Vision for Graphics

Sing Bing Kang
Interactive Visual Media Group
Microsoft Research, Redmond, WA, USA

Abstract

Creating compelling-looking content using conventional graphics techniques is often laborious and requires significant artistry and experience. Two related issues that we've been investigating are how this content-creation process can be simplified without sacrificing quality, and how rendering can be made photorealistic or compelling. In this paper, we describe how images and videos can be processed using computer vision techniques to produce a variety of outputs, such as higher dynamic range video, realistic-looking interpolated views of dynamic scenes, 3D models of plants and trees, and rain. We also describe how flash/no-flash image pairs can be easily used to produce mattes of foreground objects and how photos of paintings can be analyzed to facilitate animation.

1. Introduction

Graphics has come a long way—there are major advancements in areas such as solid modeling, illumination modeling, surface property modeling (bidirection reflectance distribution function or BRDF, surface subscattering), non-photorealistic rendering, and hardware. Despite all these achievements, the process of generating compelling-looking content is still typically laborious. Photorealism is still hard to achieve without a significant amount of effort on the user's or programmer's part.

Computer vision is increasing being used for graphics applications. There is great synergy between the two fields: computer graphics assumes all is known about the world and focuses on rendering, while computer vision is about analyzing the world through images. Typical computer vision tasks include segmentation, recovery of surface properties, tracking, stereo, and recognition. The idea of using information from images of real-world scenes is very compelling; how else can you produce photorealistic imagery if not from real-world examples?

This paper illustrates some of ways that computer vision can help computer graphics: object-based segmentation to faciliate animation of Chinese paintings [49], optic flow to produce high dynamic range video [19], image matting using flash/no-flash [42], animating scenes using high-resolution stills through motion analysis [22], stereo and matting to generate "steerable" video ("3D video" or free-viewpoint video) [51], structure from motion and segmentation to model plants and trees [29, 43], and video analysis to extract visual properties of rain for real-time rendering [45].

We start by describing an active area that is a perfect example of synergy between vision and graphics: image-based modeling and rendering. This is followed by brief descriptions on specific instances of video and image processing for graphics applications with the aid of computer vision.

2. Image-based modeling and rendering

Image-based modeling and rendering techniques [38] 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 graphics/imaging spectrum of image-based representations [20, 17], which was later expanded in [16].

2.1. Graphics/imaging spectrum

At one end of the graphics/imaging spectrum (Figure 1), 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 [24], and layered-depth images (LDI) [35], and LDI tree [4]), the model consists of a set of images of a scene and their associated depth maps. The surface light field [48] 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,

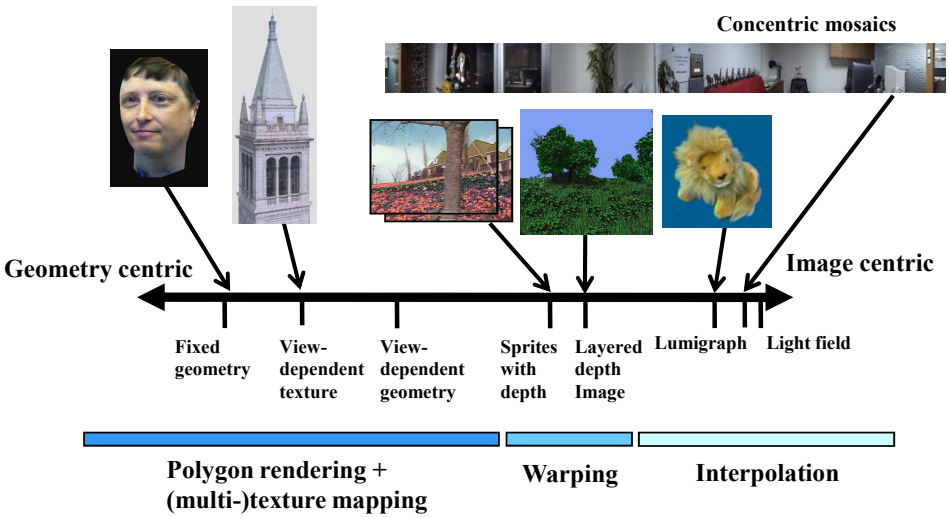

Figure 1. Graphics/imaging spectrum.

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 [5] generate novel views by interpolating optical flow between corresponding points. On the other hand, view morphing [34] 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 [21] produces a new image of a scene by appropriately filtering and interpolating a pre-acquired set of samples. The Lumigraph [12] is similar to light field rendering but it uses approximated 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 [39] 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. Unfortunately, oversampling means more intensive data acquisition, more storage, and higher redundancy.

2.2. Importance of geometry

Image-based rendering is about manipulating rays. If no geometry is available, visualization is limited to the camera path (where the input images are taken), or requires densely sampled images. If only rough geometry is available (e.g., piecewise planar approximation or using primitive shapes), high-quality virtual views can be reconstructed only if they deviate only slightly from the path of capture. Here, the quality of reconstruction is limited by the tolerance of reprojection error. Finally, if good geometry can be obtained, the range of virtual viewpoints with high-quality view reconstruction is limited by what have been observed. (View "hallucination" is usually not a good idea.)

The choice of scene representation is critical to the goal of photorealism. Since surfaces of a real scene tend to be non-Lambertian, using a single extracted 3D geometry to represent the scene is usually not recommended. An exception may be if the scene is highly structured as in the Facade work on modeling and rendering buildings [10]. In such a case, view-dependent texturing on a single geometry may be adequate. However, in general, we think the best choice would be to use *view-dependent geometries*. This has been demonstrated in a number of approaches, such as [28, 9, 15], with considerable success.

