

Image Alignment and Stitching

Richard Szeliski

ABSTRACT Stitching multiple images together to create beautiful high-resolution panoramas is one of the most popular consumer applications of image registration and blending. In this chapter, I review the motion models (geometric transformations) that underlie panoramic image stitching, discuss direct intensity-based and feature-based registration algorithms, and present global and local alignment techniques needed to establish high-accuracy correspondences between overlapping images. I then discuss various compositing options, including multi-band and gradient-domain blending, as well as techniques for removing blur and ghosted images. The resulting techniques can be used to create high-quality panoramas for static or interactive viewing.

1 Introduction

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision. Image stitching algorithms have been used for decades to create the high-resolution photo-mosaics used to produce digital maps and satellite photos [20]. Frame-rate image alignment is used in every camcorder that has an image stabilization feature. Image stitching algorithms come “out of the box” with today’s digital cameras and can be used to create beautiful high-resolution panoramas.

In film photography, special cameras were developed at the turn of the century to take wide-angle panoramas, often by exposing the film through a vertical slit as the camera rotated on its axis [18]. In the mid-1990s, image alignment techniques started being applied to the construction of wide-angle seamless panoramas from regular hand-held cameras [17, 9, 27]. More recent work in this area has addressed the need to compute globally consistent alignments [30, 23, 25], the removal of “ghosts” due to parallax and object movement [20, 10, 25, 32, 1], and dealing with varying exposures [17, 32, 1]. (A collection of some of these papers can be found in [3].) These techniques have spawned a large number of commercial stitching products [9, 22].

While most of the above techniques work by directly minimizing pixel-to-pixel dissimilarities, a different class of algorithms works by extracting a sparse set of *features* and then matching these to each other [8, 6]. Feature-based approaches have the advantage of being more robust against scene

movement, and are potentially faster. 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 [6].

What, then, are the fundamental algorithms needed for image stitching? First, we must determine the appropriate *motion model* relating pixel coordinates in one image to pixel coordinates in another (Section 2). Next, we must somehow estimate the correct alignments relating various pairs of images, using either *direct* pixel-to-pixel comparisons combined with gradient descent or *feature-based* alignment techniques (Section 3). We must also develop algorithm to compute globally consistent alignments from large collections of overlapping photos (Section 4). Once the alignments have been estimated, we must choose a final compositing surface onto which to warp and place all of the aligned images (Section 5). We also need to seamlessly blend overlapping images, even in the presence of parallax, lens distortion, scene motion, and exposure differences (Section 6). In the last section of this chapter, I discuss additional applications of image stitching and open research problems. For a more detailed tutorial on all of these components, please consult [28].

2 Motion models

Before we can stitch images to create panoramas, 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, and non-planar (e.g., cylindrical) surfaces [27, 30].

Figure 1 shows a number of commonly used 2D planar transformations, while Table 1.1 lists their mathematical form along with their intrinsic dimensionality. The easiest way to think of these is as a set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors, $\mathbf{x}' = (x', y', 1)$ and $\mathbf{x} = (x, y, 1)$, s.t.

$$\mathbf{x}' \sim \mathbf{H}\mathbf{x}, \quad (1.1)$$

where \sim denotes equality up to scale and \mathbf{H} is one of the 3×3 matrices given in Table 1.1.

2D translations are useful for tracking small patches in videos and for compensating for instantaneous camera jitter. This simple two-parameter model is the one most commonly associated with Lucas and Kanade’s patch tracker [16], although, in fact, their paper also describes how to use an affine motion model.

The three-parameter rotation+translation (also known as *2D rigid body motion* or the *2D Euclidean transformation*) is useful for modeling in-plane

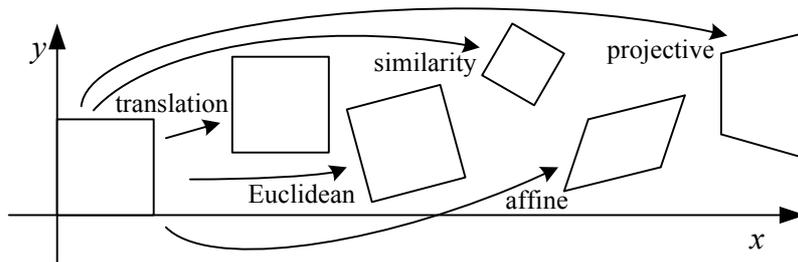


FIGURE 1. Basic set of 2D planar transformations

rotations, for example when different portions of a larger image are scanned on a flatbed scanner.

Scaled rotation, also known as the *similarity transform*, adds a fourth isotropic scale parameter s . This is a good model for a slowly panning and zooming camera, especially when the camera has a long focal length. The similarity transform preserves angles between lines.

The six parameter affine transform uses a general 2×3 matrix (or equivalently, a 3×3 matrix where the bottom row is $[0 \ 0 \ 1]$). It is a good model of local deformations induced by more complex transforms, and also models the 3D surface foreshortening observed by an orthographic camera. Affine transforms preserve parallelism between lines.

The most general planar 2D transform is the eight-parameter *perspective transform* or *homography* denoted by a general 3×3 matrix \mathbf{H} . The result of multiplying $\mathbf{H}\mathbf{x}$ must be normalized in order to obtain an inhomogeneous result, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \quad (1.2)$$

Perspective transformations preserve straight lines, and, as we will see shortly, are an appropriate model for planes observed under general 3D motion and 3D scenes observed under pure camera rotation.

