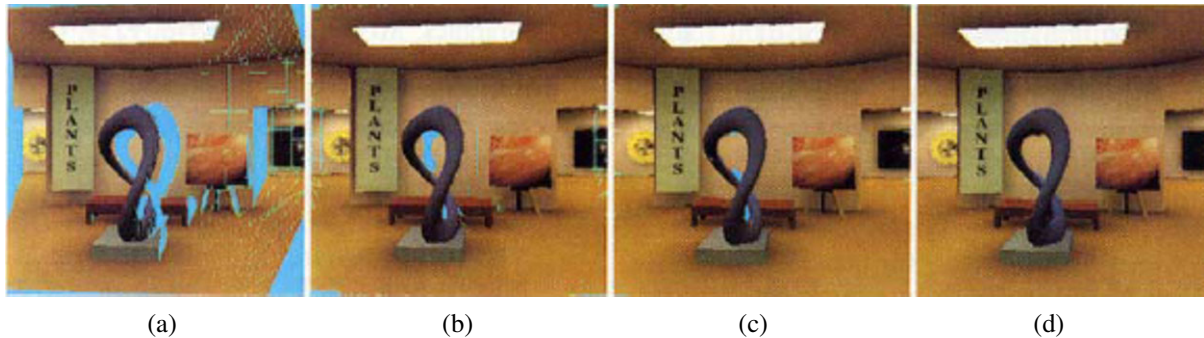


Figure 13.2 View interpolation (Chen and Williams 1993) © 1993 ACM: (a) holes from one source image (shown in blue); (b) holes after combining two widely spaced images; (c) holes after combining two closely spaced images; (d) after interpolation (hole filling).

Figure 13.2 illustrates this process in more detail. First, both source images are warped to the novel view, using both the knowledge of the reference and virtual 3D camera pose along with each image’s depth map (2.68–2.70). In the paper by Chen and Williams (1993), a *forward warping* algorithm (Algorithm 3.1 and Figure 3.46) is used. The depth maps are represented as quadtrees for both space and rendering time efficiency (Samet 1989).

During the forward warping process, multiple pixels (which occlude one another) may land on the same destination pixel. To resolve this conflict, either a *z-buffer* depth value can be associated with each destination pixel or the images can be warped in back-to-front order, which can be computed based on the knowledge of epipolar geometry (Chen and Williams 1993; Laveau and Faugeras 1994; McMillan and Bishop 1995).

Once the two reference images have been warped to the novel view (Figure 13.2a–b), they can be merged to create a coherent composite (Figure 13.2c). Whenever one of the images has a *hole* (illustrated as a cyan pixel), the other image is used as the final value. When both images have pixels to contribute, these can be blended as in usual morphing, i.e., according to the relative distances between the virtual and source cameras. Note that if the two images have very different exposures, which can happen when performing view interpolation on real images, the hole-filled regions and the blended regions will have different exposures, leading to subtle artifacts.

The final step in view interpolation (Figure 13.2d) is to fill any remaining holes or cracks due to the forward warping process or lack of source data (scene visibility). This can be done by copying pixels from the *further* pixels adjacent to the hole. (Otherwise, foreground objects are subject to a “fattening effect”.)

The above process works well for rigid scenes, although its visual quality (lack of aliasing) can be improved using a two-pass, forward–backward algorithm (Section 13.2.1) (Shade, Gortler, He *et al.* 1998) or full 3D rendering (Zitnick, Kang, Uyttendaele *et al.* 2004). In the case where the two reference images are views of a non-rigid scene, e.g., a person smiling in one image and frowning in the other, *view morphing*, which combines ideas from view interpolation with regular morphing, can be used (Seitz and Dyer 1996).

While the original view interpolation paper describes how to generate novel views based on similar pre-computed (linear perspective) images, the *plenoptic modeling* paper of McMillan and Bishop (1995) argues that cylindrical images should be used to store the pre-computed

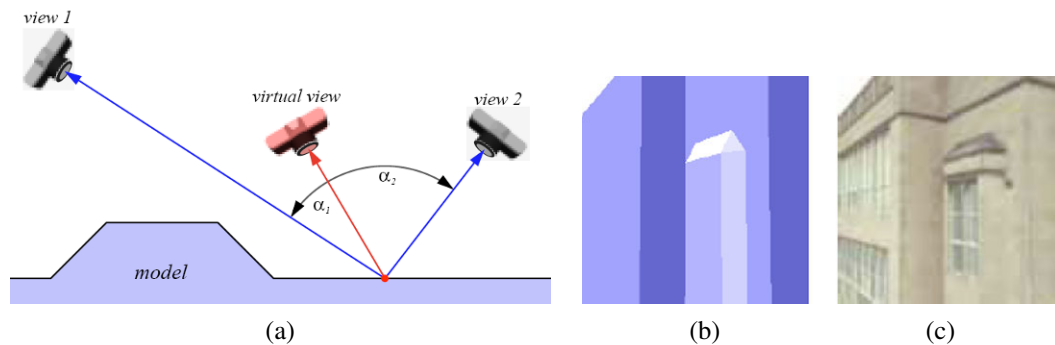


Figure 13.3 View-dependent texture mapping (Debevec, Taylor, and Malik 1996) © 1996 ACM. (a) The weighting given to each input view depends on the relative angles between the novel (virtual) view and the original views; (b) simplified 3D model geometry; (c) with view-dependent texture mapping, the geometry appears to have more detail (recessed windows).

rendering or real-world images. (Chen 1995) also propose using environment maps (cylindrical, cubic, or spherical) as source images for view interpolation.

13.1.1 View-dependent texture maps

View-dependent texture maps (Debevec, Taylor, and Malik 1996) are closely related to view interpolation. Instead of associating a separate depth map with each input image, a single 3D model is created for the scene, but different images are used as texture map sources depending on the virtual camera's current position (Figure 13.3a).³

In more detail, given a new virtual camera position, the similarity of this camera's view of each polygon (or pixel) is compared to that of potential source images. The images are then blended using a weighting that is inversely proportional to the angles α_i between the virtual view and the source views (Figure 13.3a). Even though the geometric model can be fairly coarse (Figure 13.3b), blending between different views gives a strong sense of more detailed geometry because of the parallax (visual motion) between corresponding pixels. While the original paper performs the weighted blend computation separately at each pixel or coarsened polygon face, follow-on work by Debevec, Yu, and Borshukov (1998) presents a more efficient implementation based on precomputing contributions for various portions of viewing space and then using projective texture mapping (OpenGL-ARB 1997).

The idea of view-dependent texture mapping has been used in a large number of subsequent image-based rendering systems, including facial modeling and animation (Pighin, Hecker, Lischinski *et al.* 1998) and 3D scanning and visualization (Pulli, Abi-Rached, Duchamp *et al.* 1998). Closely related to view-dependent texture mapping is the idea of blending between light rays in 4D space, which forms the basis of the Lumigraph and unstructured Lumigraph systems (Section 13.3) (Gortler, Grzeszczuk, Szeliski *et al.* 1996; Buehler, Bosse, McMillan *et al.* 2001).

³ The term *image-based modeling*, which is now commonly used to describe the creation of texture-mapped 3D models from multiple images, appears to have first been used by Debevec, Taylor, and Malik (1996), who also used the term *photogrammetric modeling* to describe the same process.

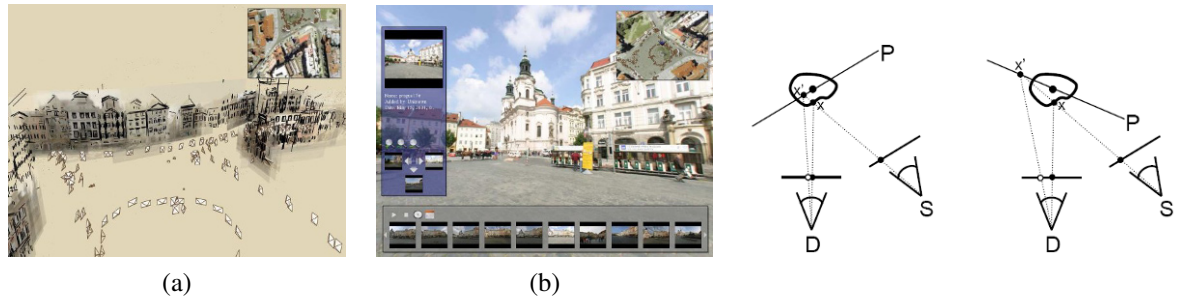


Figure 13.4 Photo Tourism (Snavely, Seitz, and Szeliski 2006): © 2006 ACM: (a) a 3D overview of the scene, with translucent washes and lines painted onto the planar impostors; (b) once the user has selected a region of interest, a set of related thumbnails is displayed along the bottom; (c) planar proxy selection for optimal stabilization (Snavely, Garg, Seitz *et al.* 2008) © 2008 ACM.

In order to provide even more realism in their Façade system, Debevec, Taylor, and Malik (1996) also include a *model-based stereo* component, which optionally computes an offset (parallax) map for each coarse planar facet of their 3D model. They call the resulting analysis and rendering system a *hybrid geometry- and image-based* approach, since it uses traditional 3D geometric modeling to create the global 3D model, but then uses local depth offsets, along with view interpolation, to add visual realism.

13.1.2 Application: Photo Tourism

While view interpolation was originally developed to accelerate the rendering of 3D scenes on low-powered processors and systems without graphics acceleration, it turns out that it can be applied directly to large collections of casually acquired photographs. The *Photo Tourism* system developed by Snavely, Seitz, and Szeliski (2006) uses structure from motion to compute the 3D locations and poses of all the cameras taking the images, along with a sparse 3D point-cloud model of the scene (Section 7.4.4, Figure 7.11).

To perform an image-based exploration of the resulting *sea of images* (Aliaga, Funkhouser, Yanovsky *et al.* 2003), Photo Tourism first associates a 3D proxy with each image. While a triangulated mesh obtained from the point cloud can sometimes form a suitable proxy, e.g., for outdoor terrain models, a simple dominant plane fit to the 3D points visible in each image often performs better, because it does not contain any erroneous segments or connections that pop out as artifacts. As automated 3D modeling techniques continue to improve, however, the pendulum may swing back to more detailed 3D geometry (Goesele, Snavely, Curless *et al.* 2007; Sinha, Steedly, and Szeliski 2009).

The resulting image-based navigation system lets users move from photo to photo, either by selecting cameras from a top-down view of the scene (Figure 13.4a) or by selecting regions of interest in an image, navigating to nearby views, or selecting related thumbnails (Figure 13.4b). To create a background for the 3D scene, e.g., when being viewed from above, non-photorealistic techniques (Section 10.5.2), such as translucent color washes or highlighted 3D line segments, can be used (Figure 13.4a). The system can also be used to annotate regions of images and to automatically propagate such annotations to other photographs.

