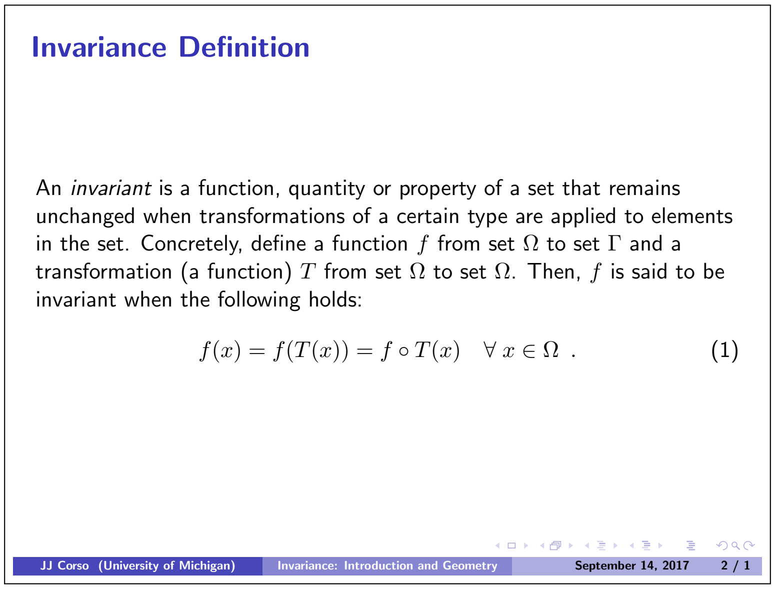
Invariance: Introduction and Geometry
Foundations of Computer Vision

Jason Corso
University of Michigan
September 14, 2017

1 Invariance Defined

Definition 1. An *invariant* is a function, quantity or property of a set that remains unchanged when transformations of a certain type are applied to elements in the set. Concretely, define a function f from set Ω to set Γ and a transformation (a function) T from set Ω to set Ω. Then, f is said to be invariant when the following holds:

$$f(x) = f(T(x)) = f \circ T(x) \quad \forall\, x \in \Omega \ . \tag{1}$$


Invariance Definition

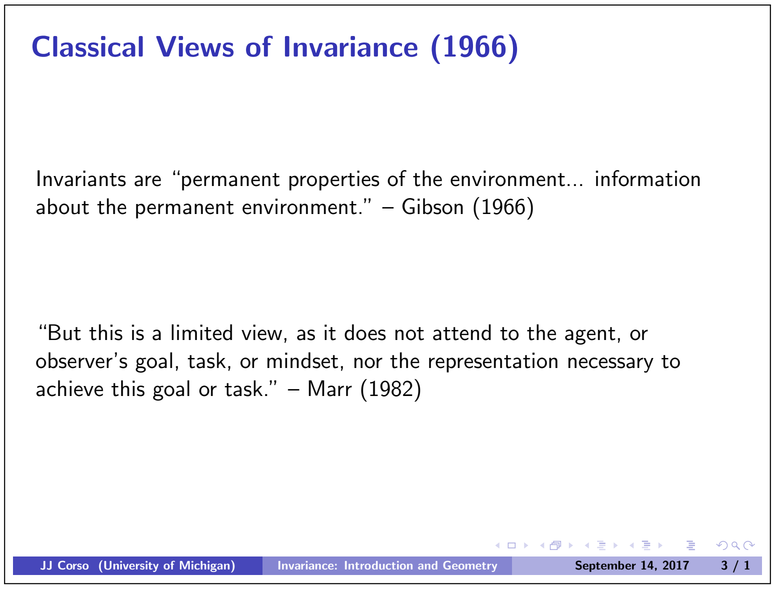
An *invariant* is a function, quantity or property of a set that remains unchanged when transformations of a certain type are applied to elements in the set. Concretely, define a function f from set Ω to set Γ and a transformation (a function) T from set Ω to set Ω. Then, f is said to be invariant when the following holds:

$$f(x) = f(T(x)) = f \circ T(x) \quad \forall\, x \in \Omega \ . \tag{1}$$

JJ Corso (University of Michigan) Invariance: Introduction and Geometry September 14, 2017 2 / 1

We can think of computer vision as a search for visual invariants. These range, indeed, from simple geometric invariants to complex category and viewpoint invariants.

Example 1 — When we want to detect a vehicle, we seek some representation that is invariant to all vehicles and how these vehicles get project down to the image place. However, if we need to discriminate between a sedan and a truck, or perhaps between an ordinary sedan and a police car, then we likely need a completely different representation. For automated driving scenarios, these representations need further to be invariant to glare, weather, lighting, shadows, etc.



Classical Views of Invariance (1966)

Invariants are “permanent properties of the environment... information about the permanent environment.” – Gibson (1966)

“But this is a limited view, as it does not attend to the agent, or observer’s goal, task, or mindset, nor the representation necessary to achieve this goal or task.” – Marr (1982)

JJ Corso (University of Michigan) Invariance: Introduction and Geometry September 14, 2017 3 / 1

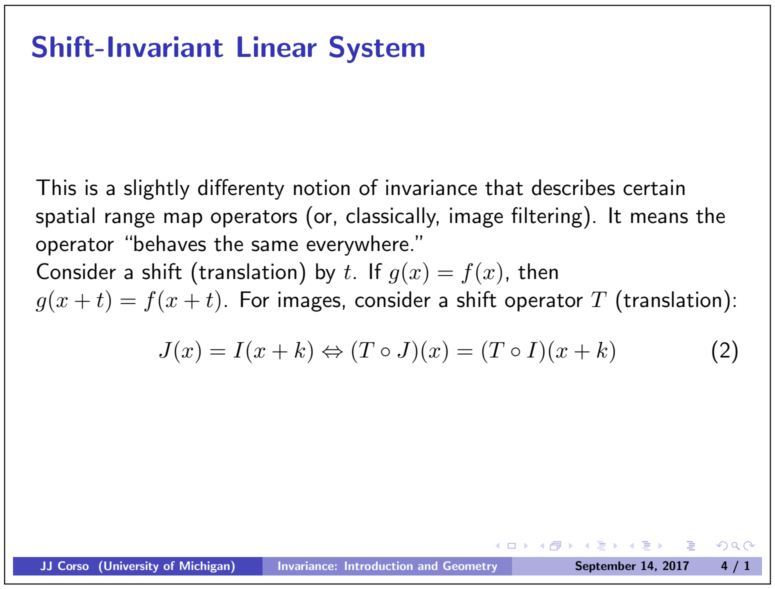
1.1 Types of Invariants and Invariance

We have seen the *geometric invariance* examples.

Example 2 — Shift-Invariant Linear System This is a slightly differenty notion of invariance that describes certain spatial range map operators (or, classically, image filtering). It means the operator “behaves the same everywhere.”

Consider a shift (translation) by t. If $g(x) = f(x)$, then $g(x+t) = f(x+t)$. For images, consider a shift operator T (translation):

$$J(x) = I(x+k) \Leftrightarrow (T \circ J)(x) = (T \circ I)(x+k) \tag{3}$$

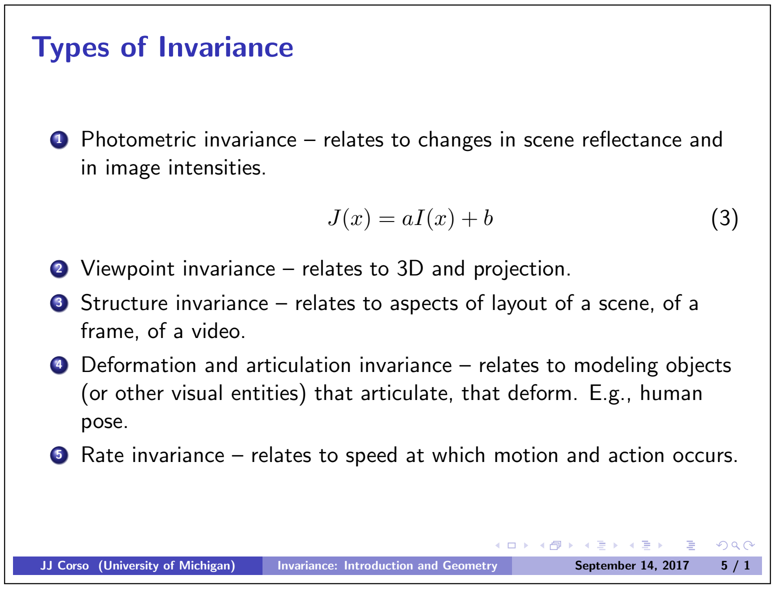


Shift-Invariant Linear System

This is a slightly differenty notion of invariance that describes certain spatial range map operators (or, classically, image filtering). It means the operator “behaves the same everywhere.”
Consider a shift (translation) by t. If $g(x) = f(x)$, then $g(x+t) = f(x+t)$. For images, consider a shift operator T (translation):

$$J(x) = I(x+k) \Leftrightarrow (T \circ J)(x) = (T \circ I)(x+k) \tag{2}$$

JJ Corso (University of Michigan) Invariance: Introduction and Geometry September 14, 2017 4 / 1

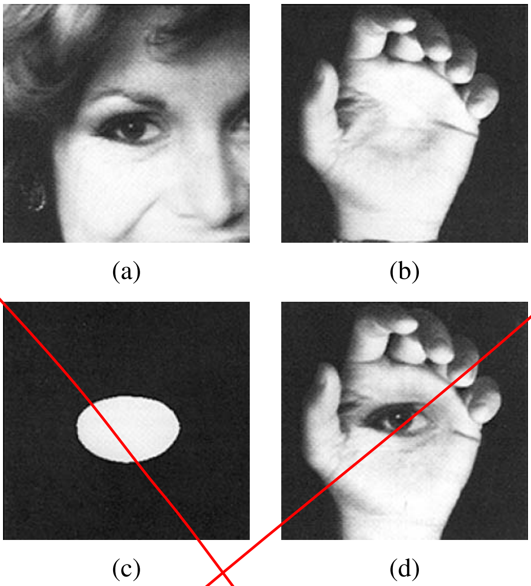


Types of Invariance

1. Photometric invariance – relates to changes in scene reflectance and in image intensities.
$$J(x) = aI(x) + b \tag{3}$$
2. Viewpoint invariance – relates to 3D and projection.
3. Structure invariance – relates to aspects of layout of a scene, of a frame, of a video.
4. Deformation and articulation invariance – relates to modeling objects (or other visual entities) that articulate, that deform. E.g., human pose.
5. Rate invariance – relates to speed at which motion and action occurs.

JJ Corso (University of Michigan) Invariance: Introduction and Geometry September 14, 2017 5 / 1

(Show the chart of invariances at the end of the lecture from the pptx slides.)



~~**Figure 3.43** Laplacian pyramid blend of two images of arbitrary shape (Burt and Adelson 1983b) © 1983 ACM: (a) first input image; (b) second input image; (c) region mask; (d) blended image.~~

~~multiplied by its corresponding Gaussian mask and the sum of these two weighted pyramids is then used to construct the final image (Figure 3.42, right column).~~

~~Figure 3.43 shows that this process can be applied to arbitrary mask images with surprising results. It is also straightforward to extend the pyramid blend to an arbitrary number of images whose pixel provenance is indicated by an integer-valued label image (see Exercise 3.20). This is particularly useful in image stitching and compositing applications, where the exposures may vary between different images, as described in Section 9.3.4.~~

$\downarrow$

3.6 Geometric transformations

In the previous sections, we saw how interpolation and decimation could be used to change the *resolution* of an image. In this section, we look at how to perform more general transformations, such as image rotations or general warps. In contrast to the point processes we saw in Section 3.1, where the function applied to an image transforms the *range* of the image,

$$g(\boldsymbol{x}) = h(f(\boldsymbol{x})), \tag{3.87}$$

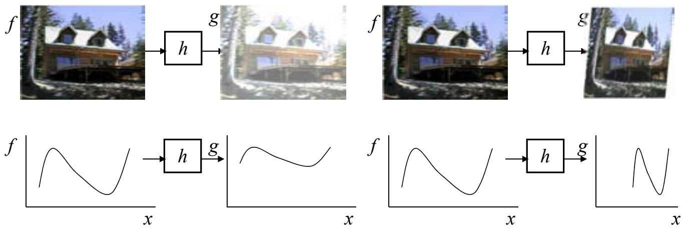


Figure 3.44 Image warping involves modifying the *domain* of an image function rather than its *range*.

