From 2D Images to 2.5D Sprites: A Layered Approach to Modeling 3D Scenes

Richard Szeliski and P. Anandan
Microsoft Research
Microsoft Corporation
Redmond, WA 98052

Simon Baker
Carnegie Mellon Unviersity
Robotics Institute
Pittsburgh, PA 15213

Abstract

We propose a framework for modeling the appearance and geometry of 3D scenes as a collection of approximately 2D layers. Each layer corresponds to a view-based representation of a portion of the scene whose disparity across the given set of views can be approximately modeled using a planar surface. The representation of each layer consists of the parameters of the plane, a color image that specifies the appearance of that portion of the scene, a per-pixel opacity map, and a per-pixel depth offset relative to the nominal plane. The layers are recovered by analyzing the pixel disparities across the input images. Depth and color information can be integrated from multiple images even in regions that may be partially occluded in some of the views. New views of the scene can be generated efficintly by rendering each individual layer from that view and combining the layer images in a back to front order. Layers from different scenes can be combined into a new synthetic scene with realistic appearance and geometric effects for multimedia authoring applications.

1 Introduction

An easy and flexible way to collect visual information about a 3D scene is to simply take multiple images of the scene from different viewpoints. Alternatively a single video camera can be moved around to collect many views of a scene. Using such a collection of images to create 3D models of the scene and to generate new views of the scene has been an ongoing research topic both in computer vision and computer graphics.

The classical approach for 3D scene modeling from multiple images is to break the problem into two sub problems: creating a 3D geometric model of the scene, and creating a texture map that captures the visual appearance of the scene. A more recent emerging paradigm is Image-Based Modeling and Rendering (IBMR) [1], which seeks to use the input images themselves (perhaps together with some explicit 3D information recovered from the images) as the basis for synthesizing new views. In many multimedia applications, where visual presentation and authoring are more important than 3D interaction, IBMR methods have several advantages.

In this paper we describe an approach for modeling 3D scenes from multiple images that fits within the IBMR paradigm. Based on analyzing the input images, we decompose the scene into a set of 2.5D *sprites*. Each sprite models a region of the 3D environment which can be approximately represented as a plane (see below for a more precise definition). It consists of: (i) the equation that describes the nominal plane with respect to a reference coordinate system associated with the sprite, (ii) a color (e.g., RGB) image that describes the appearance of the sprite from a reference view, (iii) a per-pixel opacity (α) map [12, 6] (the opacity value is in the range $[0, 1]$ with 0 indicating that the layer is completely transparent at that pixel and 1 indicating that it is completely opaque), and (iv) a per-pixel depth offset relative to the nominal plane. The reference coordinate system is chosen arbitrarily but is the same for all the sprites in the scene.

The collection of sprites constitutes our representation of a scene. Given the camera position, orientation, and the imaging parameters of a new view (relative to the reference coordinate system of the sprites), a new view is synthesized by projecting each of the sprites to that view, then combining them from back to front using standard image compositing operations [6].

An illustration of this representation is provided in Figure 1. This figure shows two images (taken from slightly different view points) from the *flower garden* sequence, which is used for MPEG compression evaluation. Based on these two, and four other images that were taken from viewpoints in between these two viewpoints, this scene was decomposed into six sprites as shown in the same figure. These sprites correspond to planes at different depths and orientations in the scene. Note that although the house is never fully visible in any single image, the different partial views of the house are integrated into the single image corresponding to the house sprite. Later in this paper, we will describe our methods for recovering the layer sprites from input images, and also show the rendering of a scene from novel view points.

