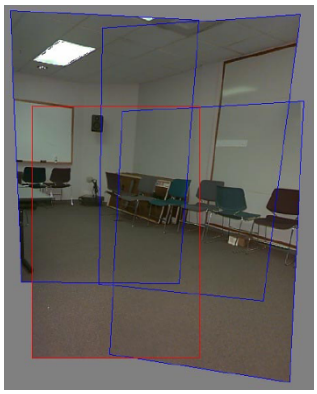
Chapter 8

Image alignment and stitching

R. Szeliski, *Computer Vision*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-34372-9_8



(a)

(b)

(c)

(d)

(e)

Figure 8.1 Image stitching: (a) geometric alignment of 2D images for stitching (Szeliski and Shum 1997) © 1997 ACM; (b) a spherical panorama constructed from 54 photographs (Szeliski and Shum 1997) © 1997 ACM; (c) a multi-image panorama automatically assembled from an unordered photo collection; a multi-image stitch (d) without and (e) with moving object removal (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE.

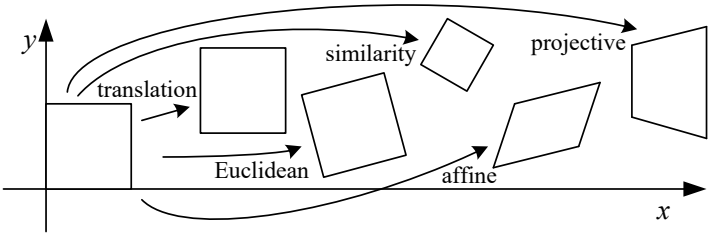


Figure 8.2 Basic set of 2D planar transformations

Once we have extracted features from images, the next stage in many vision algorithms is to match these features across different images (Section 7.1.3). An important component of this matching is to verify whether the set of matching features is geometrically consistent, e.g., whether the feature displacements can be described by a simple 2D or 3D geometric transformation. The computed motions can then be used in other applications such as image stitching (Section 8.2) or augmented reality (Section 11.2.2).

In this chapter, we look at the topic of geometric image registration, i.e., the computation of 2D and 3D transformations that map features in one image to another (Section 8.1). In Chapter 11, we look at the related problems of *pose estimation*, which is determining a camera's position relative to a known 3D object or scene, and *structure from motion*, i.e., how to simultaneously estimate 3D geometry and camera motion.

8.1 Pairwise alignment

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points. In this section, we restrict ourselves to global *parametric* transformations, such as those described in Section 2.1.1 and shown in Table 2.1 and Figure 8.2, or higher order transformation for curved surfaces (Shashua and Toelg 1997; Can, Stewart *et al.* 2002). Applications to non-rigid or elastic deformations (Bookstein 1989; Kambhamettu, Goldgof *et al.* 1994; Szeliski and Lavallée 1996; Torresani, Hertzmann, and Bregler 2008) are examined in Sections 9.2.2 and 13.6.4.

8.1.1 2D alignment using least squares

Given a set of matched feature points $\{(\mathbf{x}_i, \mathbf{x}'_i)\}$ and a planar parametric transformation[1] of the form

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}; \mathbf{p}), \tag{8.1}$$

how can we produce the best estimate of the motion parameters $\mathbf{p}$? The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals

$$E_{\mathrm{LS}} = \sum_i \|\mathbf{r}_i\|^2 = \sum_i \|\mathbf{f}(\mathbf{x}_i; \mathbf{p}) - \mathbf{x}'_i\|^2, \tag{8.2}$$

where

$$\mathbf{r}_i = \mathbf{x}'_i - \mathbf{f}(\mathbf{x}_i; \mathbf{p}) = \hat{\mathbf{x}}'_i - \tilde{\mathbf{x}}'_i \tag{8.3}$$

[1] For examples of non-planar parametric models, such as quadrics, see the work of Shashua and Toelg (1997) and Shashua and Wexler (2001).

Transform	Matrix	Parameters $\mathbf{p}$	Jacobian $\mathbf{J}$
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 8.1.3)

Table 8.1 Jacobians of the 2D coordinate transformations $\mathbf{x}' = \mathbf{f}(\mathbf{x}; \mathbf{p})$ shown in Table 2.1, where we have re-parameterized the motions so that they are identity for $\mathbf{p} = 0$.

is the *residual* between the measured location $\hat{\mathbf{x}}_i'$ and its corresponding current *predicted* location $\tilde{\mathbf{x}}_i' = \mathbf{f}(\mathbf{x}_i; \mathbf{p})$. (See Appendix A.2 for more on least squares and Appendix B.2 for a statistical justification.)

Many of the motion models presented in Section 2.1.1 and Table 2.1, i.e., translation, similarity, and affine, have a *linear* relationship between the amount of motion $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x}$ and the unknown parameters $\mathbf{p}$,

$$\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} = \mathbf{J}(\mathbf{x})\mathbf{p}, \tag{8.4}$$

where $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{p}$ is the *Jacobian* of the transformation $\mathbf{f}$ with respect to the motion parameters $\mathbf{p}$ (see Table 8.1). In this case, a simple *linear* regression (linear least squares problem) can be formulated as

$$E_{\text{LLS}} = \sum_i \|\mathbf{J}(\mathbf{x}_i)\mathbf{p} - \Delta\mathbf{x}_i\|^2 \tag{8.5}$$

$$= \mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i)\right] \mathbf{p} - 2\mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i\right] + \sum_i \|\Delta\mathbf{x}_i\|^2 \tag{8.6}$$

$$= \mathbf{p}^T\mathbf{A}\mathbf{p} - 2\mathbf{p}^T\mathbf{b} + c. \tag{8.7}$$

The minimum can be found by solving the symmetric positive definite (SPD) system of *normal equations*[2]

$$\mathbf{A}\mathbf{p} = \mathbf{b}, \tag{8.8}$$

where

$$\mathbf{A} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i) \tag{8.9}$$

[2]For poorly conditioned problems, it is better to use QR decomposition on the set of linear equations $\mathbf{J}(\mathbf{x}_i)\mathbf{p} = \Delta\mathbf{x}_i$ instead of the normal equations (Björck 1996; Golub and Van Loan 1996). However, such conditions rarely arise in image registration.

Figure 8.3 A simple panograph consisting of three images automatically aligned with a translational model and then averaged together.

is called the *Hessian* and $\mathbf{b} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i$. For the case of pure translation, the resulting equations have a particularly simple form, i.e., the translation is the average translation between corresponding points or, equivalently, the translation of the point centroids.

Uncertainty weighting. The above least squares formulation assumes that all feature points are matched with the same accuracy. This is often not the case, since certain points may fall into more textured regions than others. If we associate a scalar variance estimate σ_i^2 with each correspondence, we can minimize the *weighted least squares* problem instead,[3]

$$E_{\mathrm{WLS}} = \sum_i \sigma_i^{-2} \|\mathbf{r}_i\|^2. \tag{8.10}$$

As shown in Section 9.1.3, a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the *patch Hessian* $\mathbf{A}_i$ (9.48) with the per-pixel noise covariance σ_n^2 (9.37). Weighting each squared residual by its inverse covariance $\mathbf{\Sigma}_i^{-1} = \sigma_n^{-2}\mathbf{A}_i$ (which is called the *information matrix*), we obtain

$$E_{\mathrm{CWLS}} = \sum_i \|\mathbf{r}_i\|^2_{\mathbf{\Sigma}_i^{-1}} = \sum_i \mathbf{r}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{r}_i = \sum_i \sigma_n^{-2} \mathbf{r}_i^T \mathbf{A}_i \mathbf{r}_i. \tag{8.11}$$

8.1.2 *Application*: Panography

One of the simplest (and most fun) applications of image alignment is a special form of image stitching called *panography*. In a panograph, images are translated and optionally rotated and scaled before being blended with simple averaging (Figure 8.3). This process mimics the photographic collages created by artist David Hockney, although his compositions use an opaque overlay model, being created out of regular photographs.

In most of the examples seen on the web, the images are aligned by hand for best artistic effect.[4] However, it is also possible to use feature matching and alignment techniques to perform the registration automatically (Nomura, Zhang, and Nayar 2007; Zelnik-Manor and Perona 2007).

[3] Problems where each measurement can have a different variance or uncertainty are called *heteroscedastic models*.

[4] https://www.flickr.com/groups/panography.

Consider a simple translational model. We want all the corresponding features in different images to line up as best as possible. Let $\mathbf{t}_j$ be the location of the jth image coordinate frame in the global composite frame and $\mathbf{x}_{ij}$ be the location of the ith matched feature in the jth image. In order to align the images, we wish to minimize the least squares error

$$E_{\mathrm{PLS}} = \sum_{ij} \|(\mathbf{t}_j + \mathbf{x}_{ij}) - \mathbf{x}_i\|^2, \tag{8.12}$$

where $\mathbf{x}_i$ is the consensus (average) position of feature i in the global coordinate frame. (An alternative approach is to register each pair of overlapping images separately and then compute a consensus location for each frame—see Exercise 8.2.)

The above least squares problem is indeterminate (you can add a constant offset to all the frame and point locations $\mathbf{t}_j$ and $\mathbf{x}_i$). To fix this, either pick one frame as being at the origin or add a constraint to make the average frame offsets be 0.

The formulas for adding rotation and scale transformations are straightforward and are left as an exercise (Exercise 8.2). See if you can create some collages that you would be happy to share with others on the web.

8.1.3 Iterative algorithms

While linear least squares is the simplest method for estimating parameters, most problems in computer vision do not have a simple linear relationship between the measurements and the unknowns. In this case, the resulting problem is called *non-linear least squares* or *non-linear regression*.

Consider, for example, the problem of estimating a rigid Euclidean 2D transformation (translation plus rotation) between two sets of points. If we parameterize this transformation by the translation amount (t_x, t_y) and the rotation angle θ, as in Table 2.1, the Jacobian of this transformation, given in Table 8.1, depends on the current value of θ. Notice how in Table 8.1, we have re-parameterized the motion matrices so that they are always the identity at the origin $\mathbf{p} = 0$, which makes it easier to initialize the motion parameters.

To minimize the non-linear least squares problem, we iteratively find an update $\Delta\mathbf{p}$ to the current parameter estimate $\mathbf{p}$ by minimizing

$$E_{\mathrm{NLS}}(\Delta\mathbf{p}) = \sum_i \|\mathbf{f}(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p}) - \mathbf{x}'_i\|^2 \tag{8.13}$$

$$\approx \sum_i \|\mathbf{J}(\mathbf{x}_i; \mathbf{p})\Delta\mathbf{p} - \mathbf{r}_i\|^2 \tag{8.14}$$

$$= \Delta\mathbf{p}^T \left[\sum_i \mathbf{J}^T\mathbf{J}\right] \Delta\mathbf{p} - 2\Delta\mathbf{p}^T \left[\sum_i \mathbf{J}^T\mathbf{r}_i\right] + \sum_i \|\mathbf{r}_i\|^2 \tag{8.15}$$

$$= \Delta\mathbf{p}^T\mathbf{A}\Delta\mathbf{p} - 2\Delta\mathbf{p}^T\mathbf{b} + c, \tag{8.16}$$

where the "Hessian"[5] $\mathbf{A}$ is the same as Equation (8.9) and the right-hand side vector

$$\mathbf{b} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{r}_i \tag{8.17}$$

is now a Jacobian-weighted sum of residual vectors. This makes intuitive sense, as the parameters are pulled in the direction of the prediction error with a strength proportional to the Jacobian.

[5]The "Hessian" $\mathbf{A}$ is not the true Hessian (second derivative) of the non-linear least squares problem (8.13). Instead, it is the approximate Hessian, which neglects second (and higher) order derivatives of $\mathbf{f}(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p})$.

Once $\mathbf{A}$ and $\mathbf{b}$ have been computed, we solve for $\Delta\mathbf{p}$ using

$$(\mathbf{A} + \lambda \text{diag}(\mathbf{A}))\Delta\mathbf{p} = \mathbf{b}, \tag{8.18}$$

and update the parameter vector $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$ accordingly. The parameter λ is an additional damping parameter used to ensure that the system takes a "downhill" step in energy (squared error) and is an essential component of the Levenberg–Marquardt algorithm (described in more detail in Appendix A.3). In many applications, it can be set to 0 if the system is successfully converging.

For the case of our 2D translation+rotation, we end up with a 3×3 set of normal equations in the unknowns $(\delta t_x, \delta t_y, \delta\theta)$. An initial guess for (t_x, t_y, θ) can be obtained by fitting a four-parameter similarity transform in (t_x, t_y, c, s) and then setting $\theta = \tan^{-1}(s/c)$. An alternative approach is to estimate the translation parameters using the centroids of the 2D points and to then estimate the rotation angle using polar coordinates (Exercise 8.3).

For the other 2D motion models, the derivatives in Table 8.1 are all fairly straightforward, except for the projective 2D motion (homography), which arises in image-stitching applications (Section 8.2). These equations can be re-written from (2.21) in their new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \quad \text{and} \quad y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \tag{8.19}$$

The Jacobian is therefore

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{p}} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \tag{8.20}$$

where $D = h_{20}x + h_{21}y + 1$ is the denominator in (8.19), which depends on the current parameter settings (as do x' and y').

An initial guess for the eight unknowns $\{h_{00}, h_{01}, \ldots, h_{21}\}$ can be obtained by multiplying both sides of the equations in (8.19) through by the denominator, which yields the linear set of equations,

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{8.21}$$

However, this is not optimal from a statistical point of view, since the denominator D, which was used to multiply each equation, can vary quite a bit from point to point.[6]

One way to compensate for this is to *reweight* each equation by the inverse of the current estimate of the denominator, D,

$$\frac{1}{D}\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{8.22}$$

While this may at first seem to be the exact same set of equations as (8.21), because least squares is being used to solve the over-determined set of equations, the weightings *do* matter and produce a different set of normal equations that performs better in practice.

[6] Hartley and Zisserman (2004) call this strategy of forming linear equations from rational equations the *direct linear transform*, but that term is more commonly associated with pose estimation (Section 11.2). Note also that our definition of the h_{ij} parameters differs from that used in their book, since we define h_{ii} to be the *difference* from unity and we do not leave h_{22} as a free parameter, which means that we cannot handle certain extreme homographies.