As Zitnick *et al.* [51] (Section 2.4) have demonstrated, using view-dependent geometries as well as extracting the matting (alpha) information at the boundaries and using it for rendering have proven to be highly effective. Boundaries need to be handled correctly in order to avoid artifacts (blurring and/or cut-out effects), and using matting information has been shown to be effective. The difficulty is in avoiding the manual process of indicating the matting areas for subsequent matte extraction. The manual component is required in prior matte extraction techniques. Zitnick *et*

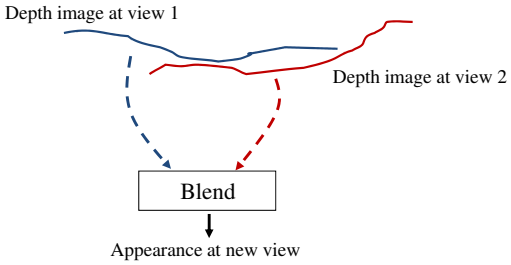

Figure 2. Concept of view-dependent depth images. The closest depth images (two in this case) are rendered at the desired virtual view and combined. Weighting of the rendered views can be based on some measure of proximity of the sampled views to the virtual view.

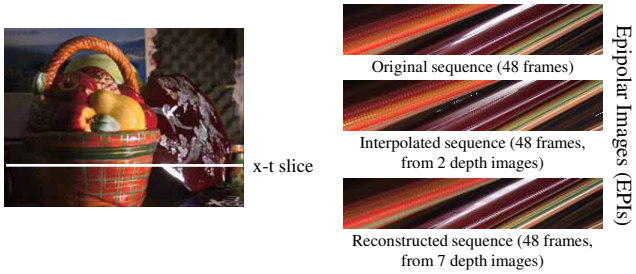

Figure 3. Results using view-dependent depth images. An epipolar image [2] is a planar slice (in this case, x-t slice) in an image stack (x-y-t). Notice how reasonably well the appearance of the complex scene is recovered with just two depth images.

al. use depth discontinuities to automatically indicate areas where foreground and background pixel colors exist, and apply an existing technique for matte extraction [6, 47]. A more systematic technique for simultaneously extracting matte information and refining depths at discontinuities uses 3D deformable contours as unifying structures [13].

The spatial-temporal view interpolation technique of Vedula *et al.* [44] is an appropriate approach to ensure temporal continuity and thus avoid flickering during rendering. Interestingly, Zitnick *et al.* showed that it is possible to produce flicker-free rendering without considering the time domain if the stereo data extracted is accurate enough (from the photoconsistency point of view). However, this feat will be difficult to replicate for general scenes with significant non-rigid effects such as specularities and translucencies.

2.3. View-dependent depth images

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.

As mentioned earlier, Debevec *et al*., in their Façade [10] 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*. [9] to further reduce the computational cost and to have smoother blending. This method employs visibility preprocessing, polygon-view maps, and projective texture mapping. For the unstructured Lumigraph work, Buehler *et al*. [3] apply a more principled way of blending textures based on relative angular position, resolution, and field-of-view. Kang and Szeliski [18] use not just view-dependent textures, but *view-dependent geometries* as well. This concept is illustrated in Figure 2. This is to account for stereo being 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 3.

There are other approaches designed to handle non-rigid effects for IBR (mostly for synthetic scenes). For example, Heidrich *et al.* [14] 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 [23] idea for handling non-diffuse scenes is based on layered depth images (LDIs) [1, 36]. 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.

Another representation that accounts for non-rigid effects is the surface light field [48], which handles complex reflections in real-world data. However, they also require detailed geometry (obtained with a laser scanner) and a very large number of input images to capture all the effects.

2.4. 3D video

The ability to interactively control viewpoint while watching a video is an exciting application of image-based rendering. The goal of [51] is to render dynamic scenes with interactive viewpoint control using a relatively small number of video cameras. There we show how high-quality video-based rendering of dynamic scenes can be accomplished using multiple synchronized video streams com-

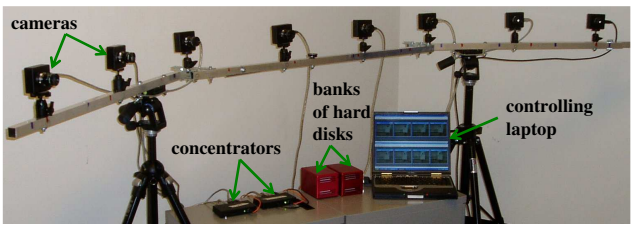

Figure 4. A configuration of our system with 8 cameras. (From [51].)

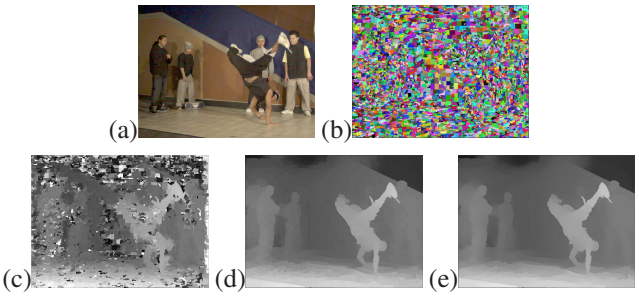

Figure 5. Sample results from stereo reconstruction stage: (a) input color image, (b) color-based segmentation, (c) initial disparity estimates, (d) refined disparity estimates, and (e) smoothed disparity estimates. (From [51].)

bined with novel image-based modeling and rendering algorithms. Once these video streams have been processed, we can synthesize any intermediate view between cameras at any time, with the potential for space-time manipulation.

In our approach, we first use a novel color segmentation-based stereo algorithm to generate high-quality photoconsistent correspondences across all camera views. Mattes for areas near depth discontinuities are then automatically extracted to reduce artifacts during view synthesis. Finally, a novel temporal two-layer compressed representation that handles matting is developed for rendering at interactive rates.

Figure 4 shows a configuration of our video capturing system with 8 cameras arranged along a horizontal arc. We use high resolution (1024×768) PtGrey color cameras to capture video at 15 fps, with 8mm lenses, yielding a horizontal field of view of about 30°. Each concentrator synchronizes four cameras and pipes the four uncompressed video streams into a bank of hard disks through a fiber optic cable. The two concentrators are synchronized via a FireWire cable.

The cameras are calibrated before every capture session using a 36" $\times$36" calibration pattern mounted on a flat plate, which is moved around in front of all the cameras. The calibration technique of Zhang [50] is used to recover all the camera parameters necessary for Euclidean stereo recovery.

