Creating Full View Panoramic Image Mosaics and Environment Maps

Richard Szeliski and Heung-Yeung Shum

Microsoft Research

Abstract

This paper presents a novel approach to creating full view panoramic mosaics from image sequences. Unlike current panoramic stitching methods, which usually require pure horizontal camera panning, our system does not require any controlled motions or constraints on how the images are taken (as long as there is no strong motion parallax). For example, images taken from a hand-held digital camera can be stitched seamlessly into panoramic mosaics. Because we represent our image mosaics using a set of transforms, there are no singularity problems such as those existing at the top and bottom of cylindrical or spherical maps. Our algorithm is fast and robust because it directly recovers 3D rotations instead of general 8 parameter planar perspective transforms. Methods to recover camera focal length are also presented. We also present an algorithm for efficiently extracting environment maps from our image mosaics. By mapping the mosaic onto an artibrary texture-mapped polyhedron surrounding the origin, we can explore the virtual environment using standard 3D graphics viewers and hardware without requiring special-purpose players.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.4 [Image Processing]: Enhancement - Registration.

Additional Keywords: full-view panoramic image mosaics, environment mapping, virtual environments, image-based rendering.

1 Introduction

Image-based rendering is a popular way to simulate a visually rich tele-presence or virtual reality experience. Instead of building and rendering a complete 3D model of the environment, a collection of images is used to render the scene while supporting virtual camera motion. For example, a single cylindrical image surrounding the viewer enables the user to pan and zoom inside an environment created from real images [4, 13]. More powerful image-based rendering systems can be built by adding a depth map to the image [3, 13], or using a larger collection of images [3, 6, 11].

In this paper, we focus on image-based rendering systems without any depth information, i.e., those which only support user panning, rotation, and zoom. Most of the commercial products based on this idea (such as QuickTime VR [22] and Surround Video [23]) use cylindrical images with a limited vertical field of view, although newer systems support full spherical maps (e.g., PhotoBubble [24], Infinite Pictures [25], and RealVR [26]).

A number of techniques have been developed for capturing panoramic images of real-world scenes (for references on computer-generated environment maps, see [7]). One way is to record an image onto a long film strip using a panoramic camera to directly capture a cylindrical panoramic image [14]. Another way is to use a lens with a very large field of view such as a fisheye lens. Mirrored pyramids and parabolic mirrors can also be used to directly capture panoramic images [27, 28].

A less hardware-intensive method for constructing full view panoramas is to take many regular photographic or video images in order to cover the whole viewing space. These images must then be aligned and composited into complete panoramic images using an image mosaic or "stitching" algorithm [12, 17, 9, 4, 13, 18]. Most stitching systems require a carefully controlled camera motion (pure pan), and only produce cylindrical images [4, 13]. In this paper, we show how uncontrolled 3D camera rotation can be used.

The case of general camera rotation has been studied previously [12, 9, 18], using an 8-parameter planar perspective motion model. By contrast, our algorithm uses a 3-parameter rotational motion model, which is more robust since it has fewer unknowns. Since this algorithm requires knowing the camera's focal length, we develop a method for computing an initial focal length estimate from a set of 8-parameter perspective registrations. We also investigate how to close the "gap" (or "overlap") due to accumulated registration errors after a complete panoramic sequence has been assembled. To demonstrate the advantages of our algorithm, we apply it to a sequence of images taken with a handheld digital camera.

In our work, we represent our mosaic by a set of transformations. Each transformation corresponds to one image frame in the input image sequence and represents the mapping between image pixels and viewing directions in the world, i.e., it represents the *camera matrix* [5]. During the stitching process, our approach makes no commitment to the final output representation (e.g. spherical or cylindrical), which allows us to avoid the singularities associated with such representations.

Once a mosaic has been constructed, it can, of course, be mapped into cylindrical or spherical coordinates, and displayed using a special purpose viewer [4]. In this paper, we argue that such specialized representations are not necessary, and represent just a particular choice of geometry and texture coordinate embedding. Instead, we show how to convert our mosaic to an environment map [7], i.e., how to map our mosaic onto *any* texture-mapped polyhedron surrounding the origin. This allows us to use standard 3D graphics APIs and 3D model formats, and to use 3D graphics accelerators for texture mapping.

The remainder of our paper is structured as follows. Sections 2 and 3 review our algorithms for panoramic mosaic construction using cylindrical coordinates and general perspective transforms. Section 4 describes our novel direct rotation recovery algorithm. Section 5 presents our technique for estimating the focal length from perspective registrations. Section 6 discusses how to eliminate the "gap" in a panorama due to accumulated registration errors. Section 7 presents our algorithm for projecting our panoramas onto texture-mapped 3D models (environment maps). We close with a discussion and a description of ongoing and future work.

2 Cylindrical and spherical panoramas

Cylindrical panoramas are commonly used because of their ease of construction. To build a cylindrical panorama, a sequence of images is taken by a camera mounted on a leveled tripod. If the camera focal length or field of view is known, each perspective image can be warped into cylindrical coordinates. Figure 1a shows two overlapping cylindrical images—notice how horizontal lines become curved.