The most principled way to do the estimation, however, is to directly minimize the squared residual Equations (8.13) using the Gauss–Newton approximation, i.e., performing a first-order Taylor series expansion in $\mathbf{p}$, as shown in (8.14), which yields the set of equations

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \tag{8.23}$$

While these look similar to (8.22), they differ in two important respects. First, the left-hand side consists of unweighted *prediction errors* rather than point displacements and the solution vector is a *perturbation* to the parameter vector $\mathbf{p}$. Second, the quantities inside $\mathbf{J}$ involve *predicted* feature locations $(\tilde{x}', \tilde{y}')$ instead of *sensed* feature locations $(\hat{x}', \hat{y}')$. Both of these differences are subtle and yet they lead to an algorithm that, when combined with proper checking for downhill steps (as in the Levenberg–Marquardt algorithm), will converge to a local minimum. Note that iterating Equations (8.22) is not guaranteed to converge, since it is not minimizing a well-defined energy function.

Equation (8.23) is analogous to the *additive* algorithm for direct intensity-based registration (Section 9.2), since the change to the full transformation is being computed. If we prepend an incremental homography to the current homography instead, i.e., we use a *compositional* algorithm (described in Section 9.2), we get $D = 1$ (since $\mathbf{p} = 0$) and the above formula simplifies to

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}, \tag{8.24}$$

where we have replaced $(\tilde{x}', \tilde{y}')$ with (x, y) for conciseness.

8.1.4 Robust least squares and RANSAC

While regular least squares is the method of choice for measurements where the noise follows a normal (Gaussian) distribution, more robust versions of least squares are required when there are outliers among the correspondences (as there almost always are). In this case, it is preferable to use an *M-estimator* (Huber 1981; Hampel, Ronchetti *et al.* 1986; Black and Rangarajan 1996; Stewart 1999), which involves applying a robust penalty function $\rho(r)$ to the residuals

$$E_{\mathrm{RLS}}(\Delta\mathbf{p}) = \sum_i \rho(\|\mathbf{r}_i\|) \tag{8.25}$$

instead of squaring them.[7]

We can take the derivative of this function with respect to $\mathbf{p}$ and set it to 0,

$$\sum_i \psi(\|\mathbf{r}_i\|) \frac{\partial \|\mathbf{r}_i\|}{\partial \mathbf{p}} = \sum_i \frac{\psi(\|\mathbf{r}_i\|)}{\|\mathbf{r}_i\|} \mathbf{r}_i^T \frac{\partial \mathbf{r}_i}{\partial \mathbf{p}} = 0, \tag{8.26}$$

where $\psi(r) = \rho'(r)$ is the derivative of ρ and is called the *influence function*. If we introduce a *weight function*, $w(r) = \psi(r)/r$, we observe that finding the stationary point of (8.25) using (8.26) is equivalent to minimizing the *iteratively reweighted least squares* (IRLS) problem

$$E_{\mathrm{IRLS}} = \sum_i w(\|\mathbf{r}_i\|) \|\mathbf{r}_i\|^2, \tag{8.27}$$

[7] The plots for some commonly used robust penalty functions ρ can be found in Figure 4.7.

where the $w(\|\mathbf{r}_i\|)$ play the same local weighting role as σ_i^{-2} in (8.10). The IRLS algorithm alternates between computing the influence functions $w(\|\mathbf{r}_i\|)$ and solving the resulting weighted least squares problem (with fixed w values). Other incremental robust least squares algorithms can be found in the work of Sawhney and Ayer (1996), Black and Anandan (1996), Black and Rangarajan (1996), and Baker, Gross *et al.* (2003) and in textbooks and tutorials on robust statistics (Huber 1981; Hampel, Ronchetti *et al.* 1986; Rousseeuw and Leroy 1987; Stewart 1999).

While M-estimators can definitely help reduce the influence of outliers, in some cases, starting with too many outliers will prevent IRLS (or other gradient descent algorithms) from converging to the global optimum. A better approach is often to find a starting set of *inlier* correspondences, i.e., points that are consistent with a dominant motion estimate.[8]

Two widely used approaches to this problem are called RANdom SAmple Consensus, or RANSAC for short (Fischler and Bolles 1981), and *least median of squares* (LMS) (Rousseeuw 1984). Both techniques start by selecting (at random) a subset of k correspondences, which is then used to compute an initial estimate for $\mathbf{p}$. The *residuals* of the full set of correspondences are then computed as

$$\mathbf{r}_i = \tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i, \tag{8.28}$$

where $\tilde{\mathbf{x}}'_i$ are the *estimated* (mapped) locations and $\hat{\mathbf{x}}'_i$ are the sensed (detected) feature point locations.[9]

The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location, i.e., whose $\|\mathbf{r}_i\| \leq \epsilon$. (The ϵ value is application dependent but is often around 1–3 pixels.) Least median of squares finds the median value of the $\|\mathbf{r}_i\|^2$ values. The random selection process is repeated S times and the sample set with the largest number of inliers (or with the smallest median residual) is kept as the final solution. Either the initial parameter guess $\mathbf{p}$ or the full set of computed inliers is then passed on to the next data fitting stage.

When the number of measurements is quite large, it may be preferable to only score a subset of the measurements in an initial round that selects the most plausible hypotheses for additional scoring and selection. This modification of RANSAC, which can significantly speed up its performance, is called *Preemptive RANSAC* (Nistér 2003). In another variant on RANSAC called PROSAC (PROgressive SAmple Consensus), random samples are initially added from the most "confident" matches, thereby speeding up the process of finding a (statistically) likely good set of inliers (Chum and Matas 2005). Raguram, Chum *et al.* (2012) provide a unified framework from which most of these techniques can be derived as well as a nice experimental comparison.

Additional variants on RANSAC include MLESAC (Torr and Zisserman 2000), DSAC (Brachmann, Krull *et al.* 2017), Graph-Cut RANSAC (Barath and Matas 2018), MAGSAC (Barath, Matas, and Noskova 2019), and ESAC (Brachmann and Rother 2019). Some of these algorithms, such as DSAC (Differentiable RANSAC), are designed to be differentiable so they can be used in end-to-end training of feature detection and matching pipelines (Section 7.1). The MAGSAC++ paper by Barath, Noskova *et al.* (2020) compares many of these variants. Yang, Antonante *et al.* (2020) claim that using a robust penalty function with a decreasing outlier parameter, i.e., *graduated non-convexity* (Blake and Zisserman 1987; Barron 2019), can outperform RANSAC in many geometric correspondence and pose estimation problems. To ensure that the random sampling has a good chance of finding a true set of inliers, a sufficient number of trials S must be evaluated. Let p be the probability that any given correspondence is valid and P be the probability of success after S trials. The likelihood in one trial that all k random samples are inliers is p^k. Therefore, the likelihood that

[8] For pixel-based alignment methods (Section 9.1.1), hierarchical (coarse-to-fine) techniques are often used to lock onto the *dominant motion* in a scene.

[9] For problems such as epipolar geometry estimation, the residual may be the distance between a point and a line.

k	p	S
3	0.5	35
6	0.6	97
6	0.5	293

Table 8.2 Number of trials S to attain a 99% probability of success (Stewart 1999).

S such trials will all fail is

$$1 - P = (1 - p^k)^S \tag{8.29}$$

and the required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}. \tag{8.30}$$

Stewart (1999) gives examples of the required number of trials S to attain a 99% probability of success. As you can see from Table 8.2, the number of trials grows quickly with the number of sample points used. This provides a strong incentive to use the *minimum* number of sample points k possible for any given trial, which is how RANSAC is normally used in practice.

Uncertainty modeling

In addition to robustly computing a good alignment, some applications require the computation of uncertainty (see Appendix B.6). For linear problems, this estimate can be obtained by inverting the Hessian matrix (8.9) and multiplying it by the feature position noise, if these have not already been used to weight the individual measurements, as in Equations (8.10) and (8.11). In statistics, the Hessian, which is the inverse covariance, is sometimes called the (Fisher) *information matrix* (Appendix B.1).

When the problem involves non-linear least squares, the inverse of the Hessian matrix provides the *Cramer–Rao lower bound* on the covariance matrix, i.e., it provides the *minimum* amount of covariance in a given solution, which can actually have a wider spread ("longer tails") if the energy flattens out away from the local minimum where the optimal solution is found.

8.1.5 3D alignment

Instead of aligning 2D sets of image features, many computer vision applications require the alignment of 3D points. In the case where the 3D transformations are linear in the motion parameters, e.g., for translation, similarity, and affine, regular least squares (8.5) can be used.

The case of rigid (Euclidean) motion,

$$E_{\mathrm{R3D}} = \sum_i \|\mathbf{x}'_i - \mathbf{R}\mathbf{x}_i - \mathbf{t}\|^2, \tag{8.31}$$

which arises more frequently and is often called the *absolute orientation* problem (Horn 1987), requires slightly different techniques. If only scalar weightings are being used (as opposed to full 3D per-point anisotropic covariance estimates), the weighted centroids of the two point clouds $\mathbf{c}$ and $\mathbf{c}'$ can be used to estimate the translation $\mathbf{t} = \mathbf{c}' - \mathbf{R}\mathbf{c}$.[10] We are then left with the problem of

[10] When full covariances are used, they are transformed by the rotation, so a closed-form solution for translation is not possible.

estimating the rotation between two sets of points $\{\hat{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{c}\}$ and $\{\hat{\mathbf{x}}'_i = \mathbf{x}'_i - \mathbf{c}'\}$ that are both centered at the origin.

One commonly used technique is called the *orthogonal Procrustes algorithm* (Golub and Van Loan 1996, p. 601) and involves computing the singular value decomposition (SVD) of the 3×3 correlation matrix

$$\mathbf{C} = \sum_i \hat{\mathbf{x}}' \hat{\mathbf{x}}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{8.32}$$

The rotation matrix is then obtained as $\mathbf{R} = \mathbf{U}\mathbf{V}^T$. (Verify this for yourself when $\hat{\mathbf{x}}' = \mathbf{R}\hat{\mathbf{x}}$.)

Another technique is the absolute orientation algorithm (Horn 1987) for estimating the unit quaternion corresponding to the rotation matrix $\mathbf{R}$, which involves forming a 4×4 matrix from the entries in $\mathbf{C}$ and then finding the eigenvector associated with its largest positive eigenvalue.

Lorusso, Eggert, and Fisher (1995) experimentally compare these two techniques to two additional techniques proposed in the literature, but find that the difference in accuracy is negligible (well below the effects of measurement noise).

In situations where these closed-form algorithms are not applicable, e.g., when full 3D covariances are being used or when the 3D alignment is part of some larger optimization, the incremental rotation update introduced in Section 2.1.3 (2.35–2.36), which is parameterized by an instantaneous rotation vector $\boldsymbol{\omega}$, can be used (See Section 8.2.3 for an application to image stitching.)

In some situations, e.g., when merging range data maps, the correspondence between data points is not known *a priori*. In this case, iterative algorithms that start by matching nearby points and then update the most likely correspondence can be used (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan *et al.* 1998; David, DeMenthon *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). These techniques are discussed in more detail in Section 13.2.1.

8.2 Image stitching

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision (Milgram 1975; Peleg 1981). Image stitching algorithms create the high-resolution photo-mosaics used to produce today's digital maps and satellite photos. They are also now a standard mode in smartphone cameras and can be used to create beautiful ultra wide-angle panoramas.

Image stitching originated in the photogrammetry community, where more manually intensive methods based on surveyed *ground control points* or manually registered *tie points* have long been used to register aerial photos into large-scale photo-mosaics (Slama 1980). One of the key advances in this community was the development of *bundle adjustment* algorithms (Section 11.4.2), which could simultaneously solve for the locations of all of the camera positions, thus yielding globally consistent solutions (Triggs, McLauchlan *et al.* 1999). Another recurring problem in creating photo-mosaics is the elimination of visible seams, for which a variety of techniques have been developed over the years (Milgram 1975, 1977; Peleg 1981; Davis 1998; Agarwala, Dontcheva *et al.* 2004)

In film photography, special cameras were developed in the 1990s to take ultra-wide-angle panoramas, often by exposing the film through a vertical slit as the camera rotated on its axis (Meehan 1990). In the mid-1990s, image alignment techniques started being applied to the construction of wide-angle seamless panoramas from regular hand-held cameras (Mann and Picard 1994; Chen 1995; Szeliski 1996). Subsequent algorithms addressed the need to compute globally consistent alignments (Szeliski and Shum 1997; Sawhney and Kumar 1999; Shum and Szeliski 2000), to remove "ghosts" due to parallax and object movement (Davis 1998; Shum and Szeliski 2000; Uyttendaele, Eden, and Szeliski 2001; Agarwala, Dontcheva *et al.* 2004), and to deal with varying

exposures (Mann and Picard 1994; Uyttendaele, Eden, and Szeliski 2001; Levin, Zomet *et al.* 2004; Eden, Uyttendaele, and Szeliski 2006; Kopf, Uyttendaele *et al.* 2007).[11]

While early techniques worked by directly minimizing pixel-to-pixel dissimilarities, today's algorithms extract a sparse set of features and match them to each other, as described in Chapter 7. Such feature-based approaches (Zoghlami, Faugeras, and Deriche 1997; Capel and Zisserman 1998; Cham and Cipolla 1998; Badra, Qumsieh, and Dudek 1998; McLauchlan and Jaenicke 2002; Brown and Lowe 2007) have the advantage of being more robust against scene movement and are usually faster,[12] Their biggest advantage, however, is the ability to "recognize panoramas", i.e., to automatically discover the adjacency (overlap) relationships among an unordered set of images, which makes them ideally suited for fully automated stitching of panoramas taken by casual users (Brown and Lowe 2007).

What, then, are the essential problems in image stitching? As with image alignment, we must first determine the appropriate mathematical model relating pixel coordinates in one image to pixel coordinates in another; Section 8.2.1 reviews the basic models we have studied and presents some new motion models related specifically to panoramic image stitching. Next, we must somehow estimate the correct alignments relating various pairs (or collections) of images. Chapter 7 discusses how distinctive features can be found in each image and then efficiently matched to rapidly establish correspondences between pairs of images. Chapter 9 discusses how direct pixel-to-pixel comparisons combined with gradient descent (and other optimization techniques) can also be used to estimate these parameters. When multiple images exist in a panorama, global optimization techniques can be used to compute a globally consistent set of alignments and to efficiently discover which images overlap one another. In Section 8.3, we look at how each of these previously developed techniques can be modified to take advantage of the imaging setups commonly used to create panoramas.

Once we have aligned the images, we must choose a final compositing surface for warping the aligned images (Section 8.4.1). We also need algorithms to seamlessly cut and blend overlapping images, even in the presence of parallax, lens distortion, scene motion, and exposure differences (Section 8.4.2–8.4.4).

8.2.1 Parametric motion models

Before we can register and align images, we need to establish the mathematical relationships that map pixel coordinates from one image to another. A variety of such *parametric motion models* are possible, from simple 2D transforms, to planar perspective models, 3D camera rotations, lens distortions, and mapping to non-planar (e.g., cylindrical) surfaces.

