13

Construction of Panoramic Image Mosaics with Global and Local Alignment

H.-Y. Shum and R. Szeliski

13.1 Introduction

The automatic construction of large, high-resolution image mosaics is an active area of research in the fields of photogrammetry, computer vision, image processing, and computer graphics. Image mosaics can be used for many different applications [163, 122]. The most traditional application is the construction of large aerial and satellite photographs from collections of images [186]. More recent applications include scene stabilization and change detection [93], video compression [125, 122, 167] and video indexing [240], increasing the field of view [105, 177, 266] and resolution [126, 50] of a camera, and even simple photo editing [38]. A particularly popular application is the emulation of traditional film-based panoramic photography [175] with digital panoramic mosaics, for applications such as the construction of virtual environments [181, 267] and virtual travel [49].

In computer vision, image mosaics are part of a larger recent trend, namely the study of *visual scene representations* [5]. The complete description of visual scenes and scene models often entails the recovery of depth or parallax information as well [161, 239, 271]. In computer graphics, image mosaics play an important role in the field of *image-based rendering*, which aims to rapidly render photorealistic novel views from collections of real (or pre-rendered) images [48, 181, 49, 84, 168, 144].

A number of techniques have been developed for capturing panoramic images of real-world scenes (for references on computer-generated environment maps, see [86]). One way is to record an image onto a long film strip using a panoramic camera to directly capture a cylindrical panoramic image [182]. Another way is to use a lens with a very large field of view such as a fisheye lens [298]. Mirrored pyramids and parabolic mirrors can also be used to directly capture panoramic images [195].

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 [177, 266, 122, 49, 181, 267].

For applications such as virtual travel and architectural walkthroughs, it is desirable to have complete (full view) panoramas, i.e., mosaics that cover the whole viewing sphere and hence allow the user to look in any direction. Unfortunately, most of the results to date have been limited to cylindrical panoramas obtained with cameras rotating on leveled tripods adjusted to minimize motion parallax [181, 49, 263, 267, 148]. This has limited the users of mosaic building to researchers and professional photographers who can afford such specialized equipment.

The goal of our work is to remove the need for pure panning motion with no motion parallax. Ideally, we would like any user to be able to "paint" a full view panoramic mosaic with a simple hand-held camera or camcorder. In order to support this vision, several problems must be overcome.

First, we need to avoid using cylindrical or spherical coordinates for constructing the mosaic, since these representations introduce singularities near the poles of the viewing sphere. We solve this problem by associating a rotation matrix (and optionally focal length) with each input image, and performing registration in the input image's coordinate system (we call such mosaics *rotational mosaics* [273]). A postprocessing stage can be used to project such mosaics onto a convenient viewing surface, i.e., to create an *environment map* represented as a texture-mapped polyhedron surrounding the origin.

Second, we need to deal with accumulated misregistration errors, which are always present in any large image mosaic. For example, if we register a sequence of images using pairwise alignments, there is usually a gap between the last image and the first one even if these two images are the same. A simple "gap closing" technique can be used to force the first and last image to be the same, to refine the focal length estimation, and to distribute the resulting corrections across the image sequence [273, 148]. Unfortunately, this approach works only for pure panning motions with uniform motion steps. In this paper, we present a global optimization technique, derived from *simultaneous bundle block adjustment* in photogrammetry [295], to find the optimal overall registration.

Third, any deviations from the pure parallax-free motion model or ideal pinhole (projective) camera model may result in local misregistrations, which are visible as a loss of detail or multiple images (*ghosting*). To overcome this problem, we compute local motion estimates (block-based optical flow) between pairs of overlapping images, and use these estimates to warp each input image so as to reduce the misregistration [258]. Note that this is less ambitious than actually recovering a projective depth value for each pixel [161, 239, 271], but has the advantage of being able to simultaneously

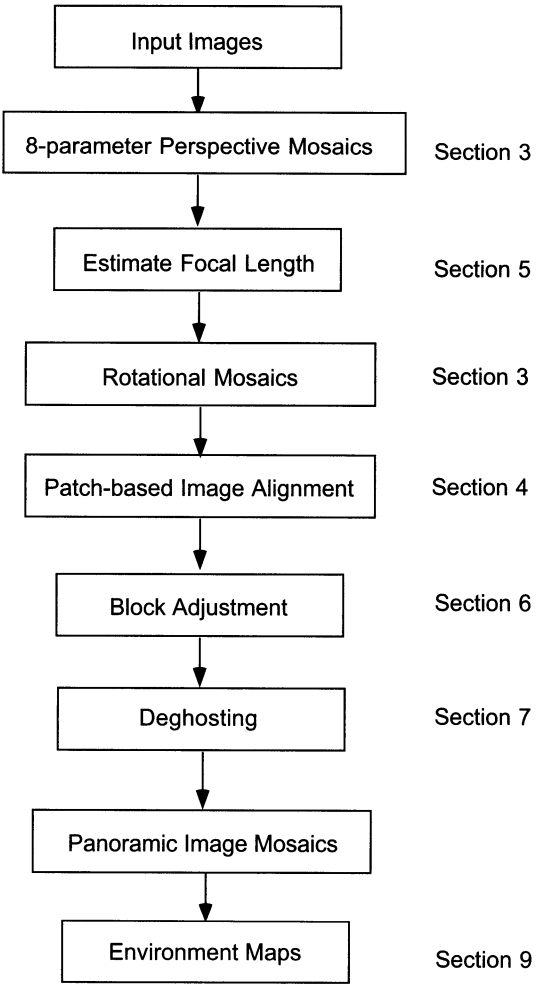


FIGURE 13.1. Panoramic image mosaicing system

model other effects such as radial lens distortions and small movements in the image.

The overall flow of processing in our mosaicing system is illustrated in Figure 13.1. First, if the camera intrinsic parameters are unknown, the user creates a small mosaic using a planar projective motion model, from which we can compute a rough estimate of the focal length (Section 13.5). Next, a complete initial panoramic mosaic is assembled sequentially (adding one image at a time and adjusting its position) using our rotational motion model (Section 13.3) and patch-based alignment technique (Section 13.4). Then, global alignment (block adjustment) is invoked to modify each image's transformation (and focal length) such that the global error across all possible overlapping image pairs is minimized (Section 13.6). This stage

also removes any large inconsistencies in the mosaic, e.g., the "gaps" that might be present in a panoramic mosaic assembled using the sequential algorithm. Lastly, the local alignment (*deghosting*) algorithm is invoked to reduce any local misregistration errors (Section 13.7). The final mosaic can be stored as a collection of images with associated transformations, or optionally converted into a texture-mapped polyhedron or environment map (Section 13.9).

The structure of our paper essentially follows the major processing stages, as outlined above. In addition, we show in Section 13.2 how to construct cylindrical and spherical panoramas, which are special cases of panoramic image mosaics with a known camera focal length and a simple translational motion model. Section 13.8 presents our experimental results[1] using both global and local alignment, and Section 13.10 discusses these results and summarizes the components in our system.

13.2 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 13.2a shows two overlapping cylindrical images—notice how horizontal lines become curved.

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

$$\theta = \tan^{-1}(X/Z) \tag{13.1}$$

$$v = Y/\sqrt{X^2 + Z^2} \tag{13.2}$$

where θ is the panning angle and v is the scanline [267]. Similarly, we can map world coordinates into 2D spherical coordinates (θ, ϕ) using

$$\theta = \tan^{-1}(X/Z) \tag{13.3}$$

$$\phi = \tan^{-1}(Y/\sqrt{X^2 + Z^2}). \tag{13.4}$$

Once we have warped each input image, constructing the panoramic mosaics becomes a pure translation problem. Ideally, to build a cylindrical or

[1] Example image sequences and results are also online www.research.microsoft.com/users/hshum/ijcv99/ijcv.htm.

[2] To convert from image coordinates (x, y) to world coordinates (directions), we use $(X, Y, Z) = (x + c_x, y + c_y, f)$, where (c_x, c_y) are the coordinates of the camera's optical center, and f is the focal length measured in pixels (see Section 13.3.2).

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 $\delta\mathbf{t} = (\delta t_x, \delta t_y)$ by minimizing the intensity error between two images[3],

$$E(\delta\mathbf{t}) = \sum_i [I_1(\mathbf{x}'_i + \delta\mathbf{t}) - I_0(\mathbf{x}_i)]^2, \tag{13.5}$$

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 [25].

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

$$E(\delta\mathbf{t}) \approx \sum_i [\mathbf{g}_i^T \delta\mathbf{t} + e_i]^2 \tag{13.6}$$

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,

$$\left(\sum_i \mathbf{g}_i \mathbf{g}_i^T\right) \delta\mathbf{t} = -\left(\sum_i e_i \mathbf{g}_i\right). \tag{13.7}$$

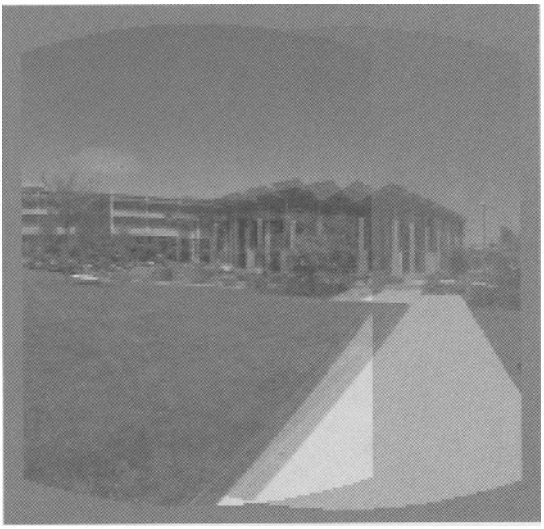
Figure 13.2b shows a portion of a cylindrical panoramic mosaic built using this simple translational alignment technique. To handle larger initial displacements, we use a hierarchical coarse-to-fine optimization scheme [25].

When blending the images into a composite mosaic, in order 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 [267]. More precisely, for each warped image being blended, we first compute the distance map, $d(\mathbf{x})$, which measures either the city block distance [228] or the Euclidean distance [54] to the nearest transparent pixel ($\alpha = 0$) or border pixel. We then blend all of the warped images using

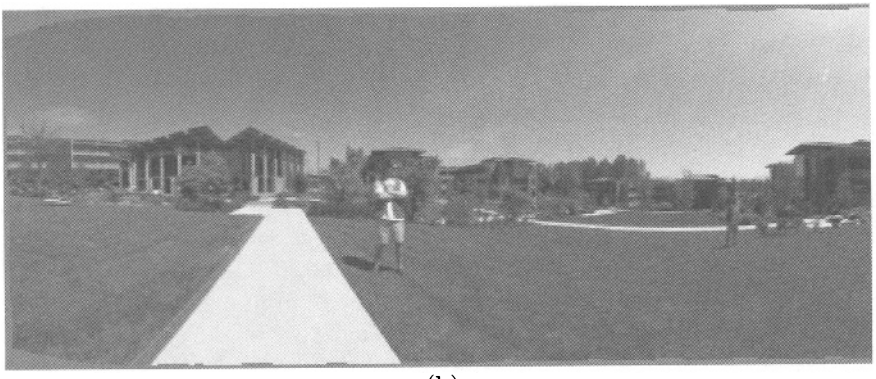
$$C(\mathbf{x}) = \frac{\sum_k w(d(\mathbf{x}))\tilde{I}_k(\mathbf{x})}{\sum_k w(d(\mathbf{x}))} \tag{13.8}$$

where w is a monotonic function (we currently use $w(x) = x$). An alternative to such weighted blending is to select pixels from only one image, but this can be tricky in practice [185, 214, 297, 56].

[3]For robust versions of this metric, see [27, 240].



(a)



(b)

FIGURE 13.2. Construction of a cylindrical panorama: (a) two warped images; (b) part of cylindrical panorama composited from a sequence of images.