The sprites are roughly akin to decomposing the 3D scene

0-7695-0253-9/99 $10.00

(a) (b) (c)

Figure 1: Results on the *flower garden* sequence: (a) first and (b) last input images; (c) the six layer sprites

into a set of layers that are positioned at different depths and orientations. In fact, if the per-pixel depth offset is ignored, these sprites are similar to "cardboard cutouts" each of which has a portion of the scene painted on it. These cutouts are "layered" in 3D space, hence the name *layered representation*. Note that, the term *sprite* has traditionally been used to describe a *2D* color image with an opacity map [12, 6], in particular *the 3D positioning of the scene elements captured in the sprite are not represented.* In contrast, our layer sprites also correspond to a plane of arbitrary orientation in 3D space together with the a per-pixel depth offset. Thus, each of our sprites is similar to a *bas-relief.* Hence we call them *2.5D sprites.*

The layered representation proposed here is suitable for operations that require reasoning about the 3D scene geometry such as synthesizing a new view of the scene. It also integrates the visual appearance and the *local* scene geometry into a single combined representation. As will be noted later, this has advantages both during the modeling and the rendering processes. Also, the sprites integrate partial and discontinuous views of neighboring portions of the scene into a single spatial coherent whole. This makes it easy to manipulate and render coherent scene elements together.

The sprite-based decomposition of a 3D scene is progressive in modeling the complexity of the scene geometry. For instance an approximate but reasonable looking rendering of the scene can be achieved by ignoring the per-pixel depth offsets. Such a "cut-out only" representation can be maintained, transmitted, and rendered with greater efficiently than a full 3D scene.

In the remainder of this paper, we provide a more precise definition of the sprite-based representation and describe our current methods for recovering them from images. Rendering using layer sprites is discussed in [14]. We will also show illustrative examples, and discuss limits of the power of this representation and our plans for future work.

2 The Layer Sprite Representation

The basic concepts of our framework are illustrated in Figure 2. The input consists of K images $I_1, I_2, \ldots, I_K$ captured by K cameras with known projection matrices

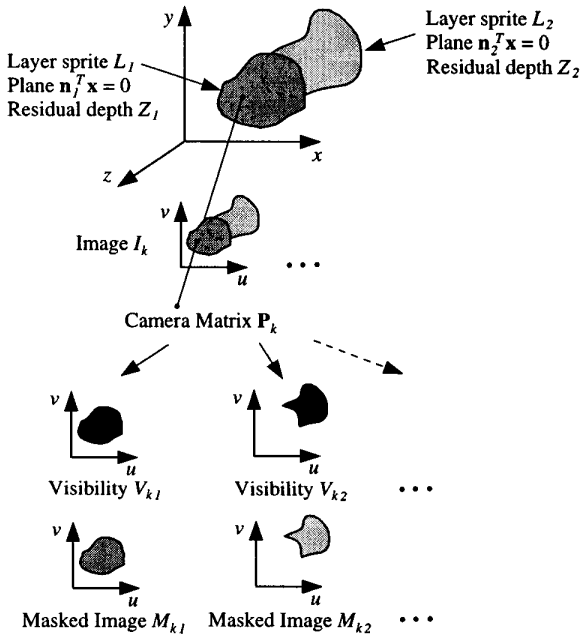

Figure 2: Suppose K images I_k are captured by K cameras $\mathbf{P}_k$. We assume that the scene can be represented by L sprite images L_l on planes $\mathbf{n}_l^T\mathbf{x} = 0$ with depth offsets Z_l. The boolean masks B_{kl} denote the pixels in image I_k from layer L_l and the masked images are given by $M_{kl} = B_{kl} \cdot I_k$. sprites

$\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_K$. Following standard conventions in computer graphics and vision, we use homogeneous coordinates for both 3D world coordinates $\mathbf{x} = (x, y, z, 1)^T$ and for 2D image coordinates $\mathbf{u} = (u, v, 1)^T$. Thus the projection matrices P_k are 3×4 matrices that map the world points to image points under perspective projection; these matrices specify the location, orientation, and the internal parameters of the imaging cameras.