We already covered several of these models in Sections 2.1 and 8.1. In particular, we saw in Section 2.1.4 how the parametric motion describing the deformation of a planar surface as viewed from different positions can be described with an eight-parameter homography (2.71) (Mann and Picard 1994; Szeliski 1996). We also saw how a camera undergoing a pure rotation induces a different kind of homography (2.72).

In this section, we review both of these models and show how they can be applied to different stitching situations. We also introduce spherical and cylindrical compositing surfaces and show how, under favorable circumstances, they can be used to perform alignment using pure translations (Section 8.2.6). Deciding which alignment model is most appropriate for a given situation or set of

[11] A collection of some of these papers was compiled by Benosman and Kang (2001) and they are surveyed by Szeliski (2006a).

[12] See a discussion of the pros and cons of direct vs. feature-based techniques in (Triggs, Zisserman, and Szeliski 2000) and in the first edition of this book (Szeliski 2010, Section 8.3.4).

(a) translation [2 dof] (b) affine [6 dof] (c) perspective [8 dof] (d) 3D rotation [3+ dof]

Figure 8.4 Two-dimensional motion models and how they can be used for image stitching.

data is a *model selection* problem (Torr 2002; Bishop 2006; Robert 2007; Hastie, Tibshirani, and Friedman 2009; Murphy 2012), an important topic we do not cover in this book.

Planar perspective motion

The simplest possible motion model to use when aligning images is to simply translate and rotate them in 2D (Figure 8.4a). This is exactly the same kind of motion that you would use if you had overlapping photographic prints. It is also the kind of technique favored by David Hockney to create the collages that he calls *joiners* (Zelnik-Manor and Perona 2007; Nomura, Zhang, and Nayar 2007). Creating such collages, which show visible seams and inconsistencies that add to the artistic effect, is popular on websites such as Flickr, where they more commonly go under the name *panography* (Section 8.1.2). Translation and rotation are also usually adequate motion models to compensate for small camera motions in applications such as photo and video stabilization and merging (Exercise 8.1 and Section 9.2.1).

In Section 2.1.4, we saw how the mapping between two cameras viewing a common plane can be described using a 3×3 homography (2.71). Consider the matrix $\mathbf{M}_{10}$ that arises when mapping a pixel in one image to a 3D point and then back onto a second image,

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{P}}_1 \tilde{\mathbf{P}}_0^{-1} \tilde{\mathbf{x}}_0 = \mathbf{M}_{10} \tilde{\mathbf{x}}_0. \tag{8.33}$$

When the last row of the $\mathbf{P}_0$ matrix is replaced with a plane equation $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + c_0$ and points are assumed to lie on this plane, i.e., their disparity is $d_0 = 0$, we can ignore the last column of $\mathbf{M}_{10}$ and also its last row, since we do not care about the final z-buffer depth. The resulting homography matrix $\tilde{\mathbf{H}}_{10}$ (the upper left 3×3 sub-matrix of $\mathbf{M}_{10}$) describes the mapping between pixels in the two images,

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{H}}_{10} \tilde{\mathbf{x}}_0. \tag{8.34}$$

This observation formed the basis of some of the earliest automated image stitching algorithms (Mann and Picard 1994; Szeliski 1994, 1996). Because reliable feature matching techniques had not yet been developed, these algorithms used direct pixel value matching, i.e., direct parametric motion estimation, as described in Section 9.2 and Equations (8.19–8.20).

More recent stitching algorithms first extract features and then match them up, often using robust techniques such as RANSAC (Section 8.1.4) to compute a good set of inliers. The final computation of the homography (8.34), i.e., the solution of the least squares fitting problem given pairs of

corresponding features,

$$x_1 = \frac{(1+h_{00})x_0 + h_{01}y_0 + h_{02}}{h_{20}x_0 + h_{21}y_0 + 1} \quad \text{and} \tag{8.35}$$

$$y_1 = \frac{h_{10}x_0 + (1+h_{11})y_0 + h_{12}}{h_{20}x_0 + h_{21}y_0 + 1}, \tag{8.36}$$

uses iterative least squares, as described in Section 8.1.3 and Equations (8.21–8.23).

8.2.2 *Application*: Whiteboard and document scanning

The simplest image-stitching application is to stitch together a number of image scans taken on a flatbed scanner. Say you have a large map, or a piece of child's artwork, that is too large to fit on your scanner. Simply take multiple scans of the document, making sure to overlap the scans by a large enough amount to ensure that there are enough common features. Next, take successive pairs of images that you know overlap, extract features, match them up, and estimate the 2D rigid transform (2.16),

$$\mathbf{x}_{k+1} = \mathbf{R}_k\mathbf{x}_k + \mathbf{t}_k, \tag{8.37}$$

that best matches the features, using two-point RANSAC, if necessary, to find a good set of inliers. Then, on a final compositing surface (aligned with the first scan, for example), resample your images (Section 3.6.1) and average them together. Can you see any potential problems with this scheme?

One complication is that a 2D rigid transformation is non-linear in the rotation angle θ, so you will have to either use non-linear least squares or constrain $\mathbf{R}$ to be orthonormal, as described in Section 8.1.3.

A bigger problem lies in the pairwise alignment process. As you align more and more pairs, the solution may drift so that it is no longer globally consistent. In this case, a global optimization procedure, as described in Section 8.3, may be required. Such global optimization often requires a large system of non-linear equations to be solved, although in some cases, such as linearized homographies (Section 8.2.3) or similarity transforms (Section 8.1.2), regular least squares may be an option.

A slightly more complex scenario is when you take multiple overlapping handheld pictures of a whiteboard or other large planar object (He and Zhang 2005; Zhang and He 2007). Here, the natural motion model to use is a homography, although a more complex model that estimates the 3D rigid motion relative to the plane (plus the focal length, if unknown), could in principle be used.

8.2.3 Rotational panoramas

The most typical case for panoramic image stitching is when the camera undergoes a pure rotation. Think of standing at the rim of the Grand Canyon. Relative to the distant geometry in the scene, as you snap away, the camera is undergoing a pure rotation, which is equivalent to assuming that all points are very far from the camera, i.e., on the *plane at infinity* (Figure 8.5).[13] Setting $\mathbf{t}_0 = \mathbf{t}_1 = 0$, we get the simplified 3×3 homography

$$\tilde{\mathbf{H}}_{10} = \mathbf{K}_1\mathbf{R}_1\mathbf{R}_0^{-1}\mathbf{K}_0^{-1} = \mathbf{K}_1\mathbf{R}_{10}\mathbf{K}_0^{-1}, \tag{8.38}$$

[13] In a more general (e.g., indoor) scene, if we want to ensure that there is no *parallax* (visible relative movement between objects at different depths), we need to rotate the camera around the lens's front *no-parallax point* (Littlefield 2006). This can be achieved by using a specialized panoramic rotation head with a built-in translation stage (Houghton 2013) or by determining the front nodal point using observations of collinear points—see Debevec, Wenger *et al.* (2002) and Szeliski (2010, Figure 6.7).

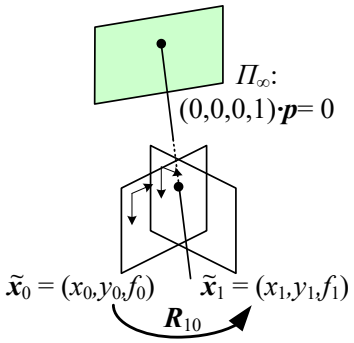


Figure 8.5 Pure 3D camera rotation. The form of the homography (mapping) is particularly simple and depends only on the 3D rotation matrix and focal lengths.

where $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$ is the simplified camera intrinsic matrix (2.59), assuming that $c_x = c_y = 0$, i.e., we are indexing the pixels starting from the image center (Szeliski 1996). This can also be re-written as

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_1 & & \\ & f_1 & \\ & & 1 \end{bmatrix} \mathbf{R}_{10} \begin{bmatrix} f_0^{-1} & & \\ & f_0^{-1} & \\ & & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{8.39}$$

or

$$\begin{bmatrix} x_1 \\ y_1 \\ f_1 \end{bmatrix} \sim \mathbf{R}_{10} \begin{bmatrix} x_0 \\ y_0 \\ f_0 \end{bmatrix}, \tag{8.40}$$

which reveals the simplicity of the mapping equations and makes all of the motion parameters explicit. Thus, instead of the general eight-parameter homography relating a pair of images, we get the three-, four-, or five-parameter *3D rotation* motion models corresponding to the cases where the focal length f is known, fixed, or variable (Szeliski and Shum 1997).[14] Estimating the 3D rotation matrix (and, optionally, focal length) associated with each image is intrinsically more stable than estimating a homography with a full eight degrees of freedom, which makes this the method of choice for large-scale image stitching algorithms (Szeliski and Shum 1997; Shum and Szeliski 2000; Brown and Lowe 2007).

Given this representation, how do we update the rotation matrices to best align two overlapping images? Given a current estimate for the homography $\tilde{\mathbf{H}}_{10}$ in (8.38), the best way to update $\mathbf{R}_{10}$ is to prepend an *incremental* rotation matrix $\mathbf{R}(\boldsymbol{\omega})$ to the current estimate $\mathbf{R}_{10}$ (Szeliski and Shum 1997; Shum and Szeliski 2000),

$$\tilde{\mathbf{H}}(\boldsymbol{\omega}) = \mathbf{K}_1 \mathbf{R}(\boldsymbol{\omega}) \mathbf{R}_{10} \mathbf{K}_0^{-1} = [\mathbf{K}_1 \mathbf{R}(\boldsymbol{\omega}) \mathbf{K}_1^{-1}][\mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1}] = \mathbf{D}\tilde{\mathbf{H}}_{10}. \tag{8.41}$$

Note that here we have written the update rule in the *compositional* form, where the incremental update $\mathbf{D}$ is *prepended* to the current homography $\tilde{\mathbf{H}}_{10}$. Using the small-angle approximation to $\mathbf{R}(\boldsymbol{\omega})$ given in (2.35), we can write the incremental update matrix as

$$\mathbf{D} = \mathbf{K}_1 \mathbf{R}(\boldsymbol{\omega}) \mathbf{K}_1^{-1} \approx \mathbf{K}_1 (\mathbf{I} + [\boldsymbol{\omega}]_\times) \mathbf{K}_1^{-1} = \begin{bmatrix} 1 & -\omega_z & f_1 \omega_y \\ \omega_z & 1 & -f_1 \omega_x \\ -\omega_y / f_1 & \omega_x / f_1 & 1 \end{bmatrix}. \tag{8.42}$$

[14] An initial estimate of the focal lengths can be obtained using the intrinsic calibration techniques described in Section 11.1.3 or from EXIF tags.

Notice how there is now a nice one-to-one correspondence between the entries in the $\mathbf{D}$ matrix and the $h_{00}, \ldots, h_{21}$ parameters used in Table 8.1 and Equation (8.19), i.e.,

$$(h_{00}, h_{01}, h_{02}, h_{00}, h_{11}, h_{12}, h_{20}, h_{21}) = (0, -\omega_z, f_1\omega_y, \omega_z, 0, -f_1\omega_x, -\omega_y/f_1, \omega_x/f_1). \quad (8.43)$$

We can therefore apply the chain rule to Equations (8.24 and 8.43) to obtain

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} -xy/f_1 & f_1 + x^2/f_1 & -y \\ -(f_1 + y^2/f_1) & xy/f_1 & x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad (8.44)$$

which give us the linearized update equations needed to estimate $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$.[15] Notice that this update rule depends on the focal length f_1 of the *target* view and is independent of the focal length f_0 of the *template* view. This is because the compositional algorithm essentially makes small perturbations to the target. Once the incremental rotation vector $\boldsymbol{\omega}$ has been computed, the $\mathbf{R}_1$ rotation matrix can be updated using $\mathbf{R}_1 \leftarrow \mathbf{R}(\boldsymbol{\omega})\mathbf{R}_1$.

The formulas for updating the focal length estimates are a little more involved and are given in Shum and Szeliski (2000). We will not repeat them here, since an alternative update rule, based on minimizing the difference between back-projected 3D rays, is given in Section 8.3.1. Figure 8.1a shows the alignment of four images under the 3D rotation motion model.

8.2.4 Gap closing

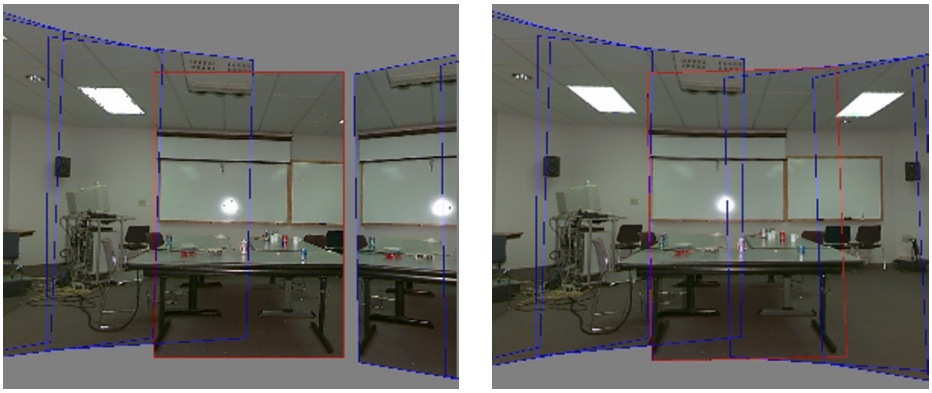
The techniques presented in this section can be used to estimate a series of rotation matrices and focal lengths, which can be chained together to create large panoramas. Unfortunately, because of accumulated errors, this approach will rarely produce a closed 360° panorama. Instead, there will invariably be either a gap or an overlap (Figure 8.6).

We can solve this problem by matching the first image in the sequence with the last one. The difference between the two rotation matrix estimates associated with the repeated first image indicates the amount of misregistration. This error can be distributed evenly across the whole sequence by taking the quotient of the two quaternions associated with these rotations and dividing this "error quaternion" by the number of images in the sequence (Szeliski and Shum 1997). We can also update the estimated focal length based on the amount of misregistration. To do this, we first convert the error quaternion into a *gap angle*, θ_g and then update the focal length using the equation $f' = f(1 - \theta_g/360°)$.

Figure 8.6a 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 ($f = 510$) was used. Figure 8.6b shows the registration after closing the gap with the correct focal length ($f = 468$). Notice that both mosaics show very little visual misregistration (except at the gap), yet Figure 8.6a has been computed using a focal length that has 9% error. Related approaches have been developed by Hartley (1994b), McMillan and Bishop (1995), Stein (1995), and Kang and Weiss (1997) to solve the focal length estimation problem using pure panning motion and cylindrical images.