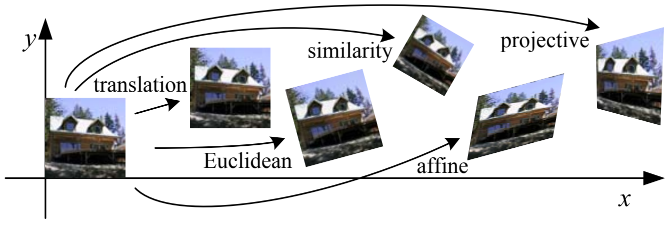


Figure 3.45 Basic set of 2D geometric image transformations.

here we look at functions that transform the *domain*,

$$g(\boldsymbol{x}) = f(\boldsymbol{h}(\boldsymbol{x})) \tag{3.88}$$

(see Figure 3.44).

We begin by studying the global *parametric* 2D transformation first introduced in Section 2.1.2. (Such a transformation is called parametric because it is controlled by a small number of parameters.) We then turn our attention to more local general deformations such as those defined on meshes (Section 3.6.2). Finally, we show how image warps can be combined with cross-dissolves to create interesting *morphs* (in-between animations) in Section 3.6.3. For readers interested in more details on these topics, there is an excellent survey by Heckbert (1986) as well as very accessible textbooks by Wolberg (1990), Gomes, Darsa, Costa *et al.* (1999) and Akenine-Möller and Haines (2002). Note that Heckbert's survey is on *texture mapping*, which is how the computer graphics community refers to the topic of warping images onto surfaces.

3.6.1 Parametric transformations

Parametric transformations apply a global deformation to an image, where the behavior of the transformation is controlled by a small number of parameters. Figure 3.45 shows a few ex-

Transformation	**Matrix**	**# DoF**	**Preserves**	**Icon**
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	□
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	◇
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	◇
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	▱
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	⏢

Table 3.5 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. 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.

amples of such transformations, which are based on the 2D geometric transformations shown in Figure 2.4. The formulas for these transformations were originally given in Table 2.1 and are reproduced here in Table 3.5 for ease of reference.

In general, given a transformation specified by a formula $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ and a source image $f(\boldsymbol{x})$, how do we compute the values of the pixels in the new image $g(\boldsymbol{x})$, as given in (3.88)? Think about this for a minute before proceeding and see if you can figure it out.

If you are like most people, you will come up with an algorithm that looks something like Algorithm 3.1. This process is called *forward warping* or *forward mapping* and is shown in Figure 3.46a. Can you think of any problems with this approach?

```
procedure forwardWarp($f, \boldsymbol{h}$, out $g$):

    For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

        1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
        2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.
```

Algorithm 3.1 Forward warping algorithm for transforming an image $f(\boldsymbol{x})$ into an image $g(\boldsymbol{x}')$ through the parametric transform $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.

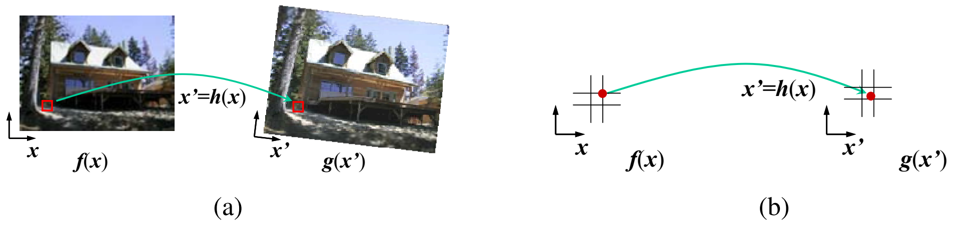


Figure 3.46 Forward warping algorithm: (a) a pixel $f(\boldsymbol{x})$ is copied to its corresponding location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ in image $g(\boldsymbol{x}')$; (b) detail of the source and destination pixel locations.

In fact, this approach suffers from several limitations. The process of copying a pixel $f(\boldsymbol{x})$ to a location $\boldsymbol{x}'$ in g is not well defined when $\boldsymbol{x}'$ has a non-integer value. What do we do in such a case? What would you do?

You can round the value of $\boldsymbol{x}'$ to the nearest integer coordinate and copy the pixel there, but the resulting image has severe aliasing and pixels that jump around a lot when animating the transformation. You can also "distribute" the value among its four nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end. This technique is called *splatting* and is sometimes used for volume rendering in the graphics community (Levoy and Whitted 1985; Levoy 1988; Westover 1989; Rusinkiewicz and Levoy 2000). Unfortunately, it suffers from both moderate amounts of aliasing and a fair amount of blur (loss of high-resolution detail).

The second major problem with forward warping is the appearance of cracks and holes, especially when magnifying an image. Filling such holes with their nearby neighbors can lead to further aliasing and blurring.

What can we do instead? A preferable solution is to use *inverse warping* (Algorithm 3.2), where each pixel in the destination image $g(\boldsymbol{x}')$ is sampled from the original image $f(\boldsymbol{x})$ (Figure 3.47).

How does this differ from the forward warping algorithm? For one thing, since $\hat{\boldsymbol{h}}(\boldsymbol{x}')$ is (presumably) defined for all pixels in $g(\boldsymbol{x}')$, we no longer have holes. More importantly, resampling an image at non-integer locations is a well-studied problem (general image interpolation, see Section 3.5.2) and high-quality filters that control aliasing can be used.

Where does the function $\hat{\boldsymbol{h}}(\boldsymbol{x}')$ come from? Quite often, it can simply be computed as the inverse of $\boldsymbol{h}(\boldsymbol{x})$. In fact, all of the parametric transforms listed in Table 3.5 have closed form solutions for the inverse transform: simply take the inverse of the 3×3 matrix specifying the transform.

In other cases, it is preferable to formulate the problem of image warping as that of resampling a source image $f(\boldsymbol{x})$ given a mapping $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$ from destination pixels $\boldsymbol{x}'$ to source pixels $\boldsymbol{x}$. For example, in optical flow (Section 8.4), we estimate the flow field as the

```
procedure inverseWarp($f, \boldsymbol{h}$, out $g$):

    For every pixel $\boldsymbol{x}'$ in $g(\boldsymbol{x}')$

        1. Compute the source location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$

        2. Resample $f(\boldsymbol{x})$ at location $\boldsymbol{x}$ and copy to $g(\boldsymbol{x}')$
```

Algorithm 3.2 Inverse warping algorithm for creating an image $g(\boldsymbol{x}')$ from an image $f(\boldsymbol{x})$ using the parametric transform $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.

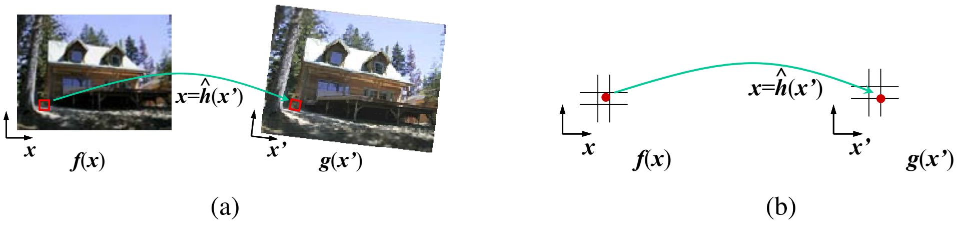


Figure 3.47 Inverse warping algorithm: (a) a pixel $g(\boldsymbol{x}')$ is sampled from its corresponding location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$ in image $f(\boldsymbol{x})$; (b) detail of the source and destination pixel locations.

location of the *source* pixel which produced the current pixel whose flow is being estimated, as opposed to computing the *destination* pixel to which it is going. Similarly, when correcting for radial distortion (Section 2.1.6), we calibrate the lens by computing for each pixel in the final (undistorted) image the corresponding pixel location in the original (distorted) image.

What kinds of interpolation filter are suitable for the resampling process? Any of the filters we studied in Section 3.5.2 can be used, including nearest neighbor, bilinear, bicubic, and windowed sinc functions. While bilinear is often used for speed (e.g., inside the inner loop of a patch-tracking algorithm, see Section 8.1.3), bicubic, and windowed sinc are preferable where visual quality is important.

To compute the value of $f(\boldsymbol{x})$ at a non-integer location $\boldsymbol{x}$, we simply apply our usual FIR resampling filter,

$$g(x, y) = \sum_{k,l} f(k, l)h(x - k, y - l), \tag{3.89}$$

where (x, y) are the sub-pixel coordinate values and $h(x, y)$ is some interpolating or smoothing kernel. Recall from Section 3.5.2 that when decimation is being performed, the smoothing kernel is stretched and re-scaled according to the downsampling rate r.

Unfortunately, for a general (non-zoom) image transformation, the resampling rate r is not well defined. Consider a transformation that stretches the x dimensions while squashing

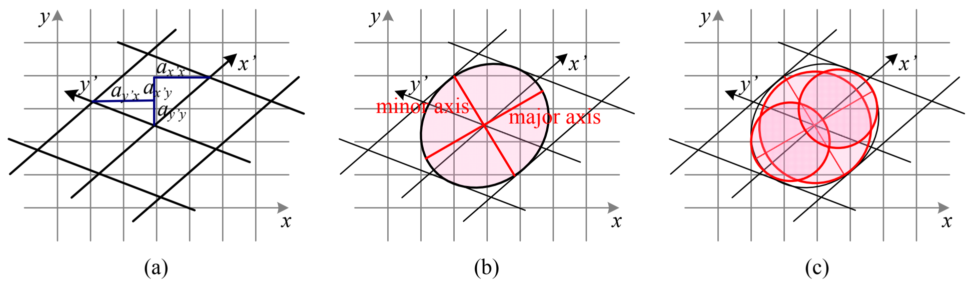


Figure 3.48 Anisotropic texture filtering: (a) Jacobian of transform $\boldsymbol{A}$ and the induced horizontal and vertical resampling rates $\{a_{x'x}, a_{x'y}, a_{y'x}, a_{y'y}\}$; (b) elliptical footprint of an EWA smoothing kernel; (c) anisotropic filtering using multiple samples along the major axis. Image pixels lie at line intersections.

the y dimensions. The resampling kernel should be performing regular interpolation along the x dimension and smoothing (to anti-alias the blurred image) in the y direction. This gets even more complicated for the case of general affine or perspective transforms.

What can we do? Fortunately, Fourier analysis can help. The two-dimensional generalization of the one-dimensional *domain scaling* law given in Table 3.1 is

$$g(\boldsymbol{A}\boldsymbol{x}) \Leftrightarrow |\boldsymbol{A}|^{-1} G(\boldsymbol{A}^{-T}\boldsymbol{f}). \tag{3.90}$$

For all of the transforms in Table 3.5 except perspective, the matrix $\boldsymbol{A}$ is already defined. For perspective transformations, the matrix $\boldsymbol{A}$ is the linearized *derivative* of the perspective transformation (Figure 3.48a), i.e., the local affine approximation to the stretching induced by the projection (Heckbert 1986; Wolberg 1990; Gomes, Darsa, Costa *et al.* 1999; Akenine-Möller and Haines 2002).