To build a cylindrical panorama, we map world coordinates $\mathbf{p} = (X, Y, Z)$ to 2D cylindrical screen coordinates (θ, v) using

$$\theta = \tan^{-1}(X/Z), \quad v = Y/\sqrt{X^2 + Z^2} \tag{1}$$

where θ is the panning angle and v is the scanline [18]. Similarly, we can map world coordinates into 2D spherical coordinates (θ, ϕ) using

$$\theta = \tan^{-1}(X/Z), \quad \phi = \tan^{-1}(Y/\sqrt{X^2 + Z^2}). \tag{2}$$

Once we have warped each input image, constructing the panoramic mosaics becomes a pure translation problem. Ideally, to build a cylindrical or spherical panorama from a horizontal panning sequence, only the unknown panning angles need to be recovered. In practice, small vertical translations are needed to compensate for vertical jitter and optical twist. Therefore, both a horizontal translation t_x and a vertical translation t_y are estimated for each input image.

To recover the translational motion, we estimate the incremental translation $\delta\mathbf{t} = (\delta t_x, \delta t_y)$ by minimizing the intensity error between two images,

$$E(\delta\mathbf{t}) = \sum_i [I_1(\mathbf{x}'_i + \delta\mathbf{t}) - I_0(\mathbf{x}_i)]^2, \tag{3}$$

where $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}'_i = (x'_i, y'_i) = (x_i + t_x, y_i + t_y)$ are corresponding points in the two images, and $\mathbf{t} = (t_x, t_y)$ is the global translational motion field which is the same for all pixels [2].

After a first order Taylor series expansion, the above equation becomes

$$E(\delta\mathbf{t}) \approx \sum_i [\mathbf{g}_i^T \delta\mathbf{t} + e_i]^2 \tag{4}$$

where $e_i = I_1(\mathbf{x}'_i) - I_0(\mathbf{x}_i)$ is the current intensity or color error, and $\mathbf{g}_i^T = \nabla I_1(\mathbf{x}'_i)$ is the image gradient of I_1 at $\mathbf{x}'_i$. This minimization problem has a simple least-squares solution,

$$\left(\sum_i \mathbf{g}_i \mathbf{g}_i^T\right) \delta\mathbf{t} = -\left(\sum_i e_i \mathbf{g}_i\right). \tag{5}$$

Figure 1b shows a portion of a cylindrical panoramic mosaic built using this simple translational alignment technique. To handle larger initial displacements, we use a hierarchical coarse-to-fine optimization scheme [2]. To reduce discontinuities in intensity and color between the images being composited, we apply a simple *feathering* algorithm, i.e., we weight the pixels in each image proportionally to their distance to the edge (or more precisely, their distance to the nearest invisible pixel) [18]. Once registration is finished, we can clip the ends (and optionally the top and bottom), and write out a single panoramic image.

Creating panoramas in cylindrical or spherical coordinates has several limitations. First, it can only handle the simple case of pure panning motion. Second, even though it is possible to convert an image to 2D spherical or cylindrical coordinates for a known tilting angle, ill-sampling at north pole and south pole causes big registration errors. Third, it requires knowing the focal length (or equivalently, field of view). While focal length can be carefully calibrated in the lab[19, 16], estimating the focal length of lens by registering two or more images is not very accurate, as we will discuss in section 5.

3 Perspective (8-parameter) panoramas

To overcome these limitations, several authors have suggested using full planar perspective motion models [12, 9, 18]. The planar perspective transform warps an image into another using 8 parameters,

$$\mathbf{x}' \sim \mathbf{M}\mathbf{x} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{6}$$

where $\mathbf{x} = (x, y, 1)$ and $\mathbf{x}' = (x', y', 1)$ are homogeneous or projective coordinates, and $\sim$ indicates equality up to scale. This equation can be re-written as

$$x' = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + 1} \tag{7}$$

$$y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1} \tag{8}$$

(in translational motion, only the two parameters m_2 and m_5 are used).

To recover the 8 paramters, we iteratively update the transform matrix using

$$\mathbf{M} \leftarrow (\mathbf{I} + \mathbf{D})\mathbf{M} \tag{9}$$

where

$$\mathbf{D} = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_3 & d_4 & d_5 \\ d_6 & d_7 & 0 \end{bmatrix}. \tag{10}$$

