Foundations and Trends® in
Computer Graphics and Vision
Vol. 2, No 3 (2006) 173–258

DOI: 10.1561/0600000012

now
the essence of knowledge

Image-Based Rendering

Sing Bing Kang[1], Yin Li[2], Xin Tong[3] and Heung-Yeung Shum[4]

[1] *Microsoft Research, USA, sbkang@microsoft.com*
[2] *Microsoft Corporation, USA, yl@microsoft.com*
[3] *Microsoft Research Asia, China, xtong@microsoft.com*
[4] *Microsoft Research Asia, China, hshum@microsoft.com*

Abstract

Image-based rendering (IBR) is unique in that it requires computer graphics, computer vision, and image processing to join forces to solve a common goal, namely photorealistic rendering through the use of images. IBR as an area of research has been around for about ten years, and substantial progress has been achieved in effectively capturing, representing, and rendering scenes. In this article, we survey the techniques used in IBR. Our survey shows that representations and rendering techniques can differ radically, depending on design decisions related to ease of capture, use of geometry, accuracy of geometry (if used), number and distribution of source images, degrees of freedom for virtual navigation, and expected scene complexity.

1

Introduction

One of the primary goals in computer graphics is photorealistic rendering. Much progress has been made over the years in graphics in a bid to attain this goal, with significant advancements in 3D representations and model acquisition, measurement and modeling of object surface properties such as the bidirectional reflectance distribution function (BRDF) and surface subscattering, illumination modeling, natural objects such as plants, and natural phenomena such as water, fog, smoke, snow, and fire. More sophisticated graphics hardware that permit very fast rendering, programmable vertex and pixel shading, larger caches and memory footprints, and floating-point pixel formats also help in the cause. In other words, a variety of well-established approaches and systems are available for rendering models. See the surveys on physically based rendering [78], global illumination methods [26], and photon mapping (an extension of ray tracing) [44].

Despite all the advancements in the more classical areas of computer graphics, it is still hard to compete with images of real scenes. The rendering quality of environments in animated movies such as *Shrek 2* and even games such as *Ghost Recon* for Xbox 360™ is excellent, but there are hints that these environments are synthetic. Websites such

as http://www.ignorancia.org/ showcase highly photorealistic images that were generated through ray tracing, which is computationally expensive. The special effects in high-budget movies blend seamlessly in real environments, but they typically involved many man-hours to create and refine. The observation that full photorealism is really hard to achieve with conventional 3D and model-based graphics has led researchers to take a "short-cut" by working directly with real images. This approach is called *image-based modeling and rendering*. Some of the special effects used in the movie industry were created using image-based rendering techniques described in this article.

Image-based modeling and rendering techniques have received a lot of attention as a powerful alternative to traditional geometry-based techniques for image synthesis. These techniques use images rather than geometry as the main primitives for rendering novel views. Previous surveys related to image-based rendering (IBR) have suggested characterizing a technique based on how image-centric or geometry-centric it is. This has resulted in the image-geometry continuum (or *IBR continuum*) of image-based representations [46, 52].

2

Representations and Rendering

All IBR techniques have the same goal: to establish a mapping relationship between parts of the representation and the screen pixels, and composite these parts to produce the virtual view (Figure 2.1). One way to characterize IBR techniques is through the amount of geometry required, producing what we call the *IBR continuum*.

2.1 IBR Continuum

For didactic purposes, we classify the various rendering techniques (and their associated representations) into three categories, namely

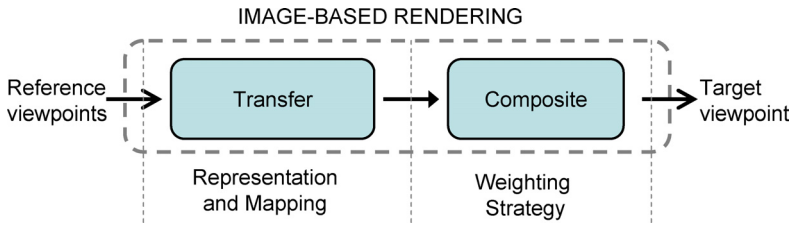


Fig. 2.1 Goals of rendering: establish mapping between representation and image screen, and blend.

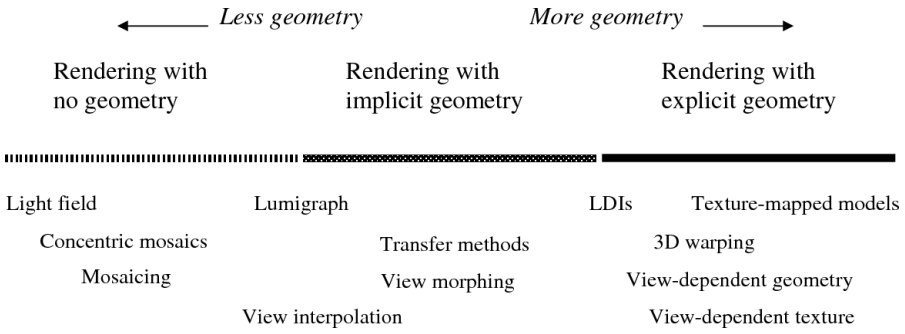


Fig. 2.2 IBR continuum. Categories used in this article are based on this continuum, with representative members shown. Note that the Lumigraph [32] is a bit of an anomaly in this continuum, since it uses explicit geometry and a relatively dense set of images.

rendering with no geometry, rendering with implicit geometry, and rendering with explicit geometry. These categories, depicted in Figure 2.2, should actually be viewed as a continuum rather than absolute discrete ones, since there are techniques that defy strict categorization.

At one end of the IBR continuum, traditional texture mapping relies on very accurate geometric models but only a few images. In an image-based rendering system with depth maps (such as 3D warping [63], and layered-depth images (LDI) [93], and LDI tree [15]), the model consists of a set of images of a scene and their associated depth maps. The surface light field [112] is another geometry-based IBR representation which uses images and Cyberware scanned range data. When depth is available for every point in an image, the image can be rendered from any nearby point of view by projecting the pixels of the image to their proper 3D locations and re-projecting them onto a new picture. For many synthetic environments or objects, depth is available. However, obtaining depth information from real images is hard even with state-of-art vision algorithms.

Some image-based rendering systems do not require explicit geometric models. Rather, they require feature correspondence between images. For example, view interpolation techniques [17] generate novel views by interpolating optical flow between corresponding points. On the other hand, view morphing [91] results in-between camera matrices along the line of two original camera centers, based on point

correspondences. Computer vision techniques are usually used to generate such correspondences.

At the other extreme, light field rendering uses many images but does not require any geometric information or correspondence. Light field rendering [53] produces a new image of a scene by appropriately filtering and interpolating a pre-acquired set of samples. The Lumigraph [32] is similar to light field rendering but it uses approximate geometry to compensate for non-uniform sampling in order to improve rendering performance. Unlike the light field and Lumigraph where cameras are placed on a two-dimensional grid, the Concentric Mosaics representation [95] reduces the amount of data by capturing a sequence of images along a circle path. In addition, it uses a very primitive form of a geometric impostor, whose radial distance is a function of the panning angle. (A geometric impostor is basically a 3D shape used in IBR techniques to improve appearance prediction by depth correction. It is also known as geometric proxy.)

Because light field rendering does not rely on any geometric impostors, it has a tendency to rely on oversampling to counter undesirable aliasing effects in output display. Oversampling means more intensive data acquisition, more storage, and higher redundancy.

2.2 Geometry-Rendering Matrix

Of course, the true picture is appreciably more complicated. The IBR continuum is a simplified version where rendering depends on the amount of geometry. We can expand on this by further categorizing rendering as either *point-based, layer-based,* or *monolithic*. Figure 2.3 shows this expanded version for IBR categorization, which we call the *geometry-rendering matrix*.

Point-based rendering is applied to 3D point clouds (such as layered depth image [93] and relief texture mapping [72]), or point correspondences (used in techniques such as view interpolation [17] and view morphing [91]). Typically, each point is rendered independently. In point-based rendering, the target view is typically restricted to be near to the reference view. The point on the reference view is usually directly mapped to the target view with no compositing operation.

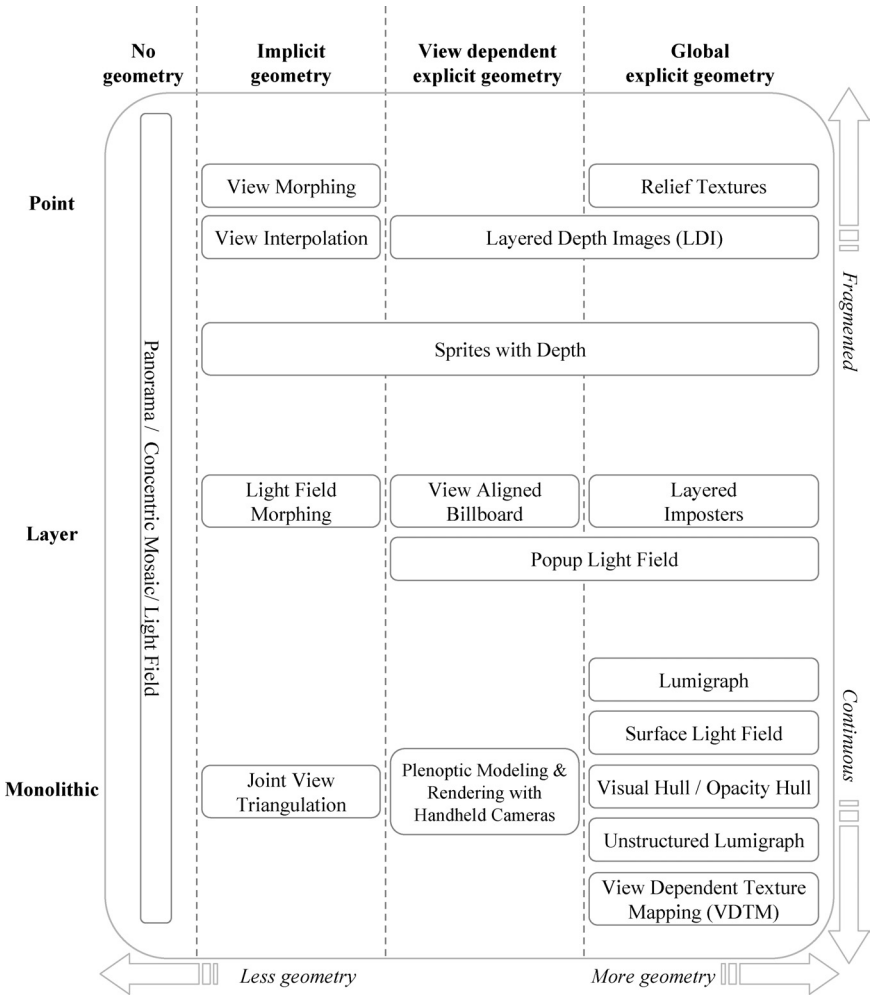


Fig. 2.3 Geometry-rendering matrix. This matrix shows the types of representations (along horizontal axis) and rendering (along vertical axis) in IBR.

Layer-based rendering is performed on a layer-by-layer basis, i.e., each layer is rendered independently and then composed to produce the final view. A planar geometry is usually assumed for each layer, which can be easily rendered as monolithic geometry. This category of techniques include layered imposters [89], sprites with depth [93], and pop-up light field [96]. Sprites with depth [93] use a view-aligned

height map to represent the geometry details; they are rendered using techniques as discussed in the point-based category.

Monolithic rendering is done on single pieces of geometry, each of which usually being a contiguous triangular mesh. We refer to such geometry as monolithic geometry. The monolithic geometry is rendered as an entire object using texture mapping techniques. Therefore, the mapping between the geometry and the target view can be easily determined through view projection. Since the global geometry is reconstructed from multiple images, the final rendering result is composited from multiple reference colors mapped on the same surface point. This category of techniques include view-dependent texture mapping [24], image-based visual hulls [64], opacity hulls [65], surface light field [112], Lumigraph [32], and unstructured Lumigraph [8]. Note that the joint view interpolation [55] also falls into this rendering category because it divides each reference view by triangles and rendered as an entire mesh.

These three rendering types operate on representations that range from being continuous (triangle meshes) to fragmented (point clouds). The rendering of continuous representations has been well exploited using the conventional graphics pipeline. However, it is difficult reconstructing such continuous representations directly from image samples. On the other hand, more fragmented representations are significantly easier to reconstruct, using 3D scanners or vision techniques such as stereo and structure from motion. Unfortunately, fragmented representations tend to be view-dependent and limited by the input image resolution; occlusion and holes are particularly difficult problems to handle.

Note that there is one category of rendering techniques that do not require geometry information. This category includes QuickTime VR [18], light field [53], Concentric Mosaics (CMs) [95], and manifold hopping [98]. (Strictly speaking, both the 4D light field and CMs do have a geometric proxy, namely the focal plane and cylinder, respectively, but they do not require any geometric reconstruction.)

3

Static Scene Representations

In the previous section, we introduced the IBR continuum that spans a variety of representations (Figure 2.2). We also introduced the geometry-rendering matrix as a more detailed version. For clarity, we structure this article based on the simplier IBR continuum: representations that rely on no geometry are described first, followed by those using implicit geometry (i.e., relationships expressed through image correspondences), and finally those with explicit 3D geometry. Where appropriate, we discuss rendering mechanisms (point-based, layer-based, and monolithic rendering) that predominate in the respective geometry-based categories.

4

Rendering with no Geometry

We start with representative techniques for rendering with unknown scene geometry. These techniques typically rely on many input images; they also rely on the characterization of the plenoptic function.

4.1 Plenoptic Modeling

The original 7D plenoptic function [1] is defined as the intensity of light rays passing through the camera center at every 3D location (V_x, V_y, V_z) at every possible angle (θ, ϕ), for every wavelength λ, at every time t, i.e., $P_7(V_x, V_y, V_z, \theta, \phi, \lambda, t)$.

Adelson and Bergen [1] considered one of the tasks of early vision as extracting a compact and useful description of the plenoptic function's local properties (e.g., low order derivatives). It has also been shown by Wong *et al.* [111] that light source directions can be incorporated into the plenoptic function for illumination control. By removing two variables, time t (therefore static environment) and light wavelength λ, McMillan and Bishop [68] introduced the notion of plenoptic modeling with the 5D complete plenoptic function of the form $P_5(V_x, V_y, V_z, \theta, \phi)$.

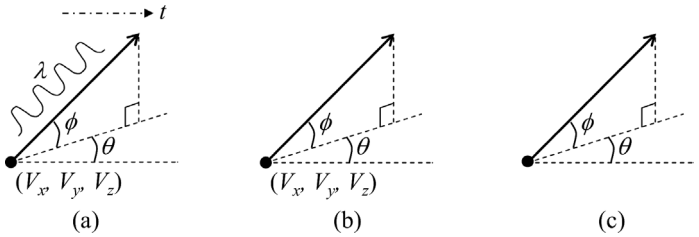


Fig. 4.1 Plenoptic functions: (a) full 7-parameter $(V_x, V_y, V_z, \theta, \phi, \lambda, t)$, (b) 5-parameter $(V_x, V_y, V_z, \theta, \phi)$, and (c) 2-parameter (θ, ϕ).

Table 4.1 A taxonomy of plenoptic functions.

Dimension	Year	View space	Name
7	1991	Free	Plenoptic function
5	1995	Free	Plenoptic modeling
4	1996	Bounding box	Lightfield/Lumigraph
3	1999	Bounding circle	Concentric mosaics
2	1994	Fixed point	Cylindrical/Spherical panorama

The simplest plenoptic function is a 2D panorama (cylindrical [18] or spherical [104]) when the viewpoint is fixed, namely $P_2(\theta, \phi)$. A regular rectilinear image with a limited field of view can be regarded as an incomplete plenoptic sample at a fixed viewpoint (Figure 4.1).

Image-based rendering, or IBR, can be viewed as a set of techniques to reconstruct a continuous representation of the plenoptic function from observed discrete samples. The issues of sampling the plenoptic function and reconstructing a continuous function from discrete samples are important research topics in IBR. As a preview, a taxonomy of plenoptic functions is shown in Table 4.1.

The cylindrical panoramas used in [68] are two-dimensional samples of the plenoptic function in two viewing directions. The two viewing directions for each panorama are panning and tilting about its center. This restriction can be relaxed if geometric information about the scene is known. In [68], stereo techniques are applied on multiple cylindrical panoramas in order to extract disparity (or inverse depth) distributions. These distributions can then be used to predict appearance (i.e., plenoptic function) at arbitrary locations. Similar work on regular stereo pairs can be found in [51], where correspondences constrained along epipolar geometry are directly used for view transfer.

4.2 Light Field and Lumigraph

It was observed in both light field rendering [53] and Lumigraph [32] systems that as long as we stay outside the convex hull (or simply a bounding box) of an object[1] and the medium is non-dispersive, we can simplify the 5D complete plenoptic function to a 4D light field plenoptic function,

$$P_4(u, v, s, t), \tag{4.1}$$

where (u, v) and (s, t) are parameters of two planes of the bounding box, as shown in Figure 4.2.

The (u, v) plane is the camera plane, where the sampling cameras are located. Figure 4.3(a) shows a visualization of the light field from the camera plane. From a point corresponding to a sampling camera location, the view is the original sampled view.

For the light field system of Levoy and Hanrahan, the (s, t) plane is the focal plane, where the scene is assumed to be located. A visualization of the light field from the focal plane is shown in Figure 4.3(b). Assuming that the surface of the scene is approximately at the focal plane, all the rays passing through a point in the focal plane are appearance samples of the same surface point from different views. This is

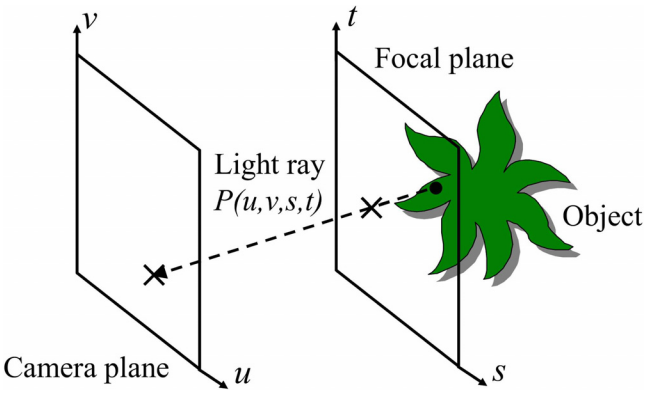


Fig. 4.2 Representation of a light field.

[1] The reverse is also true if camera views are restricted inside a convex hull.