To prevent aliasing, we need to pre-filter the image $f(\boldsymbol{x})$ with a filter whose frequency response is the projection of the final desired spectrum through the $\boldsymbol{A}^{-T}$ transform (Szeliski, Winder, and Uyttendaele 2010). In general (for non-zoom transforms), this filter is non-separable and hence is very slow to compute. Therefore, a number of approximations to this filter are used in practice, include MIP-mapping, elliptically weighted Gaussian averaging, and anisotropic filtering (Akenine-Möller and Haines 2002).

MIP-mapping

MIP-mapping was first proposed by Williams (1983) as a means to rapidly pre-filter images being used for *texture mapping* in computer graphics. A MIP-map[18] is a standard image

[18] The term 'MIP' stands for *multi in parvo*, meaning 'many in one'.

pyramid (Figure 3.32), where each level is pre-filtered with a high-quality filter rather than a poorer quality approximation, such as Burt and Adelson's (1983b) five-tap binomial. To resample an image from a MIP-map, a scalar estimate of the resampling rate r is first computed. For example, r can be the maximum of the absolute values in $\boldsymbol{A}$ (which suppresses aliasing) or it can be the minimum (which reduces blurring). Akenine-Möller and Haines (2002) discuss these issues in more detail.

Once a resampling rate has been specified, a *fractional* pyramid level is computed using the base 2 logarithm,

$$l = \log_2 r. \tag{3.91}$$

One simple solution is to resample the texture from the next higher or lower pyramid level, depending on whether it is preferable to reduce aliasing or blur. A better solution is to resample *both* images and blend them linearly using the fractional component of l. Since most MIP-map implementations use bilinear resampling within each level, this approach is usually called *trilinear MIP-mapping*. Computer graphics rendering APIs, such as OpenGL and Direct3D, have parameters that can be used to select which variant of MIP-mapping (and of the sampling rate r computation) should be used, depending on the desired tradeoff between speed and quality. Exercise 3.22 has you examine some of these tradeoffs in more detail.

Elliptical Weighted Average

The Elliptical Weighted Average (EWA) filter invented by Greene and Heckbert (1986) is based on the observation that the affine mapping $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}'$ defines a skewed two-dimensional coordinate system in the vicinity of each source pixel $\boldsymbol{x}$ (Figure 3.48a). For every destination pixel $\boldsymbol{x}'$, the ellipsoidal projection of a small pixel grid in $\boldsymbol{x}'$ onto $\boldsymbol{x}$ is computed (Figure 3.48b). This is then used to filter the source image $g(\boldsymbol{x})$ with a Gaussian whose inverse covariance matrix is this ellipsoid.

Despite its reputation as a high-quality filter (Akenine-Möller and Haines 2002), we have found in our work (Szeliski, Winder, and Uyttendaele 2010) that because a Gaussian kernel is used, the technique suffers simultaneously from both blurring and aliasing, compared to higher-quality filters. The EWA is also quite slow, although faster variants based on MIP-mapping have been proposed (Szeliski, Winder, and Uyttendaele (2010) provide some additional references).

Anisotropic filtering

An alternative approach to filtering oriented textures, which is sometimes implemented in graphics hardware (GPUs), is to use anisotropic filtering (Barkans 1997; Akenine-Möller and Haines 2002). In this approach, several samples at different resolutions (fractional levels in the MIP-map) are combined along the major axis of the EWA Gaussian (Figure 3.48c).

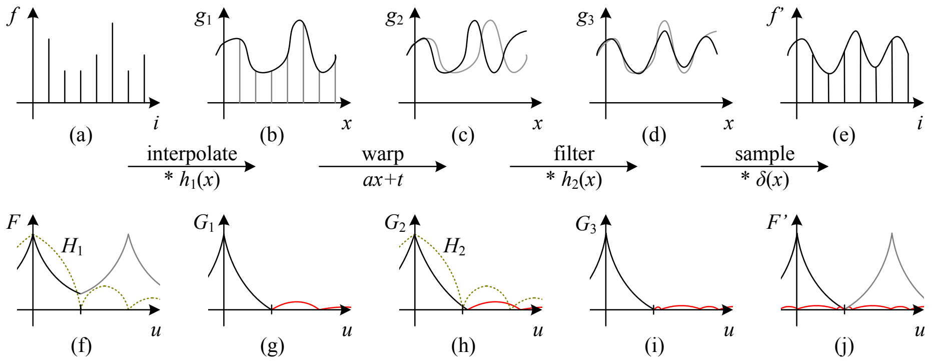


Figure 3.49 One-dimensional signal resampling (Szeliski, Winder, and Uyttendaele 2010): (a) original sampled signal $f(i)$; (b) interpolated signal $g_1(x)$; (c) warped signal $g_2(x)$; (d) filtered signal $g_3(x)$; (e) sampled signal $f'(i)$. The corresponding spectra are shown below the signals, with the aliased portions shown in red.

Multi-pass transforms

The optimal approach to warping images without excessive blurring or aliasing is to adaptively pre-filter the source image at each pixel using an ideal low-pass filter, i.e., an oriented skewed sinc or low-order (e.g., cubic) approximation (Figure 3.48a). Figure 3.49 shows how this works in one dimension. The signal is first (theoretically) interpolated to a continuous waveform, (ideally) low-pass filtered to below the new Nyquist rate, and then re-sampled to the final desired resolution. In practice, the interpolation and decimation steps are concatenated into a single *polyphase* digital filtering operation (Szeliski, Winder, and Uyttendaele 2010).

For parametric transforms, the oriented two-dimensional filtering and resampling operations can be approximated using a series of one-dimensional resampling and shearing transforms (Catmull and Smith 1980; Heckbert 1989; Wolberg 1990; Gomes, Darsa, Costa *et al.* 1999; Szeliski, Winder, and Uyttendaele 2010). The advantage of using a series of one-dimensional transforms is that they are much more efficient (in terms of basic arithmetic operations) than large, non-separable, two-dimensional filter kernels.

In order to prevent aliasing, however, it may be necessary to upsample in the opposite direction before applying a shearing transformation (Szeliski, Winder, and Uyttendaele 2010). Figure 3.50 shows this process for a rotation, where a vertical upsampling stage is added before the horizontal shearing (and upsampling) stage. The upper image shows the appearance of the letter being rotated, while the lower image shows its corresponding Fourier transform.

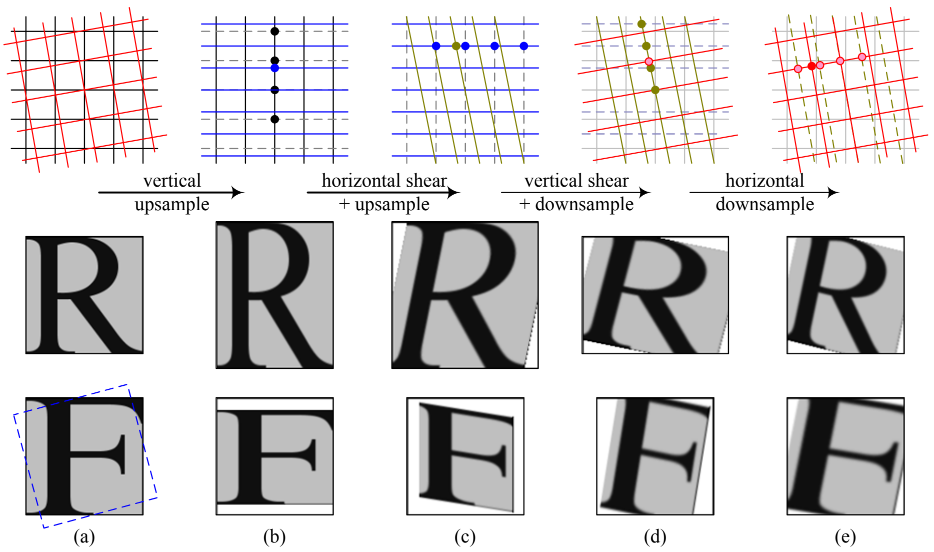


Figure 3.50 Four-pass rotation (Szeliski, Winder, and Uyttendaele 2010): (a) original pixel grid, image, and its Fourier transform; (b) vertical upsampling; (c) horizontal shear and upsampling; (d) vertical shear and downsampling; (e) horizontal downsampling. The general affine case looks similar except that the first two stages perform general resampling.

3.6.2 Mesh-based warping

While parametric transforms specified by a small number of global parameters have many uses, *local* deformations with more degrees of freedom are often required.

Consider, for example, changing the appearance of a face from a frown to a smile (Figure 3.51a). What is needed in this case is to curve the corners of the mouth upwards while leaving the rest of the face intact.[19] To perform such a transformation, different amounts of motion are required in different parts of the image. Figure 3.51 shows some of the commonly used approaches.

The first approach, shown in Figure 3.51a–b, is to specify a *sparse* set of corresponding points. The displacement of these points can then be interpolated to a dense *displacement field* (Chapter 8) using a variety of techniques (Nielson 1993). One possibility is to *triangulate* the set of points in one image (de Berg, Cheong, van Kreveld *et al.* 2006; Litwinowicz and Williams 1994; Buck, Finkelstein, Jacobs *et al.* 2000) and to use an *affine* motion model (Table 3.5), specified by the three triangle vertices, inside each triangle. If the destination

[19] Rowland and Perrett (1995); Pighin, Hecker, Lischinski *et al.* (1998); Blanz and Vetter (1999); Leyvand, Cohen-Or, Dror *et al.* (2008) show more sophisticated examples of changing facial expression and appearance.

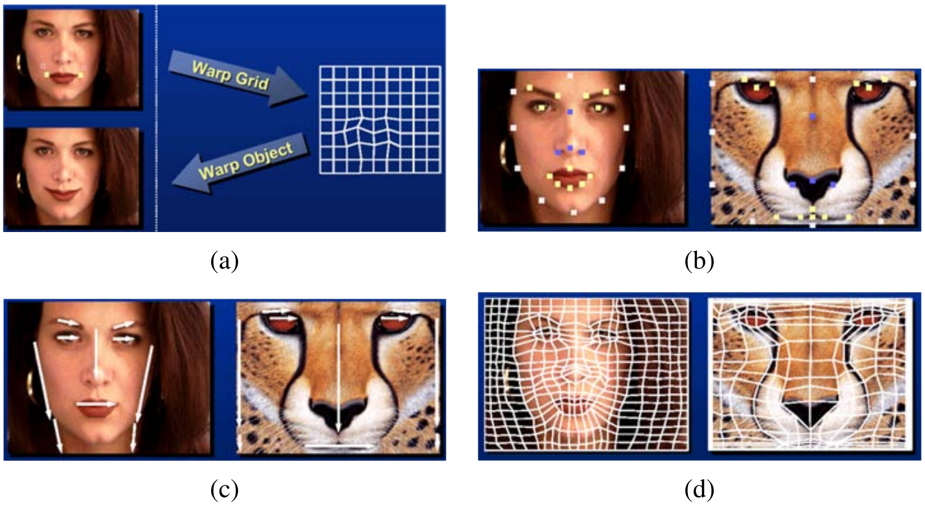


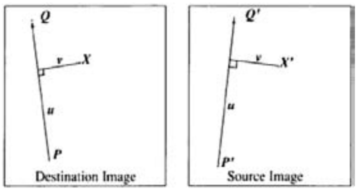
Figure 3.51 Image warping alternatives (Gomes, Darsa, Costa *et al.* 1999) © 1999 Morgan Kaufmann: (a) sparse control points $\longrightarrow$ deformation grid; (b) denser set of control point correspondences; (c) oriented line correspondences; (d) uniform quadrilateral grid.

image is triangulated according to the new vertex locations, an inverse warping algorithm (Figure 3.47) can be used. If the source image is triangulated and used as a *texture map*, computer graphics rendering algorithms can be used to draw the new image (but care must be taken along triangle edges to avoid potential aliasing).