We compute depth at each camera frame using a segmentation-based stereo algorithm. A sample result can be seen in Figure 5. We adopted a two-layer representation inspired by Layered Depth Images and sprites with depth [35]. We first locate the depth discontinuities in a depth map and create a boundary strip (layer) around these pixels. We then use a variant of Bayesian matting [6] to estimate the foreground and background colors, depths, and opacities (alpha values) within these strips. The original depth map (with texture) is called the main layer while the boundary strip (with texture) is called the boundary layer. Examples of these layers are shown in Figure 6.

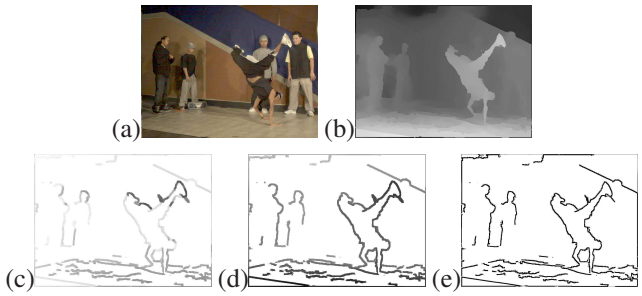

Figure 6. Sample results from matting stage: (a) main color estimates; (b) main depth estimates; (c) boundary color estimates; (d) boundary depth estimates; (e) boundary alpha (opacity) estimates. For ease of printing, the boundary images are negated, so that transparent/empty pixels show up as white. (From [51].)

To synthesize a desired virtual view, we find the two nearest sample views, warp all layers from the sample views, and combine them. A video view interpolation example can be seen in Figure 7(a-c). Figure 7(d) shows an additional capability of matting in a person captured in an earlier frame.

2.5. Plant and tree modeling

Plants and trees remain some of most difficult kinds of object to model due to their complex geometry and wide variation in appearance. While techniques have been proposed to synthetically generate realistic-looking plants, they either require expertise to use (e.g., [27]) or they are highly manual intensive. Current image-based techniques that use images of real plants have either produced models that are not easily manipulated (e.g., [30]) or models that are just rough approximations (e.g., [37]).

Note that in this paper, we differentiate between plants and trees—we consider "plants" as terrestrial flora with large discernible leaves (relative to the plant size), and "trees" as large terrestrial flora with small leaves (relative to the tree size). The spectrum of plants and trees with varying leaf sizes is shown in Figure 8. To model plants and trees, we use a hand-held camera to capture images at different views. We then apply a standard structure from motion technique to recover the camera parameters and a 3D point cloud. This allows us to recover camera poses and a preliminary 3D point cloud associated with the plant or tree.

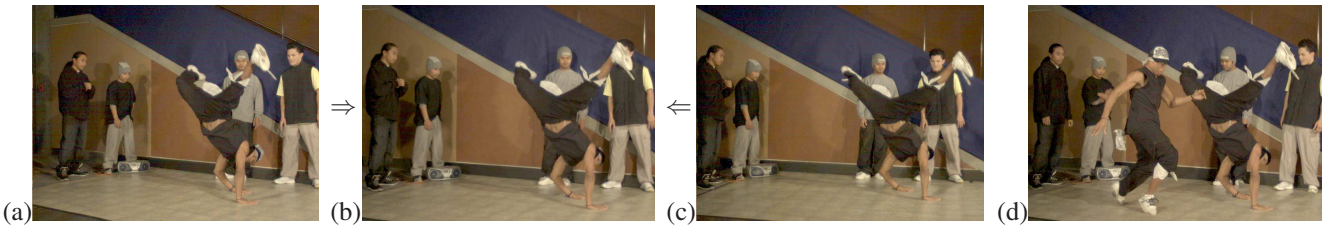

Figure 7. A video view interpolation example: (a,c) synchronized frames from two different input cameras and (b) a virtual interpolated view. (d) A depth-matted object from earlier in the sequence is inserted into the video. (From [51].)

Figure 8. Plant/tree spectrum based on relative leaf size. (From [43].)

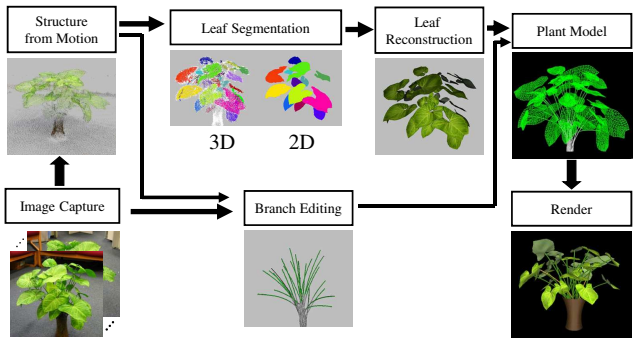

Figure 9. Overview of plant modeling system. (From [29].)

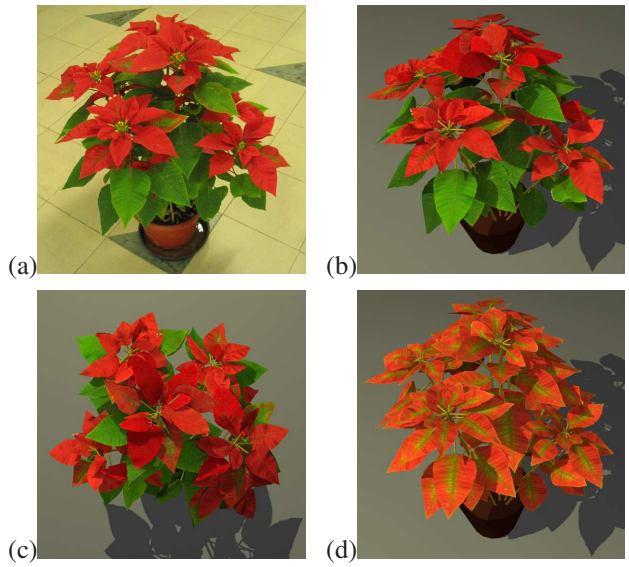

Figure 10. Image-based modeling of poinsettia plant. (a) An input image out of 35 images, (b) recovered model rendered at the same viewpoint as (a), (c) recovered model rendered at a different viewpoint, (d) recovered model with modified leaf textures. (From [29].)