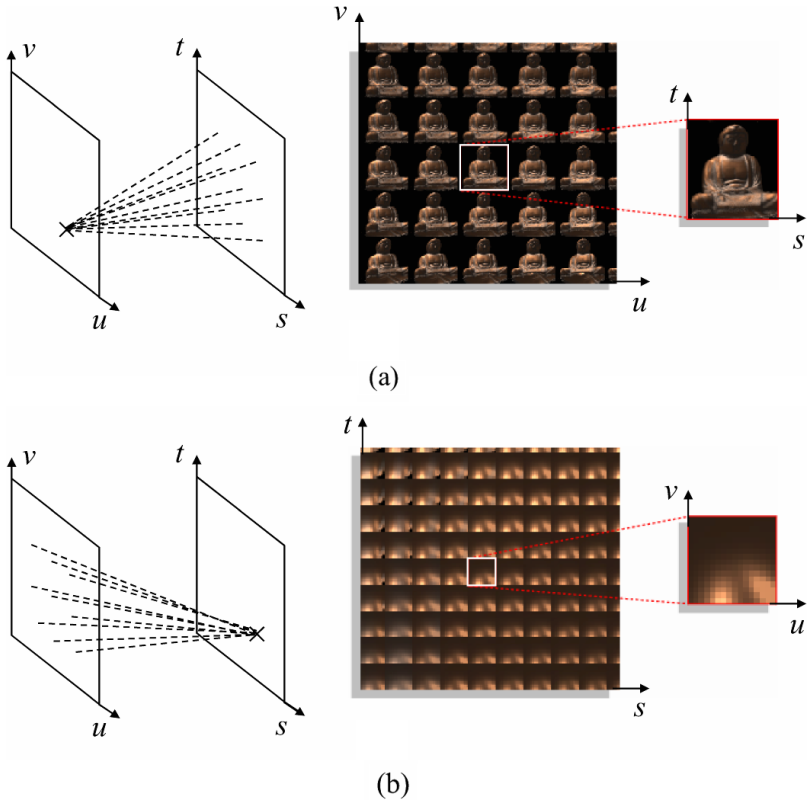


Fig. 4.3 The light field seen from (a) camera plane and (b) focal plane. The boxed subimage is observed from a single point in the parameter plane. (Images courtesy of Marc Levoy; ©1996 ACM, Inc. Included here by permission.)

akin to capturing the local radiance function associated with the scene surface for a fixed lighting condition. Rays are interpolated based on this assumption that the scene surface is close to the focal plane. More specifically, as shown in Figure 4.2, any ray passing through two planes can be indexed by the two intersection points and subsequently rendered using quadratic linear interpolation of the neighboring 16 rays. Object surfaces that are located far away from the focal plane will appear blurred at interpolated views (this will be explained in the next section). On the other hand, the Lumigraph uses an approximated 3D object surface for view interpolation, which reduces the blur problem. Note that for the Lumigraph, the (u, v) plane is the focal plane while

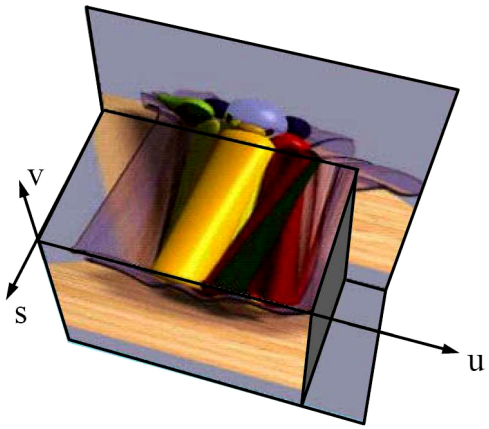


Fig. 4.4 An (s,u,v) slice of a Lumigraph. Note that the (u,v) and (s,t) parameterizations are reversed versions for the light field. (Image courtesy of Michael Cohen; ©1996 ACM, Inc. Included here by permission.)

the (s,t) is the camera plane. A visualization of a subset of the full (u,v,s,t) space for the Lumigraph is shown in Figure 4.4.

In the rest of this article, we will follow the notation of Lumigraph where (s,t) is the camera plane and (u,v) is the focal plane.

4.2.1 Sampling

A 2D subspace given by fixed values of s and t resembles an image, whereas fixed values of u and v give a hypothetical radiance function. Fixing t and v gives rise to an epipolar image, or EPI [6]. An example of a 2D light field or EPI is shown in Figure 4.5.

Let the sample intervals along s and t directions be Δs and Δt, respectively. As a result, the horizontal and vertical disparities between two grid cameras in the (s,t) plane are given by $k_1 \Delta s f/z$ and $k_2 \Delta t f/z$, respectively. Here f is the focal length of the camera, z is the depth value and $(k_1 \Delta s, k_2 \Delta t)$ is the sample interval between two grid points (s,t).

Similarly, the sample intervals along u and v directions are assumed to be Δu and Δv, respectively. A pinhole camera model is adopted to capture the light field. What a camera sees is a blurred version of the plenoptic function because of finite camera resolution. A pixel value

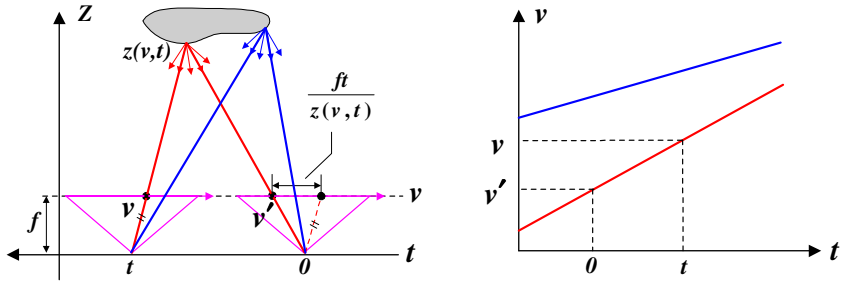


Fig. 4.5 An illustration of 2D light field or EPI. (a) a point is observed by two cameras 0 and t. (b) two lines are formed by stacking pixels captured along the camera path. Each line has a uniform color because of Lambertian assumption on object surfaces.

is a weighted integral of the illumination of the light arriving at the camera plane, or the convolution of the plenoptic function with a low-pass filter.

Let $l(u,v,s,t)$ represent the continuous light field, $p(u,v,s,t)$ the sampling pattern in light field, $r(u,v,s,t)$ the combined filtering and interpolating low-pass filter, and $i(u,v,s,t)$ the output image after reconstruction. Let L, P, R and I represent their corresponding spectra, respectively. In the spatial domain, the light field reconstruction can be computed as

$$i(u,v,s,t) = r(u,v,s,t) * [l(u,v,s,t)p(u,v,s,t)], \tag{4.2}$$

where $*$ represents the convolution operation.

In the frequency domain, we have

$$\begin{aligned} I(\Omega_u,\Omega_v,\Omega_s,\Omega_t) = {} & R(\Omega_u,\Omega_v,\Omega_s,\Omega_t)(L(\Omega_u,\Omega_v,\Omega_s,\Omega_t) \\ & * P(\Omega_u,\Omega_v,\Omega_s,\Omega_t)). \end{aligned} \tag{4.3}$$

The problem of light field reconstruction is to find a reconstruction filter $r(u,v,s,t)$ for anti-aliased light field rendering, given the sampled light field signals.

The depth function of the scene is assumed to be equal to $z(u,v,s,t)$. As shown on the left of Figure 4.5, the same 3D point is observed at v' and v in the local coordinate systems of cameras 0 and t, respectively. The disparity between the two image coordinates can be computed

easily as $v - v' = ft/z$. The right of Figure 4.5 shows an EPI image where each line represents the radiance observed from different cameras. For simplicity of analysis, the BRDF model of a real scene is assumed to be Lambertian. As a result, each line in Figure 4.5(b) has a uniform color.

Suppose the sampling is that of a rectangular lattice, and that $L_s(\Omega_u, \Omega_v, \Omega_s, \Omega_t)$ is the Fourier transform of the resulting sampled light field $l_s(u, v, s, t)$. From basic principles, $L_s(\Omega_u, \Omega_v, \Omega_s, \Omega_t)$ consists of replicas of $L(\Omega_u, \Omega_v, \Omega_s, \Omega_t)$, shifted to the 4D grid points.

These shifted spectra, or replicas, except the original one at $(0, 0, 0, 0)$, are called the alias components. If L is not bandlimited outside the Nyquist frequencies, some replicas will overlap with the others, creating aliasing artifacts.

In general, there are two ways to combat aliasing effects in output display when we render a novel image. First, we can increase the sampling rate. The higher the sampling rate, the less the aliasing effects. Indeed, uniform oversampling has been consistently employed in many IBR systems to avoid undesirable aliasing effects. However, oversampling means more effort in data acquisition and requires more storage. Though redundancy in the oversampled image database can be partially eliminated by compression, excessive samples are always wasteful.

Second, light field signals can also be made bandlimited by filtering with an appropriate filter kernel. Similar filtering has to be performed to remove the overlapping of alias components during reconstruction or rendering. The design of such a kernel is, however, related to the depth of the scene. Previous work on Lumigraph shows that approximate depth correction can significantly improve the interpolation results. The questions are: is there an optimal filter? Given the number of samples captured, how accurately should the depth be recovered? Similarly, given the depth information one can recover, how many samples can be removed from the original input?

Such questions were addressed by Chai *et al.* [14] (ignoring occlusion effects). One key observation made is that the spectral support (in frequency domain) of a light field signal is bounded by only the minimum and maximum depths, irrespective of how complicated the spectral support might be because of depth variations in the scene

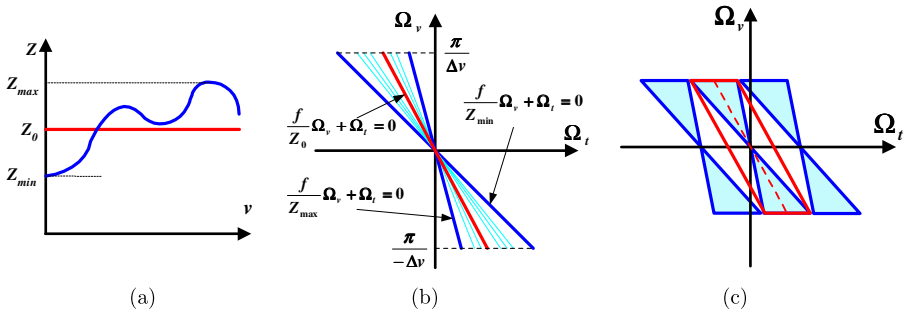


Fig. 4.6 Spectral support for light field signal with spatially varying depths: (a) a local constant depth model bounded by $z_{\min}$ and $z_{\max}$ is augmented with another depth value z_0; (b) spectral support is now bounded by two smaller regions, with the introduction of the new line of z_0; and (c) optimal depth at z_0.

(see Figures 4.6(a) and 4.6(b)). Given the minimum and maximum depths, a reconstruction filter with an optimal and constant depth can be designed to achieve anti-aliased light field rendering. More specifically, anti-aliased light field rendering can be achieved by applying the optimal filter as shown in Figure 4.6(c), where the optimal constant depth is defined as the inverse of average disparity d_0, i.e.,

$$d_0 = \frac{1}{z_0} = \frac{\left(\frac{1}{z_{\min}} + \frac{1}{z_{\max}}\right)}{2}. \tag{4.4}$$

4.2.2 Extensions

Note that in general, the (u, v) and (s, t) planes need not be parallel. There is also an implicit and important assumption that the strength of a light ray does not change along its path. For a complete description of the plenoptic function for the bounding box, six sets of such two-planes would be needed. More restricted versions of Lumigraph have also been developed by Sloan *et al.* [99] and Katayama *et al.* [49]. Here, the camera motion is restricted to a straight line.

The principles of light field rendering and Lumigraph are similar, except that the Lumigraph has the additional (approximate) object geometry for better compression and appearance prediction. For this reason, the Lumigraph belongs to the "explicit geometry" camp

(Section 7). It is discussed here because of its strong similarity with the light field.

In the light field system, a capturing rig is designed to obtain uniformly sampled images. To reduce aliasing effect, the light field is prefiltered before rendering. A vector quantization scheme is used to reduce the amount of data used in light field rendering, while achieving random access and selective decoding. On the other hand, the Lumigraph can be constructed from a set of images taken from arbitrarily placed viewpoints. A re-binning process (in this case, resampling to a regular grid using a hierarchical interpolation scheme) is therefore required. Geometric information is used to guide the choices of the basis functions. Because of the use of geometric information, the sampling density can be reduced.

The $P_4(u, v, s, t)$ two-plane parameterization is just one of many for light fields. Other types of light fields include spherical or isotropic light fields [10, 41], sphere-plane light fields [10], and hemispherically arranged light fields with geometry [59]. The issue of uniformly sampling the light field was investigated by Camahort [9]. He introduced an isotropic parameterization he calls the direction-and-point parameterization (DPP), and showed that while no parameterization is view-independent, only the DPP introduces a single bias.

Buehler *et al.* [8] extended the light field concept through a technique that uses geometric proxies (if available), handles unstructured input, and blends textures based on relative angular position, resolution, and field-of-view. They achieve real-time rendering by interpolating the blending field using a sparse set of locations.

4.3 Concentric Mosaics

Obviously, the more constraints we have on the camera location (V_x, V_y, V_z), the simpler the plenoptic function becomes. If we want to capture all viewpoints, we need a complete 5D plenoptic function. As soon as we stay in a convex hull (or conversely viewing from a convex hull) free of occluders, we have a 4D light field. If we do not translate at all, we have a 2D panorama. An interesting 3D parameterization of the plenoptic function, called Concentric Mosaics (CMs) [95],

was proposed by Shum and He; here, the sampling camera motion is constrained along concentric circles on a plane.

By constraining camera motion to planar concentric circles, CMs can be created by compositing slit images taken at different locations of each circle, as shown in Figure 4.7. Two types of CMs are shown in Figure 4.8; in the first type, rays are arranged in the *tangential* direction

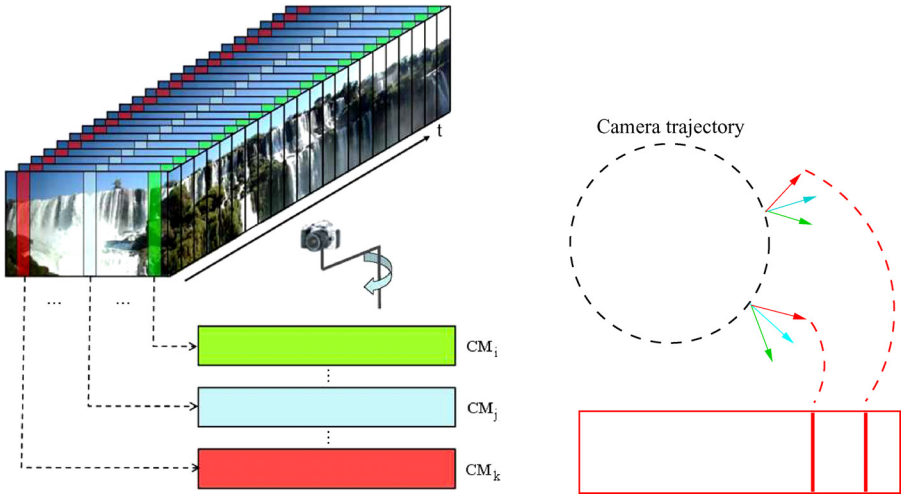


Fig. 4.7 Creation of CMs from source images. If the images are captured at regular intervals while rotated at a constant angular speed, each CM is created by just stacking the same columns from all the images in the order they are acquired. Note that the CM that consists of rays passing through the central axis of rotation is actually a (parallax-free) panorama. The left part of the figure is adapted from Figure 2.3 in [84].

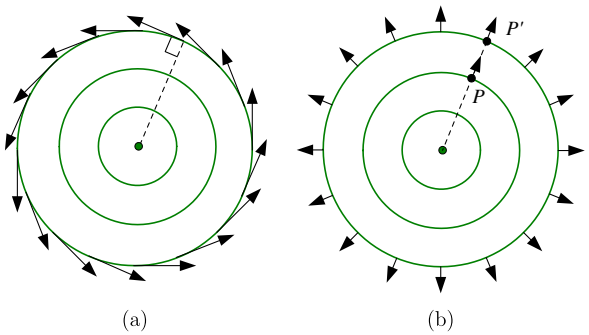


Fig. 4.8 Types of Concentric Mosaics (CMs): a plan view. A CM is assembled by unit width slit images (a) tangent to the circle; and (b) normal to the circle. We call (a) tangent CMs and (b) normal CMs. The CMs in [95] are actually tangent CMs.

(Figure 4.8(a)), and in the second type, rays are arranged in *normal* direction (Figure 4.8(b)). CMs define a 3D plenoptic function because they are sampled naturally by three parameters: rotation angle, radius, and vertical elevation. Clearly there is a one-to-one mapping between pixels in a CM and their corresponding scene points. The CMs used in [95] are actually tangent CMs; unless otherwise specified, we meant tangent CMs when we mention CMs.

Novel views are rendered by combining the appropriate captured rays in an efficient manner at rendering time. Although vertical distortions exist in the rendered images, they can be alleviated by depth correction. CMs have good space and computational efficiency. Compared with a light field or Lumigraph, CMs have much smaller file size because only a 3D plenoptic function is constructed.

Capturing CMs is almost as easy as capturing a traditional panorama except that CMs require more images. By simply spinning an off-centered camera on a rig shown in Figure 4.9, Shum and He [95] were able to construct CMs for a real scene in about 10 min. Like panoramas, CMs do not require the difficult modeling process of recovering geometric and photometric scene models. Yet CMs provide a much richer user experience by allowing the user to move freely in a circular region and observe significant parallax and lighting changes. (Parallax refers to the apparent relative change in object location within a scene due to

Fig. 4.9 Camera setup for acquiring Concentric Mosaics (CMs). The camera is counterbalanced by a weight; during image acquisition, it is rotated by a motor at a constant rotational speed.

a change in the camera viewpoint.) The ease of capturing makes CMs very attractive for many virtual reality applications.

It has been shown [95] that a novel view inside the capturing circle can be rendered from the CMs without any knowledge about the depth of the scene. Three possible techniques for resampling CMs are shown in Figure 4.10. From densely sampled CMs, a novel view image

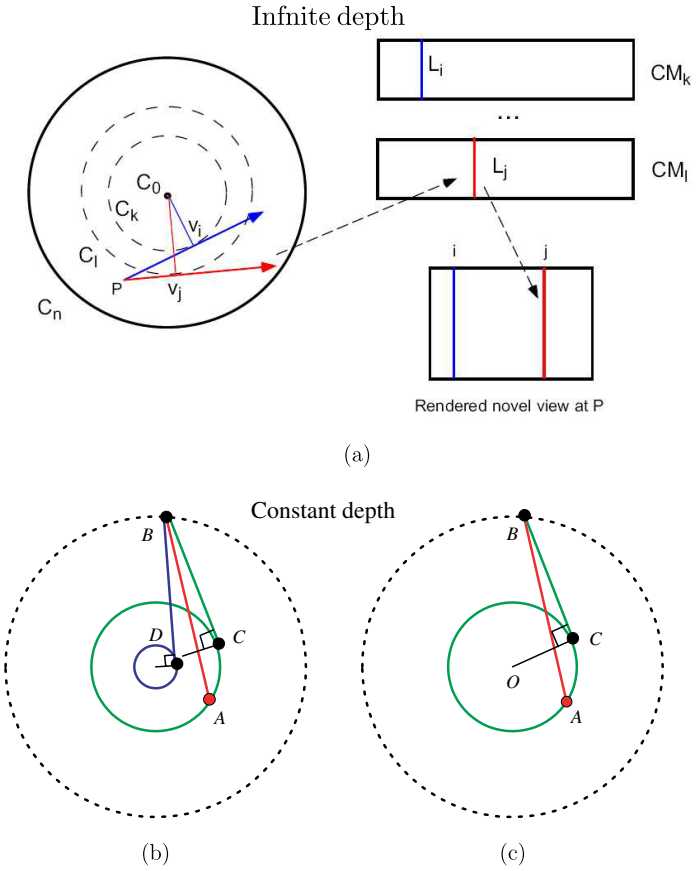


Fig. 4.10 Rendering CMs with (a) infinite depth, and (b,c) constant finite depth. (a) Rebinning: For a given ray in virtual view P (say v_j), the CM (CM_l) that is tangent to it is used. The column of pixels in CM_l that corresponds to the line of tangency with v_j is used to construct part of the new view. (b) View interpolation: A ray from viewpoint A is projected to the constant depth surface (represented as a dotted circle) at B, and interpolated by two rays BC and BD that are retrieved from neighboring CMs. (c) Local warping: A ray from viewpoint A is projected to the constant depth surface at B, and reprojected to the nearest CM by the ray BC.

can be rendered by linearly interpolating nearby rays from two neighboring CMs. In addition, a constant depth is assumed to find the best "nearby" rays for optimal rendering quality [14]. Figure 4.10(b) illustrates a rendering ray that is interpolated by two rays captured in nearby CMs. Despite the inevitable vertical distortion, CMs are very useful for wandering around (on a plane) in a virtual environment.

Rendered views of a lobby scene from captured CMs are shown in Figure 4.11. A rebinned CM at the rotation center is shown in Figure 4.11(a), while two rebinned CMs taken at exactly opposite directions are shown in Figures 4.11(b) and 4.11(c), respectively. It has also been shown in [74] that such two mosaics taken from a single rotating camera can simulate a stereo panorama. In Figure 4.11(d), strong parallax can be seen between the plant and the poster in the rendered images. More specifically, in the left image, the poster is partially obscured by the plant, while the poster and the plant do not visually overlap in the right image. This is a significant visual cue that the camera viewpoint has shifted.

