Image Alignment and Stitching: A Tutorial[1]

Richard Szeliski

Preliminary draft, January 26, 2005
Technical Report
MSR-TR-2004-92

This tutorial reviews image alignment and image stitching algorithms. Image alignment algorithms can discover the correspondence relationships among images with varying degrees of overlap. They are ideally suited for applications such as video stabilization, summarization, and the creation of panoramic photographs. Image stitching algorithms take the alignment estimates produced by such registration algorithms and blend the images in a seamless manner, taking care to deal with potential problems such as blurring or ghosting caused by parallax and scene movement as well as varying image exposures. This tutorial reviews the basic motion models underlying alignment and stitching algorithms, describes effective direct (pixel-based) and feature-based alignment algorithms, and describes blending algorithms used to produce seamless mosaics. It closes with a discussion of open research problems in the area.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

[1] A shorter version of this report will appear in *Mathematical Models of Computer Vision: The Handbook*.

Contents

1 Introduction

[Note: Change /date command above once final draft is ready...]

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision. Frame-rate image alignment is used in every camcorder that has an "image stabilization" feature. Image stitching algorithms create the high-resolution photo-mosaics used to produce today's digital maps and satellite photos. They also come "out of the box" with every digital camera currently being sold, and can be used to create beautiful ultra wide-angle panoramas.

An early example of a widely-used image registration algorithm is the patch-based translational alignment (optical flow) technique developed by Lucas and Kanade (1981). Variants of this algorithm are used in almost all motion-compensated video compression schemes such as MPEG and H.263 (Le Gall 1991). Similar parametric motion estimation algorithms have found a wide variety of applications, including video summarization (Bergen *et al.* 1992, Teodosio and Bender 1993, Kumar *et al.* 1995, Irani and Anandan 1998), video stabilization (Hansen *et al.* 1994), and video compression (Irani *et al.* 1995, Lee *et al.* 1997).

In the photogrammetry community, 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 that could simultaneously solve for the locations of all of the camera positions, thus yielding globally consistent solutions (Triggs *et al.* 1999). One of the recurring problems in creating photo-mosaics is the elimination of visible seams, for which a variety of techniques have been developed over the years (Milgram 1975, Milgram 1977, Peleg 1981, Davis 1998, Agarwala *et al.* 2004)

In film photography, special cameras were developed at the turn of the century 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, Szeliski 1994, Chen 1995, Szeliski 1996). More recent work in this area has addressed the need to compute globally consistent alignments (Szeliski and Shum 1997, Sawhney and Kumar 1999, Shum and Szeliski 2000), the removal of "ghosts" due to parallax and object movement (Davis 1998, Shum and Szeliski 2000, Uyttendaele *et al.* 2001, Agarwala *et al.* 2004), and dealing with varying exposures (Mann and Picard 1994, Uyttendaele *et al.* 2001, Agarwala *et al.* 2004). (A collection of some of these papers can be found in (Benosman and Kang 2001).) These techniques have spawned a large number of commercial stitching products (Chen 1995, Sawhney *et al.* 1998), for which reviews and comparison can be found on the Web.

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 (Zoghlami *et al.* 1997, Capel and Zisserman 1998, Cham and Cipolla 1998, Badra *et al.* 1998, McLauchlan and Jaenicke 2002, Brown and Lowe 2003). Feature-based approaches have the advantage of being more robust against scene movement and are potentially faster, if implemented the right way. 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 2003).

What, then, are the essential problems in image alignment and stitching? For image alignment, we must first determine the appropriate mathematical model relating pixel coordinates in one image to pixel coordinates in another. Section 2 reviews these basic *motion models*. Next, we must somehow estimate the correct alignments relating various pairs (or collections) of images. Section 3 discusses how *direct* pixel-to-pixel comparisons combined with gradient descent (and other optimization techniques) can be used to estimate these parameters. Section 4 discusses how distinctive *features* can be found in each image and then efficiently matched to rapidly establish correspondences between pairs of images. When multiple images exist in a panorama, techniques must be developed to compute a globally consistent set of alignments and to efficiently discover which images overlap one another. These issues are discussed in Section 5.

For image stitching, we must first choose a final compositing surface onto which to warp and place all of the aligned images (Section 6). We also need to develop algorithms 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 survey, I discuss additional applications of image stitching and open research problems.

2 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 the mapping to non-planar (e.g., cylindrical) surfaces (Szeliski 1996).

To facilitate working with images at different resolutions, we adopt a variant of the *normalized device coordinates* used in computer graphics (Watt 1995, OpenGL ARB 1997). For a typical (rectangular) image or video frame, we let the pixel coordinates range from $[-1, 1]$ along the longer axis, and $[-a, a]$ along the shorter, where a is the inverse of the *aspect ratio*.[1] For an image with

[1] In computer graphics, it is usual to have both axes range from $[-1, 1]$, but this requires the use of two different focal lengths for the vertical and horizontal dimensions, and makes it more awkward to handle mixed portrait and

width W and height H, the equations mapping integer pixel coordinates $\overline{\boldsymbol{x}} = (\overline{x}, \overline{y})$ to normalized device coordinates $\boldsymbol{x} = (x, y)$ are

$$x = \frac{2\overline{x} - W}{S} \quad \text{and} \quad y = \frac{2\overline{y} - H}{S} \qquad \text{where} \qquad S = \max(W, H). \tag{1}$$

Note that if we work with images in a *pyramid*, we need to halve the S value after each decimation step rather than recomputing it from $\max(W, H)$, since the (W, H) values may get rounded or truncated in an unpredictable manner. For the rest of this paper, we use normalized device coordinates when refering to pixel coordinates.

2.1 2D (planar) motions

Having defined our coordinate system, we can now describe how coordinates are transformed. The simplest transformations occur in the 2D plane and are illustrated in Figure 1.

Translation. 2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \tilde{\boldsymbol{x}} \tag{2}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix and $\tilde{\boldsymbol{x}} = (x, y, 1)$ is the *homogeneous* or *projective* 2D coordinate.

Rotation + translation. This transformation is also known as *2D rigid body motion* or the *2D Euclidean transformation* (since Euclidean distances are preserved). It can be written as $\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \tilde{\boldsymbol{x}} \tag{3}$$

where

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4}$$

is an orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$.

Scaled rotation. Also known as the *similarity transform*, this transform can be expressed as $\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ where s is an arbitrary scale factor. It can also be written as

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \tilde{\boldsymbol{x}} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \tilde{\boldsymbol{x}}, \tag{5}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

landscape mode images.

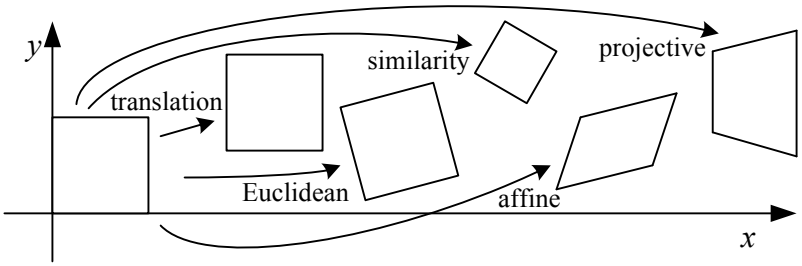


Figure 1: *Basic set of 2D planar transformations*

Affine. The affine transform is written as $\boldsymbol{x}' = \boldsymbol{A}\tilde{\boldsymbol{x}}$, where $\boldsymbol{A}$ is an arbitrary 2×3 matrix, i.e.,

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \tilde{\boldsymbol{x}}. \tag{6}$$

Parallel lines remain parallel under affine transformations.

Projective. This transform, also known as a *perspective transform* or *homography*, operates on homogeneous coordinates $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}'$,

$$\tilde{\boldsymbol{x}}' \sim \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}, \tag{7}$$

where $\sim$ denotes equality up to scale and $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix. Note that $\tilde{\boldsymbol{H}}$ is itself homogeneous, i.e., it is only defined up to a scale. The resulting homogeneous coordinate $\tilde{\boldsymbol{x}}'$ must be normalized in order to obtain an inhomogeneous result $\boldsymbol{x}'$, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \tag{8}$$

Perspective transformations preserve straight lines.

Hierarchy of 2D transformations The preceding set of transformations are illustrated in Figure 1 and summarized in Table 1. The easiest way to think of these is as a set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors. Hartley and Zisserman (2004) contains a more detailed description of the hierarchy of 2D planar transformations.

The above transformations form a nested set of *groups*, i.e., they are closed under composition and have an inverse that is a member of the same group. Each (simpler) group is a subset of the more complex group below it.

2.2 3D transformations

A similar nested hierarchy exists for 3D coordinate transformations that can be denoted using 4×4 transformation matrices, with 3D equivalents to translation, rigid body (Euclidean) and

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$	8	straight lines	

Table 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.*

affine transformations, and homographies (sometimes called collineations) (Hartley and Zisserman 2004).

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

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}, \tag{9}$$

as shown in Figure 2. The relationship between the (unit-less) focal length f and the field of view θ is given by

$$f^{-1} = \tan\frac{\theta}{2} \quad \text{or} \quad \theta = 2\tan^{-1}\frac{1}{f}. \tag{10}$$

To convert the focal length f to its more commonly used 35mm equivalent, multiply the above number by 17.5 (the half-width of a 35mm photo negative frame). To convert it to pixel coordinates, multiply it by $S/2$ (half-width for a landscape photo).

In the computer graphics literature, perspective projection is often written as a permutation matrix that permutes the last two elements of homogeneous 4-vector $\boldsymbol{p} = (x, y, z, 1)$,

$$\tilde{\boldsymbol{x}} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{p}, \tag{11}$$

followed by a scaling and translation into screen and *z-buffer* coordinates.

In computer vision, it is traditional to drop the z-buffer values, since these cannot be sensed in

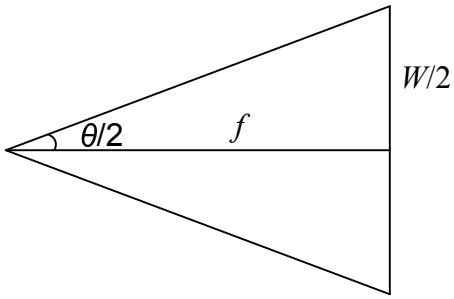


Figure 2: *Central projection, showing the relationship between the 3D and 2D coordinates $\boldsymbol{p}$ and $\boldsymbol{x}$, as well as the relationship between the focal length f W and the field of view θ.* [Note: Re-generate the figures from Formation-figs1.vsd. Colin says I need a better figure here...]

an image and to write

$$\tilde{\boldsymbol{x}} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{p} = \left[\begin{array}{c|c} \boldsymbol{K} & \boldsymbol{0} \end{array} \right] \boldsymbol{p} \tag{12}$$

where $\boldsymbol{K} = \mathrm{diag}(f, f, 1)$ is called the *intrinsic calibration* matrix.[2] This matrix can be replaced by a more general upper-triangular matrix $\boldsymbol{K}$ that accounts for non-square pixels, skew, and a variable optic center location (Hartley and Zisserman 2004). However, in practice, the simple focal length scaling used above provides high-quality results when stitching images from regular cameras.

In this paper, I prefer to use a 4×4 *projection matrix*, $\boldsymbol{P}$,

$$\tilde{\boldsymbol{x}} \sim \left[\begin{array}{c|c} \boldsymbol{K} & \boldsymbol{0} \\ \hline \boldsymbol{0}^T & 1 \end{array} \right] \boldsymbol{p} = \boldsymbol{P}\boldsymbol{p}, \tag{13}$$

which maps the homogeneous 4-vector $\boldsymbol{p} = (X, Y, Z, 1)$ to a special kind of homogemeous *screen vector* $\tilde{\boldsymbol{x}} = (x, y, 1, z)$. This allows me to denote the upper-left 3×3 portion of the projection matrix $\boldsymbol{P}$ as $\boldsymbol{K}$ (making it compatible with the computer vision literature), while not dropping altogether the inverse screen depth information d (which is alo sometimes called the *disparity* d (Okutomi and Kanade 1993)). This latter quantity is necessary to reason about mappings between images of a 3D scene, as described below.

What happens when we take two images of a 3D scene from different camera positions and/or orientations? A 3D point $\boldsymbol{p}$ gets mapped to an image coordinate $\tilde{\boldsymbol{x}}_0$ in camera 0 through the combination of a 3D rigid-body (Euclidean) motion $\boldsymbol{E}_0$,

$$\boldsymbol{x}_0 = \begin{bmatrix} \boldsymbol{R}_0 & \boldsymbol{t}_0 \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{p} = \boldsymbol{E}_0 \boldsymbol{p}, \tag{14}$$

[2] The last column of $\boldsymbol{K}$ usually contains the optical center (c_x, c_y), but this can be set to zero if we use normalized device coordinates.

and a perspective projection $\boldsymbol{P}_0$,

$$\tilde{\boldsymbol{x}}_0 \sim \boldsymbol{P}_0 \boldsymbol{E}_0 \boldsymbol{p}. \tag{15}$$

Assuming that we know the z-buffer value z_0 for a pixel in one image, we can map it back to the 3D coordinate $\boldsymbol{p}$ using

$$\boldsymbol{p} \sim \boldsymbol{E}_0^{-1} \boldsymbol{P}_0^{-1} \tilde{\boldsymbol{x}}_0 \tag{16}$$

and then project it into another image yielding

$$\tilde{\boldsymbol{x}}_1 \sim \boldsymbol{P}_1 \boldsymbol{E}_1 \boldsymbol{p} = \boldsymbol{P}_1 \boldsymbol{E}_1 \boldsymbol{E}_0^{-1} \boldsymbol{P}_0^{-1} \tilde{\boldsymbol{x}}_0 = \boldsymbol{M}_{10} \tilde{\boldsymbol{x}}_0. \tag{17}$$

Unfortunately, we do not usually have access to the depth coordinates of pixels in a regular photographic image. However, for a *planar scene*, we can replace the last row of $\boldsymbol{P}_0$ in (13) with a general *plane equation*, $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + d_0$ that maps points on the plane to $z_0 = 0$ values. Then, if we set $z_0 = 0$, we can ignore the last column of $\boldsymbol{M}_{10}$ in (17) and also its last row, since we do not care about the final z-buffer depth. The mapping equation (17) thus reduces to

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{H}}_{10} \tilde{\boldsymbol{x}}_0, \tag{18}$$

where $\tilde{\boldsymbol{H}}_{10}$ is a general 3×3 homography matrix and $\tilde{\boldsymbol{x}}_1$ and $\tilde{\boldsymbol{x}}_0$ are now 2D homogeneous coordinates (i.e., 3-vectors) (Szeliski 1994, Szeliski 1996).[3] This justifies the use of the 8-parameter homography as a general alignment model for mosaics of planar scenes (Mann and Picard 1994, Szeliski 1996).[4]

Rotational panoramas The more interesting case is when the camera undergoes pure rotation (which is equivalent to assuming all points are very far from the camera). Setting $\boldsymbol{t}_0 = \boldsymbol{t}_1 = 0$, we get the simplified 3×3 homography

$$\tilde{\boldsymbol{H}}_{10} = \boldsymbol{K}_1 \boldsymbol{R}_1 \boldsymbol{R}_0^{-1} \boldsymbol{K}_0^{-1}, \tag{19}$$

where $\boldsymbol{K}_k = \mathrm{diag}(f_k, f_k, 1)$ is the simplified camera intrinsic matrix (Szeliski 1996). This can also be re-written as

$$\begin{bmatrix} x_1 \\ y_1 \\ f_1 \end{bmatrix} \sim \boldsymbol{R}_{01} \begin{bmatrix} x_0 \\ y_0 \\ f_0 \end{bmatrix}, \tag{20}$$

[3] For points off the reference plane, we get out-of-plane *parallax* motion, which is why this representation is often called the *plane plus parallax* representation (Sawhney 1994, Szeliski and Coughlan 1994, Kumar *et al.* 1994).

[4] Note that for a single pair of images, the fact that a 3D plane is being viewed by a set of rigid cameras does not reduce the total number of degrees of freedom. However, for a large collection of images taken of a planar surface (e.g., a whiteboard) from a calibrated camera, we could reduce the number of degrees of freedom per image from 8 to 6 by assuming that the plane is at a canonical location (e.g., $z = 1$).

which elucidates the simplicity of the mapping equations and makes all of the motion parameters explicit. 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 (Szeliski and Shum 1997). Estimating the 3D rotation matrix (and optionally, focal length) associated with each image is intrinsically more stable than estimating a full 8-d.o.f. homography, which makes this the method of choice for large-scale consumer-level image stitching algorithms (Szeliski and Shum 1997, Shum and Szeliski 2000, Brown and Lowe 2003).