Once the alignment and blending steps are finished, we can clip the ends (and optionally the top and bottom) and write out a single panoramic image. An example of a cylindrical panorama is shown in Figure 13.2b. The cylindrical/spherical image can then be displayed with a special purpose viewer like QTVR or Surround Video. Alternatively, it can be wrapped onto a cylinder or sphere using texture-mapping. For example, the Direct3D graphics API has a `CreateWrap` primitive which can be used to wrap a spherical or cylindrical image around an object using texture-mapping. However, the object needs to be finely tessellated in order to avoid visible artifacts.

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[4]. Third, it requires knowing the focal length (or equivalently, field of view). While focal length can be carefully calibrated in the lab [284, 263], estimating the focal length of lens by registering two or more images is not very accurate, as we will discuss in Section 13.5.

13.3 Alignment Framework and Motion Models

In our system, we represent image mosaics as collections of images with associated geometrical transformations. The first stage of our mosaic construction algorithm computes an initial estimate for the transformation associated with each input image. We do this by processing each input image in turn, and finding the best alignment between this image and the mosaic constructed from all previous images.[5] This reduces the problem to that of parametric motion estimation [25]. We use the hierarchical motion estimation framework proposed by Bergen *et al.*, which consists of four parts: (i) pyramid construction, (ii) motion estimation, (iii) image warping, and (iv) coarse-to-fine refinement [25].

An important element of this framework, which we exploit, is to perform the motion estimation between the current new input image and a *warped* (resampled) version of the mosaic. This allows us to estimate only *incremental* deformations of images (or equivalently, *instantaneous* motion), which greatly simplifies the computation of the gradients and Hessians required in our gradient descent algorithm (e.g., compare the Hessians computed below with those presented in [267]). Thus, to register two images $I_0(\mathbf{x})$ and $I_1(\mathbf{x}')$, where $\mathbf{x}'$ is computed using some parametric motion model $\mathbf{m}$, i.e., $\mathbf{x}' = \mathbf{f}(\mathbf{x}; \mathbf{m})$, we first compute the warped image

$$\tilde{I}_1(\mathbf{x}) = I_1(\mathbf{f}(\mathbf{x}; \mathbf{m})) \tag{13.9}$$

(in our current implementation, we use bilinear pixel resampling). The task is then to find a deformation of $\tilde{I}_1(\mathbf{x})$ which brings it into closer registration with $I_0(\mathbf{x})$ and which can also be used to update the parameter $\mathbf{m}$. The warp/register/update loop can then be repeated. In the next three

[4]Note that cylindrical coordinates become undefined as you tilt your camera toward north or south pole.

[5]To speed up this part, we can optionally register with only the previous image in the sequence.

subsections, we describe how this can be done for two different transformation models, namely 8-parameter planar projective transformations and 3D rotations, and how this can be generalized to other motion models and parameters.

13.3.1 8-parameter Perspective Transformations

Given two images taken from the same viewpoint (optical center) but in potentially different directions (and/or with different intrinsic parameters), the relationship between two overlapping images can be described by a homography or planar perspective motion model [177, 266, 122, 267] (for a proof, see Section 13.3.2 below). The planar perspective transformation warps an image into another using

$$\mathbf{x}' \sim \mathbf{M}\mathbf{x} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{13.10}$$

where $\mathbf{x} = (x, y, 1)$ and $\mathbf{x}' = (x', y', 1)$ are homogeneous or projective coordinates, and $\sim$ indicates equality up to scale.[6] This equation can be re-written as

$$x' = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + m_8} \tag{13.11}$$

$$y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + m_8}. \tag{13.12}$$

To recover the parameters, we iteratively update the transformation matrix[7] using

$$\mathbf{M} \leftarrow \mathbf{M}(\mathbf{I} + \mathbf{D}) \tag{13.13}$$

where

$$\mathbf{D} = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_3 & d_4 & d_5 \\ d_6 & d_7 & d_8 \end{bmatrix}. \tag{13.14}$$

Resampling image I_1 with the new transformation $\mathbf{x}' \sim \mathbf{M}(\mathbf{I} + \mathbf{D})\mathbf{x}$ is the same as warping the resampled image $\tilde{I}_1$ by $\mathbf{x}'' \sim (\mathbf{I} + \mathbf{D})\mathbf{x}$,[8] i.e.,

$$x'' = \frac{(1 + d_0)x + d_1 y + d_2}{d_6 x + d_7 y + (1 + d_8)} \tag{13.15}$$

$$y'' = \frac{d_3 x + (1 + d_4)y + d_5}{d_6 x + d_7 y + (1 + d_8)}. \tag{13.16}$$

---

[6] Since the $\mathbf{M}$ matrix is invariant to scaling, there are only 8 independent parameters.

[7] To improve conditioning of the linear system and to speed up the convergence, we place the origin $(x, y) = (0, 0)$ at the center of the image.

[8] Ignoring errors introduced by the double resampling operation.

We wish to minimize the squared error metric

$$E(\mathbf{d}) = \sum_i [\tilde{I}_1(\mathbf{x}_i'') - I_0(\mathbf{x}_i)]^2 \tag{13.17}$$

$$\approx \sum_i [\tilde{I}_1(\mathbf{x}_i) + \nabla \tilde{I}_1(\mathbf{x}_i) \frac{\partial \mathbf{x}_i''}{\partial \mathbf{d}} \mathbf{d} - I_0(\mathbf{x}_i)]^2 = \sum_i [\mathbf{g}_i^T \mathbf{J}_i^T \mathbf{d} + e_i]^2 \tag{13.18}$$

where $e_i = \tilde{I}_1(\mathbf{x}_i) - I_0(\mathbf{x}_i)$ is the intensity or color error[9], $\mathbf{g}_i^T = \nabla \tilde{I}_1(\mathbf{x}_i)$ is the image gradient of $\tilde{I}_1$ at $\mathbf{x}_i$, $\mathbf{d} = (d_0, \ldots, d_8)$ is the incremental motion parameter vector, and $\mathbf{J}_i = \mathbf{J}_\mathbf{d}(\mathbf{x}_i)$, where

$$\mathbf{J}_\mathbf{d}(\mathbf{x}) = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy & -x \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 & -y \end{bmatrix}^T \tag{13.19}$$

is the Jacobian of the resampled point coordinate $\mathbf{x}_i''$ with respect to $\mathbf{d}$.[10]

This least-squares problem (13.18) has a simple solution through the *normal equations* [220]

$$\mathbf{A}\mathbf{d} = -\mathbf{b}, \tag{13.20}$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_i \mathbf{g}_i \mathbf{g}_i^T \mathbf{J}_i^T \tag{13.21}$$

is the *Hessian*, and

$$\mathbf{b} = \sum_i e_i \mathbf{J}_i \mathbf{g}_i \tag{13.22}$$

is the *accumulated gradient* or *residual*. These equations can be solved using a symmetric positive definite (SPD) solver such as *Cholesky* decomposition [220]. Note that for our problem, the matrix $\mathbf{A}$ is singular unless we eliminate one of the three parameters $\{d_0, d_4, d_8\}$. In practice, we set $d_8 = 0$, and therefore only solve an 8×8 system. A diagram of our alignment framework is shown in Figure 13.3.

Translational motion is a special case of the general 8-parameter perspective transformation where $\mathbf{J}$ is a 2×2 identity matrix because only the two parameters m_2 and m_5 are used. The translational motion model can be used to construct cylindrical and spherical panoramas if we warp each image to cylindrical or spherical coordinates image using a known focal length, as shown in Section 13.2.

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

[9] Currently three channels of color errors are used in our system, but we can use the luminance (intensity) error instead.

[10] The entries in the Jacobian correspond to the optical flow induced by the instantaneous motion of a plane in 3D [25].

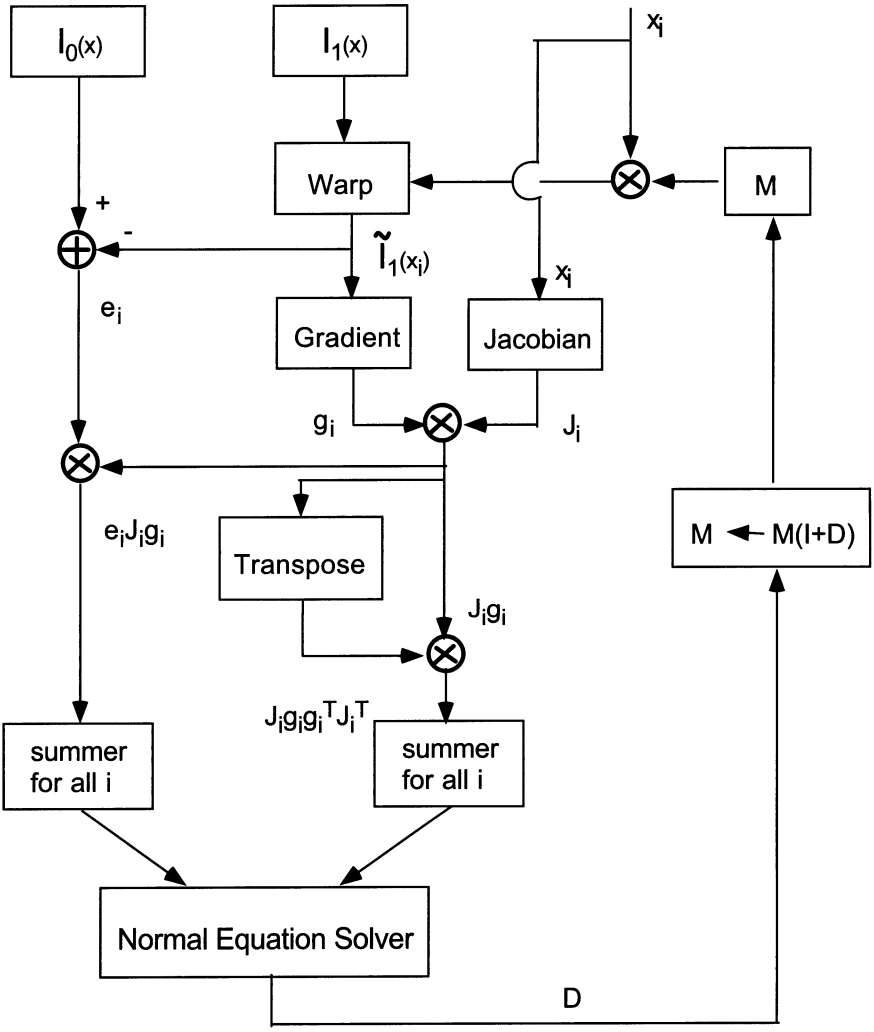


FIGURE 13.3. A diagram for our image alignment framework

than necessary, it suffers from slow convergence and sometimes gets stuck in local minima. For this reason, when we know that the camera is rotating around its optical axis, as opposed to undergoing general motion while observing a plane, we prefer to use the 3-parameter rotational model described next.

13.3.2 3D Rotations and Zooms

For a camera centered at the world origin, i.e., $(X, Y, Z) = (0, 0, 0)$, the relationship between a 3D point $\mathbf{p} = (X, Y, Z)$ and its image coordinates $\mathbf{x} = (x, y, 1)$ can be described by

$$\mathbf{x} \sim \mathbf{TVRp}, \tag{13.23}$$

where

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ and } \mathbf{R} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$$

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).[11] 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

$$\mathbf{M} \sim \mathbf{V}_k\mathbf{R}_k\mathbf{R}_l^{-1}\mathbf{V}_l^{-1} = \mathbf{V}_k\mathbf{R}_{kl}\mathbf{V}_l^{-1} \tag{13.24}$$

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, $\mathbf{V}_k = \mathbf{V}$. Our method for computing an estimate of f from an initial set of homographies is given in Section 13.5. To recover the rotation, we perform an incremental update to $\mathbf{R}_k$ based on the angular velocity $\mathbf{\Omega} = (\omega_\mathbf{x}, \omega_\mathbf{y}, \omega_\mathbf{z})$,

$$\mathbf{R}_{kl} \leftarrow \mathbf{R}_{kl}\hat{\mathbf{R}}(\mathbf{\Omega}) \qquad \text{or} \qquad \mathbf{M} \leftarrow \mathbf{V}\mathbf{R}_\mathbf{kl}\hat{\mathbf{R}}(\mathbf{\Omega})\mathbf{V}^{-1} \tag{13.25}$$

where the incremental rotation matrix $\hat{\mathbf{R}}(\mathbf{\Omega})$ is given by Rodriguez's formula [8],

$$\hat{\mathbf{R}}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + (\sin\theta)\mathbf{X}(\hat{\mathbf{n}}) + (1 - \cos\theta)\mathbf{X}(\hat{\mathbf{n}})^2 \tag{13.26}$$

[11] The above equation also assumes a unit aspect ratio, no skew, and no radial distortion.