In 3D, the process of *central projection* maps 3D coordinates $\mathbf{x} = (x, y, z)$ to 2D coordinates $\mathbf{x}' = (x', y', 1)$ through a *pinhole* at the camera origin onto a 2D projection plane a distance f along the z axis,

$$x' = f \frac{x}{z}, \quad y' = f \frac{y}{z}. \quad (1.3)$$

Perspective projection can also be denoted using an upper-triangular 3×3 *intrinsic calibration matrix* \mathbf{K} that can account for non-square pixels, skew, and a variable optic center location. However, in practice, the simple focal length scaling used above provides high-quality results when stitching images from regular cameras.

What happens when we take two images of a 3D scene from different camera positions and/or orientations? A 3D point $\mathbf{p} = (X, Y, Z, 1)$ gets

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles + ...	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism + ...	
projective	$\begin{bmatrix} \mathbf{H} \end{bmatrix}_{3 \times 3}$	8	straight lines	

TABLE 1.1. Hierarchy of 2D coordinate transformations. The 2×3 matrices are extended with a third $[\mathbf{0}^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

mapped to an image coordinate \mathbf{x}'_0 through the combination of a 3D rigid-body (Euclidean) motion \mathbf{E}_0 and a perspective projection \mathbf{K}_0 ,

$$\mathbf{x}_0 \sim \mathbf{K}_0 \mathbf{E}_0 \mathbf{p} = \mathbf{P}_0 \mathbf{p}, \quad (1.4)$$

where the 3×4 matrix \mathbf{P}_0 is often called the *camera matrix*. If we have a 2D point \mathbf{x}_0 , we can only project it back into a 3D *ray* in space. However, for a *planar scene*, we have one additional *plane equation*, $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + d_0 = 0$, which we can use to augment \mathbf{P}_0 to obtain $\tilde{\mathbf{P}}_0$, which then allows us to invert the 3D \rightarrow 2D projection. If we then project this point into another image, we obtain

$$\mathbf{x}_1 \sim \mathbf{P}_1 \tilde{\mathbf{P}}_0^{-1} \mathbf{x}_0 = \mathbf{H}_{10} \mathbf{x}_0, \quad (1.5)$$

where \mathbf{H}_{10} is a general 3×3 homography matrix and \mathbf{x}_1 and \mathbf{x}_0 are 2D homogeneous coordinates. This justifies the use of the 8-parameter homography as a general alignment model for mosaics of planar scenes [17, 27].

The more interesting case is when the camera undergoes pure rotation (which is equivalent to assuming all points are far from the camera). In this case, we get the more restricted 3×3 homography

$$\mathbf{H}_{10} = \mathbf{K}_1 \mathbf{R}_1 \mathbf{R}_0^{-1} \mathbf{K}_0^{-1} = \mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1}. \quad (1.6)$$

In practice, we usually set $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$. Thus, instead of the general 8-parameter homography relating a pair of images, we get the 3-, 4-, or 5-parameter *3D rotation* motion models corresponding to the cases where the focal length f is known, fixed, or variable [30]. Estimating the 3D rotation matrix (and optionally, the focal length) associated with each image is intrinsically much more stable than estimating a full 8-d.o.f. homography, which makes this the method of choice for large-scale image stitching algorithms [30, 25, 6].

An alternative to using homographies or 3D rotations is to first warp the images into *cylindrical* coordinates and to then use a pure translational model to align them [9]. Unfortunately, this only works if the images are all taken with a level camera or with a known tilt angle. The equations for mapping between planar and cylindrical/spherical coordinates can be found in [30, 28].

3 Direct and feature-based alignment

Once we have chosen a suitable motion model to describe the alignment between a pair of images, we need to devise some method to estimate its parameters. One approach is to shift or warp the images relative to each other and to look at how much the pixels agree. Approaches such as these are often called *direct methods*, as opposed to the *feature-based methods* described a little later.

3.1 Direct methods

To use a direct method, a suitable *error metric* must first be chosen to compare the images. Once this has been established, a suitable *search* technique must be devised. The simplest search technique is to exhaustively try all possible alignments, i.e., to do a *full search*. In practice, this may be too slow, so *hierarchical* coarse-to-fine techniques based on image pyramids have been developed [4]. Alternatively, Fourier transforms can be used to speed up the computation [28]. To get sub-pixel precision in the alignment, *incremental* methods based on a Taylor series expansion of the image function are often used [16]; these can also be applied to *parametric motion models* [16, 4]. Each of these techniques is described in more detail in [28] and summarized below.

The simplest way to establish an alignment between two images is to shift one image relative to the other. Given a *template* image $I_0(\mathbf{x})$ sampled at discrete pixel locations $\{\mathbf{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\mathbf{x})$. A least-squares solution to this problem is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{\text{SSD}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 = \sum_i e_i^2, \quad (1.7)$$

where $\mathbf{u} = (u, v)$ is the *displacement vector* and $e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$ is called the *residual error*.

In general, the displacement \mathbf{u} can be fractional, so a suitable interpolation function must be applied to image $I_1(\mathbf{x})$. In practice, a bilinear interpolant is often used, but bi-cubic interpolation may yield slightly better results.