Parameterizing 3D rotations. If we are going to represent panoramas using a combination of rotations and focal lengths, what is the best way to represent the rotations themselves? The choices include:

- the full 3×3 matrix $\boldsymbol{R}$, which has to be re-orthonormalized after each update;
- Euler angles (α, β, γ), which are a bad idea because you cannot always move smoothly from one rotation to another;
- the axis/angle (or exponential twist) representation, which represents the rotation by an axis $\hat{\boldsymbol{n}}$ and a rotation angle θ, or the product of the two,

$$\vec{\omega} = \theta \hat{\boldsymbol{n}} = (\omega_x, \omega_y, \omega_z), \tag{21}$$

  which has the minimial number of 3 parameters, but is still not unique;

- and unit quaternions, which represent rotations with unit 4-vectors,

$$\boldsymbol{q} = (x, y, z, w) = (\boldsymbol{v}, w) = (\sin \frac{\theta}{2} \hat{\boldsymbol{n}}, \cos \frac{\theta}{2}), \tag{22}$$

  where $\hat{\boldsymbol{n}}$ and θ are the rotation axis and angle.

The rotation matrix corresponding to a rotation by θ around an axis $\hat{\boldsymbol{n}}$ is

$$\boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta [\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2, \tag{23}$$

which is known as *Rodriguez's formula* (Ayache 1989), and $[\hat{\boldsymbol{n}}]_\times$ is the matrix form of the cross-product operator,

$$[\hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}. \tag{24}$$

For small (infinitessimal) rotations, the rotation reduces to

$$\boldsymbol{R}(\vec{\omega}) \approx \boldsymbol{I} + \theta[\hat{\boldsymbol{n}}]_\times = \boldsymbol{I} + [\vec{\omega}]_\times. \tag{25}$$

Using the trigonometric identities $\sin\theta = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}$ and $(1-\cos\theta) = 2\sin^2\frac{\theta}{2}$, Rodriguez's formula can be converted to

$$\begin{aligned} \boldsymbol{R}(\boldsymbol{q}) &= \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1-\cos\theta)[\hat{\boldsymbol{n}}]_\times^2 \\ &= \boldsymbol{I} + 2w[\boldsymbol{v}]_\times + 2[\boldsymbol{v}]_\times^2. \end{aligned} \tag{26}$$

This suggests a quick way to rotate a vector by a quaternion using a series of cross products, scalings, and additions. From this, we can derive the commonly used formula for $\boldsymbol{R}(\boldsymbol{q})$ as a function of (x, y, z, w),

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-zw) & 2(xz+yw) \\ 2(xy+zw) & 1-2(x^2+z^2) & 2(yz-xw) \\ 2(xz-yw) & 2(yz+xw) & 1-2(x^2+y^2) \end{bmatrix}. \tag{27}$$

[Note: Colin says this formula is inconsistent with formulas he found on the Web; he promised to send some links (follow up).] The diagonal terms can be made more symmetrical by replacing $1-2(y^2+z^2)$ with $(x^2+w^2-y^2-z^2)$, etc.

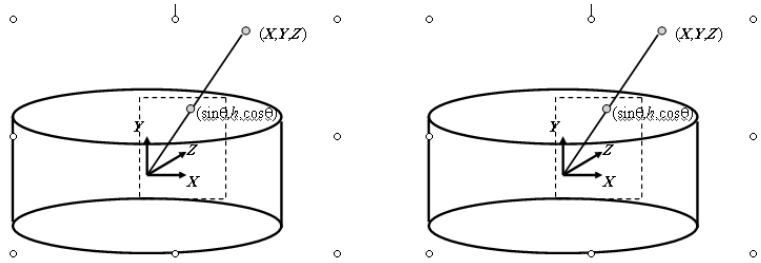
Between the axis/angle represenation and quaternions, I generally prefer unit quaternions, because they possess a nice algebra that makes it easy to take products (compositions), ratios (change in rotation), and linear interpolations (Shoemake 1985). For example, the product of two quaternions $\boldsymbol{q}_0 = (\boldsymbol{v}_0, w_0)$ and $\boldsymbol{q}_1 = (\boldsymbol{v}_1, w_1)$ is given by

$$\boldsymbol{q}_2 = \boldsymbol{q}_0\boldsymbol{q}_1 = (\boldsymbol{v}_0 \times \boldsymbol{v}_1 + w_0\boldsymbol{v}_1 + w_1\boldsymbol{v}_0,\ w_0w_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1), \tag{28}$$

with the property that $\boldsymbol{R}(\boldsymbol{q}_2) = \boldsymbol{R}(\boldsymbol{q}_0)\boldsymbol{R}(\boldsymbol{q}_1)$. (Note that quaternion multiplication is *not* commutative, just as 3D rotations and matrix multiplications are not.) Taking the inverse of a quaternion is also easy: just flip the sign of $\boldsymbol{v}$ or w (but not both!). However, when it comes time to update rotation estimates, I use an *incremental* form of the axis/angle representation (25), as described in §4.3.

2.3 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 to then use a pure translational model to align them (Chen 1995). Unfortunately, this only works if the images are all taken with a level camera or with a known tilt angle.



(a) (b)

Figure 3: *Projection from 3D to cylindrical and spherical coordinates.*
[Note: Need something better, just a placeholder for now...]

Assume for now that the camera is in its canonical position, i.e., its rotation matrix is the identity so that the optic 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 1994). 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

$$(\sin\theta, h, \cos\theta) \propto (x, y, f), \tag{29}$$

as shown in Figure 3a. From this correspondence, we can compute the formula for the *warped* or *mapped* coordinates (Szeliski and Shum 1997),

$$x' = s\theta = s\tan^{-1}\frac{x}{f}, \tag{30}$$

$$y' = sh = s\frac{y}{\sqrt{x^2+f^2}}, \tag{31}$$

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.[5] The inverse of this mapping equation is given by

$$x = f\tan\theta = f\tan\frac{x'}{s}, \tag{32}$$

$$y = h\sqrt{x^2+f^2} = \frac{y'}{s}f\sqrt{1+\tan^2 x'/s} = f\frac{y'}{s}\sec\frac{x'}{s}. \tag{33}$$

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

[5]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 §6.

(a) (b)

Figure 4: *An example of a cylindrical panorama: (a) two cylindrically warped images related by a horizontal translation; (b) part of a cylindrical panorama composited from a sequence of images.*
[Note: Find the original images from the above paper...]

strip. In this case, the sphere is parameterized by two angles (θ, ϕ), with 3D spherical coordinates given by

$$(\sin\theta\cos\phi, \sin\phi, \cos\theta\cos\phi) \propto (x, y, f), \tag{34}$$

as shown in Figure 3b. The correspondence between coordinates is now given by (Szeliski and Shum 1997)

$$x' = s\theta = s\tan^{-1}\frac{x}{f}, \tag{35}$$

$$y' = s\phi = s\tan^{-1}\frac{y}{\sqrt{x^2+f^2}}, \tag{36}$$

while the inverse is given by

$$x = f\tan\theta = f\tan\frac{x'}{s}, \tag{37}$$

$$y = \sqrt{x^2+f^2}\tan\phi = \tan\frac{y'}{s} f\sqrt{1+\tan^2 x'/s} = f\tan\frac{y'}{s}\sec\frac{x'}{s}. \tag{38}$$

Note that it may be simpler to generate a scaled (x, y, z) direction from (34) 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.[6] This makes it attractive as an initial class project in an introductory computer vision course, since the full complexity of the perspective alignment algorithm (§3.5 & §4.3) can be avoided. Figure 4 shows how two cylindrically warped images from a leveled rotational panorama are related by a pure translation (Szeliski and Shum 1997).

[6] Small vertical tilts can sometimes be compensated for with a vertical translation.

Figure 5: *An example of a spherical panorama constructed from 54 photographs.*
[Note: Find the original images from the above paper...]

Professional panoramic photographers sometimes also use a pan-tilt head that makes it easy to control the tilt and to stop at specific *detents* in the rotation angle. This not only ensures a uniform coverage of the visual field with a desired amount of image overlap, but also makes 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, since the rough relative positioning of all the input images is known ahead of time, enabling a reduced search range for alignment. Figure 5 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 optic axis rather than the vertical axis,

$$(\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\phi) = s\,(x, y, z). \tag{39}$$

In this case, the mapping equations become

$$x' = s\phi\cos\theta = s\frac{x}{r}\tan^{-1}\frac{r}{z}, \tag{40}$$

$$y' = s\phi\sin\theta = s\frac{y}{r}\tan^{-1}\frac{r}{z}, \tag{41}$$

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 *fisheyes lenses*, as discussed below. Note how for small values of (x, y), the mapping equations reduces to $x' \approx sx/z$, which suggests that s plays a role similar to the focal length f. Figure 6 shows the full 3D rotational panorama shown in Figure 5 unwrapped onto a polar compositing surface.

Figure 6: *An example of a 3D rotational panorama mapped using a polar mapping.*
[Note: Need to write the code to generate this...]

(a) (b) (c)

Figure 7: *Examples of radial lens distortion: (a) barrel, (b) pincushion, and (c) fisheye.*
[Note: Run VideoMosaic to generate (a) and (b), grab any one of your fisheyes for (c).]

2.4 Lens distortions

When images are taken with wide-angle lenses, it is often necessary to model *lens distortions* such as *radial distortion*. The radial distortion model says that coordinates in the observed images are displaced away (*barrel* distortion) or towards (*pincushion* distortion) the image center by an amount proportional to their radial distance (Figure 7a–b). The simplest radial distortion models use low-order polynomials, e.g.,

$$\begin{aligned} x' &= x(1 + \kappa_1 r^2 + \kappa_2 r^4) \\ y' &= y(1 + \kappa_1 r^2 + \kappa_2 r^4), \end{aligned} \tag{42}$$

where $r^2 = x^2 + y^2$ and κ_1 and κ_2 are called the *radial distortion parameters* (Brown 1971, Slama 1980).[7] More complex distortion models also include *tangential (decentering) distortions* (Slama 1980), but these are usually not necessary for consumer-level stitching.

A variety of techniques can be used to estimate the radial distortion parameters for a given lens. One of the simplest and most useful is to take an image of a scene with a lot of straight lines, especially lines aligned with and near the edges of the image. The radial distortion parameters can then be adjusted until all of the lines in the image are straight, which is commonly called the *plumb line method* (Brown 1971, Kang 2001, El-Melegy and Farag 2003).

Another approach is to use several overlapping images and to combine the estimation of the radial distortion parameters together with the image alignment process. Sawhney and Kumar (1999) use a hierarchy of motion models (translation, affine, projective) in a coarse-to-fine strategy coupled with a quadratic radial distortion correction term. They use direct (intensity-based) minimization to compute the alignment. Stein (1997) uses a feature-based approach combined with a general 3D motion model (and quadratic radial distortion), which requires more matches than a parallax-free rotational panorama but is potentially more general.

Fisheye lenses require a different model than traditional polynomial models of radial distortion (Figure 7c). Instead, fisheye lenses behave, to a first approximation, as *equi-distance* projectors

[7] Sometimes the relationship between x and x' is expressed the other way around, i.e., using primed (final) coordinates on the right-hand side.

of angles away from the optic axis (Xiong and Turkowski 1997), which is the same as the *polar projection* described by equations (39-41). Xiong and Turkowski (1997) describe how this model can be extended with the addition of an extra quadratic correction in ϕ, and how the unknown parameters (center of projection, scaling factor s, etc.) can be estimated from a set of overlapping fisheye images using a direct (intensity-based) non-linear minimization algorithm.

3 Direct (pixel-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 that use pixel-to-pixel matching are often called *direct methods*, as opposed to the *feature-based methods* described in the next section.

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 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. Alternatively, Fourier transforms can be used to speed up the computation. To get sub-pixel precision in the alignment, *incremental* methods based on a Taylor series expansion of the image function are often used. These can also be applied to *parametric motion models*. Each of these techniques is described in more detail below.

3.1 Error metrics

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

$$E_{\mathrm{SSD}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \sum_i e_i^2, \tag{43}$$

where $\boldsymbol{u} = (u, v)$ is the *displacement* and $e_i = I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)$ is called the *residual error* (or the *displaced frame difference* in the video coding literature).[8] (We ignore for the moment the possibility that parts of I_0 may lie outside the boundaries of I_1 or be otherwise not visible.)

[8]The usual justification for using least squares is that it is the optimal estimate with respect to Gaussian noise. See the discussion below on robust alternatives.

In general, the displacement $\boldsymbol{u}$ can be fractional, so a suitable interpolation function must be applied to image $I_1(\boldsymbol{x})$. In practice, a bilinear interpolant is often used, but bi-cubic interpolation may yield slightly better results. Color images can be processed by summing differences across all three color channels, although it is also possible to first transform the images into a different color space or to only use the luminance (which is often done in video encoders).

Robust error metrics We can make the above error metric more robust to outliers by replacing the squared error terms with a robust function $\rho(e_i)$ (Huber 1981, Hampel *et al.* 1986, Black and Anandan 1996, Stewart 1999) to obtain

$$E_{\mathrm{SRD}}(\boldsymbol{u}) = \sum_i \rho(I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)) = \sum_i \rho(e_i). \tag{44}$$

The robust norm $\rho(e)$ is a function that grows less quickly than the quadratic penalty associated with least squares. One such function, sometimes used in motion estimation for video coding because of its speed, is the *sum of absolute differences* (SAD) metric, i.e.,

$$E_{\mathrm{SAD}}(\boldsymbol{u}) = \sum_i |I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)| = \sum_i |e_i|. \tag{45}$$

However, since this function is not differentiable at the origin, it is not well suited to gradient-descent approaches such as the ones presented in §3.4.

Instead, a smoothly varying function that is quadratic for small values but grows more slowly away from the origin is often used. Black and Rangarajan (1996) discuss a variety of such functions, including the *Geman-McClure* function,

$$\rho_{\mathrm{GM}}(x) = \frac{x^2}{1 + x^2/a^2}, \tag{46}$$

where a is a constant that can be thought of as an *outlier threshold*. An appropriate value for the threshold can itself the derived using robust statistics (Huber 1981, Hampel *et al.* 1986, Rousseeuw and Leroy 1987), e.g., by computing the *median of absolute differences*, $MAD = \mathrm{med}_i|e_i|$, and multiplying by 1.4 to obtain a robust estimate of the standard deviation of the non-outlier noise process.

Spatially varying weights. The error metrics above ignore that fact that for a given alignment, some of the pixels being compared may lie outside the original image boundaries. Furthermore, we may want to partially or completely downweight the contributions of certain pixels. For example, we may want to selectively "erase" some parts of an image from consideration, e.g., when stitching a mosaic where unwanted foreground objects have been cut out. For applications such as

background stabilization, we may want to downweight the middle part of the image, which often contains independently moving objects being tracked by the camera.

All of these tasks can be accomplished by associating a spatially varying per-pixel weight value with each of the two images being matched. The error metric then become the weighted (or windowed) SSD function,

$$E_{\mathrm{WSSD}}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u}) [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{47}$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of the images.

If a large range of potential motions is allowed, the above metric can have a bias towards smaller overlap solutions. To counteract this bias, the windowed SSD score can be divided by the overlap area

$$A = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{48}$$

to compute a *per-pixel* (or mean) squared pixel error. The square root of this quantity is the *root mean squared* intensity error

$$RMS = \sqrt{E_{\mathrm{WSSD}}/A} \tag{49}$$

often seen reported in comparative studies.

Bias and gain (exposure differences). Often, the two images being aligned were not taken with the same exposure. A simple model of linear (affine) intensity variation between the two images is the *bias and gain* model,

$$I_1(\boldsymbol{x} + \boldsymbol{u}) = (1 + \alpha) I_0(\boldsymbol{x}) + \beta, \tag{50}$$

where β is the *bias* and α is the *gain* (Lucas and Kanade 1981, Gennert 1988, Fuh and Maragos 1991, Baker *et al.* 2003b). The least squares formulation then becomes

$$E_{\mathrm{BG}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - (1 + \alpha) I_0(\boldsymbol{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\boldsymbol{x}_i) + \beta - e_i]^2. \tag{51}$$

Rather than taking a simple squared difference between corresponding patches, it becomes necessary to perform a *linear regression*, which is somewhat more costly. Note that for color images, it may be necessary to estimate a different bias and gain for each color channel to compensate for the automatic *color correction* performed by some digital cameras.

A more general (spatially-varying non-parametric) model of intensity variation, which is computed as part of the registration process, is presented in (Jia and Tang 2003). This can be useful for dealing with local variations such as the *vignetting* caused by wide-angle lenses. It is also possible to pre-process the images before comparing their values, e.g., by using band-pass filtered images (Burt and Adelson 1983, Bergen *et al.* 1992) or using other local transformations such as histograms or rank transforms (Cox *et al.* 1995, Zabih and Woodfill 1994).

