Chapter 9

Scale space

Scale is one of the most important concepts in human vision. When we look at a scene, we instantaneously view its contents at multiple scale levels. For example, take a look at figure 9.1. This painting contains a wealth of objects 'living' at a wide range of scales, from small-scale objects such as the cracks in the pavement to large-scale structures such as the building in the background. Humans apparently are not easily confused by the occurrence of multiple scales in a single scene, because we have no trouble at all identifying what is building and what is not, despite the fact that the building consists of numerous details and components at lower scales.

Another vision phenomenon is that details (low-scale structures) simply seem to *disappear* if we move away from an object. For example, suppose you are facing the wall of a large brick building, close enough to distinguish individual bricks and the mortar between. If you move away from the building, the layers of mortar will disappear at a certain distance, giving the wall a uniform look. Instead of disappearing, details may also seem to *merge* under particular circumstances. For example, a row of shrubs may seem to be a single structure when seen from a large enough distance.

Including the notion of scale into computer vision and image processing techniques is no trivial matter. In the previous chapters, almost all of the techniques presented focused on small-scale image structures. The smallest scale in digital images is the scale of the individual pixels (the *inner scale* of the image), and our spatial operators mostly used discrete kernels with a size very close to this scale, such as 3×3 kernels, or continuous kernels with a small support. In this chapter, we will examine a technique to control the scale with which operators 'look' at a digital image.

9.1 Scale space

The key to adjusting an operator to a certain scale seems to be altering the image *resolution*. Put naively, we could re-sample a digital image at a low resolution to bring