Resampling image I_1 with the new transformation $\mathbf{x}' \sim (\mathbf{I} + \mathbf{D})\mathbf{M}\mathbf{x}$ is the same as warping the resampled image $\tilde{I}_1(\mathbf{x}_i) = I_1(\mathbf{x}'_i)$ by $\mathbf{x}'' \sim (\mathbf{I} + \mathbf{D})\mathbf{x}$, i.e.,

$$x'' = \frac{(1 + d_0)x + d_1 y + d_2}{d_6 x + d_7 y + 1} \tag{11}$$

$$y'' = \frac{d_3 x + (1 + d_4)y + d_5}{d_6 x + d_7 y + 1}. \tag{12}$$

Again, we wish to minimize

$$E(\mathbf{d}) = \sum_i [\tilde{I}_1(\mathbf{x}''_i) - I_0(\mathbf{x}_i)]^2 \tag{13}$$

$$\approx \sum_i [\mathbf{g}_i^T \mathbf{J}_i^T \mathbf{d} + e_i]^2 \tag{14}$$

where $\mathbf{d} = (d_0, \ldots, d_7)$ is the incremental update parameter, and $\mathbf{J}_i = \mathbf{J}_\mathbf{d}(\mathbf{x}_i)$, where

$$\mathbf{J}_\mathbf{d}(\mathbf{x}) = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}^T \tag{15}$$

is the Jacobian of the resampled point coordinate $\mathbf{x}''_i$ with respect to $\mathbf{d}$. The entries in the Jacobian correspond to the optical flow induced by the instantaneous motion of a plane in 3D [2]. The least-squares minimization problem (14) is solved using *normal equations* analogous to (5)

$$\mathbf{A}\mathbf{d} = -\mathbf{b}, \tag{16}$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_i \mathbf{g}_i \mathbf{g}_i^T \mathbf{J}_i^T \tag{17}$$

is the *Hessian*, and

$$\mathbf{b} = \sum_i e_i \mathbf{J}_i \mathbf{g}_i \tag{18}$$

is the *accumulated gradient* or *residual*.

The 8-parameter perspective transformation recovery algorithm works well provided that initial estimates of the correct transformation are close enough. However, since the motion model contains more free parameters than necessary, it suffers from slow convergence and sometimes gets stuck in local minima. For this reason, we prefer to use the 3-parameter rotational model described next.

4 Rotational (3-parameter) panoramas

For a camera centered at the origin, the relationship between a 3D point $\mathbf{p} = (X, Y, Z)$ and its image coordinates $\mathbf{x} = (x, y, 1)$ can be described by

$$\mathbf{x} \sim \mathbf{TVRp}, \tag{19}$$

where

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ and } \mathbf{R} = \begin{bmatrix} \ r_{ij} \ \end{bmatrix}$$

are the image plane translation, focal length scaling, and 3D rotation matrices. For simplicity of notation, we assume that pixels are numbered so that the origin is at the image center, i.e., $c_x = c_y = 0$, allowing us to dispense with $\mathbf{T}$ (in practice, mislocating the image center does not seem to affect mosaic registration algorithms very much). The 3D direction corresponding to a screen pixel $\mathbf{x}$ is given by $\mathbf{p} \sim \mathbf{R}^{-1}\mathbf{V}^{-1}\mathbf{x}$.

For a camera rotating around its center of projection, the mapping (perspective projection) between two images k and l is therefore given by

$$\mathbf{M} \sim \mathbf{V}_k \mathbf{R}_k \mathbf{R}_l^{-1} \mathbf{V}_l^{-1} \tag{20}$$

where each image is represented by $\mathbf{V}_k\mathbf{R}_k$, i.e., a focal length and a 3D rotation.

Assume for now that the focal length is known and is the same for all images, i.e, $\mathbf{V}_k = \mathbf{V}$. To recover the rotation, we perform an incremental update to $\mathbf{R}_k$ based on the angular velocity $\mathbf{\Omega} = (\omega_x, \omega_y, \omega_z)$,

$$\mathbf{M} \leftarrow \mathbf{V}\hat{\mathbf{R}}(\mathbf{\Omega})\mathbf{R}_k\mathbf{R}_l^{-1}\mathbf{V}^{-1} \tag{21}$$

where the incremental rotation matrix $\hat{\mathbf{R}}(\mathbf{\Omega})$ is given by Rodriguez's formula [1],

$$\hat{\mathbf{R}}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + \sin\theta \mathbf{X}(\hat{\mathbf{n}}) + (1 - \cos\theta)\mathbf{X}(\hat{\mathbf{n}})^2 \tag{22}$$

with $\theta = \|\mathbf{\Omega}\|$, $\hat{\mathbf{n}} = \mathbf{\Omega}/\theta$, and

$$\mathbf{X}(\mathbf{\Omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

is the cross product operator. Keeping only terms linear in $\mathbf{\Omega}$, we get

$$\mathbf{M}' \approx \mathbf{V}[\mathbf{I} + \mathbf{X}(\mathbf{\Omega})]\mathbf{R}_k\mathbf{R}_l^{-1}\mathbf{V}^{-1} = (\mathbf{I} + \mathbf{D}_\Omega)\mathbf{M}, \tag{23}$$

where

$$\mathbf{D}_\Omega = \mathbf{V}\mathbf{X}(\mathbf{\Omega})\mathbf{V}^{-1} = \begin{bmatrix} 0 & -\omega_z & f\omega_y \\ \omega_z & 0 & -f\omega_x \\ -\omega_y/f & \omega_x/f & 0 \end{bmatrix}$$

is the deformation matrix which plays the same role as $\mathbf{D}$ in (9).