We can make the above error metric more robust to outliers by replacing the squared error terms with a robust function $\rho(e_i)$ [26]. We can also model potential *bias and gain* variations between the images being compared, and to associate *spatially varying weights* with different pixels, which is a principled way to deal with partial overlap and regions that have been “cut out” from one of the images [2, 28]. The extended version of this chapter [28] also discusses correlation (and *phase correlation*) as an alternative to robust pixel difference matching. It also discusses how coarse-to-fine (*hierarchical*) techniques [4] and *Fourier transforms* can be used to speed up the search for optimal alignment. (Fourier transforms unfortunately only work for pure translation and for a very limited set of (small-motion) similarity transforms.)

Incremental refinement

To obtain better *sub-pixel* estimates, we can use one of several techniques. One possibility is to evaluate several discrete (integer or fractional) values of (u, v) around the best value found so far and to *interpolate* the matching score to find an analytic minimum. A more commonly used approach, first proposed by Lucas and Kanade [16], is to do *gradient descent* on the SSD energy function (1.7), using a Taylor Series expansion of the image function,

$$E_{\text{LK-SSD}}(\mathbf{u} + \Delta\mathbf{u}) \approx \sum_i [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2, \quad (1.8)$$

where

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) = \nabla I_1(\mathbf{x}_i + \mathbf{u}) = \left(\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y} \right)(\mathbf{x}_i + \mathbf{u}) \quad (1.9)$$

is the *image gradient* at $\mathbf{x}_i + \mathbf{u}$.

The above least squares problem can be minimized by solving the associated *normal equations*.

$$\mathbf{A}\Delta\mathbf{u} = \mathbf{b} \quad (1.10)$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u})\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \quad \text{and} \quad \mathbf{b} = - \sum_i e_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) \quad (1.11)$$

are called the *Hessian* and *gradient-weighted residual vector*, respectively.

The gradients required for $\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\mathbf{x}_i + \mathbf{u})$, and in fact are often computed as a side-product of image interpolation. If efficiency is a concern, these gradients can be replaced by the gradients in the *template* image,

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \approx \mathbf{J}_0(\mathbf{x}), \quad (1.12)$$

since near the correct alignment, the template and displaced target images should look similar. This has the advantage of allowing the pre-computation of the Hessian and Jacobian images, which can result in significant computational savings [2].

Parametric motion

Many image alignment tasks, for example image stitching with handheld cameras, require the use of more sophisticated motion models. Since these models typically have more parameters than pure translation, a full search over the possible range of values is impractical. Instead, the incremental Lucas-Kanade algorithm can be generalized to parametric motion models and used in conjunction with a hierarchical search algorithm [16, 4, 2].

For parametric motion, instead of using a single constant translation vector \mathbf{u} , we use a spatially varying *motion field* or *correspondence map*, $\mathbf{x}'(\mathbf{x}; \mathbf{p})$, parameterized by a low-dimensional vector \mathbf{p} , where \mathbf{x}' can be any of the motion models presented in Section 2. The parametric incremental motion update rule now becomes

$$\begin{aligned} E_{\text{LK-PM}}(\mathbf{p} + \Delta\mathbf{p}) &= \sum_i [I_1(\mathbf{x}'(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p})) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i [\mathbf{J}_1(\mathbf{x}'_i) \Delta\mathbf{p} + e_i]^2, \end{aligned} \quad (1.13)$$

where the Jacobian is now

$$\mathbf{J}_1(\mathbf{x}'_i) = \frac{\partial I_1}{\partial \mathbf{p}} = \nabla I_1(\mathbf{x}'_i) \frac{\partial \mathbf{x}'}{\partial \mathbf{p}}(\mathbf{x}_i), \quad (1.14)$$

i.e., the product of the image gradient ∇I_1 with the Jacobian of correspondence field, $\mathbf{J}_{\mathbf{x}'} = \partial \mathbf{x}' / \partial \mathbf{p}$.

The derivatives required to compute the Jacobian can be derived directly from Table 1.1 and are given in [28].

The computation of the Hessian and residual vectors for parametric motion can be significantly more expensive than for the translational case. For parametric motion with n parameters and N pixels, the accumulation of \mathbf{A} and \mathbf{b} takes $O(n^2N)$ operations [2]. One way to reduce this by a significant amount is to divide the image up into smaller sub-blocks (patches) P_j and to only accumulate the simpler 2×2 quantities (1.11) at the pixel level [25, 2, 28].

For a complex parametric motion such as a homography, the computation of the motion Jacobian becomes complicated, and may involve a per-pixel division. Szeliski and Shum [30] observed that this can be simplified by first warping the target image I_1 according to the current motion estimate $\mathbf{x}'(\mathbf{x}; \mathbf{p})$ and then comparing this *warped* image against the template $I_0(\mathbf{x})$. Baker and Matthews [2] call this the *forward compositional* algorithm, since the target image is being re-warped, and the final motion estimates are being composed, and also present an *inverse compositional* algorithm that is even more efficient.

3.2 Feature-based registration

As mentioned earlier, directly matching pixel intensities is just one possible approach to image registration. The other major approach is to first extract distinctive *features* from each image, to match individual features to establish a global correspondence, and to then estimate the geometric transformation between the images. This kind of approach has been used since the early days of stereo matching and has more recently gained popularity for image stitching applications [8, 6].