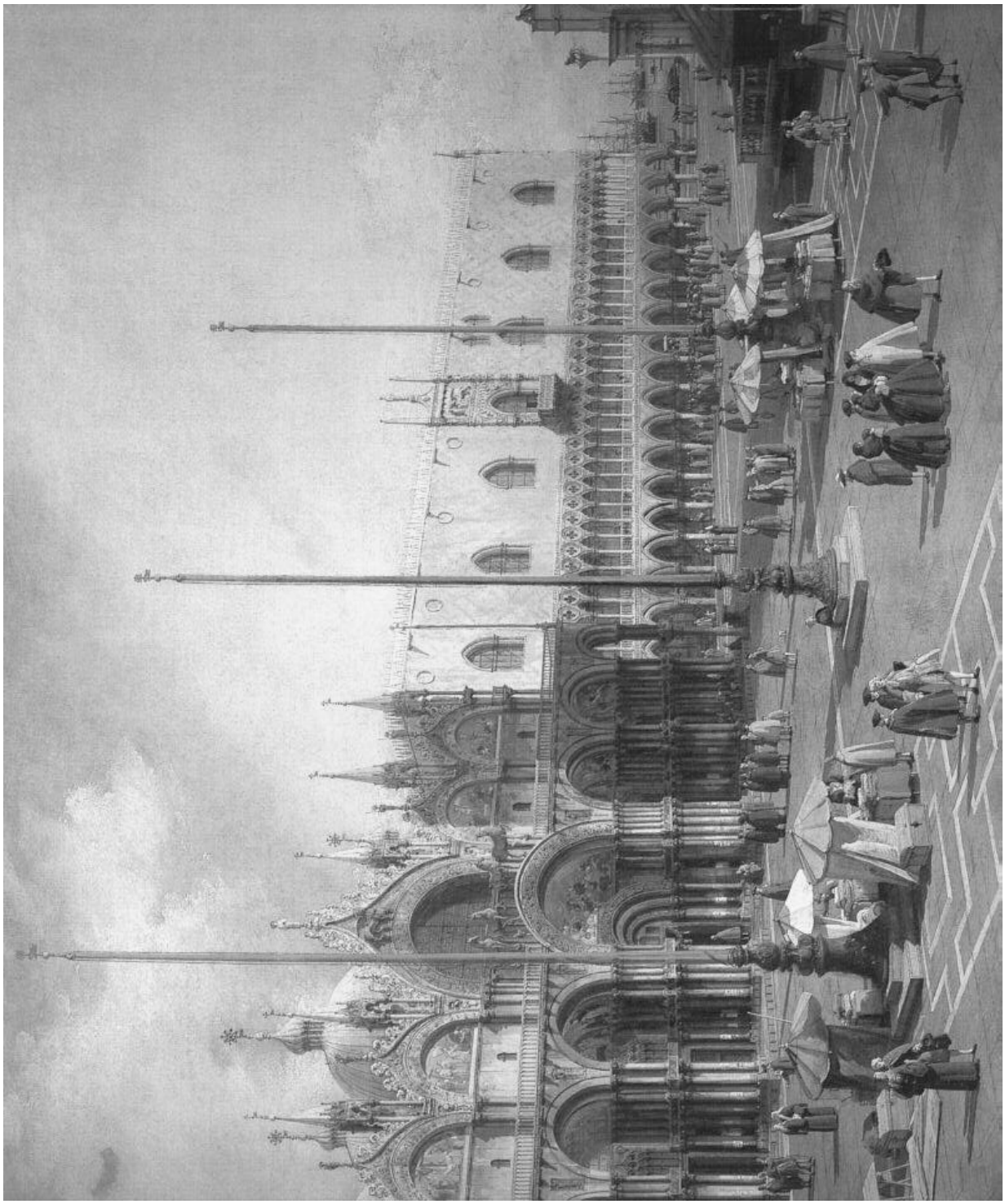


Figure 9.1 Canaletto's *San Marco* square. An example of a scene with objects at multiple scales.

large-scale object into the range of, *e.g.*, our 3×3 kernel. For an example, see figure 9.2, where we have decreased the resolution so that a 3×3 kernel operates on larger-scale structures than in the original image. The resolution of an image can be decreased by

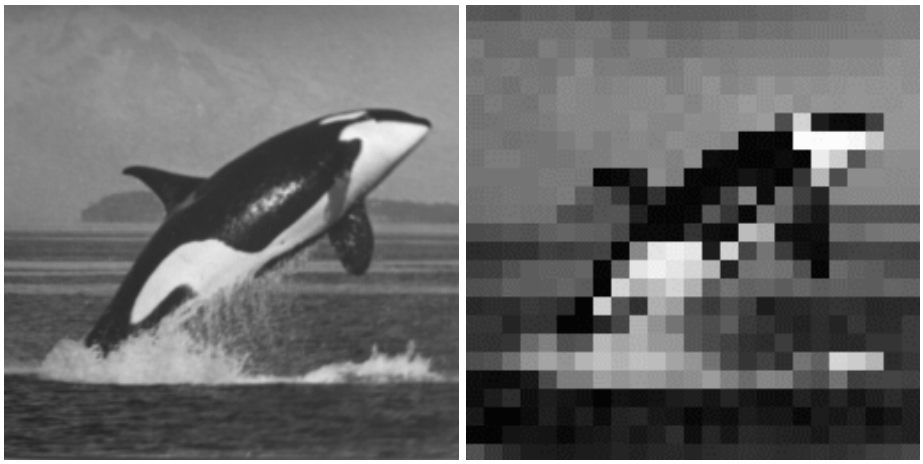


Figure 9.2 A 256×256 image, and the same image re-sampled to a resolution of 25×25 pixels using nearest neighbor interpolation. Using a 3×3 kernel on the right image now operates on structures of a much larger scale than in the original image. Notice that the re-sampling apparently introduces structures not present in the original. Consider for instance the shape and patterning of the orca's body as suggested by the low-resolution image, and compare it to the actual shape and pattern of the original.

merging the grey values of a block of pixels into the new grey value of a larger pixel. This merging can, *e.g.*, be done by averaging grey values, by taking the center pixel value, or by taking the maximum, minimum, modal, nearest neighbor value, *etc.* This technique of lowering the resolution to bring large-scale structures into 'kernel range' and remove small-scale structures, is a much-used and easy to implement technique; we will come back to this technique in section 9.3. It however suffers from some problems that are inherent to the very blocky nature of representing low-resolution images by square pixels, the most serious of which is often the occurrence of *spurious resolution*: the apparent creation of structures that are not present in the original image. Spurious resolution can be seen in figure 9.2.