Plants have relatively large leaves, which we try to model closely. Our system for modeling plants from images are shown in Figure 9. Leaf segmentation is based on both 3D distance and color information. More specifically, two 3D points belong to the same leaf if they are physically close and the projected 2D line segment between them does not contain large image gradients (which signal leaf boundary edges). Each leaf can then be modeled once they have been segmented. Because there is significant occlusion for branches, we opted for an interactive technique to model them. The user merely draws and edits curves on images, and optionally edits in 3D as well. The result of modeling a plant is shown in Figure 10.

Trees are treated differently because of their leaves have small image footprints. We use a different approach as summarized in Figure 11. (We assume that the tree has been matted out in each input image.)

Because the shapes of tree branches are approximately self-similar, we first model the visible branches and use them to generate those that are occluded (by randomizing the rotation angle between branch generations and scaling the child branches appropriately). Instead of attempting to

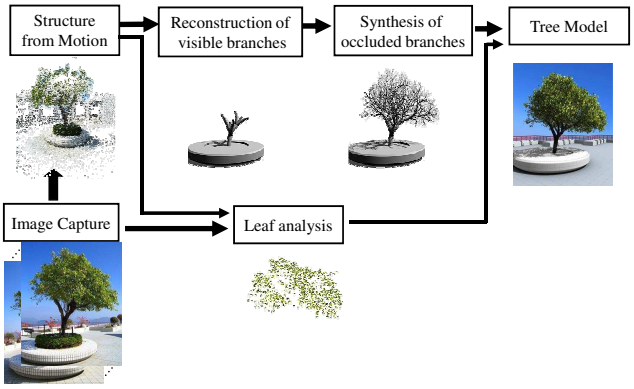

Figure 11. Overview of tree modeling system. (From [43].)

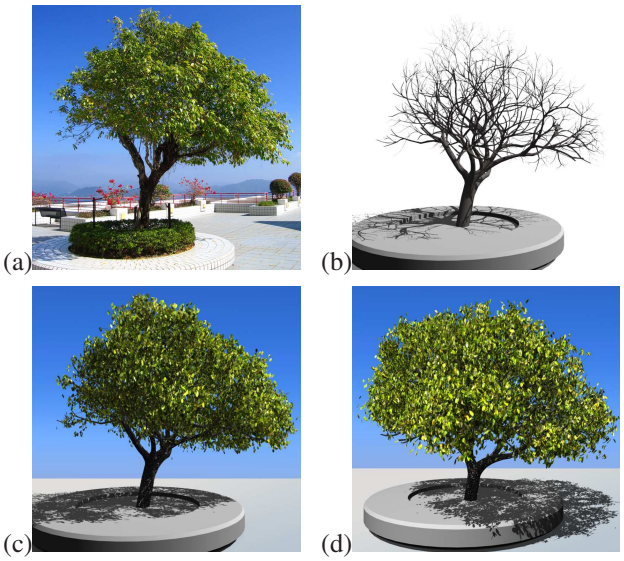

Figure 12. Result of tree modeling. (a) A source image (out of 18 images), (b) reconstructed branch structure rendered at the same viewpoint, (c) tree model rendered at the same viewpoint, and (d) tree model rendered at a different viewpoint. (From [43].)

compute point correspondences across views to recover the shape and location of leaves, we follow these steps: (1) let the user predefine a range of permissible leaf sizes, (2) cluster and segment the leaf regions for all the images, (3) for each leaf in each image, back-project onto 3D space and attach to the nearest 3D branch. Leaves that are too close to each other are culled to avoid redundancy. A result of tree modeling is shown in Figure 12.

3. Image and video processing

In this section, we describe examples of how images and video can be processed to provide interesting outputs for graphics applications.

3.1. Animating Chinese paintings

What if paintings could move? In [49], we propose a way of animating Chinese paintings by automatically decomposing an image of a painting into its hypothetical brush stroke constituents. Most Chinese paintings are typically sparse, with each brush stroke drawn very purposefully. Our method is specifically geared for handling types of paintings with such economic use of brush strokes; besides most Chinese paintings, other types include Sumi-e paintings and certain watercolor and oil paintings (e.g., Van Gogh paintings).

Our approach uses segmentation techniques and a library of brush strokes for fitting. The recovered brush strokes are basically vectorized elements, which are easy to animate. In addition to animation, the set of recovered brush strokes

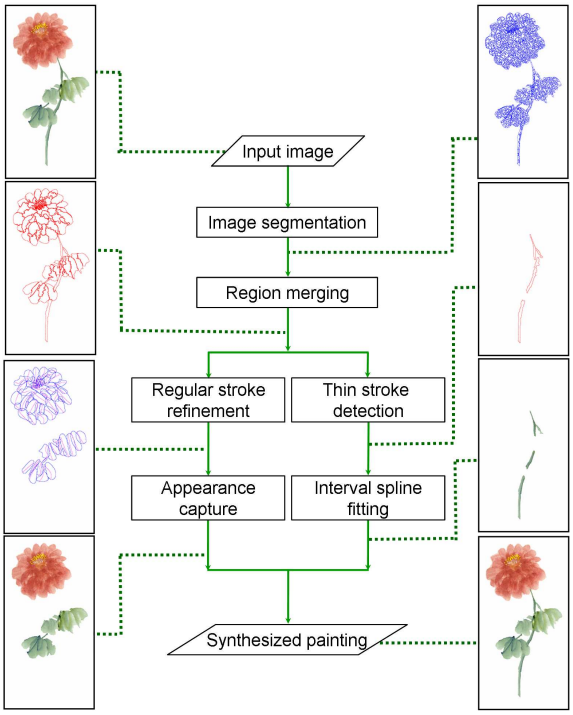

Figure 13. Overview of system to analyze and decompose an image of a Chinese painting. (From [49].)

can be used for synthesis of paintings or for manipulating images of paintings.

Before we animate a painting, we first decompose its image into a plausible set of brush strokes. A graphical overview of our decomposition approach is depicted in Figure 13. It also shows an example image, the intermediate results, and the final output. The basic idea is simple: we segment the image, use a brush library to find the best fit for each region, and refine the brush strokes found directly from the input image. The brush library used was created with the help of a painter who specializes in Chinese paintings. A few such brush strokes from the library are shown in Figure 14.