Correlation. An alternative to taking intensity differences is to perform *correlation*, i.e., to maximize the *product* (or *cross-correlation*) of the two aligned images,

$$E_{\mathrm{CC}}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u}). \tag{52}$$

At first glance, this may appear to make bias and gain modeling unnecessary, since the images will prefer to line up regardless of their relative scales and offsets. However, this is actually not true. If a very bright patch exists in $I_1(\boldsymbol{x})$, the maximum product may actually lie in that area.

For this reason, *normalized cross-correlation* is more commonly used,

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\,[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2 [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}}, \tag{53}$$

where

$$\overline{I_0} = \frac{1}{N} \sum_i I_0(\boldsymbol{x}_i) \qquad \text{and} \tag{54}$$

$$\overline{I_1} = \frac{1}{N} \sum_i I_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{55}$$

are the *mean images* of the corresponding patches and N is the number of pixels in the patch. The normalized cross-correlation score is always guaranteed to be in the range $[-1, 1]$, which makes it easier to handle in some higher-level applications (such as deciding which patches truly match). Note, however, that the NCC score is undefined if either of the two patches has zero variance (and in fact, its performance degrades for noisy low-contrast regions).

3.2 Hierarchical motion estimation

Now that we have defined an alignment cost function to optimize, how do we find its minimum? The simplest solution is to do a *full search* over some range of shifts, using either integer or sub-pixel steps. This is often the approach used for *block matching* in *motion compensated video compression*, where a range of possible motions (say ± 16 pixels) is explored.[9]

To accelerate this search process, *hierarchical motion estimation* is often used, where an image pyramid is first constructed, and a search over a smaller number of discrete pixels (corresponding to the same range of motion) is first performed at coarser levels (Quam 1984, Anandan 1989, Bergen *et al.* 1992). The motion estimate from one level of the pyramid can then be used to initialize a

[9] In stereo matching, an explicit search over all possible disparities (i.e., a *plane sweep*) is almost always performed, since the number of search hypotheses is much smaller due to the 1D nature of the potential displacements (Scharstein and Szeliski 2002).

smaller *local* search at the next finer level. While this is not guaranteed to produce the same result as full search, it usually works almost as well and is much faster.

More formally, let

$$I_k^{(l)}(\boldsymbol{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\boldsymbol{x}_j) \tag{56}$$

be the *decimated* image at level l obtained by subsampling (*downsampling*) a smoothed (pre-filtered) version of the image at level $l-1$. At the coarsest level, we search for the best displacement $\boldsymbol{u}^{(l)}$ that minimizes the difference between images $I_0^{(l)}$ and $I_1^{(l)}$. This is usually done using a full search over some range of displacements $\boldsymbol{u}^{(l)} \in 2^{-l}[-S, S]^2$ (where S is the desired *search range* at the finest (original) resolution level), optionally followed by the incremental refinement step described in §3.4.

Once a suitable motion vector has been estimated, it is used to *predict* a likely displacement

$$\hat{\boldsymbol{u}}^{(l-1)} \leftarrow 2\boldsymbol{u}^{(l)} \tag{57}$$

for the next finer level.[10] The search over displacements is then repeated at the finer level over a much narrower range of displacements, say $\hat{\boldsymbol{u}}^{(l-1)} \pm 1$, again optionally combined with an incremental refinement step (Anandan 1989). A nice description of the whole process, extended to parametric motion estimation (§3.5), can be found in (Bergen *et al.* 1992).

3.3 Fourier-based alignment

When the search range corresponds to a significant fraction of the larger image (as is the case in image stitching), the hierarchical approach may not work that well, since it is often not possible to coarsen the representation too much before significant features get blurred away. In this case, a Fourier-based approach may be preferable.

Fourier-based alignment relies on the fact that the Fourier transform of a shifted signal has the same magnitude as the original signal but linearly varying phase, i.e.,

$$\mathcal{F}\{I_1(\boldsymbol{x}+\boldsymbol{u})\} = \mathcal{F}\{I_1(\boldsymbol{x})\}\, e^{-2\pi j \boldsymbol{u}\cdot\boldsymbol{f}} = \mathcal{I}_1(\boldsymbol{f}) e^{-2\pi j \boldsymbol{u}\cdot\boldsymbol{f}}, \tag{58}$$

where $\boldsymbol{f}$ is the vector-valued frequency of the Fourier transform and we use calligraphic notation $\mathcal{I}_1(\boldsymbol{f}) = \mathcal{F}\{I_1(\boldsymbol{x})\}$ to denote the Fourier transform of a signal (Oppenheim *et al.* 1999, p. 57).

Another useful property of Fourier transforms is that convolution in the spatial domain corresponds to multiplication in the Fourier domain (Oppenheim *et al.* 1999, p. 58). Thus, the Fourier

[10] This doubling of displacements is only necessary if displacements are defined in integer *pixel* coordinates, which is the usual case in the literature, e.g., (Bergen *et al.* 1992). If *normalized device coordinates* (§2) are used instead, the displacements (and search ranges) need not change from level to level, although the step sizes will need to be adjusted (to keep search steps of roughly one pixel).

transform of the cross-correlation function E_{CC} can be written as

$$\mathcal{F}\left\{E_{\mathrm{CC}}(\boldsymbol{u})\right\} = \mathcal{F}\left\{\sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u})\right\} = \mathcal{F}\left\{I_0(\boldsymbol{u}) \overline{*} I_1(\boldsymbol{u})\right\} = \mathcal{I}_0(\boldsymbol{f}) \mathcal{I}_1^*(\boldsymbol{f}), \tag{59}$$

where

$$h(\boldsymbol{u}) = f(\boldsymbol{u}) \overline{*} g(\boldsymbol{u}) = \sum_i f(\boldsymbol{x}_i) g(\boldsymbol{x}_i + \boldsymbol{u}) \tag{60}$$

is the *correlation* function, i.e., the convolution of one signal with the reverse of the other, and $\mathcal{I}_1^*(\boldsymbol{f})$ is the *complex conjugate* of $\mathcal{I}_1(\boldsymbol{f})$. (This is because convolution is defined as the summation of one signal with the reverse of the other (Oppenheim *et al.* 1999).)

Thus, to efficiently evaluate E_{CC} over the range of all possible values of $\boldsymbol{u}$, we take the Fourier transforms of both images $I_0(\boldsymbol{x})$ and $I_1(\boldsymbol{x})$, multiply both transforms together (after conjugating the second one), and take the inverse transform of the result. The Fast Fourier Transform algorithm can compute the transform of an $N \times M$ image in $\mathrm{O}(NM \log NM)$ operations (Oppenheim *et al.* 1999). This can be significantly faster than the $\mathrm{O}(N^2 M^2)$ operations required to do a full search when the full range of image overlaps is considered.

While Fourier-based convolution is often used to accelerate the computation of image correlations, it can also be used to accelerate the sum of squared differences function (and its variants) as well. Consider the SSD formula given in (43). Its Fourier transform can be written as

$$\mathcal{F}\left\{E_{\mathrm{SSD}}(\boldsymbol{u})\right\} = \mathcal{F}\left\{\sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2\right\} = \delta(\boldsymbol{f}) \sum_i [I_0^2(\boldsymbol{x}_i) + I_1^2(\boldsymbol{x}_i)] - 2\mathcal{I}_0(\boldsymbol{f}) \mathcal{I}_1^*(\boldsymbol{f}). \tag{61}$$

Thus, the SSD function can be computed by taking twice the correlation function and subtracting it from the sum of the energies in the two images.

Windowed correlation. Unfortunately, the Fourier convolution theorem only applies when the summation over $\boldsymbol{x}_i$ is performed over *all* the pixels in both images, using a circular shift of the image when accessing pixels outside the original boundaries. While this is acceptable for small shifts and comparably sized images, it makes no sense when the images overlap by a small amount or one image is a small subset of the other.

In that case, the cross-correlation function should be replaced with a *windowed* (weighted) cross-correlation function,

$$E_{\mathrm{WCC}}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}_i) I_0(\boldsymbol{x}_i)\; w_1(\boldsymbol{x}_i + \boldsymbol{u}) I_1(\boldsymbol{x}_i + \boldsymbol{u}), \tag{62}$$

$$= [w_0(\boldsymbol{x}) I_0(\boldsymbol{x})] \overline{*} [w_1(\boldsymbol{x}) I_1(\boldsymbol{x})] \tag{63}$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of the images, and both images are padded so that circular shifts return 0 values outside the original image boundaries.

An even more interesting case is the computation of the *weighted* SSD function introduced in (47),

$$\begin{aligned} E_{\mathrm{WSSD}}(\boldsymbol{u}) &= \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i + \boldsymbol{u}) [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \\ &= w_0(\boldsymbol{x}) \overline{*} [w_1(\boldsymbol{x}) I_1^2(\boldsymbol{x})] + [w_0(\boldsymbol{x}) I_0^2(\boldsymbol{x})] \overline{*} w_1(\boldsymbol{x}) - 2[w_0(\boldsymbol{x}) I_0(\boldsymbol{x})] \overline{*} [w_1(\boldsymbol{x}) I_1(\boldsymbol{x})]. \end{aligned} \tag{64}$$

The Fourier transform of the resulting expression is therefore

$$\mathcal{F}\{E_{\mathrm{WSSD}}(\boldsymbol{u})\} = \mathcal{W}_0(\boldsymbol{f}) \mathcal{S}_1^*(\boldsymbol{f}) + \mathcal{S}_0(\boldsymbol{f}) \mathcal{W}_1^*(\boldsymbol{f}) - 2\hat{\mathcal{I}}_0(\boldsymbol{f}) \hat{\mathcal{I}}_1^*(\boldsymbol{f}), \tag{65}$$

where

$$\begin{aligned} \mathcal{W}_0 &= \mathcal{F}\{w_0(\boldsymbol{x})\}, & \mathcal{W}_1 &= \mathcal{F}\{w_1(\boldsymbol{x})\}, \\ \hat{\mathcal{I}}_0 &= \mathcal{F}\{w_0(\boldsymbol{x}) I_0(\boldsymbol{x})\}, & \hat{\mathcal{I}}_1 &= \mathcal{F}\{w_1(\boldsymbol{x}) I_1(\boldsymbol{x})\}, \\ \mathcal{S}_0 &= \mathcal{F}\{w_0(\boldsymbol{x}) I_0^2(\boldsymbol{x})\}, \quad \text{and} & \mathcal{S}_1 &= \mathcal{F}\{w_1(\boldsymbol{x}) I_1^2(\boldsymbol{x})\} \end{aligned} \tag{66}$$

are the Fourier transforms of the weighting functions and the *weighted* original and squared image signals. Thus, for the cost of a few additional image multiplies and Fourier transforms, the correct windowed SSD function can be computed. (To my knowledge, I have not seen this formulation written down before, but I have been teaching it to students for several years now.)

The same kind of derivation can be applied to the bias-gain corrected sum of squared difference function E_{BG}. Again, Fourier transforms can be used to efficiently compute all the correlations needed to perform the linear regression in the bias and gain parameters in order to estimate the exposure-compensated difference for each potential shift.

Phase correlation. A variant of regular correlation (59) that is sometimes used for motion estimation is *phase correlation* (Kuglin and Hines 1975, Brown 1992). Here, the spectrum of the two signals being matched is *whitened* by dividing each per-frequency product in (59) by the magnitudes of the Fourier transforms,

$$\mathcal{F}\{E_{\mathrm{PC}}(\boldsymbol{u})\} = \frac{\mathcal{I}_0(\boldsymbol{f}) \mathcal{I}_1^*(\boldsymbol{f})}{\|\mathcal{I}_0(\boldsymbol{f})\| \|\mathcal{I}_1(\boldsymbol{f})\|} \tag{67}$$

before taking the final inverse Fourier transform. In the case of noiseless signals with perfect (cyclic) shift, we have $I_1(\boldsymbol{x} + \boldsymbol{u}) = I_0(\boldsymbol{x})$, and hence from (58) we obtain

$$\begin{aligned} \mathcal{F}\{I_1(\boldsymbol{x} + \boldsymbol{u})\} &= \mathcal{I}_1(\boldsymbol{f}) e^{-2\pi j \boldsymbol{u} \cdot \boldsymbol{f}} = \mathcal{I}_0(\boldsymbol{f}) \quad \text{and} \\ \mathcal{F}\{E_{\mathrm{PC}}(\boldsymbol{u})\} &= e^{-2\pi j \boldsymbol{u} \cdot \boldsymbol{f}}. \end{aligned} \tag{68}$$

The output of phase correlation (under ideal conditions) is therefore a single spike (impulse) located at the correct value of $\boldsymbol{u}$, which (in principle) makes it easier to find the correct estimate.

(a) (b) (c)

Figure 8: *An image (a) and its polar (b) and log-polar (c) transforms.*
[Note: Need to write the code to generate these...]

Phase correlation has a reputation in some quarters of outperforming regular correlation, but this behavior depends on the characteristics of the signals and noise. If the original images are contaminated by noise in a narrow frequency band (e.g., low-frequency noise or peaked frequency "hum"), the whitening process effectively de-emphasizes the noise in these regions. However, if the original signals have very low signal-to-noise ratio at some frequencies (say, two blurry or low-textured images with lots of high-frequency noise), the whitening process can actually decrease performance. To my knowledge, no systematic comparison of the two approaches (together with other Fourier-based techniques such as windowed and/or bias-gain corrected differencing) has been performed, so this is an interesting area for further exploration.

Rotations and scale. While Fourier-based alignment is mostly used to estimate translational shifts between images, it can, under certain limited conditions, also be used to estimate in-plane rotations and scales. Consider two images that are related *purely* by rotation, i.e.,

$$I_1(\hat{\boldsymbol{R}}\boldsymbol{x}) = I_0(\boldsymbol{x}). \tag{69}$$

If we re-sample the images into *polar coordinates* (Figure 8b),

$$\tilde{I}_0(r,\theta) = I_0(r\cos\theta, r\sin\theta) \quad \text{and} \quad \tilde{I}_1(r,\theta) = I_1(r\cos\theta, r\sin\theta), \tag{70}$$

we obtain

$$\tilde{I}_1(r,\theta+\hat{\theta}) = \tilde{I}_0(r,\theta). \tag{71}$$

The desired rotation can then be estimated using an FFT shift-based technique. *[Note: Show an image and its polar transform.]*

If the two images are also related by a scale,

$$I_1(e^{\hat{s}}\hat{\boldsymbol{R}}\boldsymbol{x}) = I_0(\boldsymbol{x}), \tag{72}$$

we can re-sample into *log-polar coordinates* (Figure 8c),

$$\tilde{I}_0(s,\theta) = I_0(e^s\cos\theta, e^s\sin\theta) \quad \text{and} \quad \tilde{I}_1(s,\theta) = I_1(e^s\cos\theta, e^s\sin\theta), \tag{73}$$

to obtain

$$\tilde{I}_1(s+\hat{s},\theta+\hat{\theta}) = I_0(s,\theta). \tag{74}$$

In this case, care must be taken to choose a suitable range of s values that reasonably samples the original image. *[Note: Show an image and its log-polar transform, along with a rotated/scaled image and its transforms.]*

For images that are also translated by a small amount,

$$I_1(e^{\hat{s}}\hat{\boldsymbol{R}}\boldsymbol{x} + \boldsymbol{t}) = I_0(\boldsymbol{x}), \tag{75}$$

De Castro and Morandi (1987) proposed an ingenious solution that uses several steps to estimate the unknown parameters. First, both images are converted to the Fourier domain, and only the magnitudes of the transformed images are retained. In principle, the Fourier magnitude images are insensitive to translations in the image plane (although the usual caveats about border effects apply). Next, the two magnitude images are aligned in rotation and scale using the polar or log-polar representations. Once rotation and scale are estimated, one of the images can be de-rotated and scaled, and a regular translational algorithm can be applied to estimate the translational shift.

Unfortunately, this trick only applies when the images have large overlap (small translational motion). For more general motion of patches or images, the parametric motion estimator described in §3.5 or the feature-based approaches described in §4 need to be used.

3.4 Incremental refinement

The techniques described up till now can estimate translational alignment to the nearest pixel (or potentially fractional pixel if smaller search steps are used). In general, image stabilization and stitching applications require much higher accuracies to obtain acceptable results.

To obtain better *sub-pixel* estimates, we can use one of several techniques (Tian and Huhns 1986). 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 (1981), is to do *gradient descent* on the SSD energy function (43), using a Taylor Series expansion of the image function,

$$\begin{aligned} E_{\text{LK-SSD}}(\boldsymbol{u} + \Delta\boldsymbol{u}) &= \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u} + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) + \boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 \qquad (76) \\ &= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2, \qquad (77) \end{aligned}$$

where

$$\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) = \nabla I_1(\boldsymbol{x}_i + \boldsymbol{u}) = (\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y})(\boldsymbol{x}_i + \boldsymbol{u}) \tag{78}$$

is the *image gradient* at $\boldsymbol{x}_i + \boldsymbol{u}$.[11]

The above least squares problem can be minimizing by solving the associated *normal equations* (Golub and Van Loan 1996),

$$\boldsymbol{A}\Delta\boldsymbol{u} = \boldsymbol{b} \tag{79}$$

where

$$\boldsymbol{A} = \sum_i \boldsymbol{J}_1^T(\boldsymbol{x}_i + \boldsymbol{u})\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{80}$$

and

$$\boldsymbol{b} = -\sum_i e_i \boldsymbol{J}_1^T(\boldsymbol{x}_i + \boldsymbol{u}) \tag{81}$$

are called the *Hessian* and *gradient-weighted residual vector*, respectively. These matrices are also often written as

$$\boldsymbol{A} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \tag{82}$$

where the subscripts in I_x and I_y denote spatial derivatives, and I_t is called the *temporal derivative*, which makes sense if we are computing instantaneous velocity in a video sequence.