Alternative methods for interpolating a sparse set of displacements include moving nearby quadrilateral mesh vertices, as shown in Figure 3.51a, using *variational* (energy minimizing) interpolants such as regularization (Litwinowicz and Williams 1994), see Section 3.7.1, or using locally weighted (*radial basis function*) combinations of displacements (Nielson 1993). (See (Section 12.3.1) for additional *scattered data interpolation* techniques.) If quadrilateral meshes are used, it may be desirable to interpolate displacements down to individual pixel values using a smooth interpolant such as a quadratic B-spline (Farin 1996; Lee, Wolberg, Chwa *et al.* 1996).[20]

In some cases, e.g., if a dense depth map has been estimated for an image (Shade, Gortler, He *et al.* 1998), we only know the forward displacement for each pixel. As mentioned before, drawing source pixels at their destination location, i.e., forward warping (Figure 3.46), suffers from several potential problems, including aliasing and the appearance of small cracks. An alternative technique in this case is to forward warp the *displacement field* (or depth map) to

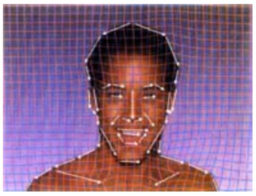
[20] Note that the *block-based* motion models used by many video compression standards (Le Gall 1991) can be thought of as a 0th-order (piecewise-constant) displacement field.



(a)

```
For each pixel X in the destination
    DSUM = (0,0)
    weightsum = 0
    For each line P_i Q_i
        calculate u,v based on P_i Q_i
        calculate X'_i based on u,v and P_i'Q_i'
        calculate displacement D_i = X_i' - X_i for this line
        dist = shortest distance from X to P_i Q_i
        weight = (length^p / (a + dist))^b
        DSUM += D_i * weight
        weightsum += weight
    X' = X + DSUM / weightsum
    destinationImage(X) = sourceImage(X')
```

(b)



(c)

Figure 3.52 Line-based image warping (Beier and Neely 1992) © 1992 ACM: (a) distance computation and position transfer; (b) rendering algorithm; (c) two intermediate warps used for morphing.

its new location, fill small holes in the resulting map, and then use inverse warping to perform the resampling (Shade, Gortler, He *et al.* 1998). The reason that this generally works better than forward warping is that displacement fields tend to be much smoother than images, so the aliasing introduced during the forward warping of the displacement field is much less noticeable.

A second approach to specifying displacements for local deformations is to use corresponding *oriented line segments* (Beier and Neely 1992), as shown in Figures 3.51c and 3.52. Pixels along each line segment are transferred from source to destination exactly as specified, and other pixels are warped using a smooth interpolation of these displacements. Each line segment correspondence specifies a translation, rotation, and scaling, i.e., a *similarity transform* (Table 3.5), for pixels in its vicinity, as shown in Figure 3.52a. Line segments influence the overall displacement of the image using a weighting function that depends on the minimum distance to the line segment (v in Figure 3.52a if $u \in [0, 1]$, else the shorter of the two distances to P and Q).

For each pixel X, the target location X' for each line correspondence is computed along with a weight that depends on the distance and the line segment length (Figure 3.52b). The weighted average of all target locations X_i' then becomes the final destination location. Note that while Beier and Neely describe this algorithm as a forward warp, an equivalent algorithm can be written by sequencing through the destination pixels. The resulting warps are not identical because line lengths or distances to lines may be different. Exercise 3.23 has you implement the Beier–Neely (line-based) warp and compare it to a number of other local deformation methods.

Yet another way of specifying correspondences in order to create image warps is to use snakes (Section 5.1.1) combined with B-splines (Lee, Wolberg, Chwa *et al.* 1996). This technique is used in Apple's Shake software and is popular in the medical imaging community.

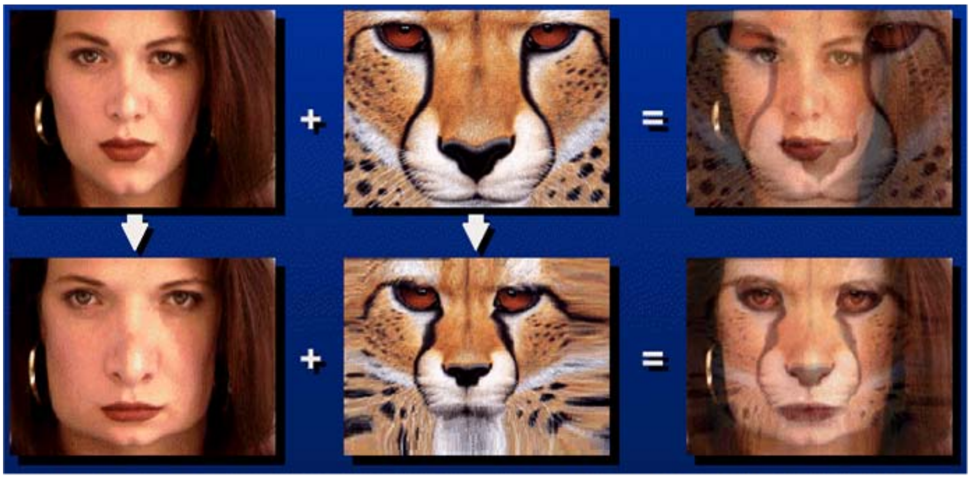


Figure 3.53 Image morphing (Gomes, Darsa, Costa *et al.* 1999) © 1999 Morgan Kaufmann. Top row: if the two images are just blended, visible ghosting results. Bottom row: both images are first warped to the same intermediate location (e.g., halfway towards the other image) and the resulting warped images are then blended resulting in a seamless morph.

One final possibility for specifying displacement fields is to use a mesh specifically *adapted* to the underlying image content, as shown in Figure 3.51d. Specifying such meshes by hand can involve a fair amount of work; Gomes, Darsa, Costa *et al.* (1999) describe an interactive system for doing this. Once the two meshes have been specified, intermediate warps can be generated using linear interpolation and the displacements at mesh nodes can be interpolated using splines.

~~3.6.3 *Application*: Feature-based morphing~~

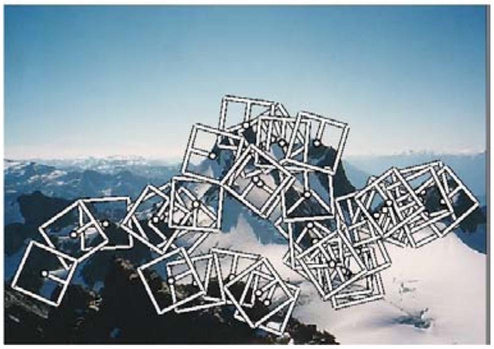
~~While warps can be used to change the appearance of or to animate a *single* image, even more powerful effects can be obtained by warping and blending two or more images using a process now commonly known as *morphing* (Beier and Neely 1992; Lee, Wolberg, Chwa *et al.* 1996; Gomes, Darsa, Costa *et al.* 1999).~~

~~Figure 3.53 shows the essence of image morphing. Instead of simply cross-dissolving between two images, which leads to ghosting as shown in the top row, each image is warped toward the other image before blending, as shown in the bottom row. If the correspondences have been set up well (using any of the techniques shown in Figure 3.51), corresponding features are aligned and no ghosting results.~~

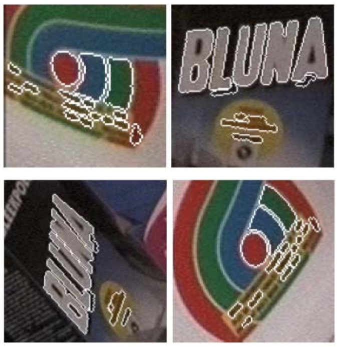
~~The above process is repeated for each intermediate frame being generated during a~~

Chapter 4

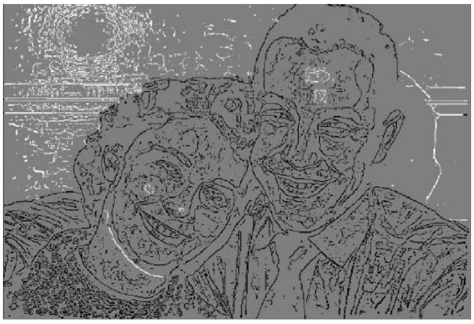
Feature detection and matching



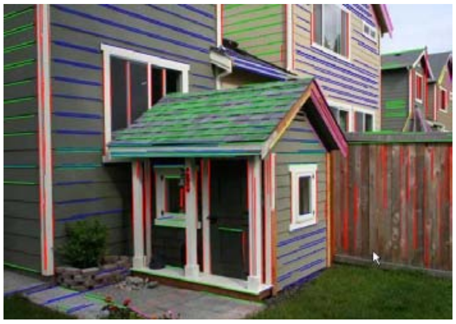
(a)



(b)



(c)



(d)

Figure 4.1 A variety of feature detectors and descriptors can be used to analyze, describe and match images: (a) point-like interest operators (Brown, Szeliski, and Winder 2005) © 2005 IEEE; (b) region-like interest operators (Matas, Chum, Urban *et al.* 2004) © 2004 Elsevier; (c) edges (Elder and Goldberg 2001) © 2001 IEEE; (d) straight lines (Sinha, Steedly, Szeliski *et al.* 2008) © 2008 ACM.

Feature detection and matching are an essential component of many computer vision applications. Consider the two pairs of images shown in Figure 4.2. For the first pair, we may wish to *align* the two images so that they can be seamlessly stitched into a composite mosaic (Chapter 9). For the second pair, we may wish to establish a dense set of *correspondences* so that a 3D model can be constructed or an in-between view can be generated (Chapter 11). In either case, what kinds of *features* should you detect and then match in order to establish such an alignment or set of correspondences? Think about this for a few moments before reading on.

The first kind of feature that you may notice are specific locations in the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized feature are often called *keypoint features* or *interest points* (or even *corners*) and are often described by the appearance of patches of pixels surrounding the point location (Section 4.1). Another class of important features are *edges*, e.g., the profile of mountains against the sky, (Section 4.2). These kinds of features can be matched based on their orientation and local appearance (edge profiles) and can also be good indicators of object boundaries and *occlusion* events in image sequences. Edges can be grouped into longer *curves* and *straight line segments*, which can be directly matched or analyzed to find *vanishing points* and hence internal and external camera parameters (Section 4.3).

In this chapter, we describe some practical approaches to detecting such features and also discuss how feature correspondences can be established across different images. Point features are now used in such a wide variety of applications that it is good practice to read and implement some of the algorithms from (Section 4.1). Edges and lines provide information that is complementary to both keypoint and region-based descriptors and are well-suited to describing object boundaries and man-made objects. These alternative descriptors, while extremely useful, can be skipped in a short introductory course.

4.1 Points and patches

Point features can be used to find a sparse set of corresponding locations in different images, often as a pre-cursor to computing camera pose (Chapter 7), which is a prerequisite for computing a denser set of correspondences using stereo matching (Chapter 11). Such correspondences can also be used to align different images, e.g., when stitching image mosaics or performing video stabilization (Chapter 9). They are also used extensively to perform object instance and category recognition (Sections 14.3 and 14.4). A key advantage of keypoints is that they permit matching even in the presence of clutter (occlusion) and large scale and orientation changes.