We are looking for an operation that lowers resolution but does not introduce new details into the image; that avoids spurious resolution. We have already come across such an operation many times: convolution with a Gaussian. Figure 9.3 shows example of resolution lowering by convolution with a Gaussian. Note that we use the term *resolution* here in a more abstract, intrinsic sense than before: upon convolution with a Gaussian, the actual number of pixels used in an image need not change, so the resolution in terms of the number of pixels used does not change. However, after convolution the smallest recognizable detail in the image has become larger, and it that sense the resolution *is* lower than before. We can also say that the intrinsic resolution of the image

had actually lowered, because it is possible to re-sample the image using fewer pixels than in the original *and retain all of the information contained in the image.* In practice, we say that Gaussian convolution lowers the image resolution, even if we represent the convolved image using the same number of pixels as with the original image. The

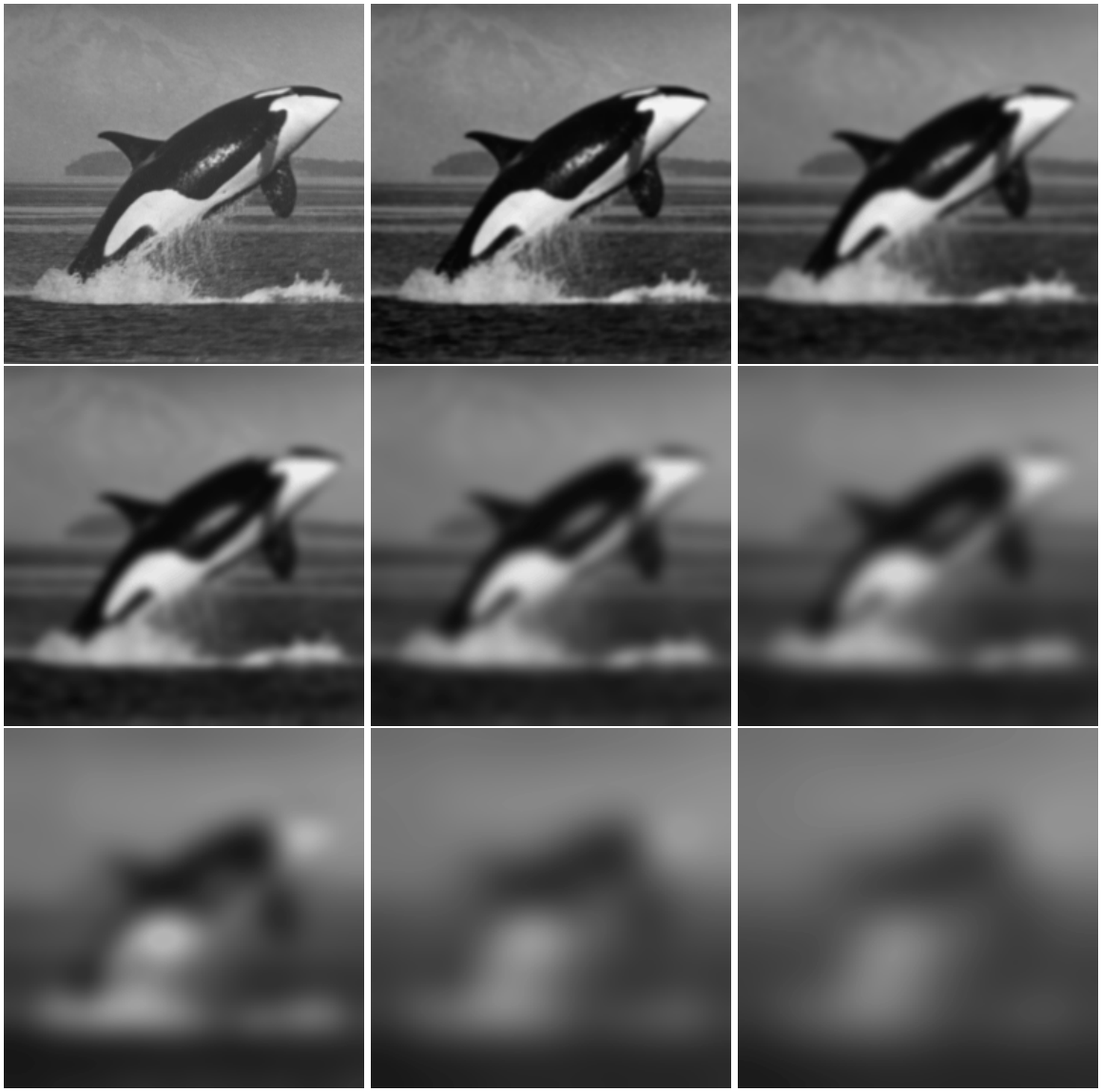


Figure 9.3 Lowering the resolution by Gaussian convolution: the top left image is convolved with a Gaussian kernel of increasing width (other images). Note that more and more details vanish from the images, and that effectively the resolution of the images is lowered.

Gaussian kernel is not the only convolution kernel that lowers resolution without creating spurious detail. But if we add the demand that we want to be able to combine two resolution lowering steps (*i.e.*, Gaussian convolution steps) into one single step, than the

Gaussian kernel *is* the only possible kernel. This demand makes intuitive sense: suppose we wish to lower the resolution by some amount a, and after that by some amount b, then it should be possible to do this in one step, lowering the resolution by an amount $a + b$.

The stack of images as seen in figure 9.3 is called a Gaussian *scale space*. We denote the Gaussian scale space of an image $f(x, y)$ by $f_\sigma(x, y)$, where σ is the standard deviation of the Gaussian kernel $g_\sigma(x, y)$ used in the convolution:

$$f_\sigma(x, y) = g_\sigma(x, y) * f(x, y).$$

The Gaussian scale space is called a one-parameter extension of the original image, because we now have three parameters (x, y, and σ) instead of two (x and y). The resolution parameter σ is called the *scale* of the image $f_\sigma(x, y)$

The Gaussian scale space can be viewed as a stack of images, where the original image is at the bottom of the stack ($f_0(x, y) = f(x, y)$), and the image resolution gets lower as we rise in the stack, see figure 9.4.

With the Gaussian scale space, we now have a construct that allows us to 'select' an image at a certain level of resolution. The next question is how to use this construct so that we are able to apply an operator to an image only at a certain scale level σ_1. In general, it is not very effective to apply such an operator directly to the image $f_{\sigma_1}(x, y)$. The reason for this is that –even though the intrinsic resolution of the image $f_{\sigma_1}(x, y)$ is lower than the resolution of the original[1]– we still use the same number of pixels as in the original image to represent the image at scale σ_1. Subsampling the image so that we use less pixels than in the original image can partially solve this problem. However, there is a method that solves this problem in a mathematically sound way when the operator we wish to apply is a differential operator. This method is covered in the next section.

9.1.1 Scaled differential operators

Given an image f_{σ_1}; an image at scale σ_1: $f_{\sigma_1} = g_{\sigma_1} * f$. Suppose we wish to compute the derivative of the image f_{σ_1} to x, *i.e.*, the derivative of f to x at scale σ_1. Consider this equation:

$$\frac{\partial f_{\sigma_1}}{\partial x} = \frac{\partial}{\partial x}(g_{\sigma_1} * f) \overset{\mathcal{F}}{\longrightarrow} 2\pi i u(G_{\sigma_1}F) = (2\pi i u G_{\sigma_1})F \overset{\mathcal{F}^{-1}}{\longrightarrow} \frac{\partial g_{\sigma_1}}{\partial x} * f.$$

[1] Assuming that $\sigma_1 \neq 0$.