(a) (b)

FIGURE 13.4. 3D rotation registration of four images taken with hand-held camera: (a) four original pictures; (b) image mosaic using 3D rotation.

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

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

is the cross product operator.[12] Keeping only terms linear in $\boldsymbol{\Omega}$, we get

$$\mathbf{M}' \approx \mathbf{V}\mathbf{R}_k\mathbf{R}_l^{-1}[\mathbf{I} + \mathbf{X}(\boldsymbol{\Omega})]\mathbf{V}^{-1} = \mathbf{M}(\mathbf{I} + \mathbf{D}_{\boldsymbol{\Omega}}), \tag{13.27}$$

where

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

is the deformation matrix which plays the same role as $\mathbf{D}$ in (13.13).

Computing the Jacobian of the entries in $\mathbf{D}_\Omega$ with respect to $\boldsymbol{\Omega}$ and applying the chain rule, we obtain the new Jacobian,[13]

$$\mathbf{J}_\Omega = \frac{\partial \mathbf{x}''}{\partial \boldsymbol{\Omega}} = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}} \frac{\partial \mathbf{d}}{\partial \boldsymbol{\Omega}} = \begin{bmatrix} -xy/f & f + x^2/f & -y \\ -f - y^2/f & xy/f & x \end{bmatrix}^T. \tag{13.28}$$

[12] This is also called the *twist* or *exponential map* representation in robotics [193].

[13] This is the same as the rotational component of instantaneous rigid flow [25].

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 (13.25).

Figure 13.4 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.[14]

13.3.3 Other Motion Models

The same general strategy can be followed to obtain the gradient and Hessian associated with any other motion parameters. For example, the focal length f_k can be adjusted by setting $f_k \leftarrow (1 + e_k)f_k$, i.e.,

$$\mathbf{M} \leftarrow \mathbf{M}(\mathbf{I} + e_k \mathbf{D}_{110}) \tag{13.29}$$

where $\mathbf{D}_{110}$ is a diagonal matrix with entries $(1, 1, 0)$. The Jacobian matrix $\mathbf{J}_{e_k}$ is thus the diagonal matrix with entries (x, y), i.e., we are estimating a simple re-scaling (dilation). This formula can be used to re-estimate the focal length in a video sequence with a variable focal length (zoom).

If we wish to update a single global focal length estimate, $f \leftarrow (1 + e)f$, the update equation and Jacobian are more complicated. We obtain

$$\mathbf{M} \leftarrow (\mathbf{I} + e\mathbf{D}_{110})\mathbf{V}\mathbf{R}_k\mathbf{R}_l^{-1}\mathbf{V}^{-1}(\mathbf{I} - e\mathbf{D}_{110}) \approx \mathbf{M}(\mathbf{I} + e\mathbf{D}_e) \tag{13.30}$$

where

$$\mathbf{D}_e = \mathbf{D}_{110} - \mathbf{M}\mathbf{D}_{110}\mathbf{M}^{-1} \tag{13.31}$$

(further simplifications of the second term are possible because of the special structure of $\mathbf{D}_{110}$). The Jacobian does not have a nice simple structure, but can nevertheless be written as the product of $\mathbf{J}_\mathbf{d}$ and $\partial\mathbf{d}/\partial e$, which is given by the entries in $\mathbf{D}_e$. Note, however, that global focal length adjustment cannot be done as part of the initial sequential mosaic creation stage, since this algorithm presupposes that only the newest image is being adjusted. We will address the issue of global focal length estimate refinement in Section 13.6.

The same methodology as presented above can be used to update any motion parameter p on which the image-to-image homography $\mathbf{M}(p)$ depends, e.g., the aspect ratio.[15] We simply set

$$\mathbf{M} \leftarrow \mathbf{M}(p + \delta p) \approx \mathbf{M}(\mathbf{I} + \mathbf{M}^{-1}\frac{\partial \mathbf{M}}{\partial p}\delta p). \tag{13.32}$$

---

[14] With only a mouse click/drag on screen, it is difficult to control 8 parameters simultaneously.

[15] Updating parameters such as radial distortion which affect the image formation process in a non-linear way requires a different method, i.e., directly taking derivatives of image pixel locations w.r.t. these parameters.

Hence, we can read off the entries in $\partial \mathbf{d}/\partial p$ from the entries in $\mathbf{M}^{-1}(\partial \mathbf{M}/\partial p)$.

13.4 Patch-based Alignment Algorithm

The normal equations given in the previous section, together with an appropriately chosen Jacobian matrix, can be used to directly improve the current motion estimate by first computing local intensity errors and gradients, and then accumulating the entries in the parameter gradient vector and Hessian matrix. This straightforward algorithm suffers from several drawbacks: it is susceptible to local minima and outliers, and is also unnecessarily inefficient. In this section, we present the implementation details of our algorithm which makes it much more robust and efficient [257].

13.4.1 Patch-based Alignment

The computational effort required to take a single gradient descent step in parameter space can be divided into three major parts: (i) the warping (resampling) of $I_1(\mathbf{x}')$ into $\tilde{I}_1(\mathbf{x})$, (ii) the computation of the local intensity errors e_i and gradients $\mathbf{g}_i$, and (iii) the accumulation of the entries in $\mathbf{A}$ and $\mathbf{b}$ (13.21–13.22). This last step can be quite expensive, since it involves the computations of the monomials in $\mathbf{J}_i$ and the formation of the products in $\mathbf{A}$ and $\mathbf{b}$.

Notice that equations (13.21–13.22) can be written as vector/matrix products of the Jacobian $\mathbf{J}(\mathbf{x}_i)$ with the *gradient-weighted intensity errors*, $e_i\mathbf{g}_i$, and the *local intensity gradient Hessians* $\mathbf{g}_i\mathbf{g}_i^T$. If we divide the image up into little patches $\mathcal{P}_j$, and make the approximation that $\mathbf{J}(\mathbf{x}_i) = \mathbf{J}_j$ is constant within each patch (say by evaluating it at the patch center), we can write the normal equations as

$$\mathbf{A} \approx \sum_j \mathbf{J}_j \mathbf{A}_j \mathbf{J}_j^T \qquad \text{with} \qquad \mathbf{A}_j = \sum_{i \in \mathcal{P}_j} \mathbf{g}_i \mathbf{g}_i^T \tag{13.33}$$

and

$$\mathbf{b} \approx \sum_j \mathbf{J}_j \mathbf{b}_j \qquad \text{with} \qquad \mathbf{b}_j = \sum_{i \in \mathcal{P}_j} e_i \mathbf{g}_i. \tag{13.34}$$

$\mathbf{A}_j$ and $\mathbf{b}_j$ are the terms that appear in patch-based optical flow algorithms [172, 25]. Our algorithm therefore augments step (ii) above with the accumulation of $\mathbf{A}_j$ and $\mathbf{b}_j$ (only 10 additional multiply/add operations, which could potentially be done using fixpoint arithmetic), and performs the computations required to evaluate $\mathbf{J}_j$ and accumulate $\mathbf{A}$ and $\mathbf{b}$ only once per patch.

A potential disadvantage of using this approximation is that it might lead to poorer convergence (more iterations) in the parameter estimation

algorithm. In practice, we have not observed this to be the case with the small patches (8×8) that we currently use.

13.4.2 Correlation-style Search

Another limitation of straightforward gradient descent is that it can get trapped in local minima, especially when the initial misregistration is more than a few pixels. A useful heuristic for enlarging the region of convergence is to use a hierarchical or coarse-to-fine algorithm, where estimates from coarser levels of the pyramid are used to initialize the registration at finer levels [221, 4, 25]. This is a remarkably effective technique, and we typically always use 3 or 4 pyramid levels in our mosaic construction algorithm. However, it may still sometimes fail if the amount of misregistration exceeds the scale at which significant image details exist (i.e., because these details may not exist or may be strongly aliased at coarse resolution levels).

To help overcome this problem, we have added a *correlation-style search* component to our registration algorithm.[16] Before doing the first gradient descent step at a given resolution level, the algorithm can be instructed to perform an independent search at each patch for the integral shift which will best align the I_0 and $\tilde{I}_1$ images (this *block-matching* technique is the basis of most MPEG coding algorithms [166]). For a search range of $\pm s$ pixels both horizontally and vertically, this requires the evaluation of $(2s+1)^2$ different shifts. For this reason, we usually only apply the correlation-style search algorithm at the coarsest level of the pyramid (unlike, say, [4], which is a dense optic flow algorithm).

Once the displacements have been estimated for each patch, they must somehow be integrated into the global parameter estimation algorithm. The easiest way to do this is to compute a new set of patch Hessians $\mathbf{A}_j$ and patch residuals $\mathbf{b}_j$ (c.f. (13.33–13.34)) to encode the results of the search. Recall that for patch-based flow algorithms [172, 25], $\mathbf{A}_j$ and $\mathbf{b}_j$ describe a local error surface

$$E(\mathbf{u}_j) = \mathbf{u}_j^T \mathbf{A}_j \mathbf{u}_j + 2\mathbf{u}_j^T \mathbf{b}_j + c \; = (\mathbf{u}_j - \mathbf{u}_j^*)^T \mathbf{A}_j (\mathbf{u}_j - \mathbf{u}_j^*) + c' \quad (13.35)$$

where

$$\mathbf{u}_j^* = -\mathbf{A}_j^{-1} \mathbf{b}_j \quad (13.36)$$

is the minimum energy (optimal) flow estimate.

We have applied two techniques for computing $\mathbf{A}_j$ and $\mathbf{b}_j$ from the results of the correlation-style search. The first is to fit (13.35) to the discretely sampled error surface which was used to determine the best shift $\mathbf{u}_0$. Since there are 5 free parameters in $\mathbf{A}_j$ and $\mathbf{b}_j$ ($\mathbf{A}_j$ is symmetric), we can simply

[16] To compensate for even larger misregistration, *phase correlation* could be used to estimate a translation for the whole image [267].

fit a bivariate quadratic surface to the central E value and its 4 nearest neighbors (more points can be used, if desired). Note that this fit will implicitly localize the results of the correlation-style search to sub-pixel precision (because of the quadratic fit).