The 3D planar proxies used in Photo Tourism and the related Photosynth system from Microsoft result in non-photorealistic transitions reminiscent of visual effects such as “page flips”. Selecting a stable 3D axis for all the planes can reduce the amount of swimming and enhance the perception of 3D (Figure 13.4c) (Snavely, Garg, Seitz *et al.* 2008). It is also possible to automatically detect objects in the scene that are seen from multiple views and create “orbits” of viewpoints around such objects. Furthermore, nearby images in both 3D position and viewing direction can be linked to create “virtual paths”, which can then be used to navigate between arbitrary pairs of images, such as those you might take yourself while walking around a popular tourist site (Snavely, Garg, Seitz *et al.* 2008).

The spatial matching of image features and regions performed by Photo Tourism can also be used to infer more information from large image collections. For example, Simon, Snavely, and Seitz (2007) show how the match graph between images of popular tourist sites can be used to find the most *iconic* (commonly photographed) objects in the collection, along with their related tags. In follow-on work, Simon and Seitz (2008) show how such tags can be propagated to sub-regions of each image, using an analysis of which 3D points appear in the central portions of photographs. Extensions of these techniques to *all* of the world’s images, including the use of GPS tags where available, have been investigated as well (Li, Wu, Zach *et al.* 2008; Quack, Leibe, and Van Gool 2008; Crandall, Backstrom, Huttenlocher *et al.* 2009; Li, Crandall, and Huttenlocher 2009; Zheng, Zhao, Song *et al.* 2009).

13.2 Layered depth images

Traditional view interpolation techniques associate a single depth map with each source or reference image. Unfortunately, when such a depth map is warped to a novel view, holes and cracks inevitably appear behind the foreground objects. One way to alleviate this problem is to keep several depth and color values (*depth pixels*) at every pixel in a reference image (or, at least for pixels near foreground–background transitions) (Figure 13.5). The resulting data structure, which is called a *layered depth image* (LDI), can be used to render new views using a back-to-front forward warping (splatting) algorithm (Shade, Gortler, He *et al.* 1998).

13.2.1 Impostors, sprites, and layers

An alternative to keeping lists of color-depth values at each pixel, as is done in the LDI, is to organize objects into different *layers* or *sprites*. The term *sprite* originates in the computer game industry, where it is used to designate flat animated characters in games such as Pac-Man or Mario Bros. When put into a 3D setting, such objects are often called *impostors*, because they use a piece of flat, alpha-matted geometry to represent simplified versions of 3D objects that are far away from the camera (Shade, Lischinski, Salesin *et al.* 1996; Lengyel and Snyder 1997; Torborg and Kajiya 1996). In computer vision, such representations are usually called *layers* (Wang and Adelson 1994; Baker, Szeliski, and Anandan 1998; Torr, Szeliski, and Anandan 1999; Birchfield, Natarajan, and Tomasi 2007). Section 8.5.2 discusses the topics of transparent layers and reflections, which occur on specular and transparent surfaces such as glass.

While flat layers can often serve as an adequate representation of geometry and appearance for far-away objects, better geometric fidelity can be achieved by also modeling the

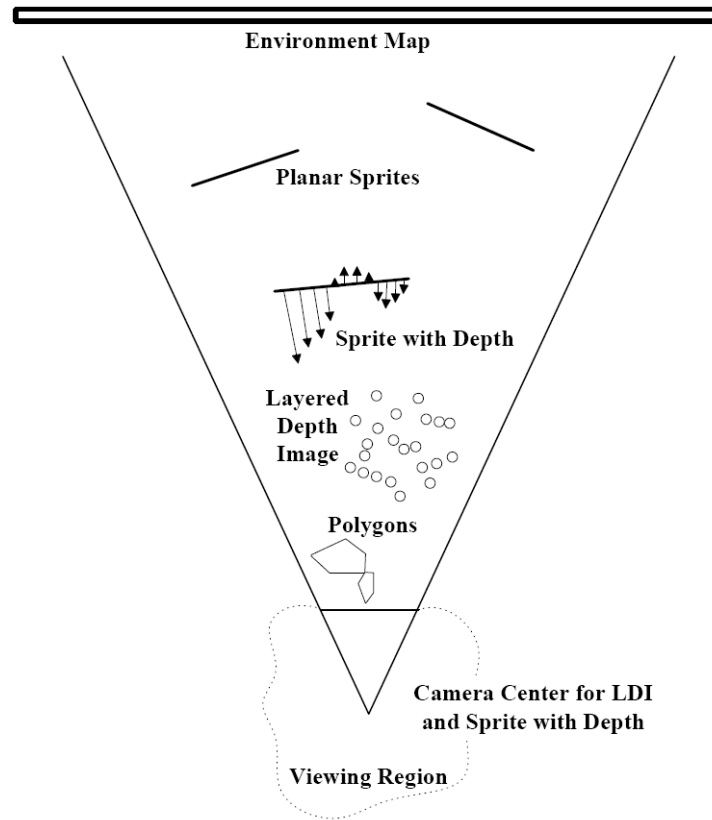


Figure 13.5 A variety of image-based rendering primitives, which can be used depending on the distance between the camera and the object of interest (Shade, Gortler, He *et al.* 1998) © 1998 ACM. Closer objects may require more detailed polygonal representations, while mid-level objects can use a layered depth image (LDI), and far-away objects can use sprites (potentially with depth) and environment maps.



Figure 13.6 Sprites with depth (Shade, Gortler, He *et al.* 1998) © 1998 ACM: (a) alpha-matted color sprite; (b) corresponding relative depth or parallax; (c) rendering without relative depth; (d) rendering with depth (note the curved object boundaries).

per-pixel offsets relative to a base plane, as shown in Figures 13.5 and 13.6a–b. Such representations are called *plane plus parallax* in the computer vision literature (Kumar, Anandan, and Hanna 1994; Sawhney 1994; Szeliski and Coughlan 1997; Baker, Szeliski, and Anandan 1998), as discussed in Section 8.5 (Figure 8.16). In addition to fully automated stereo techniques, it is also possible to paint in depth layers (Kang 1998; Oh, Chen, Dorsey *et al.* 2001; Shum, Sun, Yamazaki *et al.* 2004) or to infer their 3D structure from monocular image cues (Section 14.4.4) (Hoiem, Efros, and Hebert 2005b; Saxena, Sun, and Ng 2009).

How can we render a sprite with depth from a novel viewpoint? One possibility, as with a regular depth map, is to just forward warp each pixel to its new location, which can cause aliasing and cracks. A better way, which we already mentioned in Section 3.6.2, is to first warp the depth (or (u, v) displacement) map to the novel view, fill in the cracks, and then use higher-quality inverse warping to resample the color image (Shade, Gortler, He *et al.* 1998). Figure 13.6d shows the results of applying such a two-pass rendering algorithm. From this still image, you can appreciate that the foreground sprites look more rounded; however, to fully appreciate the improvement in realism, you would have to look at the actual animated sequence.

Sprites with depth can also be rendered using conventional graphics hardware, as described in (Zitnick, Kang, Uyttendaele *et al.* 2004). Rogmans, Lu, Bekaert *et al.* (2009) describe GPU implementations of both real-time stereo matching and real-time forward and inverse rendering algorithms.

13.3 Light fields and Lumigraphs

While image-based rendering approaches can synthesize scene renderings from novel viewpoints, they raise the following more general question:

Is it possible to capture and render the appearance of a scene from all possible viewpoints and, if so, what is the complexity of the resulting structure?

Let us assume that we are looking at a static scene, i.e., one where the objects and illuminants are fixed, and only the observer is moving around. Under these conditions, we can describe each image by the location and orientation of the virtual camera (6 dof) as well as its intrinsics (e.g., its focal length). However, if we capture a two-dimensional *spherical* image around each possible camera location, we can re-render any view from this information.⁴ Thus, taking the cross-product of the three-dimensional space of camera positions with the 2D space of spherical images, we obtain the 5D *plenoptic function* of Adelson and Bergen (1991), which forms the basis of the image-based rendering system of McMillan and Bishop (1995).

Notice, however, that when there is no light dispersion in the scene, i.e., no smoke or fog, all the coincident rays along a portion of free space (between solid or refractive objects) have the same color value. Under these conditions, we can reduce the 5D plenoptic function to the 4D *light field* of all possible rays (Gortler, Grzeszczuk, Szeliski *et al.* 1996; Levoy and Hanrahan 1996; Levoy 2006).⁵

⁴ Since we are counting dimensions, we ignore for now any sampling or resolution issues.

⁵ Levoy and Hanrahan (1996) borrowed the term *light field* from a paper by Gershun (1939). Another name for this representation is the *photic field* (Moon and Spencer 1981).

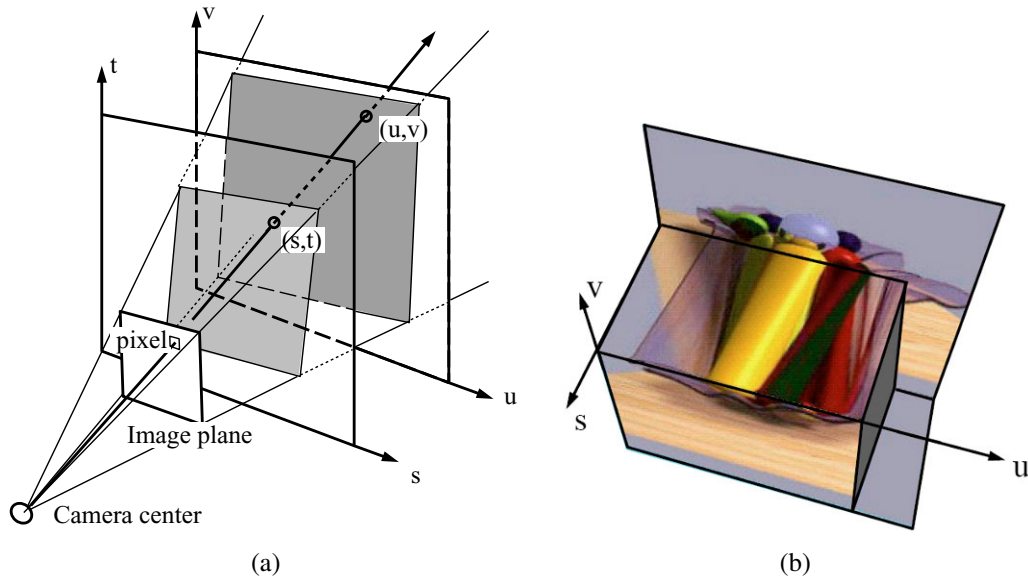


Figure 13.7 The Lumigraph (Gortler, Grzeszczuk, Szeliski *et al.* 1996) © 1996 ACM: (a) a ray is represented by its 4D two-plane parameters (s, t) and (u, v) ; (b) a slice through the 3D light field subset (u, v, s) .