Schmid *et al.* [24] survey the vast literature on interest point detection and perform some experimental comparisons to determine the *repeatability* of feature detectors. They also measure the *information content* available at each detected feature point. Among the techniques they survey, they find that an improved version of the Harris operator works best.

More recently, feature detectors that are more invariant to scale [15] and affine transformations have been proposed. These can be very useful when matching images that have different scales or different aspects (e.g., for 3D object recognition).

After detecting the features (interest points), we must *match* them, i.e., determine which features come from corresponding locations in different images. In some situations, e.g., for video sequences or for stereo pairs that have been *rectified*, the local motion around each feature point may be mostly translational. In this case, the error metrics introduced previously can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid [19] discussed below uses cross-correlation.)

If features are being tracked over longer image sequences, their appearance can undergo larger changes. In this case, it makes sense to compare appearances using an *affine* motion model. Because the features can appear at different orientations or scales, a more *view invariant* kind of representation must be used. Mikolajczyk and Schmid [19] review some recently developed view-invariant local image descriptors and experimentally compare their performance.

The simplest method to compensate for in-plane rotations is to find a *dominant orientation* at each feature point location before sampling the patch or otherwise computing the descriptor. Mikolajczyk and Schmid use the direction of the average gradient orientation, computed within a small neighborhood of each feature point. The descriptor can be made invariant to scale by only selecting feature points that are local maxima in scale space. Among the local descriptors that Mikolajczyk and Schmid compared, David Lowe's Scale Invariant Feature Transform (SIFT) [15] performed the best.

The simplest way to find all corresponding feature points in an image pair is to compare all features in one image against all features in the other, using one of the local descriptors described above. Unfortunately, this is quadratic in the expected number of features, which makes it impractical

for some applications. More efficient matching algorithms can be devised using different kinds of *indexing schemes*, many of which are based on the idea of finding nearest neighbors in high-dimensional spaces.

Once an initial set of feature correspondences has been computed, we need to find a set that will produce a high-accuracy alignment. One possible approach is to simply compute a least squares estimate, or to use a robustified version of least squares. However, in many cases, it is better to first find a good starting set of *inlier* correspondences, i.e., points that are all consistent with some particular motion estimate. Two widely used solutions to this problem are RANdom SAMple Consensus (RANSAC) and *least median of squares* (LMS) [26]. Both techniques start by selecting a random subset of k correspondences, which is then used to compute a motion estimate \mathbf{p} . The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location. Least median of squares finds the median value of the $\|\mathbf{r}_i\|$ values. The random selection process is repeated S times, and the sample set with largest number of inliers (or with the smallest median residual) is kept as the final solution.

Geometric registration

Once we have computed a set of matched feature point correspondences, we still need to estimate the motion parameters \mathbf{p} that best register the two images. The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals given by

$$E_{\text{LS}} = \sum_i \|\mathbf{r}_i\|^2 = \|\tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i\|^2, \quad (1.15)$$

where $\tilde{\mathbf{x}}'_i$ are the *estimated* (mapped) locations, and $\hat{\mathbf{x}}'_i$ are the sensed (detected) feature point locations corresponding to point \mathbf{x}_i in the other image.

Many of the motion models presented in Section 2, i.e., translation, similarity, and affine, have a *linear* relationship between the motion and the unknown parameters \mathbf{p} . In this case, a simple linear regression (least squares) using normal equations works well.

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 in more textured regions than others. If we associate a variance estimate σ_i^2 with each correspondence, we can minimize *weighted least squares* instead,

$$E_{\text{WLS}} = \sum_i \sigma_i^{-2} \|\mathbf{r}_i\|^2. \quad (1.16)$$

As discussed in [28], a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the Hessian with the per-pixel noise estimate. Weighting each squared residual by the inverse covariance

$\Sigma_i^{-1} = \sigma_n^{-2} \mathbf{A}_i$ (which is called the *information matrix*), we obtain

$$E_{\text{CWLS}} = \sum_i \|\mathbf{r}_i\|_{\Sigma_i^{-1}}^2 = \sum_i \mathbf{r}_i^T \Sigma_i^{-1} \mathbf{r}_i = \sum_i \sigma_n^{-2} \mathbf{r}_i^T \mathbf{A}_i \mathbf{r}_i, \quad (1.17)$$

where \mathbf{A}_i is the *patch Hessian*.

If there are outliers among the feature-based correspondences, it is better to use a robust version of least squares, even if an initial RANSAC or MLS stage has been used to select plausible inliers. The robust least squares cost metric is then

$$E_{\text{RLS}}(\mathbf{u}) = \sum_i \rho(\|\mathbf{r}_i\|_{\Sigma_i^{-1}}). \quad (1.18)$$

For motion models that are not linear in the motion parameters, *non-linear least squares* must be used instead. Deriving the Jacobian of each residual equation with respect to the motion parameters is relatively straightforward, once a suitable parameterization has been chosen [28].

3.3 Direct vs. feature-based

Given that there are these two alternative approaches to aligning images, which is preferable?

My original work in image stitching was firmly in the direct (image-based) camp [27, 30, 25]. Early feature-based methods seemed to get confused in regions that were either too textured or not textured enough. The features would often be distributed unevenly over the images, thereby failing to match image pairs that should have been aligned. Furthermore, establishing correspondences relied on simple cross-correlation between patches surrounding the feature points, which did not work well when the images were rotated or had foreshortening due to homographies.