More specifically, the image is initially oversegmented using the mean-shift algorithm [8]. We merge the segments based on the observation that the gradient along the brush stroke tends to be smaller than the gradient across the stroke. We distinguish between regular and long strokes by the aspect ratio (long strokes have aspect ratios greater than 10 to 1). Each long stroke is represented by an interval spline, with nodes containing thickness and color information for appearance reconstruction. Regular strokes are then compared against the library shapes to further refine their shapes through further merging. The final step involves dilation of each brush stroke to completely cover its texture.

One difficult issue that we had to deal with is recovery of color at areas of intersection between brush strokes.

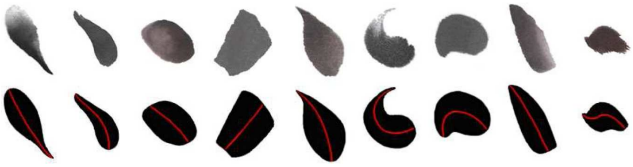

Figure 14. Sample brush strokes in the library (9 shown out of 62). (From [49].)

Figure 15. Two frames of the animated flower video clip. (From [49].)

We adapted the mixture model proposed by Porter and Duff [26] to model overlapping strokes as matted objects because the combination color in the overlapping brush region is generally the result of mixing and optical superimposition of different pigment layers. The solution is simplified by assuming smooth color distributions across brush strokes.

Once the brush strokes have been recovered, they can then be animated using a user interface. This interface allows a variety of operations that include addition and removal of strokes, and manipulating single or groups of brush strokes. Brush strokes can be edited by moving boundary points or points on skeletons, or by applying global transforms. Results of animating the flower example can be seen in Figure 15.

3.2. Image matting using flash/no-flash

Image matting from a single image is fundamentally under-determined. The most common approach for pulling a matte is blue screen matting [40], in which a foreground object is captured in front of a known solid-colored background, usually blue or green. Blue screen matting is the standard technique employed in the movie and TV industries because a known background B greatly simplifies the matting problem. However, blue screen matting requires an expensive well-controlled studio environment to reduce artifacts such as blue spill, backing shadows, and backing impurities [40]. In addition, blue screen matting is less suitable for outdoor scenes.

Natural image matting approaches (e.g., [32, 6]) were proposed to directly compute the matte from a single natural image. First, the input image is manually partitioned into definitely foreground, definitely background, and unknown regions. These three regions are collectively referred to as the *trimap* [6]. Then, α, F, and B are estimated for all pixels in the unknown region. Natural image matting uses additional information from the trimap's spatially coherent distribution of pixels in each region. Unfortunately, they fail to work when the foreground and the background are alike or highly textured. In addition, specifying a good trimap can be labor intensive, especially for complex objects.

In [42], we propose *flash matting* for natural image matting using a flash/no-flash image pair. Using additional information from a flash image, our flash matting can extract a high-quality matte even when the foreground and the background have similar colors or when the background is complex. In addition, no special studio environment is required in our approach.

In flash photography, the flash intensity falls off as inverse distance squared. This is a drawback in flash photography because a distant background may be poorly lit by the flash compared to the foreground (producing the "tunnel" effect). We use the "tunnel" effect for effective image matting. The difference between the flash and no-flash images (which we call *flash-only image*) allows reliable foreground-background separation.

For a static foreground and a fixed camera, we assume that the alpha channel of the foreground is unchanged in the no-flash image I (also called ambient image) and the flash image I^f. We have the matting equations for I and I^f:

$$I = \alpha F + (1-\alpha)B, \quad (1)$$

$$I^f = \alpha F^f + (1-\alpha)B^f, \quad (2)$$

where $\{F, B\}$ are the ambient foreground and background colors, and $\{F^f, B^f\}$ are the flash foreground and background colors, respectively.

When the camera and flash unit are together and the background scene is distant from the camera, the intensity change of the background in flash and no-flash images will be small, $B^f \approx B$. Thus, we have the following *flash matting equation*:

$$I^f = \alpha F^f + (1-\alpha)B. \quad (3)$$

Subtracting (1) from (3), we have the *foreground flash matting equation*:

$$I' = I^f - I = \alpha(F^f - F) = \alpha F', \quad (4)$$

where $F' = (F^f - F)$ is the additional flash foreground color. We refer to this difference image I' as the flash-only image.

Equation (4) is the foundation of our foreground flash matting because the flash-only image I' is independent of

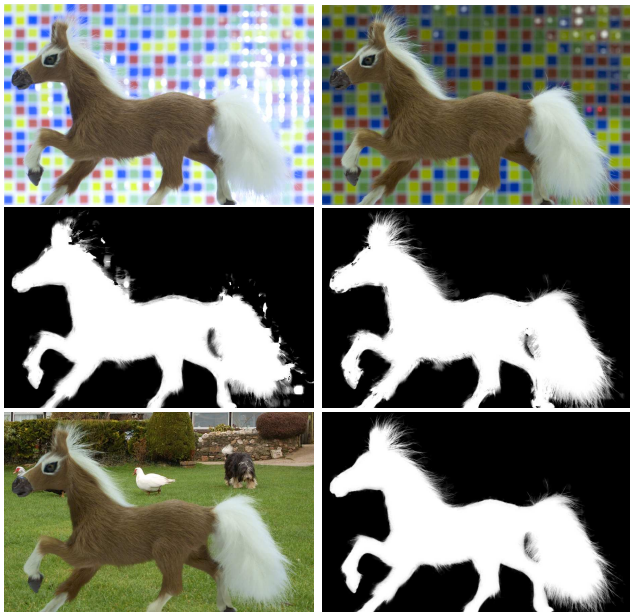

Figure 17. Background flash matting. Top row: Flash/no-flash image pair. Middle row: Bayesian matting results on the no-flash image and flash image. Bottom row: Recovered alpha matte and composition result. The flash was placed facing the background and away from the foreground object. It was triggered by a wireless flash transmitter. (From [42].)

how similar the foreground and background are or the complexity of the background. However, the foreground flash matting problem is still under-constrained; to solve it, a trimap remains necessary. We extract the trimap from the flash-only image.