A second approach is to compute $\mathbf{A}_j$ and $\mathbf{b}_j$ using the gradient-based approach (13.33–13.34), but with image $\tilde{I}(\mathbf{x})$ shifted by the estimated amount $\mathbf{u}_0$. After accumulating the new Hessian $\hat{\mathbf{A}}_j$ and residual $\hat{\mathbf{b}}_j$ with respect to the shifted image, we can compute the new gradient-based sub-pixel estimate

$$\hat{\mathbf{u}}_j^* = -\hat{\mathbf{A}}_j^{-1}\hat{\mathbf{b}}_j. \tag{13.37}$$

Adding $\hat{\mathbf{u}}_j^*$ to the correlation-style search displacement $\mathbf{u}_0$, i.e.,

$$\mathbf{u}_j^* = \hat{\mathbf{u}}_j^* + \mathbf{u}_0 \tag{13.38}$$

is equivalent to setting

$$\mathbf{A}_j = \hat{\mathbf{A}}_j, \qquad \mathbf{b}_j = \hat{\mathbf{b}}_j - \mathbf{A}_j\mathbf{u}_0. \tag{13.39}$$

We prefer this second approach, since it results in $\mathbf{A}_j$ estimates which are non-negative definite (important for ensuring that the normal equations can be solved stably), and since it better reflects the certainty in a local match.[17]

13.5 Estimating the Focal Length

In order to apply our 3D rotation technique, we must first obtain an estimate for the camera's focal length. We can obtain such an estimate 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

$$\mathbf{M} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \sim \begin{bmatrix} r_{00} & r_{01} & r_{02}f_0 \\ r_{10} & r_{11} & r_{12}f_0 \\ r_{20}/f_1 & r_{21}/f_1 & r_{22}f_0/f_1 \end{bmatrix} \tag{13.40}$$

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

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

$$m_0^2 + m_1^2 + m_2^2/f_0^2 = m_3^2 + m_4^2 + m_5^2/f_0^2 \tag{13.41}$$

$$m_0m_3 + m_1m_4 + m_2m_5/f_0^2 = 0 \tag{13.42}$$

[17] An analysis of the relationship between these two approaches can be found in [277].

and

$$m_0{}^2 + m_3{}^2 + m_6{}^2 f_1{}^2 = m_1{}^2 + m_4{}^2 + m_7{}^2 f_1{}^2 \tag{13.43}$$

$$m_0 m_1 + m_3 m_4 + m_6 m_7 f_1{}^2 = 0. \tag{13.44}$$

From this, we can compute the estimates

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

or

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

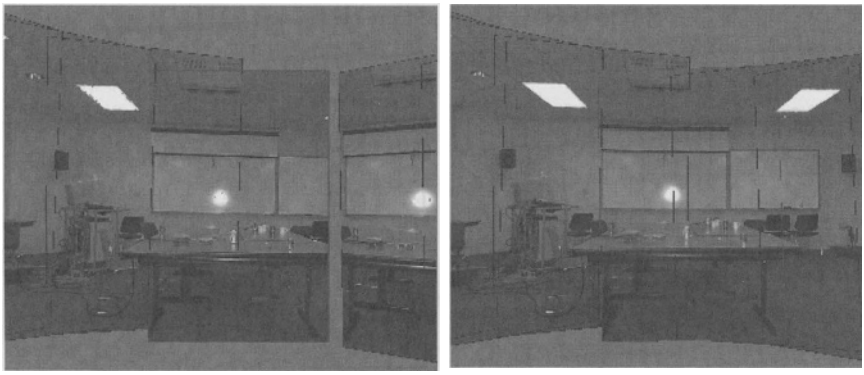
Similar result can be obtained for f_1 as well. If the focal length is fixed for two images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, the median value is used as the final estimate.

13.5.1 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 the 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 $f' = f(1 - \theta_g/360°)$.

Figure 13.5a 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 13.5b 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 13.5a has been computed using a focal length which has 9% error. Related approaches have been developed by [98, 181, 263, 148] to solve the focal length estimation problem using pure panning motion and cylindrical images. In next section, we present a different approach to removing gaps and overlaps which works for arbitrary image sequences.



(a) (b)

FIGURE 13.5. Gap closing: (a) a gap is visible when the focal length is wrong ($f = 510$); (b) no gap is visible for the correct focal length ($f = 468$).

13.6 Global Alignment (Block Adjustment)

The sequential mosaic construction techniques described in Sections 13.3 and 13.4 do a good job of aligning each new image with the previously composited mosaic. Unfortunately, for long image sequences, this approach suffers from the problem of accumulated misregistration errors. This problem is particularly severe for panoramic mosaics, where a visible gap (or overlap) will exist between the first and last images in a sequence, even if these two images are the same, as we have seen in the previous section.

In this section, we present our global alignment method, which reduces accumulated error by simultaneously minimizing the misregistration between all overlapping pairs of images. Our method is similar to the *"simultaneous bundle block adjustment"* [295] technique used in photogrammetry but has the following distinct characteristics:

- Corresponding points between pairs of images are automatically obtained using patch-based alignment.
- Our objective function minimizes the difference between ray directions going through corresponding points, and uses a rotational panoramic representation.
- The minimization is formulated as a constrained least-squares problem with hard linear constraints for identical focal lengths and repeated frames.[18]

[18]We have found that it is easier to use certain frames in the sequence more than once during the sequential mosaic formation process (say at the beginning and at the end), and to then use the global alignment stage to make sure that these all have the same associated location.

13.6.1 Establishing the Point Correspondences

Our global alignment algorithm is *feature-based*, i.e., it relies on first establishing point correspondences between overlapping images, rather than doing direct intensity difference minimization (as in the sequential algorithm).

To find our features, we divide each image into a number of patches (e.g., 16×16 pixels), and use the patch centers as prospective feature points. For each patch center, its corresponding point in another image could be determined directly by the current inter-frame transformation $\mathbf{M}_k\mathbf{M}_l^{-1}$. However, since we do not believe that these alignments are optimal, we instead invoke the correlation-style search-based patch alignment algorithm described in Section 13.4.2. (The results of this patch-based alignment are also used for the deghosting technique discussed in the next section.)

Pairs of images are examined only if they have significant overlap, for example, more than a quarter of the image size (see [242] for a general discussion of topology inference). In addition, instead of using all patch centers, we select only those with high confidence (or low uncertainty) measure. Currently we set a threshold for the minimum eigenvalue of each 2×2 patch Hessian (available from patch-based alignment algorithm) so that patches with uniform texture are excluded [253]. Other measures such as the ratio between two eigenvalues can also be used so that patches where the aperture problem exists can be ignored. Raw intensity error, however, would not make a useful measure for selecting feature patches because of potentially large inter-frame intensity variations (varying exposures, vignetting, etc.).

13.6.2 Optimality Criteria

For a patch j in image k, let $l \in \mathcal{N}_{jk}$ be the set of overlapping images in which patch j is totally contained (under the current set of transformations). Let $\mathbf{x}_{jk}$ be the center of this patch. To compute the patch alignment, we use image k as I_0 and image l as I_1 and invoke the algorithm of Section 13.4.2, which returns an estimated displacement $\mathbf{u}_{jl} = \mathbf{u}_j^*$. The corresponding point in the warped image $\tilde{I}_1$ is thus $\tilde{\mathbf{x}}_{jl} = \mathbf{x}_{jk} + \mathbf{u}_{jl}$. In image l, this point's coordinate is $\mathbf{x}_{jl} \sim \mathbf{M}_l\mathbf{M}_k^{-1}\tilde{\mathbf{x}}_{jl}$, or $\mathbf{x}_{jl} \sim \mathbf{V}_l\mathbf{R}_l\mathbf{R}_k^{-1}\mathbf{V}_k^{-1}\tilde{\mathbf{x}}_{jl}$ if the rotational panoramic representation is used.

Given these point correspondences, one way to formulate the global alignment is to minimize the difference between screen coordinates of all overlapping pairs of images,

$$E(\{\mathbf{M}_k\}) = \sum_{j,k,l \in \mathcal{N}_{jk}} \|\mathbf{x}_{jk} - \mathcal{P}(\mathbf{M}_k\mathbf{M}_l^{-1}\mathbf{x}_{jl})\|^2 \qquad (13.45)$$

where $\mathcal{P}(\mathbf{M}_k\mathbf{M}_l^{-1}\mathbf{x}_{jl})$ is the projected screen coordinate of $\mathbf{x}_{jl}$ under the transformation $\mathbf{M}_k\mathbf{M}_l^{-1}$ ($\mathbf{M}_k$ could be a general homography, or could be

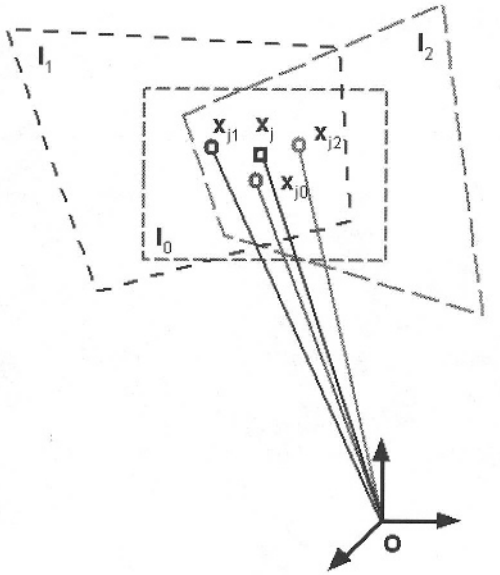


FIGURE 13.6. Illustration of simultaneous bundle block adjustment: we adjust the bundle of rays going through corresponding point features $\mathbf{x}_{jk}$ on overlapping images so that they converge to the ray going through $\mathbf{x}_j$.

based on the rotational panoramic representation). This has the advantage of being able to incorporate local certainties in the point matches (by making the above norm be a matrix norm based on the local Hessian $\mathbf{A}_{jk}$). The disadvantage, however, is that the gradients with respect to the motion parameters are complicated (Section 13.3). We shall return to this problem in Section 13.6.4.

A simpler formulation can be obtained by minimizing the difference between the ray directions of corresponding points using a rotational panoramic representation with unknown focal length. Geometrically, this is equivalent to adjusting the rotation and focal length for each frame so that the bundle of corresponding rays converge, as shown in Figure 13.6.

Let the ray direction in the final composited image mosaic be a unit vector $\mathbf{p}_j$, and its corresponding ray direction in the kth frame as $\mathbf{p}_{jk} \sim \mathbf{R}_k^{-1}\mathbf{V}_k^{-1}\mathbf{x}_{jk}$. We can formulate block adjustment to simultaneously optimize over both the pose (rotation and focal length $\{\mathbf{R}_k, f_k\}$) and structure (ray direction $\{\mathbf{p}_j\}$) parameters,

$$E(\{\mathbf{R}_k, f_k\}, \{\mathbf{p}_j\}) = \sum_{j,k} \|\mathbf{p}_{jk} - \mathbf{p}_j\|^2 = \sum_{j,k} \|\mathbf{R}_k^{-1}\hat{\mathbf{x}}_{jk} - \mathbf{p}_j\|^2 \qquad (13.46)$$

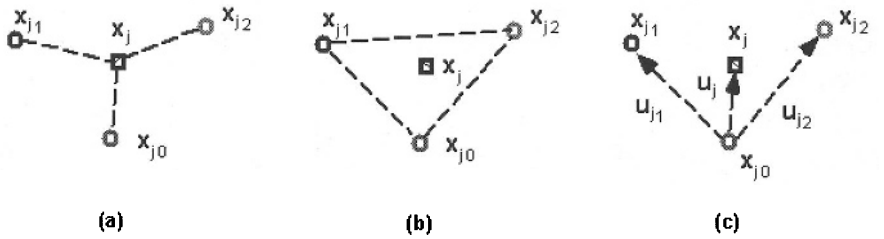


FIGURE 13.7. Comparison between two global registration methods: (a) minimizing the difference between all rays $\mathbf{p}_{jk}$ (with measured feature locations $\mathbf{x}_{jk}$) and $\mathbf{p}_j$ (at the predicted location $\mathbf{x}_j$); (b) minimizing the difference between all pairs of rays $\mathbf{p}_{jk}$ and $\mathbf{p}_{jl}$ (with feature locations $\mathbf{x}_{jk}$ and $\mathbf{x}_{jl}$ in overlapping images). (c) The desired flow for deghosting $\bar{\mathbf{u}}_{jk}$ is a down-weighted average of all pairwise flows $\mathbf{u}_{jl}$.

where

$$\hat{\mathbf{x}}_{jk} = \begin{bmatrix} x_{jk} \\ y_{jk} \\ f_k \end{bmatrix} / l_{jk} \tag{13.47}$$

is the ray direction going through the jth feature point located at (x_{jk}, y_{jk}) in the kth frame, and

$$l_{jk} = \sqrt{{x_{jk}}^2 + {y_{jk}}^2 + {f_k}^2} \tag{13.48}$$

(note that this absorbs the f_k parameter in V_k into the coordinate definition).