Today, feature detection and matching schemes are remarkably robust and can even be used for known object recognition from widely separated views [15]. Because they operate in scale-space and use a dominant orientation (or orientation invariant descriptors), they can match images that differ in scale, orientation, and even foreshortening. My own recent experience is that if the features are well distributed over the image and the descriptors reasonably designed for repeatability, enough correspondences to permit image stitching can usually be found.

The other major reason I used to prefer direct methods was that they make optimal use of the information available in image alignment, since they measure the contribution of *every* pixel in the image. Furthermore, assuming a Gaussian noise model (or a robustified version of it), they properly weight the contribution of different pixels, e.g., by emphasizing the contribution of high-gradient pixels. (See Baker *et al.* [2], who suggest that adding even more weight at strong gradients is preferable because of noise in the gradient estimates.)

The biggest disadvantage of direct techniques is that they have a limited range of convergence. Even though hierarchical (coarse-to-fine) techniques can help, it is hard to use more than two or three levels of a pyramid before important details start get blurred. For matching sequential frames in a video, the direct approach can usually be made to work. However, for matching partially overlapping images in photo-based panoramas, they fail too often to be useful.

Is there no rôle then for direct registration? I believe there is. Once a pair of images has been aligned with a feature-based approach, we can warp the two images to a common reference frame and re-compute a more accurate correspondence using patch-based alignment. Notice how there is a close correspondence between the patch-based approximation to direct alignment and the inverse covariance weighted feature-based least squares error metric (1.17).

In fact, if we divide the template images up into patches and place an imaginary “feature point” at the center of each patch, the two approaches return exactly the same answer (assuming that the correct correspondences are found in each case). However, for this approach to succeed, we still have to deal with “outliers”, i.e., regions that do not fit the selected motion model due to either parallax or moving objects. While a feature-based approach may make it somewhat easier to reason about outliers (features can be classified as inliers or outliers), the patch-based approach, since it establishes correspondences more densely, is potentially more useful for removing local mis-registration (parallax).

4 Global registration

So far, I have discussed how to register pairs of images using both direct and feature-based methods. In most applications, we are given more than a single pair of images to register. The goal is to find a *globally consistent* set of alignment parameters that minimize the mis-registration between all pairs of images [30, 25, 23]. In order to do this, we need to extend the pairwise matching criteria to a global energy function that involves all of the per-image pose parameters. Once we have computed the global alignment, we need to perform *local adjustments* such as *parallax removal* to reduce double images and blurring due to local mis-registration. 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.

4.1 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 previ-

ous ones already in the collection [30], and discovering, if necessary, which images it overlaps [23]. 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* [30]. However, a better alternative is to simultaneously align all the images together using a least squares framework to evenly distribute any mis-registration errors.

The process of simultaneously adjusting pose parameters for a large collection of overlapping images is called *bundle adjustment* in the photogrammetry community [31]. In computer vision, it was first applied to the general structure from motion problem [29], and then later specialized for panoramic image stitching [25, 23].

In this section, I 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 by dividing images into patches and creating a virtual feature correspondence for each one [25].

Consider the feature-based alignment problem given in (1.15). 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 i th feature point in the j th image denoted by \mathbf{x}_{ij} and its scalar confidence (inverse variance) denoted by c_{ij} . Each image also has some associated *pose* parameters.

In this section, I 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 [30, 23]. The equation mapping a 3D point \mathbf{x}_i into a point \mathbf{x}_{ij} in frame j can be re-written from (1.4–1.6) as

$$\mathbf{x}_{ij} \sim \mathbf{K}_j \mathbf{R}_j \mathbf{x}_i \quad \text{and} \quad \mathbf{x}_i \sim \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \mathbf{x}_{ij}, \quad (1.19)$$

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

$$\mathbf{x}_{ik} \sim \mathbf{H}_{kj} \mathbf{x}_{ij} = \mathbf{K}_k \mathbf{R}_k \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \mathbf{x}_{ij}. \quad (1.20)$$

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 to a multiview formulation,

$$E_{\text{all-pairs-2D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_{ik}(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j, \mathbf{R}_k, f_k) - \hat{\mathbf{x}}_{ik}\|^2, \quad (1.21)$$

where the $\tilde{\mathbf{x}}_{ik}$ function is the *predicted* location of feature i in frame k given by (1.20), $\hat{\mathbf{x}}_{ij}$ is the *observed* location, and the “2D” in the subscript indicates that an image-plane error is being minimized.

While this approach works well in practice, it suffers from two potential disadvantages. First, since a summation is taken over all pairs with corresponding features, features that are observed many times get overweighted in the final solution. Second, the derivatives of $\tilde{\mathbf{x}}_{ik}$ w.r.t. the $\{(\mathbf{R}_j, f_j)\}$ are a little cumbersome.

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 $\{\mathbf{x}_i\}$,

$$E_{\text{BA-2D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j) - \hat{\mathbf{x}}_{ij}\|^2, \quad (1.22)$$

where $\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j)$ is given by (1.19). The disadvantage of full bundle adjustment is that there are more variables to solve for, so both each iteration and the overall convergence may be slower. However, the computational complexity of each linearized Gauss-Newton step can be reduced using sparse matrix techniques [29, 25, 31].