Rather than computing the matte from either the no-flash image I or flash image I^f, we compute the matte *jointly* from these images using a Bayesian formulation. Details of our joint Bayesian flash matting can be found in [42]. An example using this technique can be seen in Figure 16. Other variations are possible; for example, if the background is close to the foreground, we can direct the flash at the background away from the foreground. We call this *background flash matting*, and one such example can be seen in Figure 17.

3.3. High dynamic range video

The real world has a great deal more brightness variation than can be captured by the sensors available in most cameras today. The radiance of a single scene might contain four orders of magnitude from shadows to fully lit regions; however a typical CCD or CMOS sensor only captures about 256 levels (the non-linear allocation of levels through a gamma curve can improve this slightly).

Our work described in [19] addresses the problem of capturing and rendering high dynamic range (HDR) dynamic scenes. An overview of our approach is shown in Figure 18. Our approach consists of automatically determining the optimal exposure bracketing in time during capture, mapping exposure information across neighboring images, and optionally tonemapping to produce an image viewable with conventional displays. Our capture solution differs from previous efforts in that it employs a simple reprogramming of the auto gain mechanism in a standard video camera. This allows us to use the inexpensive and high resolution sensors available today, unlike the novel sensor designs which currently are not widely available and suffer from a lack of resolution. We use an auto gain algorithm which intelligently varies the exposure from frame to frame in order to capture different parts of a scene's radiance map.

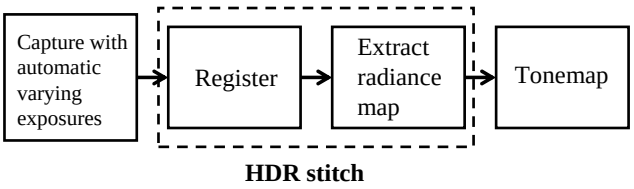

Figure 18. Overview of HDR video system. (From [19].)

This is followed by an offline process to motion-compensate the captured video and create the full radiance map at each frame time. This operation, which we call HDR stitching, establishes correspondence between images (i.e., motion compensate) in order to combine pixels of different exposures of the same part of the scene to produce their HDR version. Note that only pixels that are neither saturated nor of low contrast are used in estimating the scene radiance. HDR stitching can be viewed as sub-sampling along a different dimension when compared to the spatially-varying pixel exposures work [25]. In that work, the trade-off is spatial resolution for greater dynamic range, here the trade-off is temporal resolution for greater dynamic range.

In order to view the HDR video, it must be tonemapped. We showed that applying one of the existing algorithms [31] on a frame by frame basis is not sufficient as this can lead to visible temporal inconsistencies in the mapping. In order to compensate for this we have extended one of these techniques to operate on HDR video; it is customized to use statistics from neighboring frames in order to produce smoothly varying tonemapped images in time.

Results for a video of a driving scene are shown in Figure 19. Notice that the tonemapped video has good contrast in both interior and exterior of the car.

3.4. Animating scenes using high-resolution stills

Current techniques for generating animated scenes involve either videos (whose resolution is limited, as in video textures [33]) or single images (which require a significant

Figure 16. Flash matting results for flower scene with complex foreground and background color distributions. From left to right: flash image, no-flash image, automatically extracted matte, composite image with a different background. (From [42].)

Figure 19. High dynamic range video of a driving scene. Top row: Input video with alternating short and long exposures. Bottom row: High dynamic range video (tonemapped). (From [19].)

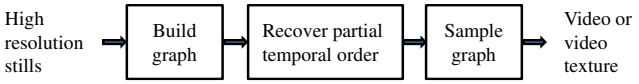

Figure 20. Overview of system for animating scenes from high-resolution stills. (From [22].)

amount of user interaction, as in [7]). In [22], we describe a system that allows the user to quickly and easily produce a compelling-looking animation from a small collection of *high resolution* stills. The approach is summarized in Figure 20.

There are two unique features in our system. First, it applies an automatic partial temporal order recovery algorithm to the stills in order to approximate the original scene dynamics. The output sequence is subsequently extracted using a second-order Markov Chain model. Second, a region with large motion variation can be automatically decomposed into semi-autonomous regions such that their temporal orderings are softly constrained. This is to ensure motion smoothness throughout the original region. The final animation is obtained by frame interpolation and feathering. Our system also provides a simple-to-use interface to help the user to finetune the motion of the animated scene. Using our system, an animated scene can be generated in minutes.

Our scene animation system is capable of generating a video or video texture from a small collection of stills. Our system first builds a graph that links similar images. It then recovers partial temporal orders among the input images and uses a second-order Markov Chain model to generate an image sequence of the video or video texture (Figure 20). Our system is designed to allow the user to easily finetune the animation. For example, the user has the option to manually specify regions where animation occurs independently (which we term *independent animated regions* (IAR)) so that different time instances of each IAR can be used independently. An IAR with large motion variation can further be automatically decomposed into *semi-independent animated regions* (SIARs) in order to further improve the naturalness of motion. The user also has the option to modify the dynamics (e.g., speed up or slow down the motion, or choose different motion parameters) through a simple interface. Finally, all regions are frame interpolated and feathered at their boundaries to produce the final animation. The user needs only a few minutes of interaction to finish the whole process. In our work, we limit our scope to quasi-periodic motion, i.e., dynamic textures. Results for automatic decomposition of IARs into SIARs for a lake scene are shown in Figure 21.

Figure 21. Image decomposition and the creation of SIARs from an IAR for a lake scene. Left: IAR (blue), the stationary region (pink), and the slow moving region (yellow). Middle: common low-frequency area A_{LF} (black) of the IAR. Right: final boundaries selected to produce SIARs. (From [22].)

4. Capturing and rendering natural phenomena: Rain

Weather simulation is an important effect that adds to the realism of a rendered scene. The ability to simulate weather effectively would be exceedingly useful for applications such as film post-production, games, simulators, and virtual reality. A lot of work has been done on simulating rain, fog, cloud, and snow.

In [45], we propose a new technique to render realistic

rain in real-time using a commodity graphics accelerator. Our technique consists of two parts: off-line image analysis of rain videos, and real-time particle-based synthesis of rain. Videos of real rain are analyzed to extract the rain mattes; random samples of these rain stroke mattes are then used for online synthesis. We incorporate pre-computed radiance transfer (PRT) in our particle system to take into account the scene radiance. We show that there is a closed-form solution to the transfer function of a raindrop, making its evaluation highly efficient. Our approach achieves high realism with low computation cost and a small memory footprint, making it very practical.