Fig. 4.11 Rendering a lobby [95]: rebinned Concentric Mosaic (a) at the rotation center; (b) at the outermost circle; (c) at the outermost circle but looking at the opposite direction of (b); and (d) parallax change between the plant and the poster.

Based on a similar representation, manifold hopping [98] can be regarded as a 2.5D representation since it restricts the viewpoint on a set of discrete concentric circles. It provides user experience of comparable quality to CMs, while using significantly less data. The rendering of manifold hopping is also simplified to 2D image warping.

4.4 Multiperspective Images and Manifold Mosaics

A multiperspective image is assembled from rays captured from multiple viewpoints (e.g., [118]). Multiperspective images have also been called MCOP images [83], multiperspective panoramas [113], pushbroom images [35], and manifold mosaics [76], among other names. Let us consider the case of Peleg *et al.*'s notion of manifold mosaics. The manifold mosaic is created by projecting thin strips from images; the shape of these thin strips depend on the camera motion. More specifically, for each strip, the boundaries are perpendicular to the optic flow, and the width is proportional to the amount of motion. The basic idea is depicted in Figure 4.12, which also shows an example mosaic.

In this article, we define a *manifold mosaic* as a multiperspective image where each pixel has a one-to-one mapping with a scene point.[2] Therefore, a conventional perspective image, or a single perspective panorama, can be regarded as a degenerate manifold mosaic in which all rays are captured at the same viewpoint.

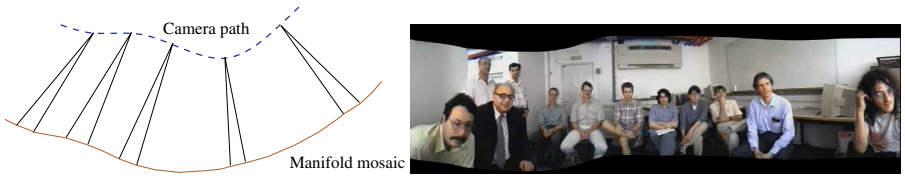


Fig. 4.12 Manifold mosaic [76]. Top: Graphical depiction of general camera motion. The wedges indicate representative parts of images used to create the manifold mosaic. Bottom: An actual manifold mosaic created using a hand-held camera. (Image (courtesy of Shmuel Peleg) is from S. Peleg and J. Herman, "Panoramic mosaics by manifold projection," in IEEE Conference on Computer Vision and Pattern Recognition, pp. 338–343, June 1997. ©1997 IEEE.)

[2] By this definition, MCOP images are *not* manifold mosaics.

We adopt the term manifold mosaic from [75] because the viewpoints are generally taken along a continuous path or a manifold (surface or curve). For example, CMs are manifold mosaics constructed from rays taken along concentric circles [95]. Note that the concept of the manifold mosaic is widely used in Manifold hopping [98].

Although many previous image-based rendering techniques (such as view interpolation and 3D warping) were developed for perspective images, they can be applied to manifold mosaics as well. For example, 3D warping has been used to reproject a multiple-center-of-projection (MCOP) image in [72, 83] where each pixel of an MCOP image has an associated depth.

4.5 Image Mosaicing

A complete plenoptic function at a fixed viewpoint can be constructed from incomplete samples. Specifically, a panoramic mosaic is constructed by registering multiple regular images. For example, if the camera focal length is known and fixed, one can project each image to its cylindrical map and the relationship between the cylindrical images becomes a simple translation. For arbitrary camera rotation, one can first register the images by recovering the camera movement, before converting to a final cylindrical/spherical map.

Many systems have been built to construct cylindrical and spherical panoramas by stitching multiple images together, e.g., [18, 62, 68, 101, 104] among others. When the camera motion is very small, it is possible to put together only small strips from registered images, i.e., slit images (e.g., [75, 118]), to form a large panoramic mosaic. Capturing panoramas is even easier if omnidirectional cameras (e.g., [69, 70]) or fisheye lens [115] are used.

Szeliski and Shum [104] presented a complete system for constructing *panoramic image mosaics* from sequences of images. Their mosaic representation associates a transformation matrix with each input image, rather than explicitly projecting all of the images onto a common surface, such as a cylinder. In particular, to construct a full view panorama, a *rotational mosaic* representation associates a rotation matrix (and optionally a focal length) with each input image.

A *patch-based alignment* algorithm is developed to quickly align two images given motion models. Techniques for estimating and refining camera focal lengths are also presented.

In order to reduce accumulated registration errors, global alignment through block adjustment is applied to the whole sequence of images, which results in an optimally registered image mosaic. To compensate for small amounts of motion parallax introduced by translations of the camera and other unmodeled distortions, a local alignment (*deghosting*) technique [97] warps each image-based on the results of pairwise local image registrations. Combining both global and local alignment significantly improves the quality of image mosaics, thereby enabling the creation of full view panoramic mosaics with hand-held cameras.

A tessellated spherical map of the full view panorama is shown in Figure 4.13. Three panoramic image sequences of a building lobby were taken with the camera on a tripod tilted at three different angles. Twenty two images were taken for the middle sequence, 22 images for the upper sequence, and 10 images for the top sequence. The camera motion covers more than two-thirds of the viewing sphere, including the top.

Apart from blending images to directly produce wider fields of view, one can use the multiple images to generate higher resolution panoramas as well (e.g., using maximum likelihood algorithms [42] or learnt image models [12]). There are also techniques to handle the exposure differences in the source image. For example, Uyttendaele *et al.* [107] perform block-based intensity adjustment to compensate for differences

Fig. 4.13 Tessellated spherical panorama covering the north pole (constructed from 54 images) [104].

in exposures. More principled techniques have been used to compensate for the exposure through radiometric self-calibration (e.g., [11,31,61]).

To produce high-quality navigation in a large environment (along a constrained set of paths), Uyttendaele *et al.* [106] capture panoramic video using Point Grey's Ladybug^{TM} 6-camera system. The resolution of each camera is 1024×768, and the capture rate was 15 fps. They mounted the Ladybug^{TM} on a tripod stand and dolly that can then be manually moved, as well as on a flattop skydiving helmet (Figures 4.14(a) and 4.14(b)). Once the images were processed to remove radial distortion and vignetting effects, they were then stitched frame by frame. The resulting panoramic video was stabilized using tracked features to provide smooth virtual navigation.

Zomet *et al.* [119] recently introduced a different way of producing mosaics called *crossed-slits projection*, or X-slits projection. What is interesting about this rendering technique is that the sampled rays passes *two* non-parallel slits, an example of which is shown at the top of Figure 4.15 (where the slits are perpendicular). The benefits are twofold: the generated mosaics appear closer to being perspective, and interesting virtual navigation can be obtained merely by changing the location of one slit. The bottom of Figure 4.15 shows examples of visualization that can be obtained through X-slits.

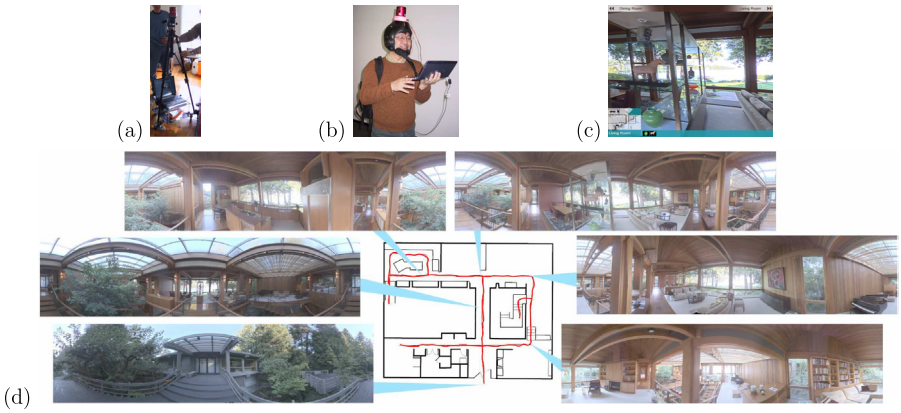


Fig. 4.14 Panoramic video [106]. (a), (b) Two versions of the capture system with Point Grey's Ladybug^{TM} 6-camera system, (c) screen shot of user interface for navigation, and (d) sample panoramas along camera path network.

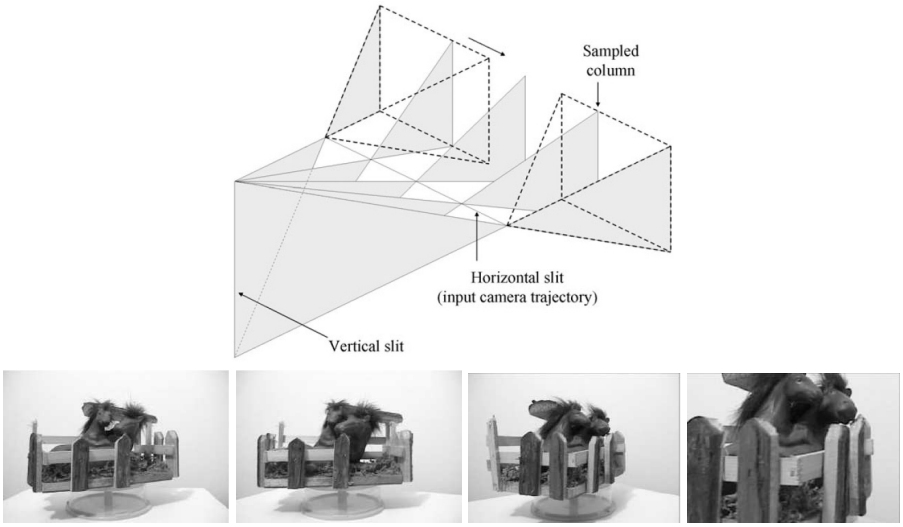


Fig. 4.15 Rendering using crossed-slits projection [119]. Top: A depiction of the idea with perpendicular slits. Bottom: The two left images are example source images of a rotating object, and the right two images are synthesized views of a virtual looming camera. (Images (courtesy of Assaf Zomet, Doron Feldman, Shmuel Peleg, and Daphna Weinshall) are from A. Zomet, D. Feldman, S. Peleg, and D. Weinshall, "Mosaicing new views: The crossed-slits projection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, no. 6, pp. 741–754, June 2003. ©2003 IEEE.)

4.6 Other Forms of Interpolation

It is worth noting that interpolation in ray space does not necessarily involve sampling only the nearest rays. It is possible to avoid artifacts such as blurring, ghosting, and pixelation due to insufficient image samples or incorrectly placed geometric proxy, by using *image priors* [30]. In this technique, the output texture is constrained by local statistics of the input images – in other words, each local patch of the output must be similar to some patch in the input image. It is formulated in a Bayesian framework where the prior is a collection of 5×5 patch samples. Unfortunately, the (iterative) energy minimization process is computationally expensive, since it requires computing similarity with all sampled patches at every step. A faster version using a coarse-to-fine strategy was later proposed [114]. However, despite the two orders of magnitude rendering speed-up, it still takes seconds to render an 800×600 frame.

In the dynamically reparameterized light field [43], rendering can take place with a larger support around the target ray to emulate a large aperture optical lens. By using this technique of synthetic aperture, varying depths of field appearance can be simulated. Objects at the desired depth can be rendered sharply with objects at other depths appearing defocused. In addition, variable focus can also be simulated by varying the focal plane. For a constant number of cameras, the rendering complexity is related to aperture size and generally is $O(N^4)$, where N is the image width in pixels. Ng [71] proposed a method to decrease the rendering complexity into $O(N^2 \log N)$ by Fourier slicing in 4D frequency space.

4.7 Hardware Rendering

Current commodity graphics hardware does not support 4D textures and quadratic interpolation. However, they usually support bilinear interpolation of 2D textures. Approaches such as [85] have been devised to exploit ways for rendering the light field using conventional graphics hardware. They typically decompose the 4D light field into a set of 2D textures, each of which represents a reference view. Multiple texture mapping is then used for interpolation. As shown in Figure 4.16, for a virtual viewpoint C, all reference view centers are projected onto the image plane of C and triangulated. The pixels inside a triangle T

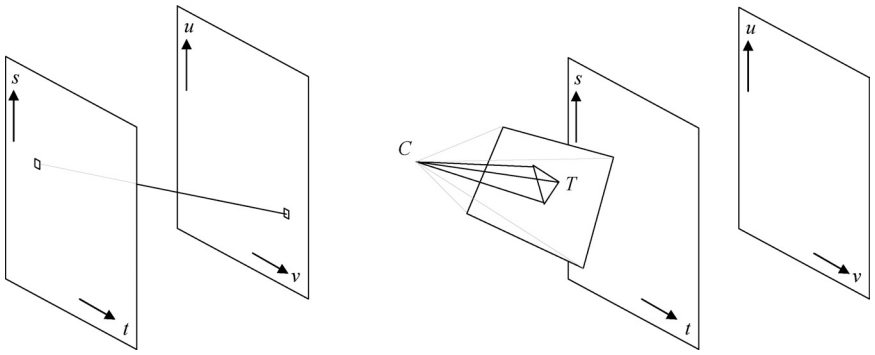


Fig. 4.16 Light field/Lumigraph rendering. Note that we used the Lumigraph (u, v, s, t) parameterization here.

are rendered using the textures of three reference views corresponding to the three vertices of the triangle. Since the blending weight can be computed using barycentric distance, it is possible to use pixel shader to composite the three textures with proper weights in a single rendering pass [96]. The composition can be done using conventional graphics hardware through multi-pass rendering [99].

Barycentric coordinates In computer graphics, barycentric coordinates are commonly used to characterize points within convex polygons. Consider a set of N points $S = \{P_1, \ldots, P_N\}$ and consider the set of all affine combinations taken from these points, i.e., $P = a_1 P_1 + \cdots + a_N P_N$. P is inside the convex hull of S if $a_1 + \cdots + a_N = 1$, with $a_i \geq 0$ for all $i = 1, \ldots, N$. The N-tuple $(a_1, \ldots, a_N)$ is the *barycentric coordinates* with respect to S. The barycentric weight or distance associated with P_i is a_i. More properties of the barycentric coordinates can be found in Section 13.7 (pages 216–221) in [21].

Fixed function vs. programmable The graphics pipeline typically consists of two stages: the vertex processing stage and pixel (fragment) processing stage. In conventional fixed-function graphics hardware, the operations in both the vertex processing stage and the pixel processing stage are fixed. The user can only change the parameters of the rendering operations. More specifically, in the vertex processing stage, the triangle vertices are first transformed into camera space before lighting is applied to the vertices. These triangles are then projected to screen space and rasterized to pixels. In the pixel processing stage, for each rasterized pixel, the color is interpolated from the colors on the triangle vertices. Texture mapping is applied to each pixel; using depth comparison to remove occluded colors, the final color is obtained by compositing unoccluded colors in the color buffer.

Recent advances in graphics hardware have enabled it to be programmable, giving rise to vertex and pixel shaders. The programmable graphics hardware allows the user to customize the rendering process at different stages of the pipeline. Vertex shaders manipulate the vertex data values, such as 3D coordinates, normals, and colors. Three-dimensional mesh deformation can be done using vertex shaders,

for example. On the other hand, pixel shaders (also known as fragment shader), affect the pixel processing stage of the graphics pipeline. They calculate effects on a per-pixel basis, e.g., texturing pixels and adding atmospheric effects. Pixel shaders often require data from vertex shaders (such as orientation at vertices or light vector) to work.

4.8 Handling Dynamic Elements in Panoramas

Early approaches for generating panoramas from rotated images do not compensate for exposure changes or moving elements in the scene. Once the relative transforms for the images have been computed, Davis [22] handles moving elements in the scene by segmenting the panorama into disjoint regions and sampling pixels in each region from a single input image. Uyttendaele *et al.* [107] cast the moving element problem as a graph, with nodes representing moving objects (i.e., objects that appear in one image but not in another). A vertex cover algorithm is then used to remove all but one instance of each object. A result of their technique can be seen in Figure 4.17; notice the dramatic improvement in the final panorama.

If the image sampling is reasonably dense enough (e.g., slowly panning a camera on a scene with quasi-repetitive motion), manifold mosaics may be used (as described in Section 4.4). However, an interesting effect may be obtained by globally stabilizing the images in time and considering slices of the resulting space–time volume as mosaics.

Fig. 4.17 Graph-based deghosting [107]. Left: Without deghosting. Right: With deghosting. (Images (courtesy of Matthew Uyttendaele) are from M. Uyttendaele, A. Eden, and R. Szeliski, "Eliminating ghosting and exposure artifacts in image mosaics," in IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 2–9, Dec. 2001. ©2001 IEEE.)

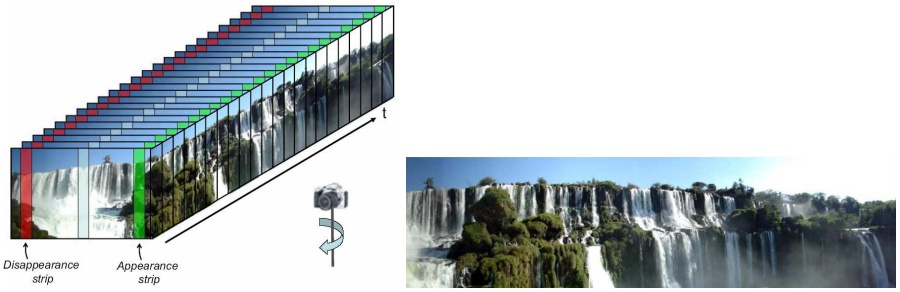


Fig. 4.18 Dynamosaic [84]. (Images (courtesy of Alex Rav-Acha and Shmuel Peleg) are from A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg, "Dynamosaics: Video mosaics with non-chronological time," in IEEE Conference on Computer Vision and Pattern Recognition, pp. 58–65, June 2005. ©2005 IEEE.)

Rav-Acha *et al.* [84] refer to such a sequence of mosaics as a dynamosaic. As a simple example shown in Figure 4.18, the camera pans from left to right. The first mosaic is constructed by taking the "appearance strip" of every image, and the last constructed using the "disappearance strip" of every image. One such mosaic is shown at the bottom of Figure 4.18. As the mosaic is played in sequence using strips shifting from "appearance strip" to "disappearance strip," the video shows a panoramic movie of the falls, with the water flowing down. The slicing scheme can be arbitrary, creating specific effects as desired. (Given a video of a swimming meet, for example, by manipulating the spatial temporal slice shapes, a swimmer can be made to appear to swim faster or slower.)

Agarwala *et al.* [2] use a different approach to produce a video panorama from similar input. Their technique is based on video textures [90], where similar frames at different times are found and used to produce new seamless video. To produce a video panorama from just a panning video, they globally register the frames and manually tag regions as static and dynamic. The basic concept of video texture is applied to the dynamic regions to ensure that the video panorama can be played indefinitely. They construct the objective function to minimize difference between static and dynamic areas that overlap and ensure local spatial consistency for hypothesized time offsets. This function is set up as an MRF and solved.

5

Rendering with Implicit Geometry

The techniques described in the previous section sample directly from the source images to produce virtual views. Relative transforms between cameras or optic flow fields are computed mainly for stabilization for panorama creation. In this section, we describe a class of techniques that relies on positional correspondences (typically across a small number of images) to render new views. This class has the term *implicit* to express the fact that geometry is not directly available; 3D information is computed only using the usual projection calculations. In certain cases where the cameras are only weakly calibrated, Euclidean 3D information is not available even with correspondence information. New views are computed based on direct manipulation of these positional correspondences, which are usually point features.

The approaches under this class are view interpolation, view morphing, joint view interpolation, and transfer methods with fundamental matrices and trifocal (or trilinear) tensors. View interpolation uses general dense optic flow to directly generate intermediate views. The intermediate view may not necessarily be geometrically correct. View morphing is a specialized version of view interpolation, except that the interpolated views are always geometrically correct. The geometric

correctness is ensured because of the linear camera motion. Transfer methods also produce geometrically correct views, except that the camera viewpoints can be arbitrarily positioned.