Computing the Jacobian of the entries in $\mathbf{D}_\Omega$ with respect to $\mathbf{\Omega}$ and applying the chain rule, we obtain the new Jacobian,

$$\mathbf{J}_\Omega = \frac{\partial \mathbf{x}''}{\partial \mathbf{\Omega}} = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}} \frac{\partial \mathbf{d}}{\partial \mathbf{\Omega}} = \begin{bmatrix} -xy/f & f + x^2/f & -y \\ -f - y^2/f & xy/f & x \end{bmatrix}^T. \tag{24}$$

This Jacobian is then plugged into the previous minimization pipeline to estimate the incremental rotation vector $(\omega_x\ \omega_y\ \omega_z)$, after which $\mathbf{R}_k$ can be updated using (21).

Figure 2 shows how our method can be used to register four images with arbitrary (non-panning) rotation. Compared to the 8-parameter perspective model, it is much easier and more intuitive to interactively adjust images using the 3-parameter rotational model.

5 Estimating the focal length

In order to apply our 3D rotation technique, we must first obtain an estimate for the camera's focal length. A convenient way to obtain this estimate to deduce the value from one or more perspective transforms computed using the 8-parameter algorithm. Expanding the $\mathbf{V}_1\mathbf{R}\mathbf{V}_0^{-1}$ formulation, we have

$$\mathbf{M} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \sim \begin{bmatrix} r_{00} & r_{01} & r_{02}f_0 \\ r_{10} & r_{11} & r_{12}f_0 \\ r_{20}/f_1 & r_{21}/f_1 & r_{22}f_0/f_1 \end{bmatrix} \tag{25}$$

where $\mathbf{R} = [r_{ij}]$.

In order to estimate focal lengths f_0 and f_1, we observe that the first two rows (columns) of $\mathbf{R}$ must have the same norm and be orthogonal (even if the matrix is scaled), i.e.,

$$m_0{}^2 + m_1{}^2 + m_2{}^2/f_0{}^2 = m_3{}^2 + m_4{}^2 + m_5{}^2/f_0{}^2 \tag{26}$$

$$m_0 m_3 + m_1 m_4 + m_2 m_5 / f_0{}^2 = 0. \tag{27}$$

From this, we can compute the estimates

$$f_0{}^2 = \sqrt{\frac{m_0{}^2 + m_1{}^2 - m_3{}^2 - m_4{}^2}{m_5{}^2 - m_2{}^2}} \quad \text{if } m_5 \neq m_2$$

or

$$f_0{}^2 = -\frac{m_0 m_3 + m_1 m_4}{m_2 m_5} \quad \text{if } m_5 \neq 0 \text{ and } m_2 \neq 0.$$

Similar result can be obtained for f_1 as well. If the focal length is fixed for two images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, the median value is used as the final estimate.

Alternative techniques for estimating the focal length are presented in [8, 16, 13, 10]. The first technique [8] uses more than two frames and assumes a more general camera model (e.g., unknown optical center and aspect ratio). The other techniques either assume known rotation angles or use a complete panorama (similar to the technique described in section 6).

Once an initial set of f estimates is available, we can improve these estimates as part of the image registration process, using the same kind of least squares approach as for the rotation [15].

6 Closing the gap in a panorama

Even with our best algorithms for recovering rotations and focal length, when a complete panoramic sequence is stitched together, there will invariably be either a gap or an overlap (due to accumulated errors in the rotation estimates). We solve this problem by

registering the same image at both the beginning and the end of the sequence.

The difference in the rotation matrices (actually, their quotient) directly tells us the amount of misregistration. This error can be distributed evenly across the whole sequence by converting the error in rotation into a quaternion, and dividing the quaternion by the number of images in the sequence (for lack of a better guess).

We can also update the estimated focal length based on the amount of misregistration. To do this, we first convert the quaternion describing the misregistration into a *gap angle* θ_g. We can then update the focal length using the equation

$$f' = \frac{360^\circ - \theta_g}{360^\circ} * f. \tag{28}$$

Figure 3a shows the end of registered image sequence and the first image. There is a big gap between the last image and the first which are in fact the same image. The gap is 32° because the wrong estimate of focal length (510) was used. Figure 3b shows the registration after closing the gap with the correct focal length (468). Notice that both mosaics show very little visual misregistration (except at the gap), yet Figure 3a has been computed using a focal length which has 9% error.

Related approaches have been developed by [13, 16, 10] to solve the focal length estimation problem using pure panning motion and cylindrical images. In recent work, we have developed an alternative approach to removing gaps and overlaps which works for arbitrary image sequences (see Section 8).

7 Environment map construction

Once we have constructed a complete panoramic mosaic, we need to convert the set of input images and associated transforms into one or more images which can be quickly rendered or viewed.