Feature-based correspondence techniques have been used since the early days of stereo

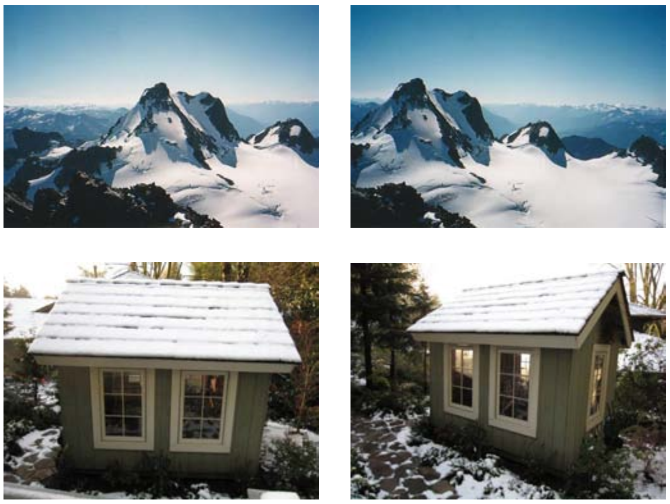


Figure 4.2 Two pairs of images to be matched. What kinds of feature might one use to establish a set of *correspondences* between these images?

matching (Hannah 1974; Moravec 1983; Hannah 1988) and have more recently gained popularity for image-stitching applications (Zoghlami, Faugeras, and Deriche 1997; Brown and Lowe 2007) as well as fully automated 3D modeling (Beardsley, Torr, and Zisserman 1996; Schaffalitzky and Zisserman 2002; Brown and Lowe 2003; Snavely, Seitz, and Szeliski 2006).

There are two main approaches to finding feature points and their correspondences. The first is to find features in one image that can be accurately *tracked* using a local search technique, such as correlation or least squares (Section 4.1.4). The second is to independently detect features in all the images under consideration and then *match* features based on their local appearance (Section 4.1.3). The former approach is more suitable when images are taken from nearby viewpoints or in rapid succession (e.g., video sequences), while the latter is more suitable when a large amount of motion or appearance change is expected, e.g., in stitching together panoramas (Brown and Lowe 2007), establishing correspondences in *wide baseline stereo* (Schaffalitzky and Zisserman 2002), or performing object recognition (Fergus, Perona, and Zisserman 2007).

In this section, we split the keypoint detection and matching pipeline into four separate stages. During the *feature detection* (extraction) stage (Section 4.1.1), each image is searched for locations that are likely to match well in other images. At the *feature description* stage (Section 4.1.2), each region around detected keypoint locations is converted into a more compact and stable (invariant) *descriptor* that can be matched against other descriptors. The

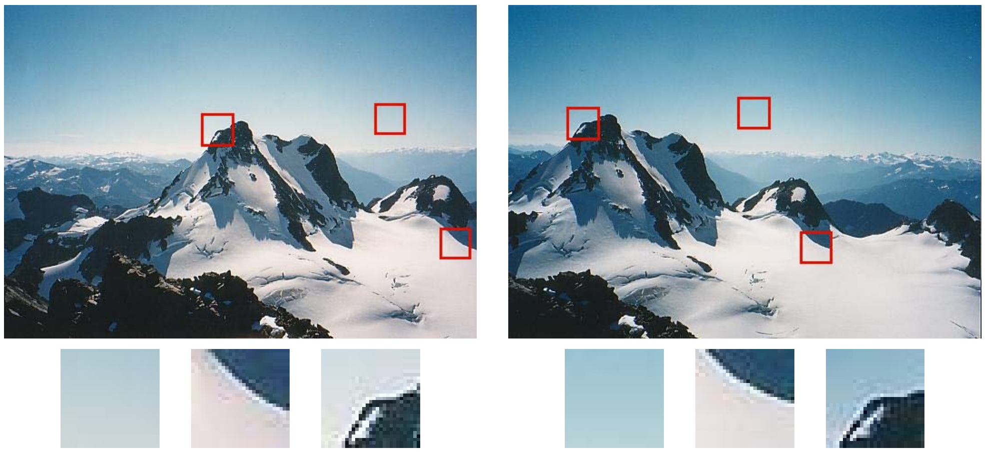


Figure 4.3 Image pairs with extracted patches below. Notice how some patches can be localized or matched with higher accuracy than others.

feature matching stage (Section 4.1.3) efficiently searches for likely matching candidates in other images. The *feature tracking* stage (Section 4.1.4) is an alternative to the third stage that only searches a small neighborhood around each detected feature and is therefore more suitable for video processing.

A wonderful example of all of these stages can be found in David Lowe's (2004) paper, which describes the development and refinement of his *Scale Invariant Feature Transform* (SIFT). Comprehensive descriptions of alternative techniques can be found in a series of survey and evaluation papers covering both feature detection (Schmid, Mohr, and Bauckhage 2000; Mikolajczyk, Tuytelaars, Schmid *et al.* 2005; Tuytelaars and Mikolajczyk 2007) and feature descriptors (Mikolajczyk and Schmid 2005). Shi and Tomasi (1994) and Triggs (2004) also provide nice reviews of feature detection techniques.

4.1.1 Feature detectors

How can we find image locations where we can reliably find correspondences with other images, i.e., what are good features to track (Shi and Tomasi 1994; Triggs 2004)? Look again at the image pair shown in Figure 4.3 and at the three sample *patches* to see how well they might be matched or tracked. As you may notice, textureless patches are nearly impossible to localize. Patches with large contrast changes (gradients) are easier to localize, although straight line segments at a single orientation suffer from the *aperture problem* (Horn and Schunck 1981; Lucas and Kanade 1981; Anandan 1989), i.e., it is only possible to align the patches along the direction *normal* to the edge direction (Figure 4.4b). Patches with

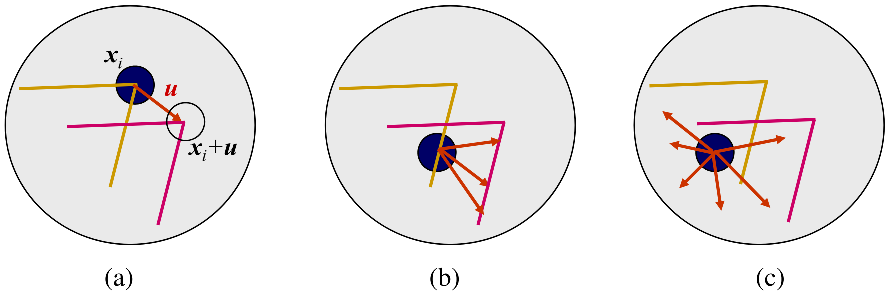


Figure 4.4 Aperture problems for different image patches: (a) stable ("corner-like") flow; (b) classic aperture problem (barber-pole illusion); (c) textureless region. The two images I_0 (yellow) and I_1 (red) are overlaid. The red vector $\boldsymbol{u}$ indicates the displacement between the patch centers and the $w(\boldsymbol{x}_i)$ weighting function (patch window) is shown as a dark circle.

gradients in at least two (significantly) different orientations are the easiest to localize, as shown schematically in Figure 4.4a.

These intuitions can be formalized by looking at the simplest possible matching criterion for comparing two image patches, i.e., their (weighted) summed square difference,

$$E_{\text{WSSD}}(\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{4.1}$$

where I_0 and I_1 are the two images being compared, $\boldsymbol{u} = (u, v)$ is the *displacement* vector, $w(\boldsymbol{x})$ is a spatially varying weighting (or window) function, and the summation i is over all the pixels in the patch. Note that this is the same formulation we later use to estimate motion between complete images (Section 8.1).

When performing feature detection, we do not know which other image locations the feature will end up being matched against. Therefore, we can only compute how stable this metric is with respect to small variations in position $\Delta\boldsymbol{u}$ by comparing an image patch against itself, which is known as an *auto-correlation function* or *surface*

$$E_{\text{AC}}(\Delta\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \tag{4.2}$$

(Figure 4.5).[1] Note how the auto-correlation surface for the textured flower bed (Figure 4.5b and the red cross in the lower right quadrant of Figure 4.5a) exhibits a strong minimum, indicating that it can be well localized. The correlation surface corresponding to the roof edge (Figure 4.5c) has a strong ambiguity along one direction, while the correlation surface corresponding to the cloud region (Figure 4.5d) has no stable minimum.

[1] Strictly speaking, a correlation is the *product* of two patches (3.12); I'm using the term here in a more qualitative sense. The weighted sum of squared differences is often called an *SSD surface* (Section 8.1).

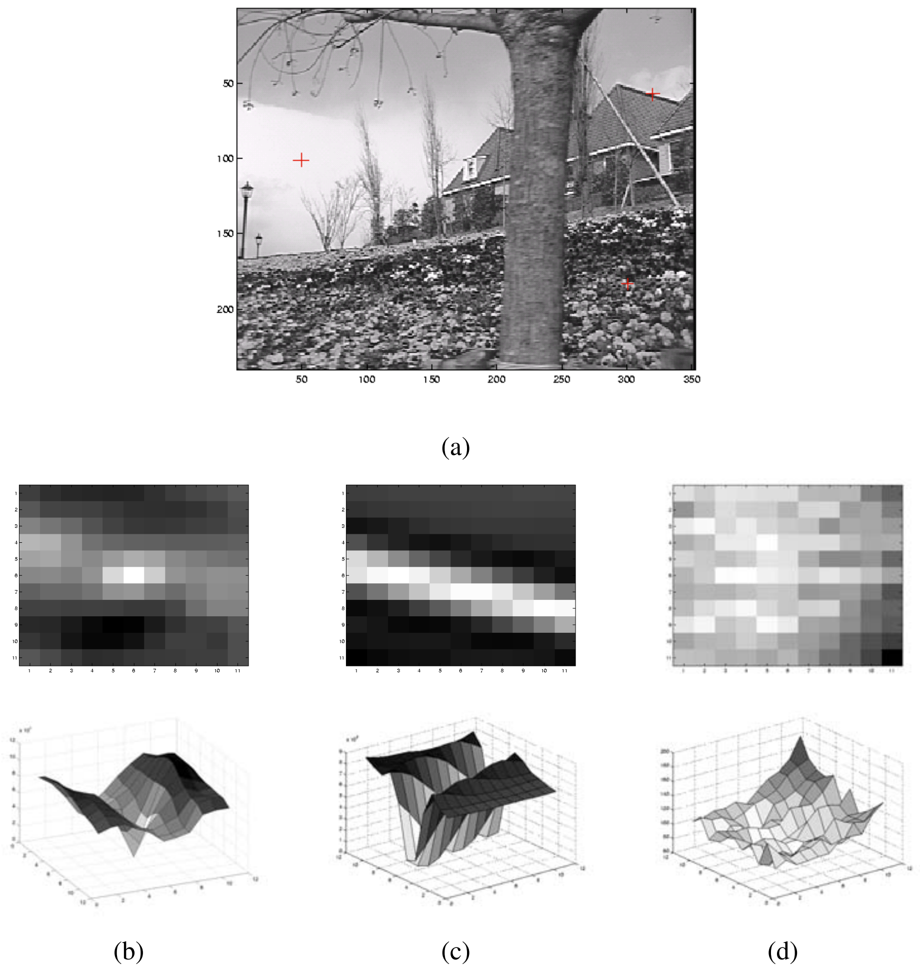


Figure 4.5 Three auto-correlation surfaces $E_{\mathrm{AC}}(\Delta \boldsymbol{u})$ shown as both grayscale images and surface plots: (a) The original image is marked with three red crosses to denote where the auto-correlation surfaces were computed; (b) this patch is from the flower bed (good unique minimum); (c) this patch is from the roof edge (one-dimensional aperture problem); and (d) this patch is from the cloud (no good peak). Each grid point in figures b–d is one value of $\Delta \boldsymbol{u}$.