5.1 View Interpolation

Chen and Williams' view interpolation method [17] is capable of reconstructing arbitrary viewpoints given two input images and dense optical flow between them. This method works well when two input views are close by, so that visibility ambiguity does not pose a serious problem. Otherwise, flow fields have to be constrained so as to prevent foldovers. In addition, when two views are far apart, the overlapping parts of two images may become too small. Chen and Williams' approach works particularly well when all the input images share a common gaze direction, and the output images are restricted to have a gaze angle less than 90°.

Establishing flow fields for view interpolation can be difficult, in particular for real images. Computer vision techniques such as feature correspondence or stereo must be employed. For synthetic images, flow fields can be obtained from the known depth values (Figure 5.1).

5.2 View Morphing

From two input images, Seitz and Dyer's view morphing technique [91] reconstructs any viewpoint on the line linking two optical centers of the original cameras. Intermediate views are exactly linear combinations of two views only if the camera motion associated with the intermediate views are perpendicular to the camera viewing direction. To see this, let us assume the projection matrices for the two sampled viewpoints are Π_1 and Π_2. Without loss of generality, we can set $\Pi_0 = M_0(I_{3\times3}|\mathbf{0})$ and $\Pi_1 = M_1(I_{3\times3}|\mathbf{p})$. $I_{3\times3}$ is the 3×3 identity matrix, M_0 and M_1 are the intrinsic matrices, with

$$M_i = \begin{pmatrix} f_i & s_i & q_{xi} \\ 0 & a_i f_i & q_{yi} \\ 0 & 0 & 1 \end{pmatrix}.$$

Fig. 5.1 Video panorama [2]. Top: representative input frames. Bottom: A frame (cropped) of the extracted video panorama. (Images courtesy of Aseem Agarwala and Colin Zheng.) ©2005 ACM, Inc. Included here by permission.

f_i, a_i, s_i, (q_{xi}, q_{yi}) are the focal length, aspect ratio, skew, and principal point, respectively. $\mathbf{p} = (p_x \;\; p_y \;\; 0)^{\mathrm{T}}$ is the relative camera motion. Note that the z component of $\mathbf{p}$ is zero, which is the key to image linearity.

A given 3D point $\mathbf{x} = (X, Y, Z, 1)^{\mathrm{T}}$ is projected to $\mathbf{u}_0 = \frac{1}{Z}\Pi_0\mathbf{x}$ in view 0 and $\mathbf{u}_1 = \frac{1}{Z}\Pi_1\mathbf{x}$ in view 1. Suppose we linearly interpolate the 2D position in virtual view $\hat{I}_t$ as $\mathbf{u}_t = (1-t)\mathbf{u}_0 + t\mathbf{u}_1$, with $0 \leq t \leq 1$. Interestingly, we have $\mathbf{u}_t = \frac{1}{Z}\Pi_t\mathbf{x}$, with $\Pi_t = (1-t)\Pi_0 + t\Pi_1$ being a valid (but virtual) intermediate intrinsic matrix. As a result, for parallel source views, it is physically correct to just linearly interpolate point positions (assuming the point correspondences are valid).

If the two source images are not parallel, a pre-warp stage can be employed to rectify two input images so that corresponding scan lines are parallel. Accordingly, a post-warp stage can be used to un-rectify the intermediate images. Note that this is possible without fully calibrating the camera. Scharstein [88] extends this framework to camera motion in a plane. He assumes, however, that the camera parameters are known (Figure 5.2).

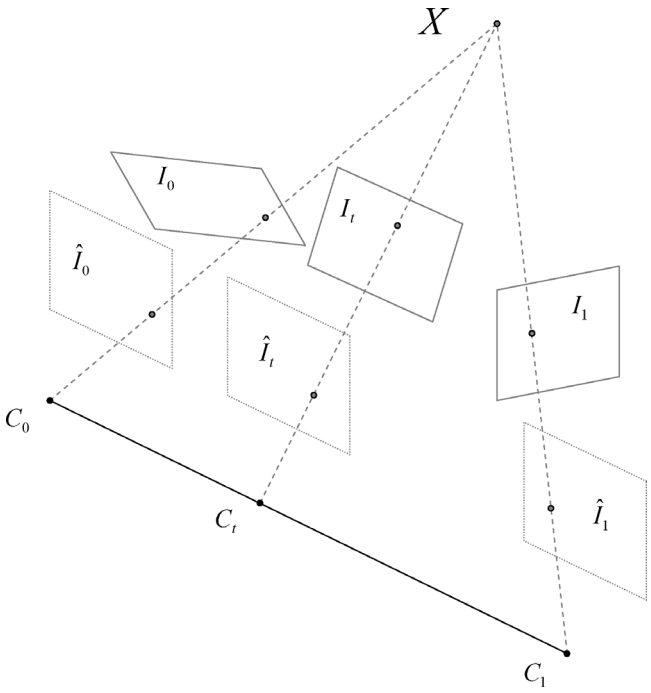


Fig. 5.2 View morphing [91]. I_0 and I_1 are the two source views at C_0 and C_1, respectively, and I_t is the synthesized view. The idea of view morphing is to rectify the source views (yielding $\hat{I}_0$ and $\hat{I}_1$), linearly interpolate (producing $\hat{I}_t$), and transform back to unrectified state (I_t). ©1996 ACM, Inc. Included here by permission.

In a more recent work, Aliaga and Carlbom [3] describe an interactive virtual walkthrough system that uses a large network of omnidirectional images taken within a 2D plane. To construct a view, the system uses the closest set of images, warps them using precomputed corresponding features, and blends the results (Figure 5.3).

5.3 Joint View Triangulation

The biggest problems associated with view interpolation are pixel matching and visibility reasoning. Visibility reasoning is especially difficult in cases where the source images are uncalibrated; as a result, there is no relative depth information to predict occlusion in new views. Lhuillier and Quan [55] proposed the idea of *joint view triangulation* (JVT) to handle these problems.

Fig. 5.3 Two examples of view morphing. Top row: Interpolation result (middle image) for two images of the same face. Bottom row: Morphing between two different faces. In both cases, point correspondences were manually established. (Images courtesy of Steve Seitz; ©1996 ACM, Inc. Included here by permission.)

There are two pre-processing steps to JVT: quasi-dense matching and planar patch construction. Quasi-dense matching consists of interest point extraction and matching (using zero-mean normalized cross-correlation). This step produces an initial list of correspondences sorted by the correlation score. This list is traversed in order, beginning with the best score, to search within the neighborhood of the point correspondence for more matches. The uniqueness constraint is used to ensure the final list consists of non-replicated points. The second step of planar patch construction assumes that the scene is piecewise smooth. It is also performed to remove possible mismatches. One of the images is subdivided into a regular patch grid; RANSAC (Random Sample Consensus) [29] is then applied to each patch to extract its homography.

Quasi-dense matching and planar patch construction are followed by the actual JVT algorithm. The basic idea of JVT is to generate Delaunay triangulations on both source images such that there is one-to-one correspondence in vertices and edges. The vertices and edges correspond to those of the precomputed patches. The patches are added raster style; they are labeled as matched or unmatched as appropriate.

Patches that have no matches are given hypothesized transforms to preserve continuity with those that have matches. View interpolation is then done by rendering unmatched patches, followed by matched patches.

Lhuillier and Quan [56] later extended their work to using epipolar geometry for more robust correspondence extraction. To overcome the restrictions of using coarse preset patches, they added edge-based partitions to better fit object boundaries. Results for an outdoor scene using their JVT algorithm can be seen in Figure 5.4.

5.4 Transfer Methods

Transfer methods (a term used within the photogrammetric community) are characterized by the use of a relatively small number of images with the application of geometric constraints (either recovered at some stage or known *a priori*) to reproject image pixels appropriately at a given virtual camera viewpoint. The geometric constraints can be of the form of known depth values at each pixel, *epipolar constraints* between pairs of images, or *trifocal/trilinear tensors* that link correspondences between triplets of images. The view interpolation and view morphing methods above are actually specific instances of transfer methods.

Using fundamental matrix Laveau and Faugeras [51] use a collection of images called reference views and the principle of the *fundamental matrix* [27] to produce virtual views. The fundamental matrix F is a 3×3 matrix of rank 2. More specifically, if $\mathbf{u}_1$ and $\mathbf{u}_2$ are two corresponding points in views 1 and 2, respectively, we have $\mathbf{u}_2^{\mathrm{T}} F_{12} \mathbf{u}_1 = 0$. Also, $\mathbf{u}_2$ lies in the *epipolar line* given by $F_{12}\mathbf{u}_1$. Another important concept is the *epipole*: All epipolar lines intersect at the epipole, and the epipole is the projection of the other camera projection center. The epipole $\mathbf{e}_{12}$ in view 2 is the null space of F_{12}, i.e., $F_{12}\mathbf{e}_{12} = 0$.

Suppose the point correspondences and fundamental matrix for a pair of images have been extracted. The virtual camera viewpoint (view 3) is specified by the user choosing two points $\mathbf{e}_{13}$ (in image 1)

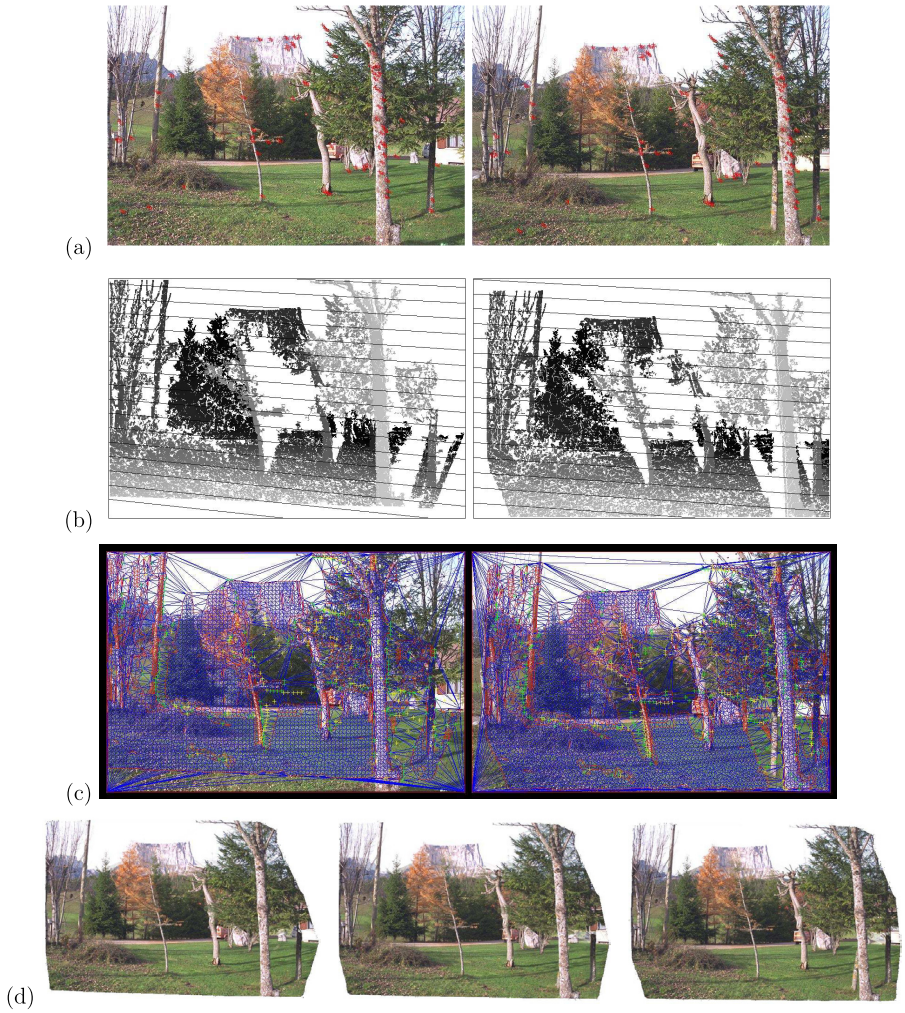


Fig. 5.4 Joint view triangulation. (a) Source views with extracted point matches. (b) Computed disparity at points with correspondence. The darker the pixel, the smaller the disparity. White pixels represent those without any correspondence. Epipolar lines (shown as dark lines) are superimposed. (c) Resulting meshes with constraint edges (in red). (d) Interpolated views. Images courtesy of Maxime Lhuillier.

and $\mathbf{e}_{23}$ (in image 2) such that $\mathbf{e}_{23}^{\mathrm{T}} F_{12} \mathbf{e}_{13} = 0$. The image plane associated with view 3 is then interactively choosen by specifying three pairs of corresponding points plus one point in one of the images. It is not necessary to manually pick the last corresponding point in the other

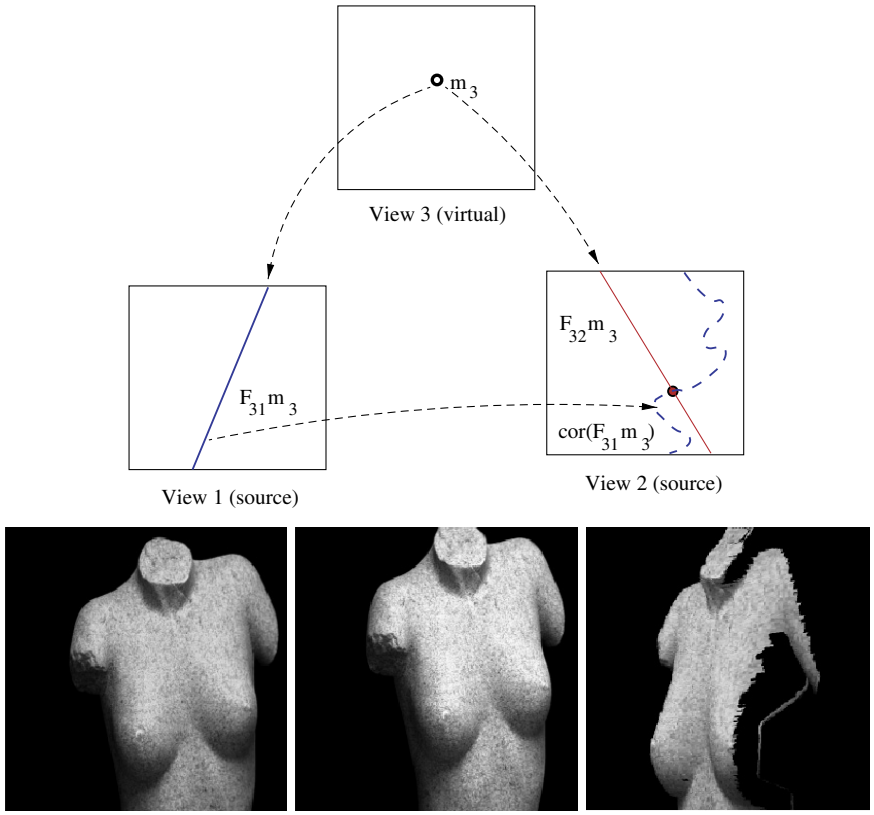


Fig. 5.5 View synthesis using the fundamental matrix [51]. Top: Process of finding the corresponding points in source views 1 and 2 that projects to point $\mathbf{m}_3$ in virtual view 3. Bottom, from left to right: two source images, and novel oblique view. (Images (courtesy of Stephane Laveau and Olivier Faugeras) are from S. Laveau and O.D. Faugeras, "3-D scene representation as a collection of images," International Conference on Pattern Recognition, vol. A, pp. 689–691, October 1994. ©1994 IEEE.)

image because it can be automatically obtained using the collinearity and epipolar constraints.

To avoid holes, the new view is computed using a reverse mapping or raytracing process, as shown at the top of Figure 5.5. For every pixel $\mathbf{m_3}$ in the new target image, a search is performed to locate the pair of image correspondences in two reference views. More specifically, for the i-th pixel $\mathbf{m}_{1i}$ along the epipolar line in view 1 (given by $F_{31}\mathbf{m}_3$), we check if its corresponding point $\mathbf{m}_{2i}$ (part of $\text{cor}(F_{31}\mathbf{m}_3)$) in

view 2 satisfies the epipolar constraint $\mathbf{m}_{2i}^{\mathrm{T}} F_{32}\mathbf{m}_3 = 0$. In other words, we search along $F_{31}\mathbf{m}_3$ until the curve $\mathrm{cor}(F_{31}\mathbf{m}_3)$ and epipolar line $F_{32}\mathbf{m}_3$ intersects. The pixel is transferred if such a point of intersection is found. Cases where no such point exists or multiple locations exist are discussed in [51]. An example result of using this technique is shown in Figure 5.5.

Note that if the camera is only weakly calibrated, the recovered viewpoint will be that of a projective structure (see [27] for more details). This is because there is a class of 3D projections and structures that will result in exactly the same source images. Since angles and areas are not preserved, the resulting viewpoint may appear warped. Knowing the internal parameters of the camera removes this problem. In a later work, Faugeras *et al.* [28] use geometric information of the scene (such as line orthogonality) to recover Euclidean structure from uncalibrated images.

Using trifocal tensor While the fundamental matrix establishes projective relationship between two rectilinear views without any knowledge of scene structure, the trifocal (or trilinear) tensor establishes a projective relationship across *three* views. The trifocal tensor is a $3 \times 3 \times 3$ matrix with a number of properties related to relationships of points and lines across three views and extraction of fundamental and projection matrices [37].

One particularly interesting property is that given a pair of point correspondences in two source images, the trifocal tensor can be used to compute the corresponding point in the third image without resorting to explicit 3D computation. Note that the 2-image epipolar search technique of [51] fails when the two epipolar lines in the virtual image are coincident (or becomes numerically unstable and sensitive to noise nearing this condition). Fortunately, the trifocal tensor avoids this degenerate case because of the flexible nature of relationships between points and lines across the three views. For example, suppose we wish to compute point $\mathbf{m}_3$ in the third view given corresponding points $\mathbf{m}_1$ and $\mathbf{m}_2$ in the first and second views, respectively, and trifocal tensor $\mathcal{T}$, with the (i, j, k)-th element indexed as $\mathcal{T}_i^{jk}$ (using the terminology

in [37]). We can find line $\mathbf{l}_2$ perpendicular to the epipolar line given by $F_{21}\mathbf{m}_2$, after which $\mathbf{m}_3$ can be determined from the relationship $(\mathbf{m}_3)^k = (\mathbf{m}_1)^i(\mathbf{l}_2)_j\mathcal{T}_i^{jk}$. $(\mathbf{m})^k$ refers to the k-th element of point $\mathbf{m}$ and $(\mathbf{l})_j$ refers to the j-th element of line $\mathbf{l}$.

The point transfer property of the trifocal tensor has been used to generate novel views from either two or three source images [4]. Here, the idea of generating novel views from two or three source images is rather straightforward. First, the "reference" trilinear tensor is computed from the point correspondences between the source images. In the case of only two source images, one of the images is replicated and regarded as the "third" image. In [4], the camera's intrinsic parameters are assumed known, which simplifies the specification of the new view. The trifocal tensor associated with the new view can be computed from the known pose change (i.e., changes in rotation and translation) with respect to the third camera location. With the trifocal tensor and correspondences across two source views known, points can then be transferred through forward mapping (i.e., transferring pixels from source to virtual views). It is not clear how visibility is handled in [4], but the modified painter's algorithm can be used without explicit depth reasoning. In addition, splatting, where a pixel in the source image is mapped to multiple pixels, can be used to remove holes in the new

Fig. 5.6 Example of visualizing using the trilinear tensor. The left-most column are the source images, with the rest synthesized at arbitrary viewpoints.

view. A set of novel views created using this approach can be seen in Figure 5.6.

Transfer methods rely on techniques such as forward mapping, modified painter's algorithm, and splatting for effective rendering. All these techniques are part of point-based rendering, which we now describe.

6

Point-Based Rendering

Point-based rendering is applied to representations that are created from 3D point clouds or 2D correspondences between reference images. Each point is usually mapped independently. Because of this flexibility, object details can be captured well. More importantly, point-based rendering is a more natural choice for data extracted using certain geometry acquisition methods such as a 3D scanner or active rangefinder, stereo reconstruction, and structure from motion techniques. Excellent surveys on point-based rendering can be found in [50, 87].