The gradients required for $\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\boldsymbol{x}_i + \boldsymbol{u})$, 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,

$$\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) \approx \boldsymbol{J}_0(\boldsymbol{x}), \tag{83}$$

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 (Hager and Belhumeur 1998, Baker and Matthews 2004).

The effectiveness of the above incremental update rule relies on the quality of the Taylor series approximation. When far away from the true displacement (say 1-2 pixels), several iterations may be needed. (It is possible, however, to estimate a value for $\boldsymbol{J}_1$ using a least-squares fit to a series of larger displacements in order to increase the range of convergence (Jurie and Dhome 2002).) When started in the vicinity of the correct solution, only a few iterations usually suffice. A commonly used stopping criterion is to monitor the magnitude of the displacement correction $\|\boldsymbol{u}\|$ and to stop when it drops below a certain threshold (say $1/10^{th}$ of a pixel). For larger motions, it is usual to combine the incremental update rule with a hierarchical coarse-to-fine search strategy, as described in §3.2.

[11] We follow the convention, commonly used in robotics and in (Baker and Matthews 2004), that derivatives with respect to (column) vectors result in row vectors, so that fewer transposes are needed in the formulas.

Conditioning and aperture problems. Sometimes, the inversion of the linear system (79) can be poorly conditioned because of lack of two-dimensional texture in the patch being aligned. A commonly occurring example of this is the *aperture problem*, first identified in some of the early papers on optic flow (Horn and Schunck 1981) and then studied more extensively by Anandan (1989). Consider an image patch that consists of a slanted edge moving to the right. Only the *normal* component of the velocity (displacement) can be reliably recovered in this case. This manifests itself in (79) as a *rank-deficient* matrix $\boldsymbol{A}$, i.e., one whose smaller eigenvalue is very close to zero.[12]

When equation (79) is solved, the component of the displacement along the edge is very poorly conditioned and can result in wild guesses under small noise perturbations. One way to mitigate this problem is to add a *prior* (soft constraint) on the expected range of motions (Simoncelli *et al.* 1991, Baker *et al.* 2004). This can be accomplished by adding a small value to the diagonal of $\boldsymbol{A}$, which essentially biases the solution towards smaller $\Delta\boldsymbol{u}$ values that still (mostly) minimize the squared error.

However, the pure Gaussian model assumed when using a simple (fixed) quadratic prior, as in (Simoncelli *et al.* 1991), does not always hold in practice, e.g., because of aliasing along strong edges (Triggs 2004). For this reason, it may be prudent to add some small fraction (say 5%) of the larger eigenvalue to the smaller one before doing the matrix inversion.

Uncertainty modeling The reliability of a particular patch-based motion estimate can be captured more formally with an *uncertainty model*. The simplest such model is a *covariance matrix*, which captures the expected variance in the motion estimate in all possible directions. Under small amounts of additive Gaussian noise, it can be shown that the covariance matrix $\Sigma_{\boldsymbol{u}}$ is proportional to the inverse of the Hessian $\boldsymbol{A}$,

$$\Sigma_{\boldsymbol{u}} = \sigma_n^2 \boldsymbol{A}^{-1}, \tag{84}$$

where σ_n^2 is the variance of the additive Gaussian noise (Anandan 1989, Matthies *et al.* 1989, Szeliski 1989). For larger amounts of noise, the linearization performed by the Lucas-Kanade algorithm in (77) is only approximate, so the above quantity becomes the *Cramer-Rao lower bound* on the true covariance. Thus, the minimum and maximum eigenvalues of the Hessian $\boldsymbol{A}$ can now be interpreted as the (scaled) inverse variances in the least-certain and most-certain directions of motion.

[12] The matrix $\boldsymbol{A}$ is by construction always guaranteed to be symmetric positive semi-definite, i.e., it has real non-negative eigenvalues.

Bias and gain, weighting, and robust error metrics. The Lucas-Kanade update rule can also be applied to the bias-gain equation (51) to obtain

$$E_{\mathrm{LK-BG}}(\boldsymbol{u}+\Delta\boldsymbol{u}) = \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} + e_i - \alpha I_0(\boldsymbol{x}_i) - \beta]^2 \tag{85}$$

(Lucas and Kanade 1981, Gennert 1988, Fuh and Maragos 1991, Baker *et al.* 2003b). The resulting 4×4 system of equations in can be solved to simultaneously estimate the translational displacement update $\Delta\boldsymbol{u}$ and the bias and gain parameters β and α.[13]

A similar formulation can be derived for images (templates) that have a *linear appearance variation*,

$$I_1(\boldsymbol{x}+\boldsymbol{u}) \approx I_0(\boldsymbol{x}) + \sum_j \lambda_j B_j(\boldsymbol{x}), \tag{86}$$

where the $B_j(\boldsymbol{x})$ are the *basis images* and the λ_j are the unknown coefficients (Hager and Belhumeur 1998, Baker *et al.* 2003a, Baker *et al.* 2003b). Potential linear appearance variations include illumination changes (Hager and Belhumeur 1998) and small non-rigid deformations (Black and Jepson 1998).

A weighted (windowed) version of the Lucas-Kanade algorithm is also possible,

$$E_{\mathrm{LK-WSSD}}(\boldsymbol{u}+\Delta\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}) w_1(\boldsymbol{x}_i+\boldsymbol{u})[\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2. \tag{87}$$

Note that here, in deriving the Lucas-Kanade update from the original weighted SSD function (47), we have neglected taking the derivative of $w_1(\boldsymbol{x}_i+\boldsymbol{u})$ weighting function with respect to $\boldsymbol{u}$, which is usually acceptable in practice, especially if the weighting function is a binary mask with relatively few transitions.

Baker *et al.* (2003a) only use the $w_0(\boldsymbol{x})$ term, which is reasonable if the two images have the same extent and no (independent) cutouts in the overlap region. They also discuss the idea of making the weighting proportional to $\nabla I(\boldsymbol{x})$, which helps for very noisy images, where the gradient itself is noisy. Similar observation, formulated in terms of *total least squares* (Huffel and Vandewalle 1991), have been made by other researchers studying optic flow (motion) estimation (Weber and Malik 1995). Lastly, Baker *et al.* (2003a) show how evaluating (87) at just the *most reliable* (highest gradient) pixels does not significantly reduce performance for large enough images, even if only 5%-10% of the pixels are used. (This idea was originally proposed by Dellaert and Collins (1999), who used a more sophisticated selection criterion.)

The Lucas-Kanade incremental refinement step can also be applied to the robust error metric introduced in §3.1,

$$E_{\mathrm{LK-SRD}}(\boldsymbol{u}+\Delta\boldsymbol{u}) = \sum_i \rho(\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} + e_i). \tag{88}$$

[13] In practice, it may be possible to decouple the bias-gain and motion update parameters, i.e., to solve two independent 2×2 systems, which is a little faster.

We can take the derivative of this function w.r.t. $\boldsymbol{u}$ and set it to 0,

$$\sum_i \psi(e_i) \frac{\partial e_i}{\partial \boldsymbol{u}} = \sum_i \psi(e_i) \boldsymbol{J}_1(\boldsymbol{x} + \boldsymbol{u}) = 0, \tag{89}$$

where $\Psi(e) = \rho'(e)$ is the derivative of ρ. If we introduce a weight function $w(e) = \Psi(e)/e$, we can write this as

$$\sum_i w(e_i) \boldsymbol{J}_1^T(\boldsymbol{x} + \boldsymbol{u})[\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i] = 0. \tag{90}$$

This results in the *Iteratively Re-weighted Least Squares* algorithm, which alternates between computing the weight functions $w(e_i)$ and solving the above weighted least squares problem (Huber 1981, Stewart 1999). Alternative incremental robust least squares algorithms can be found in (Sawhney and Ayer 1996, Black and Anandan 1996, Black and Rangarajan 1996, Baker *et al.* 2003a) and textbooks and tutorials on robust statistics (Huber 1981, Hampel *et al.* 1986, Rousseeuw and Leroy 1987, Stewart 1999).

3.5 Parametric motion

Many image alignment tasks, for example image stitching with handheld cameras, require the use of more sophisticated motion models, as described in §2. 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 (Lucas and Kanade 1981, Rehg and Witkin 1991, Fuh and Maragos 1991, Bergen *et al.* 1992, Baker and Matthews 2004).

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

$$\begin{aligned} E_{\mathrm{LK-PM}}(\boldsymbol{p} + \Delta\boldsymbol{p}) &= \sum_i [I_1(\boldsymbol{x}'(\boldsymbol{x}_i; \boldsymbol{p} + \Delta\boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 & (91) \\ &\approx \sum_i [I_1(\boldsymbol{x}'_i) + \boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} - I_0(\boldsymbol{x}_i)]^2 & (92) \\ &= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} + e_i]^2, & (93) \end{aligned}$$

where the Jacobian is now

$$\boldsymbol{J}_1(\boldsymbol{x}'_i) = \frac{\partial I_1}{\partial \boldsymbol{p}} = \nabla I_1(\boldsymbol{x}'_i) \frac{\partial \boldsymbol{x}'}{\partial \boldsymbol{p}}(\boldsymbol{x}_i), \tag{94}$$

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

Transform	Matrix	Parameters	Jacobian $\boldsymbol{J}_{\boldsymbol{x}'}$
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	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, \ldots, h_{21})$	(see text)

Table 2: *Jacobians of the 2D coordinate transformations.*

Table 2 shows the motion Jacobians $\boldsymbol{J}_{\boldsymbol{x}'}$ for the 2D planar transformations introduced in §2. Note how I have re-parameterized the motion matrices so that they are always the identity at the origin $\boldsymbol{p} = 0$. This will become useful below, when we talk about the compositional and inverse compositional algorithms. (It also makes it easier to impose priors on the motions.)

The derivatives in Table 2 are all fairly straightforward, except for the projective 2-D motion (homography), which requires a per-pixel division to evaluate, c.f. (8), re-written here in its new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \quad \text{and} \quad y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \tag{95}$$

The Jacobian is therefore

$$\boldsymbol{J}_{\boldsymbol{x}'} = \frac{\partial \boldsymbol{x}'}{\partial \boldsymbol{p}} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \tag{96}$$

where D is the denominator in (95), which depends on the current parameter settings (as do x' and y').

For parametric motion, the *Hessian* and *gradient-weighted residual vector* become

$$\boldsymbol{A} = \sum_i \boldsymbol{J}_{\boldsymbol{x}'}^T(\boldsymbol{x}_i)[\nabla I_1^T(\boldsymbol{x}_i')\nabla I_1(\boldsymbol{x}_i')]\boldsymbol{J}_{\boldsymbol{x}'}(\boldsymbol{x}_i) \tag{97}$$

and

$$\boldsymbol{b} = -\sum_i \boldsymbol{J}_{\boldsymbol{x}'}^T(\boldsymbol{x}_i)[e_i \nabla I_1^T(\boldsymbol{x}_i')]. \tag{98}$$

Note how the expressions inside the square brackets are the same ones evaluated for the simpler translational motion case (80–81).

Patch-based approximation. 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 $\boldsymbol{A}$ and $\boldsymbol{b}$ takes $\mathrm{O}(n^2 N)$ operations (Baker and Matthews 2004). 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 inside the square brackets at the pixel level (Shum and Szeliski 2000),

$$\boldsymbol{A}_j = \sum_{i \in P_j} \nabla I_1^T(\boldsymbol{x}_i') \nabla I_1(\boldsymbol{x}_i') \tag{99}$$

$$\boldsymbol{b}_j = \sum_{i \in P_j} e_i \nabla I_1^T(\boldsymbol{x}_i'). \tag{100}$$

The full Hessian and residual can then be approximated as

$$\boldsymbol{A} \approx \sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)[\sum_{i \in P_j} \nabla I_1^T(\boldsymbol{x}_i') \nabla I_1(\boldsymbol{x}_i')] \boldsymbol{J}_{\boldsymbol{x}'}(\hat{\boldsymbol{x}}_j) = \sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j) \boldsymbol{A}_j \boldsymbol{J}_{\boldsymbol{x}'}(\hat{\boldsymbol{x}}_j) \tag{101}$$

and

$$\boldsymbol{b} \approx -\sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)[\sum_{i \in P_j} e_i \nabla I_1^T(\boldsymbol{x}_i')] = -\sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j) \boldsymbol{b}_j, \tag{102}$$

where $\hat{\boldsymbol{x}}_j$ is the *center* of each patch P_j (Shum and Szeliski 2000). This is equivalent to replacing the true motion Jacobian with a piecewise-constant approximation. In practice, this works quite well. The relationship of this approximation to feature-based registration is discussed in §4.4.

Compositional approach 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 (1997) observed that this can be simplified by first warping the target image I_1 according to the current motion estimate $\boldsymbol{x}'(\boldsymbol{x}; \boldsymbol{p})$,

$$\tilde{I}_1(\boldsymbol{x}) = I_1(\boldsymbol{x}'(\boldsymbol{x}; \boldsymbol{p})), \tag{103}$$

and then comparing this *warped* image against the template $I_0(\boldsymbol{x})$,

$$\begin{aligned} E_{\mathrm{LK-SS}}(\Delta \boldsymbol{p}) &= \sum_i [\tilde{I}_1(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i; \Delta \boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 \\ &\approx \sum_i [\tilde{J}_1(\boldsymbol{x}_i) \Delta \boldsymbol{p} + e_i]^2 \qquad (104) \\ &= \sum_i [\nabla \tilde{I}_1(\boldsymbol{x}_i) \boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i) \Delta \boldsymbol{p} + e_i]^2. \qquad (105) \end{aligned}$$

Note that since the two images are assumed to be fairly similar, only an *incremental* parametric motion is required, i.e., the incremental motion can be evaluated around $\boldsymbol{p} = 0$, which can lead to considerable simplifications. For example, the Jacobian of the planar projective transform (95) now becomes

$$\boldsymbol{J}_{\tilde{\boldsymbol{x}}} = \frac{\partial \tilde{\boldsymbol{x}}}{\partial \boldsymbol{p}}\bigg|_{\boldsymbol{p}=0} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}. \tag{106}$$

Once the incremental motion $\tilde{\boldsymbol{x}}$ has been computed, it can be *prepended* to the previously estimated motion, which is easy to do for motions represented with transformation matrices, such as those given in Tables 1–2. Baker and Matthews (2004) call this the *forward compositional* algorithm, since the target image is being re-warped, and the final motion estimates are being composed.

If the appearance of the warped and template images is similar enough, we can replace the gradient of $\tilde{I}_1(\boldsymbol{x})$ with the gradient of $I_0(\boldsymbol{x})$, as suggested previously in (83). This has potentially a big advantage in that it allows the pre-computation (and inversion) of the Hessian matrix $\boldsymbol{A}$ given in (97). The residual vector $\boldsymbol{b}$ (98) can also be partially precomputed, i.e., the *steepest descent* images $\nabla I_0(\boldsymbol{x}) \boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x})$ can precomputed and stored for later multiplication with the $e(\boldsymbol{x}) = \tilde{I}_1(\boldsymbol{x}) - I_0(\boldsymbol{x})$ error images (Baker and Matthews 2004). This idea was first suggested by Hager and Belhumeur (1998) in what Baker and Matthews (2004) call a *forward additive* scheme.

Baker and Matthews (2004) introduce one more variant they call the *inverse compositional* algorithm. Rather than (conceptually) re-warping the warped target image $\tilde{I}_1(\boldsymbol{x})$, they instead warp the template image $I_0(\boldsymbol{x})$ and minimize

$$\begin{aligned} E_{\mathrm{LK-BM}}(\Delta \boldsymbol{p}) &= \sum_i [\tilde{I}_1(\boldsymbol{x}_i) - I_0(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i; \Delta \boldsymbol{p}))]^2 \\ &\approx \sum_i [\nabla I_0(\boldsymbol{x}_i) \boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i) \Delta \boldsymbol{p} - e_i]^2. \end{aligned} \tag{107}$$

This is identical to the forward warped algorithm (105) with the gradients $\nabla \tilde{I}_1(\boldsymbol{x})$ replaced by the gradients $\nabla I_0(\boldsymbol{x})$, except for the sign of e_i. The resulting update $\Delta \boldsymbol{p}$ is the *negative* of the one computed by the modified (105), and hence the *inverse* of the incremental transformation must be prepended to the current transform. Because the inverse compositional algorithm has the potential of pre-computing the inverse Hessian and the steepest descent images, this makes it the preferred approach of those surveyed in (Baker and Matthews 2004).