An alternative formulation is to minimize the error in 3D projected ray directions [25], i.e.,

$$E_{\text{BA-3D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \mathbf{x}_i\|^2, \quad (1.23)$$

where $\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j)$ is given by the second half of (1.19).

However, if we eliminate the 3D rays \mathbf{x}_i , we can derive a pairwise energy formulated in 3D ray space [25],

$$E_{\text{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ik}; \mathbf{R}_k, f_k)\|^2. \quad (1.24)$$

This results in the simplest set of update equations [25], since the f_k can be folded into the creation of the homogeneous coordinate vector. Thus, even though this formula over-weights features that occur more frequently, it is the method used both by Shum and Szeliski [25] and in my current feature-based aligner. In order to reduce the bias towards longer focal lengths, I 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.

4.2 Parallax removal

Once we have estimated 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 may be caused by a variety of factors, including unmodeled radial distortion, 3D parallax (failure to rotate the camera around its optical center), small



FIGURE 2. A set of images and the panorama discovered in them

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 using one of several classic calibration techniques. 3D parallax can be attacked by doing a full 3D bundle adjustment. The 3D positions of the matched features points and cameras can then be simultaneously recovered, although this can be significantly more expensive than parallax-free image registration.

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 [10, 1], as discussed in Section 6. However, when the motion is reasonably small (on the order of a few pixels), general 2-D motion estimation (optic flow) can be used to perform an appropriate correction before blending using a process called *local alignment* [25]. 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.

4.3 Recognizing panoramas

The final piece needed to perform fully automated image stitching is a technique to determine which images actually go together, which Brown and Lowe call *recognizing panoramas* [6]. If the user takes images in sequence so that each image overlaps its predecessor, bundle adjustment combined with the process of *topology inference* can be used to automatically assemble a panorama [23]. However, users often jump around when taking panoramas, e.g., they may start a new row on top of a previous one, or jump back to take a repeated shot, or create 360° panoramas where end-to-end overlaps need to be discovered. Furthermore, the ability to automatically discover multiple panoramas taken by a user can be a big convenience.

To recognize panoramas, Brown and Lowe [6] 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 2). First, they use Lowe’s Scale Invariant Feature Transform (SIFT features) [15] followed by nearest neighbor matching. RANSAC is then used to find a set of *inliers*, using pairs of matches to hypothesize similarity motion estimates. Once pairwise alignments have been computed, a global registration (bundle adjustment) stage is used to compute a globally consistent alignment for all of the images. Finally, a two-level Laplacian pyramid is used to seamlessly blend the images [6].

5 Choosing a compositing surface

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, e.g., flat, cylindrical, or spherical. It may also involve computing an optimal *reference view* to ensure that the scene appears to be *upright*, as described in [28].

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 the reference coordinate system. The resulting composite is called a *flat* panorama, since the projection onto the final surface is still a perspective projection, and hence straight lines remain straight.

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 [9] or spherical [30] projection. In fact, any surface used for *environment mapping* in computer graphics can be used, including a *cube map* that represents the full viewing sphere with the six square faces of a box [30].

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 an interesting topic for future research.

6 Seam selection and pixel blending

Once the source pixels have been mapped onto the final composite surface, we must 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 (any pixel combination will do). However,

for real images, visible seams (due to exposure differences), blurring (due to mis-registration), 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, I 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. However, this usually does not work very well, since exposure differences, mis-registrations, and scene movement are all very visible (Figure 3a). 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 [12].

A better approach 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 pixels near the edges of both cutouts and edges is preferable. This can be done by computing a *distance map* or *grass-fire transform*, where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel. Weighted averaging with a distance map is often called *feathering* [30, 32], and does a reasonable job of blending over exposure differences. However, blurring and ghosting can still be problems (Figure 3b).

One way to improve feathering is to raise the distance map values to some power. The weighted averages then become dominated by the larger values, i.e., they act like a *p-norm*. The resulting composite can often provide a reasonable tradeoff between visible exposure differences and blur.

In the limit as $p \rightarrow \infty$, only the pixel with the maximum distance value gets selected, which is equivalent to computing the *Vornoi diagram*. 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.

Optimal seam selection

Computing the Vornoi diagram is one way to select the *seams* between regions where different images contribute to the final composite. However, Vornoi 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 [10]. For a pair of images, this process can be formulated as a simple dynamic program starting from one edge of