The advantage of the above direct minimization (13.46) is that both pose and structure can be solved independently for each frame. For instance, we can solve $\mathbf{p}_j$ using linear least-squares, $\mathbf{R}_k$ using relative orientation, and f_k using nonlinear least-squares. The disadvantage of this method is its slow convergence due to the highly coupled nature of the equations and unknowns.[19]

For the purpose of global alignment, however, it is not necessary to explicitly recover the ray directions. We can reformulate block adjustment to only minimize over pose ($\{\mathbf{R}_k, f_k\}$) for all frames k, without computing the $\{\mathbf{p}_j\}$. More specifically, we estimate the pose by minimizing the difference in ray directions between all pairs (k and l) of overlapping images,

$$E(\{\mathbf{R}_k, f_k\}) = \sum_{j,k,l \in \mathcal{N}_{jk}} \|\mathbf{p}_{jk} - \mathbf{p}_{jl}\|^2 = \sum_{j,k,l \in \mathcal{N}_{jk}} \|\mathbf{R}_k^{-1}\hat{\mathbf{x}}_{jk} - \mathbf{R}_l^{-1}\hat{\mathbf{x}}_{jl}\|^2 \tag{13.49}$$

[19]Imagine a chain of spring-connected masses. If we pull one end sharply, and then set each mass to the average of its neighbors, it will take the process a long time to reach equilibrium. This situation is analogous.

Once the pose has been computed, we can compute the estimated directions $\mathbf{p}_j$ using the known correspondence from all overlapping frames $\mathcal{N}_{jk}$ where the feature point j is visible,

$$\mathbf{p}_j \sim \frac{1}{n_{jk}+1} \sum_{l \in \{\mathcal{N}_{jk} \cup k\}} \mathbf{R}_l^{-1} \mathbf{V}_l^{-1} \mathbf{x}_{jl}. \tag{13.50}$$

where $n_{jk} = |\mathcal{N}_{jk}|$ is the number of overlapping images where patch j is completely visible (this information will be used later in the deghosting stage).

Figure 13.7 shows the difference between the above two formulations. Figure 13.7a shows how the difference being minimized (dashed lines) is between the expected feature location $\mathbf{x}_j$ (or the expected ray $\mathbf{p}_j$ going through) and all feature locations $\mathbf{x}_{jk}$ (or the rays $\mathbf{p}_{jk}$ going through), while Figure 13.7b shows minimizing the difference between all pairs of features $\mathbf{x}_{jk}$ (or the rays $\mathbf{p}_{jk}$ going through) on the overlapping images.

13.6.3 Solution Technique

The least-squares problem (13.49) can be solved using regular gradient descent method. To recover the pose $\{\mathbf{R}_k, f_k\}$, we iteratively update the ray directions $\mathbf{p}_{jk}(\mathbf{x}_{jk}; \mathbf{R}_k, f_k)$ to

$$\mathbf{R}_k^{-1} \leftarrow \hat{\mathbf{R}}(\boldsymbol{\Omega}_\mathbf{k}) \mathbf{R}_\mathbf{k}^{-1} \quad \text{and} \quad \mathbf{f_k} \leftarrow \mathbf{f_k} + \delta \mathbf{f_k}. \tag{13.51}$$

The minimization problem (13.49) can be rewritten as

$$E(\{\mathbf{R}_k, f_k\}) = \sum_{j,k,l \in \mathcal{N}_{jk}} \|\mathbf{H}_{jk}\mathbf{y}_k - \mathbf{H}_{jl}\mathbf{y}_l + \mathbf{e}_j\|^2 \tag{13.52}$$

where

$$\begin{aligned} \mathbf{e}_j &= \mathbf{p}_{jk} - \mathbf{p}_{jl}, \\ \mathbf{y}_k &= \begin{bmatrix} \boldsymbol{\Omega}_k \\ \delta f_k \end{bmatrix}, \\ \mathbf{H}_{jk} &= \begin{bmatrix} \frac{\partial \mathbf{p}_{jk}}{\partial \boldsymbol{\Omega}_k} \\ \frac{\partial \mathbf{p}_{jk}}{\partial f_k} \end{bmatrix}, \end{aligned}$$

and

$$\frac{\partial \mathbf{p}_{jk}}{\partial \boldsymbol{\Omega}_k} = \frac{\partial (\mathbf{I} + \mathbf{X}(\boldsymbol{\Omega}))\mathbf{p}_{jk}}{\partial \boldsymbol{\Omega}_k} = \frac{\partial}{\partial \boldsymbol{\Omega}_k} \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix} \mathbf{p}_{jk} = -\mathbf{X}(\mathbf{p}_{jk}), \tag{13.53}$$

$$\frac{\partial \mathbf{p}_{jk}}{\partial f_k} = \mathbf{R}_k^{-1} \frac{\partial \tilde{\mathbf{x}}_{jk}}{\partial f_j} = \mathbf{R}_k^{-1} \begin{bmatrix} -x_{jk} f_k \\ -y_{jk} f_k \\ {l_{jk}}^2 - {f_k}^2 \end{bmatrix} / {l_{jk}}^3. \tag{13.54}$$

We therefore have the following linear equation for each point j matched in both frames k and l,

$$\begin{bmatrix} \mathbf{H}_{jk} & -\mathbf{H}_{jl} \end{bmatrix} \begin{bmatrix} \mathbf{y}_j \\ \mathbf{y}_k \end{bmatrix} = -\mathbf{e}_i \tag{13.55}$$

which leads to normal equations

$$\mathbf{A}\mathbf{y} = -\mathbf{b} \tag{13.56}$$

where the 4×4 (k,k)th block diagonal term and (k,l)th block off-diagonal term [20] of the symmetric $\mathbf{A}$ are defined by

$$\mathbf{A}_{kk} = \sum_j {\mathbf{H}_{jk}}^T \mathbf{H}_{jk} \tag{13.57}$$

$$\mathbf{A}_{kl} = -\sum_j {\mathbf{H}_{jk}}^T \mathbf{H}_{jl} \tag{13.58}$$

and the kth and lth 4×1 blocks of $\mathbf{b}$ are

$$\mathbf{b}_k = \sum_j {\mathbf{H}_{jk}}^T \mathbf{e}_j \tag{13.59}$$

$$\mathbf{b}_l = -\sum_j {\mathbf{H}_{jl}}^T \mathbf{e}_j. \tag{13.60}$$

Because $\mathbf{A}$ is symmetric, the normal equations can be stably solved using a symmetric positive definite (SPD) linear system solver. By incorporating additional constraints on the pose, we can formulate our minimization problem (13.49) as a constrained least-squares problem which can be solved using Lagrange multipliers. Details of the constrained least-squares can be found in Appendix 13.11. Possible linear constraints include:

- $\Omega_0 = 0$. First frame pose is unchanged. For example, the first frame can be chosen as the world coordinate system.
- $\delta f_k = 0$ for all N frames $j = 0, 1, ..., N-1$. All focal lengths are known.

[20] The sequential pairwise alignment algorithm described in Section 2 and Section 3 can be regarded as a special case of the global alignment (13.56) where the off-diagonal terms $\mathbf{A}_{kl}$ (13.58) and $\mathbf{b}_l$ (13.60) are zero if frame k is set fixed.

- $\delta f_k = \delta f_0$ for $j = 1, ..., N$. All focal lengths are the same but unknown.
- $\delta f_k = \delta f_l$, $\Omega_k = \Omega_l$, Frame j is the same as frame k. In order to apply this constraint, we also need to set $f_k = f_l$ and $\mathbf{R}_k = \mathbf{R}_l$.

The above minimization process converges quickly (several iterations) in practice. The running time for the iterative non-linear least-squares solver is much less than the time required to build the point correspondences.

13.6.4 Optimizing in Screen Coordinates

Now we return to Equation (13.45) to solve global alignment using screen coordinates. If we update $\mathbf{M}_k$ and $\mathbf{M}_l$ by

$$\mathbf{M}_k \leftarrow (\mathbf{I} + \mathbf{D}_k)\mathbf{M}_k \quad \text{and} \quad \mathbf{M}_l \leftarrow (\mathbf{I} + \mathbf{D}_l)\mathbf{M}_l, \tag{13.61}$$

we get

$$\begin{aligned} \mathbf{M}_{kl} &\leftarrow (\mathbf{I} + \mathbf{D}_{kl})\mathbf{M}_{kl} \\ &= (\mathbf{I} + \mathbf{D}_k)\mathbf{M}_k\mathbf{M}_l^{-1}(\mathbf{I} - \mathbf{D}_l) \\ &= (\mathbf{I} + \mathbf{D}_k - \mathbf{M}_{kl}\mathbf{D}_l\mathbf{M}_{kl}^{-1})\mathbf{M}_{kl}. \end{aligned}$$

Because of linear relationship between $\mathbf{D}_{kl}$ and $\mathbf{D}_k$, $\mathbf{D}_l$, we can find out the Jacobians

$$\mathbf{J}_k = \frac{\partial \mathbf{d}_{kl}}{\partial \mathbf{d}_k} \quad \text{and} \quad \mathbf{J}_l = \frac{\partial \mathbf{d}_{kl}}{\partial \mathbf{d}_l}. \tag{13.62}$$

In fact, $\mathbf{J}_k = \mathbf{I}$. Since we know how to estimate $\mathbf{D}_{kl}$ from patch-based alignment, we can expand the original 8×8 system (assuming perspective case) $\mathbf{A}\mathbf{d}_{kl} = \mathbf{b}$ to four blocks of 8×8 system, much like equations (57)-(60). An example of global alignment in screen coordinates is shown in Section 13.8.

13.7 Deghosting (Local Alignment)

After the global alignment has been run, there may still be localized mis-registrations present in the image mosaic, due to deviations from the idealized parallax-free camera model. Such deviations might include camera translation (especially for hand-held cameras), radial distortion, the mis-location of the optical center (which can be significant for scanned photographs or Photo CDs), and moving objects.

To compensate for these effects, we would like to quantify the amount of mis-registration and to then locally warp each image so that the overall

mosaic does not contain visible *ghosting* (double images) or blurred details. If our mosaic contains just a few images, we could choose one image as the *base*, and then compute the optical flow between it and all other images, which could then be deformed to match the base. Another possibility would be to explicitly estimate the camera motion and residual parallax [161, 239, 271], but this would not compensate for other distortions.

However, since we are dealing with large image mosaics, we need an approach which makes all of the images globally consistent, without a preferred base. One approach might be to warp each image so that it best matches the current mosaic. For small amounts of misregistration, where most of the visual effects are simple blurring (loss of detail), this should work fairly well. However, for large misregistrations, where ghosting is present, the local motion estimation would likely fail. Another approach is to select pixels from only one image at a time [185, 214, 297, 56].

The approach we have adopted is to compute the flow between all pairs of images, and to then infer the desired local warps from these computations. While in principle any motion estimation or optical flow technique could be used, we use the the patch-based alignment algorithm described in Section 13.4.2, since it provides us with the required information and allows us to reason about geometric consistency.

Recall that the block adjustment algorithm (Section 13.50) provides an estimate $\mathbf{p}_j$ of the true direction in space corresponding to the jth patch center in the kth image, $\mathbf{x}_{jk}$. The projection of this direction onto the kth image is

$$\mathbf{x}_{jk} \sim \mathbf{V}_k \mathbf{R}_k \frac{1}{n_{jk}+1} \sum_{l \in \{\mathcal{N}_{jk} \cup k\}} \mathbf{R}_l^{-1} \mathbf{V}_l^{-1} \mathbf{x}_{jl} = \frac{1}{n_{jk}+1} \left(\mathbf{x}_{jk} + \sum_{l \in \mathcal{N}_{jk}} \tilde{\mathbf{x}}_{jl} \right). \tag{13.63}$$

This can be converted into a motion estimate

$$\bar{\mathbf{u}}_{jk} = \mathbf{x}_{jk} - \mathbf{x}_{jk} = \frac{1}{n_{jk}+1} \sum_{l \in \mathcal{N}_{jk}} (\tilde{\mathbf{x}}_{jl} - \mathbf{x}_{jk}) = \frac{1}{n_{jk}+1} \sum_{l \in \mathcal{N}_{jk}} \mathbf{u}_{jl}. \tag{13.64}$$

This formula has a very nice, intuitively satisfying explanation (Figure 13.7c). The local motion required to bring patch center j in image k into global registration is simply the average of the pairwise motion estimates with all overlapping images, *downweighted* by the fraction $n_{jk}/(n_{jk}+1)$. This factor prevents local motion estimates from "overshooting" in their corrections (consider, for example, just two images, where each image warps itself to match its neighbor). Thus, we can compute the location motion estimate for each image by simply examining its misregistration with its neighbors, without having to worry about what warps these other neighbors might be undergoing themselves.