Using a Taylor Series expansion of the image function $I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) \approx I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i) \cdot \Delta\boldsymbol{u}$ (Lucas and Kanade 1981; Shi and Tomasi 1994), we can approximate the auto-correlation surface as

$$\begin{aligned} E_{\mathrm{AC}}(\Delta\boldsymbol{u}) &= \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 && (4.3)\\ &\approx \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i) \cdot \Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 && (4.4)\\ &= \sum_i w(\boldsymbol{x}_i)[\nabla I_0(\boldsymbol{x}_i) \cdot \Delta\boldsymbol{u}]^2 && (4.5)\\ &= \Delta\boldsymbol{u}^T \boldsymbol{A} \Delta\boldsymbol{u}, && (4.6) \end{aligned}$$

where

$$\nabla I_0(\boldsymbol{x}_i) = (\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y})(\boldsymbol{x}_i) \tag{4.7}$$

is the *image gradient* at $\boldsymbol{x}_i$. This gradient can be computed using a variety of techniques (Schmid, Mohr, and Bauckhage 2000). The classic "Harris" detector (Harris and Stephens 1988) uses a [-2 -1 0 1 2] filter, but more modern variants (Schmid, Mohr, and Bauckhage 2000; Triggs 2004) convolve the image with horizontal and vertical derivatives of a Gaussian (typically with $\sigma = 1$).

The auto-correlation matrix $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{4.8}$$

where we have replaced the weighted summations with discrete convolutions with the weighting kernel w. This matrix can be interpreted as a tensor (multiband) image, where the outer products of the gradients ∇I are convolved with a weighting function w to provide a per-pixel estimate of the local (quadratic) shape of the auto-correlation function.

As first shown by Anandan (1984; 1989) and further discussed in Section 8.1.3 and (8.44), the inverse of the matrix $\boldsymbol{A}$ provides a lower bound on the uncertainty in the location of a matching patch. It is therefore a useful indicator of which patches can be reliably matched. The easiest way to visualize and reason about this uncertainty is to perform an eigenvalue analysis of the auto-correlation matrix $\boldsymbol{A}$, which produces two eigenvalues (λ_0, λ_1) and two eigenvector directions (Figure 4.6). Since the larger uncertainty depends on the smaller eigenvalue, i.e., $\lambda_0^{-1/2}$, it makes sense to find maxima in the smaller eigenvalue to locate good features to track (Shi and Tomasi 1994).

Förstner–Harris. While Anandan and Lucas and Kanade (1981) were the first to analyze the uncertainty structure of the auto-correlation matrix, they did so in the context of associating certainties with optic flow measurements. Förstner (1986) and Harris and Stephens

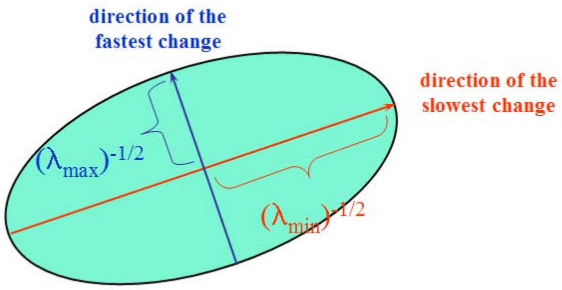

direction of the fastest change
direction of the slowest change
$(\lambda_{max})^{-1/2}$
$(\lambda_{min})^{-1/2}$

Figure 4.6 Uncertainty ellipse corresponding to an eigenvalue analysis of the auto-correlation matrix $\boldsymbol{A}$.

(1988) were the first to propose using local maxima in rotationally invariant scalar measures derived from the auto-correlation matrix to locate keypoints for the purpose of sparse feature matching. (Schmid, Mohr, and Bauckhage (2000); Triggs (2004) give more detailed historical reviews of feature detection algorithms.) Both of these techniques also proposed using a Gaussian weighting window instead of the previously used square patches, which makes the detector response insensitive to in-plane image rotations.

The minimum eigenvalue λ_0 (Shi and Tomasi 1994) is not the only quantity that can be used to find keypoints. A simpler quantity, proposed by Harris and Stephens (1988), is

$$\det(\boldsymbol{A}) - \alpha \operatorname{trace}(\boldsymbol{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \tag{4.9}$$

with $\alpha = 0.06$. Unlike eigenvalue analysis, this quantity does not require the use of square roots and yet is still rotationally invariant and also downweights edge-like features where $\lambda_1 \gg \lambda_0$. Triggs (2004) suggests using the quantity

$$\lambda_0 - \alpha\lambda_1 \tag{4.10}$$

(say, with $\alpha = 0.05$), which also reduces the response at 1D edges, where aliasing errors sometimes inflate 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. Brown, Szeliski, and Winder (2005), on the other hand, use the harmonic mean,

$$\frac{\det \boldsymbol{A}}{\operatorname{tr} \boldsymbol{A}} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1}, \tag{4.11}$$

which is a smoother function in the region where $\lambda_0 \approx \lambda_1$. Figure 4.7 shows isocontours of the various interest point operators, from which we can see how the two eigenvalues are blended to determine the final interest value.

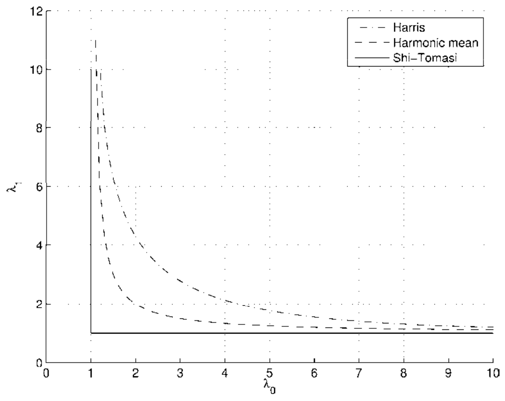


Figure 4.7 Isocontours of popular keypoint detection functions (Brown, Szeliski, and Winder 2004). Each detector looks for points where the eigenvalues λ_0, λ_1 of $\boldsymbol{A} = w * \nabla I \nabla I^T$ are both large.

1. Compute the horizontal and vertical derivatives of the image I_x and I_y by convolving the original image with derivatives of Gaussians (Section 3.2.3).
2. Compute the three images corresponding to the outer products of these gradients. (The matrix $\boldsymbol{A}$ is symmetric, so only three entries are needed.)
3. Convolve each of these images with a larger Gaussian.
4. Compute a scalar interest measure using one of the formulas discussed above.
5. Find local maxima above a certain threshold and report them as detected feature point locations.

Algorithm 4.1 Outline of a basic feature detection algorithm.

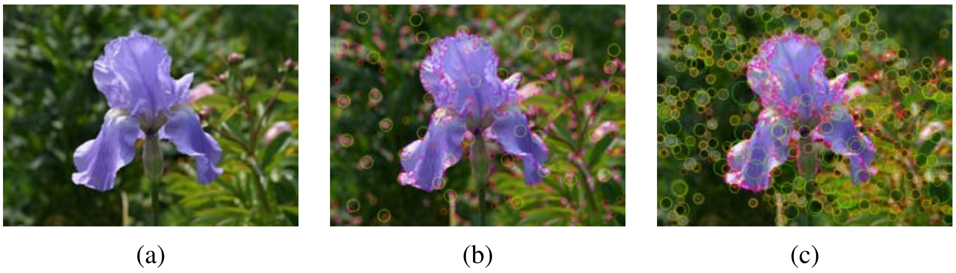
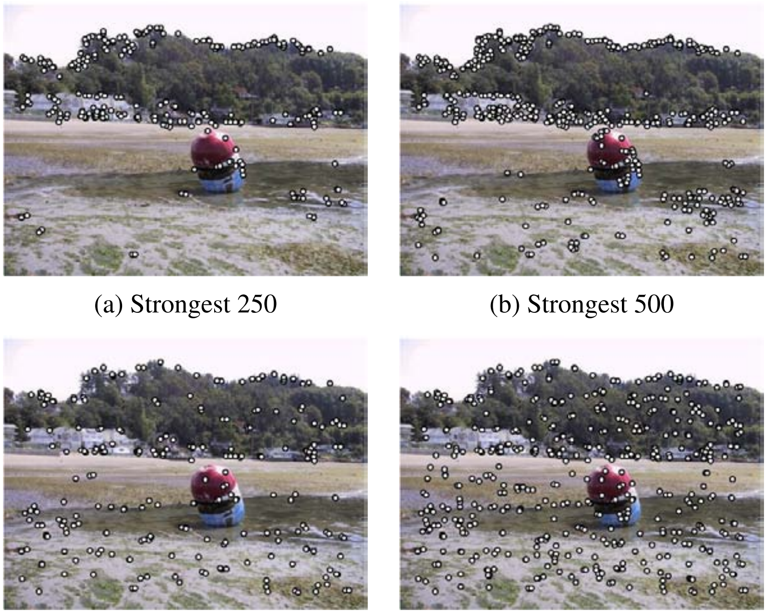


Figure 4.8 Interest operator responses: (a) Sample image, (b) Harris response, and (c) DoG response. The circle sizes and colors indicate the scale at which each interest point was detected. Notice how the two detectors tend to respond at complementary locations.

The steps in the basic auto-correlation-based keypoint detector are summarized in Algorithm 4.1. Figure 4.8 shows the resulting interest operator responses for the classic Harris detector as well as the difference of Gaussian (DoG) detector discussed below.

Adaptive non-maximal suppression (ANMS). While most feature detectors simply look for local maxima in the interest function, this can lead to an uneven distribution of feature points across the image, e.g., points will be denser in regions of higher contrast. To mitigate this problem, Brown, Szeliski, and Winder (2005) only detect features that are both local maxima and whose response value is significantly (10%) greater than that of all of its neighbors within a radius r (Figure 4.9c–d). They devise an efficient way to associate suppression radii with all local maxima by first sorting them by their response strength and then creating a second list sorted by decreasing suppression radius (Brown, Szeliski, and Winder 2005). Figure 4.9 shows a qualitative comparison of selecting the top n features and using ANMS.

Measuring repeatability. Given the large number of feature detectors that have been developed in computer vision, how can we decide which ones to use? Schmid, Mohr, and Bauckhage (2000) were the first to propose measuring the *repeatability* of feature detectors, which they define as the frequency with which keypoints detected in one image are found within ϵ (say, $\epsilon = 1.5$) pixels of the corresponding location in a transformed image. In their paper, they transform their planar images by applying rotations, scale changes, illumination changes, viewpoint changes, and adding noise. 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 the improved (Gaussian derivative) version of the Harris operator with $\sigma_d = 1$ (scale of the derivative Gaussian) and $\sigma_i = 2$ (scale of the integration Gaussian) works best.


(a) Strongest 250 (b) Strongest 500

(c) ANMS 250, $r = 24$ (d) ANMS 500, $r = 16$

Figure 4.9 Adaptive non-maximal suppression (ANMS) (Brown, Szeliski, and Winder 2005) © 2005 IEEE: The upper two images show the strongest 250 and 500 interest points, while the lower two images show the interest points selected with adaptive non-maximal suppression, along with the corresponding suppression radius r. Note how the latter features have a much more uniform spatial distribution across the image.