A traditional way to do this is to choose either a cylindrical or spherical map (Section 2). When being used as an environment map, such a representation is sometimes called a latitude-longitude projection [7]. The color associated with each pixel is computed by first converting the pixel address to a 3D ray, and then mapping this ray into each input image through our known transformation. The colors picked up from each image are then blended using the weighting function (feathering) described earlier. For example, we can convert our rotational panorama to spherical panorama using the following algorithm:

1. for each pixel (θ, ϕ) in the spherical map, compute its corresponding 3D position on unit sphere $\mathbf{p} = (X, Y, Z)$ where $X = cos(\phi)sin(\theta), Y = sin(\phi)$, and $Z = cos(\phi)cos(\theta)$;
2. for each $\mathbf{p}$, determine its mapping into each image k using $\mathbf{x} \sim \mathbf{T}_k\mathbf{V}_k\mathbf{R}_k\mathbf{p}$;
3. form a composite (blended) image from the above warped images.

Unfortunately, such a map requires a specialized viewer, and thus cannot take advantage of any hardware texture-mapping acceleration (without approximating the cylinder's or sphere's shape with a polyhedron, which would introduce distortions into the rendering). For true full-view panoramas, spherical maps also introduce a distortion around each pole.

As an alternative, we propose the use of traditional texture-mapped models, i.e., environment maps [7]. The shape of the model and the embedding of each face into texture space are left up to the user. This choice can range from something as simple as a cube with six separate texture maps [7], to something as complicated as a subdivided dodecahedron, or even a latitude-longitude tesselated globe.[1] This choice will depend on the characteristics of the rendering hardware and the desired quality (e.g., minimizing distortions or local changes in pixel size), and on external considerations such as the ease of painting on the resulting texture maps (since some embeddings may leave gaps in the texture map).

In this section, we describe how to efficiently compute texture map color values for any geometry and choice of texture map coordinates. A generalization of this algorithm can be used to project a collection of images onto an arbitrary model, e.g., non-convex models which do not surround the viewer.

We assume that the object model is a triangulated surface, i.e., a collection of triangles and vertices, where each vertex is tagged with its 3D (X, Y, Z) coordinates and (u, v) texture coordinates (faces may be assigned to different texture maps). We restrict the model to triangular faces in order to obtain a simple, closed-form solution (projective map, potentially different for each triangle) between texture coordinates and image coordinates. The output of our algorithm is a set of colored texture maps, with undefined (invisible) pixels flagged (e.g., if an alpha channel is used, then $\alpha \leftarrow 0$).

Our algorithm consists of the following four steps:

1. paint each triangle in (u, v) space a unique color;
2. for each triangle, determine its $(u, v, 1) \rightarrow (X, Y, Z)$ mapping;
3. for each triangle, form a composite (blended) image;
4. paint the composite image into the final texture map using the color values computed in step 1 as a stencil.

These four steps are described in more detail below.

The pseudocoloring (triangle painting) step uses an auxilliary buffer the same size as the texture map. We use an RGB image, which means that 2^{24} colors are available. After the initial coloring, we grow the colors into invisible regions using a simple dilation operation, i.e., iteratively replacing invisible pixels with one of their visible neighbor pseudocolors. This operation is performed in order to eliminate small gaps in the texture map, and to support filtering operations such as bilinear texture mapping and MIP mapping [21]. For example, when using a six-sided cube, we set the (u, v) coordinates of each square vertex to be slightly inside the margins of the texture map. Thus, each texture map covers a little more region than it needs to, but operation such a texture filtering and MIP mapping can be performed without worrying about edge effects.

In the second step, we compute the $(u, v, 1) \rightarrow (X, Y, Z)$ mapping for each triangle T by finding the 3×3 matrix $\mathbf{M}_T$ which satisfies

$$\mathbf{u}_i = \mathbf{M}_T\mathbf{p}_i$$

for each of the three triangle vertices i. Thus, $\mathbf{M}_T = \mathbf{U}\mathbf{P}^{-1}$, where $\mathbf{U} = [\mathbf{u}_0|\mathbf{u}_1|\mathbf{u}_2]$ and $\mathbf{P} = [\mathbf{p}_0|\mathbf{p}_1|\mathbf{p}_2]$ are formed by concatenating the $\mathbf{u}_i$ and $\mathbf{p}_i$ 3-vectors. This mapping is essentially a mapping from 3D directions in space (since the cameras are all at the origin) to (u, v) coordinates.

In the third step, we compute a bounding box around each triangle in (u, v) space and enlarge it slightly (by the same amount as the dilation in step 1). We then form a composite image by blending all of the input images j according to the transformation $\mathbf{u} = \mathbf{M}_T\mathbf{R}_k^{-1}\mathbf{V}_k^{-1}\mathbf{x}$. This is a full, 8-parameter perspective transformation. It is *not* the same as the 6-parameter affine map which would be obtained by simply projecting a triangle's vertices into the image, and then mapping these 2D image coordinates into 2D texture space (in essence ignoring the foreshortening in the projection

[1] This latter representation is equivalent to a spherical map in the limit as the globe facets become infinitessimally small. The important difference is that even with large facets, an exact rendering can be obtained with regular texture-mapping algorithms and hardware.

onto the 3D model). The error in applying this naive but erroneous method to large texture map facets (e.g., those of a simple unrefined cube) would be quite large.

In the fourth step, we find the pseudocolor associated with each pixel inside the composited patch, and paint the composited color into the texture map if the pseudocolor matches the face id.