[Note: Add low-res copy of figure from PDFs, then ask Simon for permission to use Figure 2 from (Baker and Matthews 2004) and 3 from (Baker et al. *2003a), which show the contrast between the regular and inverse compositional algorithm. Similar (but much less elegant) diagrams appear in (Szeliski and Coughlan 1997), but don't bother mentioning this.]*

Baker and Matthews (2004) also discusses the advantage of using Gauss-Newton iteration (i.e., the first order expansion of the least squares, as above) vs. other approaches such as steepest descent and Levenberg-Marquardt. Subsequent parts of the series (Baker *et al.* 2003a, Baker *et al.*

2003b, Baker *et al.* 2004) discuss more advanced topics such as per-pixel weighting, pixel selection for efficiency, a more in-depth discussion of robust metrics and algorithms, linear appearance variations, and priors on parameters. They make for invaluable reading for anyone interested in implementing a highly tuned implementation of incremental image registration.

[Note: The following should be discussed later, e.g., in §4.3: Describe how 3D rotation updates are made incrementally (Szeliski and Shum 1997). Focal length estimation from homographies (Szeliski and Shum 1997), which should probably go in 3D motion models (see placeholder). Gap closing (Szeliski and Shum 1997).]

4 Feature-based registration

As I 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 next 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 (Hannah 1974, Hannah 1988) and has more recently gained popularity for image stitching applications (Zoghlami *et al.* 1997, Capel and Zisserman 1998, Cham and Cipolla 1998, Badra *et al.* 1998, McLauchlan and Jaenicke 2002, Brown and Lowe 2003).

In this section, I review methods for detecting distinctive points, for matching them, and for computing the image registration, including the 3D rotation model introduced in §2.2. I also discuss the relative advantages and disadvantages of direct and feature-based approaches.

4.1 Interest point detectors

As we saw in §3.4, the reliability of a motion estimate depends most critically on the size of the smallest eigenvalue of the image Hessian matrix, λ_0 (Anandan 1989). This makes it a reasonable candidate for finding points in the image that can be matched with high accuracy. (Older terminology in this field talked about "corner-like" features, but the modern usage is *interest points*.) Indeed, Shi and Tomasi (1994) propose using this quantity to find *good features to track*, and then use a combination of translational and affine-based patch alignment to track such points through an image sequence.

Using a square patch with equal weighting may not be the best choice. Instead, a Gaussian weighting function can be used. Förstner (1986) and Harris and Stephens (1988) both proposed finding interest points using such an approach. The Hessian and eigenvalue images can be efficiently evaluated using a sequence of filters and algebraic operations,

$$G_x(\boldsymbol{x}) = \frac{\partial}{\partial x} G_{\sigma_d}(\boldsymbol{x}) * I(\boldsymbol{x}), \tag{108}$$

$$G_y(\boldsymbol{x}) = \frac{\partial}{\partial y} G_{\sigma_d}(\boldsymbol{x}) * I(\boldsymbol{x}), \tag{109}$$

$$\boldsymbol{B}(\boldsymbol{x}) = \begin{bmatrix} G_x^2(\boldsymbol{x}) & G_x(\boldsymbol{x})G_y(\boldsymbol{x}) \\ G_x(\boldsymbol{x})G_y(\boldsymbol{x}) & G_y^2(\boldsymbol{x}) \end{bmatrix}, \tag{110}$$

$$\boldsymbol{A}(\boldsymbol{x}) = G_{\sigma_i}(\boldsymbol{x}) * \boldsymbol{B}(\boldsymbol{x}) \tag{111}$$

$$\lambda_{0,1}(\boldsymbol{x}) = \frac{a_{00} + a_{11} \mp \sqrt{(a_{00} - a_{11})^2 + a_{01}a_{10}}}{2}, \tag{112}$$

where G_{σ_d} is a noise-reducing pre-smoothing "derivative" filter of width σ_d, and G_{σ_i} is the integration filter whose scale σ_i controls the effective patch size. (The a_{ij} are the entries in the $\boldsymbol{A}(\boldsymbol{x})$ matrix, where I have dropped the $(\boldsymbol{x})$ for succinctness.) For example, Förstner (1994) uses $\sigma_d = 0.7$ and $\sigma_i = 2$. Once the minimum eigenvalue image has been computed, local maxima can be found as potential interest points.

The minimum eigenvalue is not the only quantity that can be used to find interest points. A simpler quantity, proposed by Harris and Stephens (1988) is

$$\det(\boldsymbol{A}) - \alpha \operatorname{trace}(\boldsymbol{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \tag{113}$$

with $\alpha = 0.06$. Triggs (2004) suggest using the quantity

$$\lambda_0 - \alpha\lambda_1 \tag{114}$$

(say with $\alpha = 0.05$), which reduces the response at 1D edges, where aliasing errors sometimes affect the smaller eigenvalue. He also shows how the basic 2×2 Hessian can be extended to parametric motions to detect points that are also accurately localizable in scale and rotation.

Schmid *et al.* (2000) survey the vast literature on interest point detection and perform some experimental comparisons to determine the *repeatability* of feature detectors, which is defined as the frequency with which interest points detected in one image are found within $\epsilon = 1.5$ pixels of the corresponding location in a warped image. They also measure the *information content* available at each detected feature point, which they define as the entropy of a set of rotationally invariant local grayscale descriptors. Among the techniques they survey, they find that an *improved* version of the Harris operator with $\sigma_d = 1$ and $\sigma_i = 2$ works best. (The original Harris and Stephens (1988) paper uses a discrete $[-2\ -1\ 0\ 1\ 2]$ filter to perform the initial derivative computations, and performs much worse.)

More recently, feature detectors that are more invariant to scale (Lowe 2004, Mikolajczyk and Schmid 2004) and affine transformations (Mikolajczyk and Schmid 2004) have been proposed. These can be very useful when matching images that have different scales or aspects (e.g., for 3D object recognition). A simple way to achieve scale invariance is to look for scale-space maxima in a Difference of Gaussian (DOG) (Lowe 2004) or Harris corner (Mikolajczyk and Schmid 2004,

Triggs 2004) detector over a sub-octave pyramid, i.e., an image pyramid where the subsampling between adjacent levels is less than a factor of two. Lowe's original (2004) paper uses a half-octave ($\sqrt{2}$) pyramid, whereas Triggs (2004) recommends using a quarter-octave ($\sqrt[4]{2}$).

Of course, interest points are not the only kind of features that can be used for registering images. Zoghlami *et al.* (1997) use line segments as well as point-like features to estimate homographies between pairs of images, whereas (Bartoli *et al.* 2004) use line segments with local correspondences along the edges to extract 3D structure and motion. Tuytelaars and Van Gool (2004) use affine invariant regions to detect correspondeces for wide baseline stereo matching. Matas *et al.* (2004) detect *maximally stable regions*, using an algorithm related to watershed detection, whereas Kadir *et al.* (2004) detect salient regions where patch entropy and its rate of change with scale are locally maximal. While these techniques can be used to solve image registration problems, they will not be covered in more detail in this survey.

4.2 Feature matching

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 (Shi and Tomasi 1994) or for stereo pairs that have been *rectified* (Loop and Zhang 1999, Scharstein and Szeliski 2002), the local motion around each feature point may be mostly translational. In this case, the error metrics introduced in §3.1 such as E_{SSD} or E_{NCC} can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid (2003) discussed below uses cross-correlation.) Because feature points may not be exactly located, a more accurate matching score can be computed by performing incremental motion refinement as described in §3.4, but this can be time consuming.

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. Shi and Tomasi (1994) compare patches using a translational model between neighboring frames, and then use the location estimate produced by this step to initialize an affine registration between the patch in the current frame and the base frame where a feature was first detected. In fact, features are only detected infrequently, i.e., only in region where tracking has failed. In the usual case, an area around the current *predicted* location of the feature is searched with an incremental registration algorithm. This kind of algorithm is a *detect then track* approach, since detection occurs infrequently. It is appropriate for video sequences where the expected locations of feature points can be reasonably well predicted.

For larger motions, or for matching collections of images where the geometric relationship between them is unknown (Schaffalitzky and Zisserman 2002, Brown and Lowe 2003), a *detect then match* approach in which feature points are first detected in all images is more appropriate.

Because the features can appear at different orientations or scales, a more *view invariant* kind of representation must be used. Mikolajczyk and Schmid (2003) 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 (2003) 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, as discussed in §4.1. Making the descriptors invariant to affine deformations (stretch, squash, skew) is even harder; Mikolajczyk and Schmid (2004) use the local second moment matrix around a feature point to define a canonical frame.

Among the local descriptors that Mikolajczyk and Schmid (2003) compared, In their experimental comparisons, Mikolajczyk and Schmid (2003) found that SIFT features generally performed best, followed by steerable filters,

David Lowe's (2004) Scale Invariant Feature Transform (SIFT) generally performed the best, followed by Freeman and Adelson's (1991) steerable filters and then cross-correlation (which could potentially be improved with an incremental refinement of location and pose). Differential invariants, whose descriptors are insensitive to changes in orientation by design, did not do as well.

SIFT features are computed by first estimating a local orientation using a histogram of the local gradient orientations, which is potentially more accurate than just the average orientation. Once the local frame has been established, gradients are copied into different orientation planes, and blurred resampled versions of these images as used as the features. This provides the descriptor with some insensitivity to small feature localization errors and geometric distortions (Lowe 2004).

Steerable filters are combinations of derivative of Gaussian filters that permit the rapid computation of even and odd (symmetric and anti-symmetric) edge-like and corner-like features at all possible orientations (Freeman and Adelson 1991). Because they use reasonably broad Gaussians, they too are somewhat insensitive to localization and orientation errors.

Rapid indexing and matching. 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 these are based on the idea of finding nearest neighbors in high-dimensional spaces. For example, Nene and Nayar (1997) developed a technique they call *slicing* that uses a series of 1D binary searches to efficiently cull down a list of candidate points that lie within a hypercube of the query point. They also provide a nice review of previous work in this area, including spatial data

structures such as k-d trees (Samet 1989). Beis and Lowe (1997) propose a Best-Bin-First (BBF) algorithm, which uses a modified search ordering for a k-d tree algorithm so that bins in feature space are searched in the order of their closest distance from query location. Shakhnarovich *et al.* (2003) extend a previously developed technique called *locality-sensitive hashing*, which uses unions of independently computed hashing functions, to be more sensitive to the distribution of points in parameter space, which they call *parameter-sensitive hashing*. Despite all of this promising work, the rapid computation of image feature correspondences is far from being a solved problem.

RANSAC and LMS. Once an initial set of feature correspondences has been computed, we need to find a set that is will produce a high-accuracy alignment. One possible approach is to simply compute a least squares estimate, or to use a robustified (iteratively re-weighted) version of least squares, as discussed below (§4.3). 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.[14]

Two widely used solution 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 a motion estimate $\boldsymbol{p}$, as described in §4.3. The *residuals* of the full set of correspondences are then computed as

$$\boldsymbol{r}_i = \tilde{\boldsymbol{x}}'_i(\boldsymbol{x}_i; \boldsymbol{p}) - \hat{\boldsymbol{x}}'_i, \tag{115}$$

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

The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location, i.e., whose $\|\boldsymbol{r}_i\| \leq \epsilon$. (The ϵ value is application dependent, but often is around 1-3 pixels.) Least median of squares finds the median value of the $\|\boldsymbol{r}_i\|$ 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. Either the initial parameter guess $\boldsymbol{p}$ or the full set of computed inliers is then passed on to the next data fitting stage.

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 tried. Let p be the probability that any given correspondence is valid, and P be the total 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 S such trials will all fail is

$$1 - P = (1 - p^k)^S \tag{116}$$

[14]For direct estimation methods, hierarchical (coarse-to-fine) techniques are often used to lock onto the *dominant motion* in a scene (Bergen *et al.* 1992).

and the required minimum number of trials is

$$S = \frac{\log(1-P)}{\log(1-p^k)}. \tag{117}$$

Stewart (1999) gives the following examples of the required number of trials S to attain a 99% probability of success:

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

.

As you can see, 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 in practice is how RANSAC is normally used.

4.3 Geometric registration

Once we have computed a set of matched feature point correspondences, the next stip is to estimate the motion parameters $\boldsymbol{p}$ 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 (115),

$$E_{\mathrm{LS}} = \sum_i \|\boldsymbol{r}_i\|^2 = \|\tilde{\boldsymbol{x}}'_i(\boldsymbol{x}_i;\boldsymbol{p}) - \hat{\boldsymbol{x}}'_i\|^2. \tag{118}$$

Many of the motion models presented in §2, i.e., translation, similarity, and affine, have a *linear* relationship between the motion and the unknown parameters $\boldsymbol{p}$.[15] In this case, a simple linear regression (least squares) using normal equations $\boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}$ works well.

Uncertainty weighting and robust regression. 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,

$$E_{\mathrm{WLS}} = \sum_i \sigma_i^{-2} \|\boldsymbol{r}_i\|^2. \tag{119}$$

As discussed in §3.4, a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the Hessian with the per-pixel noise estimate (84). Weighting each squared

[15] 2-D Euclidean motion can be estimated with a linear algorithm by first estimating the cosine and sine entries independently, and then normalizing them so their magnitude is 1.

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

$$E_{\text{CWLS}} = \sum_i \|\boldsymbol{r}_i\|^2_{\Sigma_i^{-1}} = \sum_i \boldsymbol{r}_i^T \Sigma_i^{-1} \boldsymbol{r}_i = \sum_i \sigma_n^{-2} \boldsymbol{r}_i^T \boldsymbol{A}_i \boldsymbol{r}_i, \tag{120}$$

where $\boldsymbol{A}_i$ is the *patch Hessian* (99).

If there are outliers among the feature-based correspondences (and there almost always are), 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 (analogous to (44)) is then

$$E_{\text{RLS}}(\boldsymbol{u}) = \sum_i \rho(\|\boldsymbol{r}_i\|_{\Sigma_i^{-1}}). \tag{121}$$

As before, a commonly used approach to minimize this quantity is to use iteratively re-weighted least squares, as described in §3.4.

Homography update. For *non-linear* measurement equations such as the homography given in (95), rewritten here as

$$\hat{x}' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \quad \text{and} \quad \hat{y}' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}, \tag{122}$$

an iterative solution is required to obtain accurate results. An initial guess for the 8 unknowns $\{h_{00}, \ldots, h_{21}\}$ can be obtained by multiplying both sides of the equations through by the denominator, which yields the linear set of equations,

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{123}$$

However, this is not optimal from a statistical point of view, since the denominator can vary quite a bit from point to point.

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

$$\frac{1}{D}\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{124}$$

While this may at first seem to be the exact same set of equations as (123), since least squares is being used to solve the over-determined set of equations, the weightings *do* matter, and result in a different set of normal equations that perform better in practice (with noisy data).

The most principled way to do the estimation, however, is to directly minimize the squared residual equations (115) using the Gauss-Newton approximation, i.e., performing a first-order Taylor series expansion in $\boldsymbol{p}$, which yields,

$$\hat{\boldsymbol{x}}_i' - \tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i; \boldsymbol{p}) = \boldsymbol{J}_{\boldsymbol{x}'} \Delta \boldsymbol{p} \tag{125}$$

or

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \tag{126}$$

While this looks similar to (124), it differs in two important respects. First, the left hand side consists of unweighted *prediction errors* rather than pixel displacements, and the solution vector is a *perturbation* to the parameter vector $\boldsymbol{p}$. Second the quantities inside $\boldsymbol{J}_{\boldsymbol{x}'}$ involve *predicted* feature locations $(\tilde{x}', \tilde{y}')$ instead of *sensed* feature locations $(\hat{x}', \hat{y}')$. Both of these 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 minimum. (Iterating the (124) equations is not guaranteed to do so, since it is not minimizing a well-defined energy function.)

The above formula is analagous to the *additive* algorithm for direct registration 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, we get $D = 1$ (since $\boldsymbol{p} = 0$) and the above formula simplifies to

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}, \tag{127}$$

where I have replaced $(\tilde{x}', \tilde{y}')$ with (x, y) for conciseness. (Notice how this results in the same Jacobian as (106).)

Rotational panorama update. As described in §2.2, representing the alignment of images in a panorama using a collection of rotation matrices and focal lengths results in a much more stable estimation problem than directly using homographies (Szeliski 1996, Szeliski and Shum 1997). Given this representation, how do we update the rotation matrices to best align two overlapping images?