Scale invariance

In many situations, detecting features at the finest stable scale possible may not be appropriate. For example, when matching images with little high frequency detail (e.g., clouds), fine-scale features may not exist.

One solution to the problem is to extract features at a variety of scales, e.g., by performing the same operations at multiple resolutions in a pyramid and then matching features at the same level. This kind of approach is suitable when the images being matched do not undergo large scale changes, e.g., when matching successive aerial images taken from an airplane or stitching panoramas taken with a fixed-focal-length camera. Figure 4.10 shows the output of one such approach, the multi-scale, oriented patch detector of Brown, Szeliski, and Winder (2005), for which responses at five different scales are shown.

However, for most object recognition applications, the scale of the object in the image

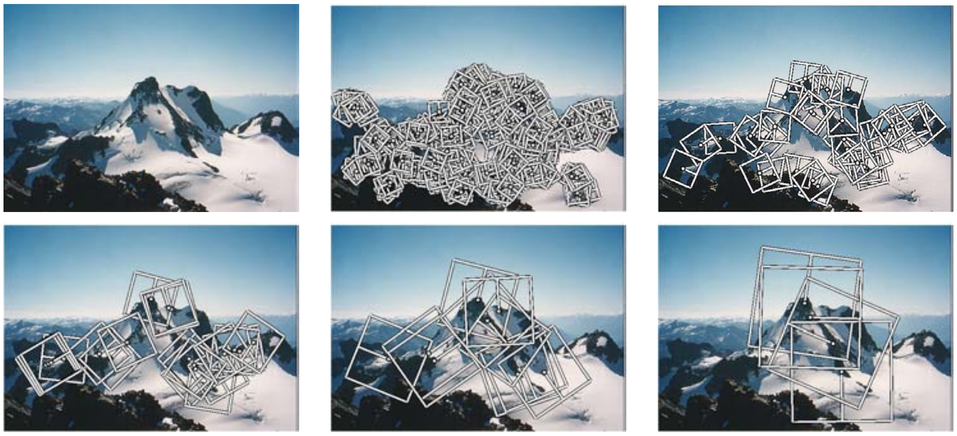


Figure 4.10 Multi-scale oriented patches (MOPS) extracted at five pyramid levels (Brown, Szeliski, and Winder 2005) © 2005 IEEE. The boxes show the feature orientation and the region from which the descriptor vectors are sampled.

is unknown. Instead of extracting features at many different scales and then matching all of them, it is more efficient to extract features that are stable in both location *and* scale (Lowe 2004; Mikolajczyk and Schmid 2004).

Early investigations into scale selection were performed by Lindeberg (1993; 1998b), who first proposed using extrema in the Laplacian of Gaussian (LoG) function as interest point locations. Based on this work, Lowe (2004) proposed computing a set of sub-octave Difference of Gaussian filters (Figure 4.11a), looking for 3D (space+scale) maxima in the resulting structure (Figure 4.11b), and then computing a sub-pixel space+scale location using a quadratic fit (Brown and Lowe 2002). The number of sub-octave levels was determined, after careful empirical investigation, to be three, which corresponds to a quarter-octave pyramid, which is the same as used by Triggs (2004).

As with the Harris operator, pixels where there is strong asymmetry in the local curvature of the indicator function (in this case, the DoG) are rejected. This is implemented by first computing the local Hessian of the difference image D,

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}, \tag{4.12}$$

and then rejecting keypoints for which

$$\frac{\mathrm{Tr}(\boldsymbol{H})^2}{\mathrm{Det}(\boldsymbol{H})} > 10. \tag{4.13}$$

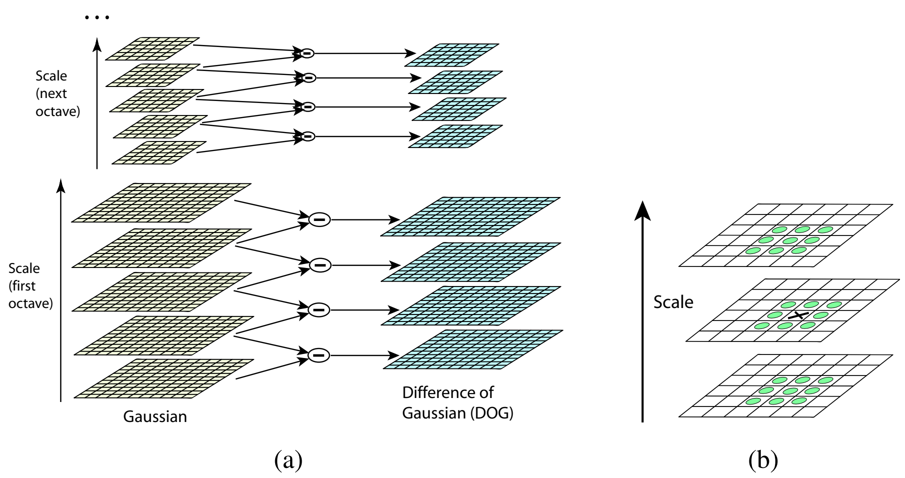


Figure 4.11 Scale-space feature detection using a sub-octave Difference of Gaussian pyramid (Lowe 2004) © 2004 Springer: (a) Adjacent levels of a sub-octave Gaussian pyramid are subtracted to produce Difference of Gaussian images; (b) extrema (maxima and minima) in the resulting 3D volume are detected by comparing a pixel to its 26 neighbors.

While Lowe's Scale Invariant Feature Transform (SIFT) performs well in practice, it is not based on the same theoretical foundation of maximum spatial stability as the auto-correlation-based detectors. (In fact, its detection locations are often complementary to those produced by such techniques and can therefore be used in conjunction with these other approaches.) In order to add a scale selection mechanism to the Harris corner detector, Mikolajczyk and Schmid (2004) evaluate the Laplacian of Gaussian function at each detected Harris point (in a multi-scale pyramid) and keep only those points for which the Laplacian is extremal (larger or smaller than both its coarser and finer-level values). An optional iterative refinement for both scale and position is also proposed and evaluated. Additional examples of scale invariant region detectors are discussed by Mikolajczyk, Tuytelaars, Schmid *et al.* (2005); Tuytelaars and Mikolajczyk (2007).

Rotational invariance and orientation estimation

In addition to dealing with scale changes, most image matching and object recognition algorithms need to deal with (at least) in-plane image rotation. One way to deal with this problem is to design descriptors that are rotationally invariant (Schmid and Mohr 1997), but such descriptors have poor discriminability, i.e. they map different looking patches to the same descriptor.

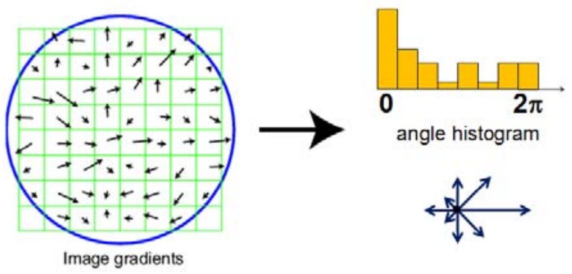


Figure 4.12 A dominant orientation estimate can be computed by creating a histogram of all the gradient orientations (weighted by their magnitudes or after thresholding out small gradients) and then finding the significant peaks in this distribution (Lowe 2004) © 2004 Springer.

A better method is to estimate a *dominant orientation* at each detected keypoint. Once the local orientation and scale of a keypoint have been estimated, a scaled and oriented patch around the detected point can be extracted and used to form a feature descriptor (Figures 4.10 and 4.17).

The simplest possible orientation estimate is the average gradient within a region around the keypoint. If a Gaussian weighting function is used (Brown, Szeliski, and Winder 2005), this average gradient is equivalent to a first-order steerable filter (Section 3.2.3), i.e., it can be computed using an image convolution with the horizontal and vertical derivatives of Gaussian filter (Freeman and Adelson 1991). In order to make this estimate more reliable, it is usually preferable to use a larger aggregation window (Gaussian kernel size) than detection window (Brown, Szeliski, and Winder 2005). The orientations of the square boxes shown in Figure 4.10 were computed using this technique.

Sometimes, however, the averaged (signed) gradient in a region can be small and therefore an unreliable indicator of orientation. A more reliable technique is to look at the *histogram* of orientations computed around the keypoint. Lowe (2004) computes a 36-bin histogram of edge orientations weighted by both gradient magnitude and Gaussian distance to the center, finds all peaks within 80% of the global maximum, and then computes a more accurate orientation estimate using a three-bin parabolic fit (Figure 4.12).

Affine invariance

While scale and rotation invariance are highly desirable, for many applications such as *wide baseline stereo matching* (Pritchett and Zisserman 1998; Schaffalitzky and Zisserman 2002) or location recognition (Chum, Philbin, Sivic *et al.* 2007), full affine invariance is preferred.

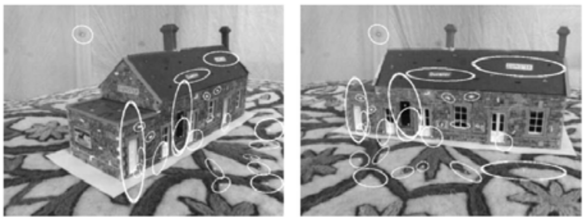


Figure 4.13 Affine region detectors used to match two images taken from dramatically different viewpoints (Mikolajczyk and Schmid 2004) © 2004 Springer.

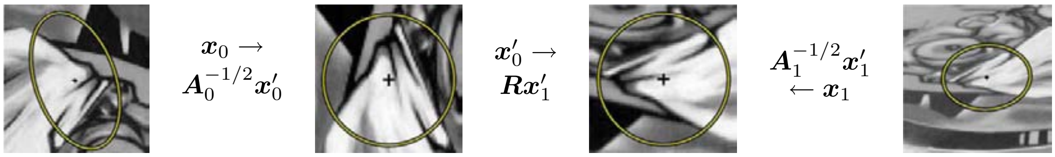

$\boldsymbol{x}_0 \rightarrow \boldsymbol{A}_0^{-1/2}\boldsymbol{x}'_0$
$\boldsymbol{x}'_0 \rightarrow \boldsymbol{R}\boldsymbol{x}'_1$
$\boldsymbol{A}_1^{-1/2}\boldsymbol{x}'_1 \leftarrow \boldsymbol{x}_1$

Figure 4.14 Affine normalization using the second moment matrices, as described by Mikolajczyk, Tuytelaars, Schmid *et al.* (2005) © 2005 Springer. After image coordinates are transformed using the matrices $\boldsymbol{A}_0^{-1/2}$ and $\boldsymbol{A}_1^{-1/2}$, they are related by a pure rotation $\boldsymbol{R}$, which can be estimated using a dominant orientation technique.

Affine-invariant detectors not only respond at consistent locations after scale and orientation changes, they also respond consistently across affine deformations such as (local) perspective foreshortening (Figure 4.13). In fact, for a small enough patch, any continuous image warping can be well approximated by an affine deformation.

To introduce affine invariance, several authors have proposed fitting an ellipse to the autocorrelation or Hessian matrix (using eigenvalue analysis) and then using the principal axes and ratios of this fit as the affine coordinate frame (Lindeberg and Garding 1997; Baumberg 2000; Mikolajczyk and Schmid 2004; Mikolajczyk, Tuytelaars, Schmid *et al.* 2005; Tuytelaars and Mikolajczyk 2007). Figure 4.14 shows how the square root of the moment matrix can be used to transform local patches into a frame which is similar up to rotation.