The 3D scene is described by a collection of L approximately planar layers. Each layer sprite consists of:

1. a layer sprite image – following [6], this consists of *pre-multiplied image*, i.e.,

$$L_l(\mathbf{u}_l) = (\alpha_l \cdot r_l, \alpha_l \cdot g_l, \alpha_l \cdot b_l, \alpha_l) \quad (1)$$

where $r_l = r_l(\mathbf{u}_l)$ is the red band, $g_l = g_l(\mathbf{u}_l)$ is the green band, $b_l = b_l(\mathbf{u}_l)$ is the blue band, and $\alpha_l = \alpha_l(\mathbf{u}_l)$ is the opacity. and $\mathbf{u}_l$ are the coordinates of the layer sprite.

2. a homogeneous vector $\mathbf{n}_l$ (which defines the plane equation of the layer via $\mathbf{n}_l^T\mathbf{x} = 0$) and

3. a per-pixel residual depth offset $Z_l(\mathbf{u}_l)$.

The recovery of the sprites from input images is discussed in Section 3 below. But first, we describe how these layer sprites are used to synthesize images.

2.1 Image Synthesis From Sprites

We formulate the generative (forward) model of the image formation process using image compositing operations [6], i.e. by painting the sprites one over another in a back-to-front order. The basic operator used to overlay the sprites is the *over* operator:

$$F \odot B \equiv F + (1 - \alpha_F)B, \tag{2}$$

where F and B are the foreground and background sprites, and α_F is the opacity of the foreground [12, 6]. This definition of the over operator assumes pre-multiplied opacities, as in Equation (1). The generative model consists of the following two steps:

1. Using the camera matrices, plane equations, and residual depths, warp each layer onto the coordinate frame of image I_k. We denote the *warped* sprite l to image k by U_{kl}. Note that the opacities should be warped along with the color values [6].
2. Composite the warped sprites in back-to-front order (which can be computed from the plane equations):

$$S_k = \bigodot_{l=1}^{L} U_{kl} = U_{k1} \odot \cdots \odot U_{kl} \tag{3}$$

to obtain the *synthesized* image S_k. If we have solved the stereo reconstruction problem, and neglecting resampling issues, S_k should match the input I_k.

This last step can be re-written as three simpler steps:

2a. Compute the *visibility* of each warped sprite [16]:

$$V_{kl} = V_{k(l-1)}\,(1 - \alpha_{k(l-1)}) = \prod_{l'=1}^{l-1}(1 - \alpha_{kl'}) \tag{4}$$

where α_{kl} is the alpha channel of U_{kl}, and $V_{k1} = 1$.

2b. Compute the masked images, $M_{kl} = V_{kl}U_{kl}$.

2c. Sum up the masked images, $S_k = \sum_{l=1}^{L} M_{kl}$.

In these last three substeps, the visibility map makes the contribution of each sprite pixel to the image S_k explicit.

2.2 Discussion

The definition of the 2.5D sprites is sufficient for representing any 3D *Lambertian* scene. The current definitions, however, do not capture specular effects. (For the present we will focus only on Lambertian scenes.) Likewise, the generative process described above can be used to synthesize any image of a given scene.

A given 3D scene can be decomposed into sprites in many ways in a manner consistent with our definitions. Although we present a particular approach for sprite decomposition in this paper, the representational framework is neutral on this issue. In practice, we would expect the number of sprites to be small both in order to keep the representation compact and to facilitate ease of manipulation. In other words, the decomposition of a scene into sprites should be parsimonious. Likewise, the decomposition should be "meaningful" to a human user, i.e., each layer sprite should correspond to a physically coherent portion of the scene, such as a surface or an object.

Note that the sprite representation described here is ultimately based on an analysis of image disparities over the collection of input images. However, disparities are inversely proportional to *depth*, namely the distance of the scene elements from the cameras. It is likely in practice that an algorithm will group scene elements into sprites based on their *disparities* (as opposed to *depths*), since the image re-synthesis error will be directly proportional to disparity error. This has the practical impact that the sprites associated with regions of the scene far from the cameras will cover larger 3D segments than sprites associated with nearer portions of the scene.