Recall from (18–19) that the equations relating two views can be written as

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{H}}_{10} \tilde{\boldsymbol{x}}_0 \quad \text{with} \quad \tilde{\boldsymbol{H}}_{10} = \boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1}, \tag{128}$$

where $\boldsymbol{K}_k = \text{diag}(f_k, f_k, 1)$ is the calibration matrix and $\boldsymbol{R}_{10} = \boldsymbol{R}_1 \boldsymbol{R}_0^{-1}$ is rotation *between* the two views. The best way to update $\boldsymbol{R}_{10}$ is to prepend an *incremental* rotation matrix $\boldsymbol{R}(\vec{\omega})$ to the current estimate $\boldsymbol{R}_{10}$ (Szeliski and Shum 1997, Shum and Szeliski 2000),

$$\tilde{\boldsymbol{H}}(\vec{\omega}) = \boldsymbol{K}_1 \boldsymbol{R}(\vec{\omega}) \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1} = [\boldsymbol{K}_1 \boldsymbol{R}(\vec{\omega}) \boldsymbol{K}_1^{-1}][\boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1}] = \boldsymbol{D}\tilde{\boldsymbol{H}}_{10}. \tag{129}$$

Note that here I have written the update rule in the *compositional* form, where the incremental update $\boldsymbol{D}$ is *prepended* to the current homography $\tilde{\boldsymbol{H}}_{10}$. Using the small-angle approximation to $\boldsymbol{R}(\vec{\omega})$ given in (25), we can write the incremental update matrix as

$$\boldsymbol{D} = \boldsymbol{K}_1 \boldsymbol{R}(\vec{\omega}) \boldsymbol{K}_1^{-1} \approx \boldsymbol{K}_1 (\boldsymbol{I} + [\vec{\omega}]_\times) \boldsymbol{K}_1^{-1} = \begin{bmatrix} 1 & -\omega_z & f_1\omega_y \\ \omega_z & 1 & -f_1\omega_x \\ -\omega_y/f_1 & \omega_x/f_1 & 1 \end{bmatrix} .b \tag{130}$$

Notice how there is now a nice one-to-one correspondence between the entries in the $\boldsymbol{D}$ matrix and the $h_{00}, \ldots, h_{21}$ parameters used in Table 2 and (122), i.e.,

$$(h_{00}, h_{01}, h_{02}, h_{00}, h_{11}, h_{12}, h_{20}, h_{21}) = (0, -\omega_z, f_1\omega_y, \omega_z, 0, -f_1\omega_x, -\omega_y/f_1, \omega_x/f_1). \tag{131}$$

We can therefore apply the chain rule to (127) and (131) to obtain

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} -xy/f_1 & f_1 + x^2/f_1 & -y \\ -(f_1 + y^2/f_1) & xy/f_1 & x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{132}$$

which give us the linearized update equations needed to estimate $\vec{\omega} = (\omega_x, \omega_y, \omega_z)$.[16] 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. *[Note: see (Szeliski and Shum 1997) for some figures to add here.]*

The formulas for updating the focal length estimates are a little more involved, and are given in (Shum and Szeliski 2000). I will not repeat them here, since an alternative update rule, based on minimizing the difference between back-projected 3D rays, will be given in §5.1.

Focal length initialization *[Note: Copy the text here from (Szeliski and Shum 1997).]*

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

4.4 Direct vs. feature-based

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

I used to be firmly in the direct the direct matching camp (Triggs *et al.* 2000). 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 (Lowe 2004). Features not only respond to regions of high "cornerness" (Förstner 1986, Harris and Stephens 1988), but also to "blob-like" regions (Lowe 2004), as well as uniform areas (Tuytelaars and Van Gool 2004). Furthermore, 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 in working with feature-based approaches 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.* (2003a), who suggest that adding even more weight at strong gradients is preferable because of noise in the gradient estimates.) One could argue that for a blurry image with only slowly varying gradients, a direct approach will find an alignment, whereas a feature detector will fail to find anything. However, such images rarely occur in practice, and the use of scale-space features means that some features can be found at lower resolutions.

The biggest disadvantage of direct techniques is that they have a limited range of convergence. Even though they can be used in a hierarchical (coarse-to-fine) estimation framework, but in practice, it is hard to use more than two or three levels of a pyramid before important details start to be blurred away. 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. My older systems for image stitching (Szeliski 1996, Szeliski and Shum 1997) relied on Fourier-based correlation of cylindrical images and motion prediction to automatically align images, but had to be corrected by hand for more complex sequences. My newer system, built in collaboration with Matthew Brown (who first suggested recognizing panoramas), uses features and has a good success rate at automatically stitching panoramas without any user

intervention. *[Note: Add a cite here to MOPs paper, either as a MSR-TR, or just submission to CVPR.]*

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 estimate using patch-based alignment. Notice how there is a close correspondence between the patch-based approximation to direct alignment given in (101–102) and the inverse covariance weighted feature-based least squares error metric (120).

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 don't fit the selected motion model due to either parallax (§5.2) or moving objects (§6.2). 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), as we discuss in §5.2.

5 Global registration

So far, I have discussed how to register pairs of images using both direct and feature-based methods using a variety of motion models. 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 (Szeliski and Shum 1997, Shum and Szeliski 2000, Sawhney and Kumar 1999, Coorg and Teller 2000). In order to do this, we need to extend the pairwise matching criteria (43), (92), and (118) to a global energy function that involves all of the per-image pose parameters (§5.1). 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 mis-registrations (§5.2). 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 §5.3.

5.1 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* (Szeliski and Shum 1997). However, a better alternative is to simultaneously align all the images together using a least squares framework to correctly 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 (Triggs *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, 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 either by dividing images into patches and creating a virtual feature correspondence for each one (as discussed in §4.4 and (Shum and Szeliski 2000)), or by replacing the per-feature error metrics with per-pixel metrics.

Consider the feature-based alignment problem given in (118), i.e.,

$$E_{\text{pairwise}-\text{LS}} = \sum_i \|\boldsymbol{r}_i\|^2 = \|\tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i;\boldsymbol{p}) - \hat{\boldsymbol{x}}_i'\|^2. \tag{133}$$

For multi-image alignment, instead of having a single collection of pairwise feature correspondences, $\{(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i')\}$, we have a collection of n features, with the location of the ith feature point in the jth image denoted by $\boldsymbol{x}_{ij}$ and its scalar confidence (inverse variance) denoted by c_{ij}.[17] Each image also has some associated *pose* parameters.

In this section, I assume that this pose consists of a rotation matrix $\boldsymbol{R}_j$ and a focal length f_j, although formulations in terms of homographies are also possible (Shum and Szeliski 1997, Sawhney and Kumar 1999). The equation mapping a 3D point $\boldsymbol{x}_i$ into a point $\boldsymbol{x}_{ij}$ in frame j can be re-written from (15–19) as

$$\tilde{\boldsymbol{x}}_{ij} \sim \boldsymbol{K}_j \boldsymbol{R}_j \boldsymbol{x}_i \quad \text{and} \quad \boldsymbol{x}_i \sim \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} \tilde{\boldsymbol{x}}_{ij}, \tag{134}$$

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

$$\tilde{\boldsymbol{x}}_{ik} \sim \tilde{\boldsymbol{H}}_{kj} \tilde{\boldsymbol{x}}_{ij} = \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} \tilde{\boldsymbol{x}}_{ij}. \tag{135}$$

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

[17] Features that not seen in image j have $c_{ij} = 0$. We can also use 2×2 inverse covariance matrices Σ_{ij} in place of c_{ij}, as shown in (120).

One approach is to directly extend the pairwise energy $E_{\text{pairwise}-\text{LS}}$ (133) to a multiview formulation,

$$E_{\text{all}-\text{pairs}-\text{2D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\boldsymbol{x}}_{ik}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j, \boldsymbol{R}_k, f_k) - \hat{\boldsymbol{x}}_{ik}\|^2, \tag{136}$$

where the $\tilde{\boldsymbol{x}}_{ik}$ function is the *predicted* location of feature i in frame k given by (135), $\hat{\boldsymbol{x}}_{ij}$ is the *observed* location, and the "2D" in the subscript indicates than an image-plane error is being minimized (Shum and Szeliski 1997). Note that since $\tilde{\boldsymbol{x}}_{ik}$ depends on the $\hat{\boldsymbol{x}}_{ij}$ observed value, we actually have an *errors-in-variable* problem, which in principle requires more sophisticated techniques than least squares to solve. 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.[18]

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. (In effect, a feature observed m times gets counted $\binom{m}{2}$ times instead of m times.) Second, the derivatives of $\tilde{\boldsymbol{x}}_{ik}$ w.r.t. the $\{(\boldsymbol{R}_j, f_j)\}$ are a little cumbersome, although using the incremental correction to $\boldsymbol{R}_j$ introduced in §2.2 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 $\{(\boldsymbol{R}_j, f_j)\}$ but also for the 3D point positions $\{\boldsymbol{x}_i\}$,

$$E_{\text{BA}-\text{2D}} = \sum_i \sum_j c_{ij} \|\tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j) - \hat{\boldsymbol{x}}_{ij}\|^2, \tag{137}$$

where $\tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j)$ is given by (134). 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. (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 (Szeliski and Kang 1994, Shum and Szeliski 2000, Triggs *et al.* 1999).

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

$$E_{\text{BA}-\text{3D}} = \sum_i \sum_j c_{ij} \|\tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \boldsymbol{x}_i\|^2, \tag{138}$$

where $\tilde{\boldsymbol{x}}_i(\boldsymbol{x}_{ij}; \boldsymbol{R}_j, f_j)$ is given by the second half of (134). This in itself has no particular advantage over (137). 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.

[18] While there exists an overall pose ambiguity in the solution, i.e., all the $\boldsymbol{R}_j$ can be post-multiplied by an arbitrary rotation $\boldsymbol{R}_{\text{g}}$, a well-conditioned non-linear least squares algorithm such as Levenberg Marquardt will handle this degeneracy without trouble.

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

$$E_{\mathrm{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ik}; \boldsymbol{R}_k, f_k)\|^2. \tag{139}$$

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 (20). Thus, even though this formula over-weights features that occur more frequently, it is the method used both by Shum and Szeliski (2000) and in my current work. 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, and which seems to work well in practice.

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 do have a preference for the final stitched image to be "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 vertical scene lines parallel to the vertical edge of the image. In other words, the horizontal edge of your camera (its x-axis) usually stays 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 (134) that the 3D→2D projection is given by

$$\tilde{\boldsymbol{x}}_{ij} \sim \boldsymbol{K}_j \boldsymbol{R}_j \boldsymbol{x}_i. \tag{140}$$

We wish to post-multiply each rotation matrix $\boldsymbol{R}_k$ by a global rotation $\boldsymbol{R}_{\mathrm{g}}$ such that the projection of the global y-axis, $\hat{\boldsymbol{j}} = (0, 1, 0)$ is perpendicular to the image x-axis, $\hat{\boldsymbol{i}} = (1, 0, 0)$.[19]

This constraint can be written as

$$\hat{\boldsymbol{i}}^T \boldsymbol{R}_k \boldsymbol{R}_{\mathrm{g}} \hat{\boldsymbol{j}} = 0 \tag{141}$$

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

$$\boldsymbol{r}_{\mathrm{g}1} = \arg\min_{\boldsymbol{r}} \sum_k (\boldsymbol{r}^T \boldsymbol{r}_{k0})^2 = \arg\min_{\boldsymbol{r}} \boldsymbol{r}^T \left[\sum_k \boldsymbol{r}_{k0} \boldsymbol{r}_{k0}^T \right] \boldsymbol{r}. \tag{142}$$

[19] 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, $\boldsymbol{r}_{\mathrm{g}1}$ 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 $\boldsymbol{R}_{\mathrm{g}}$ global rotation, we need to specify one additional constraint. This is related to the *view selection* problem discussed in §6.1. One simple heuristic is to prefer the average z-axis of the individual rotation matrices, $\overline{\boldsymbol{k}} = \sum_k \hat{\boldsymbol{k}}^T \boldsymbol{R}_k$ to be close to the world z-axis, $\boldsymbol{r}_{\mathrm{g}2} = \boldsymbol{R}_{\mathrm{g}}\hat{\boldsymbol{k}}$. We can therefore compute the full rotation matrix $\boldsymbol{R}_{\mathrm{g}}$ in three steps:

1. $\boldsymbol{r}_{\mathrm{g}1} = \min$ eigenvector $(\sum_k \boldsymbol{r}_{k0}\boldsymbol{r}_{k0}^T)$;
2. $\boldsymbol{r}_{\mathrm{g}0} = \mathcal{N}((\sum_k \boldsymbol{r}_{k2}) \times \boldsymbol{r}_{\mathrm{g}1})$;
3. $\boldsymbol{r}_{\mathrm{g}2} = \boldsymbol{r}_{\mathrm{g}0} \times \boldsymbol{r}_{\mathrm{g}1}$,

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

5.2 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 optical center), 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 before the camera's first use) using one of the techniques discussed in §2.4. 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 mis-registration is reduced in image overlap areas (Stein 1997, Sawhney and Kumar 1999).

3D parallax can be attacked by doing a full 3D bundle adjustment, i.e., replacing the projection equation (134) used in (137) with (15), which models camera translations. The 3D positions of the matched features points and cameras can then be simultaneously recovered, although this can be significantly more expensive that 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, in order to do this, dense *stereo* correspondence needs to be performed (Kumar *et al.* 1995, Szeliski and Kang 1995, Scharstein and Szeliski 2002), which may not be possible if the images only contain 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 (Uyttendaele *et al.* 2004, Kang *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 *et al.* 2004), as discussed in §6.2. 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* (Shum and Szeliski 2000, Kang *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.

The local alignment technique introduced by Shum and Szeliski (2000) starts with the global bundle adjustment (139) used to optimize the camera poses. Once these have been estimated, the *desired* location of a 3D point $\boldsymbol{x}_i$ can be estimated as the *average* of the back-projected 3D locations,

$$\overline{\boldsymbol{x}}_i \sim \sum_j c_{ij} \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j), \tag{143}$$

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

$$\boldsymbol{u}_{ij} = \overline{\boldsymbol{x}}_{ij} - \boldsymbol{x}_{ij}, \tag{144}$$

which can be interpolated to form a dense correction field $\boldsymbol{u}_j(\boldsymbol{x}_j)$. In their system, Shum and Szeliski (2000) use an *inverse warping* algorithm where the sparse $-\boldsymbol{u}_{ij}$ values are placed at the new target locations $\overline{\boldsymbol{x}}_{ij}$, interpolated using bilinear kernel functions (Nielson 1993) and then added to the original pixel coordinates when computing the warped (corrected) image. In order 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 of features extracted using an interest operator.

An alternative approach to motion-based de-ghosting was proposed by Kang *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 therefore used to compute a high dynamic range radiance estimate, which is the goal of their overall algorithm. The requirement for having a reference image makes their approach less applicable to general image mosaicing, although an extension to this case could certainly be envisaged.

5.3 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 (2003) 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, 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 discover multiple panoramas taken by a user over an extended period of time can be a big convenience.

To recognize panoramas, Brown and Lowe (2003) 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 ???). *[Note: Add figure here from (Brown and Lowe 2003), then ask Matt Brown if it's o.k.]* The feature-based matching stage first extracts SIFT feature locations and feature descriptors (Lowe 2004) from all the input images and then places these in an indexing structure, as described in §4.2. 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 then used to find a set of *inlier* matches, using a pairs of matches to hypothesize a similarity motion model that is then used to count the number of inliers.

[Note: The following still needs to be written:

In practice, may be erroneous matches, need a way to "back out" or make the process more robust.

Show example where solution is ambiguous: Doorway w/ tree or cathedral with windows vs. moving persons]

6 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, I review techniques that address these problems, namely compositing surface parameterization, pixel/seam selection, blending, and exposure compensation. My 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, e.g., (Agarwala *et al.* 2004, Li *et al.* 2004a, Rother *et al.* 2004), which I will not cover, except where they provide automated solutions to our problems.

6.1 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 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 (which is often a desirable attribute).

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 (Szeliski 1994, Chen 1995) or spherical (Szeliski and Shum 1997) projection, as described in §2.3. 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 (Greene 1986, Szeliski and Shum 1997). Cartographers have also developed a number of alternative methods for representing the globe *[Note: find a reference here; ask MattU?]*.

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.

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.

For larger (e.g., cylindrical or spherical) panoramas, we can still use the same heuristic if a subset of the viewing sphere has been imaged. If the case of full 360° panoramas, a better choice might be 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 §5.1 is working correctly, this can be as simple as panning over the image or setting a vertical "center line" for the final panorama. *[Note: Add a figure here showing the difference before and after up vector estimation.]*

Coordinate transformations. Once we have selected 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 (19). 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 be possible to accelerate this process by computing exact pixel mappings on a coarser grid and then interpolating these values. An efficient way to roughly know which portions of the final panorama are covered by which input images can also be helpful.