Unfortunately, this gap-closing heuristic only works for the kind of "one-dimensional" panorama where the camera is continuously turning in the same direction. In Section 8.3, we describe a different approach to removing gaps and overlaps that works for arbitrary camera motions.

[15] This is the same as the rotational component of instantaneous rigid flow (Bergen, Anandan *et al.* 1992) and the update equations given by Szeliski and Shum (1997) and Shum and Szeliski (2000).



(a) (b)

Figure 8.6 Gap closing (Szeliski and Shum 1997) © 1997 ACM: (a) A gap is visible when the focal length is wrong ($f = 510$). (b) No gap is visible for the correct focal length ($f = 468$).

8.2.5 *Application*: Video summarization and compression

An interesting application of image stitching is the ability to summarize and compress videos taken with a panning camera. This application was first suggested by Teodosio and Bender (1993), who called their mosaic-based summaries *salient stills*. These ideas were then extended by Irani, Hsu, and Anandan (1995) and Irani and Anandan (1998) to additional applications, such as video compression and video indexing. While these early approaches used affine motion models and were therefore restricted to long focal lengths, the techniques were generalized by Lee, Chen *et al.* (1997) to full eight-parameter homographies and incorporated into the MPEG-4 video compression standard, where the stitched background layers were called *video sprites* (Figure 8.7).

While video stitching is in many ways a straightforward generalization of multiple-image stitching (Steedly, Pal, and Szeliski 2005; Baudisch, Tan *et al.* 2006), the potential presence of large amounts of independent motion, camera zoom, and the desire to visualize dynamic events impose additional challenges. For example, moving foreground objects can often be removed using *median filtering*. Alternatively, foreground objects can be extracted into a separate layer (Sawhney and Ayer 1996) and later composited back into the stitched panoramas, sometimes as multiple instances to give the impressions of a "Chronophotograph" (Massey and Bender 1996) and sometimes as video overlays (Irani and Anandan 1998). Videos can also be used to create animated *panoramic video textures* (Section 14.5.2), in which different portions of a panoramic scene are animated with independently moving video loops (Agarwala, Zheng *et al.* 2005; Rav-Acha, Pritch *et al.* 2005; Joshi, Mehta *et al.* 2012; Yan, Liu, and Furukawa 2017; He, Liao *et al.* 2017; Oh, Joo *et al.* 2017), or to shine "video flashlights" onto a composite mosaic of a scene (Sawhney, Arpa *et al.* 2002).

Video can also provide an interesting source of content for creating panoramas taken from moving cameras. While this invalidates the usual assumption of a single point of view (optical center), interesting results can still be obtained. For example, the VideoBrush system of Sawhney, Kumar *et al.* (1998) uses thin strips taken from the center of the image to create a panorama taken from a horizontally moving camera. This idea can be generalized to other camera motions and compositing surfaces using the concept of mosaics on an adaptive manifold (Peleg, Rousso *et al.* 2000), and also used to generate panoramic stereograms (Ishiguro, Yamamoto, and Tsuji 1992; Peleg, Ben-Ezra, and Pritch 2001).[16] Related ideas have been used to create panoramic matte paintings for multi-

[16] A similar technique was likely used in the Google Cardboard Camera, https://blog.google/products/google-vr/

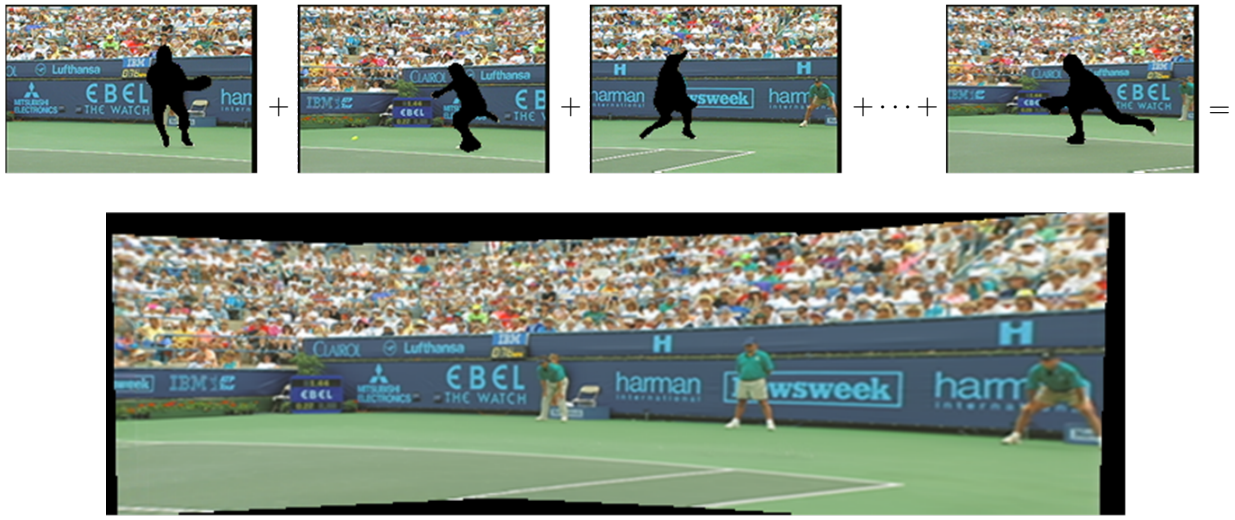


Figure 8.7 Video stitching the background scene to create a single *sprite* image that can be transmitted and used to re-create the background in each frame (Lee, Chen *et al.* 1997) © 1997 IEEE.

plane cel animation (Wood, Finkelstein *et al.* 1997), for creating stitched images of scenes with parallax (Kumar, Anandan *et al.* 1995), and as 3D representations of more complex scenes using *multiple-center-of-projection images* (Rademacher and Bishop 1998) and *multi-perspective panoramas* (Román, Garg, and Levoy 2004; Román and Lensch 2006; Agarwala, Agrawala *et al.* 2006; Kopf, Chen *et al.* 2010).

Another interesting variant on video-based panoramas is *concentric mosaics* (Section 14.3.3) (Shum and He 1999). Here, rather than trying to produce a single panoramic image, the complete original video is kept and used to re-synthesize views (from different camera origins) using ray remapping (light field rendering), thus endowing the panorama with a sense of 3D depth. The same dataset can also be used to explicitly reconstruct the depth using multi-baseline stereo (Ishiguro, Yamamoto, and Tsuji 1992; Peleg, Ben-Ezra, and Pritch 2001; Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007).

8.2.6 Cylindrical and spherical coordinates

An alternative to using homographies or 3D motions to align images is to first warp the images into *cylindrical* coordinates and then use a pure translational model to align them (Chen 1995; Szeliski 1996). Unfortunately, this only works if the images are all taken with a level camera or with a known tilt angle.

Assume for now that the camera is in its canonical position, i.e., its rotation matrix is the identity, $\mathbf{R} = \mathbf{I}$, so that the optical axis is aligned with the z-axis and the y-axis is aligned vertically. The 3D ray corresponding to an (x, y) pixel is therefore (x, y, f).

We wish to project this image onto a *cylindrical surface* of unit radius (Szeliski 1996). Points on this surface are parameterized by an angle θ and a height h, with the 3D cylindrical coordinates corresponding to (θ, h) given by

$$(\sin\theta, h, \cos\theta) \propto (x, y, f), \tag{8.45}$$

cardboard-camera-ios.

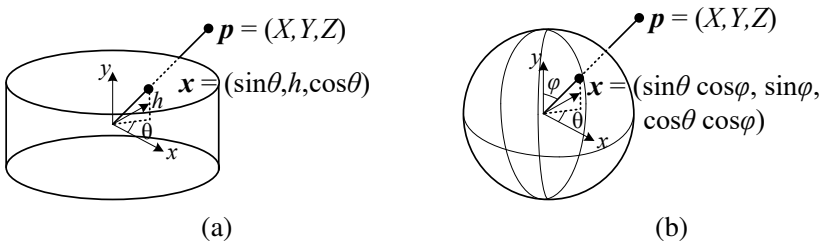


Figure 8.8 Projection from 3D to (a) cylindrical and (b) spherical coordinates.

as shown in Figure 8.8a. From this correspondence, we can compute the formula for the *warped* or *mapped* coordinates (Szeliski and Shum 1997),

$$x' = s\theta = s \tan^{-1} \frac{x}{f}, \tag{8.46}$$

$$y' = sh = s \frac{y}{\sqrt{x^2 + f^2}}, \tag{8.47}$$

where s is an arbitrary scaling factor (sometimes called the *radius* of the cylinder) that can be set to $s = f$ to minimize the distortion (scaling) near the center of the image.[17] The inverse of this mapping equation is given by

$$x = f \tan\theta = f \tan \frac{x'}{s}, \tag{8.48}$$

$$y = h\sqrt{x^2 + f^2} = \frac{y'}{s} f \sqrt{1 + \tan^2 x'/s} = f \frac{y'}{s} \sec \frac{x'}{s}. \tag{8.49}$$

Images can also be projected onto a *spherical surface* (Szeliski and Shum 1997), which is useful if the final panorama includes a full sphere or hemisphere of views, instead of just a cylindrical strip. In this case, the sphere is parameterized by two angles (θ, ϕ), with 3D spherical coordinates given by

$$(\sin\theta \cos\phi, \sin\phi, \cos\theta \cos\phi) \propto (x, y, f), \tag{8.50}$$

as shown in Figure 8.8b.[18] The correspondence between coordinates is now given by (Szeliski and Shum 1997):

$$x' = s\theta = s \tan^{-1} \frac{x}{f}, \tag{8.51}$$

$$y' = s\phi = s \tan^{-1} \frac{y}{\sqrt{x^2 + f^2}}, \tag{8.52}$$

while the inverse is given by

$$x = f \tan\theta = f \tan \frac{x'}{s}, \tag{8.53}$$

$$y = \sqrt{x^2 + f^2} \tan\phi = \tan \frac{y'}{s} f \sqrt{1 + \tan^2 x'/s} = f \tan \frac{y'}{s} \sec \frac{x'}{s}. \tag{8.54}$$

[17] The scale can also be set to a larger or smaller value for the final compositing surface, depending on the desired output panorama resolution—see Section 8.4.

[18] Note that these are not the usual spherical coordinates, first presented in Equation (2.8). Here, the y-axis points at the north pole instead of the z-axis, since we are used to viewing images taken horizontally, i.e., with the y-axis pointing in the direction of the gravity vector.

(a) (b)

Figure 8.9 A cylindrical panorama (Szeliski and Shum 1997) © 1997 ACM: (a) two cylindrically warped images related by a horizontal translation; (b) part of a cylindrical panorama composited from a sequence of images.

Note that it may be simpler to generate a scaled (x, y, z) direction from Equation (8.50) followed by a perspective division by z and a scaling by f.

Cylindrical image stitching algorithms are most commonly used when the camera is known to be level and only rotating around its vertical axis (Chen 1995). Under these conditions, images at different rotations are related by a pure horizontal translation.[19] This makes it attractive as an initial class project in an introductory computer vision course, since the full complexity of the perspective alignment algorithm (Sections 8.1, 9.2, and 8.2.3) can be avoided. Figure 8.9 shows how two cylindrically warped images from a leveled rotational panorama are related by a pure translation (Szeliski and Shum 1997).

Professional panoramic photographers often use pan-tilt heads that make it easy to control the tilt and to stop at specific *detents* in the rotation angle. Motorized rotation heads are also sometimes used for the acquisition of larger panoramas (Kopf, Uyttendaele *et al.* 2007).[20] Not only do they ensure a uniform coverage of the visual field with a desired amount of image overlap but they also make it possible to stitch the images using cylindrical or spherical coordinates and pure translations. In this case, pixel coordinates (x, y, f) must first be rotated using the known tilt and panning angles before being projected into cylindrical or spherical coordinates (Chen 1995). Having a roughly known panning angle also makes it easier to compute the alignment, as the rough relative positioning of all the input images is known ahead of time, enabling a reduced search range for alignment. Figure 8.1b shows a full 3D rotational panorama unwrapped onto the surface of a sphere (Szeliski and Shum 1997).

One final coordinate mapping worth mentioning is the *polar* mapping, where the north pole lies along the optical axis rather than the vertical axis,

$$(\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\phi) = s\,(x, y, z). \tag{8.55}$$

In this case, the mapping equations become

$$x' = s\phi\cos\theta = s\frac{x}{r}\tan^{-1}\frac{r}{z}, \tag{8.56}$$

$$y' = s\phi\sin\theta = s\frac{y}{r}\tan^{-1}\frac{r}{z}, \tag{8.57}$$

[19] Small vertical tilts can sometimes be compensated for with vertical translations.

[20] See also https://gigapan.org.

where $r = \sqrt{x^2 + y^2}$ is the *radial distance* in the (x, y) plane and $s\phi$ plays a similar role in the (x', y') plane. This mapping provides an attractive visualization surface for certain kinds of wide-angle panoramas and is also a good model for the distortion induced by *fisheye lenses*, as discussed in Section 2.1.5. Note how for small values of (x, y), the mapping equations reduce to $x' \approx sx/z$, which suggests that s plays a role similar to the focal length f.

8.3 Global alignment

So far, we have discussed how to register pairs of images using a variety of motion models. In most applications, we are given more than a single pair of images to register. The goal is then to find a globally consistent set of alignment parameters that minimize the misregistration between all pairs of images (Szeliski and Shum 1997; Shum and Szeliski 2000; Sawhney and Kumar 1999; Coorg and Teller 2000).

In this section, we extend the pairwise matching criteria (8.2, 9.1, and 9.43) to a global energy function that involves all of the per-image pose parameters (Section 8.3.1). Once we have computed the global alignment, we often need to perform *local adjustments*, such as *parallax removal*, to reduce double images and blurring due to local misregistrations (Section 8.3.2). Finally, if we are given an unordered set of images to register, we need to discover which images go together to form one or more panoramas. This process of *panorama recognition* is described in Section 8.3.3.

8.3.1 Bundle adjustment

One way to register a large number of images is to add new images to the panorama one at a time, aligning the most recent image with the previous ones already in the collection (Szeliski and Shum 1997) and discovering, if necessary, which images it overlaps (Sawhney and Kumar 1999). In the case of 360° panoramas, accumulated error may lead to the presence of a gap (or excessive overlap) between the two ends of the panorama, which can be fixed by stretching the alignment of all the images using a process called *gap closing* (Section 8.2.4). However, a better alternative is to simultaneously align all the images using a least-squares framework to correctly distribute any misregistration errors.