Once the local motion estimates have been computed, we need an algorithm to warp each image so as to reduce ghosting. One possibility would be to use a *forward mapping* algorithm [294] to convert each image I_k into a new image I'_k. However, this has the disadvantage of being expensive to compute, and of being susceptible to tears and foldovers.

Instead, we use an *inverse mapping* algorithm, which was already present in our system to perform warpings into cylindrical coordinates and to optionally compensate for radial lens distortions [273]. Thus, for each pixel in the *new* (warped) image I'_k, we need to know the relative distance (flow) to the appropriate source pixel. We compute this field using a sparse data interpolation technique [202]. The input to this algorithm is the set of negative flows $-\overline{\mathbf{u}}_{jk}$ located at pixel coordinates $\mathbf{x}_{jk} = \mathbf{x}_{jk} + \overline{\mathbf{u}}_{jk}$. At present, we simply place a tent (bilinear) function over each flow sample (the size is currently twice the patch size). To make this interpolator locally *reproducing* (no "dips" in the interpolated surface), we divide each accumulated flow value by the accumulated weight (plus a small amount, say 0.1, to round the transitions into regions with no motion estimates[21]).[22]

The results of our deghosting technique can be seen in Figures 13.11–13.14 along with some sample computed warp fields. Note that since the deghosting technique may not give perfect results (because it is patch-based, and not pixel-based), we may wish to iteratively apply the algorithm (the warp field is simply incrementally updated).

Even though we have formulated local alignment using rotational mosaic representation, the deghosting equation (13.63) is valid for other motion models (e.g., 8-parameter perspective) as well. We need only to modify (13.63) to

$$\mathbf{x}_{jk} \sim \mathbf{M}_k \frac{1}{n_{jk}+1} \sum_{l \in \{\mathcal{N}_{jk} \cup k\}} \mathbf{M}_l^{-1} \mathbf{x}_{jl} = \frac{1}{n_{jk}+1} \left(\mathbf{x}_{jk} + \sum_{l \in \mathcal{N}_{jk}} \tilde{\mathbf{x}}_{jl} \right). \tag{13.65}$$

13.8 Experiments

In this section we present the results of applying our global and local alignment techniques to image mosaicing. We have tested our methods on a number of real image sequences. In all of the experiments, we have used the rotational panoramic representation with unknown focal length. In general, two neighbor images have about 50% overlap.

The speed of our patch-based image alignment depends on the following parameters: motion model, image size, alignment accuracy, level of pyra-

[21] This makes the interpolant no longer perfectly reproducing.

[22] In computer graphics, this kind of interpolation is often called *splatting* [292].

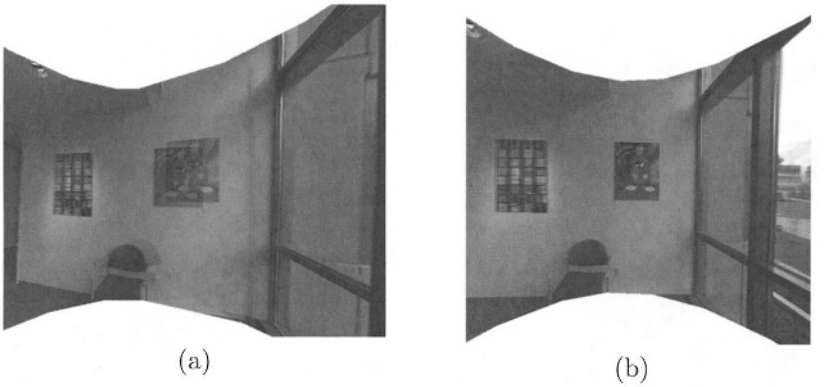


FIGURE 13.8. Reducing accumulated errors of image mosaics by block adjustment. (a): image mosaics with gaps/overlap; (b): corresponding mosaics after applying block adjustment.

mid, patch size, and initial misalignment. Typically, we set the patch size 16, the alignment accuracy 0.04 pixel, and we use a 3 level pyramid. Using our rotational model, it takes a few seconds (on a Pentium 200MHz PC) to align two images of size 384×300, with an initial misregistration of about 30 pixels. The speed of global alignment and local alignment mainly depends on the correlation-style search range while building feature correspondence. It takes several minutes to do the block adjustment for a sequence of 20 images with a patch size of 16 and a search range of 4.

13.8.1 Global Alignment

The first example shows how misregistration errors quickly accumulate in sequential registration. Figure 13.8a shows a big gap at the end of registering a sequence of 24 images (image size 384×300) where an initial estimate of focal length 256 is used. The double image of the right painting on the wall signals a big misalignment. This double image is removed, as shown in Figure 13.8b, by applying our global alignment method which simultaneously adjusts all frame rotations and computes a new estimated focal length of 251.8. To reduce the search range for correspondence, we append the first image at the end of image sequence and enforce a hard constraint that the first image has to be the last one.

In Section 13.5, we proposed an alternative "gap closing" technique to handle the accumulated misregistration error. However, this technique only works well for a sequence of images with uniform motion steps. It also requires that the sequence of images follow a great circle on the viewing sphere. The global alignment method in Section 13.6, on the other hand, does not make such assumptions. For example, our global alignment method can handle the misalignment (Figure 13.9c, which is the close-up of double image on the middle left side of Figure 13.9a) of an image mosaic

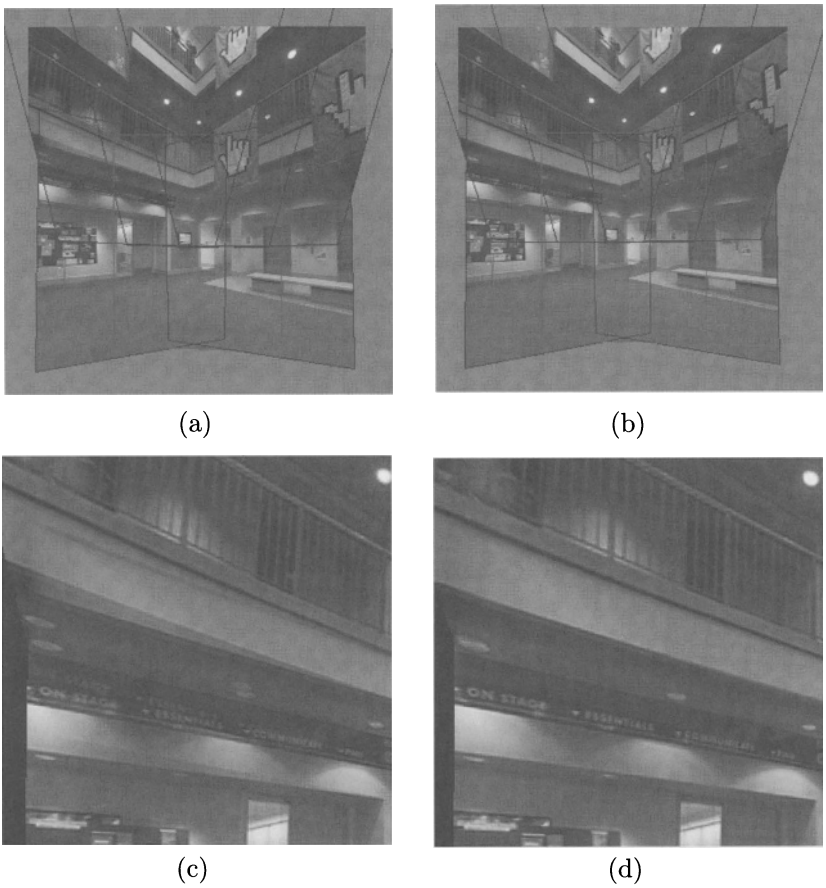


FIGURE 13.9. Reducing accumulated errors of image mosaics by block adjustment. (a): image mosaics with gaps/overlap; (b): corresponding mosaics after applying block adjustment; (c) and (d): close-ups of left middle regions of (a) and (b), respectively.

which is constructed from 6 images taken with a camera leveled and tilted up. Figures 13.9b and 13.9d (close-up) show the image mosaic after block adjustment where the visible artifacts are no longer apparent. In this example we do not enforce the constraint that the first frame has to be the last one (i.e., do not add the first image to the end of sequence), nor do the images form a great circle on the viewing sphere (only six images are used).

As discussed in Section 13.6.4, our global alignment method also works for non-rotational motion models (e.g., perspective) where optimization is formulated in screen coordinates. Figure 13.10a shows an image mosaic composed of 5 whiteboard images using 8-parameter perspective model. The double image in the top left region of Figure 13.10a (or Figure 13.10c

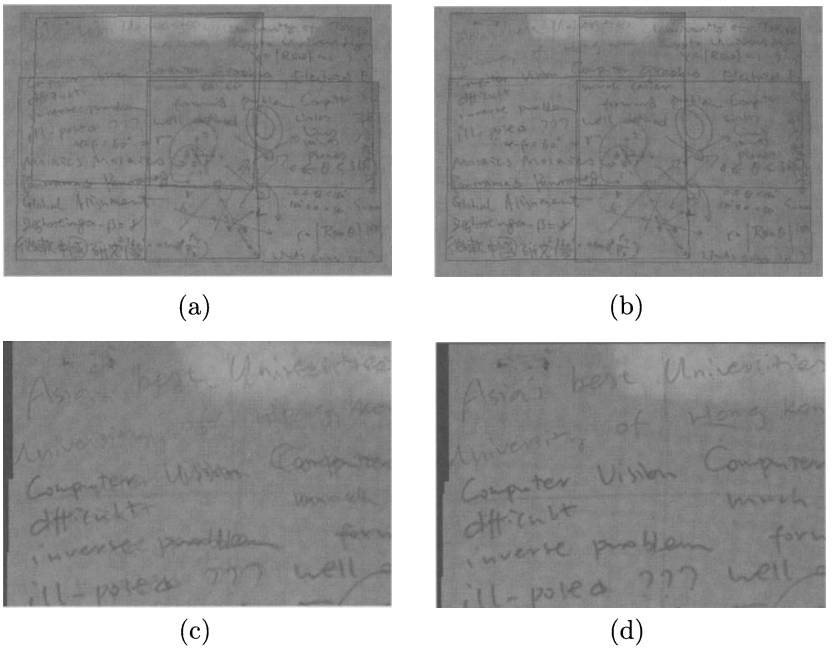
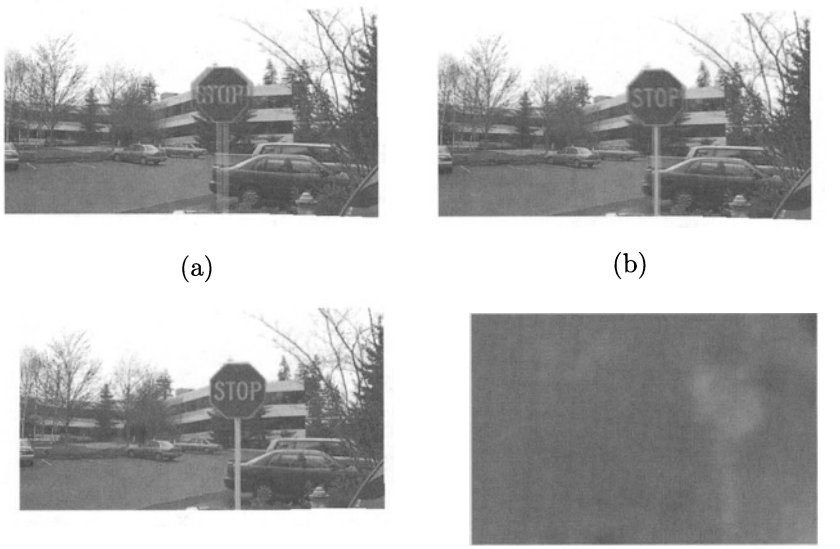


FIGURE 13.10. Reducing accumulated errors of image mosaics by global alignment in screen coordinates. (a): image mosaic with error accumulation (at the top left region); (c): corresponding mosaic after applying global alignment; (b) and (d): close-ups of (a) and (c) respectively.

for a close-up) due to accumulated registration error is significantly reduced after applying our global alignment method, as shown in Figure 13.10b and 13.10d.