Our algorithm can also be used to project a collection of images onto an arbitrary object, i.e., to do true inverse texture mapping, by extending our algorithm to handle occlusions. To do this, we simply paint the pseudocolored polyhedral model into each input image using a z-buffering algorithm (this is called an *item buffer* in ray tracing [20]). When compositing the image for each face, we then check to see which pixels match the desired pseudocolor, and set those which do not match to be invisible (i.e., not to contribute to the final composite).

Figure 4 shows the results of mapping a panoramic mosaic onto a longitude-latitude tesselated globe. The white triangles at the top are the parts of the texture map not covered in the 3D tesselated globe model (due to triangular elements at the poles). Figures 5–7 show the results of mapping three different panoramic mosaics onto cubical environment maps. We can see that the mosaics are of very high quality, and also get a good sense for the extent of viewing sphere covered by these full-view mosaics. Note that Figure 5 uses images taken with a hand-held digital camera.

Once the texture-mapped 3D models have been constructed, they can be rendered directly with a standard 3D graphics system. For our work, we are currently using a simple 3D viewer written on top of the Direct3D API running on a personal computer with no hardware graphics acceleration.

8 Discussion

In this paper, we have developed some new techniques for building full view panoramic image mosaics. Our system does not place constraints on how the input images are taken, and allows the images to be taken with hand held cameras. By taking many overlapping images, we can significantly increase the field of view of the constructed panorama and remove the need for expensive fisheye lenses. Our method is accurate and robust because we estimate only 3 unknowns in the rotation matrix instead of 8 parameters in the general perspective transforms. Our method greatly increases accuracy, flexibility, and ease of use of previous techniques. We have also developed techniques for estimating the focal length from an image sequence, and for recovering from accumulated registration errors when a full panoramic mosaic is completed.

When building an image mosaic from a long sequence of images, we have to deal with error accumulation problems. In this paper we have presented a "gap closing" technique which updates the focal length and rotation matrices after a complete panorama is constructed. More recently we have developed a new method based on *block adjustment* which simultaneously adjusts all rotation matrices and focal lengths so that the sum of registration errors between all matching pairs of images is minimized [15].

In theory, panoramas can only be constructed if all images are taken by a camera whose optical centers never moves. In practice, this depends on the amount of camera translation relative to the nearest objects in front of the camera. With our 3-D rotation mosaicing method, we have demonstrated that images taken by a hand held digital camera can be seamlessly stitched. To compensate for local misregistration caused by larger amounts of motion parallax (e.g., camera translation), we have recently developed a *deghosting* technique [15]. We divide each image into small patches and compute patch-based alignments. Each image is then locally warped so that the overall mosaic does not contain visible ghosting. This deghosting method has been used to build the image mosaic of the Space Shuttle flight deck (Figure 8) from a sequence of images (with significant motion parallax) taken by an astronant with a hand-held camera.

We have also presented an algorithm for extracting texture maps from the image mosaics. We can map image mosaics onto any 3-D model and exploit 3-D graphics hardware and APIs. Compared with using special purpose players (e.g., cylindrical and spherical viewers), our inverse texture mapping approach can be much more easily integrated as backdrops for virtual worlds and games. In the future, we would like to explore how to extract the three-dimensional world descriptions from full-view panoramic image mosaics.

References

[1] N. Ayache. *Vision Stéréoscopique et Perception Multisensorielle*. InterEditions., Paris, 1989.

[2] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Second European Conference on Computer Vision (ECCV'92)*, pages 237–252, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

[3] S. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics (SIGGRAPH'93)*, pages 279–288, August 1993.

[4] S. E. Chen. QuickTime VR – an image-based approach to virtual environment navigation. *Computer Graphics (SIGGRAPH'95)*, pages 29–38, August 1995.

[5] O. Faugeras. *Three-dimensional computer vision: A geometric viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Computer Graphics Proceedings, Annual Conference Series*, pages 43–54, Proc. SIGGRAPH'96 (New Orleans), August 1996. ACM SIGGRAPH.

[7] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[8] R. I. Hartley. Self-calibration from multiple views of a rotating camera. In *Third European Conference on Computer Vision (ECCV'94)*, volume 1, pages 471–478, Stockholm, Sweden, May 1994. Springer-Verlag.

[9] M. Irani, P. Anandan, and S. Hsu. Mosaic based representations of video sequences and their applications. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 605–611, Cambridge, Massachusetts, June 1995.

[10] S. B. Kang and R Weiss. Characterization of errors in compositing panoramic images. Technical Report 96/2, Digital Equipment Corporation, Cambridge Research Lab, June 1996.

[11] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics Proceedings, Annual Conference Series*, pages 31–42, Proc. SIGGRAPH'96 (New Orleans), August 1996. ACM SIGGRAPH.

[12] S. Mann and R. W. Picard. Virtual bellows: Constructing high-quality images from video. In *First IEEE International Conference on Image Processing (ICIP-94)*, volume I, pages 363–367, Austin, Texas, November 1994.

[13] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics (SIGGRAPH'95)*, pages 39–46, August 1995.