3 Layer Sprite Extraction From Images

In this section, we briefly describe our approach for recovering layer sprites from a set of input images. A more detailed description can be found in [2]. For the purposes of this paper, we assume that the camera projection matrices $\mathbf{P}_k$ are known or previously estimated using known methods (e.g., see [8, 18]). Our goal is to estimate the layer sprites L_l, the plane vectors $\mathbf{n}_l$, and the residual depths Z_l.

To compute these quantities, we initially assume that the opacities are boolean, but later refine the sprites allowing real-valued opacities. We introduce auxiliary boolean mask images B_{kl} that denote the pixels in image I_k which are images of points in layer L_l. Thus, $B_{kl} = 1$ if and only if L_l is the front-most layer which is opaque at that pixel in image I_k. Hence, in addition to L_l, $\mathbf{n}_l$, and Z_l, we also need to estimate the boolean masks B_{kl}. Once we have estimated these masks, we can compute masked input images $M_{kl} = B_{kl} \cdot I_k$ (see Figure 2).

Given any three of L_l, $\mathbf{n}_l$, Z_l, and B_{kl}, there are techniques for estimating the remaining one. Our algorithm therefore consists of first initializing these quantities. Then,

we iteratively estimate each of theses quantities in turn fixing the other three.

A number of automatic techniques have been developed to initialize the layers, e.g, merging [19, 13, 5], splitting [9, 13], color segmentation [3] and plane fitting to a recovered depth map. For the present, we interactively initialize the layers – an automatic method for layer initialization is currently a topic of research. However, in many applications, such as model acquisition [7] and video parsing [13], using a semi-automatic algorithm and requiring limited user input is acceptable.

3.1 Estimation of Layer Sprites

In order to compute the plane equation vector $\mathbf{n}_l$, we need to be able to map points in masked image M_{kl} onto the plane $\mathbf{n}_l^T\mathbf{x} = 0$. If $\mathbf{x}$ is a 3D world coordinate of a point and $\mathbf{u}_k$ is the image of $\mathbf{x}$ in camera $\mathbf{P}_k$, we have:

$$\mathbf{u}_k \;=\; \mathbf{P}_k\mathbf{x}$$

where equality is in the 2D projective space $\mathcal{P}^2$.

It can be shown see ([2]) that we can map this point onto its image in another camera $\mathbf{P}_{k'}$ as follows:

$$\mathbf{u}_{k'} \;=\; \mathbf{P}_{k'}\left((\mathbf{n}_l^T\mathbf{p}_k)\mathbf{I} - \mathbf{p}_k\mathbf{n}_l^T\right)\mathbf{P}_k^*\mathbf{u}_k \;\equiv\; \mathbf{H}_{kk'}^l\mathbf{u}_k \quad (5)$$

where $\mathbf{H}_{kk'}^l$ is a homography (collineation of $\mathcal{P}^2$). $\mathbf{P}_k^* = \mathbf{P}_k^T(\mathbf{P}_k\mathbf{P}_k^T)^{-1}$ is the pseudoinverse of $\mathbf{P}_k$, and $\mathbf{p}_k$ is a vector in the null space of $\mathbf{P}_k$, i.e. $\mathbf{P}_k\mathbf{p}_k = 0$. If $\mathbf{x}$ lies on the plane $\mathbf{n}_l^T\mathbf{x} = 0$ Equation (5) describes the image coordinate warp between the two images M_{kl} and $M_{k'l}$ which would hold if all the masked image pixels were images of world points on the plane $\mathbf{n}_l^T\mathbf{x} = 0$. Using this relation, we can warp all of the masked images onto the coordinate frame of one distinguished image w.l.o.g. image M_{1l}, as follows:

$$\left(\mathbf{H}_{1k}^l \circ M_{kl}\right)(\mathbf{u}_1) \;\equiv\; M_{kl}\left(\mathbf{H}_{1k}^l\mathbf{u}_1\right).$$

Here, $\mathbf{H}_{1k}^l \circ M_{kl}$ is the masked image M_{kl} warped into the coordinate frame of M_{1l}.