To make the parameterization of this 4D function simpler, let us put two planes in the 3D scene roughly bounding the area of interest, as shown in Figure 13.7a. Any light ray terminating at a camera that lives in front of the st plane (assuming that this space is empty) passes through the two planes at (s, t) and (u, v) and can be described by its 4D coordinate (s, t, u, v) . This diagram (and parameterization) can be interpreted as describing a family of cameras living on the st plane with their image planes being the uv plane. The uv plane can be placed at infinity, which corresponds to all the virtual cameras looking in the same direction.

In practice, if the planes are of finite extent, the finite *light slab* $L(s, t, u, v)$ can be used to generate any synthetic view that a camera would see through a (finite) *viewport* in the st plane with a view frustum that wholly intersects the far uv plane. To enable the camera to move all the way around an object, the 3D space surrounding the object can be split into multiple domains, each with its own light slab parameterization. Conversely, if the camera is moving inside a bounded volume of free space looking outward, multiple cube faces surrounding the camera can be used as (s, t) planes.

Thinking about 4D spaces is difficult, so let us drop our visualization by one dimension. If we fix the row value t and constrain our camera to move along the s axis while looking at the uv plane, we can stack all of the stabilized images the camera sees to get the (u, v, s) *epipolar volume*, which we discussed in Section 11.6. A “horizontal” cross-section through this volume is the well-known *epipolar plane image* (Bolles, Baker, and Marimont 1987), which is the us slice shown in Figure 13.7b.

As you can see in this slice, each color pixel moves along a linear track whose slope is related to its depth (parallax) from the uv plane. (Pixels exactly on the uv plane appear “vertical”, i.e., they do not move as the camera moves along s .) Furthermore, pixel tracks

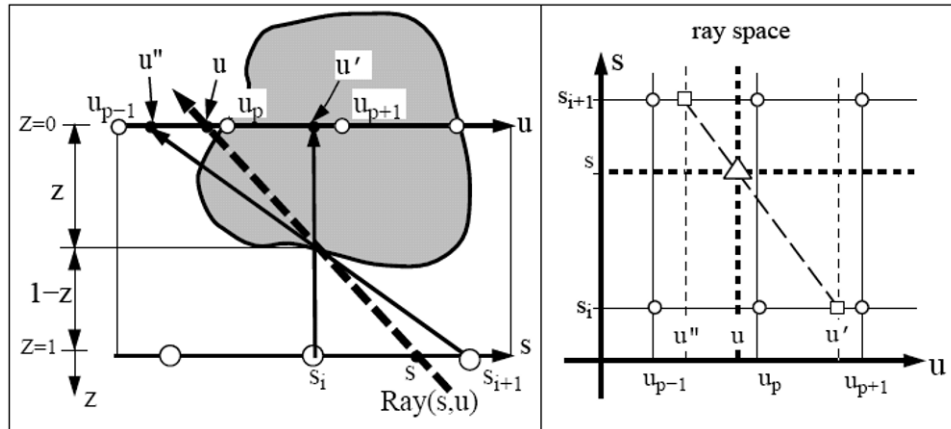


Figure 13.8 Depth compensation in the Lumigraph (Gortler, Grzeszczuk, Szeliski *et al.* 1996) © 1996 ACM. To resample the (s, u) dashed light ray, the u parameter corresponding to each discrete s_i camera location is modified according to the out-of-plane depth z to yield new coordinates u and u' ; in (u, s) ray space, the original sample (\triangle) is resampled from the (s_i, u') and (s_{i+1}, u'') samples, which are themselves linear blends of their adjacent (o) samples.

occlude one another as their corresponding 3D surface elements occlude. Translucent pixels, however, composite *over* background pixels (Section 3.1.3, (3.8)) rather than occluding them. Thus, we can think of adjacent pixels sharing a similar planar geometry as *EPI strips* or *EPI tubes* (Criminisi, Kang, Swaminathan *et al.* 2005).

The equations mapping from pixels (x, y) in a virtual camera and the corresponding (s, t, u, v) coordinates are relatively straightforward to derive and are sketched out in Exercise 13.7. It is also possible to show that the set of pixels corresponding to a regular orthographic or perspective camera, i.e., one that has a linear projective relationship between 3D points and (x, y) pixels (2.63), lie along a two-dimensional hyperplane in the (s, t, u, v) light field (Exercise 13.7).

While a light field can be used to render a complex 3D scene from novel viewpoints, a much better rendering (with less ghosting) can be obtained if something is known about its 3D geometry. The Lumigraph system of Gortler, Grzeszczuk, Szeliski *et al.* (1996) extends the basic light field rendering approach by taking into account the 3D location of surface points corresponding to each 3D ray.

Consider the ray (s, u) corresponding to the dashed line in Figure 13.8, which intersects the object's surface at a distance z from the uv plane. When we look up the pixel's color in camera s_i (assuming that the light field is discretely sampled on a regular 4D (s, t, u, v) grid), the actual pixel coordinate is u' , instead of the original u value specified by the (s, u) ray. Similarly, for camera s_{i+1} (where $s_i \leq s \leq s_{i+1}$), pixel address u'' is used. Thus, instead of using quadri-linear interpolation of the nearest sampled (s, t, u, v) values around a given ray to determine its color, the (u, v) values are modified for each discrete (s_i, t_i) camera.

Figure 13.8 also shows the same reasoning in *ray space*. Here, the original continuous-valued (s, u) ray is represented by a triangle and the nearby sampled discrete values are shown as circles. Instead of just blending the four nearest samples, as would be indicated

by the vertical and horizontal dashed lines, the modified (s_i, u') and (s_{i+1}, u'') values are sampled instead and their values are then blended.

The resulting rendering system produces images of much better quality than a proxy-free light field and is the method of choice whenever 3D geometry can be inferred. In subsequent work, Isaksen, McMillan, and Gortler (2000) show how a planar proxy for the scene, which is a simpler 3D model, can be used to simplify the resampling equations. They also describe how to create synthetic aperture photos, which mimic what might be seen by a wide-aperture lens, by blending more nearby samples (Levoy and Hanrahan 1996). A similar approach can be used to re-focus images taken with a plenoptic (microlens array) camera (Ng, Levoy, Brédif *et al.* 2005; Ng 2005) or a light field microscope (Levoy, Ng, Adams *et al.* 2006). It can also be used to see through obstacles, using extremely large synthetic apertures focused on a background that can blur out foreground objects and make them appear translucent (Wilburn, Joshi, Vaish *et al.* 2005; Vaish, Szeliski, Zitnick *et al.* 2006).

Now that we understand how to render new images from a light field, how do we go about capturing such data sets? One answer is to move a calibrated camera with a motion control rig or gantry.⁶ Another approach is to take handheld photographs and to determine the pose and intrinsic calibration of each image using either a calibrated stage or structure from motion. In this case, the images need to be *rebinned* into a regular 4D (s, t, u, v) space before they can be used for rendering (Gortler, Grzeszczuk, Szeliski *et al.* 1996). Alternatively, the original images can be used directly using a process called the *unstructured Lumigraph*, which we describe below.

Because of the large number of images involved, light fields and Lumigraphs can be quite voluminous to store and transmit. Fortunately, as you can tell from Figure 13.7b, there is a tremendous amount of redundancy (coherence) in a light field, which can be made even more explicit by first computing a 3D model, as in the Lumigraph. A number of techniques have been developed to compress and progressively transmit such representations (Gortler, Grzeszczuk, Szeliski *et al.* 1996; Levoy and Hanrahan 1996; Rademacher and Bishop 1998; Magnor and Girod 2000; Wood, Azuma, Aldinger *et al.* 2000; Shum, Kang, and Chan 2003; Magnor, Ramanathan, and Girod 2003; Shum, Chan, and Kang 2007).

13.3.1 Unstructured Lumigraph

When the images in a Lumigraph are acquired in an unstructured (irregular) manner, it can be counterproductive to resample the resulting light rays into a regularly binned (s, t, u, v) data structure. This is both because resampling always introduces a certain amount of aliasing and because the resulting gridded light field can be populated very sparsely or irregularly.

The alternative is to render directly from the acquired images, by finding for each light ray in a virtual camera the closest pixels in the original images. The *unstructured Lumigraph* rendering (ULR) system of Buehler, Bosse, McMillan *et al.* (2001) describes how to select such pixels by combining a number of fidelity criteria, including *epipole consistency* (distance of rays to a source camera's center), *angular deviation* (similar incidence direction on the surface), *resolution* (similar sampling density along the surface), *continuity* (to nearby pixels), and *consistency* (along the ray). These criteria can all be combined to determine a weighting

⁶ See <http://lightfield.stanford.edu/acq.html> for a description of some of the gantries and camera arrays built at the Stanford Computer Graphics Laboratory. This Web site also provides a number of light field data sets that are a great source of research and project material.

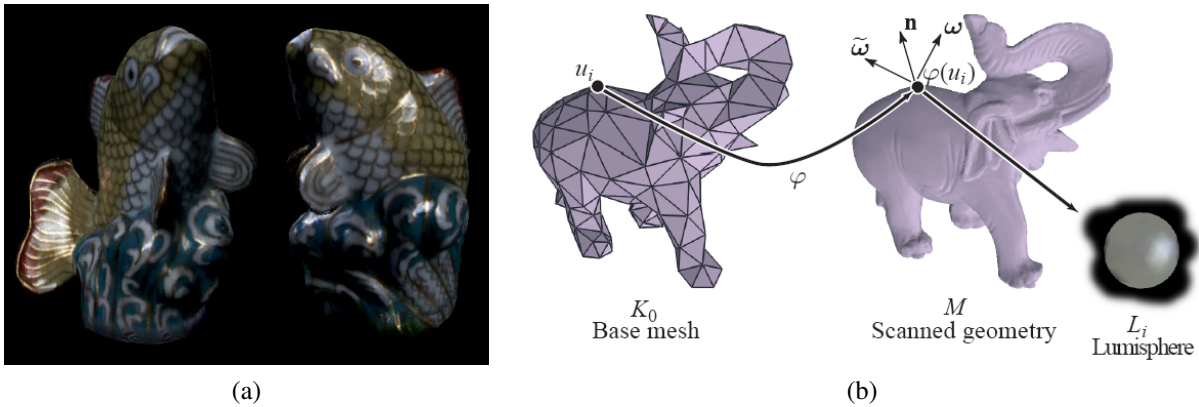


Figure 13.9 Surface light fields (Wood, Azuma, Aldinger *et al.* 2000) © 2000 ACM: (a) example of a highly specular object with strong inter-reflections; (b) the surface light field stores the light emanating from each surface point in all visible directions as a “Lumisphere”.

function between each virtual camera’s pixel and a number of candidate input cameras from which it can draw colors. To make the algorithm more efficient, the computations are performed by discretizing the virtual camera’s image plane using a regular grid overlaid with the polyhedral object mesh model and the input camera centers of projection and interpolating the weighting functions between vertices.