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

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, pre-filtering 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 pre-filter, which depends on the distance (and arrangement) between neighboring samples in a source image. 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 pre-filter may be necessary (Wang *et al.* 2001). Under certain conditions, it may also be possible to produce images with a higher resolution than the input images using a process called *super-resolution* (§7).

Figure 9: *Final composites computed by a variety of algorithms: (a) average, (b) median, (c) feathered average, (d)* p-norm $p = ?$*, (e) Vornoi, (f) weighted ROD vertex cover with feathering, (g) graph cut seams, (h) graph cut seams with Poisson blending.*

6.2 Pixel selection and weighting

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 (any pixel or 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 though 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,

$$C(\boldsymbol{x}) = \sum_k w_k(\boldsymbol{x}) \tilde{I}_k(\boldsymbol{x}) \Big/ \sum_k w_k(\boldsymbol{x}) \ , \tag{145}$$

where $\tilde{I}_k(\boldsymbol{x})$ are the *warped* (re-sampled) images and $w_k(\boldsymbol{x})$ 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, mis-registrations, and scene movement are all very visible (Figure 9a). 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 (Irani and Anandan 1998) (Figure 9b). Conversely, center-weighting (discussed below) and *minimum likelihood* selection (Agarwala *et al.* 2004) can sometimes be used to retain multiple copies of a moving object. *[Note: Insert the figure with the moving snowboarder?]*

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

$$w_k(\boldsymbol{x}) = \left\| \arg\min_{\boldsymbol{y}} \{ \|\boldsymbol{y}\| \mid \tilde{I}_k(\boldsymbol{x}+\boldsymbol{y}) \text{ is invalid } \} \right\|, \tag{146}$$

where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel. 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, Uyttendaele *et al.* 2001), and does a reasonable job of blending over exposure differences. However, blurring and ghosting can still be problems (Figure 9c). Note that weighted averaging is *not* the same as compositing the individual images with the classic *over* operation (Porter and Duff 1984, Blinn 1994), 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 $w_k^p(\boldsymbol{x})$ in (145). 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 9d).

In the limit as $p \to \infty$, only the pixel with the maximum distance value gets selected,

$$C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}), \tag{147}$$

where

$$l = \arg\max_k w_k(\boldsymbol{x}) \tag{148}$$

is the *label assignment* or *pixel selection* function that selects which image to use at each pixel. This hard pixel selection process produces a visibility mask-sensitive variant of the familiar *Vornoi diagram*, which assigns each pixel to the nearest image center in the set (Wood *et al.* 1997, Peleg *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 9e).

Xiong and Turkowski (1998) use this Vornoi 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 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 (Davis 1998). For a pair of images, this process can be formulated as a simple dynamic program starting from one (short) edge of the overlap region and ending at the other (Milgram 1975, Milgram 1977, Davis 1998, Efros and Freeman 2001).

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 *et al.* (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. *[Note: Add a figure here, taken from the paper.]* 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 (Uyttendaele *et al.* 2001). The algorithm therefore prefers removing regions that are near the edge of the image, which reduces the likelihood that objects which are only partially visible 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 9f).

A different approach to pixel selection and seam placement was recently proposed by Agarwala *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,

$$\mathcal{C}_D = \sum_{\boldsymbol{x}} D_{l(\boldsymbol{x})}(\boldsymbol{x}), \tag{149}$$

where $D_{l(\boldsymbol{x})}(\boldsymbol{x})$ is the *data penalty* associated with choosing image l at pixel $\boldsymbol{x}$. In their system, users can select which pixels to use by "painting" over an image with the desired object or appearance, which sets $D(\boldsymbol{x}, l)$ to a large value for all labels l other than the one selected by the user. 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* that penalizes differences in labelings between adjacent images,

$$C_S = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{N}} S_{l(\boldsymbol{x}),l(\boldsymbol{y})}(\boldsymbol{x},\boldsymbol{y}) \tag{150}$$

where $S_{l(\boldsymbol{x}),l(\boldsymbol{y})}(\boldsymbol{x},\boldsymbol{y})$ is the image-dependent *interaction penalty* or *seam cost* of placing a seam between pixels $\boldsymbol{x}$ and $\boldsymbol{y}$, and $\mathcal{N}$ is the set of N_4 neighboring pixels. For example, the simple color-based seam penalty used in (Kwatra *et al.* 2003, Agarwala *et al.* 2004) can be written as

$$S_{l(\boldsymbol{x}),l(\boldsymbol{y})}(\boldsymbol{x},\boldsymbol{y}) = \|\tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}) - \tilde{I}_{l(\boldsymbol{y})}(\boldsymbol{x})\| + \|\tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{y}) - \tilde{I}_{l(\boldsymbol{y})}(\boldsymbol{y})\|. \tag{151}$$

More sophisticated seam penalties can also look at image gradients or the presence of image edges (Agarwala *et al.* 2004). Seam penalties are widely used in other computer vision applications such as stereo matching (Boykov *et al.* 2001) to give the labeling function its *coherence* or *smoothness*.

The sum of the two objective functions is often called the *Markov Random Field* (MRF) energy, since it arises as the negative log-likelihood of an MRF distribution (Geman and Geman 1984). For general energy functions, finding the minimum can be NP-hard (Boykov *et al.* 2001). However, a variety of approximate optimization techniques have been developed over the years, including *simulated annealing* (Geman and Geman 1984), graph cuts (Boykov *et al.* 2001), and loopy belief propagation (Tappen and Freeman 2003). Both Kwatra *et al.* (2003) and Agarwala *et al.* (2004) 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 (Boykov *et al.* 2001).

For the result shown in Figure 9g, Agarwala *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 differences, including those that border the image and which might result in cut off objects. Graph cuts (Agarwala *et al.* 2004) and vertex cover (Uyttendaele *et al.* 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.

6.3 Blending

Once the seams have been placed and unwanted object removed, we still need to blend the images to compensate for exposure differences and other mis-alignments. 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 does help).

Laplacian pyramid blending. An attractive solution to this problem was developed by Burt and Adelson (1983). 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. The process operates as follows.

First, each warped image is converted into a band-pass (Laplacian) pyramid, which involves smoothing each level with a $1/16(1, 4, 6, 4, 1)$ binomial kernel, subsampling the smoothed image by a factor of 2, and subtracting the reconstructed (low-pass) image from the original. This creates a reversible, overcomplete representation of the image signal. Invalid and edge pixels are filled with neighboring values to make this process well defined.

Next, the *mask* (valid pixel) image associated with each source image is converted into a low-pass (Gaussian) pyramid. These blurred and subsampled masks become the weights used to perform a per-level feathered blend of the band-pass source images.

Finally, the composite image is reconstructed by interpolating and summing all of the pyramid levels (band-pass images). The result of applying this pyramid blending is shown in Figure ?. *[Note: Need to add a figure here.]*

Gradient domain blending. An alternative approach to multi-badn 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). More recently, related ideas have been used for reconstructing images from their edges (Elder and Golderg 2001), removing shadows from images (Weiss 2001), and *tone mapping* high dynamic range images by reducing the magnitude of image edges (gradients) (Fattal *et al.* 2002).

Pérez *et al.* (2003) showed how gradient domain reconstruction can be used to do seamless object insertion in image editing applications. Rather than copying pixels, the *gradients* of the new image fragment are copied instead. The actual pixel values for the copied image 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 *et al.* (2003) show that this is equivalent to computing an additive *membrane* interpolant of the mismatch between the source and destination images along the boundary. (The membrane interpolant is known to have nicer interpolation properties for arbitrary-shaped constraints than frequency-domain interpolants (Nielson 1993).) In prior work, Peleg (1981) also proposed adding a smooth function to force a consistency along the seam curve.

Agarwala *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 *et al.* (2004) directly minimize *variational problem*,

$$\min_{C(\boldsymbol{x})} \|\nabla C(\boldsymbol{x}) - \nabla \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x})\|^2. \tag{152}$$

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

$$C(\boldsymbol{x}+\hat{\boldsymbol{\imath}}) - C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}+\hat{\boldsymbol{\imath}}) - \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}) \quad \text{and} \tag{153}$$

$$C(\boldsymbol{x}+\hat{\boldsymbol{\jmath}}) - C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}+\hat{\boldsymbol{\jmath}}) - \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}), \tag{154}$$

where $\hat{\boldsymbol{\imath}} = (1, 0)$ and $\hat{\boldsymbol{\jmath}} = (0, 1)$ are unit vectors in the x and y directions.[20] They then solve the associated sparse least squares problem. Since this system of equations is only defined up to an additive constraint, Agarwala *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 *et al.* 2002) use multigrid, whereas (Agarwala *et al.* 2004) have recently been using hierarchical basis preconditioned conjugate gradient descent (Szeliski 1990). The resulting seam blending work very well in practice (Figure 9h), 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 *et al.* (2004) examines 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 as in (152). 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.* (2004)'s approach) 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.* (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

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

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 be repeated (typically 3 times). The results in (Uyttendaele *et al.* 2001) 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 to exposure compensation is to estimate a single *high dynamic range* (HDR) radiance map from of the differently exposed images (Mann and Picard 1995, Debevec and Malik 1997, Mitsunaga and Nayar 1999). All of these papers assume that the input images were taken with a fixed camera whose pixel values

$$I_k(\boldsymbol{x}) = f(c_k R(\boldsymbol{x}); \boldsymbol{p}) \tag{155}$$

are the result of applying a parameterized *radiometric transfer function* $f(R, \boldsymbol{p})$ to scaled radiance values $c_k R(\boldsymbol{x})$. 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 fitting process.

The form of the parametric function differs in each of these papers. Mann and Picard (1995) use a three-parameter $f(R) = \alpha + \beta R^\gamma$ function, Debevec and Malik (1997) use a thin-plate cubic spline, while Mitsunaga and Nayar (1999) use a low-order ($N \leq 10$) polynomial for the *inverse* of the transfer function.

To blend the estimated (noisy) radiance values into a final composite, Mann and Picard (1995) use a hat function (accentuating mid-tone pixels), Debevec and Malik (1997) use the derivative of the response function, while Mitsunaga and Nayar (1999) optimize the signal-to-noise ratio (SNR), which emphasizes both higher pixel values and larger gradients in the transfer function.

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 (Fattal *et al.* 2002, Durand and Dorsey 2002, Reinhard *et al.* 2002)

Unfortunately, casually acquired images may not be perfectly registered and may contain moving objects. Kang *et al.* (2003) 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.

Even this approach, however, may not work when the camera is simultaneously undergoing large panning motions and exposure changes, which is a common occurrence in casually acquired

panoramas. Under such conditions, different parts of the image may be seen at one or more exposures. Devising a method to blend all of these different sources while avoiding sharp transitions and dealing with scene motion is a challenging open research problem.

In the long term, the need to compute high dynamic range images from multiple exposures may be eliminated by advances in camera sensor technology (Nayar and Mitsunaga 2000, Kang *et al.* 2003). However, the need to blend such images and to tone map them to a pleasing final result will likely remain.

7 Extensions and open issues

Multiple resolutions (zoom) and super-resolution (Keren *et al.* 1988, Irani and Peleg 1991, Cheeseman *et al.* 1993, Mann and Picard 1994, Chiang and Boult 1996, Bascle *et al.* 1996, Capel and Zisserman 1998, Capel and Zisserman 2000, Capel and Zisserman 2001, Smelyanskiy *et al.* 2000).

Video stitching (Irani and Anandan 1998); adding temporal elements (Sarnoff's mosaics with video (Lamplight?)); VideoBrush (Sawhney *et al.* 1998); see also Salient Stills (Teodosio and Bender 1993)) or (Massey & Bender, IBM Systems Journal 1996).

Peleg's manifold mosaics (Peleg *et al.* 2000) and stereo mosaics (where)? Strip mosaics for artistic rendering and multi-plane pan (Wood *et al.* 1997). Multi-center-of-projection images (Rademacher and Bishop 1998).

3-D parallax (Kumar *et al.* 1995).

Concentric mosaics (Shum and He 1999, Shum and Szeliski 1999, Li *et al.* 2004b).

Other applications: document scanning with a mouse (Nakao *et al.* 1998); retinal image mosaics (Can *et al.* 2002).

Open issues. How to really make these things work automatically: repeated pattern, matching subsets, moving objects, parallax. Hard to get the last 3%. (Mention internal test data suite, shipping in product.)

Automated object removal: like intelligent PhotoMontage (semantic stitching, photographer's assistant)

Large parallax: need to do 3D reconstruction. But, not possible if no overlap in some regions (MPPS gets around this with a hack). Ideally, want 70% overlap to tie inter-frame motions strongly together (also for better blending). Video-rate cameras with on-board stitching may some day solve this...

References

De Castro, E. and Morandi, C. (1987). Registration of translated and rotated iimages using finite fourier transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9(5)*, 700–703.

OpenGL ARB. (1997). *OpenGL Reference Manual : The Official Reference Document to OpenGL, Version 1.1*. Addison-Wesley, Reading, MA, 2 edition.

Agarwala, A. *et al.*. (2004). Interactive digital photomontage. *ACM Transactions on Graphics, 23(3)*, 292–300.

Anandan, P. (1989). A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision, 2(3)*, 283–310.

Ayache, N. (1989). *Vision Stéréoscopique et Perception Multisensorielle*. InterEditions., Paris.

Badra, F., Qumsieh, A., and Dudek, G. (1998). Rotation and zooming in image mosaicing. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 50–55, IEEE Computer Society, Princeton.

Baker, S. and Matthews, I. (2004). 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.

Baker, S. *et al.*. (2003a). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 2*. Technical Report CMU-RI-TR-03-01, The Robotics Institute, Carnegie Mellon University.

Baker, S. *et al.*. (2003b). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 3*. Technical Report CMU-RI-TR-03-35, The Robotics Institute, Carnegie Mellon University.

Baker, S. *et al.*. (2004). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 4*. Technical Report CMU-RI-TR-04-14, The Robotics Institute, Carnegie Mellon University.

Bartoli, A., Coquerelle, M., and Sturm, P. (2004). A framework for pencil-of-points structure-from-motion. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 28–40, Springer-Verlag, Prague.

Bascle, B., Blake, A., and Zisserman, A. (1996). Motion deblurring and super-resolution from an image sequence. In *Fourth European Conference on Computer Vision (ECCV'96)*, pages 573–582, Springer-Verlag, Cambridge, England.

Beis, J. S. and Lowe, D. G. (1997). Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 1000–1006, San Juan, Puerto Rico.

Benosman, R. and Kang, S. B., editors. (2001). *Panoramic Vision: Sensors, Theory, and Applications*, Springer, New York.

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

Black, M. J. and Anandan, P. (1996). The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding, 63(1)*, 75–104.

Black, M. J. and Jepson, A. D. (1998). EigenTracking: robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision, 26(1)*, 63–84.

Black, M. J. and Rangarajan, A. (1996). On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision, 19(1)*, 57–91.

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

Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics and Image Processing, 34(3)*, 227–248.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11)*, 1222–1239.

Brown, D. C. (1971). Close-range camera calibration. *Photogrammetric Engineering, 37(8)*, 855–866.

Brown, L. G. (1992). A survey of image registration techniques. *Computing Surveys, 24(4)*, 325–376.

Brown, M. and Lowe, D. (2003). Recognizing panoramas. In *Ninth International Conference on Computer Vision (ICCV'03)*, pages 1218–1225, Nice, France.

Burt, P. J. and Adelson, E. H. (1983). A multiresolution spline with applications to image mosaics. *ACM Transactions on Graphics, 2(4)*, 217–236.

Can, A. *et al.*. (2002). A feature-based, robust, hierarchical algorithm for registering pairs of images of the curved human retina. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(3)*, 347–364.

Capel, D. and Zisserman, A. (1998). Automated mosaicing with super-resolution zoom. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 885–891, Santa Barbara.

Capel, D. and Zisserman, A. (2000). Super-resolution enhancement of text image sequences. In *Fifteenth International Conference on Pattern Recognition (ICPR'2000)*, pages 600–605, IEEE Computer Society Press, Barcelona, Spain.

Capel, D. and Zisserman, A. (2001). Super-resolution from multiple views using learnt image models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 627–634, Kauai, Hawaii.

Cham, T. J. and Cipolla, R. (1998). A statistical framework for long-range feature matching in uncalibrated image mosaicing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 442–447, Santa Barbara.

Cheeseman, P., Kanefsky, B., Hanson, R., and Stutz, J. (1993). *Super-Resolved Surface Reconstruction From Multiple Images*. Technical Report FIA-93-02, NASA Ames Research Center, Artificial Intelligence Branch.

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

Chiang, M.-C. and Boult, T. E. (1996). Efficient image warping and super-resolution. In *IEEE Workshop on Applications of Computer Vision (WACV'96)*, pages 56–61, IEEE Computer Society, Sarasota.

Coorg, S. and Teller, S. (2000). Spherical mosaics with quaternions and dense correlation. *International Journal of Computer Vision, 37(3)*, 259–273.