The process of simultaneously adjusting pose parameters and 3D point locations for a large collection of overlapping images is called *bundle adjustment* in the photogrammetry community (Triggs, McLauchlan *et al.* 1999). In computer vision, it was first applied to the general structure from motion problem (Szeliski and Kang 1994) and then later specialized for panoramic image stitching (Shum and Szeliski 2000; Sawhney and Kumar 1999; Coorg and Teller 2000).

In this section, we formulate the problem of global alignment using a feature-based approach, since this results in a simpler system. An equivalent direct approach can be obtained either by dividing images into patches and creating a virtual feature correspondence for each one (Shum and Szeliski 2000) or by replacing the per-feature error metrics with per-pixel metrics (Irani and Anandan 1999).

Before we describe this in more details, we should mention that a simpler, although less accurate, approach is to compute pairwise rotation estimates between overlapping images, and to then use a *rotation averaging* approach to estimate a global rotation for each camera (Hartley, Trumpf *et al.* 2013). However, since the measurement errors in each feature point location are not being counted correctly, as is the case in bundle adjustment, the solution will not have the same theoretical optimality.

Consider the feature-based alignment problem given in Equation (8.2), i.e.,

$$E_{\text{pairwise-LS}} = \sum_i \|\mathbf{r}_i\|^2 = \|\tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i\|^2. \tag{8.58}$$

For multi-image alignment, instead of having a single collection of pairwise feature correspondences, $\{(\mathbf{x}_i, \hat{\mathbf{x}}'_i)\}$, we have a collection of n features, with the location of the ith feature point in the jth image denoted by $\mathbf{x}_{ij}$ and its scalar confidence (i.e., inverse variance) denoted by c_{ij}.[21] Each image also has some associated pose parameters.

In this section, we assume that this pose consists of a rotation matrix $\mathbf{R}_j$ and a focal length f_j, although formulations in terms of homographies are also possible (Szeliski and Shum 1997; Sawhney and Kumar 1999). The equation mapping a 3D point $\mathbf{x}_i$ into a point $\mathbf{x}_{ij}$ in frame j can be re-written from Equations (2.68) and (8.38) as

$$\tilde{\mathbf{x}}_{ij} \sim \mathbf{K}_j \mathbf{R}_j \mathbf{x}_i \quad \text{and} \quad \mathbf{x}_i \sim \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_{ij}, \tag{8.59}$$

where $\mathbf{K}_j = \text{diag}(f_j, f_j, 1)$ is the simplified form of the calibration matrix. The motion mapping a point $\mathbf{x}_{ij}$ from frame j into a point $\mathbf{x}_{ik}$ in frame k is similarly given by

$$\tilde{\mathbf{x}}_{ik} \sim \tilde{\mathbf{H}}_{kj} \tilde{\mathbf{x}}_{ij} = \mathbf{K}_k \mathbf{R}_k \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_{ij}. \tag{8.60}$$

Given an initial set of $\{(\mathbf{R}_j, f_j)\}$ estimates obtained from chaining pairwise alignments, how do we refine these estimates?

One approach is to directly extend the pairwise energy $E_{\text{pairwise-LS}}$ (8.58) to a multiview formulation,

$$E_{\text{all-pairs-2D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_{ik}(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j, \mathbf{R}_k, f_k) - \hat{\mathbf{x}}_{ik}\|^2, \tag{8.61}$$

where the $\tilde{\mathbf{x}}_{ik}$ function is the *predicted* location of feature i in frame k given by (8.60), $\hat{\mathbf{x}}_{ij}$ is the *observed* location, and the "2D" in the subscript indicates that an image-plane error is being minimized (Shum and Szeliski 2000). Note that since $\tilde{\mathbf{x}}_{ik}$ depends on the $\hat{\mathbf{x}}_{ij}$ observed value, we actually have an *errors-in-variable* problem, which in principle requires more sophisticated techniques than least squares to solve (Van Huffel and Lemmerling 2002; Matei and Meer 2006). However, in practice, if we have enough features, we can directly minimize the above quantity using regular non-linear least squares and obtain an accurate multi-frame alignment.

While this approach works pretty well, it suffers from two potential disadvantages. First, because a summation is taken over all pairs with corresponding features, features that are observed many times are overweighted in the final solution. (In effect, a feature observed m times gets counted $\binom{m}{2}$ times instead of m times.) Second, the derivatives of $\tilde{\mathbf{x}}_{ik}$ with respect to the $\{(\mathbf{R}_j, f_j)\}$ are a little cumbersome, although using the incremental correction to $\mathbf{R}_j$ introduced in Section 8.2.3 makes this more tractable.

An alternative way to formulate the optimization is to use true bundle adjustment, i.e., to solve not only for the pose parameters $\{(\mathbf{R}_j, f_j)\}$ but also for the 3D point positions $\{\mathbf{x}_i\}$,

$$E_{\text{BA-2D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j) - \hat{\mathbf{x}}_{ij}\|^2, \tag{8.62}$$

where $\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j)$ is given by (8.59). The disadvantage of full bundle adjustment is that there are more variables to solve for, so each iteration and also the overall convergence may be slower.

[21] Features that are not seen in image j have $c_{ij} = 0$. We can also use 2×2 inverse covariance matrices Σ_{ij}^{-1} in place of c_{ij}, as shown in Equation (8.11).

(Imagine how the 3D points need to "shift" each time some rotation matrices are updated.) However, the computational complexity of each linearized Gauss–Newton step can be reduced using sparse matrix techniques (Section 11.4.3) (Szeliski and Kang 1994; Triggs, McLauchlan *et al.* 1999; Hartley and Zisserman 2004).

An alternative formulation is to minimize the error in 3D projected ray directions (Shum and Szeliski 2000), i.e.,

$$E_{\mathrm{BA-3D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \mathbf{x}_i\|^2, \tag{8.63}$$

where $\tilde{\mathbf{x}}_i(\mathbf{x}_{ij}; \mathbf{R}_j, f_j)$ is given by the second half of (8.59). This has no particular advantage over (8.62). In fact, since errors are being minimized in 3D ray space, there is a bias towards estimating longer focal lengths, since the angles between rays become smaller as f increases.

However, if we eliminate the 3D rays $\mathbf{x}_i$, we can derive a pairwise energy formulated in 3D ray space (Shum and Szeliski 2000),

$$E_{\mathrm{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ik}; \mathbf{R}_k, f_k)\|^2. \tag{8.64}$$

This results in the simplest set of update equations (Shum and Szeliski 2000), since the f_k can be folded into the creation of the homogeneous coordinate vector as in Equation (8.40). Thus, even though this formula over-weights features that occur more frequently, it is the method used by Shum and Szeliski (2000) and Brown, Szeliski, and Winder (2005). To reduce the bias towards longer focal lengths, we multiply each residual (3D error) by $\sqrt{f_j f_k}$, which is similar to projecting the 3D rays into a "virtual camera" of intermediate focal length.

Up vector selection. As mentioned above, there exists a global ambiguity in the pose of the 3D cameras computed by the above methods. While this may not appear to matter, people prefer that the final stitched image is "upright" rather than twisted or tilted. More concretely, people are used to seeing photographs displayed so that the vertical (gravity) axis points straight up in the image. Consider how you usually shoot photographs: while you may pan and tilt the camera any which way, you usually keep the horizontal edge of your camera (its x-axis) parallel to the ground plane (perpendicular to the world gravity direction).

Mathematically, this constraint on the rotation matrices can be expressed as follows. Recall from Equation (8.59) that the 3D to 2D projection is given by

$$\tilde{\mathbf{x}}_{ik} \sim \mathbf{K}_k \mathbf{R}_k \mathbf{x}_i. \tag{8.65}$$

We wish to post-multiply each rotation matrix $\mathbf{R}_k$ by a global rotation $\mathbf{R}_{\mathrm{G}}$ such that the projection of the global y-axis, $\hat{\jmath} = (0, 1, 0)$ is perpendicular to the image x-axis, $\hat{\imath} = (1, 0, 0)$.[22]

This constraint can be written as

$$\hat{\imath}^T \mathbf{R}_k \mathbf{R}_{\mathrm{G}} \hat{\jmath} = 0 \tag{8.66}$$

(note that the scaling by the calibration matrix is irrelevant here). This is equivalent to requiring that the first row of $\mathbf{R}_k$, $\mathbf{r}_{k0} = \hat{\imath}^T \mathbf{R}_k$ be perpendicular to the second column of $\mathbf{R}_{\mathrm{G}}$, $\mathbf{r}_{\mathrm{G}1} = \mathbf{R}_{\mathrm{G}} \hat{\jmath}$. This set of constraints (one per input image) can be written as a least squares problem,

$$\mathbf{r}_{\mathrm{G}1} = \arg\min_{\mathbf{r}} \sum_k (\mathbf{r}^T \mathbf{r}_{k0})^2 = \arg\min_{\mathbf{r}} \mathbf{r}^T \left[\sum_k \mathbf{r}_{k0} \mathbf{r}_{k0}^T \right] \mathbf{r}. \tag{8.67}$$

[22] Note that here we use the convention common in computer graphics that the vertical world axis corresponds to y. This is a natural choice if we wish the rotation matrix associated with a "regular" image taken horizontally to be the identity, rather than a 90° rotation around the x-axis.

Thus, $\mathbf{r}_{G1}$ is the smallest eigenvector of the *scatter* or *moment* matrix spanned by the individual camera rotation x-vectors, which should generally be of the form $(c, 0, s)$ when the cameras are upright.

To fully specify the $\mathbf{R}_G$ global rotation, we need to specify one additional constraint. This is related to the *view selection* problem discussed in Section 8.4.1. One simple heuristic is to prefer the average z-axis of the individual rotation matrices, $\overline{\mathbf{k}} = \sum_k \hat{\mathbf{k}}^T \mathbf{R}_k$ to be close to the world z-axis, $\mathbf{r}_{G2} = \mathbf{R}_G \hat{\mathbf{k}}$. We can therefore compute the full rotation matrix $\mathbf{R}_G$ in three steps:

1. $\mathbf{r}_{G1} = \min$ eigenvector $(\sum_k \mathbf{r}_{k0}\mathbf{r}_{k0}^T)$;
2. $\mathbf{r}_{G0} = \mathcal{N}((\sum_k \mathbf{r}_{k2}) \times \mathbf{r}_{G1})$;
3. $\mathbf{r}_{G2} = \mathbf{r}_{G0} \times \mathbf{r}_{G1}$,

where $\mathcal{N}(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|$ normalizes a vector $\mathbf{v}$.

8.3.2 Parallax removal

Once we have optimized the global orientations and focal lengths of our cameras, we may find that the images are still not perfectly aligned, i.e., the resulting stitched image looks blurry or ghosted in some places. This can be caused by a variety of factors, including unmodeled radial distortion, 3D parallax (failure to rotate the camera around its front nodal point), small scene motions such as waving tree branches, and large-scale scene motions such as people moving in and out of pictures.

Each of these problems can be treated with a different approach. Radial distortion can be estimated (potentially ahead of time) using one of the techniques discussed in Section 2.1.5. For example, the *plumb-line method* (Brown 1971; Kang 2001; El-Melegy and Farag 2003) adjusts radial distortion parameters until slightly curved lines become straight, while mosaic-based approaches adjust them until misregistration is reduced in image overlap areas (Stein 1997; Sawhney and Kumar 1999).

3D parallax can be handled by doing a full 3D bundle adjustment, i.e., by replacing the projection Equation (8.59) used in Equation (8.62) with Equation (2.68), which models camera translations. The 3D positions of the matched feature points and cameras can then be simultaneously recovered, although this can be significantly more expensive than parallax-free image registration. Once the 3D structure has been recovered, the scene could (in theory) be projected to a single (central) viewpoint that contains no parallax. However, to do this, dense *stereo* correspondence needs to be performed (Section 12.3) (Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007), which may not be possible if the images contain only partial overlap. In that case, it may be necessary to correct for parallax only in the overlap areas, which can be accomplished using a *multi-perspective plane sweep* (MPPS) algorithm (Kang, Szeliski, and Uyttendaele 2004; Uyttendaele, Criminisi *et al.* 2004).

When the motion in the scene is very large, i.e., when objects appear and disappear completely, a sensible solution is to simply *select* pixels from only one image at a time as the source for the final composite (Milgram 1977; Davis 1998; Agarwala, Dontcheva *et al.* 2004), as discussed in Section 8.4.2. However, when the motion is reasonably small (on the order of a few pixels), general 2D motion estimation (optical flow) can be used to perform an appropriate correction before blending using a process called *local alignment* (Shum and Szeliski 2000; Kang, Uyttendaele *et al.* 2003). This same process can also be used to compensate for radial distortion and 3D parallax, although it uses a weaker motion model than explicitly modeling the source of error and may, therefore, fail more often or introduce unwanted distortions.

(a) (b) (c)

Figure 8.10 Deghosting a mosaic with motion parallax (Shum and Szeliski 2000) © 2000 IEEE: (a) composite with parallax; (b) after a single deghosting step (patch size 32); (c) after multiple steps (sizes 32, 16 and 8).

The local alignment technique introduced by Shum and Szeliski (2000) starts with the global bundle adjustment (8.64) used to optimize the camera poses. Once these have been estimated, the *desired* location of a 3D point $\mathbf{x}_i$ can be estimated as the *average* of the back-projected 3D locations,

$$\bar{\mathbf{x}}_i \sim \sum_j c_{ij} \tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) \Big/ \sum_j c_{ij} \ , \tag{8.68}$$

which can be projected into each image j to obtain a *target location* $\bar{\mathbf{x}}_{ij}$. The difference between the target locations $\bar{\mathbf{x}}_{ij}$ and the original features $\mathbf{x}_{ij}$ provide a set of local motion estimates

$$\mathbf{u}_{ij} = \bar{\mathbf{x}}_{ij} - \mathbf{x}_{ij}, \tag{8.69}$$

which can be interpolated to form a dense correction field $\mathbf{u}_j(\mathbf{x}_j)$. In their system, Shum and Szeliski (2000) use an *inverse warping* algorithm where the sparse $-\mathbf{u}_{ij}$ values are placed at the new target locations $\bar{\mathbf{x}}_{ij}$, interpolated using bilinear kernel functions (Nielson 1993) and then added to the original pixel coordinates when computing the warped (corrected) image. To get a reasonably dense set of features to interpolate, Shum and Szeliski (2000) place a feature point at the center of each patch (the patch size controls the smoothness in the local alignment stage), rather than relying on features extracted using an interest operator (Figure 8.10).