The unstructured Lumigraph generalizes previous work in both image-based rendering and light field rendering. When the input cameras are gridded, the ULR behaves the same way as regular Lumigraph rendering. When fewer cameras are available but the geometry is accurate, the algorithm behaves similarly to view-dependent texture mapping (Section 13.1.1).

13.3.2 Surface light fields

Of course, using a two-plane parameterization for a light field is not the only possible choice. (It is the one usually presented first since the projection equations and visualizations are the easiest to draw and understand.) As we mentioned on the topic of light field compression, if we know the 3D shape of the object or scene whose light field is being modeled, we can effectively compress the field because nearby rays emanating from nearby surface elements have similar color values.

In fact, if the object is totally diffuse, ignoring occlusions, which can be handled using 3D graphics algorithms or z-buffering, all rays passing through a given surface point will have the same color value. Hence, the light field “collapses” to the usual 2D texture-map defined over an object’s surface. Conversely, if the surface is totally specular (e.g., mirrored), each surface point reflects a miniature copy of the environment surrounding that point. In the absence of inter-reflections (e.g., a convex object in a large open space), each surface point simply reflects the far-field *environment map* (Section 2.2.1), which again is two-dimensional. Therefore, it seems that re-parameterizing the 4D light field to lie on the object’s surface can be extremely beneficial.

These observations underlie the *surface light field* representation introduced by Wood, Azuma, Aldinger *et al.* (2000). In their system, an accurate 3D model is built of the object

being represented. Then the *Lumisphere* of all rays emanating from each surface point is estimated or captured (Figure 13.9). Nearby Lumispheres will be highly correlated and hence amenable to both compression and manipulation.

To estimate the diffuse component of each Lumisphere, a median filtering over all visible exiting directions is first performed for each channel. Once this has been subtracted from the Lumisphere, the remaining values, which should consist mostly of the specular components, are *reflected* around the local surface normal (2.89), which turns each Lumisphere into a copy of the local environment around that point. Nearby Lumispheres can then be compressed using predictive coding, vector quantization, or principal component analysis.

The decomposition into a diffuse and specular component can also be used to perform editing or manipulation operations, such as re-painting the surface, changing the specular component of the reflection (e.g., by blurring or sharpening the specular Lumispheres), or even geometrically deforming the object while preserving detailed surface appearance.

13.3.3 Application: Concentric mosaics

A useful and simple version of light field rendering is a panoramic image with parallax, i.e., a video or series of photographs taken from a camera swinging in front of some rotation point. Such panoramas can be captured by placing a camera on a boom on a tripod, or even more simply, by holding a camera at arm's length while rotating your body around a fixed axis.

The resulting set of images can be thought of as a *concentric mosaic* (Shum and He 1999; Shum, Wang, Chai *et al.* 2002) or a *layered depth panorama* (Zheng, Kang, Cohen *et al.* 2007). The term “concentric mosaic” comes from a particular structure that can be used to re-bin all of the sampled rays, essentially associating each column of pixels with the “radius” of the concentric circle to which it is tangent (Shum and He 1999; Peleg, Ben-Ezra, and Pritch 2001).

Rendering from such data structures is fast and straightforward. If we assume that the scene is far enough away, for any virtual camera location, we can associate each column of pixels in the virtual camera with the nearest column of pixels in the input image set. (For a regularly captured set of images, this computation can be performed analytically.) If we have some rough knowledge of the depth of such pixels, columns can be stretched vertically to compensate for the change in depth between the two cameras. If we have an even more detailed depth map (Peleg, Ben-Ezra, and Pritch 2001; Li, Shum, Tang *et al.* 2004; Zheng, Kang, Cohen *et al.* 2007), we can perform pixel-by-pixel depth corrections.

While the virtual camera's motion is constrained to lie in the plane of the original cameras and within the radius of the original capture ring, the resulting experience can exhibit complex rendering phenomena, such as reflections and translucencies, which cannot be captured using a texture-mapped 3D model of the world. Exercise 13.10 has you construct a concentric mosaic rendering system from a series of hand-held photos or video.

13.4 Environment mattes

So far in this chapter, we have dealt with view interpolation and light fields, which are techniques for modeling and rendering complex static scenes seen from different viewpoints.

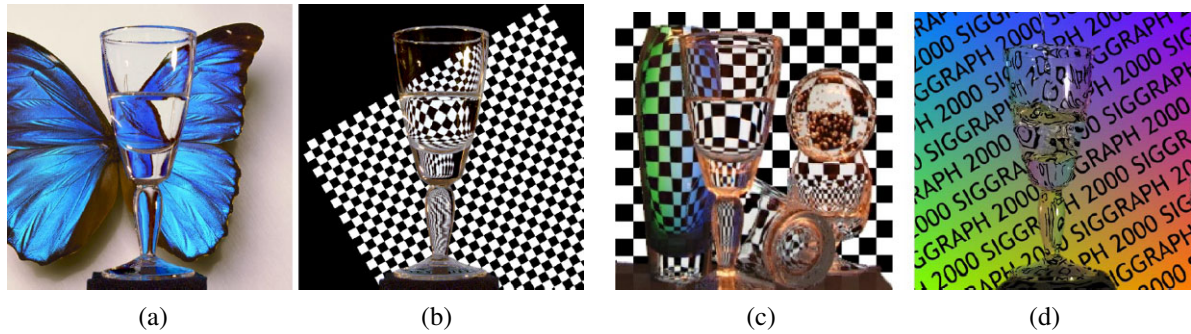


Figure 13.10 Environment mattes: (a–b) a refractive object can be placed in front of a series of backgrounds and their light patterns will be correctly refracted (Zongker, Werner, Curless *et al.* 1999) (c) multiple refractions can be handled using a mixture of Gaussians model and (d) real-time mattes can be pulled using a single graded colored background (Chuang, Zongker, Hindorff *et al.* 2000) © 2000 ACM.

What if instead of moving around a virtual camera, we take a complex, refractive object, such as the water goblet shown in Figure 13.10, and place it in front of a new background? Instead of modeling the 4D space of rays emanating from a scene, we now need to model how each pixel in our view of this object refracts incident light coming from its environment.

What is the intrinsic dimensionality of such a representation and how do we go about capturing it? Let us assume that if we trace a light ray from the camera at pixel (x, y) toward the object, it is reflected or refracted back out toward its environment at an angle (ϕ, θ) . If we assume that other objects and illuminants are sufficiently distant (the same assumption we made for surface light fields in Section 13.3.2), this 4D mapping $(x, y) \rightarrow (\phi, \theta)$ captures all the information between a refractive object and its environment. Zongker, Werner, Curless *et al.* (1999) call such a representation an *environment matte*, since it generalizes the process of object matting (Section 10.4) to not only cut and paste an object from one image into another but also take into account the subtle refractive or reflective interplay between the object and its environment.

Recall from Equations (3.8) and (10.30) that a foreground object can be represented by its premultiplied colors and opacities $(\alpha F, \alpha)$. Such a matte can then be composited onto a new background B using

$$C_i = \alpha_i F_i + (1 - \alpha_i) B_i, \quad (13.1)$$

where i is the pixel under consideration. In environment matting, we augment this equation with a reflective or refractive term to model indirect light paths between the environment and the camera. In the original work of Zongker, Werner, Curless *et al.* (1999), this indirect component I_i is modeled as

$$I_i = R_i \int A_i(\mathbf{x}) B(\mathbf{x}) d\mathbf{x}, \quad (13.2)$$

where A_i is the rectangular *area of support* for that pixel, R_i is the colored reflectance or transmittance (for colored glossy surfaces or glass), and $B(\mathbf{x})$ is the background (environment) image, which is integrated over the area $A_i(\mathbf{x})$. In follow-on work, Chuang, Zongker,

Hindorff *et al.* (2000) use a superposition of oriented Gaussians,

$$I_i = \sum_j R_{ij} \int G_{ij}(\mathbf{x}) B(\mathbf{x}) d\mathbf{x}, \quad (13.3)$$

where each 2D Gaussian

$$G_{ij}(\mathbf{x}) = G_{2D}(\mathbf{x}; \mathbf{c}_{ij}, \boldsymbol{\sigma}_{ij}, \theta_{ij}) \quad (13.4)$$

is modeled by its center \mathbf{c}_{ij} , unrotated widths $\boldsymbol{\sigma}_{ij} = (\sigma_{ij}^x, \sigma_{ij}^y)$, and orientation θ_{ij} .

Given a representation for an environment matte, how can we go about estimating it for a particular object? The trick is to place the object in front of a monitor (or surrounded by a set of monitors), where we can change the illumination patterns $B(\mathbf{x})$ and observe the value of each composite pixel C_i .⁷

As with traditional two-screen matting (Section 10.4.1), we can use a variety of solid colored backgrounds to estimate each pixel's foreground color $\alpha_i F_i$ and partial coverage (opacity) α_i . To estimate the area of support A_i in (13.2), Zongker, Werner, Curless *et al.* (1999) use a series of periodic horizontal and vertical solid stripes at different frequencies and phases, which is reminiscent of the structured light patterns used in active rangefinding (Section 12.2). For the more sophisticated mixture of Gaussian model (13.3), Chuang, Zongker, Hindorff *et al.* (2000) sweep a series of narrow Gaussian stripes at four different orientations (horizontal, vertical, and two diagonals), which enables them to estimate multiple oriented Gaussian responses at each pixel.

Once an environment matte has been “pulled”, it is then a simple matter to replace the background with a new image $B(\mathbf{x})$ to obtain a novel composite of the object placed in a different environment (Figure 13.10a–c). The use of multiple backgrounds during the matting process, however, precludes the use of this technique with dynamic scenes, e.g., water pouring into a glass (Figure 13.10d). In this case, a single graded color background can be used to estimate a single 2D monochromatic displacement for each pixel (Chuang, Zongker, Hindorff *et al.* 2000).