Although rain strokes can be manually designed, the process is tedious, and it is not guaranteed that the resulting simulated strokes will appear natural or sufficiently varied. As a result, we chose to extract rain strokes from actual footage involving real static scenes. As pointed out in [41] and [11], if the video was not captured using a high-speed camcorder, the rain in successive frames are practically independent. Using this reasoning, Starik and Werman [41] and Garg and Nayar [11] simply used a median filter to extract or detect rain.

Unfortunately, the median filter is applied pixel by pixel independently. It does not take into account two relevant factors: rain is globally directional and a typical scene has a mostly continuous color distribution. As a result, it is less resilient to image noise. We incorporate these two factors in our rain extraction formulation as a nonlinear least-squares problem:

$$(\mathbf{S},\alpha)=\arg\min_{\mathbf{S},\alpha}\sum_{i,j,k}\left(I^k_{i,j}-(1-\alpha^k_i)S_{i,j}-\alpha^k_i C_j\right)^2 +\lambda\sum_{i,j}||\nabla S_{i,j}||^2+\mu\sum_{i,k}\left(\nabla\alpha^k_i\cdot\mathbf{D}_0\right)^2, \quad (5)$$

where indices i, j, and k are used for referencing pixels, color channels, and frames, respectively. I is the observed color, α is the fractional blending weight due to rain, S is the color of the static background, C is the "color" of the rain (due to the environment), $\mathbf{D}_0$ is the principal rain direction, and λ and μ are weights. The first term in the right of (5) assumes the observed color is a linear blend of the colors of the scene and rain. The second term encourages smoothness in the scene color distribution, while the third favors smoothness of the alpha distribution along the rain direction $\mathbf{D}_0$. An Expectation-Maximization (EM) algorithm is applied to solve α and $\mathbf{S}$ alternately; the median filtered version of $\mathbf{S}$ is used to initialize $\mathbf{S}$. $\mathbf{D}_0$ is initialized as the vertical direction. It is refined immediately after estimating α by detecting the most salient line in the frequency spectrum of α that passes through the origin.

In order to be able to exercise more control over the rain and enable the rain to appear natural with the scene, we chose to use a particle system for rendering, rather than using a single layer of rain [41]. Figure 22 shows the outline of the rendering process.

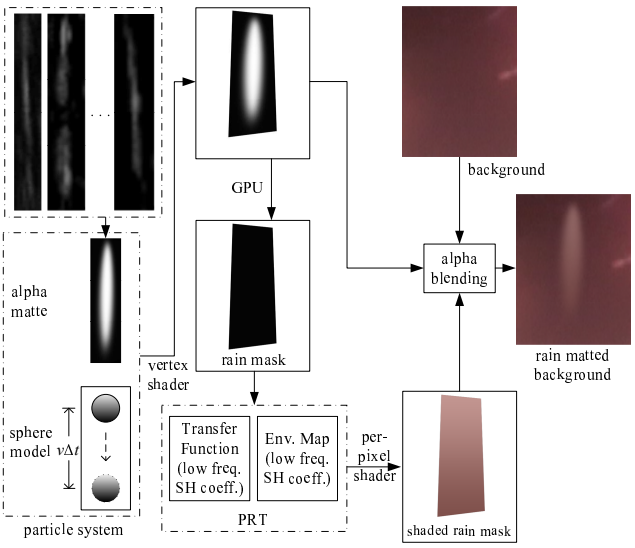

Figure 22. Overview of system for rendering rain. Each particle consists of a rain stroke matte that is randomly chosen from the rain stroke samples, a sphere model, and attributes such as velocity. The vertex shaders determine the 3D position and orientation of the alpha matte and which pixel on the screen to shade. Our precomputed radiance transfer (PRT) based per-pixel shader then computes the color of each pixel on the rain stroke. Finally, the rain stroke and the background are blended using the rain alpha matte. "SH" refers to "spherical harmonics." (From [45].)

Each particle has a sphere model with the refractive index of water, a random rain stroke matte, and other physical attributes such as position and velocity. The rain color is computed on the fly.

The particles are assumed to be uniformly distributed in space; we model the position as a uniform random variable. The length of the matte shape is estimated based on the particle's current velocity and the exposure time of the virtual camera. The matte shape is achieved by shrinking or stretching its original shape accordingly through interpolation. The diameter of the sphere that models the raindrop corresponds to the width of the rain matte after being mapped to the scene coordinate frame. We show that there is a closed-form solution to the transfer function required by PRT, which makes its evaluation highly efficient.

Our rendering technique is purely GPU based. (It requires pixel shader 2.0 and vertex shader 1.1, which are supported by most commercially available graphics cards.) After the 3D position and orientation of each particle have been computed by a vertex shader, the next step is to color the particle. The GPU first determines which part of the screen is to be shaded. A per-pixel PRT shader then computes the intensity distribution of the particle. Details of how the PRT shader works are given in [46]. The environ-

Figure 23. Adding synthetic rain to video of a street scene. Top: an original frame of resolution 720×480. Bottom: result of adding rain and close-up of the boxed area. Notice how the synthetic rain was affected by the background. Here, 80,000 rain strokes (all visible to the virtual camera) were rendered at 78 fps on our Intel Pentium® 4 3.20GHz PC (with an nVIDIA® GeForce 7800 GT graphics card). The memory footprint dedicated to computing the lighting effect on the rain is only about 6 MB for the 33 second video. (From [45].)

ment map is used to compute the intensity and color of each pixel of a particle.

Finally, the GPU blends the appearance of the particle with the background using the computed intensity and alpha distributions. Note that if rough geometry of the scene is known, the particle will be automatically clipped by the rendering engine at the scene surface (defined using a mesh of a synthetic scene or the depth map of a real video). This gives a volumetric appearance of rain. Results for a street scene is given in Figure 23.