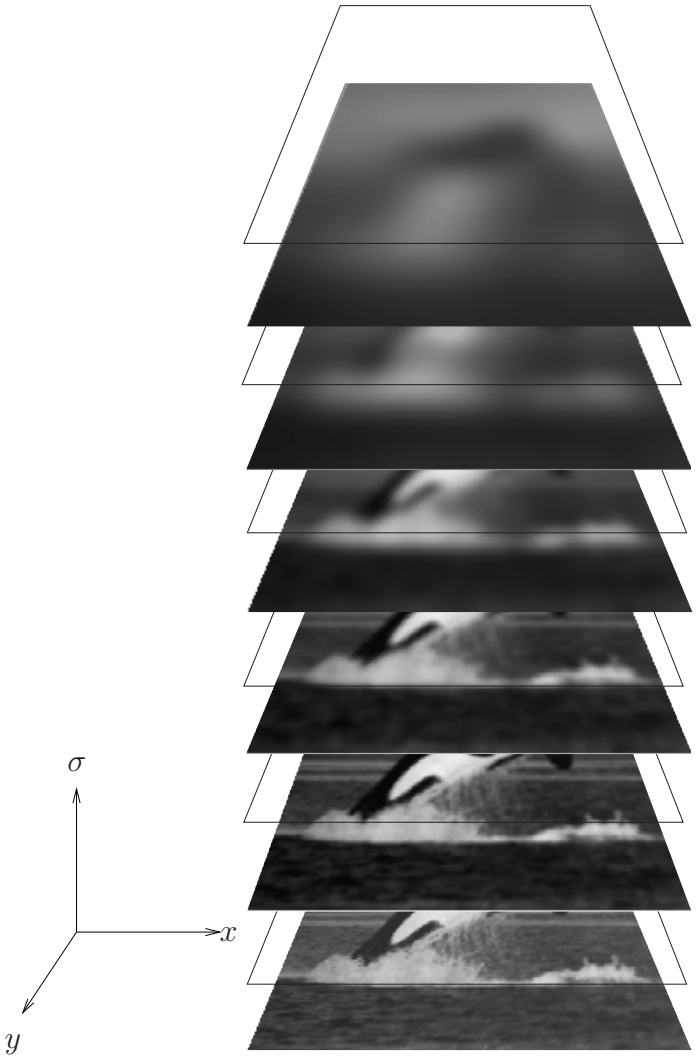


Figure 9.4 The Gaussian scale space of an image. The original image rests at the bottom of the stack ($\sigma = 0$), and the scale σ increases as we rise in the stack.

Here, $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier and inverse Fourier transform. In this equation, we have used the properties that convolution in the spatial domain becomes multiplication in the Fourier domain, and that taking the derivative to x in the spatial domain becomes multiplication with $2\pi iu$ in the Fourier domain. Cutting out the middle part of the equation gives us:

$$\frac{\partial f_{\sigma_1}}{\partial x} = \frac{\partial g_{\sigma_1}}{\partial x} * f.$$

The left term shows us what we wish to compute: the scaled (σ_1) derivative of f to x. The right term shows us how we can compute this in a well-posed mathematical way:

we compute the derivative *of the Gaussian kernel* and convolve the result with our original image f. This way, we avoid having to compute the derivative of the digital image f, which is an ill-posed problem. Instead, we only need to compute the derivative of a Gaussian, which is a well-posed problem, because the Gaussian function is a differentiable function. If we compute the scaled derivative entirely in the Fourier domain, only multiplications are needed.

Figure 9.5 shows some examples of scaled x derivatives of an image.