13.8.2 Local Alignment

The next two examples illustrate the use of local alignment for sequences where the global motion model is clearly violated. The first example consists of two images taken with a hand-held digital camera (Kodak DC40) where some camera translation is present. The parallax introduced by this camera translation can be observed in the registered image (Figure 13.11a) where the front object (a stop sign) causes a double image because of the misregistration. This misalignment is significantly reduced using our local alignment method (Figure 13.11b). However, some visual artifacts still exist because our local alignment is patch-based (e.g. patch size 32 is used in Figure 13.11b). To overcome this problem, we repeatedly apply local alignment with successively smaller patches, which has the advantage of being able to handle large motion parallax and refine local alignment. Figure 13.11c shows the result after applying local alignment three times with patch sizes of 32, 16 and 8. The search range has been set to be half of



(a) (b)

(c) (d)

FIGURE 13.11. Deghosting an image mosaic with motion parallax: (a) image mosaic with parallax; (b) after single deghosting step (patch size 32); (c) after multiple deghosting steps (patch sizes 32, 16 and 8); (d) flow field of the left image.

the patch size for reliable patch motion estimation. Figure 13.11d shows the flow field corresponding to the left image (Figure 13.11e). Red values indicate rightward motion (e.g. the stop sign).

The global motion model is also invalid when registering two images with strong optical distortion. One way to deal with radial distortion is to carefully calibrate the camera. Another way is to use local alignment, making it possible to register images with optical distortion without using explicit camera calibration (i.e., recovering lens radial distortion).[23] Figure 13.12d shows one of two images taken with a Pulnix camera and a Fujinon F2.8 wide angle lens. This picture shows significant radial distortion; notice how straight lines (e.g., the door) are curved. The registration result is shown in Figure 13.12a. The mosaic after deghosting with a patch size 32 and search range 16 is shown in Figure 13.12b. Figure 13.12c shows an improved mosaic using repeated local alignment with patch sizes 32, 16, 8. The flow fields in Figure 13.12e–f show that the flow becomes larger towards the corner of the image due to radial distortion (bright green is upward motion, red is rightward motion). Notice however that these warp fields do *not* encode

[23]The recovered deformation field is not guaranteed, however, to be the true radial distortion, especially when only a few images are being registered. Recall that the minimum norm field is selected at each deghosting step.

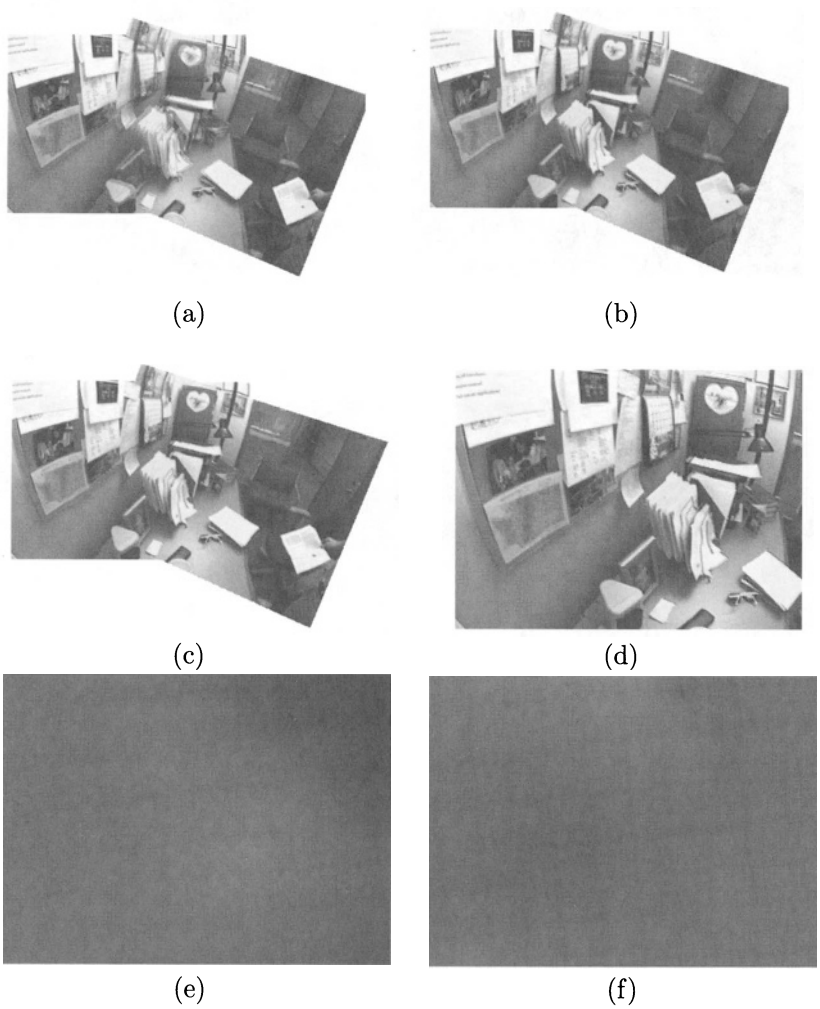


FIGURE 13.12. Deghosting an image mosaic with optical distortion: (a) image mosaic with distortion; (b) after single deghosting step (patch size 32); (c) after multiple deghosting steps (patch sizes 32, 16 and 8); (d) original left image; (e–f) flow fields of the two images after local alignment.

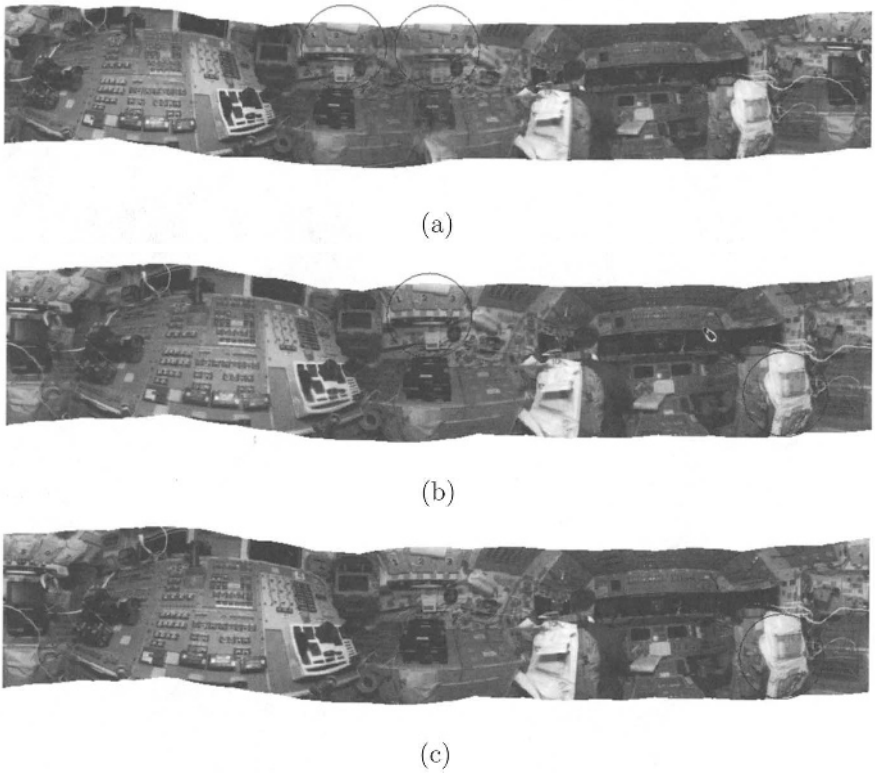


FIGURE 13.13. Panoramic image mosaics constructed from images taken with a hand-held camera: (a) significant accumulated error is visible in the center (repeated numbers 1-2-3); (b) with block adjustment, only small imperfections remain, such as the double image on the right pilot's chair; (c) with deghosting, the mosaic is virtually perfect.

the true radial distortion. A parametric deformation model (e.g., the usual quadratic plus quartic terms) would have to be used instead.

13.8.3 Additional Examples

We present two additional examples of large panoramic mosaics. The first mosaic uses a sequence of 14 images taken with a hand-held camera by an astronaut on the Space Shuttle flight deck. This sequence of images has significant motion parallax and is very difficult to register. The accumulated error causes a very big gap between the first and last images as shown in Figure 13.13a (notice the repeated "1 2 3" numbers, which should only appear once). We are able to construct a good quality panorama (Figure 13.13b) using our block adjustment technique (there is some visible ghosting, however, near the right pilot chair). This panorama is further

FIGURE 13.14. Four views of an image mosaic of lobby constructed from 3 sequences of 50 images.

refined with deghosting as shown in Figure 13.13c. These panoramas were rendered by projecting the image mosaic onto a tessellated spherical map.

The final example shows how to build a full view panoramic mosaic. Three panoramic image sequences of a building lobby were taken with the camera on a tripod tilted at three different angles (with 22 images for the middle sequence, 22 images for the upper sequence, and 10 images for the top sequence). The camera motion covers more than two thirds of the viewing sphere, including the top. After registering all of the images sequentially with patch-based alignment, we apply our global and local alignment techniques to obtain the final image mosaic, shown in Figure 13.14. These four views of the final image mosaic are equivalent to images taken with a very large rectilinear lens. Each view is twice as big as the input image (300×384 with focal length 268), therefore, is equivalent to vertical field of view 110 degrees. A tessellated spherical map of the full view panorama is shown in Figure 13.15. Our algorithm for building texture-mapped polyhedra from panoramic image mosaics is described in the next section.

FIGURE 13.15. Tessellated spherical panorama covering the north pole (constructed from 50 images). The white triangles at the top are the parts of the texture map not covered in the 3D tessellated globe model (due to triangular elements at the poles).

13.9 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 13.2). When being used as an environment map, such a representation is sometimes called a latitude-longitude projection [86]. 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 [86]. 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 [86], to something as complicated as a subdivided dodecahedron, or even a latitude-longitude tessellated globe.[24] 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 auxiliary 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 [293]. For example, when using a six-sided cube, we set the (u, v)

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

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 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 [290]). 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 13.15 shows the results of mapping a panoramic mosaic onto a longitude-latitude tessellated globe. The white triangles at the top are the parts of the texture map not covered in the 3D tessellated globe model (due to triangular elements at the poles). Figures 13.16–13.18 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 13.16 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

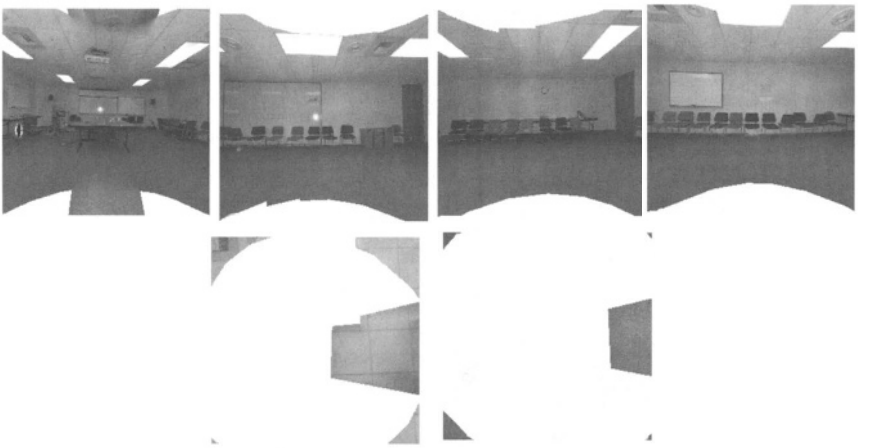


FIGURE 13.16. Cubical texture-mapped model of conference room (from 75 images taken with a hand-held digital camera).

FIGURE 13.17. Cubical texture-mapped model of lobby (from 50 images).

are currently using a simple 3D viewer written on top of the Direct3D API running on a personal computer.