An alternative approach to motion-based deghosting was proposed by Kang, Uyttendaele *et al.* (2003), who estimate dense optical flow between each input image and a central *reference* image. The accuracy of the flow vector is checked using a photo-consistency measure before a given warped pixel is considered valid and is used to compute a high dynamic range radiance estimate, which is the goal of their overall algorithm. The requirement for a reference image makes their approach less applicable to general image mosaicing, although an extension to this case could certainly be envisaged.

The idea of combining *global* parametric warps with *local* mesh-based warps or multiple motion models to compensate for parallax has been refined in a number of more recent papers (Zaragoza, Chin *et al.* 2013; Zhang and Liu 2014; Lin, Pankanti *et al.* 2015; Lin, Jiang *et al.* 2016; Herrmann, Wang *et al.* 2018b; Lee and Sim 2020). Some of these papers use *content-preserving warps* (Liu, Gleicher *et al.* 2009) for their local deformations, while others include a rolling shutter model (Zhuang and Tran 2020).

8.3.3 Recognizing panoramas

The final piece needed to perform fully automated image stitching is a technique to recognize which images actually go together, which Brown and Lowe (2007) call *recognizing panoramas*. If the user

takes images in sequence so that each image overlaps its predecessor and also specifies the first and last images to be stitched, bundle adjustment combined with the process of *topology inference* can be used to automatically assemble a panorama (Sawhney and Kumar 1999). However, users often jump around when taking panoramas, e.g., they may start a new row on top of a previous one, jump back to take a repeat shot, or create 360° panoramas where end-to-end overlaps need to be discovered. Furthermore, the ability to discover multiple panoramas taken by a user over an extended period of time can be a big convenience.

To recognize panoramas, Brown and Lowe (2007) first find all pairwise image overlaps using a feature-based method and then find connected components in the overlap graph to "recognize" individual panoramas (Figure 8.11). The feature-based matching stage first extracts scale invariant feature transform (SIFT) feature locations and feature descriptors (Lowe 2004) from all the input images and places them in an indexing structure, as described in Section 7.1.3. For each image pair under consideration, the nearest matching neighbor is found for each feature in the first image, using the indexing structure to rapidly find candidates and then comparing feature descriptors to find the best match. RANSAC is used to find a set of *inlier* matches; pairs of matches are used to hypothesize similarity motion models that are then used to count the number of inliers. A RANSAC algorithm tailored specifically for rotational panoramas is described by Brown, Hartley, and Nistér (2007).

In practice, the most difficult part of getting a fully automated stitching algorithm to work is deciding which pairs of images actually correspond to the same parts of the scene. Repeated structures such as windows (Figure 8.12) can lead to false matches when using a feature-based approach. One way to mitigate this problem is to perform a direct pixel-based comparison between the registered images to determine if they actually are different views of the same scene. Unfortunately, this heuristic may fail if there are moving objects in the scene (Figure 8.13). While there is no magic bullet for this problem, short of full scene understanding, further improvements can likely be made by applying domain-specific heuristics, such as priors on typical camera motions as well as machine learning techniques applied to the problem of match validation.

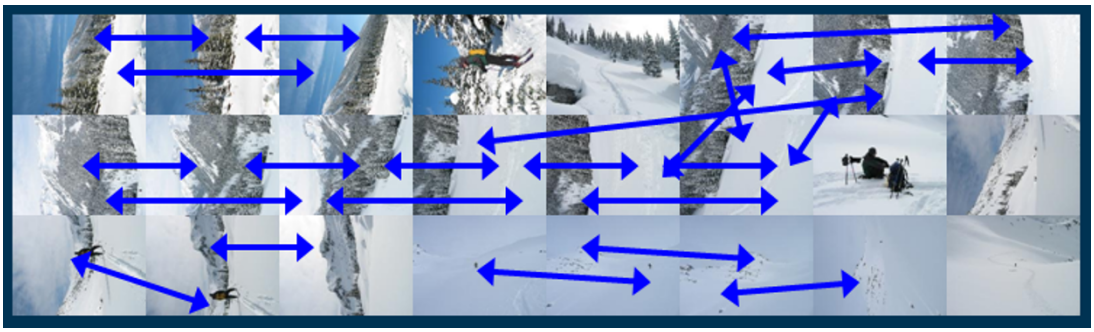
8.4 Compositing

Once we have registered all of the input images with respect to each other, we need to decide how to produce the final stitched mosaic image. This involves selecting a final compositing surface (flat, cylindrical, spherical, etc.) and view (reference image). It also involves selecting which pixels contribute to the final composite and how to optimally blend these pixels to minimize visible seams, blur, and ghosting.

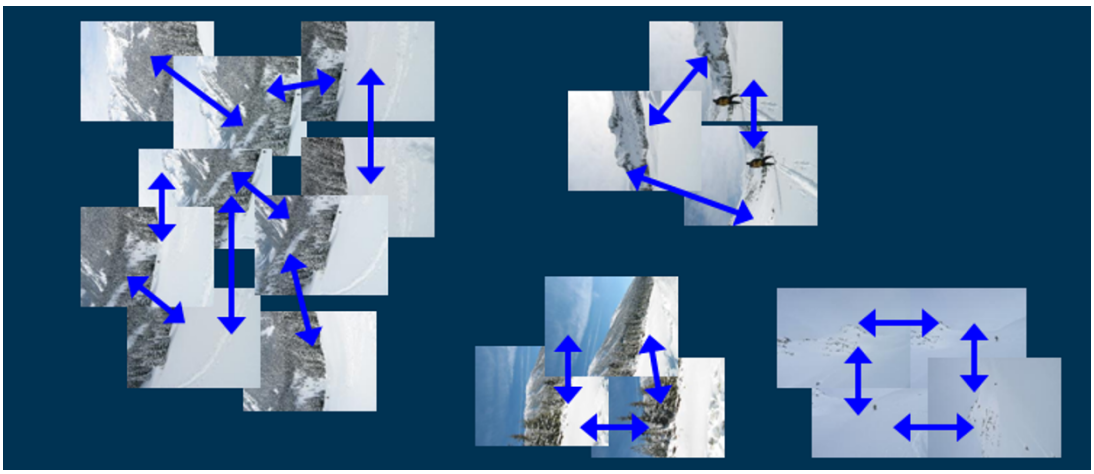
In this section, we review techniques that address these problems, namely compositing surface parameterization, pixel and seam selection, blending, and exposure compensation. Our emphasis is on fully automated approaches to the problem. Since the creation of high-quality panoramas and composites is as much an artistic endeavor as a computational one, various interactive tools have been developed to assist this process (Agarwala, Dontcheva *et al.* 2004; Li, Sun *et al.* 2004; Rother, Kolmogorov, and Blake 2004). Some of these are covered in more detail in Section 10.4.

8.4.1 Choosing a compositing surface

The first choice to be made is how to represent the final image. If only a few images are stitched together, a natural approach is to select one of the images as the *reference* and to then warp all of the other images into its reference coordinate system. The resulting composite is sometimes called



(a)



(b)

(c)

Figure 8.11 Recognizing panoramas (Brown, Szeliski, and Winder 2005), figures courtesy of Matthew Brown: (a) input images with pairwise matches; (b) images grouped into connected components (panoramas); (c) individual panoramas registered and blended into stitched composites.

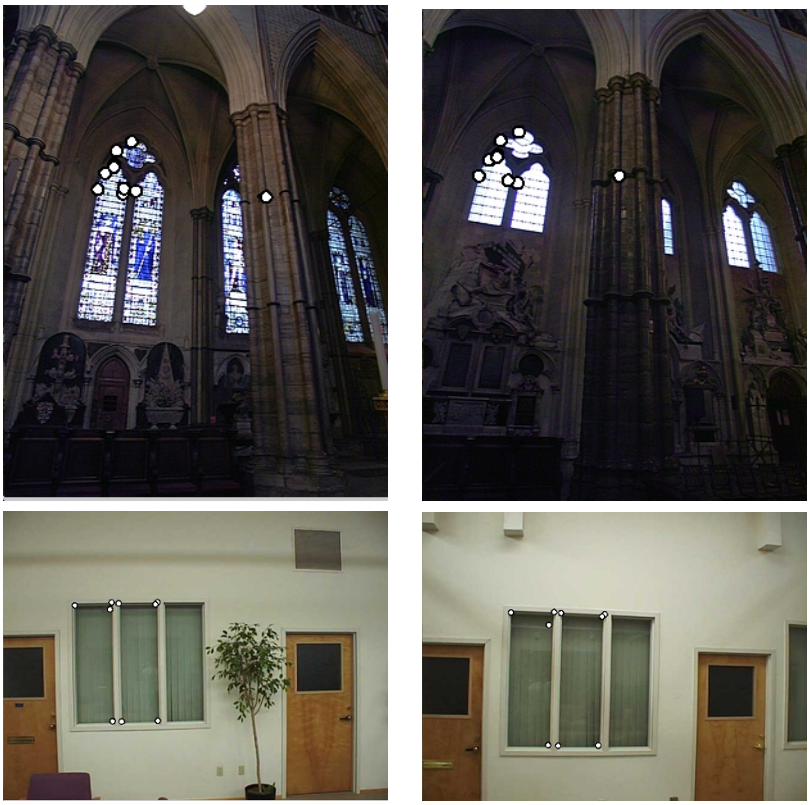


Figure 8.12 Matching errors (Brown, Szeliski, and Winder 2004): accidental matching of several features can lead to matches between pairs of images that do not actually overlap.

Figure 8.13 Validation of image matches by direct pixel error comparison can fail when the scene contains moving objects (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE.

a *flat* panorama, since the projection onto the final surface is still a perspective projection, and hence straight lines remain straight (which is often a desirable attribute).[23]

For larger fields of view, however, we cannot maintain a flat representation without excessively stretching pixels near the border of the image. (In practice, flat panoramas start to look severely distorted once the field of view exceeds 90° or so.) The usual choice for compositing larger panoramas is to use a cylindrical (Chen 1995; Szeliski 1996) or spherical (Szeliski and Shum 1997) projection, as described in Section 8.2.6. In fact, any surface used for *environment mapping* in computer graphics can be used, including a *cube map*, which represents the full viewing sphere with the six square faces of a cube (Greene 1986; Szeliski and Shum 1997). Cartographers have also developed a number of alternative methods for representing the globe (Bugayevskiy and Snyder 1995).

The choice of parameterization is somewhat application-dependent and involves a tradeoff between keeping the local appearance undistorted (e.g., keeping straight lines straight) and providing a reasonably uniform sampling of the environment. Automatically making this selection and smoothly transitioning between representations based on the extent of the panorama is discussed in Kopf, Uyttendaele *et al.* (2007). A recent trend in panoramic photography has been the use of stereographic projections looking down at the ground (in an outdoor scene) to create "little planet" renderings.[24]

View selection. Once we have chosen the output parameterization, we still need to determine which part of the scene will be *centered* in the final view. As mentioned above, for a flat composite, we can choose one of the images as a reference. Often, a reasonable choice is the one that is geometrically most central. For example, for rotational panoramas represented as a collection of 3D rotation matrices, we can choose the image whose z-axis is closest to the average z-axis (assuming a reasonable field of view). Alternatively, we can use the average z-axis (or quaternion, but this is trickier) to define the reference rotation matrix.

For larger, e.g., cylindrical or spherical, panoramas, we can use the same heuristic if a subset of the viewing sphere has been imaged. In the case of full 360° panoramas, a better choice is to choose the middle image from the sequence of inputs, or sometimes the first image, assuming this contains the object of greatest interest. In all of these cases, having the user control the final view is often highly desirable. If the "up vector" computation described in Section 8.3.1 is working correctly, this can be as simple as panning over the image or setting a vertical "center line" for the final panorama.

Coordinate transformations. After selecting the parameterization and reference view, we still need to compute the mappings between the input and output pixels coordinates.

If the final compositing surface is flat (e.g., a single plane or the face of a cube map) and the input images have no radial distortion, the coordinate transformation is the simple homography described by Equation (8.38). This kind of warping can be performed in graphics hardware by appropriately setting texture mapping coordinates and rendering a single quadrilateral.

If the final composite surface has some other analytic form (e.g., cylindrical or spherical), we need to convert every pixel in the final panorama into a viewing ray (3D point) and then map it back into each image according to the projection (and optionally radial distortion) equations. This process can be made more efficient by precomputing some lookup tables, e.g., the partial trigonometric functions needed to map cylindrical or spherical coordinates to 3D coordinates or the radial distortion field at each pixel. It is also possible to accelerate this process by computing exact pixel mappings on a coarser grid and then interpolating these values.

[23] Techniques have also been developed to straighten curved lines in cylindrical and spherical panoramas (Carroll, Agrawala, and Agarwala 2009; Kopf, Lischinski *et al.* 2009; Carroll, Agarwala, and Agrawala 2010).

[24] These are inspired by *The Little Prince* by Antoine De Saint-Exupery. Go to https://www.flickr.com and search for "little planet projection".

When the final compositing surface is a texture-mapped polyhedron, a slightly more sophisticated algorithm must be used. Not only do the 3D and texture map coordinates have to be properly handled, but a small amount of *overdraw* outside the triangle footprints in the texture map is necessary, to ensure that the texture pixels being interpolated during 3D rendering have valid values (Szeliski and Shum 1997).

Sampling issues. While the above computations can yield the correct (fractional) pixel addresses in each input image, we still need to pay attention to sampling issues. For example, if the final panorama has a lower resolution than the input images, prefiltering the input images is necessary to avoid aliasing. These issues have been extensively studied in both the image processing and computer graphics communities. The basic problem is to compute the appropriate prefilter, which depends on the distance (and arrangement) between neighboring samples in a source image. As discussed in Sections 3.5.2 and 3.6.1, various approximate solutions, such as MIP mapping (Williams 1983) or elliptically weighted Gaussian averaging (Greene and Heckbert 1986) have been developed in the graphics community. For highest visual quality, a higher order (e.g., cubic) interpolator combined with a spatially adaptive prefilter may be necessary (Wang, Kang *et al.* 2001). Under certain conditions, it may also be possible to produce images with a higher resolution than the input images using the process of *super-resolution* (Section 10.3).

8.4.2 Pixel selection and weighting (deghosting)

Once the source pixels have been mapped onto the final composite surface, we must still decide how to blend them in order to create an attractive-looking panorama. If all of the images are in perfect registration and identically exposed, this is an easy problem, i.e., any pixel or combination will do. However, for real images, visible seams (due to exposure differences), blurring (due to misregistration), or ghosting (due to moving objects) can occur.

Creating clean, pleasing-looking panoramas involves both deciding which pixels to use and how to weight or blend them. The distinction between these two stages is a little fluid, since per-pixel weighting can be thought of as a combination of selection and blending. In this section, we discuss spatially varying weighting, pixel selection (seam placement), and then more sophisticated blending.