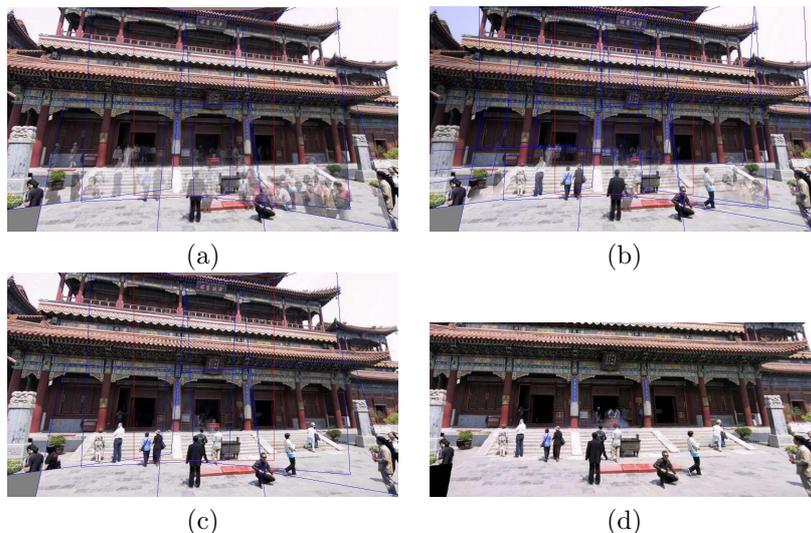


FIGURE 3. Final composites computed by a variety of algorithms: (a) average, (b) feathered average, (c) weighted ROD vertex cover with feathering, (d) graph cut seams with Poisson blending. Notice how the regular average cuts off moving people near the edges of images, while the feathered average slowly blends them in. The vertex cover and graph cut algorithms produce similar results.

the overlap region and ending at the other [20, 10]. Unfortunately, when multiple images are being composited, the dynamic program idea does not readily generalize.

To overcome this problem, Uyttendaele *et al.* [32] 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. 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 unremoved 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. 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. Once the required regions of difference have been removed, the final composite is created using a feathered blend (Figure 3c).

A different approach to pixel selection and seam placement was recently proposed by Agarwala *et al.* [1]. Their system computes the label assign-

ment that optimizes the sum of two objective functions. The first is a per-pixel *image objective* \mathcal{C}_D that determines which pixels are likely to produce good composites. In their system, users can select which pixels to use by “painting” over an image with the desired object or appearance. Alternatively, automated selection criteria can be used, such as *maximum likelihood* that prefers pixels which occur repeatedly (for object removal), or *minimum likelihood* for objects that occur infrequently (for greatest object retention).

The second term is a *seam objective* \mathcal{C}_S that penalizes differences in labelings between adjacent images. For example, the simple color-based seam penalty used in [1] measures the color difference between corresponding pixels on both sides of the seam. The global energy function that is the sum of the data and seam costs can be minimized using a variety of techniques [28]. Agarwala *et al.* [1] use graph cuts, which involves cycling through a set of simpler α -*expansion* re-labelings, each of which can be solved with a graph cut (max-flow) polynomial-time algorithm [5].

For the result shown in Figure 3d, Agarwala *et al.* [1] use a large data penalty for invalid pixels and 0 for valid pixels. Notice how the seam placement algorithm avoids regions of differences, including those that border the image and which might result in cut off objects. Graph cuts [1] and vertex cover [32] 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.

Laplacian pyramid blending

Once the seams have been placed and unwanted objects removed, we still need to blend the images to compensate for exposure differences and other mis-alignments. An attractive solution to this problem was developed by Burt and Adelson [7]. 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 a function of the pyramid level. First, each warped image is converted into a band-pass (Laplacian) pyramid. Next, the *masks* associated with each source image are converted into a low-pass (Gaussian) pyramid and used to perform a per-level feathered blend of the band-pass images. Finally, the composite image is reconstructed by interpolating and summing all of the pyramid levels (band-pass images).

Gradient domain blending

An alternative approach to multi-band image blending is to perform the operations in the *gradient domain*. Here, instead of working with the initial color values, the image gradients from each source image are copied; in a second pass, an image that best matches these gradients is reconstructed [1]. Copying gradients directly from the source images after seam placement is just one approach to gradient domain blending. Levin *et al.* [14] examine

several different variants on 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 L1 norm in performing the reconstruction of the image from the gradient field, rather than using an L2 norm. Their preferred technique is the L1 optimization of a feathered (blended) cost function on the original image gradients (which they call GIST1- l_1). While L1 optimization using linear programming can be slow, a faster iterative median-based algorithm in a multigrid framework works well in practice. Visual comparisons between their preferred approach and what they call *optimal seam on the gradients* (which is equivalent to Agarwala *et al.*'s approach [1]) 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 *et al.* [32] 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* between neighboring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed, and the process is repeated (typically 3 times). The results in [32] demonstrate that this does a better job of exposure compensation than simple feathering and can handle local variations in exposure due to effects like lens vignetting.

High dynamic range imaging

A more principled approach is to estimate a single *high dynamic range* (HDR) radiance map from the differently exposed images [11, 21]. This approach assumes that the input images were taken with a fixed camera whose pixel values are the result of applying a parameterized *radiometric transfer function* $f(R, \mathbf{p})$ to scaled radiance values $c_k R(\mathbf{x})$. The exposure values c_k are either known (by experimental setup, or from a camera's EXIF tags), or are computed as part of the parameter estimation process. After the transfer function has been estimated, radiance values from different exposures can be combined to emphasize reliable pixels.

Once a radiance map has been computed, it is usually necessary to display it on a lower gamut (i.e., 8-bit) screen or printer. A variety of *tone mapping* techniques have been developed for this purpose, which involve either computing spatially varying transfer functions or reducing image

gradients to fit the the available dynamic range.

Unfortunately, most casually acquired images may not be perfectly registered and may contain moving objects. Kang *et al.* [13] present an algorithm that combines global registration with local motion estimation (optic flow) to accurately align the images before blending their radiance estimates. Since the images may have widely different exposures, care must be taken when producing the motion estimates, which must themselves be checked for consistency to avoid the creation of ghosts and object fragments.