Points are mapped to the target screen through *forward mapping* or *backward mapping* (also referred to as *inverse mapping*). Referring to Figure 6.1, the mapping can be written as

$$X = C_r + \rho_r P_r x_r = C_t + \rho_t P_t x_t. \tag{6.1}$$

Here, x_t and x_r are homogeneous coordinates of the projection of 3D point X on target screen and reference images, respectively. C and P are camera center and projection matrix, respectively. ρ is a scale factor. Point-based rendering is based on (6.1); the direction of mapping depends on which 2D coordinates are evaluated.

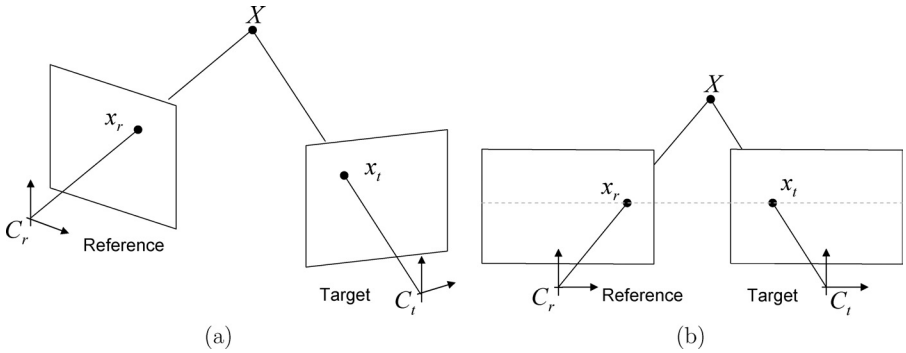


Fig. 6.1 Relationship between 2D points x_r and x_t in the reference and target images, respectively, and 3D point X. On the left shows an arbitrary virtual view while the right shows a lateral motion (with rectified geometry).

6.1 Forward Mapping

Forward mapping techniques map each pixel on the reference view(s) to the target view using some form of geometry, e.g., depth map (explicit geometry) or correspondences between views (implicit geometry). Using (6.1), we evaluate x_t:

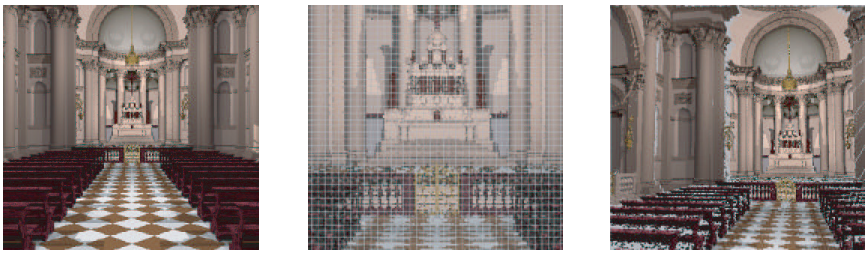
$$\rho_t x_t = P_t^{-1}(C_r - C_t) + \rho_r P_t^{-1} P_r x_r. \tag{6.2}$$

Since P_t and C_t are known, ρ_t can be computed using the depth of X with respect to target camera C_t and focus length f_t: $\rho_t = (0, 0, 1/f_t)^{\mathrm{T}} \cdot P_t^{-1}(X - C_t)$. Therefore, given x_r and ρ_r, we can compute the exact position of x_t on the target screen and transfer the color from x_r to x_t. This process is called forward mapping.

x_t is almost always at a subpixel location. If we map pixels from reference images to the target screen using the nearest neighbor scheme, gaps may appear. Unfortunately, even if x_t is exactly at a pixel location, gaps may still appear. There are two other possible reasons for the gaps or holes in the target screen: magnification and disocclusion.

A straightforward reason for the occurrence of gaps is magnification due to the virtual camera moving closer to the scene. An example of holes created this way can be seen in Figure 6.2(b).

Splatting techniques [33, 54] have been proposed to handle the subpixel target location and alleviate the gap problem caused by the larger



(a) (b) (c)

Fig. 6.2 Hole creation with forward mapping. (a) A viewpoint with no holes, (b) zoomed viewpoint with holes caused by significant changes in spatial footprint, and (c) viewpoint with holes caused mostly by depth discontinuities (and missing data).

target footprint. A filter kernel is used to cover an area larger than a pixel to make up for the expected larger image footprint for the rendered scene. The shape and size of the kernel depend on the spatial relationship between the reference and target cameras and the distance of X to the target screen. The Gaussian filter kernel is the most commonly used, and the corresponding technique is called the Elliptical Weighted Average [33].

Splatting requires a post-processing stage to normalize contributing colors and opacities at each pixel. While this can be slow using pure software implementation, a recent effort has shown that hardware acceleration is possible [7, 86], speeding up rendering by at least an order of magnitude. Unfortunately, splatting tends to blur the target image. The work on surfel rendering [77] showed how to choose the kernel to achieve necessary hole filling yet avoid over-blurring of the target image.

Gaps can also occur in the target screen if there is disocclusion caused by depth discontinuity in the scene (see Figure 6.2(c)). Such gaps cannot be filled merely by splatting because the missing pixels on the target screen are not visible from the reference view. A typical solution is to rely on other reference views to fill in the missing information. The multi-view technique of Pulli *et al.* [82] shows how multiple textured range data sets are used to generate complete views of objects.

Apart from the gap or hole problem, we also have to contend with the issue of multiple pixels from the reference view landing on the same

pixel in the target view. In this case, we need to decide which pixel or pixels to use in the final rendering. The most straightforward solution is to use the Z-buffer to make this decision. In [82], depth thresholds are used to pick the frontmost mapped pixels (whose colors are then linearly combined).

There is a more efficient rendering algorithm that obviates the need for the Z-buffer, namely the modified painter's algorithm [67]. The modified painter's algorithm uses the epipolar geometry to find the order in which pixels are scanned (Figure 6.3). This order, interestingly, is independent of the depth of the scene. To find the order, the epipole [27] $\mathbf{e}$ is first computed by projecting the camera center of the virtual view C_t onto the reference camera. If the virtual camera is behind the reference camera, we render the pixels away from $\mathbf{e}$; otherwise, we render toward $\mathbf{e}$.

In some cases, forward mapping can be simplified. For example, as shown in Figure 6.1(b), the target camera is a laterally translated version of the reference camera, so that scanlines are parallel to the camera offset $C_t - C_r$. In computer vision, the images are considered *rectified*. Here, $\rho_t = \rho_r = \rho = (0,0,1/f)^{\mathrm{T}} \cdot (X - C_r)$ and $P_t = P_r = P$.

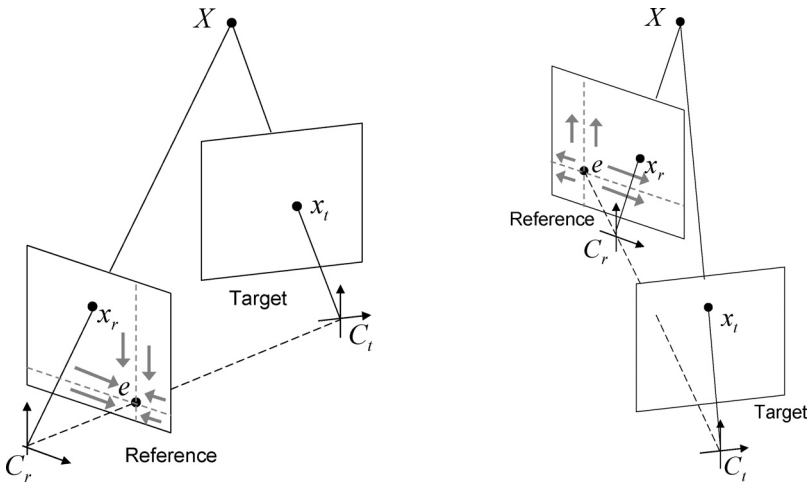


Fig. 6.3 The modified painter's algorithm can be used for forward mapping without requiring the Z-buffer. Left: Target camera is in front of reference (source) camera. Right: Target camera is behind the source camera.

As a result, (6.1) can be simplified to

$$x_t = x_r + \frac{1}{\rho} P^{-1}(C_t - C_r) = x_r + u(x_r). \tag{6.3}$$

Here, $u(x_r)$ is the disparity associated with pixel x_r, which is proportional to the depth of 3D point X. We can then easily determine the position of x_t given x_r and its depth on reference image.

Another interesting feature of this lateral-translate configuration is that only disparity $u(x_r)$ is needed for view transfer. This property is capitalized in techniques based on implicit geometry (i.e., point correspondences), such as Chen and Williams's view interpolation approach [17] and view morphing [91]. Rendering involves computing 2D pixel correspondences in the form of $x_t = x_r + u(x_r)$, without knowing any explicit 3D information. Since mapping occurs along the same scanline, rendering can also be simplified to 1D splatting. Szeliski and Cohen [103] suggested line drawing instead of splatting to fill the gaps. They also introduced a two-pass rendering method to reduce the gap filling operation; we describe this method later in this section.

6.2 Backward Mapping

In backward mapping, also known as inverse mapping, the pixel mapping relationship is found by tracing the ray from the target view back to the reference view(s). Given a pixel on the target screen x_t, we can rewrite (6.1) as

$$\rho_r x_r = P_r^{-1}(C_t - C_r) + \rho_t P_r^{-1} P_t x_t, \tag{6.4}$$

which can be further simplified to

$$x_r \equiv \mathbf{H} x_t + d\mathbf{e}. \tag{6.5}$$

Here, $\mathbf{H} = P_r^{-1} P_t$ defines the 2D planar perspective transformation (also known as a homography) from target screen plane to reference camera plane. $\mathbf{e}$ is the epipole [27], and can be obtained by intersecting the line $C_t - C_r$ with the reference view image plane. d is a scale factor and $d\mathbf{e}$ defines a line called the epipolar line (shown as l_r in Figure 6.4). This line can be obtained by intersecting the reference camera plane with the plane defined by C_r, X, and C_t (also called the epipolar plane).

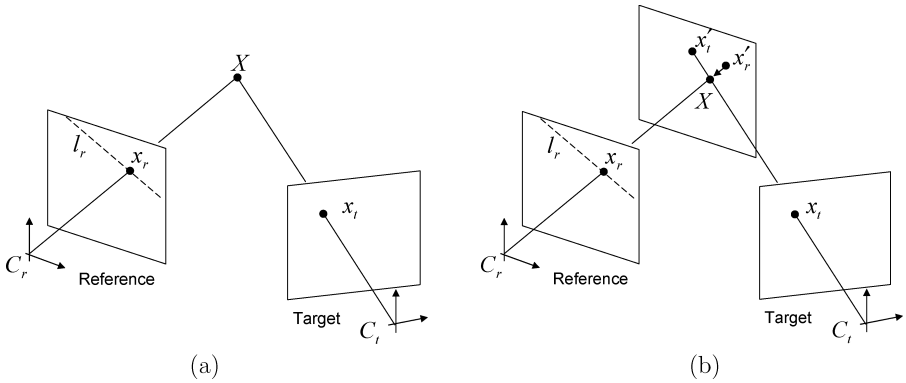


Fig. 6.4 Backward mapping from target screen to reference view.

Consequently, given x_t, x_r can be obtained by searching along the epipolar line for the pixel that fulfills (6.5) with minimum depth to target camera C_t. This process is called backward mapping or inverse mapping. Each pixel on target screen can be mapped to an unambiguous location x_r in the reference view, and can be rendered through resampling. This ensures that there are no gaps or holes in the target view. However, the search yields an invalid result if the pixel at x_t is occluded in the reference view.

In the special case where all 3D pixels are located on a 3D plane, the backward mapping process can be implemented as perspective texture mapping. Here, the mapping reduces to $x_r = \mathbf{H}'x_t$, where $\mathbf{H}'$ is defined by P_t, P_r and the location of the 3D plane. This is supported by current commodity graphics hardware and hence performed very efficiently.

In general, however, the 3D points do not lie on a plane. As a result, backward mapping involves a search for d, and is therefore typically slow. In the next section, we discuss a special case when the 3D points are reasonably close to a 3D plane, enabling a hybrid approach that uses forward mapping followed by backward mapping.

6.3 Hybrid Methods

Backward mapping involves searching and is typically slower than forward mapping, unless the 3D object is a plane, in which case backward

mapping degenerates into a perspective mapping (which is fast). On the other hand, forward mapping may be slowed down by the splatting process necessary for filling gaps and holes. When the geometry is represented as a depth field on the reference camera, forward mapping can be performed quickly with simple pixel offset and scanline-based splatting as described earlier.

The approaches in [93, 103] reformulated (6.5) as

$$x_t = \mathbf{H}'(x_r + d\mathbf{e}') = \mathbf{H}'x'_t. \tag{6.6}$$

As shown in Figure 6.4(b), rendering can be decomposed into two stages (referred to as the pre-warp stage and texture mapping stage [72]). In the pre-warp stage, x' is rendered using forward mapping from the reference view to an intermediate plane that is parallel to the reference camera plane. Since the geometry can be represented as a depth field, x'_t can be rendered quickly using 1D splatting and modified painter's algorithm as discussed above. Moreover, in order to make full use of scanline-based splatting, Oliviera *et al.* [72] proposed a two-pass pre-warp process, which forward maps from x_r to x'_t vertically and horizontally. After the pre-warp stage, the reference image is then warped to a 3D plane – which can then be very quickly mapped to the target screen using perspective mapping in the second (texture-mapping) stage. The overall performance of the hybrid two-pass technique is significantly better than traditional backward mapping.

There is a cost associated with the hybrid two-pass method: the reference images are resampled multiple times before finally rendered on the target screen. This causes the rendering result to look slightly blurrier; because of the multiple resampling, the filters used need to be more carefully designed.

6.4 Hardware Acceleration

As described above, if the 3D geometry is just a 3D plane, backward mapping reduces to perspective texture mapping, which can capitalize on the conventional graphics pipeline. In general, however, hardware acceleration is not trivial to implement for point-based render-

ing on graphics hardware. The conventional graphics pipeline can be easily used for forward mapping, except for the hole filling process. In order to fill in the gaps caused by an increase in the object footprint, each pixel from the reference image is typically rendered using a micro facet larger than a pixel's area. Some techniques [63] build tiny triangular meshes on the reference image before rendering and allow the texture mapping engine to resample the texture and subsequently fill in the gaps. However, this usually involves a large number of vertices and is not practical unless top-of-the-line accelerators are used.

Approaches to hardware-accelerated surface splatting (for general non-planar points) are similar in that they involve three rendering passes. The first pass is *visibility splatting*; here, the object is rendered without lighting to fill the depth buffer only. This is followed by the *blending pass* where colors and weights (alphas) of pixels with small depth differentials are accummulated. The final *normalization pass* involves division of the weighted sum of colors by the sum of weights, which can be implemented on the GPU (e.g., [34]).

Coconu and Hege [19] implemented a version of hardware-accelerated splatting with restricted shape and size of filter kernels. Ren *et al.* [86] implemented a hardware-accelerated version of Elliptical Weighted Average (EWA) [33] surface splatting; they represent each splat by an alpha-textured quad in the splat rasterization stage. On the other hand, Botsch *et al.* [7] use per-pixel Phong shading and a simple approximation to the EWA.

If the geometry can be represented as a 3D plane plus a small amount of depth variation (e.g., sprites with depth [93] and relief texture [72]), the hybrid mapping methods discussed in Section 6.3 can be used to take advantage of conventional hardware acceleration of the projective texture mapping. As shown in Figure 6.5, the source image is forward mapped using the depth map to an intermediate texture, which can then be fed to a conventional graphics pipeline for final backward mapping (traditional texture mapping).

With programmable graphics hardware, backward mapping using view-aligned depth fields can also be accelerated (e.g., real-time relief mapping [79]). In this approach, the points are represented as a depth

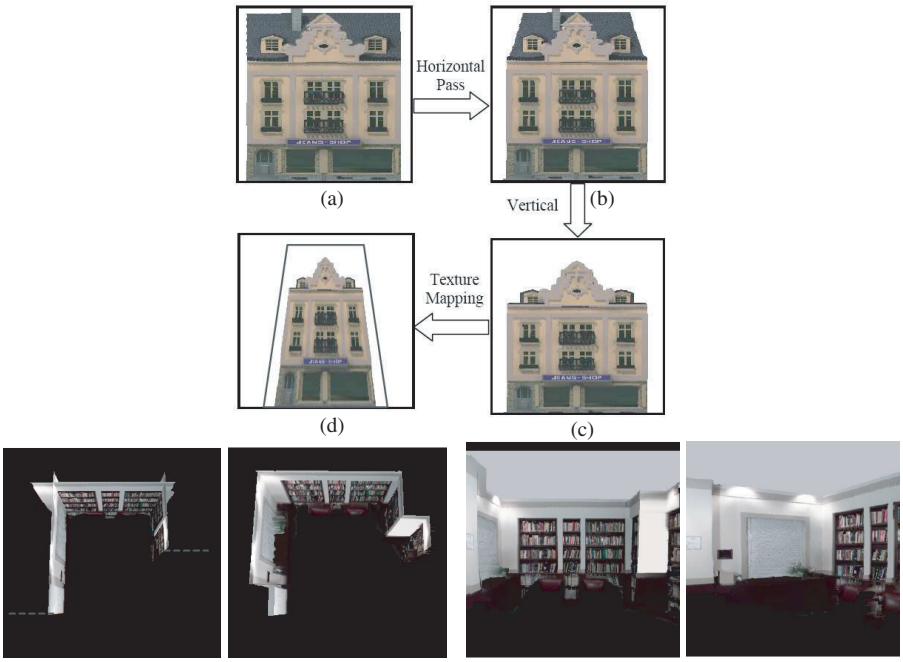


Fig. 6.5 Relief textures. (Images courtesy of Manuel Oliveira; ©2000 ACM, Inc. Included here by permission.)

map and stored as texture. For each pixel on target screen, the search along the EPI is executed in the pixel shader. After finding the point that is projected to this pixel, its texture coordinate is then used to index the point color (which is stored in color texture) for final rendering.

7

Representations with Explicit Geometry

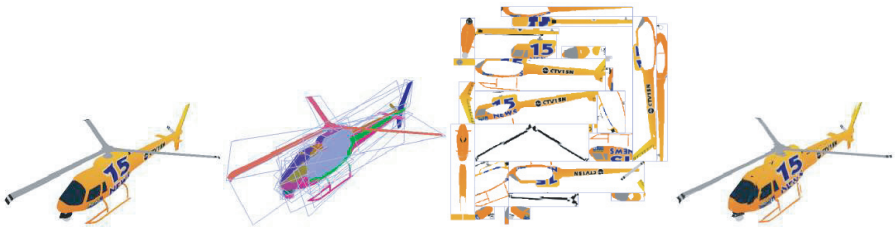
Representations that do not rely on geometry typically require a lot of images for rendering, and representations that rely on implicit geometry require accurate image registration for high-quality view synthesis. In this section, we describe IBR representations that use explicit geometry. Such representations have direct 3D information encoded in them, either in the form of depth along known lines-of-sight, or 3D coordinates. The more traditional 3D model with a single texture map is a special case in this category (not described here, since its rendering directly uses the conventional graphics pipeline).

Representations with explicit geometry include billboards, sprites, relief textures, Layered Depth Images (LDIs), and view-dependent textures and geometry. Sprites can be planar or have arbitrary depth distributions; new views are generated through 3D warping. LDIs are extensions of depth per-pixel representations, since they can encode multiple depths along a given ray. View-dependent texture mapping refers to mapping multiple texture maps to the same 3D surface, with their colors averaged using weights based on proximity of the virtual viewpoint relative to the source viewpoints.

7.1 Billboards

In games, *billboards* are often used to represent complex objects such as trees. They are either single texture-mapped rectangles that are kept fronto-parallel with respect to the viewing camera (i.e., view aligned), or sets of two rectangles arranged in a cross. Their popularity stems from the low footprint and ease of rendering (directly using the traditional graphics pipeline), but they typically work well only when viewed at a distance. The flat appearance is very pronounced when seen close up; very complex objects may appear unsatisfactory even at a reasonable distance.