Another important affine invariant region detector is the maximally stable extremal region (MSER) detector developed by Matas, Chum, Urban *et al.* (2004). To detect MSERs, binary regions are computed by thresholding the image at all possible gray levels (the technique therefore only works for grayscale images). This operation can be performed efficiently by first sorting all pixels by gray value and then incrementally adding pixels to each connected component as the threshold is changed (Nistér and Stewénius 2008). As the threshold is changed, the area of each component (region) is monitored; regions whose rate of change of area with respect to the threshold is minimal are defined as *maximally stable*. Such regions

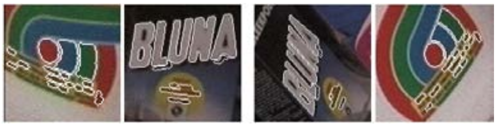


Figure 4.15 Maximally stable extremal regions (MSERs) extracted and matched from a number of images (Matas, Chum, Urban *et al.* 2004) © 2004 Elsevier.

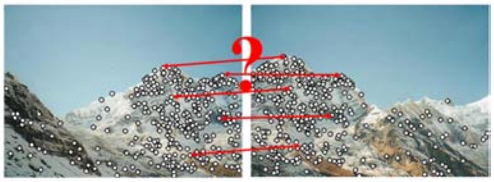


Figure 4.16 Feature matching: how can we extract local descriptors that are invariant to inter-image variations and yet still discriminative enough to establish correct correspondences?

are therefore invariant to both affine geometric and photometric (linear bias-gain or smooth monotonic) transformations (Figure 4.15). If desired, an affine coordinate frame can be fit to each detected region using its moment matrix.

The area of feature point detectors continues to be very active, with papers appearing every year at major computer vision conferences (Xiao and Shah 2003; Koethe 2003; Carneiro and Jepson 2005; Kenney, Zuliani, and Manjunath 2005; Bay, Tuytelaars, and Van Gool 2006; Platel, Balmachnova, Florack *et al.* 2006; Rosten and Drummond 2006). Mikolajczyk, Tuytelaars, Schmid *et al.* (2005) survey a number of popular affine region detectors and provide experimental comparisons of their invariance to common image transformations such as scaling, rotations, noise, and blur. These experimental results, code, and pointers to the surveyed papers can be found on their Web site at http://www.robots.ox.ac.uk/~vgg/research/affine/.

Of course, keypoints are not the only features that can be used for registering images. Zoghlami, Faugeras, and Deriche (1997) use line segments as well as point-like features to estimate homographies between pairs of images, whereas Bartoli, Coquerelle, and Sturm (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 correspondences for wide baseline stereo matching, whereas Kadir, Zisserman, and Brady (2004) detect salient regions where patch entropy and its rate of change with scale are locally maximal. Corso and Hager (2005) use a related technique to fit 2D oriented Gaussian kernels to homogeneous regions. More details on techniques for finding and matching curves, lines, and regions can be found later in this chapter.

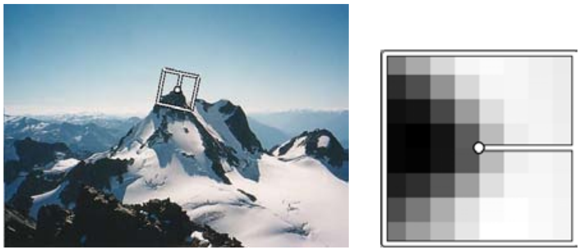


Figure 4.17 MOPS descriptors are formed using an 8×8 sampling of bias and gain normalized intensity values, with a sample spacing of five pixels relative to the detection scale (Brown, Szeliski, and Winder 2005) © 2005 IEEE. This low frequency sampling gives the features some robustness to interest point location error and is achieved by sampling at a higher pyramid level than the detection scale.

4.1.2 Feature descriptors

After detecting features (keypoints), we must *match* them, i.e., we must 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* (Zhang, Deriche, Faugeras *et al.* 1995; Loop and Zhang 1999; Scharstein and Szeliski 2002), the local motion around each feature point may be mostly translational. In this case, simple error metrics, such as the *sum of squared differences* or *normalized cross-correlation*, described in Section 8.1 can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid (2005), 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 Section 8.1.3 but this can be time consuming and can sometimes even decrease performance (Brown, Szeliski, and Winder 2005).

In most cases, however, the local appearance of features will change in orientation and scale, and sometimes even undergo affine deformations. Extracting a local scale, orientation, or affine frame estimate and then using this to resample the patch before forming the feature descriptor is thus usually preferable (Figure 4.17).

Even after compensating for these changes, the local appearance of image patches will usually still vary from image to image. How can we make image descriptors more invariant to such changes, while still preserving discriminability between different (non-corresponding) patches (Figure 4.16)? Mikolajczyk and Schmid (2005) review some recently developed view-invariant local image descriptors and experimentally compare their performance. Below, we describe a few of these descriptors in more detail.

Bias and gain normalization (MOPS). For tasks that do not exhibit large amounts of foreshortening, such as image stitching, simple normalized intensity patches perform reasonably well and are simple to implement (Brown, Szeliski, and Winder 2005) (Figure 4.17). In order to compensate for slight inaccuracies in the feature point detector (location, orientation, and scale), these multi-scale oriented patches (MOPS) are sampled at a spacing of five pixels relative to the detection scale, using a coarser level of the image pyramid to avoid aliasing. To compensate for affine photometric variations (linear exposure changes or bias and gain, (3.3)), patch intensities are re-scaled so that their mean is zero and their variance is one.

Scale invariant feature transform (SIFT). SIFT features are formed by computing the gradient at each pixel in a 16×16 window around the detected keypoint, using the appropriate level of the Gaussian pyramid at which the keypoint was detected. The gradient magnitudes are downweighted by a Gaussian fall-off function (shown as a blue circle in (Figure 4.18a) in order to reduce the influence of gradients far from the center, as these are more affected by small misregistrations.

In each 4×4 quadrant, a gradient orientation histogram is formed by (conceptually) adding the weighted gradient value to one of eight orientation histogram bins. To reduce the effects of location and dominant orientation misestimation, each of the original 256 weighted gradient magnitudes is softly added to $2 \times 2 \times 2$ histogram bins using trilinear interpolation. Softly distributing values to adjacent histogram bins is generally a good idea in any application where histograms are being computed, e.g., for Hough transforms (Section 4.3.2) or local histogram equalization (Section 3.1.4).

The resulting 128 non-negative values form a raw version of the SIFT descriptor vector. To reduce the effects of contrast or gain (additive variations are already removed by the gradient), the 128-D vector is normalized to unit length. To further make the descriptor robust to other photometric variations, values are clipped to 0.2 and the resulting vector is once again renormalized to unit length.

PCA-SIFT. Ke and Sukthankar (2004) propose a simpler way to compute descriptors inspired by SIFT; it computes the x and y (gradient) derivatives over a 39×39 patch and then reduces the resulting 3042-dimensional vector to 36 using principal component analysis (PCA) (Section 14.2.1 and Appendix A.1.2). Another popular variant of SIFT is SURF (Bay, Tuytelaars, and Van Gool 2006), which uses box filters to approximate the derivatives and integrals used in SIFT.

Gradient location-orientation histogram (GLOH). This descriptor, developed by Mikolajczyk and Schmid (2005), is a variant on SIFT that uses a log-polar binning structure instead of the four quadrants used by Lowe (2004) (Figure 4.19). The spatial bins are of radius 6,

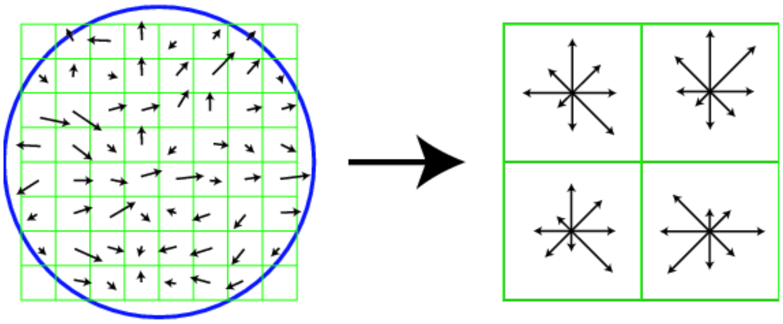


Figure 4.18 A schematic representation of Lowe's (2004) scale invariant feature transform (SIFT): (a) Gradient orientations and magnitudes are computed at each pixel and weighted by a Gaussian fall-off function (blue circle). (b) A weighted gradient orientation histogram is then computed in each subregion, using trilinear interpolation. While this figure shows an 8×8 pixel patch and a 2×2 descriptor array, Lowe's actual implementation uses 16×16 patches and a 4×4 array of eight-bin histograms.

11, and 15, with eight angular bins (except for the central region), for a total of 17 spatial bins and 16 orientation bins. The 272-dimensional histogram is then projected onto a 128-dimensional descriptor using PCA trained on a large database. In their evaluation, Mikolajczyk and Schmid (2005) found that GLOH, which has the best performance overall, outperforms SIFT by a small margin.

Steerable filters. Steerable filters (Section 3.2.3) 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.

Performance of local descriptors. Among the local descriptors that Mikolajczyk and Schmid (2005) compared, they found that GLOH performed best, followed closely by SIFT (see Figure 4.25). They also present results for many other descriptors not covered in this book.

The field of feature descriptors continues to evolve rapidly, with some of the newer techniques looking at local color information (van de Weijer and Schmid 2006; Abdel-Hakim and Farag 2006). Winder and Brown (2007) develop a multi-stage framework for feature descriptor computation that subsumes both SIFT and GLOH (Figure 4.20a) and also allows them to learn optimal parameters for newer descriptors that outperform previous hand-tuned

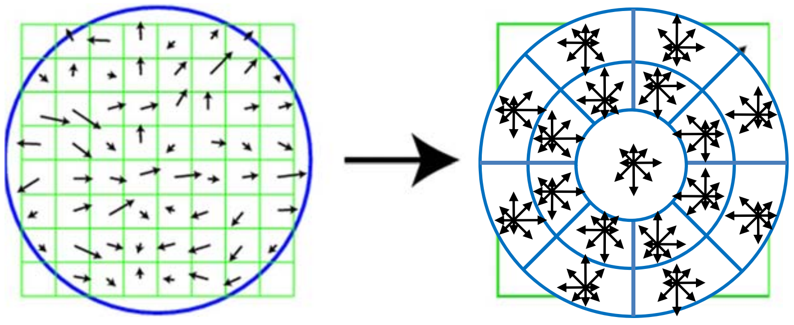


Figure 4.19 The gradient location-orientation histogram (GLOH) descriptor uses log-polar bins instead of square bins to compute orientation histograms (Mikolajczyk and Schmid 2005).

descriptors. Hua, Brown, and Winder (2007) extend this work by learning lower-dimensional projections of higher-dimensional descriptors that have the best discriminative power. Both of these papers use a database of real-world image patches (Figure 4.20b) obtained by sampling images at locations that were reliably matched using a robust structure-from-motion algorithm applied to Internet photo collections (Snavely, Seitz, and Szeliski 2006; Goesele, Snavely, Curless *et al.* 2007). In concurrent work, Tola, Lepetit, and Fua (2010) developed a similar DAISY descriptor for dense stereo matching and optimized its parameters based on ground truth stereo data.

While these techniques construct feature detectors that optimize for repeatability across *all* object classes, it is also possible to develop class- or instance-specific feature detectors that maximize *discriminability* from other classes (Ferencz, Learned-Miller, and Malik 2008).

4.1.3 Feature matching

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images. In this section, we divide this problem into two separate components. The first is to select a *matching strategy*, which determines which correspondences are passed on to the next stage for further processing. The second is to devise efficient *data structures* and *algorithms* to perform this matching as quickly as possible. (See the discussion of related techniques in Section 14.3.2.)