13.4.1 Higher-dimensional light fields

As you can tell from the preceding discussion, an environment matte in principle maps every pixel (x, y) into a 4D distribution over light rays and is, hence, a six-dimensional representation. (In practice, each 2D pixel's response is parameterized using a dozen or so parameters, e.g., $\{F, \alpha, B, R, A\}$, instead of a full mapping.) What if we want to model an object's refractive properties from every potential point of view? In this case, we need a mapping from every incoming 4D light ray to every potential exiting 4D light ray, which is an 8D representation. If we use the same trick as with surface light fields, we can parameterize each surface point by its 4D BRDF to reduce this mapping back down to 6D but this loses the ability to handle multiple refractive paths.

If we want to handle dynamic light fields, we need to add another temporal dimension. (Wenger, Gardner, Tchou *et al.* (2005) gives a nice example of a dynamic appearance and illumination acquisition system.) Similarly, if we want a continuous distribution over wavelengths, this becomes another dimension.

⁷ If we relax the assumption that the environment is distant, the monitor can be placed at several depths to estimate a depth-dependent mapping function (Zongker, Werner, Curless *et al.* 1999).

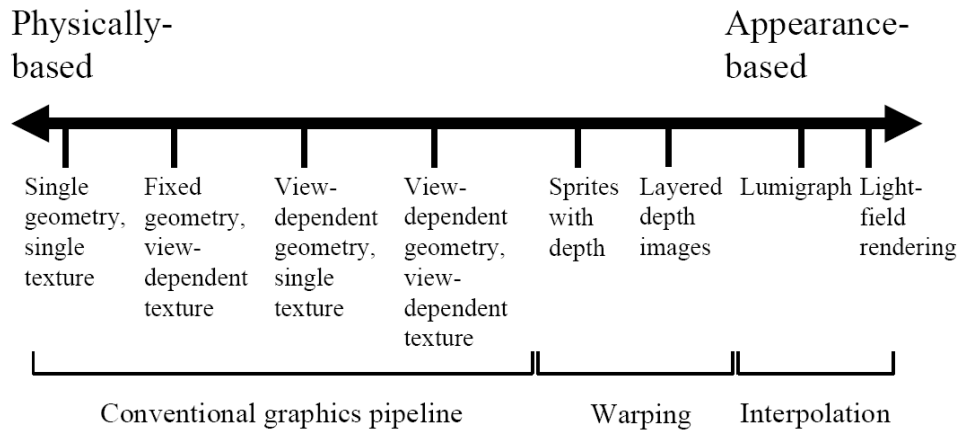


Figure 13.11 The geometry-image continuum in image-based rendering (Kang, Szeliski, and Anandan 2000) © 2000 IEEE. Representations at the left of the spectrum use more detailed geometry and simpler image representations, while representations and algorithms on the right use more images and less geometry.

These examples illustrate how modeling the full complexity of a visual scene through sampling can be extremely expensive. Fortunately, constructing specialized models, which exploit knowledge about the physics of light transport along with the natural coherence of real-world objects, can make these problems more tractable.

13.4.2 The modeling to rendering continuum

The image-based rendering representations and algorithms we have studied in this chapter span a continuum ranging from classic 3D texture-mapped models all the way to pure sampled ray-based representations such as light fields (Figure 13.11). Representations such as view-dependent texture maps and Lumigraphs still use a single global geometric model, but select the colors to map onto these surfaces from nearby images. View-dependent geometry, e.g., multiple depth maps, sidestep the need for coherent 3D geometry, and can sometimes better model local non-rigid effects such as specular motion (Swaminathan, Kang, Szeliski *et al.* 2002; Criminisi, Kang, Swaminathan *et al.* 2005). Sprites with depth and layered depth images use image-based representations of both color and geometry and can be efficiently rendered using warping operations rather than 3D geometric rasterization.

The best choice of representation and rendering algorithm depends on both the quantity and quality of the input imagery as well as the intended application. When nearby views are being rendered, image-based representations capture more of the visual fidelity of the real world because they directly sample its appearance. On the other hand, if only a few input images are available or the image-based models need to be manipulated, e.g., to change their shape or appearance, more abstract 3D representations such as geometric and local reflection models are a better fit. As we continue to capture and manipulate increasingly larger quantities of visual data, research into these aspects of image-based modeling and rendering will continue to evolve.



Figure 13.12 Video Rewrite (Bregler, Covell, and Slaney 1997) © 1997 ACM: the video frames are composed from bits and pieces of old video footage matched to a new audio track.

13.5 Video-based rendering

Since multiple images can be used to render new images or interactive experiences, can something similar be done with video? In fact, a fair amount of work has been done in the area of *video-based rendering* and *video-based animation*, two terms first introduced by Schödl, Szeliski, Salesin *et al.* (2000) to denote the process of generating new video sequences from captured video footage. An early example of such work is Video Rewrite (Bregler, Covell, and Slaney 1997), in which archival video footage is “re-animated” by having actors say new utterances (Figure 13.12). More recently, the term video-based rendering has been used by some researchers to denote the creation of virtual camera moves from a set of synchronized video cameras placed in a studio (Magnor 2005). (The terms *free-viewpoint video* and *3D video* are also sometimes used, see Section 13.5.4.)

In this section, we present a number of video-based rendering systems and applications. We start with *video-based animation* (Section 13.5.1), in which video footage is re-arranged or modified, e.g., in the capture and re-rendering of facial expressions. A special case of this are *video textures* (Section 13.5.2), in which source video is automatically cut into segments and re-looped to create infinitely long video animations. It is also possible to create such animations from still pictures or paintings, by segmenting the image into separately moving regions and animating them using stochastic motion fields (Section 13.5.3).

Next, we turn our attention to *3D video* (Section 13.5.4), in which multiple synchronized video cameras are used to film a scene from different directions. The source video frames can then be re-combined using image-based rendering techniques, such as view interpolation, to create virtual camera paths between the source cameras as part of a real-time viewing experience. Finally, we discuss capturing environments by driving or walking through them with panoramic video cameras in order to create interactive video-based walkthrough experiences (Section 13.5.5).

13.5.1 Video-based animation

As we mentioned above, an early example of video-based animation is Video Rewrite, in which frames from original video footage are rearranged in order to match them to novel spoken utterances, e.g., for movie dubbing (Figure 13.12). This is similar in spirit to the way that *concatenative speech synthesis* systems work (Taylor 2009).

In their system, Bregler, Covell, and Slaney (1997) first use speech recognition to extract phonemes from both the source video material and the novel audio stream. Phonemes are grouped into *triphones* (triplets of phonemes), since these better model the *coarticulation* effect present when people speak. Matching triphones are then found in the source footage and audio track. The mouth images corresponding to the selected video frames are then cut and pasted into the desired video footage being re-animated or dubbed, with appropriate geometric transformations to account for head motion. During the analysis phase, features corresponding to the lips, chin, and head are tracked using computer vision techniques. During synthesis, image morphing techniques are used to blend and stitch adjacent mouth shapes into a more coherent whole. In more recent work, Ezzat, Geiger, and Poggio (2002) describe how to use a *multidimensional morphable model* (Section 12.6.2) combined with regularized trajectory synthesis to improve these results.

A more sophisticated version of this system, called *face transfer*, uses a novel source video, instead of just an audio track, to drive the animation of a previously captured video, i.e., to re-render a video of a talking head with the appropriate visual speech, expression, and head pose elements (Vlasic, Brand, Pfister *et al.* 2005). This work is one of many *performance-driven animation* systems (Section 4.1.5), which are often used to animate 3D facial models (Figures 12.18–12.19). While traditional performance-driven animation systems use marker-based motion capture (Williams 1990; Litwinowicz and Williams 1994; Ma, Jones, Chiang *et al.* 2008), video footage can now often be used directly to control the animation (Buck, Finkelstein, Jacobs *et al.* 2000; Pighin, Szeliski, and Salesin 2002; Zhang, Snavely, Curless *et al.* 2004; Vlasic, Brand, Pfister *et al.* 2005; Roble and Zafar 2009).

In addition to its most common application to facial animation, video-based animation can also be applied to whole body motion (Section 12.6.4), e.g., by matching the flow fields between two different source videos and using one to drive the other (Efros, Berg, Mori *et al.* 2003). Another approach to video-based rendering is to use flow or 3D modeling to *unwrap* surface textures into stabilized images, which can then be manipulated and re-rendered onto the original video (Pighin, Szeliski, and Salesin 2002; Rav-Acha, Kohli, Fitzgibbon *et al.* 2008).

13.5.2 Video textures

Video-based animation is a powerful means of creating photo-realistic videos by re-purposing existing video footage to match some other desired activity or script. What if instead of constructing a special animation or narrative, we simply want the video to continue playing in a plausible manner? For example, many Web sites use images or videos to highlight their destinations, e.g., to portray attractive beaches with surf and palm trees waving in the wind. Instead of using a static image or a video clip that has a discontinuity when it loops, can we transform the video clip into an infinite-length animation that plays forever?

This idea is the basis of *video textures*, in which a short video clip can be arbitrarily extended by re-arranging video frames while preserving visual continuity (Schödl, Szeliski, Salesin *et al.* 2000). The basic problem in creating video textures is how to perform this re-arrangement without introducing visual artifacts. Can you think of how you might do this?

The simplest approach is to match frames by visual similarity (e.g., L_2 distance) and to jump between frames that appear similar. Unfortunately, if the motions in the two frames are different, a dramatic visual artifact will occur (the video will appear to “stutter”). For



Figure 13.13 Video textures (Schödl, Szeliski, Salesin *et al.* 2000) © 2000 ACM: (a) a clock pendulum, with correctly matched direction of motion; (b) a candle flame, showing temporal transition arcs; (c) the flag is generated using morphing at jumps; (d) a bonfire uses longer cross-dissolves; (e) a waterfall cross-dissolves several sequences at once; (f) a smiling animated face; (g) two swinging children are animated separately; (h) the balloons are automatically segmented into separate moving regions; (i) a synthetic fish tank consisting of bubbles, plants, and fish. Videos corresponding to these images can be found at <http://www.cc.gatech.edu/gvu/perception/projects/videotexture/>.

example, if we fail to match the motions of the clock pendulum in Figure 13.13a, it can suddenly change direction in mid-swing.

How can we extend our basic frame matching to also match motion? In principle, we could compute optic flow at each frame and match this. However, flow estimates are often unreliable (especially in textureless regions) and it is not clear how to weight the visual and motion similarities relative to each other. As an alternative, Schödl, Szeliski, Salesin *et al.* (2000) suggest matching *triplets* or larger neighborhoods of adjacent video frames, much in the same way as Video Rewrite matches triphones. Once we have constructed an $n \times n$ similarity matrix between all video frames (where n is the number of frames), a simple finite impulse response (FIR) filtering of each match sequence can be used to emphasize subsequences that match well.