[14] J. Meehan. *Panoramic Photography*. Watson-Guptill, 1990.

[15] H.-Y. Shum and R. Szeliski. Construction and refinement of panoramic mosaics with global and local alignment. Submitted for review, April 1997.

[16] G. Stein. Accurate internal camera calibration using rotation, with analysis of sources of error. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 230–236, Cambridge, Massachusetts, June 1995.

[17] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE Workshop on Applications of Computer Vision (WACV'94)*, pages 44–53, Sarasota, Florida, December 1994. IEEE Computer Society.

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

[19] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[20] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52069, January 1984.

[21] L. Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, July 1983.

[22] http://qtvr.quicktime.apple.com.

[23] http://www.bdiamon.com.

[24] http://www.omniview.com.

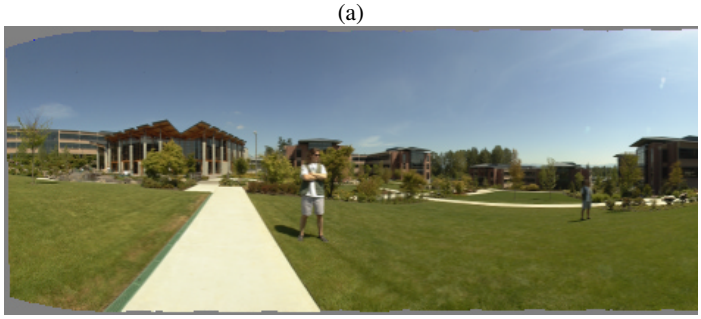
[25] http://www.smoothmove.com.

[26] http://www.rlspace.com.

[27] http://www.behere.com.

[28] http://www.cs.columbia.edu/cave/omnicam.

(a)



(b)

Figure 1: Construction of a cylindrical panorama: (a) two warped images; (b) part of cylindrical panorama composited from a sequence of images.

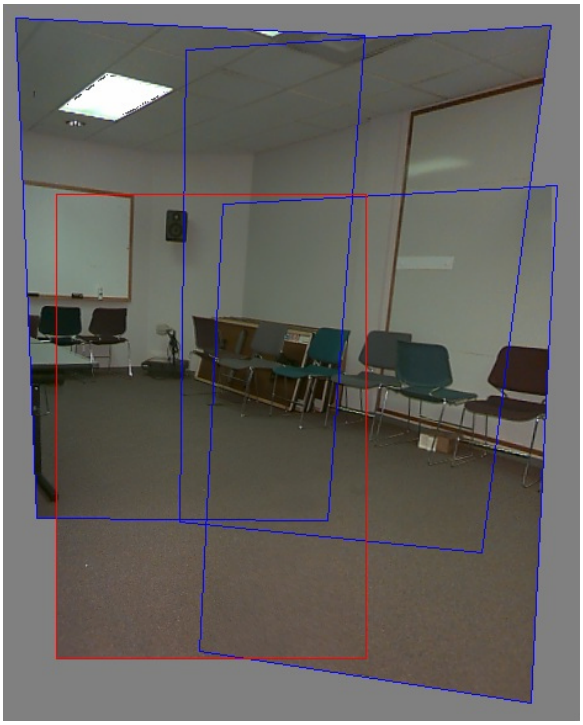
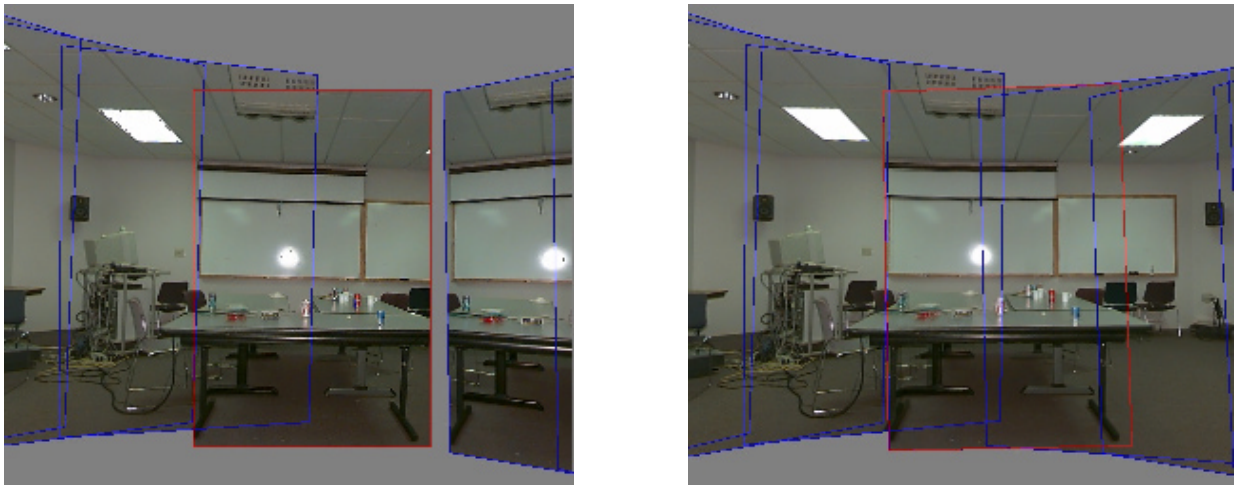


Figure 2: 3D rotation registration of four images taken with a hand-held camera.