Cox, I. J., Roy, S., and Hingorani, S. L. (1995). Dynamic histogram warping of image pairs for constant image brightness. In *IEEE International Conference on Image Processing (ICIP'95)*, pages 366–369, IEEE Computer Society.

Danielsson, P. E. (1980). Euclidean distance mapping. *Computer Graphics and Image Processing, 14(3)*, 227–248.

Davis, J. (1998). Mosaics of scenes with moving objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 354–360, Santa Barbara.

Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. *Proceedings of SIGGRAPH 97*, , 369–378. ISBN 0-89791-896-7. Held in Los Angeles, California.

Dellaert, F. and Collins, R. (1999). Fast image-based tracking by selective pixel integration. In *ICCV Workshop on Frame-Rate Vision*, pages 1–22.

Durand, F. and Dorsey, J. (2002). Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (TOG), 21(3)*, 257–266.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 341–346, ACM Press / ACM SIGGRAPH.

El-Melegy, M. and Farag, A. (2003). Nonmetric lens distortion calibration: Closed-form solutions, robust estimation and model selection. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 554–559, Nice, France.

Elder, J. H. and Golderg, R. M. (2001). Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(3)*, 291–296.

Fattal, R., Lischinski, D., and Werman, M. (2002). Gradient domain high dynamic range compression. *ACM Transactions on Graphics (TOG), 21(3)*, 249–256.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM, 24(6)*, 381–395.

Förstner, W. (1986). A feature-based correspondence algorithm for image matching. *Int.l Arch. Photogrammetry & Remote Sensing, 26(3)*, 150–166.

Förstner, W. (1994). A framework for low level feature extraction. In *Third European Conference on Computer Vision (ECCV'94)*, pages 383–394, Springer-Verlag, Stockholm, Sweden.

Freeman, W. T. and Adelson, E. H. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(9)*, 891–906.

Fuh, C.-S. and Maragos, P. (1991). Motion displacement estimation using an affine model for image matching. *Optical Engineering, 30(7)*, 881–887.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6(6)*, 721–741.

Gennert, M. A. (1988). Brightness-based stereo matching. In *Second International Conference on Computer Vision (ICCV'88)*, pages 139–143, IEEE Computer Society Press, Tampa.

Golub, G. and Van Loan, C. F. (1996). *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London.

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

Greene, N. and Heckbert, P. (1986). Creating raster Omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications, 6(6)*, 21–27.

Hager, G. D. and Belhumeur, P. N. (1998). Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(10)*, 1025–1039.

Hampel, F. R. *et al.*. (1986). *Robust Statistics : The Approach Based on Influence Functions*. Wiley, New York.

Hannah, M. J. (1974). *Computer Matching of Areas in Stereo Images*. Ph.D. thesis, Stanford University.

Hannah, M. J. (1988). Test results from SRI's stereo system. In *Image Understanding Workshop*, pages 740–744, Morgan Kaufmann Publishers, Cambridge, Massachusetts.

Hansen, M., Anandan, P., Dana, K., van der Wal, G., and Burt, P. (1994). Real-time scene stabilization and mosaic construction. In *IEEE Workshop on Applications of Computer Vision (WACV'94)*, pages 54–62, IEEE Computer Society, Sarasota.

Harris, C. and Stephens, M. J. (1988). A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–152.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry*. Cambridge University Press, Cambridge, UK.

Horn, B. K. P. (1974). Determining lightness from an image. *Computer Graphics and Image Processing, 3(1)*, 277–299.

Horn, B. K. P. (1986). *Robot Vision*. MIT Press, Cambridge, Massachusetts.

Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press, Cambridge, Massachusetts.

Horn, B. K. P. and Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*, 185–203.

Huber, P. J. (1981). *Robust Statistics*. John Wiley & Sons, New York.

Huffel, S. v. and Vandewalle, J. (1991). *The Total Least Squares Problem: Computational Aspects and Analysis*. Society for Industrial and Applied Mathematics, Philadephia.

Irani, M. and Anandan, P. (1998). Video indexing based on mosaic representations. *Proceedings of the IEEE, 86(5)*, 905–921.

Irani, M. and Peleg, S. (1991). Improving resolution by image registration. *Graphical Models and Image Processing*, *53(3)*, 231–239.

Irani, M., Hsu, S., and Anandan, P. (1995). Video compression using mosaic representations. *Signal Processing: Image Communication, 7*, 529–552.

Jia, J. and Tang, C.-K. (2003). Image registration with global and local luminance alignment. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 156–163, Nice, France.

Jurie, F. and Dhome, M. (2002). Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7)*, 996–1000.

Kadir, T., Zisserman, A., and Brady, M. (2004). An affine invariant salient region detector. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 228–241, Springer-Verlag, Prague.

Kang, S. B. (2001). Radial distortion snakes. *IEICE Trans. Inf. & Syst., E84-D(12)*, 1603–1611.

Kang, S. B. *et al.*. (2003). High dynamic range video. *ACM Transactions on Graphics, 22(3)*, 319–325.

Kang, S. B., Szeliski, R., and Uyttendaele, M. (2004). *Seamless Stitching using Multi-Perspective Plane Sweep*. Technical Report MSR-TR-2004-48, Microsoft Research.

Keren, D., Peleg, S., and Brada, R. (1988). Image sequence enhancement using sub-pixel displacements. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'88)*, pages 742–746, IEEE Computer Society Press, Ann Arbor, Michigan.

Kuglin, C. D. and Hines, D. C. (1975). The phase correlation image alignment method. In *IEEE 1975 Conference on Cybernetics and Society*, pages 163–165, New York.

Kumar, R., Anandan, P., and Hanna, K. (1994). Direct recovery of shape from multiple views: A parallax based approach. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, pages 685–688, IEEE Computer Society Press, Jerusalem, Israel.

Kumar, R., Anandan, P., Irani, M., Bergen, J., and Hanna, K. (1995). Representation of scenes from collections of images. In *IEEE Workshop on Representations of Visual Scenes*, pages 10–17, Cambridge, Massachusetts.

Kwatra, V. *et al.*. (2003). Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, 22(3)*, 277–286.

Le Gall, D. (1991). MPEG: A video compression standard for multimedia applications. *Communications of the ACM, 34(4)*, 44–58.

Lee, M.-C. *et al.*. (1997). A layered video object coding system using sprite and affine motion model. *IEEE Transactions on Circuits and Systems for Video Technology, 7(1)*, 130–145.

Levin, A., Zomet, A., Peleg, S., and Weiss, Y. (2004). Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 377–389, Springer-Verlag, Prague.

Li, Y. *et al.*. (2004a). Lazy snapping. *ACM Transactions on Graphics, 23(3)*, 303–308.

Li, Y. *et al.*. (2004b). Stereo reconstruction from multiperspective panoramas. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(1)*, 44–62.

Loop, C. and Zhang, Z. (1999). Computing rectifying homographies for stereo vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 125–131, Fort Collins.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, 60(2)*, 91–110.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 674–679, Vancouver.

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

Mann, S. and Picard, R. W. (1995). On being 'undigital' with digital cameras: Extending dynamic range by combining differently exposed pictures. In *IS&T's 48th Annual Conference*, pages 422–428, Society for Imaging Science and Technology, Washington, D. C.

Matas, J. *et al.*. (2004). Robust wide baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, *22(10)*, 761–767.

Matthies, L. H., Szeliski, R., and Kanade, T. (1989). Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, *3*, 209–236.

McLauchlan, P. F. and Jaenicke, A. (2002). Image mosaicing using sequential bundle adjustment. *Image and Vision Computing*, *20(9-10)*, 751–759.

Meehan, J. (1990). *Panoramic Photography*. Watson-Guptill.

Mikolajczyk, K. and Schmid, C. (2003). A performance evaluation of local descriptors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 257–263, Madison, WI.

Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, *60(1)*, 63–86.

Milgram, D. L. (1975). Computer methods for creating photomosaics. *IEEE Transactions on Computers*, *C-24(11)*, 1113–1119.

Milgram, D. L. (1977). Adaptive techniques for photomosaicking. *IEEE Transactions on Computers*, *C-26(11)*, 1175–1180.

Mitsunaga, T. and Nayar, S. K. (1999). Radiometric self calibration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 374–380, Fort Collins.

Nakao, T., Kashitani, A., and Kaneyoshi, A. (1998). Scanning a document with a small camera attached to a mouse. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 63–68, IEEE Computer Society, Princeton.

Nayar, S. K. and Mitsunaga, T. (2000). High dynamic range imaging: Spatially varying pixel exposures. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 472–479, Hilton Head Island.

Nene, S. and Nayar, S. K. (1997). A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19(9)*, 989–1003.

Nielson, G. M. (1993). Scattered data modeling. *IEEE Computer Graphics and Applications, 13(1)*, 60–70.

Okutomi, M. and Kanade, T. (1993). A multiple baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(4)*, 353–363.

Oppenheim, A. V., Schafer, R. W., and Buck, J. R. (1999). *Discrete-Time Signal Processing*. Pearson Education, 2nd edition.

Peleg, S. (1981). Elimination of seams from photomosaics. *Computer Vision, Graphics, and Image Processing, 16*, 1206–1210.

Peleg, S. *et al.*. (2000). Mosaicing on adaptive manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(10)*, 1144–1154.

Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Transactions on Graphics, 22(3)*, 313–318.

Porter, T. and Duff, T. (1984). Compositing digital images. *Computer Graphics (SIGGRAPH'84), 18(3)*, 253–259.

Quam, L. H. (1984). Hierarchical warp stereo. In *Image Understanding Workshop*, pages 149–155, Science Applications International Corporation, New Orleans.

Rademacher, P. and Bishop, G. (1998). Multiple-center-of-projection images. In *Computer Graphics Proceedings, Annual Conference Series*, pages 199–206, ACM SIGGRAPH, Proc. SIGGRAPH'98 (Orlando).

Rehg, J. and Witkin, A. (1991). Visual tracking with deformation models. In *IEEE International Conference on Robotics and Automation*, pages 844–850, IEEE Computer Society Press, Sacramento.

Reinhard, E. *et al.*. (2002). Photographic tone reproduction for digital images. *ACM Transactions on Graphics (TOG), 21(3)*, 267–276.

Rother, C., Kolmogorov, V., and Blake, A. (2004). "GrabCut" - interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics, 23(3)*, 309–314.

Rousseeuw, P. J. (1984). Least median of squares regresssion. *Journal of the American Statistical Association, 79*, 871–880.

Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust Regression and Outlier Detection*. Wiley, New York.

Samet, H. (1989). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts.

Sawhney, H. S. (1994). 3D geometry from planar parallax. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 929–934, IEEE Computer Society, Seattle.

Sawhney, H. S. and Ayer, S. (1996). Compact representation of videos through dominant multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *18(8)*, 814–830.

Sawhney, H. S. and Kumar, R. (1999). 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.

Sawhney, H. S. *et al.*. (1998). Videobrush: Experiences with consumer video mosaicing. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 56–62, IEEE Computer Society, Princeton.

Schaffalitzky, F. and Zisserman, A. (2002). Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?". In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 414–431, Springer-Verlag, Copenhagen.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, *47(1)*, 7–42.

Schmid, C., Mohr, R., and Bauckhage, C. (2000). Evaluation of interest point detectors. *International Journal of Computer Vision*, *37(2)*, 151–172.

Shakhnarovich, G., Viola, P., and Darrell, T. (2003). Fast pose estimation with parameter sensitive hashing. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 750–757, Nice, France.

Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, IEEE Computer Society, Seattle.

Shoemake, K. (1985). Animating rotation with quaternion curves. *Computer Graphics (SIGGRAPH'85)*, *19(3)*, 245–254.

Shum, H.-Y. and He, L.-W. (1999). Rendering with concentric mosaics. In *SIGGRAPH'99*, pages 299–306, ACM SIGGRAPH, Los Angeles.

Shum, H.-Y. and Szeliski, R. (1997). *Panoramic Image Mosaicing*. Technical Report MSR-TR-97-23, Microsoft Research.

Shum, H.-Y. and Szeliski, R. (1999). Stereo reconstruction from multiperspective panoramas. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 14–21, Kerkyra, Greece.

Shum, H.-Y. and Szeliski, R. (2000). Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision, 36(2)*, 101–130. Erratum published July 2002, 48(2):151-152.

Simoncelli, E. P., Adelson, E. H., and Heeger, D. J. (1991). Probability distributions of optic flow. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 310–315, IEEE Computer Society Press, Maui, Hawaii.

Slama, C. C., editor. (1980). *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Virginia, fourth edition.

Smelyanskiy, V. N., Cheeseman, P., Maluf, D. A., and Morris, R. D. (2000). Bayesian super-resolved surface reconstruction from images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 375–382, Hilton Head Island.

Stein, G. (1997). Lens distortion calibration using point correspondences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 602–608, San Juan, Puerto Rico.

Stewart, C. V. (1999). Robust parameter estimation in computer vision. *SIAM Reviews, 41(3)*, 513–537.

Szeliski, R. (1989). *Bayesian Modeling of Uncertainty in Low-Level Vision*. Kluwer Academic Publishers, Boston.

Szeliski, R. (1990). Fast surface interpolation using hierarchical basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(6)*, 513–528.

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

Szeliski, R. (1996). Video mosaics for virtual environments. *IEEE Computer Graphics and Applications, 16(2)*, 22–30.

Szeliski, R. and Coughlan, J. (1994). Hierarchical spline-based image registration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 194–201, IEEE Computer Society, Seattle.

Szeliski, R. and Coughlan, J. (1997). Spline-based image registration. *International Journal of Computer Vision, 22(3)*, 199–218.

Szeliski, R. and Kang, S. B. (1994). Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation, 5(1)*, 10–28.

Szeliski, R. and Kang, S. B. (1995). Direct methods for visual scene reconstruction. In *IEEE Workshop on Representations of Visual Scenes*, pages 26–33, Cambridge, Massachusetts.

Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and texture-mapped models. *Computer Graphics (SIGGRAPH'97 Proceedings)*, , 251–258.

Tappen, M. F. and Freeman, W. T. (2003). Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 900–907, Nice, France.

Teodosio, L. and Bender, W. (1993). Salient video stills: Content and context preserved. In *ACM Multimedia 93*, pages 39–46, Anaheim, California.

Tian, Q. and Huhns, M. N. (1986). Algorithms for subpixel registration. *Computer Vision, Graphics, and Image Processing, 35*, 220–233.

Triggs, B. (2004). Detecting keypoints with stable position, orientation, and scale under illumination changes. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 100–113, Springer-Verlag, Prague.

Triggs, B. *et al.*. (1999). Bundle adjustment — a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372, Springer, Kerkyra, Greece.

Triggs, B., Zisserman, A., and Szeliski, R., editors. (2000). *Vision Algorithms: Theory and Practice*, Springer, Berlin. Proceedings of the International Workshop on Vision Algorithms, Corfu, Greece, September 1999.

Tuytelaars, T. and Van Gool, L. (2004). Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision, 59(1)*, 61–85.

Uyttendaele, M. *et al.*. (2004). Image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications, 24(3)*.

Uyttendaele, M., Eden, A., and Szeliski, R. (2001). Eliminating ghosting and exposure artifacts in image mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 509–516, Kauai, Hawaii.

Wang, L., Kang, S. B., Szeliski, R., and Shum, H.-Y. (2001). Optimal texture map reconstruction from multiple views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 347–354, Kauai, Hawaii.

Watt, A. (1995). *3D Computer Graphics*. Addison-Wesley, third edition.

Weber, J. and Malik, J. (1995). Robust computation of optical flow in a multi-scale differential framework. *International Journal of Computer Vision, 14(1)*, 67–81.

Weiss, Y. (2001). Deriving intrinsic images from image sequences. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 7–14, Vancouver, Canada.

Williams, L. (1983). Pyramidal parametrics. *Computer Graphics, 17(3)*, 1–11.

Wood, D. N. *et al.*. (1997). Multiperspective panoramas for cel animation. In *Computer Graphics Proceedings, Annual Conference Series*, pages 243–250, ACM SIGGRAPH, Proc. SIGGRAPH'97 (Los Angeles).

Woodham, R. J. (1981). Analysing images of curved surfaces. *Artificial Intelligence, 17*, 117–140.

Xiong, Y. and Turkowski, K. (1997). Creating image-based VR using a self-calibrating fisheye lens. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 237–243, San Juan, Puerto Rico.

Xiong, Y. and Turkowski, K. (1998). Registration, calibration and blending in creating high quality panoramas. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 69–74, IEEE Computer Society, Princeton.

Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In *Third European Conference on Computer Vision (ECCV'94)*, pages 151–158, Springer-Verlag, Stockholm, Sweden.

Zoghlami, I., Faugeras, O., and Deriche, R. (1997). Using geometric corners to build a 2D mosaic from a set of images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 420–425, San Juan, Puerto Rico.