5. Concluding remarks

Computer graphics and computer vision complement each other well: graphics is about making compelling-looking imagery and videos while vision is about analyzing images and videos. Using both allows more realistic rendering with less effort.

6. Acknowledgments

This paper is based on work done with the following collaborators: Tian Fang, Yin Li, Zhouchen Lin, Long Quan, David Salesin, Harry Shum, Jian Sun, Rick Szeliski, Ping Tan, Matthew Uyttendaele, Jingdong Wang, Lifeng Wang, Yunbo Wang, Simon Winder, Songhua Xu, Yingqing Xu, Xu Yang, Xuan Yu, Lu Yuan, Gang Zeng, and C. Lawrence Zitnick.

References

[1] S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 434–441, June 1998.

[2] 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*, 1:7–55, 1987.

[3] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured Lumigraph rendering. In *Computer Graphics (SIGGRAPH)*, pages 425–432, Los Angeles, CA, August 2001.

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

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

[6] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A Bayesian approach to digital matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2001.

[7] Y.-Y. Chuang, D. B. Goldman, K. C. Zheng, B. Curless, D. Salesin, and R. Szeliski. Animating pictures with stochastic motion textures. *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, 24(3):853–860, July 2005.

[8] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

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

[10] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *ACM SIGGRAPH*, pages 11–20, August 1996.

[11] K. Garg and S. Nayar. Detection and removal of rain from videos. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 528–535, 2004.

[12] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. *Computer Graphics (SIGGRAPH)*, pages 43–54, August 1996.

[13] S. W. Hasinoff, S. B. Kang, and R. Szeliski. Boundary matting for view synthesis. In *IEEE Workshop on Image and Video Registration*, July 2004.

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

[15] 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*, pages 94–101, 1999.

[16] S. Kang, Y. Li, X. Tong, and H.-Y. Shum. Image-based rendering. *Foundations and Trends in Computer Graphics and Vision*, 2(3), April 2007.

[17] S. B. Kang. A survey of image-based rendering techniques. In *Videometrics VI (SPIE International Symposium on Electronic Imaging: Science and Technology)*, volume 3641, pages 2–16, January 1999.

[18] S. B. Kang and R. Szeliski. Extracting view-dependent depth maps from a collection of images. *International Journal of Computer Vision*, 58(2):139–163, July 2004.

[19] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM SIGGRAPH and ACM Trans. on Graphics*, 22(3):319–325, July 2003.

[20] J. Lengyel. The convergence of graphics and vision. *IEEE Computer*, 31(7):46–53, 1998.

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

[22] Z. Lin, L. Wang, Y. Wang, S. B. Kang, and T. Fang. High resolution animated scenes from stills. *IEEE Trans. on Visualization and Computer Graphics*, 13(3):562–568, May/June 2007.

[23] D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop*, pages 301–314, June 1998.

[24] W. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In *Symposium on I3D Graphics*, pages 7–16, April 1997.

[25] T. Mitsunaga and S. K. Nayar. High dynamic range imaging: Spatially varying pixel exposures. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 472–479, June 2000.

[26] T. Porter and T. Duff. Compositing digital images. *ACM SIGGRAPH*, pages 253–259, July 1984.

[27] P. Prusinkiewicz, M. James, and R. Mech. Synthetic topiary. In *Computer Graphics (SIGGRAPH)*, pages 351–358, July 1994.

[28] 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*, June 1997.

[29] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang. Image-based plant modeling. *ACM SIGGRAPH and ACM Trans. on Graphics*, 25(3):772–778, August 2006.

[30] A. Reche-Martinez, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *Proceedings of SIGGRAPH (ACM Transactions on Graphics)*, 23(3):720–727, August 2004.

[31] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. *ACM SIGGRAPH*, 21(3):267–276, 2002.

[32] M. A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 18–25, 2000.

[33] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *ACM SIGGRAPH*, pages 489–498, July 2000.

[34] S. M. Seitz and C. M. Dyer. View morphing. In *Computer Graphics (SIGGRAPH)*, pages 21–30, New Orleans, LA, August 1996.

[35] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. *Computer Graphics (SIGGRAPH)*, pages 231–242, 1998.

[36] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. In *Computer Graphics (SIGGRAPH)*, pages 231–242, July 1998.

[37] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61, May/June 2001.

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

[39] H.-Y. Shum and L.-W. He. Rendering with concentric mosaics. *Computer Graphics (SIGGRAPH)*, 33:299–306, 1999.

[40] A. R. Smith and J. F. Blinn. Blue screen matting. In *Computer Graphics (SIGGRAPH)*, pages 259–268, 1996.

[41] S. Starik and M. Werman. Simulation of rain in video. In *Proceedings of the 3rd International Workshop on Texture Analysis and Synthesis*, pages 95–100, 2002.

[42] J. Sun, Y. Li, S. B. Kang, and H.-Y. Shum. Flash matting. *ACM SIGGRAPH and ACM Trans. on Graphics*, 25(3):599–604, August 2006.

[43] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. Image-based tree modeling. *ACM SIGGRAPH and ACM Trans. on Graphics*, 26(3), August 2007.

[44] S. Vedula, S. Baker, S. Seitz, and T. Kanade. Shape and motion carving in 6D. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 592–598, Hilton Head Island, NC, June 2000.

[45] L. Wang, Z. Lin, T. Fang, X. Yang, X. Yu, and S. B. Kang. Real-time rendering of realistic rain. In *ACM SIGGRAPH (Sketch)*, August 2006.

[46] L. Wang, Z. Lin, T. Fang, X. Yang, X. Yu, and S. B. Kang. Real-time rendering of realistic rain. Technical Report MSR-TR-2006-102, Microsoft Corporation, July 2006.

[47] Y. Wexler, A. W. Fitzgibbon, and A. Zisserman. Bayesian estimation of layers from multiple images. In *European Conference on Computer Vision*, volume 3, pages 487–501, 2002.

[48] 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)*, pages 287–296, July 2000.

[49] S. Xu, Y. Xu, S. B. Kang, D. H. Salesin, Y. Pan, and H.-Y. Shum. Animating Chinese paintings through stroke-based decomposition. *ACM Trans. on Graphics*, 25(2):239–267, April 2006.

[50] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[51] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3):600–608, August 2004.