Feathering and center-weighting. The simplest way to create a final composite is to simply take an *average* value at each pixel,

$$C(\mathbf{x}) = \sum_k w_k(\mathbf{x}) \tilde{I}_k(\mathbf{x}) \Big/ \sum_k w_k(\mathbf{x}) \ , \tag{8.70}$$

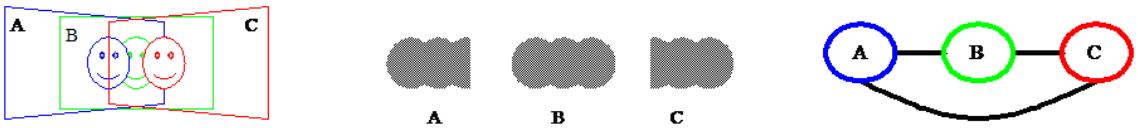
where $\tilde{I}_k(\mathbf{x})$ are the *warped* (re-sampled) images and $w_k(\mathbf{x})$ is 1 at valid pixels and 0 elsewhere. On computer graphics hardware, this kind of summation can be performed in an *accumulation buffer* (using the A channel as the weight).

Simple averaging usually does not work very well, since exposure differences, misregistrations, and scene movement are all very visible (Figure 8.14a). If rapidly moving objects are the only problem, taking a *median* filter (which is a kind of pixel selection operator) can often be used to remove them (Figure 8.14b) (Irani and Anandan 1998). Conversely, center-weighting (discussed below) and *minimum likelihood* selection (Agarwala, Dontcheva *et al.* 2004) can sometimes be used to retain multiple copies of a moving object (Figure 8.17).

A better approach to averaging is to weight pixels near the center of the image more heavily and to down-weight pixels near the edges. When an image has some cutout regions, down-weighting

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 8.14 Final composites computed by a variety of algorithms (Szeliski 2006a): (a) average, (b) median, (c) feathered average, (d) *p-norm* $p = 10$, (e) Voronoi, (f) weighted ROD vertex cover with feathering, (g) graph cut seams with Poisson blending, and (h) with pyramid blending.



(a) (b) (c)

Figure 8.15 Computation of regions of difference (RODs) (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE: (a) three overlapping images with a moving face; (b) corresponding RODs; (c) graph of coincident RODs.

pixels near the edges of both cutouts and the image is preferable. This can be done by computing a *distance map* or *grassfire transform*,

$$w_k(\mathbf{x}) = \arg\min_{\mathbf{y}} \{\|\mathbf{y}\| \mid \tilde{I}_k(\mathbf{x}+\mathbf{y}) \text{ is invalid } \}, \tag{8.71}$$

where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel (Section 3.3.3). The Euclidean distance map can be efficiently computed using a two-pass raster algorithm (Danielsson 1980; Borgefors 1986).

Weighted averaging with a distance map is often called *feathering* (Szeliski and Shum 1997; Chen and Klette 1999; Uyttendaele, Eden, and Szeliski 2001) and does a reasonable job of blending over exposure differences. However, blurring and ghosting can still be problems (Figure 8.14c). Note that weighted averaging is *not* the same as compositing the individual images with the classic *over* operation (Porter and Duff 1984; Blinn 1994a), even when using the weight values (normalized to sum up to one) as *alpha* (translucency) channels. This is because the over operation attenuates the values from more distant surfaces and, hence, is not equivalent to a direct sum.

One way to improve feathering is to raise the distance map values to some large power, i.e., to use $w_k^p(\mathbf{x})$ in Equation (8.70). The weighted averages then become dominated by the larger values, i.e., they act somewhat like a *p-norm*. The resulting composite can often provide a reasonable tradeoff between visible exposure differences and blur (Figure 8.14d).

In the limit as $p \to \infty$, only the pixel with the maximum weight is selected. This hard pixel selection process produces a visibility mask-sensitive variant of the familiar *Voronoi diagram*, which assigns each pixel to the nearest image center in the set (Wood, Finkelstein *et al.* 1997; Peleg, Rousso *et al.* 2000). The resulting composite, while useful for artistic guidance and in high-overlap panoramas (*manifold mosaics*) tends to have very hard edges with noticeable seams when the exposures vary (Figure 8.14e).

Xiong and Turkowski (1998) use this Voronoi idea (local maximum of the grassfire transform) to select seams for Laplacian pyramid blending (which is discussed below). However, since the seam selection is performed sequentially as new images are added in, some artifacts can occur.

Optimal seam selection. Computing the Voronoi diagram is one way to select the *seams* between regions where different images contribute to the final composite. However, Voronoi images totally ignore the local image structure underlying the seam. A better approach is to place the seams in regions where the images agree, so that transitions from one source to another are not visible. In this way, the algorithm avoids "cutting through" moving objects where a seam would look unnatural (Davis 1998). For a pair of images, this process can be formulated as a simple dynamic program starting from one edge of the overlap region and ending at the other (Milgram 1975, 1977; Davis 1998; Efros and Freeman 2001).

Figure 8.16 Photomontage (Agarwala, Dontcheva *et al.* 2004) © 2004 ACM. From a set of five source images (of which four are shown on the left), Photomontage quickly creates a composite family portrait in which everyone is smiling and looking at the camera (right). Users simply flip through the stack and coarsely draw strokes using the designated source image objective over the people they wish to add to the composite. The user-applied strokes and computed regions (middle) are color-coded by the borders of the source images on the left.

When multiple images are being composited, the dynamic program idea does not readily generalize. (For square texture tiles being composited sequentially, Efros and Freeman (2001) run a dynamic program along each of the four tile sides.)

To overcome this problem, Uyttendaele, Eden, and Szeliski (2001) observed that, for well-registered images, moving objects produce the most visible artifacts, namely translucent looking *ghosts*. Their system therefore decides which objects to keep and which ones to erase. First, the algorithm compares all overlapping input image pairs to determine *regions of difference* (RODs) where the images disagree. Next, a graph is constructed with the RODs as vertices and edges representing ROD pairs that overlap in the final composite (Figure 8.15). Since the presence of an edge indicates an area of disagreement, vertices (regions) must be removed from the final composite until no edge spans a pair of remaining vertices. The smallest such set can be computed using a *vertex cover* algorithm. Since several such covers may exist, a *weighted vertex cover* is used instead, where the vertex weights are computed by summing the feather weights in the ROD (Uyttendaele, Eden, and Szeliski 2001). The algorithm therefore prefers removing regions that are near the edge of the image, which reduces the likelihood that partially visible objects will appear in the final composite. (It is also possible to infer which object in a region of difference is the foreground object by the "edginess" (pixel differences) across the ROD boundary, which should be higher when an object is present (Herley 2005).) Once the desired excess regions of difference have been removed, the final composite can be created by feathering (Figure 8.14f).

A different approach to pixel selection and seam placement is described by Agarwala, Dontcheva *et al.* (2004). Their system computes the label assignment that optimizes the sum of two objective functions. The first is a per-pixel *image objective* that determines which pixels are likely to produce good composites,

$$E_{\mathrm{D}} = \sum_{\mathbf{x}} D(\mathbf{x}, l(\mathbf{x})), \tag{8.72}$$

where $D(\mathbf{x}, l)$ is the *data penalty* associated with choosing image l at pixel $\mathbf{x}$. In their system, users can select which pixels to use by "painting" over an image with the desired object or appearance, which sets $D(\mathbf{x}, l)$ to a large value for all labels l other than the one selected by the user (Figure 8.16). Alternatively, automated selection criteria can be used, such as *maximum likelihood*, which prefers pixels that occur repeatedly in the background (for object removal), or *minimum likelihood* for objects that occur infrequently, i.e., for moving object retention. Using a more traditional

Figure 8.17 Set of five photos tracking a snowboarder's jump stitched together into a seamless composite. Because the algorithm prefers pixels near the center of the image, multiple copies of the boarder are retained.

center-weighted data term tends to favor objects that are centered in the input images (Figure 8.17).

The second term is a *seam objective* that penalizes differences in labels between adjacent images,

$$E_{\mathrm{S}} = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{N}} S(\mathbf{x},\mathbf{y},l(\mathbf{x}),l(\mathbf{y})), \tag{8.73}$$

where $S(\mathbf{x},\mathbf{y},l_x,l_y)$ is the image-dependent *interaction penalty* or *seam cost* of placing a seam between pixels $\mathbf{x}$ and $\mathbf{y}$, and $\mathcal{N}$ is the set of $\mathcal{N}_4$ neighboring pixels. For example, the simple color-based seam penalty used in Kwatra, Schödl *et al.* (2003) and Agarwala, Dontcheva *et al.* (2004) can be written as

$$S(\mathbf{x},\mathbf{y},l_x,l_y) = \|\tilde{I}_{l_x}(\mathbf{x}) - \tilde{I}_{l_y}(\mathbf{x})\| + \|\tilde{I}_{l_x}(\mathbf{y}) - \tilde{I}_{l_y}(\mathbf{y})\|. \tag{8.74}$$

More sophisticated seam penalties can also look at image gradients or the presence of image edges (Agarwala, Dontcheva *et al.* 2004). Seam penalties are widely used in other computer vision applications such as stereo matching (Boykov, Veksler, and Zabih 2001) to give the labeling function its *coherence* or *smoothness*. An alternative approach, which places seams along strong consistent edges in overlapping images using a watershed computation is described by Soille (2006).

The sum of these two objective functions gives rise to a *Markov random field* (MRF), for which good optimization algorithms are described in Sections 4.3 and 4.3.2 and Appendix B.5. For label computations of this kind, the *α-expansion* algorithm developed by Boykov, Veksler, and Zabih (2001) works particularly well (Szeliski, Zabih *et al.* 2008).

For the result shown in Figure 8.14g, Agarwala, Dontcheva *et al.* (2004) use a large data penalty for invalid pixels and 0 for valid pixels. Notice how the seam placement algorithm avoids regions of difference, including those that border the image and that might result in objects being cut off. Graph cuts (Agarwala, Dontcheva *et al.* 2004) and vertex cover (Uyttendaele, Eden, and Szeliski 2001) often produce similar looking results, although the former is significantly slower since it optimizes over all pixels, while the latter is more sensitive to the thresholds used to determine regions of difference. More recent approaches to seam selection include SEAGULL (Lin, Jiang *et al.* 2016), which jointly optimizes local alignment and seam selection, and object-centered image stitching (Herrmann, Wang *et al.* 2018a), which uses an off-the-shelf object detector to avoid cutting through objects.

8.4.3 *Application*: Photomontage

While image stitching is normally used to composite partially overlapping photographs, it can also be used to composite repeated shots of a scene taken with the aim of obtaining the best possible composition and appearance of each element.

Figure 8.16 shows the *Photomontage* system developed by Agarwala, Dontcheva *et al.* (2004), where users draw strokes over a set of pre-aligned images to indicate which regions they wish to keep from each image. Once the system solves the resulting multi-label graph cut (8.72–8.73), the various pieces taken from each source photo are blended together using a variant of Poisson image blending (8.75–8.77). Their system can also be used to automatically composite an all-focus image from a series of bracketed focus images (Hasinoff, Kutulakos *et al.* 2009) or to remove wires and other unwanted elements from sets of photographs. Exercise 8.14 has you implement this system and try out some of its variants.

8.4.4 Blending

Once the seams between images have been determined and unwanted objects removed, we still need to blend the images to compensate for exposure differences and other misalignments. The spatially varying weighting (feathering) previously discussed can often be used to accomplish this. However, it is difficult in practice to achieve a pleasing balance between smoothing out low-frequency exposure variations and retaining sharp enough transitions to prevent blurring (although using a high exponent in feathering can help).

Laplacian pyramid blending. An attractive solution to this problem is the Laplacian pyramid blending technique developed by Burt and Adelson (1983b), which we discussed in Section 3.5.5. Instead of using a single transition width, a frequency-adaptive width is used by creating a band-pass (Laplacian) pyramid and making the transition widths within each level a function of the level, i.e., the same width in pixels. In practice, a small number of levels, i.e., as few as two (Brown and Lowe 2007), may be adequate to compensate for differences in exposure. The result of applying this pyramid blending is shown in Figure 8.14h.

Gradient domain blending. An alternative approach to multi-band image blending is to perform the operations in the *gradient domain*. Reconstructing images from their gradient fields has a long history in computer vision (Horn 1986), starting originally with work in brightness constancy (Horn 1974), shape from shading (Horn and Brooks 1989), and photometric stereo (Woodham 1981). Related ideas have also been used for reconstructing images from their edges (Elder and Goldberg 2001), removing shadows from images (Weiss 2001), separating reflections from a single image (Levin, Zomet, and Weiss 2004; Levin and Weiss 2007), and *tone mapping* high dynamic range images by reducing the magnitude of image edges (gradients) (Fattal, Lischinski, and Werman 2002).

Pérez, Gangnet, and Blake (2003) show how gradient domain reconstruction can be used to do seamless object insertion in image editing applications (Figure 8.18). Rather than copying pixels, the *gradients* of the new image fragment are copied instead. The actual pixel values for the copied area are then computed by solving a *Poisson equation* that locally matches the gradients while obeying the fixed *Dirichlet* (exact matching) conditions at the seam boundary. Pérez, Gangnet, and Blake (2003) show that this is equivalent to computing an additive *membrane* interpolant of the mismatch

(a) (b) (c)

Figure 8.18 Poisson image editing (Pérez, Gangnet, and Blake 2003) © 2003 ACM: (a) The dog and the two children are chosen as source images to be pasted into the destination swimming pool. (b) Simple pasting fails to match the colors at the boundaries, whereas (c) Poisson image blending masks these differences.

between the source and destination images along the boundary.[25] In earlier work, Peleg (1981) also proposed adding a smooth function to enforce consistency along the seam curve.

Agarwala, Dontcheva *et al.* (2004) extended this idea to a multi-source formulation, where it no longer makes sense to talk of a destination image whose exact pixel values must be matched at the seam. Instead, *each* source image contributes its own gradient field and the Poisson equation is solved using *Neumann* boundary conditions, i.e., dropping any equations that involve pixels outside the boundary of the image.