We can therefore solve for $\mathbf{n}_l$ by finding the value for which the homographies $\mathbf{H}_{1k}^l$ defined in Equation (5) best register the images onto each other. Typically, this value is found using some form of gradient decent, such as the Gauss-Newton method, and the optimization is performed in a hierarchical (i.e. pyramid based) fashion to avoid local extrema [4].

In order to create the layer sprite images, we need to choose 2D coordinate systems for the planes. Such coordinate systems can be specified by a collection of arbitrary (rank 3) camera matrices $\mathbf{Q}_l$.[1] The image coordinates $\mathbf{u}_k$ of the point in image M_{kl} which is projected onto the point $\mathbf{u}_l$ on the plane $\mathbf{n}_l^T\mathbf{x} = 0$ is given by:

$$\mathbf{u}_k \;=\; \mathbf{P}_k\left((\mathbf{n}_l^T\mathbf{q}_l)\mathbf{I} - \mathbf{q}_l\mathbf{n}_l^T\right)\mathbf{Q}_l^*\mathbf{u}_l \;\equiv\; \mathbf{H}_k^l\mathbf{u}_k \quad (6)$$

[1] One possible choice for $\mathbf{Q}_l$ would be one of the camera matrices $\mathbf{P}_k$.

where $\mathbf{Q}_l^*$ is the pseudo-inverse of $\mathbf{Q}_l$ and $\mathbf{q}_l$ is a vector in the null space of $\mathbf{Q}_l$. The homography $\mathbf{H}_k^l$ warps the coordinate frame of the plane backward onto that of image M_{kl}. The homography can be used to warp the image M_{kl} forward onto the plane, the result of which is denoted $\mathbf{H}_k^l \circ M_{kl}$. After we have warped the masked image onto the plane, we can estimate the layer sprite (with boolean opacities) by "blending" the warped images:

$$L_l \;=\; \bigoplus_{k=1}^{K} \mathbf{H}_k^l \circ M_{kl} \quad (7)$$

where $\bigoplus$ is a blending operator.

There are a number of ways the blending could be performed. One simple method would be to take the mean of the color or intensity values. A refinement would be to use a "feathering" algorithm, where the averaging is weighted by the distance of each pixel from the nearest invisible pixel in M_{kl} [15]. Alternatively, robust techniques could be used to estimate L_l from the warped images. The simplest example of such a technique is the median, but many others exist. Finally, it is also possible to model effects such as the bias and gain of the cameras during the estimation of L_l. If such effects have not already been normalized, their magnitudes can be estimated and corrected for.

3.2 Estimation of Residual Depth

In general, the scene will not be exactly piecewise planar. To model any non-planarity, we assume that the point $\mathbf{u}_l$ on the plane $\mathbf{n}_l^T\mathbf{x} = 0$ is displaced slightly. We assume it is displaced in the direction of the ray through $\mathbf{u}_l$ defined by the camera matrix $\mathbf{Q}_l$, and that the distance it is displaced is $Z_l(\mathbf{u}_l)$, measured in the direction normal to the plane. In this case, the homographic warps used in the previous section are not applicable. However, using a similar argument to that in Sections 3.1 and 3.1, it is easy to show (see also [10, 7]) that:

$$\mathbf{u}_k \;=\; \mathbf{H}_k^l\mathbf{u}_l + Z_l(\mathbf{u}_l)\mathbf{t}_{kl} \quad (8)$$

where $\mathbf{H}_k^l = \mathbf{P}_k\left((\mathbf{n}_l^T\mathbf{q}_l)\mathbf{I} - \mathbf{q}_l\mathbf{n}_l^T\right)\mathbf{Q}_l^*$ is the planar homography of Section 3.1, $\mathbf{t}_{kl} = \mathbf{P}_k\mathbf{q}_l$ is the epipole, and it is assumed that the plane equation vector $\mathbf{n}_l = (n_x, n_y, n_z, n_d)^T$ has been normalized so that $n_x^2 + n_y^2 + n_z^2 = 1$. Equation (8) can be used to map plane coordinates $\mathbf{u}_l$ backwards to image coordinates $\mathbf{u}_k$, or to map the image M_{kl} forwards onto the plane. We denote the result of this warp by $(\mathbf{H}_k^l, \mathbf{t}_{kl}, Z_l) \circ M_{kl}$, or $\mathbf{W}_k^l \circ M_{kl}$ for more concise notation.