To reduce these problems, Decoret *et al.* [25] proposed the use of *billboard clouds* (see Figure 7.1). A billboard cloud is just a set of textured, partially transparent billboards, with each billboard having an independent size, orientation, and texture resolution. Because a billboard cloud does not require topological information such as polygon connectivity, its format is easy to create, store, and read. Starting with a 3D model, Decoret *et al.* use an optimization approach to produce a set of representative textured planes that produce geometric errors within a specified threshold. To simplify the search, plane parameters are discretized into bins; planes are extracted sequentially by iteratively picking the bin with the minimum error. (There is the subsequent adaptive refinement in plane space – details can be found in [25].) Despite the improvements over regular billboards, billboard clouds are not intended for extreme close-ups.



(a) (b) (c) (d)

Fig. 7.1 Example of billboard cloud [25]. (a) Original 3D model, (b) locations of billboards, (c) textured billboards, and (d) view of combined billboards. (Images courtesy of Xavier Décoret; ©2003 ACM, Inc. Included here by permission.)

7.2 3D Warping

When the depth information is available for every point in one or more images, 3D warping techniques (e.g., [66]) can be used to render nearly all viewpoints. An image can be rendered from any nearby point of view by projecting the pixels of the original image to their proper 3D locations and re-projecting them onto the new picture. The most significant problem in 3D warping is how to deal with holes generated in the warped image. Holes are due to the difference of sampling resolution between the input and output images, and the disocclusion where part of the scene is seen by the output image but not by the input images. To fill in holes, splatting is used.

To improve the rendering speed of 3D warping, the warping process can be factored into a relatively simple pre-warping step and a traditional texture mapping step. The texture mapping step can be performed by standard graphics hardware. This is the idea behind relief texture, a rendering technique proposed by Oliveira and Bishop [72]. A similar factoring approach has been proposed by Shade *et al.* in a two-step algorithm [93], where the depth is first forward warped before the pixel is backward mapped onto the output image.

The 3D warping techniques can be applied not only to the traditional perspective images, but also multi-perspective images as well. For example, Rademacher and Bishop [83] proposed to render novel views by warping multiple-center-of-projection images, or MCOP images.

7.3 Layered Depth Images

To deal with the disocclusion artifacts in 3D warping, Shade *et al.* proposed Layered Depth Images, or LDIs [93], to store not only what is visible in the input image, but also what is behind the visible surface. In their paper, the LDI is constructed either using stereo on a sequence of images with known camera motion (to extract multiple overlapping layers, see Figure 7.2) or directly from synthetic environments with known geometries. In an LDI, each pixel in the input image contains a list of depth and color values where the ray from the pixel intersects with the environment.

Fig. 7.2 Layered depth image example [93]. Five source images were used to generate the layered representation of the scene using the technique in [5]. Left: Extracted layers. Middle and Right: Reconstructed views. (Images (courtesy of Richard Szeliski) from S. Baker, R. Szeliski, and P. Anandan, "A layered approach to stereo reconstruction," in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 434–441, June 1998. ©1998 IEEE.)

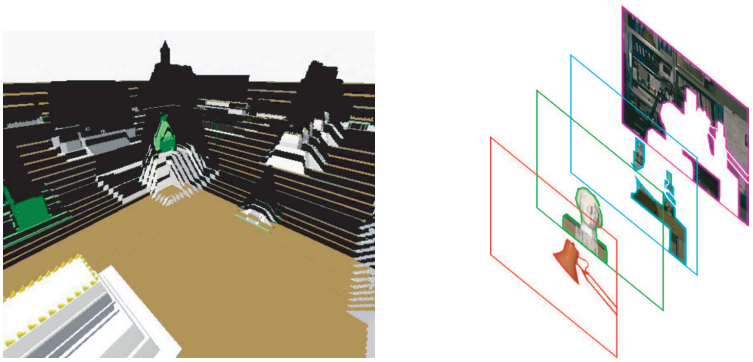
Though an LDI has the simplicity of warping a single image, it does not consider the issue of sampling density. Chang *et al.* [15] proposed LDI trees so that the sampling rates of the source images are preserved by adaptively selecting an LDI in the LDI tree for each pixel. While rendering the LDI tree, only the level of LDI tree that is comparable to the sampling rate of the output image need to be traversed.

The LDI is an example of layer representation. Before we discuss the issue of view-dependent texture mapping in Section 7.5, we describe the more general technique of rendering layers.

7.4 Layer-Based Rendering

Layered techniques usually discretize the scene into a collection of planar layers, with each layer consisting of a 3D plane with texture and optionally a transparency map. One version, the LDI, was described in the previous subsection. Two additional versions of such a representation are shown in Figure 7.3.

Compared to point-based rendering (Section 6), layer-based rendering is easier to implement using the GPU. The layers can be thought of as a discontinuous set of polygonal models, and as such, very amenable to conventional texture mapping (and to view-dependent texture mapping as well). In addition, compared to monolithic representations (Section 8), a layer-based representation is usually easier to construct since no connectivity between layers is required. The lack of connectivity



(a) (b)

Fig. 7.3 Planar layers to approximate scene geometry. (a) Planar impostors [45] (courtesy of Stefan Jeschke), (b) pop-up light field [96].

information is also a disadvantage: it can severely limit where the scene can be viewed.

Jeschke *et al.* [45] overcome the limitation in viewpoint range in two ways: use layers to represent only far-away portions of the scene (with the nearby parts being regular polygon meshes), and make the sets of layers location dependent (with the viewspace discretized into separate *view cells*). The layer representation described in [89] also adopts a view-dependent geometry solution; more specifically, when target view is too far from the current source views, it automatically generates a new set of source views with new layers. Other representations such as the pop-up light field [96] use texture synthesis to fill holes in the background layers.

Layer-based rendering usually consists of two phases. First, each layer is rendered using either point-based or monolithic rendering techniques as discussed in Sections 6 and 8, respectively. Subsequently, all rendered layers are composed in back-to-front order to produce the final view.

The painter's algorithm is often used to combine the layers, i.e., the layers are rendered from back to front relative to the target image plane. Occlusions are automatically handled this way. As with point splatting, the layers can also be rendered in an arbitrary order, with the help of the Z-buffer and A-buffer as described in [65]. This technique can be used when the order of layers relative to the target view is unknown.

The layers are rendered to the color buffer with the Z-buffer dictating occlusion between layers; meanwhile, the A-buffer accumulates the alpha weights used for normalization in the final stage.

It is worth noting that when rendering a layer with semi-transparency using multiple textures from different source views [96], the colors should be pre-multiplied by the alpha value before blending for higher computational efficiency. Wallace [108] derived a recursive blending equation in which two semi-transparent images are combined to produce another semi-transparent image. Porter and Duff [81] later simplified the recursive blending equation by substituting the original colors with colors with pre-multiplied alpha. This recursive blending equation is significant for compositing three or more layers, because it preserves associativity. In other words,

$$\text{layer}_1 \oplus (\text{layer}_2 \oplus \text{layer}_3) = (\text{layer}_1 \oplus \text{layer}_2) \oplus \text{layer}_3,$$

with $\oplus$ being the compositing operator. Rather than applying a linear operation (e.g., scaling) to each of the layers prior to compositing, it is more efficient to composite the layers *first*, followed by applying the linear operation on the composited result.

7.5 View-Dependent Texture Mapping

Texture maps are widely used in computer graphics for generating photo-realistic environments. Texture-mapped models can be created using a CAD modeler for a synthetic environment. For real environments, these models can be generated using a 3D scanner or applying computer vision techniques to captured images. Unfortunately, vision techniques are not robust enough to recover accurate 3D models. In addition, it is difficult to capture visual effects such as highlights, reflections, and transparency using a single texture-mapped model.

To obtain these visual effects of a reconstructed architectural environment, Debevec *et al.* in their Façade [24] work, used view-dependent texture mapping to render new views by warping and compositing several input images of an environment. This is the same as conventional texture mapping, except that multiple textures from different sampled viewpoints are warped to the same surface and averaged,

with weights computed based on proximity of the current viewpoint to the sampled viewpoints. A three-step view-dependent texture mapping method was also proposed later by Debevec *et al.* [23] to further reduce the computational cost and to have smoother blending. This method employs visibility preprocessing, polygon-view maps, and projective texture mapping. Porquet *et al.* [80] achieved real-time rendering by precomputing textures of three nearest viewpoints and applying pixel shading on a decimated mesh.

For the unstructured Lumigraph work, Buehler *et al.* [8] apply a more principled way of blending textures based on relative angular position, resolution, and field-of-view. Kang and Szeliski [47] use not just view-dependent textures, but *view-dependent geometries* as well. This is to account for the fact that stereo is only locally valid for scenes with non-Lambertian properties. They blend warped depth images (depth maps and textures) to produce new views, as shown in Figure 7.4.

If an object has a complex appearance (such as specular, glossy, or furry), having an accurate geometry but few images will typically be inadequate. To handle specular or glossy objects, Wood *et al.* [112] scanned highly accurate range data and acquired hundreds of images

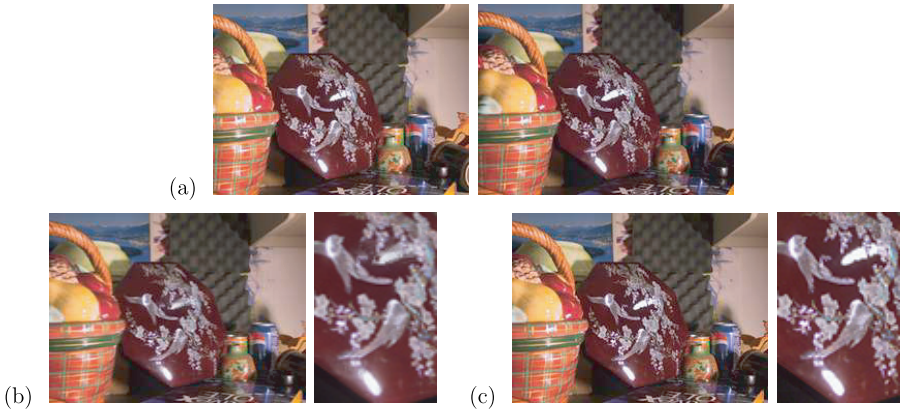


Fig. 7.4 Importance of view-dependent texture and geometry. Depth maps were extracted with the source images as reference views using the multi-view stereo technique described in [48]. (a) Source images. Notice the significant changes in the highlights. (b,c) Interpolated and actual views, respectively, with close-ups of the highlights. The highlights are a little blurred in the virtual view but resemble the actual version.

at a fixed lighting condition to create the surface light field. They used vector quantization and principal component analysis (PCA) to compress the data and represent its representation in a piecewise linear fashion. The result is a remarkably accurate visualization of the complicated object at interactive rates. Matusik *et al.* [65] model objects that cannot be scanned accurately, such as objects with fur and feathers. They take thousands of images of the object at various poses and lighting directions against a plasma display, which acts as a green screen for matting. The estimated visual hull (with opacity) is used for rendering. They also compress the data by applying PCA on each set of common viewpoints (each set with varying illumination). Interpolation is performed over the four closest views.

There are other approaches designed to handle layered reflection effects for IBR (mostly for synthetic scenes). For example, Heidrich *et al.* [39] handle reflections and refractions by decoupling geometry and illumination. This is accomplished by replacing the usual ray-color mapping with ray–ray mapping. Rendering is done by constructing this geometry light field and using it to look up the illumination from an environment map. On the other hand, Lischinski and Rappoport's [58] idea for handling non-diffuse scenes is based on layered depth images (LDIs) [5,93]. They partition the scene into diffuse (view-independent) and non-diffuse parts. The view-independent parts are represented as three orthogonal high-resolution LDIs while the non-diffuse parts are represented as view-dependent lower-resolution LDIs. Rendering is done by warping the appropriate LDIs.

In Section 9, we detail an IBR representation which we designed to handle some layered reflection effects without the use of detailed geometry. Issues associated with the difficulty of modeling layered reflection effects are also discussed. First, we discuss issues associated with rendering structures that are neither point-based nor layer-based, namely structures that are single entities, which we refer to as "monolithic geometries."

8

Monolithic Rendering

A monolithic geometry is usually represented as continuous polygon meshes with textures, which can be readily rendered using graphics hardware. This geometry can be obtained from 3D scanners, such as in those featured in the surface light field [112] work. Other sources include geometric proxies produced by interactive modeling systems (such as Façade [24]), convex hulls (e.g., visual hulls [64] and opacity hulls [65]), and reconstructed by stereo algorithms (such as joint view interpolation [55], structure from motion algorithms as used in the Lumigraph [32], unstructured Lumigraph [8], and plenoptic modeling with a hand-held camera [40]).

Rendering polygonal mesh model with textures has been well-explored. (For an excellent survey on texture mapping, see [38].) In IBR, view-dependent texture mapping (Figure 8.2) is usually necessary for photorealism. The major challenge for IBR with 3D polygonal models is in designing the compositing stage where the reference views and their blending weights have to be computed. We now describe the compositing stage for representations with implicit geometry, followed by the compositing stage for representations with explicit geometry.

IBR techniques which use explicit geometry (such as 3D surface mesh and depth maps) operate in Euclidean space. This makes spatial reasoning about rays easier: reasoning about ray and viewpoint proximity and ray-object interaction is more intuitive. As mentioned in Section 3, having explicit geometry reduces the number of input images required for high-quality view reconstruction. This explicit geometry is also known as a geometric proxy or impostor. The simplest case is when high precision geometry is available with only a limited number of input images; this is where view-dependent texture mapping is appropriate.

In general, however, the geometry used is not always accurate. As Figure 8.1(b) shows, the geometric proxy may just be a rough approximation. Chai *et al.* [14] showed that there is an inverse relationship between how accurate the geometric proxy is and how densely sampled the input images should be for alias-free view reconstruction.

Choosing the rays from the input images to render at a virtual viewpoint is based on the notion of ray proximity – we ideally want to choose rays that are "close" to the virtual ray. The proximity of rays is not only related to the input viewpoints, but also determined by the geometric proxy itself. As shown in Figure 8.1(a), a natural strategy is

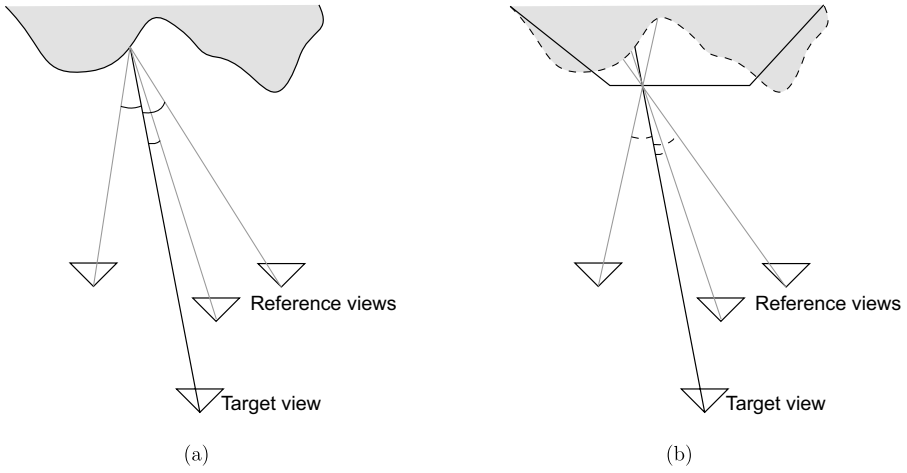


Fig. 8.1 Geometric proxy, and proximity of rays being based on angular distance at the surface of the geometric proxy.

to choose rays with smaller angle deviations with respect to the virtual ray on the geometry surface.

Other strategies for ray selection have been used. In the work of Debevec *et al.* [24], the same set of views are used to render a polygon. For each polygon, the weights are computed using the angles between the polygon normal and the viewing directions of the sampled views (the smaller the angle, the larger the weight, with a maximum of 1).

In [40], the images were obtained by moving the camera approximately within a 2D plane in a serpentine manner. Their technique for ray/view selection is to project all reference camera centers to the target camera and triangulate these points on the target screen (Figure 8.2(b)). For each pixel, the "nearest" three input views correspond to the vertices of the triangle that contains it. The blending weights assigned to these input views correspond the barycentric coordinates (see Section 4.7). View synthesis results can be seen in Figure 8.3.

The ray selection strategy of the unstructured Lumigraph [8] combines a number of properties: use of geometric proxies, epipole consistency (i.e., if a ray passes through the center of a source camera, the virtual ray can be trivially constructed), matching of resolution, virtual ray consistency (the same ray, regardless of the location of the virtual

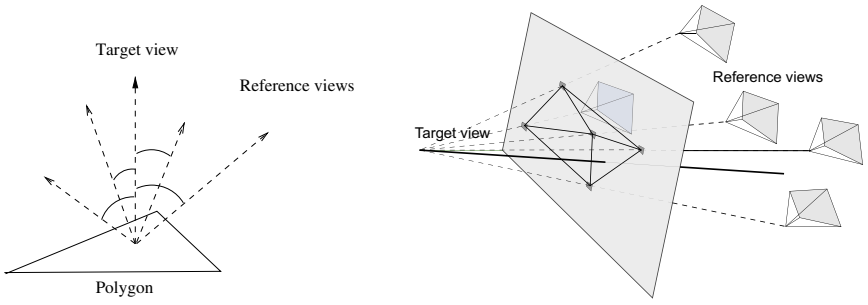


Fig. 8.2 Input view selection strategies. (a) In the work of [24], the weights associated with the input (reference) views are inversely proportional to the angle deviation. (b) In the work of [40], all the input camera centers are first projected to the target image, followed by 2D triangulation. Each triangle is associated with three input viewpoints corresponding to its vertices. These input viewpoints are the "nearest" three cameras for all the pixels that are contained within the triangle.

Fig. 8.3 Plenoptic modeling using a hand-held camera. Top left: Sample image of the scene and depiction of recovered camera poses and 3D points. Top right: View rendered using adaptive subdivision. A triangle that is considered too large is split into four triangles; the 3D locations of the mid-points of the original triangle are obtained using local depth maps. Middle row: Reduction of ghosting with refined geometry. Bottom row: Illustration of view-dependent effect. (Images courtesy of Marc Pollefeys.)

camera, should have the same set of "nearest" source cameras), and minimum angular deviation. To enable real-time rendering, Buehler *et al.* [8] compute the camera blending only at a discrete set of points in the image plane. These points are triangulated, and the dense blend weight distribution is subsequently computed through interpolation.

9

Handling Layered Reflection Effects

In this section, we describe an IBR representation to handle layered reflection effects compactly, which we call *locally reparameterized Lumigraph* (LRL).[1] The LRL is based on the use of local and separate *diffuse* and *non-diffuse geometries*. The diffuse geometry is associated with true or approximately true depth while the non-diffuse geometry has virtual depth that provide local photoconsistency with respect to its neighbors. This is similar to [58] in that there is the notion of using layers. In contrast to [58], however: (1) all local geometries are view-dependent, and (2) the rendering mechanism is different. The local geometries are used for depth correction, and do not contain radiance information. In [58], rendering is accomplished by warping LDIs. The LRL was designed to handle two common layered reflection effects: planar reflection and transparency, and specularity of low-curvature surfaces.

The concept of the LRL can be explained by first analyzing the diffuse and non-diffuse effects using the Epipolar Plane Image (EPI) [6] as a visualization tool. An EPI is basically a 3D representation (u, v, t)

[1] This name is actually inspired by the term "dynamically reparameterized Light Field" used in [43].

of a stacked sequence of camera images taken along a path, with (u, v) being the image coordinates and t being the frame index, and is therefore a 3D slice through a 4D Light Field. It has been used for stereo and multiview rendering (e.g., [36]).

9.1 Analysis Using the EPI

For the case of a laterally translated camera (along the x-direction) as shown in Figure 9.1(a), a typical EPI slice parallel to the $u - t$ plane is shown in Figure 9.1(b). In the EPI slice, we observe multiple trails corresponding to points similar in color and brightness moving across the EPI image. Trails that correspond to diffuse parts of the scene track the same points, and thus are straight. In fact, the slope of a diffuse trail k is proportional to the depth of its corresponding point. On the other hand, a specular trail does not necessarily track the same scene point; as a result, it is often curved.