The results of this match computation gives us a *jump table* or, equivalently, a transition probability between any two frames in the original video. This is shown schematically as red arcs in Figure 13.13b, where the red bar indicates which video frame is currently being displayed, and arcs light up as a forward or backward transition is taken. We can view these transition probabilities as encoding the *hidden Markov model* (HMM) that underlies a stochastic video generation process.

Sometimes, it is not possible to find exactly matching subsequences in the original video. In this case, morphing, i.e., warping and blending frames during transitions (Section 3.6.3) can be used to hide the visual differences (Figure 13.13c). If the motion is chaotic enough, as in a bonfire or a waterfall (Figures 13.13d–e), simple blending (extended cross-dissolves) may be sufficient. Improved transitions can also be obtained by performing 3D graph cuts on the spatio-temporal volume around a transition (Kwatra, Schödl, Essa *et al.* 2003).

Video textures need not be restricted to chaotic random phenomena such as fire, wind, and water. Pleasing video textures can be created of people, e.g., a smiling face (as in Figure 13.13f) or someone running on a treadmill (Schödl, Szeliski, Salesin *et al.* 2000). When multiple people or objects are moving independently, as in Figures 13.13g–h, we must first segment the video into independently moving regions and animate each region separately. It is also possible to create large panoramic video textures from a slowly panning camera (Agarwala, Zheng, Pal *et al.* 2005).

Instead of just playing back the original frames in a stochastic (random) manner, video textures can also be used to create scripted or interactive animations. If we extract individual elements, such as fish in a fishtank (Figure 13.13i) into separate *video sprites*, we can animate them along pre-specified paths (by matching the path direction with the original sprite motion) to make our video elements move in a desired fashion (Schödl and Essa 2002). In fact, work on video textures inspired research on systems that re-synthesize new motion sequences from motion capture data, which some people refer to as “mocap soup” (Arikan and Forsyth 2002; Kovar, Gleicher, and Pighin 2002; Lee, Chai, Reitsma *et al.* 2002; Li, Wang, and Shum 2002; Pullen and Bregler 2002).

While video textures primarily analyze the video as a sequence of frames (or regions) that can be re-arranged in time, *temporal textures* (Szummer and Picard 1996; Bar-Joseph, El-Yaniv, Lischinski *et al.* 2001) and *dynamic textures* (Doretto, Chiuso, Wu *et al.* 2003; Yuan, Wen, Liu *et al.* 2004; Doretto and Soatto 2006) treat the video as a 3D spatio-temporal volume with textural properties, which can be described using auto-regressive temporal models.

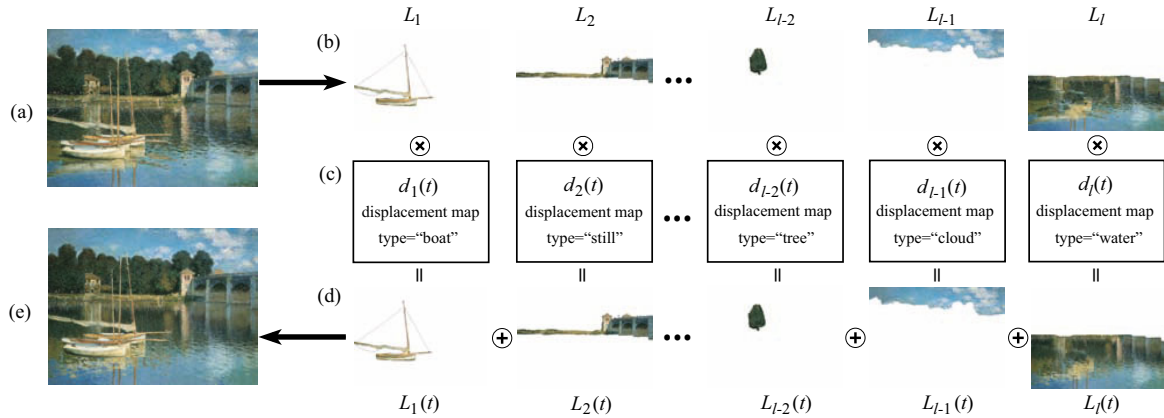


Figure 13.14 Animating still pictures (Chuang, Goldman, Zheng *et al.* 2005) © 2005 ACM. (a) The input still image is manually segmented into (b) several layers. (c) Each layer is then animated with a different stochastic motion texture (d) The animated layers are then composited to produce (e) the final animation

13.5.3 Application: Animating pictures

While video textures can turn a short video clip into an infinitely long video, can the same thing be done with a single still image? The answer is yes, if you are willing to first segment the image into different layers and then animate each layer separately.

Chuang, Goldman, Zheng *et al.* (2005) describe how an image can be decomposed into separate layers using interactive matting techniques. Each layer is then animated using a class-specific synthetic motion. As shown in Figure 13.14, boats rock back and forth, trees sway in the wind, clouds move horizontally, and water ripples, using a shaped noise displacement map. All of these effects can be tied to some global control parameters, such as the velocity and direction of a virtual wind. After being individually animated, the layers can be composited to create a final dynamic rendering.

13.5.4 3D Video

In recent years, the popularity of 3D movies has grown dramatically, with recent releases ranging from *Hannah Montana*, through U2's 3D concert movie, to James Cameron's *Avatar*. Currently, such releases are filmed using stereoscopic camera rigs and displayed in theaters (or at home) to viewers wearing polarized glasses.⁸ In the future, however, home audiences may wish to view such movies with multi-zone auto-stereoscopic displays, where each person gets his or her own customized stereo stream and can move around a scene to see it from different perspectives.⁹

The stereo matching techniques developed in the computer vision community along with image-based rendering (view interpolation) techniques from graphics are both essential components in such scenarios, which are sometimes called *free-viewpoint video* (Carranza, Theobalt, Magnor *et al.* 2003) or *virtual viewpoint video* (Zitnick, Kang, Uyttendaele *et al.* 2004). In

⁸ <http://www.3d-summit.com/>.

⁹ <http://www.siggraph.org/s2008/attendees/caf/3d/>.

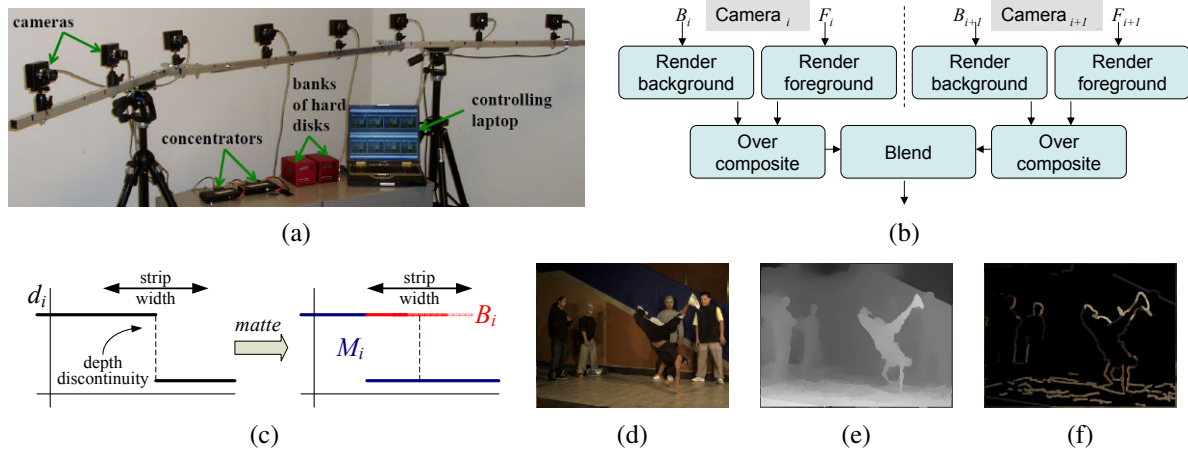


Figure 13.15 Video view interpolation (Zitnick, Kang, Uyttendaele *et al.* 2004) © 2004 ACM: (a) the capture hardware consists of eight synchronized cameras; (b) the background and foreground images from each camera are rendered and composited before blending; (c) the two-layer representation, before and after boundary matting; (d) background color estimates; (e) background depth estimates; (f) foreground color estimates.

addition to solving a series of per-frame reconstruction and view interpolation problems, the depth maps or proxies produced by the analysis phase must be temporally consistent in order to avoid flickering artifacts.

Shum, Chan, and Kang (2007) and Magnor (2005) present nice overviews of various video view interpolation techniques and systems. These include the Virtualized Reality system of Kanade, Rander, and Narayanan (1997) and Vedula, Baker, and Kanade (2005), Immersive Video (Moezzi, Katkere, Kuramura *et al.* 1996), Image-Based Visual Hulls (Matusik, Buehler, Raskar *et al.* 2000; Matusik, Buehler, and McMillan 2001), and Free-Viewpoint Video (Carranza, Theobalt, Magnor *et al.* 2003), which all use global 3D geometric models (surface-based (Section 12.3) or volumetric (Section 12.5)) as their proxies for rendering. The work of Vedula, Baker, and Kanade (2005) also computes *scene flow*, i.e., the 3D motion between corresponding surface elements, which can then be used to perform spatio-temporal interpolation of the multi-view video stream.

The Virtual Viewpoint Video system of Zitnick, Kang, Uyttendaele *et al.* (2004), on the other hand, associates a two-layer depth map with each input image, which allows them to accurately model occlusion effects such as the mixed pixels that occur at object boundaries. Their system, which consists of eight synchronized video cameras connected to a disk array (Figure 13.15a), first uses segmentation-based stereo to extract a depth map for each input image (Figure 13.15e). Near object boundaries (depth discontinuities), the background layer is extended along a strip behind the foreground object (Figure 13.15c) and its color is estimated from the neighboring images where it is not occluded (Figure 13.15d). Automated matting techniques (Section 10.4) are then used to estimate the fractional opacity and color of boundary pixels in the foreground layer (Figure 13.15f).

At render time, given a new virtual camera that lies between two of the original cameras, the layers in the neighboring cameras are rendered as texture-mapped triangles and the foreground layer (which may have fractional opacities) is then composited over the background

layer (Figure 13.15b). The resulting two images are merged and blended by comparing their respective z-buffer values. (Whenever the two z-values are sufficiently close, a linear blend of the two colors is computed.) The interactive rendering system runs in real time using regular graphics hardware. It can therefore be used to change the observer's viewpoint while playing the video or to freeze the scene and explore it in 3D. More recently, Rogmans, Lu, Bekaert *et al.* (2009) have developed GPU implementations of both real-time stereo matching and real-time rendering algorithms, which enable them to explore algorithmic alternatives in a real-time setting.