To compute the residual depth map Z_l, we could optimize the same (or a similar) consistency metric as that used in Section 2.2 to estimate the plane equation. Doing so is essentially solving a simpler (or what [7] would call "model-based") stereo problem. In fact, almost any stereo

algorithm could be used to compute Z_l. The one property the algorithm should have is that it favors small disparities.

3.3 Pixel Assignment to Layers

In the previous three sections, we have assumed a known assignment of pixels to layer, i.e., known boolean masks B_{kl} which allow us to compute the masked image M_{kl} using $M_{kl} = B_{kl} \cdot I_k$. We now describe how to estimate the pixel assignments from $\mathbf{n}_l$, L_l, and Z_l.

We could try to update the pixel assignments by comparing the warped images $\mathbf{W}_k^l \circ M_{kl}$ to the layer sprite images L_l. However, if we compared these images, we would not be able to deduce anything about the pixel assignments outside of the current estimates of the masked regions. To allow the boolean mask B_{kl} to "grow", we therefore compare $\mathbf{W}_k^l \circ I_k$ with:

$$\tilde{L}_l = \bigoplus_{k=1}^{K} \mathbf{W}_k^l \circ \tilde{M}_{kl},$$

where $\tilde{M}_{kl} = \tilde{B}_{kl} \cdot I_k$ and $\tilde{B}_{kl}$ is a dilated version of B_{kl} (if necessary, Z_l is also enlarged so that it declines to zero outside the masked region).

Given the enlarged layer sprites $\tilde{L}_l$, our approach to pixel assignment is as follows. We first compute a measure $P_{kl}(\mathbf{u}_l)$ of the likelihood that the pixel $\mathbf{W}_k^l \circ I_k(\mathbf{u}_l)$ is the warped image of the pixel $\mathbf{u}_l$ in the enlarged layer sprite $\tilde{L}_l$. Next, P_{kl} is warped back into the coordinate system of the input image I_k to yield :

$$\hat{P}_{kl} = (\mathbf{W}_k^l)^{-1} \circ P_{kl}.$$

This warping tends to blur P_{kl}, but this is acceptable since we will want to smooth the pixel assignment anyway. The pixel assignment can then be computed by choosing the best possible layer for each pixel:

$$B_{lk}(\mathbf{u}_k) = \begin{cases} 1 & \text{if } \hat{P}_{kl}(\mathbf{u}_k) = \min_{l'} \hat{P}_{kl'}(\mathbf{u}_k) \\ 0 & \text{otherwise} \end{cases}.$$

There are a number of possible ways of defining P_{kl}. The simplest is the *residual intensity difference* [13]. Another possibility is the *residual normal flow magnitude*:

$$P_{kl} = \frac{\|\mathbf{W}_k^l \circ I_k - \tilde{L}_l\|}{\|\nabla \tilde{L}_l\|}.$$

A third possibility would be to compute the optical flow between $\mathbf{W}_k^l \circ I_k$ and $\tilde{L}_l$ and then use the magnitude of the flow for P_{kl}.

3.4 Real-valued Opacities

The layered stereo algorithm described above is limited to recovering binary masks B_{kl} for the assignment of input pixels to layers. While this may be true for a large class of scenes and images, there are two cases where this is not adequate. First is when the scene contains partially transparent surfaces and/or inter surface reflections. The second occurs in most images at the boundary between layers. At these boundaries, the color values in an input image will be a combination of the color of the layers on either side of that boundary due to the finite resolution of the image pixels. Such "mixed" pixels can be modeled as having real-valued opacities.