Figure 9.2 shows a different common phenomenon, that of planar reflection. The flower painting is being reflected off the glass covering the Mona Lisa painting. The corresponding EPI slice shows two overlapping sets of trails. The first set corresponds to those in the Mona Lisa painting, while the second corresponds to the flower painting. The slopes for the Mona Lisa trails are linked to their depths. The slopes for the flower trails are linked to their *virtual* depths, by considering

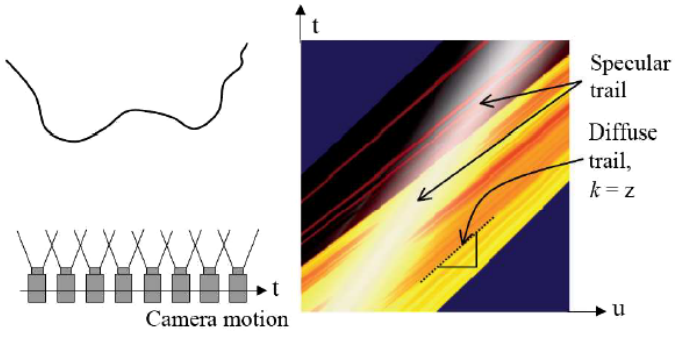


Fig. 9.1 EPI. (a) Camera configuration, (b) An EPI slice with highlights.

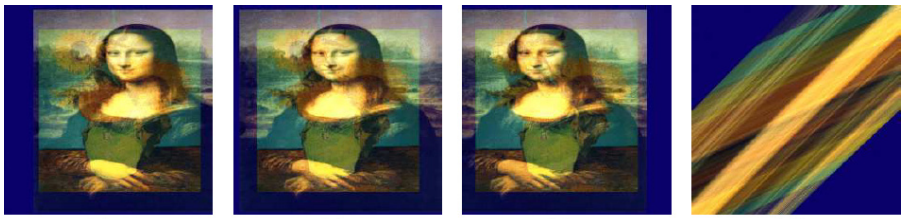


Fig. 9.2 Visualizing the planar reflection effect for an object moving from right to left. (a–c) Three snapshots of the sequence, (d) EPI of a highlight constructed by stacking middle rows of images.

the principles of optical reflection. Note that the flower trail slopes are less steep than those of the Mona Lisa counterparts, indicating that the flower painting is (at least virtually) *behind* the Mona Lisa painting.

It is thus easy to see why using a single global or even multiple view-dependent local geometries may not be adequate to handle such effects. The easiest recourse would be to sample the images more densely. However, there is a much better way: we model diffuse and non-diffuse scene components separately using what we call local diffuse and non-diffuse geometries.

9.2 Local Diffuse and Non-Diffuse Geometries

If we look within the vicinity of where a highlight or reflection occurs within an EPI slice, we notice two local slopes: one corresponding to the diffuse component, the other the non-diffuse component. We propose to represent both of them with the appropriately named local geometries to provide depth compensation for Light Field rendering. (In general, the non-diffuse trail may not stay within an EPI slice, but rather jump from slice to slice. It is more proper to say that we are tracking *3D EPI trails*, and the argument of local geometries still applies. This simple scenario is used for illustrative purposes.)

We define local geometry to be view-dependent geometry used for depth compensation only within the neighborhood of the viewing camera location. The tighter the sampling camera configuration, the smaller this neighborhood is. This is in the same spirit as [82], for example. Note that the local geometry associated with a non-diffuse area is *virtual*,

i.e., there may be no physical entity in the scene that corresponds to that area, as shown by the reflection phenomenon. *The function of local geometry, real or virtual, is to approximate the EPI trail as much as possible.* In our work, we use a fronto-parallel plane to represent our local geometry. We chose not to use stereo data of real scenes because such data tend to be unreliable in the presence of occlusions, layered reflection effects, and untextured surfaces, all of which are prevalent in our image sets. In addition, the analysis detailed in [14] showed that it is not necessary to use exact geometry for anti-aliased rendering.

In synthetic environments where the geometry, diffuse, and non-diffuse parts are known, using a single global diffuse geometry is adequate. However, for images of real scenes, it is very likely that multiple local diffuse geometries will be needed. This is to compensate for errors in camera parameters and shape, or incorrect separation of diffuse and non-diffuse components. However, it is expected that the local diffuse geometry would change much more slowly than its non-diffuse counterpart.

The implications for using two different "layers" in the form of these geometries can be seen in Figure 9.3. The analysis shown by Chai *et al.* [14] indicates that it is the depth variation and not absolute depth in the scene that dictates the sampling rate. The bigger the

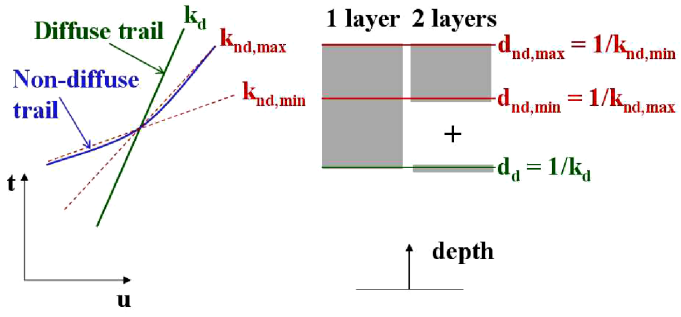


Fig. 9.3 Benefit of using separate layers. (a) Closeup view of vicinity of non-diffuse trail, (b) Depth variation (shaded regions) required to represent both diffuse and non-diffuse component using 1 and 2 layers. The dotted lines in (a) are the bounding slopes for the non-diffuse trail at the point of intersection between the diffuse and non-diffuse trails.

depth variation, the larger the sampling rate required for anti-aliased rendering. The presence of the non-diffuse component has the effect of expanding the perceived depth variation, as can be seen in Figure 9.3. If we use just a single geometry (be it view-dependent or global), then the sampling rate has to be high to accommodate the non-diffuse effect. However, if we are able to separate the non-diffuse from the diffuse, we can then compensate them separately, yielding a tighter perceived depth variation, as seen to the right of Figure 9.3(b). As a result, we can get away with a lower sampling rate for anti-aliased rendering.

9.3 Implementation

To separate the non-diffuse from the diffuse in our real scene experiment with planar reflection, we did the following:

(1) Choose the image with little or no reflection as the reference.
(2) Perform dominant motion estimation between the reference and the others. In principle, a 2D perspective or homography motion should be used. We used an affine transformation because it was adequate and had fewer parameters to estimate.
(3) Compute a *min-composite* of the registered images, since this in principle optimally removes the reflection [102]. A min-composite is extracted by taking the color associated with the *minimum* luminance (across all registered images) at each pixel.
(4) For each image, perform image difference between it and the motion-compensated min-composite to estimate the reflective components of the scene.

A big assumption here is that the reflective components are additive and that the surface is uniformly reflective. This is reasonable as long as there is no pixel intensity saturation. Portions of the reflected image do, of course, get "trimmed" (e.g., by the borders of the picture frame shown in Figure 9.4). This is one reason why a single global view of the reflection layer is inadequate, and a local view-based representation

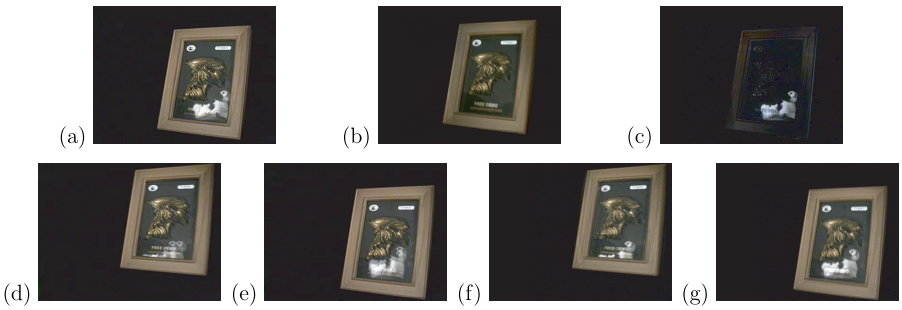


Fig. 9.4 Results for a real scene with reflection: (a) An original image, (b) Same image with only diffuse component, (c) Same image with only non-diffuse (reflection) component, (d,e) Rendering with single local depth at two different virtual camera poses, (f,g) Rendering with two local depths at the same virtual camera poses.

is preferable. For details on a more accurate means for separating the reflection component from the diffuse component, see the work of [102].

Our renderer is similar to the one described for the Lumigraph [32], i.e., it uses the two-slab, 4D parameterization of light rays. As with the Lumigraph, each rendering ray is computed based on quadrilinear interpolation of rays from the nearest four sampling cameras using the local geometry for depth compensation. After rendering each layer separately, the results are then directly added to produce the output view.

9.4 Results with Two Real Scenes

We performed two experiments involving real scenes, the first with strong reflection components and the second with highlights. Both sets are acquired using a camera attached to a vertical precision X–Y table that can accurately translate the camera to programmed positions. The first scene consists of a picture frame with a toy dog placed at an angle to it on the same side as the camera. A grid of 9×9 images, each of resolution 384×288, were captured. Figure 9.4 shows our rendering results for this image set. The rendering resolution is also 384×288. The layers are separated using the dominant motion estimation technique as descibed earlier. The rendering mechanism is exactly the same as in the previous synthetic experiment, with local

fronto-parallel planes as local geometries. These planes are prespecified in our experiments.

As can be seen from Figure 9.4(a–c), the layers have mostly been separated, with some residual errors. Despite these errors, the rendering results using the LRL representation look markedly better than using just a single geometry. The rendered reflections shown in Figure 9.4(f,g), which are the result of using two local geometries (LRL), are much sharper than those shown in Figure 9.4(d,e), which are the result of using only one local geometry. The slightly blurred frame and picture in Figure 9.4(f,g) are caused by errors in separating the layers.

In the second set, the scene captured was that of a collection of household articles, including a cup and a plate. The images were taken at positions in a 65×5 grid, and both the original and rendering resolutions are 768×576. In this case, we took two sets of images at the same camera locations; one set with the lamp switched off and another with the lamp on. The highlights are computed based on the difference between these two sets. Even with this crude means of extracting layers, we are able to generate good rendering results, as shown in Figure 9.5. The highlights as shown in Figure 9.5(f,g), which are the result of using two local geometries (LRL), are visibly crisper than those shown in Figure 9.5(d,e), which are the result of using only one local geometry.

9.5 Issues with LRL

In addition to handling the diffuse and non-diffuse components separately, another important feature of our representation is that it can handle *negative depths* to account for possible negative slopes in the EPI trail. On the other hand, the system of Lischinski and Rappoport [58], which is based on warping of LDIs, cannot accommodate negative depths explicitly.

Our current version of the LRL uses only two local "layers" or geometries. It would be reasonably straightforward to extend this representation to accommodate multiple local geometries (diffuse and multiple non-diffuse). This would be useful in handling cases where multiple non-diffuse components overlap within the captured images.

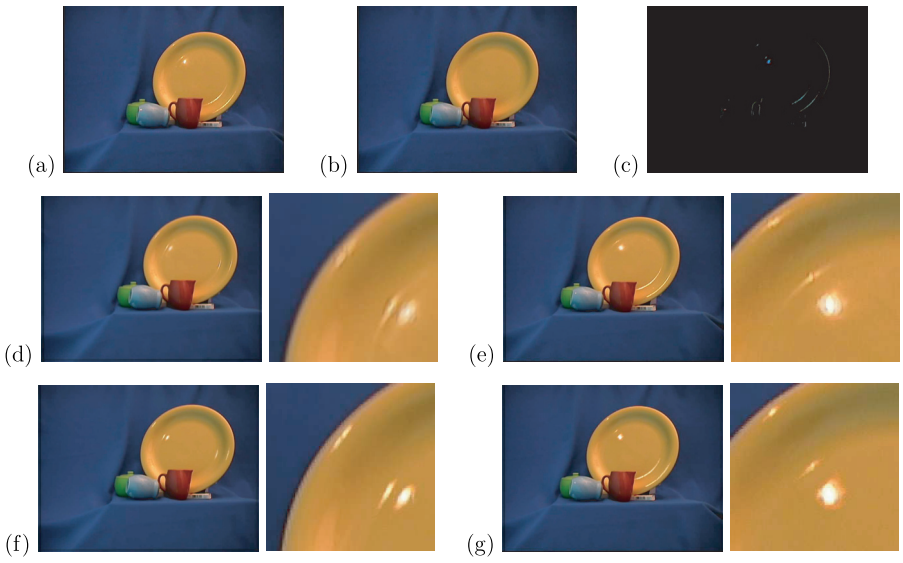


Fig. 9.5 Results for a real scene with highlights: (a) An original image, (b) Same image with only diffuse component, (c) Same image with only non-diffuse (specular) component, (d,e) Rendering with single local depth at two different virtual camera poses, (f,g) Rendering with two local depths at the same virtual camera poses. The right subimages in (d–g) are closeups of one of the highlight areas.

In practice, it is difficult to separate the diffuse and non-diffuse components completely for images of real scenes. However, we believe that partial separation is better than none at all. We have seen this to a certain extent in our experiments involving real scenes, since our separation was not perfect.

One limitation of our representation is that it is not able to adequately account for complex BRDF behavior, such as rapidly changing BRDF over the scene surface. An example would be the shimmering surface of satin. In addition, occlusions cannot be handled perfectly, as geometry within the vicinity of scene discontinuity cannot be extracted and hence represented exactly for real scenes. Furthermore, human eyes are very sensitive to edges and can detect anomalies very easily, which compounds the problem.

10

Software and Hardware Issues

Geometry is often extracted as a means for reducing the number of source images required for high-quality rendering. In addition, explicit geometry models can be efficiently rendered by conventional graphics hardware. However, the process of recovering geometry is often performed on the CPU side, which is slow.

There is an emerging field on using the GPU to perform non-graphics related computation. This field is called "general-purpose computation using graphics hardware," otherwise referred to as GPGPU (see [73] and http://gpgpu.org/). As an example, Yang *et al.* developed a hardware-accelerated version (using pixel shaders) of the plane-sweep algorithm [20] to compute depth at interactive rates with multiple cameras [116]. Woetzel and Koch [110] later extended the multi-camera stereo system to incorporate shiftable windows and view selection for occlusion handling. The technique of [116] uses a winner-take-all strategy to select the depths, which is prone to noise. Another attempt on porting stereo algorithms to the GPU is [60] where binocular stereo based on a variational deformable model has been shown to run in interactive rates.

The conventional graphics pipeline supports IBR through texture mapping, especially with multi-texture extensions. However, this support is applicable to only a subset of IBR representations, and a fair amount of work is required to fully capitalize on the capabilities of the GPU. Why do not we merely rely on the CPU? While CPU speeds are getting faster, memory access speeds remain about the same. Unfortunately, IBR techniques tend to be memory intensive – as a result, it is critical for IBR systems to have fast memory access. In addition, for a hardware system to be more "IBR-compliant," it must be capable of forward mapping. Whitted [109] discussed various IBR-related software and hardware issues, and provided an outline of a generic forward mapping processor.

11

Which Representation to Choose?

There are many factors influencing the choice of the representation to use: ease of data capture, ease of processing, rendering speed, memory footprint, database size, degrees of freedom and spatial extents of navigation, and quality of reconstruction. We discuss a subset of these factors here. (Sampling and compression issues are discussed in detail elsewhere [94].)

By definition, for IBR representations, only images are required. As we have seen, most IBR representations require additional data, be it image correspondence or geometry. In IBR approaches such as those described in [82, 112], geometry captured using a 3D scanner is used. In others (e.g., [40, 47]), stereo is used to automatically extract depth maps. Manual assistance has also been used to produce the desired morphing results (e.g., [91]) or geometry for rendering (e.g., [8, 24]).

The image capture process varies substantially from one representation to another. Custom equipment is required for light fields [53] and Concentric Mosaics (CMs) [95]. Such a requirement is relaxed for the Lumigraph [32] and the plenoptic modeling work of [40], where a hand-held camera is used. Similarly, Chai *et al.* [13] demonstrated visualizations with similar quality to CMs using images taken with only approximate circular trajectories.

View-dependent textures on global geometry are used to account for view-dependent appearance changes such as highlights and non-Lambertian behavior (e.g., [24]). However, this assumes that such a consistent global geometry can be extracted easily. This is not true for a general real scene with unknown surface properties and lighting conditions. To reduce the severity of this problem, Pulli *et al.* [82] and Kang and Szeliski [47] use view-dependent geometry as well.

For representations that are based on implicit geometry, correspondence between the source images has to be somehow obtained for view transfer. Ideally, establishing correspondence should be automatic, and indeed, techniques for this do exist [37] (though for sparse correspondence). In addition, to facilitate novel viewpoint specification, full camera calibration is required to ensure Euclidean view reconstruction. It is much less intuitive to specify a new viewpoint if cameras are only weakly calibrated; in addition, view reconstruction is only up to a projective transform. This may result in an unnatural-looking skewed scene. However, if the centers of the source cameras lie in a line (or in general within a plane), it can be shown that the disparities are linear with camera motion once the images have been rectified [91]. Regardless, *full* frame correspondence is in general a very difficult problem, especially in the presence of occlusion and non-linear effects such as highlights and transparency.

For a representation to be compelling, it has to allow a reasonably wide range of viewpoints to be selected. While most representations allow a wide selection of viewpoints, they require a substantial amount of data to be captured – this is not attractive from a practical standpoint. In addition, a certain amount of specialized knowledge about cameras is required to minimize the number of images captured. Content creators need to have some measure of understanding of complex issues such as trade-offs between field of view and resolution, type of scenes to avoid, relative placement of the camera to the scene (to avoid degenerate camera motions), and density of sampling. It is thus not surprising that despite IBR as a field having been around for some number of years, most of its representations have not been adopted for widespread commercial use. The notable exceptions are the simplest ones such as panoramas.

12

Challenges

As this article shows, IBR techniques differ in many ways, from capture and representation to rendering mechanism. Their design is highly dependent on factors such as ease of capture, number and density of source images, availability of geometry, expected accuracy of geometry, and expected scene complexity. While substantial progress has been achieved in effectively capturing, representing, and rendering scenes, many challenges remain.

The ability to handle general complex scenes remains a big issue for IBR. The easiest scenes to render remain those with mostly Lambertian surfaces. While there are techniques that can handle reflection or translucency and highlights to a certain extent (e.g., [47, 57, 100, 105] and the LRL described in Section 9), a substantial amount of work is still required to ensure robustness. The surface light field [112] handles such effects, but it requires accurate geometry and many source images. What if the scene is highly complicated, like a bush or a very cluttered office? How can we capture the surface subscattering or inter-reflection effect of an object with just images? How many images are enough? Should the new representation be view-dependent and multi-layered to account for depth, matting, and non-linear effects?

Since IBR, by definition, uses source images for rendering, interacting with IBR representations remains a challenging issue. Various limited operations were proposed: direct image editing [92] (where changes in one image are propagated to the other source images), light field morphing [117], and light field deformation [16]. Operations such as object removal and insertion are simple to implement for 3D models, but remain challenging for image-intensive representations such as the light field and Lumigraph. There is also the difficult problem of relighting real scenes using IBR representations.

IBR techniques that use transfer methods for generating virtual views tend to use a relatively small number of source images. The issues associated with the standard computer vision problems of feature selection and correspondence, occlusion handling, and structure from motion apply. Again, most techniques assume Lambertian surfaces.

There are two other important issues: rendering speed and compression. Even if all other problems associated with capturing and constructing the representation are adequately handled, the representation would still not be considered practical if it is slow to render or cannot be efficiently compressed. Realistically, the representation would need to capitalize on off-the-shelf graphics hardware for accelerated performance; in the short run, there is little commercial incentive for using specialized hardware accelerators.

Most of the IBR techniques described in this article are designed for static scenes. While photorealistic visualization of static scenes can be compelling, there is a limit on the amount of information that can be conveyed from an appearance frozen in time. The ability to rendering dynamic scenes seems more appealing. A survey on systems for capturing and rendering dynamic scenes is beyond the scope of this article, and can be found in [94].