13.10 Discussion

In this paper, we have presented our system for constructing full view panoramic image mosaics from image sequences. Instead of projecting all of the images onto a common surface (e.g., a cylinder or a sphere), we use a representation that associates a rotation matrix and a focal length with each input image. Based on this rotational panoramic representation, we use block adjustment (global alignment) and deghosting (local alignment)

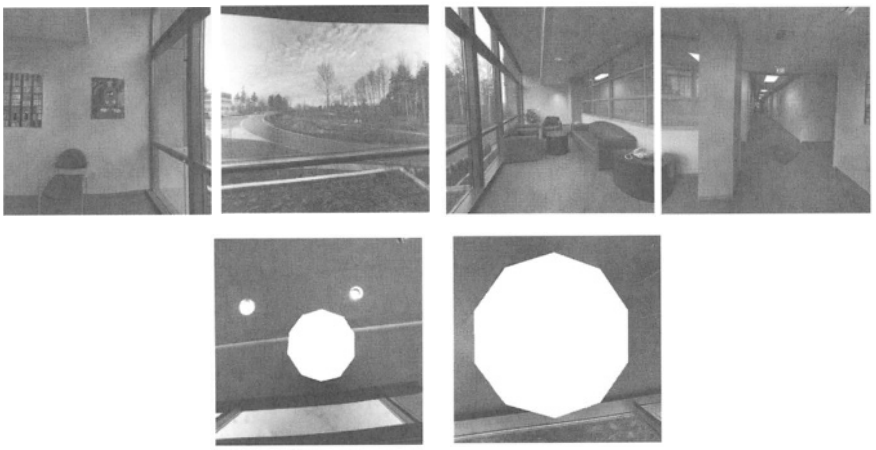


FIGURE 13.18. Cubical texture-mapped model of hallway and sitting area (from 70 images).

techniques to significantly improve the quality of image mosaics, thereby enabling the construction of mosaics from images taken by hand-held cameras.

When constructing an image mosaic from a long sequence of images, we have to deal with error accumulation problems. Our system simultaneously adjusts all frame poses (rotations and focal lengths) so that the sum of registration errors between all matching pairs of images is minimized. Geometrically, this is equivalent to adjusting all ray directions of corresponding pixels in overlapping frames until they converge. Using corresponding "features" in neighboring frames, which are obtained automatically using our patch-based alignment method, we formulate the minimization problem to recover the poses without explicitly computing the converged ray directions. This leads to a linearly-constrained non-linear least-squares problem which can be solved very efficiently.[25]

To compensate for local misregistration caused by inadequate motion models (e.g., camera translation[26] or moving object) or imperfect camera projection models (e.g., lens distortion), we refine the image mosaic using a deghosting method. We divide each image into small patches and compute patch-based alignments. We then locally warp each image so that the overall mosaic does not contain visible ghosting. To handle large parallax

[25]If we were only adjusting one rotation matrix at a time, we could use Horn's absolute orientation algorithm [112, 113, 143]. Unfortunately, this would be converge more slowly than solving a single global optimization problem.

[26]We assume in our work that the camera translation is relatively small. When camera translation is significant, a "manifold mosaic" [214] can still be constructed from a dense sequence of images using only center columns of each image. However, the resulting mosaic is no longer metric.

or distortion, we start the deghosting with a large patch size. This deghosting step is then repeated with smaller patches so that local patch motion can be estimated more accurately. In the future, we plan to implement a multiresolution patch-based flow algorithm so that the alignment process can be sped up and made to work over larger displacements. We also plan to develop more robust versions of our alignment algorithms.

Our deghosting algorithm can also be applied to the problem of extracting texture maps for general 3D objects from images [236]. When constructing such texture maps by averaging a number of views projected onto the model, even slight misregistrations can cause blurring or ghosting effects. One potential way to compensate for this is to refine the surface geometry to bring all projected colors into registration [76]. Our deghosting algorithm can be used as an alternative, and can inherently compensate for problems such as errors in the estimated camera geometry and intrinsic camera models.

To summarize, the collection of global and local alignment algorithms presented in this paper, together with our efficient patch-based implementation, make it easy to quickly and reliably construct high-quality full view panoramic mosaics from arbitrary collections of images, without the need for special photographic equipment. We believe that this will make panoramic photography and the construction of virtual environments much more interesting to a wide range of users, and stimulate further research and development in image-based rendering and the representation of visual scenes.

13.11 Appendix: Linearly-constrained Least-squares

We would like to solve the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{13.66}$$

subject to

$$\mathbf{C}\mathbf{x} = \mathbf{q}. \tag{13.67}$$

It is equivalent to minimizing

$$\sum_i (\mathbf{A}_i^T \mathbf{x} - b_i)^2 \tag{13.68}$$

subject to

$$\mathbf{c}_j^T \mathbf{x} - q_j = 0 \tag{13.69}$$

for all j, where $\mathbf{c}_j$ are rows of $\mathbf{C}$.

13.11.1 Lagrange Multipliers

This problem is a special case of constrained nonlinear programming (or more specifically quadratic programming). Thus, it can be formulated using Lagrange multipliers by minimizing

$$e = \sum_i (\mathbf{A}_i^T \mathbf{x} - b_i)^2 + \sum_j 2\lambda_j (\mathbf{C}_j^T \mathbf{x} - q_j). \tag{13.70}$$

Taking first order derivatives of e with respect to $\mathbf{x}$ and λ, we have

$$\frac{\partial e}{\partial \mathbf{x}} = \sum_i \mathbf{A}_i \mathbf{A}_i^T \mathbf{x} - \sum_i b_i \mathbf{A}_i + \sum_j \lambda_j \mathbf{C}_j = 0 \tag{13.71}$$

and

$$\frac{\partial e}{\partial \lambda_j} = \mathbf{C}_j^T \mathbf{x}_j + q_j = 0 \tag{13.72}$$

or

$$\begin{bmatrix} \mathbf{H} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ \mathbf{g} \end{bmatrix} \tag{13.73}$$

where $\mathbf{H} = \sum_i \mathbf{A}_i \mathbf{A}_i^T$, $\mathbf{d} = \sum_i \mathbf{A}_i b_i$, $\mathbf{x} = [x_j]$, $\lambda = [\lambda_j]$, and $\mathbf{g} = [g_j]$.

If $\mathbf{H}$ is invertible, we can simply solve the above system by

$$\begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{H}^{-1} - \mathbf{K}\mathbf{C}\mathbf{H}^{-1} & \mathbf{K} \\ \mathbf{K}^T & -\mathbf{P} \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \mathbf{g} \end{bmatrix} \tag{13.74}$$

where

$$\begin{aligned} \mathbf{P} &= (\mathbf{C}\mathbf{H}^{-1}\mathbf{C}^T)^{-1} \\ \mathbf{K} &= \mathbf{H}^{-1}\mathbf{C}^T\mathbf{P} \end{aligned}$$

Unfortunately, this requires additional matrix inversion operations.

13.11.2 Elimination Method

We now present a simple method to solve the linearly-constrained least-squares problem by eliminating redundant variables using given hard constraints.

If there are no hard constraints (i.e., $\mathbf{C}\mathbf{x} = \mathbf{q}$), we can easily solve the least-squares problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ using normal equations, i.e.,

$$\mathbf{H}\mathbf{x} = \mathbf{d} \tag{13.75}$$

where $\mathbf{H} = \mathbf{A}^T\mathbf{A}$, and $\mathbf{d} = \mathbf{A}^T\mathbf{b}$. The normal equations can be solved stably using a SPD solver. We would like to modify the normal equations using the given hard linear constraints so that we can formulate new normal equations $\tilde{\mathbf{H}}\mathbf{x} = \tilde{\mathbf{d}}$ which are also SPD and of the same dimensions as $\mathbf{H}$.

Without loss of generality, we consider only one linear constraint and assume the biggest entry is $l_k = 1$. Let $\mathbf{H}_k$ be the kth column of $\mathbf{H}$, and $\mathbf{A}_k$ be the kth column of $\mathbf{A}$. If we subtract the linear constraint properly from each row of $\mathbf{A}$ so that its k-th column becomes zero, we change the original system to

$$\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{d}} \tag{13.76}$$

subject to

$$\mathbf{c}^T\mathbf{x} = q \tag{13.77}$$

where $\tilde{\mathbf{A}} = \mathbf{A} - \mathbf{A}_k\mathbf{c}^T$ and $\tilde{\mathbf{d}} = \mathbf{d} - \mathbf{A}_k q$.

Because the constraint (13.77) is linearly independent of the linear system (13.76), we can formulate new normal equations with

$$\begin{aligned}\tilde{\mathbf{H}} &= \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \mathbf{c}\mathbf{c}^T \\ &= (\mathbf{A} - \mathbf{A}_k\mathbf{c}^T)^T\mathbf{A} - \mathbf{A}_k\mathbf{c}^T + \mathbf{c}\mathbf{c}^T \\ &= \mathbf{H} - \mathbf{c}\mathbf{H}_k^T - \mathbf{H}_k\mathbf{c}^T + (1 + h_{kk})\mathbf{c}\mathbf{c}^T\end{aligned}$$

and

$$\begin{aligned}\tilde{\mathbf{d}} &= \tilde{\mathbf{A}}^T\tilde{\mathbf{d}} + \mathbf{c}q \\ &= \mathbf{d} - \mathbf{H}_k q - \mathbf{c}d_k + (1 + h_{kk})\mathbf{c}q\end{aligned}$$

where $\mathbf{H}_k$ is the kth column of $\mathbf{H}$ and $h_{kk} = \mathbf{A}_k^T\mathbf{A}_k$ is the kth diagonal element of $\mathbf{H}$.

It is interesting to note that the new normal equations are not unique because we can arbitrarily scale the hard constraint.[27] For example, if we scale Equation (13.77) by h_{kk}, we have $\tilde{\mathbf{H}} = \mathbf{H} - \mathbf{c}\mathbf{H}_k^T - \mathbf{H}_k\mathbf{c}^T + 2h_{kk}\mathbf{c}\mathbf{c}^T$ and $\tilde{\mathbf{d}} = \mathbf{d} - \mathbf{H}_k q - \mathbf{c}d_k + 2h_{kk}\mathbf{c}q$.

To add multiple constraints, we simply adjust the original system multiple times, one constraint at a time. The order of adding multiple constraints does not matter.

13.11.3 QR Factorization

The elimination method is very efficient if we have only a few constraints. When the number of constraints increases, we can use QR factorization to solve the linearly-constrained least-squares [83]. Suppose $\mathbf{A}$ and $\mathbf{C}$ are of full ranks, let

$$\mathbf{C}^T = \mathbf{Q}\begin{bmatrix}\mathbf{R} \\ 0\end{bmatrix} \tag{13.78}$$

[27] One can not simply scale any soft constraints (i.e., the linear equations $\mathbf{A}_i x_i = b_i$) because it adds different weights to the least-squares formulation, that leads to incorrect solutions.

be the QR factorization of $\mathbf{c}^T$ where $\mathbf{Q}$ is orthorgonal, $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. If we define $\mathbf{Q}^T\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$, $\mathbf{A}\mathbf{Q} = (\mathbf{A}_1, \mathbf{A}_2)$, we can solve $\mathbf{x}_1$ because $\mathbf{R}$ is upper diagonal and

$$\begin{aligned} \mathbf{C}\mathbf{x} &= \mathbf{C}\mathbf{Q}\mathbf{Q}^T\mathbf{x} \\ &= \mathbf{R}^T\mathbf{x}_1 = \mathbf{q}. \end{aligned}$$

Then we solve $\mathbf{x}_2$ from the unconstrained least-squares $\|\mathbf{A}_2\mathbf{x}_2 - (\mathbf{b} - \mathbf{A}_1\mathbf{x}_1)\|^2$ because

$$\begin{aligned} \mathbf{A}\mathbf{x} - \mathbf{b} &= \mathbf{A}\mathbf{Q}\mathbf{Q}^T\mathbf{x} - \mathbf{b} \\ &= \mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 - \mathbf{b} \\ &= \mathbf{A}_2\mathbf{x}_2 - (\mathbf{b} - \mathbf{A}_1\mathbf{x}_1). \end{aligned}$$

Finally $\mathbf{x} = \mathbf{Q}\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$. Note that this method requires two factorizations.