Our current algorithms are restricted to boolean-valued opacities. Our proposed approach for recovering real-valued opacities is to refine the layer estimates by minimizing the prediction error:

$$C = \sum_k \sum_{\mathbf{u}_k} \|S_k(\mathbf{u}_k) - I_k(\mathbf{u}_k)\|^2 \qquad (9)$$

using a gradient descent algorithm. (In order to further constrain the space of possible solutions, we can add smoothness constraints on the colors and opacities [16].) Rather than trying to optimize over all of the parameters (L_l, $\mathbf{n}_l$, and Z_l) simultaneously, we only adjust the sprite colors and opacities in L_l, and then re-run the previous motion estimation steps to adjust $\mathbf{n}_l$ and Z_l.

This approach is currently under development.

4 Examples

We illustrate the layer sprite representation via two examples. The first one was briefly described in Section 1. The second consists of 40 images taken simultaneously. The camera geometry is not given for either sequence, so we used point tracking and a standard structure from motion algorithm to estimate the camera matrices. To initialize our algorithm, we first decided how many layers were required, and then performed a rough assignment of pixels to layers by hand. Next, the automatic hierarchical parametric motion estimation algorithm described in [17] was used to find the 8-parameter homographies between the layers and estimate the layer sprites. (For the experiments presented in this paper, we set $\mathbf{Q}_l = \mathbf{P}_1$, i.e. we reconstructed the sprites in the coordinate system of the first camera.) Using the computed homographies, we found the best plane estimate for each layer using a Euclidean structure from motion algorithm.

The results of applying these steps to the MPEG *flower garden* sequence are shown in Figure 1. Figures 1(a) and (b) show the first and last image in the subsequence we used (the first nine even images). Figures 1(c) shows the sprite images corresponding to each of the six layers, re-arranged for more compact display. (These sprites are actually the ones computed after residual depth estimation.) Note that because of the blending that takes place during sprite construction, each sprite is larger than its footprint in any one of the input images. (e.g., the background house sprite is complete, even though it is always seen in two pieces in each image). This sprite representation makes it very easy to re-synthesize novel images without leaving gaps in the

Figure 3: Results on the *symposium* sequence: (a) third of five images; (b) fifth of five images; (c) recovered depth map (darker denotes closer); (d) and (e) the five layer sprites; (f) residual depth image for fifth layer.

new image, unlike approaches based on a single painted depth map [11]. Also notice how the depth discontinuities are much crisper and cleaner than those available with traditional stereo correspondence algorithms.

Our second set of experiments uses five images of a 40-image stereo data set taken at a graphics symposium. Figure 3(a) shows the middle input image, Figure 3(b) shows one of the other images, Figure 3(c) shows the recovered planar depth map, and Figure 3(f) shows the residual depth map for one of the layers. Figures 3(d) and (e) show the recovered sprites. Figure 4(a) shows the middle image resynthesized from these sprites.

Figure 4(b) shows the same sprite collection seen from a novel viewpoint (well outside the range of the original views). The gaps in this figure correspond to parts of the scene which where not visible in any of the five input images. Figure 4(c) shows a synthesized scene in which we merged layer sprites selected from each of our two examples into a single scene.

5 Discussion

In this section, we briefly discuss the domain of applicability and usefulness of the proposed representation. The layer sprite representation is suitable for a class of image based modeling problems, namely that of modeling a static 3D environment. In this class of problems, especially for outdoor scenes or other large scale environments, typically the range of camera positions is small compared to the overall extent of the environment. In this case, the observed disparities of local portions of the scene (e.g., the ground surface, a wall, group of trees, etc.) may be approximated by a planar homography. The residual disparities will be small, and their effects during reprojection to new views will be correspondingly small.

The generative process and the resulting reprojection technique associated with the layer sprite representation has the following property. Rendering the scene simply with the sprite images and the associated plane equations, but *ignoring the residual depths* will result in an image that is not exact, but still a good approximation to the actual image. Moreover, the large scale occlusion effects are easily and naturally handled. This is akin to treating the scene as a collection of "cardboard cutouts" at different orientations. This representation is highly compact, since the data acquired from a collection of several images is condensed into a few sprites, whose total data volume is slightly more than that of a single image. The cost of encoding the plane parameters is negligible. The residual depth nearly doubles the data volume. This representation also lends itself easily to progressive transmission and rendering.