Acknowledgments

We would like to thank Rick Szeliski for his constructive comments on this article.

References

[1] E. H. Adelson and J. R. Bergen, "The plenoptic function and elements of early vision," *Computational Models of Visual Processing*, pp. 3–20, 1991.

[2] A. Agarwala, C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic video textures," in *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, August 2005.

[3] D. G. Aliaga and I. Carlbom, "Plenoptic stitching: A scalable method for reconstructing 3D interactive walkthroughs," *Computer Graphics (SIGGRAPH)*, pp. 443–450, August 2001.

[4] S. Avidan and A. Shashua, "Novel view synthesis in tensor space," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Juan, Puerto Rico), pp. 1034–1040, June 1997.

[5] S. Baker, R. Szeliski, and P. Anandan, "A layered approach to stereo reconstruction," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Santa Barbara), pp. 434–441, June 1998.

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

[7] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's GPUs," in *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pp. 17–24, 2005.

[8] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured lumigraph rendering," in *Computer Graphics (SIGGRAPH)*, (Los Angeles, CA), pp. 425–432, August 2001.

[9] E. Camahort, "4D light-field modeling and rendering," Tech. Rep. TR01-52, The University of Texas at Austin, May 2001.

[10] E. Camahort, A. Lerios, and D. Fussell, "Uniformly sampled light fields," in *9th Eurographics Workshop on Rendering*, (Vienna, Austria), pp. 117–130, June/July 1998.

[11] F. M. Candocia, "Simultaneous homographic and comparametric alignment of multiple exposure-adjusted pictures of the same scene," *IEEE Transactions on Image Processing*, vol. 12, no. 12, pp. 1485–1494, December 2003.

[12] D. Capel and A. Zisserman, "Super-resolution from multiple views using learnt image models," in *Conference on Computer Vision and Pattern Recognition*, (Kauai, HI), vol. 2, pp. 627–634, December 2001.

[13] J.-X. Chai, S. B. Kang, and H.-Y. Shum, "Rendering with non-uniform approximate concentric mosaics," in *3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, (Dublin, Ireland), pp. 94–108, July 2000.

[14] J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum, "Plenoptic sampling," *Computer Graphics (SIGGRAPH)*, pp. 307–318, July 2000.

[15] C. Chang, G. Bishop, and A. Lastra, "LDI tree: A hierarchical representation for image-based rendering," *Computer Graphics (SIGGRAPH)*, pp. 291–298, August 1999.

[16] B. Chen, E. Ofek, H.-Y. Shum, and M. Levoy, "Interactive deformation of light fields," in *Symposium on Interactive 3D Graphics and Games (I3D)*, (Washington, D.C.), pp. 139–146, 2005.

[17] S. Chen and L. Williams, "View interpolation for image synthesis," *Computer Graphics (SIGGRAPH)*, pp. 279–288, August 1993.

[18] S. E. Chen, "QuickTime VR – An image-based approach to virtual environment navigation," *Computer Graphics (SIGGRAPH)*, pp. 29–38, August 1995.

[19] L. Coconu and H.-C. Hege, "Hardware-accelerated point-based rendering of complex scenes," in *Eurographics Workshop on Rendering*, (Aire-la-Ville, Switzerland), pp. 43–52, 2002.

[20] R. T. Collins, "A space-sweep approach to true multi-image matching," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Francisco), pp. 358–363, June 1996.

[21] H. S. M. Coxeter, *Introduction to Geometry*. John Wiley and Sons, 1969.

[22] J. Davis, "Mosaics of scenes with moving objects," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Santa Barbara, CA), pp. 354–360, June 1998.

[23] P. Debevec, Y. Yu, and G. Borshukov, "Efficient view-dependent image-based rendering with projective texture-mapping," in *Eurographics Workshop on Rendering*, pp. 105–116, 1998.

[24] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and Rendering Architecture from photographs: A hybrid geometry- and image-based approach," *Computer Graphics (SIGGRAPH)*, pp. 11–20, August 1996.

[25] X. Decoret, F. Durand, F. X. Sillion, and J. Dorsey, "Billboard clouds for extreme model simplification," *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, pp. 689–696, July 2003.

[26] P. Dutré, P. Bekaert, and K. Bala, *Advanced Global Illumination*. Natick, MA: AK Peters, 2003.

[27] O. Faugeras, *Three-dimensional Computer Vision: A Geometric Viewpoint*. Cambridge, MA: MIT Press, 1993.

[28] O. Faugeras, L. Robert, S. Laveau, G. Csurka, C. Zeller, C. Gauclin, and I. Zoghlami, "3-D reconstruction of urban scenes from image sequences," *Computer Vision and Image Understanding*, vol. 69, no. 3, pp. 292–309, March 1998.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

[30] A. Fitzgibbon, Y. Wexler, and A. Zisserman, "Image-based rendering using image-based priors," in *International Conference on Computer Vision*, vol. 2, pp. 1176–1183, 2003.

[31] D. B. Goldman and J.-H. Chen, "Vignette and exposure calibration and compensation," in *International Conference on Computer Vision*, (Beijing, China), pp. 899–906, October 2005.

[32] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *Computer Graphics (SIGGRAPH)*, (New Orleans), pp. 43–54, August 1996.

[33] N. Greene and P. S. Heckbert, "Creating raster Omnimax images from multiple perspective views using the Elliptical Weighted Average filter," *IEEE Computer Graphics and Applications*, vol. 6, no. 6, pp. 21–27, June 1986.

[34] G. Guennebaud and M. Paulin, "Efficient screen space approach for hardware accelerated surfel rendering," in *Workshop on Vision, Modeling, and Visualization*, pp. 1–10, 2003.

[35] R. Gupta and R. Hartley, "Linear pushbroom cameras," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, pp. 963–975, September 1997.

[36] M. Halle, "Multiple viewpoint rendering," *Computer Graphics (SIGGRAPH)*, pp. 243–254, July 1998.

[37] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, Second Edition, 2004.

[38] P. S. Heckbert, "Survey of texture mapping," *IEEE Computer Graphics and Applications*, vol. 11, no. 6, pp. 56–67, November 1986.

[39] W. Heidrich, H. Lensch, M. F. Cohen, and H.-P. Seidel, "Light field techniques for reflections and refractions," in *Eurographics Rendering Workshop*, pp. 195–375, June 1999.

[40] B. Heigl, R. Koch, M. Pollefeys, J. Denzler, and L. Van Gool, "Plenoptic modeling and rendering from image sequences taken by hand-held camera," in *DAGM*, pp. 94–101, 1999.

[41] I. Ihm, S. Park, and R. Lee, "Rendering of spherical light fields," in *Pacific Graphics*, (Seoul, Korea), pp. 59–68, October 1997.

[42] M. Irani and S. Peleg, "Improving resolution by image registration," *Graphical Models and Image Processing*, vol. 53, no. 3, pp. 231–239, May 1991.

[43] A. Isaksen, L. McMillan, and S. Gortler, "Dynamically reparameterized light fields," *Computer Graphics (SIGGRAPH)*, pp. 297–306, July 2000.

[44] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*. A K Peters Ltd., 2001.

[45] S. Jeschke, M. Wimmer, and H. Schumann, "Layered environment-map impostors for arbitrary scenes," in *Proceedings of Graphics Interface*, pp. 1–8, May 2002.

[46] S. B. Kang, "A survey of image-based rendering techniques," in *Videometrics VI (SPIE International Symposium on Electronic Imaging: Science and Technology)*, (San Jose, CA), vol. 3641, pp. 2–16, January 1999.

[47] S. B. Kang and R. Szeliski, "Extracting view-dependent depth maps from a collection of images," *International Journal of Computer Vision*, vol. 58, no. 2, pp. 139–163, July 2004.

[48] S. B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view stereo," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Kauai, HI), vol. I, pp. 103–110, December 2001.

[49] A. Katayama, K. Tanaka, T. Oshino, and H. Tamura, "A viewpoint dependent stereoscopic display using interpolation of multi-viewpoint images," in *Stereoscopic Displays and Virtual Reality Systems II (SPIE)*, (S. Fisher, J. Merritt, and B. Bolas, eds.), vol. 2409, pp. 11–20, 1995.

[50] L. Kobbelt and M. Botsch, "A survey of point-based techniques in computer graphics," *Computers and Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[51] S. Laveau and O. D. Faugeras, "3-D scene representation as a collection of images," in *International Conference on Pattern Recognition*, (Jerusalem, Israel), vol. A, pp. 689–691, October 1994.

[52] J. Lengyel, "The convergence of graphics and vision," *IEEE Computer*, vol. 31, no. 7, pp. 46–53, 1998.

[53] M. Levoy and P. Hanrahan, "Light field rendering," *Computer Graphics (SIGGRAPH)*, pp. 31–42, August 1996.

[54] M. Levoy and T. Whitted, "The use of points as a display primitive," Tech. Rep., UNC Technical Report 85-022, University of North Carolina, Chapel Hill, NC, 1985.

[55] M. Lhuillier and L. Quan, "Image interpolation by joint view triangulation," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Fort Collins, CO), vol. 2, pp. 139–145, June 1999.

[56] M. Lhuillier and L. Quan, "Image-based rendering by joint view triangulation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 11, pp. 1051–1063, November 2003.

[57] S. Lin, Y. Li, S. B. Kang, X. Tong, and H.-Y. Shum, "Simultaneous separation and depth recovery of specular reflections," in *European Conference on Computer Vision*, (Copenhagen, Denmark), vol. 3, pp. 210–224, May/June 2002.

[58] D. Lischinski and A. Rappoport, "Image-based rendering for non-diffuse synthetic scenes," in *Eurographics Rendering Workshop*, pp. 301–314, June 1998.

[59] M. Magnor and B. Girod, "Model-based coding of multi-viewpoint imagery," in *SPIE Visual Communication and Image Processing*, (Perth, Australia), vol. 4067(2), pp. 14–22, June 2000.

[60] J. Mairal and R. Keriven, "A GPU implementation of variational stereo," Tech. Rep. Research Report 05-13, CERTIS, November 2005.

[61] S. Mann, "Pencigraphy with AGC: Joint parameter estimation in both domain and range of functions in same orbit of the projective-Wyckoff group," in *International Conference on Image Processing*, (Los Alamitos, CA), vol. 3, pp. 193–196, 1996.

[62] S. Mann and R. W. Picard, "Virtual bellows: Constructing high-quality images from video," in *International Conference on Image Processing*, (Austin, TX), vol. I, pp. 363–367, November 1994.

[63] W. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D Warping," in *Symposium on I3D Graphics*, pp. 7–16, April 1997.

[64] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan, "Image-based visual hulls," *Computer Graphics (SIGGRAPH)*, pp. 369–374, July 2000.

[65] W. Matusik, H. Pfister, P. Ngan, P. Beardsley, R. Ziegler, and L. McMillan, "Image-based 3D photography using opacity hulls," *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, pp. 427–437, July 2002.

[66] L. McMillan, "An image-based approach to three-dimensional computer graphics," Tech. Rep., Ph.D. Dissertation, UNC Computer Science TR97-013, 1999.

[67] L. McMillan and G. Bishop, "Head-tracked stereoscopic display using image warping," in *Stereoscopic Displays and Virtual Reality Systems II (SPIE)*, pp. 21–30, February 1995.

[68] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," *Computer Graphics (SIGGRAPH)*, pp. 39–46, August 1995.

[69] V. S. Nalwa, "A true omnidirectional viewer," Tech. Rep., Bell Laboratories, Holmdel, NJ, February 1996.

[70] S. K. Nayar, "Catadioptric omnidirectional camera," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Juan, Puerto Rico), pp. 482–488, June 1997.

[71] R. Ng, "Fourier slice photography," *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, vol. 24, no. 3, pp. 735–744, July 2005.

[72] M. M. Oliveira, G. Bishop, and D. McAllister, "Relief texture mapping," in *Computer Graphics (SIGGRAPH)*, (New Orleans, LA), pp. 359–368, July 2000.

[73] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics, State of the Art Reports*, pp. 21–51, August 2005.

[74] S. Peleg and M. Ben-Ezra, "Stereo panorama with a single camera," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Fort Collins, CO), pp. 395–401, June 1999.

[75] S. Peleg and J. Herman, "Panoramic mosaics by manifold projection," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Juan, Puerto Rico), pp. 338–343, June 1997.

[76] S. Peleg, B. Rousso, A. Rav-Acha, and A. Zomet, "Mosaicing on adaptive manifolds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1144–1154, October 2000.

[77] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Computer Graphics (SIGGRAPH)*, pp. 335–342, July 2000.

[78] M. Pharr and G. Humphreys, *Physically Based Rendering*. Morgan Kaufmann, 2004.

[79] F. Policarpo, M. M. Oliveira, and J. L. D. Comba, "Real-time relief mapping on arbitrary polygonal surfaces," in *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, April 2005.

[80] D. Porquet, J.-M. Dischler, and D. Ghazanfarpour, "Real-time high quality view-dependent texture mapping Using per-pixel visibility," in *International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, November/December 2005.

[81] T. Porter and T. Duff, "Compositing digital images," in *Computer Graphics (SIGGRAPH)*, pp. 253–259, July 1984.

[82] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle, "View-based rendering: Visualizing real objects from scanned range and color data," in *Eurographics Workshop on Rendering*, (St. Etienne, France), June 1997.

[83] P. Rademacher and G. Bishop, "Multiple-center-of-projection images," in *Computer Graphics (SIGGRAPH)*, (Orlando, FL), pp. 199–206, July 1998.

[84] A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg, "Dynamosaicing: Video mosaics with non-chronological time," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Diego, CA), pp. 58–65, June 2005.

[85] M. J. P. Regan, G. S. P. Miller, S. M. Rubin, and C. Kogelnik, "A real-time low-latency hardware light-field renderer," in *Computer Graphics (SIGGRAPH)*, (Los Angeles, CA), pp. 287–290, August 1999.

[86] L. Ren, H. Pfister, and M. Zwicker, "Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering," *Eurographics, Computer Graphics Forum*, vol. 21, no. 3, pp. 461–470, 2002.

[87] M. Sainz and R. Pajarola, "Point-based rendering techniques," *Computers and Graphics*, vol. 28, no. 6, pp. 869–879, 2004.

[88] D. Scharstein, "Stereo vision for view synthesis," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Francisco, CA), pp. 852–857, June 1996.

[89] G. Schaufler, "Per-object image warping with layered impostors," in *Eurographics Workshop on Rendering*, pp. 145–156, June/July 1998.

[90] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa, "Video textures," in *Computer Graphics (SIGGRAPH)*, (New Orleans, LA), pp. 489–498, July 2000.

[91] S. M. Seitz and C. M. Dyer, "View morphing," in *Computer Graphics (SIGGRAPH)*, (New Orleans, LA), pp. 21–30, August 1996.

[92] S. M. Seitz and K. N. Kutulakos, "Plenoptic image editing," in *International Conference on Computer Vision*, pp. 17–24, 1998.

[93] J. Shade, S. Gortler, L.-W. He, and R. Szeliski, "Layered depth images," in *Computer Graphics (SIGGRAPH)*, (Orlando), pp. 231–242, July 1998.

[94] H.-Y. Shum, S.-C. Chan, and S. B. Kang, *Image-Based Rendering*. Springer, 2006.

[95] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," in *Computer Graphics (SIGGRAPH)*, (Los Angeles), pp. 299–306, August 1999.

[96] H.-Y. Shum, J. Sun, S. Yamazaki, Y. Li, and C. K. Tang, "Pop-up light field: An interactive image-based modeling and rendering system," *ACM Transactions on Graphics*, vol. 23, no. 2, pp. 143–162, April 2004.

[97] H.-Y. Shum and R. Szeliski, "Construction and refinement of panoramic mosaics with global and local alignment," in *International Conference on Computer Vision*, (Bombay, India), pp. 953–958, January 1998.

[98] H.-Y. Shum, L. Wang, J.-X. Chai, and X. Tong, "Rendering by manifold hopping," *International Journal of Computer Vision*, vol. 50, no. 2, pp. 185–201, 2002.

[99] P. P. Sloan, M. F. Cohen, and S. J. Gortler, "Time critical lumigraph rendering," in *Symposium on Interactive 3D Graphics*, (Providence, RI), pp. 17–23, April 1997.

[100] R. Swaminathan, S. B. Kang, R. Szeliski, A. Criminisi, and S. K. Nayar, "On the motion and appearance of specularities in image sequences," in *European Conference on Computer Vision*, (Copenhagen, Denmark), vol. 1, pp. 508–523, May/June 2002.

[101] R. Szeliski, "Video mosaics for virtual environments," *IEEE Computer Graphics and Applications*, pp. 22–30, March 1996.

[102] R. Szeliski, S. Avidan, and P. Anandan, "Layer extraction from multiple images containing reflections and transparency," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Hilton Head Island, NC), pp. 246–253, June 2000.

[103] R. Szeliski and M. Cohen, "Sprites with depth–fast rendering techniques for sprites with depth offsets," Tech. Rep., Microsoft Research Vision Technology Group, Technical Note No. 5, 1998.

[104] R. Szeliski and H.-Y. Shum, "Creating full view panoramic image mosaics and environment maps," *Computer Graphics (SIGGRAPH)*, pp. 251–258, August 1997.

[105] Y. Tsin, S. B. Kang, and R. Szeliski, "Stereo matching with reflections and translucency," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Madison, WI), vol. 1, pp. 702–709, June 2003.

[106] M. Uyttendaele, A. Criminisi, S. B. Kang, S. Winder, R. Hartley, and R. Szeliski, "High-quality image-based interactive exploration of real-world environments," *IEEE Computer Graphics and Applications*, vol. 24, no. 3, pp. 52–63, May/June 2004.

[107] M. Uyttendaele, A. Eden, and R. Szeliski, "Eliminating ghosting and exposure artifacts in image mosaics," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 509–516, December 2001.

[108] B. Wallace, "Merging and transformation of raster images for cartoon animation," in *Computer Graphics (SIGGRAPH)*, (Dallas, TX), pp. 253–262, 1981.

[109] T. Whitted, "Overview of IBR: Software and hardware issues," in *International Conference on Image Processing*, (Vancouver, Canada), vol. 2, p. 14, September 2000.

[110] J. Woetzel and R. Koch, "Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling," in *1st European Conference on Visual Media Production (CVMP)*, (London, UK), March 2004.

[111] T. Wong, P. Heng, S. Or, and W. Ng, "Image-based rendering with controllable illumination," in *Eurographics Workshop on Rendering*, (St. Etienne, France), pp. 13–22, June 1997.

[112] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle, "Surface light fields for 3D photography," in *Computer Graphics (SIGGRAPH), Annual Conference Series*, (New Orleans, LA), pp. 287–296, July 2000.

[113] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin, "Multiperspective panoramas for cel animation," in *Computer Graphics (SIGGRAPH)*, (Los Angeles, CA), pp. 243–250, August 1997.

[114] O. Woodford and A. Fitzgibbon, "Fast image-based rendering using hierarchical image-based priors," in *British Machine Vision Conference*, (Oxford, UK), September 2005.

[115] Y. Xiong and K. Turkowski, "Creating image-based VR using a self-calibrating fisheye lens," in *IEEE Conference on Computer Vision and Pattern Recognition*, (San Juan, Puerto Rico), pp. 237–243, June 1997.

[116] R. Yang, G. Welch, and G. Bishop, "Real-time consensus-based scene reconstruction using commodity graphics hardware," in *Pacific Graphics*, (Beijing, China), pp. 225–234, 2002.

[117] Z. Zhang, L. Wang, B. Guo, and H.-Y. Shum, "Feature-based light field morphing," *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, pp. 457–464, July 2002.

[118] J. Y. Zheng and S. Tsuji, "Panoramic representation for route recognition by a mobile robot," *International Journal of Computer Vision*, vol. 9, pp. 55–76, 1992.

[119] A. Zomet, D. Feldman, S. Peleg, and D. Weinshall, "Mosaicing new views: The crossed-slits projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 6, pp. 741–754, June 2003.