(a) (b)

Figure 3: Gap closing after sequentially registering 24 images: (a) a gap is visible when the focal length is wrong ($f = 510$); (b) no gap is visible for the correct focal length ($f = 468$).

Figure 4: Tessellated spherical panorama covering the north pole (constructed from 54 images). The white triangles at the top are the parts of the texture map not covered in the 3D tesselated globe model (due to triangular elements at the poles).

Figure 5: Cubical texture-mapped model of conference room (from 75 images taken with a hand-held digital camera).

Figure 6: Cubical texture-mapped model of lobby (from 54 images).

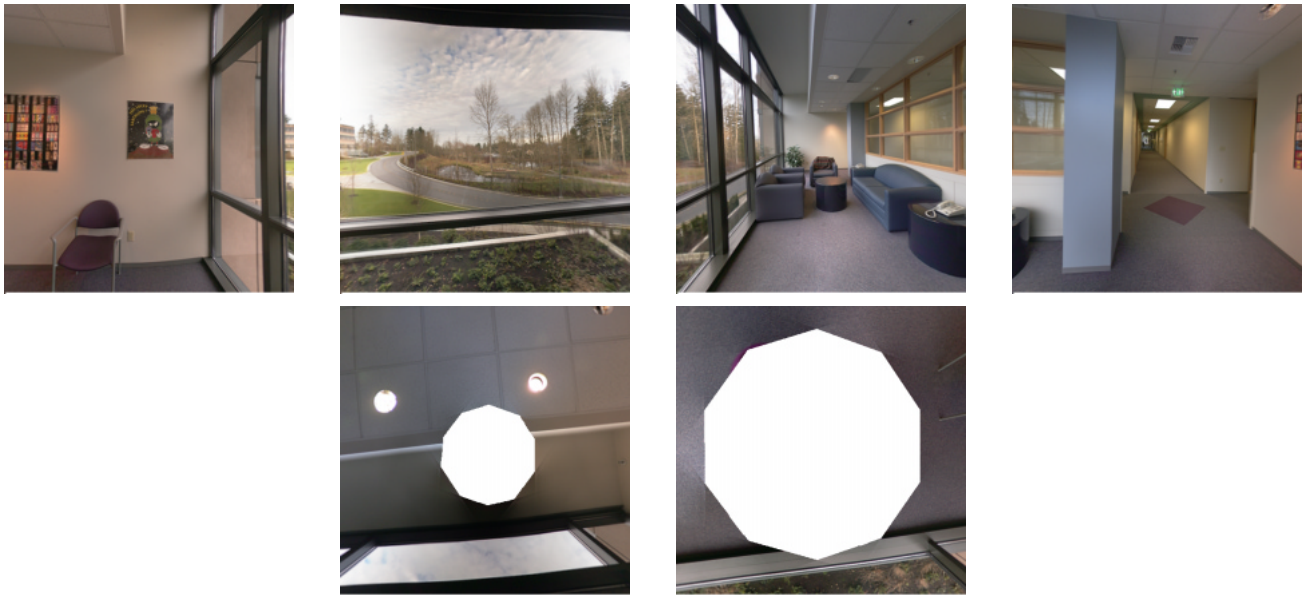


Figure 7: Cubical texture-mapped model of hallway and sitting area (from 36 images).

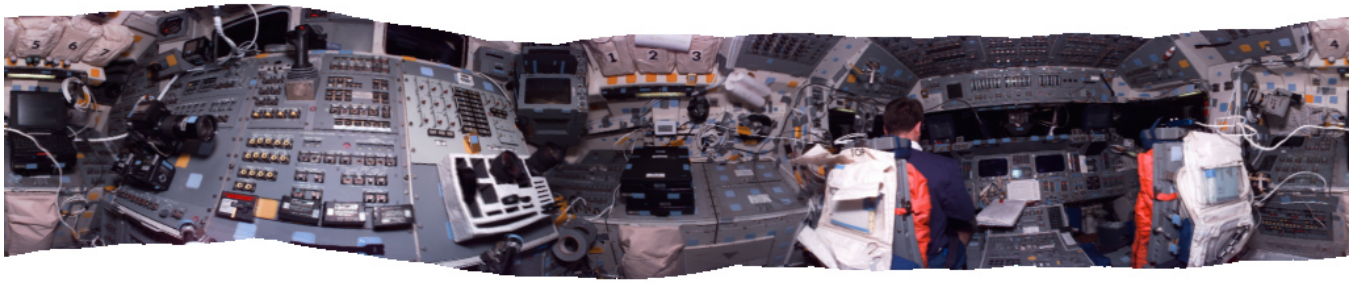


Figure 8: Panorama of Space Shuttle flight deck from 14 images taken with a hand-held camera (using *deghosting* technique).