However, for other types of modeling problems such as compact 3D object modeling, it is not clear that this representation is particularly suitable. In these problems, the object is typically viewed from a variety of directions (per-

(a) (b) (c)

Figure 4: 3D views of the reconstructed *symposium* scene: (a) re-synthesized third image (note extended field of view). (b) novel view without residual depth; (c) a synthesized scene containing elements from both our examples.

haps from all around the object), and there is a considerable change in what is seen across these views. The viewpoints are likely to be close to the object, and the parallax effects between different portions of the object are likely to be significant. Also, a single object is usually modeled and a detailed representation of its shape is critical for good visual presentation. In such cases, the planar layer + residual depth representation is not likely to be compact. Moreover, manipulations are likely to involve object deformations (e.g., morphing), articulation of parts, etc. These are not well supported by the proposed layer sprite representations.

6 Conclusion

We have described an image based approach for modeling 3D scenes based on multiple 2D images of the scene. We have proposed a representational framework that decomposes the scene into a collection of 2.5D layer sprites that encode portions of the scene. We have described the basic algorithmic steps involved in the creation of this representation from a given set of images and a method for reprojecting the representation into new views of the scene. The proposed representation is naturally suitable for modeling 3D environments and meets many of the requirements for multimedia authoring and presentation applications. Our work in this area continues, with particular focus on the automatic segmentation of the scene into sprites.

References

[1] Workshop on image-based modeling and rendering. //graphics.stanford.edu/im98/, March 1998.

[2] S. Baker, R. Szeliski, and P. Anandan. A Layered Approach to Stereo Reconstruction. In *CVPR 98*, pages 434-441, June 1998.

[3] S. Ayer, P. Schroeter, and J. Bigun. Segmentation of moving objects by robust parameter estimation over multiple frames. In *3rd ECCV*, pages 316–327, 1994.

[4] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *2nd ECCV*, pages 237–252, 1992.

[5] M.J. Black and A.D. Jepson. Estimating optical flow in segmented images using variable-order parametric models with local deformations. *PAMI*, 18(10):972–986, 1996.

[6] J.F. Blinn. Jim Blinn's corner: Compositing, part 1: Theory. *IEEE Computer Graphics and Applications*, 14(5):83–87, September 1994.

[7] P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96*, pages 11–20, 1996.

[8] O.D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[9] M. Irani, P. Anandan, and S. Hsu. Mosiac based representations of video sequences and their applications. In *5th ICCV*, pages 605–611, 1995.

[10] R. Kumar, P. Anandan, and K. Hanna. Direct recovery of shape from multiple views: A parallax based approach. In *12th ICPR*, pages 685–688, 1994.

[11] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH '95*, pages 39–46, 1995.

[12] T. Porter and T. Duff. Compositing digital images. *SIGGRAPH '84*, pages 253–259, 1984.

[13] H.S. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *PAMI*, 18(8):814–830, 1996.

[14] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. In *Computer Graphics (SIGGRAPH'98) Proceedings*, pages 231–242, Orlando, July 1998. ACM SIGGRAPH.

[15] H.-Y. Shum and R. Szeliski. Panoramic image mosaicing. Technical Report MSR-TR-97-23, Microsoft Research, September 1997.

[16] R. Szeliski and P. Golland. Stereo matching with transparency and matting. In *6th ICCV*, 1998.

[17] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and texture-mapped models. In *SIGGRAPH '97*, pages 251–258, 1997.

[18] R. Szeliski and P. Torr. Geometrically constrained structure from motion: Points on planes. In *European Workshop on 3D Structure from Multiple Images of Large-scale Environments (SMILE)*, (Submitted) 1998.

[19] J.Y.A. Wang and E.H. Adelson. Layered representation for motion analysis. In *CVPR '93*, pages 361–366, 1993.