At present, the depth maps computed from the eight stereo cameras using off-line stereo matching have produced the highest quality depth maps associated with live video.¹⁰ They are therefore often used in studies of 3D video compression, which is an active area of research (Smolic and Kauff 2005; Gotchev and Rosenhahn 2009). Active video-rate depth sensing cameras, such as the 3DV Zcam (Iddan and Yahav 2001), which we discussed in Section 12.2.1, are another potential source of such data.

When large numbers of closely spaced cameras are available, as in the Stanford Light Field Camera (Wilburn, Joshi, Vaish *et al.* 2005), it may not always be necessary to compute explicit depth maps to create video-based rendering effects, although the results are usually of higher quality if you do (Vaish, Szeliski, Zitnick *et al.* 2006).

13.5.5 Application: Video-based walkthroughs

Video camera arrays enable the simultaneous capture of 3D dynamic scenes from multiple viewpoints, which can then enable the viewer to explore the scene from viewpoints near the original capture locations. What if instead we wish to capture an extended area, such as a home, a movie set, or even an entire city?

In this case, it makes more sense to move the camera through the environment and play back the video as an interactive video-based walkthrough. In order to allow the viewer to look around in all directions, it is preferable to use a panoramic video camera (Uyttendaele, Criminisi, Kang *et al.* 2004).¹¹

One way to structure the acquisition process is to capture these images in a 2D horizontal plane, e.g., over a grid superimposed inside a room. The resulting *sea of images* (Aliaga, Funkhouser, Yanovsky *et al.* 2003) can be used to enable continuous motion between the captured locations.¹² However, extending this idea to larger settings, e.g., beyond a single room, can become tedious and data-intensive.

Instead, a natural way to explore a space is often to just walk through it along some pre-specified paths, just as museums or home tours guide users along a particular path, say down the middle of each room.¹³ Similarly, city-level exploration can be achieved by driving down the middle of each street and allowing the user to branch at each intersection. This idea dates back to the Aspen MovieMap project (Lippman 1980), which recorded analog video taken from moving cars onto videodiscs for later interactive playback.

¹⁰ <http://research.microsoft.com/en-us/um/redmond/groups/ivm/vvv/>.

¹¹ See <http://www.cis.upenn.edu/~kostas/omni.html> for descriptions of panoramic (omnidirectional) vision systems and associated workshops.

¹² (The Photo Tourism system of Snavely, Seitz, and Szeliski (2006) applies this idea to less structured collections.

¹³ In computer games, restricting a player to forward and backward motion along predetermined paths is called *rail-based gaming*.

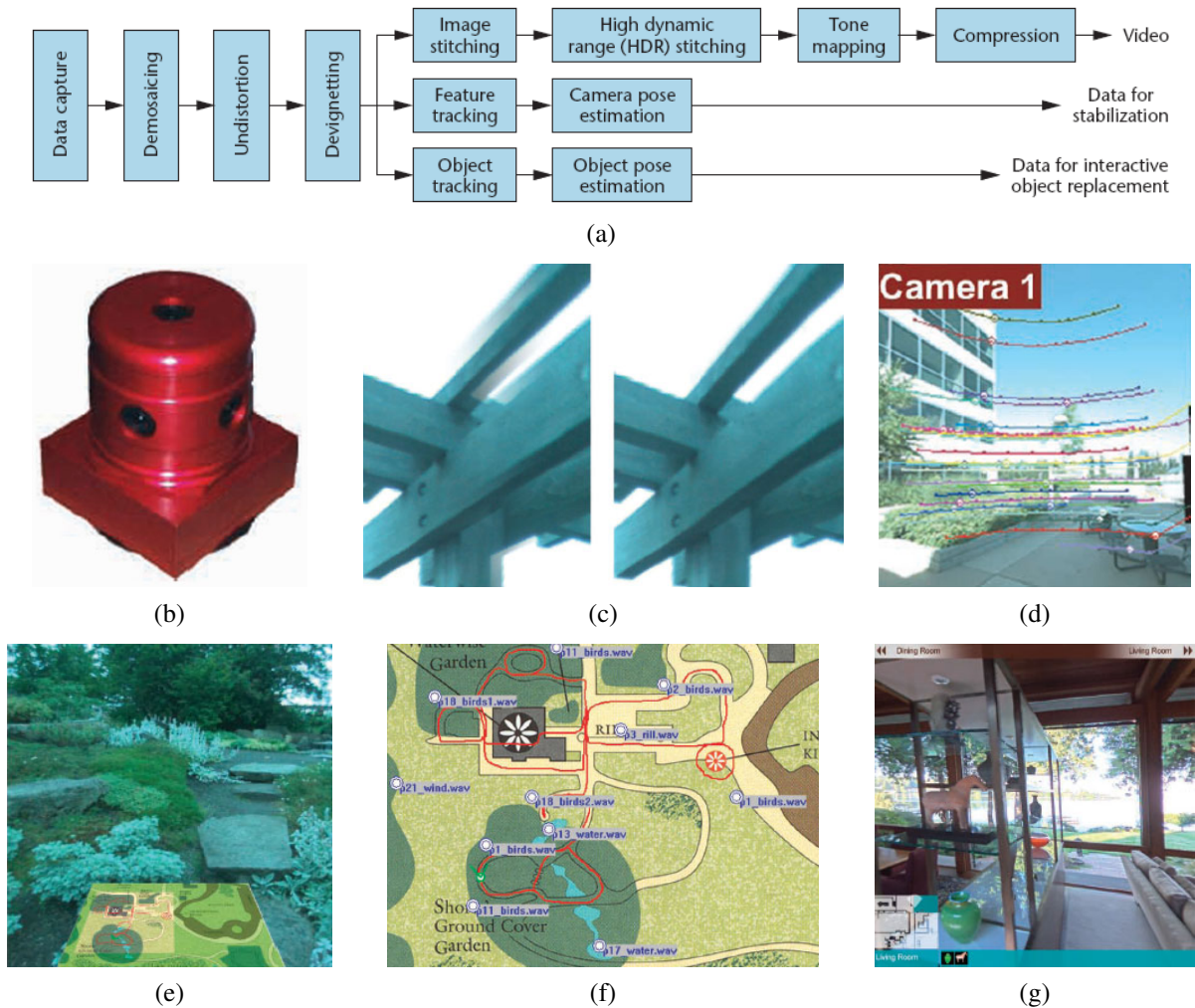


Figure 13.16 Video-based walkthroughs (Uyttendaele, Criminisi, Kang *et al.* 2004) © 2004 IEEE: (a) system diagram of video pre-processing; (b) the Point Grey Ladybug camera; (c) ghost removal using multi-perspective plane sweep; (d) point tracking, used both for calibration and stabilization; (e) interactive garden walkthrough with map below; (f) overhead map authoring and sound placement; (g) interactive home walkthrough with navigation bar (top) and icons of interest (bottom).

Recent improvements in video technology now enable the capture of panoramic (spherical) video using a small co-located array of cameras, such as the Point Grey Ladybug camera¹⁴ (Figure 13.16b) developed by Uyttendaele, Criminisi, Kang *et al.* (2004) for their interactive video-based walkthrough project. In their system, the synchronized video streams from the six cameras (Figure 13.16a) are stitched together into 360° panoramas using a variety of techniques developed specifically for this project.

Because the cameras do not share the same center of projection, parallax between the cameras can lead to ghosting in the overlapping fields of view (Figure 13.16c). To remove this, a multi-perspective plane sweep stereo algorithm is used to estimate per-pixel depths at each column in the overlap area. To calibrate the cameras relative to each other, the camera is spun in place and a constrained structure from motion algorithm (Figure 7.8) is used to estimate the relative camera poses and intrinsics. Feature tracking is then run on the walkthrough video in order to stabilize the video sequence—Liu, Gleicher, Jin *et al.* (2009) have carried out more recent work along these lines.

Indoor environments with windows, as well as sunny outdoor environments with strong shadows, often have a dynamic range that exceeds the capabilities of video sensors. For this reason, the Ladybug camera has a programmable exposure capability that enables the bracketing of exposures at subsequent video frames. In order to merge the resulting video frames into high dynamic range (HDR) video, pixels from adjacent frames need to be motion-compensated before being merged (Kang, Uyttendaele, Winder *et al.* 2003).

The interactive walk-through experience becomes much richer and more navigable if an overview map is available as part of the experience. In Figure 13.16f, the map has annotations, which can show up during the tour, and localized sound sources, which play (with different volumes) when the viewer is nearby. The process of aligning the video sequence with the map can be automated using a process called *map correlation* (Levin and Szeliski 2004).

All of these elements combine to provide the user with a rich, interactive, and immersive experience. Figure 13.16e shows a walk through the Bellevue Botanical Gardens, with an overview map in perspective below the live video window. Arrows on the ground are used to indicate potential directions of travel. The viewer simply orients his view towards one of the arrows (the experience can be driven using a game controller) and “walks” forward along the desired path.

Figure 13.16g shows an indoor home tour experience. In addition to a schematic map in the lower left corner and adjacent room names along the top navigation bar, icons appear along the bottom whenever items of interest, such as a homeowner’s art pieces, are visible in the main window. These icons can then be clicked to provide more information and 3D views.

The development of interactive video tours spurred a renewed interest in 360° video-based virtual travel and mapping experiences, as evidenced by commercial sites such as Google’s Street View and Bing Maps. The same videos can also be used to generate turn-by-turn driving directions, taking advantage of both expanded fields of view and image-based rendering to enhance the experience (Chen, Neubert, Ofek *et al.* 2009).

As we continue to capture more and more of our real world with large amounts of high-quality imagery and video, the interactive modeling, exploration, and rendering techniques described in this chapter will play an even bigger role in bringing virtual experiences based

¹⁴ <http://www.ptgrey.com/>.

on remote areas of the world closer to everyone.

13.6 Additional reading

Two good recent surveys of image-based rendering are by Kang, Li, Tong *et al.* (2006) and Shum, Chan, and Kang (2007), with earlier surveys available from Kang (1999), McMillan and Gortler (1999), and Debevec (1999). The term *image-based rendering* was introduced by McMillan and Bishop (1995), although the seminal paper in the field is the view interpolation paper by Chen and Williams (1993). Debevec, Taylor, and Malik (1996) describe their Façade system, which not only created a variety of image-based modeling tools but also introduced the widely used technique of *view-dependent texture mapping*.

Early work on planar impostors and layers was carried out by Shade, Lischinski, Salesin *et al.* (1996), Lengyel and Snyder (1997), and Torborg and Kajiya (1996), while newer work based on *sprites with depth* is described by Shade, Gortler, He *et al.* (1998).

The two foundational papers in image-based rendering are *Light field rendering* by Levoy and Hanrahan (1996) and *The Lumigraph* by Gortler, Grzeszczuk, Szeliski *et al.* (1996). Buehler, Bosse, McMillan *et al.* (2001) generalize the Lumigraph approach to irregularly spaced collections of images, while Levoy (2006) provides a survey and more gentle introduction to the topic of light field and image-based rendering.