Rather than solving the Poisson partial differential equations, Agarwala, Dontcheva *et al.* (2004) directly minimize a *variational problem*,

$$\min_{C(\mathbf{x})} \|\nabla C(\mathbf{x}) - \nabla \tilde{I}_{l(\mathbf{x})}(\mathbf{x})\|^2. \tag{8.75}$$

The discretized form of this equation is a set of gradient constraint equations

$$C(\mathbf{x}+\hat{\imath}) - C(\mathbf{x}) = \tilde{I}_{l(\mathbf{x})}(\mathbf{x}+\hat{\imath}) - \tilde{I}_{l(\mathbf{x})}(\mathbf{x}) \quad \text{and} \tag{8.76}$$

$$C(\mathbf{x}+\hat{\jmath}) - C(\mathbf{x}) = \tilde{I}_{l(\mathbf{x})}(\mathbf{x}+\hat{\jmath}) - \tilde{I}_{l(\mathbf{x})}(\mathbf{x}), \tag{8.77}$$

where $\hat{\imath} = (1,0)$ and $\hat{\jmath} = (0,1)$ are unit vectors in the x and y directions.[26] They then solve the associated sparse least squares problem. Since this system of equations is only defined up to an additive constraint, Agarwala, Dontcheva *et al.* (2004) ask the user to select the value of one pixel. In practice, a better choice might be to weakly bias the solution towards reproducing the original color values.

In order to accelerate the solution of this sparse linear system, Fattal, Lischinski, and Werman (2002) use multigrid, whereas Agarwala, Dontcheva *et al.* (2004) use hierarchical basis preconditioned conjugate gradient descent (Szeliski 1990b, 2006b; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013) (Appendix A.5). In subsequent work, Agarwala (2007) shows how using a quadtree representation for the solution can further accelerate the computation with minimal loss

[25] The membrane interpolant is known to have nicer interpolation properties for arbitrary-shaped constraints than frequency-domain interpolants (Nielson 1993).

[26] At seam locations, the right-hand side is replaced by the average of the gradients in the two source images.

in accuracy, while Szeliski, Uyttendaele, and Steedly (2008) show how representing the per-image offset fields using coarser splines is even faster. This latter work also argues that blending in the log domain, i.e., using multiplicative rather than additive offsets, is preferable, as it more closely matches texture contrasts across seam boundaries. The resulting seam blending works very well in practice (Figure 8.14h), although care must be taken when copying large gradient values near seams so that a "double edge" is not introduced.

Copying gradients directly from the source images after seam placement is just one approach to gradient domain blending. The paper by Levin, Zomet *et al.* (2004) examines several different variants of this approach, which they call *Gradient-domain Image STitching* (GIST). The techniques they examine include feathering (blending) the gradients from the source images, as well as using an L_1norm in performing the reconstruction of the image from the gradient field, rather than using an L_2norm as in Equation (8.75). Their preferred technique is the L_1 optimization of a feathered (blended) cost function on the original image gradients (which they call GIST1-l_1). Since L_1optimization using linear programming can be slow, they develop a faster iterative median-based algorithm in a multigrid framework. Visual comparisons between their preferred approach and what they call *optimal seam on the gradients* (which is equivalent to the approach of Agarwala, Dontcheva *et al.* (2004)) show similar results, while significantly improving on pyramid blending and feathering algorithms.

Exposure compensation. Pyramid and gradient domain blending can do a good job of compensating for moderate amounts of exposure differences between images. However, when the exposure differences become large, alternative approaches may be necessary.

Uyttendaele, Eden, and Szeliski (2001) iteratively estimate a local correction between each source image and a blended composite. First, a block-based quadratic transfer function is fit between each source image and an initial feathered composite. Next, transfer functions are averaged with their neighbors to get a smoother mapping and per-pixel transfer functions are computed by *splining* (interpolating) between neighboring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed and the process is repeated (typically three times). The results shown by Uyttendaele, Eden, and Szeliski (2001) demonstrate that this does a better job of exposure compensation than simple feathering and can handle local variations in exposure due to effects such as lens vignetting.

Ultimately, however, the most principled way to deal with exposure differences is to stitch images in the radiance domain, i.e., to convert each image into a radiance image using its exposure value and then create a stitched, high dynamic range image, as discussed in Section 10.2 and Eden, Uyttendaele, and Szeliski (2006).

8.5 Additional reading

Hartley and Zisserman (2004) provide a wonderful introduction to the topics of feature-based alignment and optimal motion estimation. Techniques for robust estimation are discussed in more detail in Appendix B.3 and in monographs and review articles on this topic (Huber 1981; Hampel, Ronchetti *et al.* 1986; Rousseeuw and Leroy 1987; Black and Rangarajan 1996; Stewart 1999). The most commonly used robust initialization technique in computer vision is RANdom SAmple Consensus (RANSAC) (Fischler and Bolles 1981), which has spawned a series of more efficient variants (Torr and Zisserman 2000; Nistér 2003; Chum and Matas 2005; Raguram, Chum *et al.* 2012; Brachmann, Krull *et al.* 2017; Barath and Matas 2018; Barath, Matas, and Noskova 2019; Brachmann and

Rother 2019). The MAGSAC++ paper by Barath, Noskova *et al.* (2020) compares many of these variants.

The literature on image stitching dates back to work in the photogrammetry community in the 1970s (Milgram 1975, 1977; Slama 1980). In computer vision, papers started appearing in the early 1980s (Peleg 1981), while the development of fully automated techniques came about a decade later (Mann and Picard 1994; Chen 1995; Szeliski 1996; Szeliski and Shum 1997; Sawhney and Kumar 1999; Shum and Szeliski 2000). Those techniques used direct pixel-based alignment but feature-based approaches are now the norm (Zoghlami, Faugeras, and Deriche 1997; Capel and Zisserman 1998; Cham and Cipolla 1998; Badra, Qumsieh, and Dudek 1998; McLauchlan and Jaenicke 2002; Brown and Lowe 2007). A collection of some of these papers can be found in the book by Benosman and Kang (2001). Szeliski (2006a) provides a comprehensive survey of image stitching, on which the material in this chapter is based. More recent publications include Zaragoza, Chin *et al.* (2013), Zhang and Liu (2014), Lin, Pankanti *et al.* (2015), Lin, Jiang *et al.* (2016), Herrmann, Wang *et al.* (2018b), Lee and Sim (2020), and Zhuang and Tran (2020).

High-quality techniques for optimal seam finding and blending are another important component of image stitching systems. Important developments in this field include work by Milgram (1977), Burt and Adelson (1983b), Davis (1998), Uyttendaele, Eden, and Szeliski (2001), Pérez, Gangnet, and Blake (2003), Levin, Zomet *et al.* (2004), Agarwala, Dontcheva *et al.* (2004), Eden, Uyttendaele, and Szeliski (2006), Kopf, Uyttendaele *et al.* (2007), Lin, Jiang *et al.* (2016), and Herrmann, Wang *et al.* (2018a).

In addition to the merging of multiple overlapping photographs taken for aerial or terrestrial panoramic image creation, stitching techniques can be used for automated whiteboard scanning (He and Zhang 2005; Zhang and He 2007), scanning with a mouse (Nakao, Kashitani, and Kaneyoshi 1998), and retinal image mosaics (Can, Stewart *et al.* 2002). They can also be applied to video sequences (Teodosio and Bender 1993; Irani, Hsu, and Anandan 1995; Kumar, Anandan *et al.* 1995; Sawhney and Ayer 1996; Massey and Bender 1996; Irani and Anandan 1998; Sawhney, Arpa *et al.* 2002; Agarwala, Zheng *et al.* 2005; Rav-Acha, Pritch *et al.* 2005; Steedly, Pal, and Szeliski 2005; Baudisch, Tan *et al.* 2006) and can even be used for video compression (Lee, Chen *et al.* 1997).

8.6 Exercises

Ex 8.1: Feature-based image alignment for flip-book animations. Take a set of photos of an action scene or portrait (preferably in burst shooting mode) and align them to make a composite or flip-book animation.

1. Extract features and feature descriptors using some of the techniques described in Sections 7.1.1–7.1.2.
2. Match your features using nearest neighbor matching with a nearest neighbor distance ratio test (7.18).
3. Compute an optimal 2D translation and rotation between the first image and all subsequent images, using least squares (Section 8.1.1) with optional RANSAC for robustness (Section 8.1.4).
4. Resample all of the images onto the first image's coordinate frame (Section 3.6.1) using either bilinear or bicubic resampling and optionally crop them to their common area.
5. Convert the resulting images into an animated GIF (using software available from the web) or optionally implement cross-dissolves to turn them into a "slo-mo" video.

6. (Optional) Combine this technique with feature-based (Exercise 3.25) morphing.

Ex 8.2: Panography. Create the kind of panograph discussed in Section 8.1.2 and commonly found on the web.

1. Take a series of interesting overlapping photos.
2. Use the feature detector, descriptor, and matcher developed in Exercises 7.1–7.4 (or existing software) to match features among the images.
3. Turn each connected component of matching features into a *track*, i.e., assign a unique index i to each track, discarding any tracks that are inconsistent (contain two different features in the same image).
4. Compute a global translation for each image using Equation (8.12).
5. Since your matches probably contain errors, turn the above least square metric into a robust metric (8.25) and re-solve your system using iteratively reweighted least squares.
6. Compute the size of the resulting composite canvas and resample each image into its final position on the canvas. (Keeping track of bounding boxes will make this more efficient.)
7. Average all of the images, or choose some kind of ordering and implement translucent *over* compositing (3.8).
8. (Optional) Extend your parametric motion model to include rotations and scale, i.e., the similarity transform given in Table 8.1. Discuss how you could handle the case of translations and rotations only (no scale).
9. (Optional) Write a simple tool to let the user adjust the ordering and opacity, and add or remove images.
10. (Optional) Write down a different least squares problem that involves pairwise matching of images. Discuss why this might be better or worse than the global matching formula given in (8.12).

Ex 8.3: 2D rigid/Euclidean matching. Several alternative approaches are given in Section 8.1.3 for estimating a 2D rigid (Euclidean) alignment.

1. Implement the various alternatives and compare their accuracy on synthetic data, i.e., random 2D point clouds with noisy feature positions.
2. One approach is to estimate the translations from the centroids and then estimate rotation in polar coordinates. Do you need to weight the angles obtained from a polar decomposition in some way to get the statistically correct estimate?
3. How can you modify your techniques to take into account either scalar (8.10) or full two-dimensional point covariance weightings (8.11)? Do all of the previously developed "shortcuts" still work or does full weighting require iterative optimization?

Ex 8.4: 2D match move/augmented reality. Replace a picture in a magazine or a book with a different image or video.

1. Take a picture of a magazine or book page.

2. Outline a figure or picture on the page with a rectangle, i.e., draw over the four sides as they appear in the image.
3. Match features in this area with each new image frame.
4. Replace the original image with an "advertising" insert, warping the new image with the appropriate homography.
5. Try your approach on a clip from a sporting event (e.g., indoor or outdoor soccer) to implement a billboard replacement.

Ex 8.5: Direct pixel-based alignment. Take a pair of images, compute a coarse-to-fine affine alignment (Exercise 9.2) and then blend them using either averaging (Exercise 8.2) or a Laplacian pyramid (Exercise 3.18). Extend your motion model from affine to perspective (homography) to better deal with rotational mosaics and planar surfaces seen under arbitrary motion.

Ex 8.6: Featured-based stitching. Extend your feature-based alignment technique from Exercise 8.2 to use a full perspective model and then blend the resulting mosaic using either averaging or more sophisticated distance-based feathering (Exercise 8.13).

Ex 8.7: Cylindrical strip panoramas. To generate cylindrical or spherical panoramas from a horizontally panning (rotating) camera, it is best to use a tripod. Set your camera up to take a series of 50% overlapped photos and then use the following steps to create your panorama:

1. Estimate the amount of radial distortion by taking some pictures with lots of long straight lines near the edges of the image and then using the plumb-line method from Exercise 11.5.
2. Compute the focal length either by using a ruler and paper (Debevec, Wenger *et al.* 2002) or by rotating your camera on the tripod, overlapping the images by exactly 0% and counting the number of images it takes to make a 360° panorama.
3. Convert each of your images to cylindrical coordinates using (8.45–8.49).
4. Line up the images with a translational motion model using either a direct pixel-based technique, such as coarse-to-fine incremental or an FFT, or a feature-based technique.
5. (Optional) If doing a complete 360° panorama, align the first and last images. Compute the amount of accumulated vertical misregistration and re-distribute this among the images.
6. Blend the resulting images using feathering or some other technique.

Ex 8.8: Coarse alignment. Use FFT or phase correlation (Section 9.1.2) to estimate the initial alignment between successive images. How well does this work? Over what range of overlaps? If it does not work, does aligning sub-sections (e.g., quarters) do better?

Ex 8.9: Automated mosaicing. Use feature-based alignment with four-point RANSAC for homographies (Section 8.1.3, Equations (8.19–8.23)) or three-point RANSAC for rotational motions (Brown, Hartley, and Nistér 2007) to match up all pairs of overlapping images.

Merge these pairwise estimates together by finding a spanning tree of pairwise relations. Visualize the resulting global alignment, e.g., by displaying a blend of each image with all other images that overlap it.

For greater robustness, try multiple spanning trees (perhaps randomly sampled based on the confidence in pairwise alignments) to see if you can recover from bad pairwise matches (Zach, Klopschitz, and Pollefeys 2010). As a measure of fitness, count how many pairwise estimates are consistent with the global alignment.

Ex 8.10: Global optimization. Use the initialization from the previous algorithm to perform a full bundle adjustment over all of the camera rotations and focal lengths, as described in Section 11.4.2 and by Shum and Szeliski (2000). Optionally, estimate radial distortion parameters as well or support fisheye lenses (Section 2.1.5).

As in the previous exercise, visualize the quality of your registration by creating composites of each input image with its neighbors, optionally blinking between the original image and the composite to better see misalignment artifacts.

Ex 8.11: Deghosting. Use the results of the previous bundle adjustment to predict the location of each feature in a consensus geometry. Use the difference between the predicted and actual feature locations to correct for small misregistrations, as described in Section 8.3.2 (Shum and Szeliski 2000).

Ex 8.12: Compositing surface. Choose a compositing surface (Section 8.4.1), e.g., a single reference image extended to a larger plane, a sphere represented using cylindrical or spherical coordinates, a stereographic "little planet" projection, or a cube map.

Project all of your images onto this surface and blend them with equal weighting, for now (just to see where the original image seams are).

Ex 8.13: Feathering and blending. Compute a feather (distance) map for each warped source image and use these maps to blend the warped images.

Alternatively, use Laplacian pyramid blending (Exercise 3.18) or gradient domain blending.

Ex 8.14: Photomontage and object removal. Implement a "Photomontage" system in which users can indicate desired or unwanted regions in pre-registered images using strokes or other primitives (such as bounding boxes).

(Optional) Devise an automatic moving objects remover (or "keeper") by analyzing which inconsistent regions are more or less typical given some consensus (e.g., median filtering) of the aligned images. Figure 8.17 shows an example where the moving object was kept. Try to make this work for sequences with large amounts of overlaps and consider averaging the images to make the moving object look more ghosted.