7 Extensions and open issues

While image stitching has now reached a point where it is commonly used in consumer photo editing products, there are still a lot of open research problems that need to be addressed.

The first of these is improving the reliability of fully automated stitching. Whenever images contain small amounts of overlap, repeated textures, or large regions of difference because of moving objects, it becomes increasingly difficult to disambiguate between accidental and correct alignments. Global reasoning about a compatible set of correspondences might be the solution, as might be improvements in robust (partial) feature matching.

Dealing with motion and parallax is another important area, since pictures are often taken with handheld cameras in highly dynamic situations. At some point, full 3D reconstruction with moving object detection and layer extraction may be required, which also raises interesting issues in designing quick and easy user interfaces to specify the desired final output.

Dealing with images at different resolutions and zoom factors is another interesting area, especially since variable resolution image representations and viewers are not common. A related issue is super-resolution, i.e., enhancing image resolution through the combination of jittered photographs of the same region [17, 8]. Unfortunately, because of limitations in optics and motion estimation, there seems to be a very limited ($< 2\times$) improvement that can be achieved in practice.

Stitching videos is another area that is likely to grow as more digital cameras start to include the ability to take videos. Examples of stitching videos to obtain summary panoramas have been around for a while [12, 22]. In the future, we are likely to see the construction of “live” panoramas that include moving elements along with still portions [27].

Ultimately, image alignment and stitching will become part of a repertoire of computer vision algorithms used to merge multiple images (with different orientations, exposures, and other attributes) to create enhanced and innovative composite pictures and photographic experiences.

8 REFERENCES

- [1] A. Agarwala et al. Interactive digital photomontage. *ACM Transactions on Graphics*, 23(3):292–300, August 2004.
- [2] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework: Part 1: The quantity approximated, the warp update rule, and the gradient descent approximation. *International Journal of Computer Vision*, 56(3):221–255, March 2004.
- [3] R. Benosman and S. B. Kang, editors. *Panoramic Vision: Sensors, Theory, and Applications*, New York, 2001. Springer.
- [4] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Second European Conference on Computer Vision*, pages 237–252, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.
- [5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.
- [6] M. Brown and D. Lowe. Recognizing panoramas. In *Ninth International Conference on Computer Vision*, pages 1218–1225, Nice, France, October 2003.
- [7] P. J. Burt and E. H. Adelson. A multiresolution spline with applications to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, October 1983.
- [8] D. Capel and A. Zisserman. Automated mosaicing with super-resolution zoom. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 885–891, Santa Barbara, June 1998.
- [9] S. E. Chen. QuickTime VR – an image-based approach to virtual environment navigation. *Computer Graphics (SIGGRAPH'95)*, pages 29–38, August 1995.
- [10] J. Davis. Mosaics of scenes with moving objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 354–360, Santa Barbara, June 1998.
- [11] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. *Proceedings of SIGGRAPH 97*, pages 369–378, August 1997.
- [12] M. Irani and P. Anandan. Video indexing based on mosaic representations. *Proceedings of the IEEE*, 86(5):905–921, May 1998.

- [13] S. B. Kang et al. High dynamic range video. *ACM Transactions on Graphics*, 22(3):319–325, July 2003.
- [14] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision*, volume IV, pages 377–389, Prague, May 2004.
- [15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.
- [16] B. D. Lucas and T. Kanade. An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence*, pages 674–679, Vancouver, 1981.
- [17] S. Mann and R. W. Picard. Virtual bellows: Constructing high-quality images from video. In *First IEEE International Conference on Image Processing*, volume I, pages 363–367, Austin, November 1994.
- [18] J. Meehan. *Panoramic Photography*. Watson-Guptill, 1990.
- [19] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume II, pages 257–263, Madison, WI, June 2003.
- [20] D. L. Milgram. Computer methods for creating photomosaics. *IEEE Transactions on Computers*, C-24(11):1113–1119, November 1975.
- [21] T. Mitsunaga and S. K. Nayar. Radiometric self calibration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 374–380, Fort Collins, June 1999.
- [22] H. S. Sawhney et al. Videobrush: Experiences with consumer video mosaicing. In *IEEE Workshop on Applications of Computer Vision*, pages 56–62, Princeton, October 1998.
- [23] H. S. Sawhney and R. Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3):235–243, March 1999.
- [24] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, June 2000.
- [25] H.-Y. Shum and R. Szeliski. Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2):101–130, February 2000. Erratum published July 2002, 48(2):151-152.

- [26] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Reviews*, 41(3):513–537, September 1999.
- [27] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, March 1996.
- [28] R. Szeliski. Image alignment and stitching: A tutorial. Technical Report MSR-TR-2004-92, Microsoft Research, December 2004.
- [29] R. Szeliski and S. B. Kang. Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.
- [30] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and texture-mapped models. *Computer Graphics (SIG-GRAPH'97 Proceedings)*, pages 251–258, August 1997.
- [31] B. Triggs et al. Bundle adjustment — a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372, Kerkyra, Greece, September 1999.
- [32] M. Uyttendaele, A. Eden, and R. Szeliski. Eliminating ghosting and exposure artifacts in image mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume II, pages 509–516, Kauai, Hawaii, December 2001.