Surface light fields (Wood, Azuma, Aldinger *et al.* 2000) provide an alternative parameterization for light fields with accurately known surface geometry and support both better compression and the possibility of editing surface properties. Concentric mosaics (Shum and He 1999; Shum, Wang, Chai *et al.* 2002) and panoramas with depth (Peleg, Ben-Ezra, and Pritch 2001; Li, Shum, Tang *et al.* 2004; Zheng, Kang, Cohen *et al.* 2007), provide useful parameterizations for light fields captured with panning cameras. Multi-perspective images (Rademacher and Bishop 1998) and manifold projections (Peleg and Herman 1997), although not true light fields, are also closely related to these ideas.

Among the possible extensions of light fields to higher-dimensional structures, environment mattes (Zongker, Werner, Curless *et al.* 1999; Chuang, Zongker, Hindorff *et al.* 2000) are the most useful, especially for placing captured objects into new scenes.

Video-based rendering, i.e., the re-use of video to create new animations or virtual experiences, started with the seminal work of Szummer and Picard (1996), Bregler, Covell, and Slaney (1997), and Schödl, Szeliski, Salesin *et al.* (2000). Important follow-on work to these basic re-targeting approaches was carried out by Schödl and Essa (2002), Kwatra, Schödl, Essa *et al.* (2003), Doretto, Chiuso, Wu *et al.* (2003), Wang and Zhu (2003), Zhong and Sclaroff (2003), Yuan, Wen, Liu *et al.* (2004), Doretto and Soatto (2006), Zhao and Pietikäinen (2007), and Chan and Vasconcelos (2009).

Systems that allow users to change their 3D viewpoint based on multiple synchronized video streams include those by Moezzi, Katkere, Kuramura *et al.* (1996), Kanade, Rander, and Narayanan (1997), Matusik, Buehler, Raskar *et al.* (2000), Matusik, Buehler, and McMillan (2001), Carranza, Theobalt, Magnor *et al.* (2003), Zitnick, Kang, Uyttendaele *et al.* (2004), Magnor (2005), and Vedula, Baker, and Kanade (2005). 3D (multiview) video coding and compression is also an active area of research (Smolic and Kauff 2005; Gotchev and Rosenhahn 2009), with 3D Blu-Ray discs, encoded using the multiview video coding (MVC) extension to H.264/MPEG-4 AVC, expected by the end of 2010.

13.7 Exercises

Ex 13.1: Depth image rendering Develop a “view extrapolation” algorithm to re-render a previously computed stereo depth map coupled with its corresponding reference color image.

1. Use a 3D graphics mesh rendering system such as OpenGL or Direct3D, with two triangles per pixel quad and perspective (projective) texture mapping (Debevec, Yu, and Borshukov 1998).
2. Alternatively, use the one- or two-pass forward warper you constructed in Exercise 3.24, extended using (2.68–2.70) to convert from disparities or depths into displacements.
3. (Optional) Kinks in straight lines introduced during view interpolation or extrapolation are visually noticeable, which is one reason why image morphing systems let you specify line correspondences (Beier and Neely 1992). Modify your depth estimation algorithm to match and estimate the geometry of straight lines and incorporate it into your image-based rendering algorithm.

Ex 13.2: View interpolation Extend the system you created in the previous exercise to render two reference views and then blend the images using a combination of z-buffering, hole filling, and blending (morphing) to create the final image (Section 13.1).

1. (Optional) If the two source images have very different exposures, the hole-filled regions and the blended regions will have different exposures. Can you extend your algorithm to mitigate this?
2. (Optional) Extend your algorithm to perform three-way (trilinear) interpolation between neighboring views. You can triangulate the reference camera poses and use barycentric coordinates for the virtual camera in order to determine the blending weights.

Ex 13.3: View morphing Modify your view interpolation algorithm to perform morphs between views of a non-rigid object, such as a person changing expressions.

1. Instead of using a pure stereo algorithm, use a general flow algorithm to compute displacements, but separate them into a rigid displacement due to camera motion and a non-rigid deformation.
2. At render time, use the rigid geometry to determine the new pixel location but then add a fraction of the non-rigid displacement as well.
3. Alternatively, compute a stereo depth map but let the user specify additional correspondences or use a feature-based matching algorithm to provide them automatically.
4. (Optional) Take a single image, such as the Mona Lisa or a friend’s picture, and create an animated 3D view morph (Seitz and Dyer 1996).
 - (a) Find the vertical axis of symmetry in the image and reflect your reference image to provide a virtual pair (assuming the person’s hairstyle is somewhat symmetric).
 - (b) Use structure from motion to determine the relative camera pose of the pair.

- (c) Use dense stereo matching to estimate the 3D shape.
- (d) Use view morphing to create a 3D animation.

Ex 13.4: View dependent texture mapping Use a 3D model you created along with the original images to implement a view-dependent texture mapping system.

1. Use one of the 3D reconstruction techniques you developed in Exercises 7.3, 11.9, 11.10, or 12.8 to build a triangulated 3D image-based model from multiple photographs.
2. Extract textures for each model face from your photographs, either by performing the appropriate resampling or by figuring out how to use the texture mapping software to directly access the source images.
3. At run time, for each new camera view, select the best source image for each visible model face.
4. Extend this to blend between the top two or three textures. This is trickier, since it involves the use of texture blending or pixel shading (Debevec, Taylor, and Malik 1996; Debevec, Yu, and Borshukov 1998; Pighin, Hecker, Lischinski *et al.* 1998).

Ex 13.5: Layered depth images Extend your view interpolation algorithm (Exercise 13.2) to store more than one depth or color value per pixel (Shade, Gortler, He *et al.* 1998), i.e., a layered depth image (LDI). Modify your rendering algorithm accordingly. For your data, you can use synthetic ray tracing, a layered reconstructed model, or a volumetric reconstruction.

Ex 13.6: Rendering from sprites or layers Extend your view interpolation algorithm to handle multiple planes or sprites (Section 13.2.1) (Shade, Gortler, He *et al.* 1998).

1. Extract your layers using the technique you developed in Exercise 8.9.
2. Alternatively, use an interactive painting and 3D placement system to extract your layers (Kang 1998; Oh, Chen, Dorsey *et al.* 2001; Shum, Sun, Yamazaki *et al.* 2004).
3. Determine a back-to-front order based on expected visibility or add a z-buffer to your rendering algorithm to handle occlusions.
4. Render and composite all of the resulting layers, with optional alpha matting to handle the edges of layers and sprites.

Ex 13.7: Light field transformations Derive the equations relating regular images to 4D light field coordinates.

1. Determine the mapping between the far plane (u, v) coordinates and a virtual camera's (x, y) coordinates.
 - (a) Start by parameterizing a 3D point on the uv plane in terms of its (u, v) coordinates.
 - (b) Project the resulting 3D point to the camera pixels $(x, y, 1)$ using the usual 3×4 camera matrix P (2.63).
 - (c) Derive the 2D homography relating (u, v) and (x, y) coordinates.

2. Write down a similar transformation for (s, t) to (x, y) coordinates.
3. Prove that if the virtual camera is actually on the (s, t) plane, the (s, t) value depends only on the camera's optical center and is independent of (x, y) .
4. Prove that an image taken by a regular orthographic or perspective camera, i.e., one that has a linear projective relationship between 3D points and (x, y) pixels (2.63), samples the (s, t, u, v) light field along a two-dimensional hyperplane.

Ex 13.8: Light field and Lumigraph rendering Implement a light field or Lumigraph rendering system:

1. Download one of the light field data sets from <http://lightfield.stanford.edu/>.
2. Write an algorithm to synthesize a new view from this light field, using quadri-linear interpolation of (s, t, u, v) ray samples.
3. Try varying the focal plane corresponding to your desired view (Isaksen, McMillan, and Gortler 2000) and see if the resulting image looks sharper.
4. Determine a 3D proxy for the objects in your scene. You can do this by running multi-view stereo over one of your light fields to obtain a depth map per image.
5. Implement the Lumigraph rendering algorithm, which modifies the sampling of rays according to the 3D location of each surface element.
6. Collect a set of images yourself and determine their pose using structure from motion.
7. Implement the unstructured Lumigraph rendering algorithm from Buehler, Bosse, McMillan *et al.* (2001).

Ex 13.9: Surface light fields Construct a surface light field (Wood, Azuma, Aldinger *et al.* 2000) and see how well you can compress it.

1. Acquire an interesting light field of a specular scene or object, or download one from <http://lightfield.stanford.edu/>.
2. Build a 3D model of the object using a multi-view stereo algorithm that is robust to outliers due to specularities.
3. Estimate the Lumisphere for each surface point on the object.
4. Estimate its diffuse components. Is the median the best way to do this? Why not use the minimum color value? What happens if there is Lambertian shading on the diffuse component?
5. Model and compress the remaining portion of the Lumisphere using one of the techniques suggested by Wood, Azuma, Aldinger *et al.* (2000) or invent one of your own.
6. Study how well your compression algorithm works and what artifacts it produces.
7. (Optional) Develop a system to edit and manipulate your surface light field.

Ex 13.10: Handheld concentric mosaics Develop a system to navigate a handheld concentric mosaic.

1. Stand in the middle of a room with a camcorder held at arm's length in front of you and spin in a circle.
2. Use a structure from motion system to determine the camera pose and sparse 3D structure for each input frame.
3. (Optional) Re-bin your image pixels into a more regular concentric mosaic structure.
4. At view time, determine from the new camera's view (which should be near the plane of your original capture) which source pixels to display. You can simplify your computations to determine a source column (and scaling) for each output column.
5. (Optional) Use your sparse 3D structure, interpolated to a dense depth map, to improve your rendering (Zheng, Kang, Cohen *et al.* 2007).

Ex 13.11: Video textures Capture some videos of natural phenomena, such as a water fountain, fire, or smiling face, and loop the video seamlessly into an infinite length video (Schödl, Szeliski, Salesin *et al.* 2000).

1. Compare all the frames in the original clip using an L_2 (sum of square difference) metric. (This assumes the videos were shot on a tripod or have already been stabilized.)
2. Filter the comparison table temporally to accentuate temporal sub-sequences that match well together.
3. Convert your similarity table into a jump probability table through some exponential distribution. Be sure to modify transitions near the end so you do not get "stuck" in the last frame.
4. Starting with the first frame, use your transition table to decide whether to jump forward, backward, or continue to the next frame.
5. (Optional) Add any of the other extensions to the original video textures idea, such as multiple moving regions, interactive control, or graph cut spatio-temporal texture seaming.