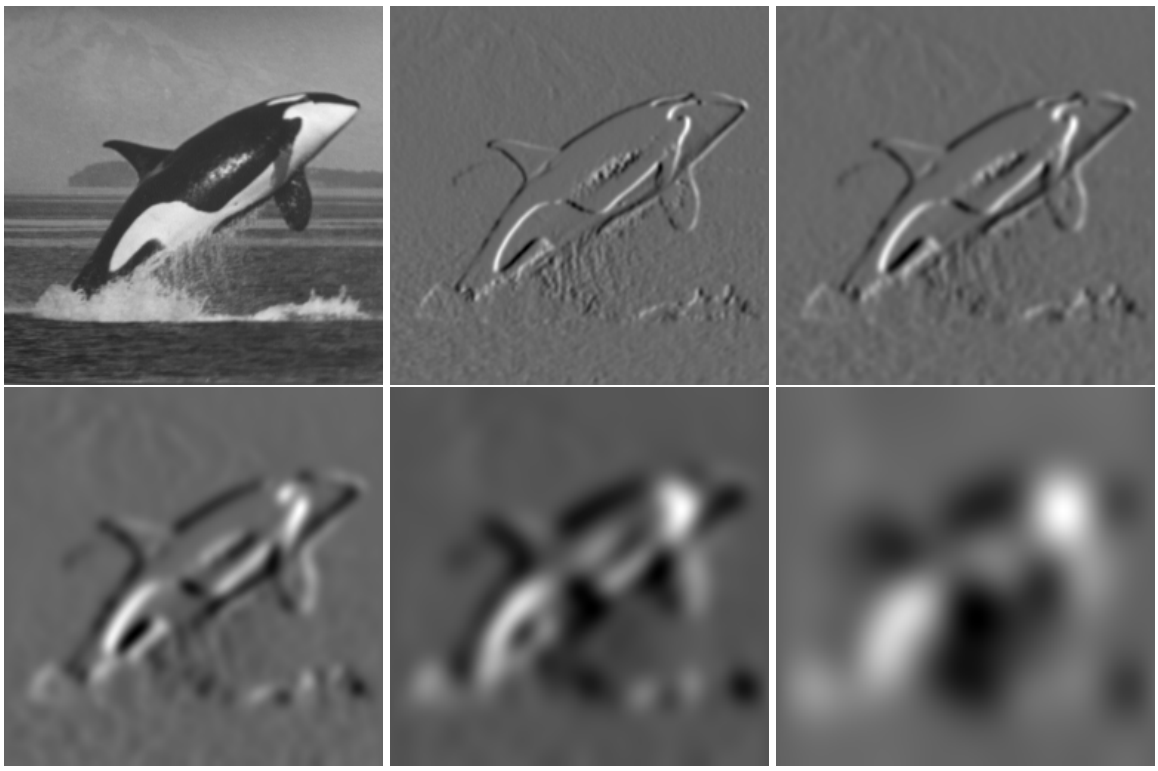


Figure 9.5 Scaled x derivatives of the top-left (256×256) image, respectively with $\sigma = 1, 2, 4, 8, 16$ pixels.

This example with the scaled x derivative of an image can be generalized to all spatial derivatives: to compute a scaled spatial derivative, we need only compute the appropriate derivative of the Gaussian kernel (with standard deviation σ equal to the desired scale), and convolve the result with the original image. So, if we denote partial derivatives by subscripts x or y, for example:

$$
\begin{aligned}
f_{\sigma_1,x} &= g_{\sigma_1,x} * f \\
f_{\sigma_1,y} &= g_{\sigma_1,y} * f \\
f_{\sigma_1,xy} &= g_{\sigma_1,xy} * f \\
f_{\sigma_1,xxx} &= g_{\sigma_1,xxx} * f \\
&\textit{etc.}
\end{aligned}
$$

9.1.2 Scaled differential features

Many image features, such as edges, ridges, corners, *etc.* can be detected using differential operators. In many cases, the detection of interesting features benefits from tuning the differential operators to the appropriate scale.

Invariants. In many applications, features like the x or y derivative of an image are not by themselves interesting features. More interesting are so-called *invariant* features –also concisely denoted simply by *invariants*– *i.e.,* features that are insensitive to some kind of transformation. For example, the x derivative of an image is not invariant under rotation, because the following two operations give different results: (1) computing the x derivative, and (2) rotating the image, computing the x derivative, then rotating the result back. The gradient norm ($\sqrt{f_x^2(a) + f_y^2(a)}$, in a point a), however, *is* invariant under rotation because the two operations: (1) computing the gradient norm, and (2) rotating the image, computing the gradient norm, then rotating the result back, yields the same result. All of the features treated in the below are invariant under translations and rotations.

Gradient-based coordinate system. The gradient vector plays an important role in the detection of many different types of features. It is therefore often convenient to abandon the usual (x, y) Cartesian coordinate frame, and replace it with a *local* coordinate frame based on the gradient vector w and its right-handed normal vector v. This local (v, w) frame is sometimes called a *gauge* or *Frenet* frame. In formulas, the v and w vectors can be written as

$$w = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \quad \text{and} \quad v = \begin{pmatrix} f_y \\ -f_x \end{pmatrix}.$$

Figure 9.6 shows an example of an image where we have drawn several (v, w) frames. Note that the gradient vector w always points 'uphill', and that the normal vector v is always in the direction of an isophote (*i.e.,* a curve of equal image intensity). It may at first seem strange to use a coordinate system that may point a different way in each pixel, but it is in fact very convenient when we consider features invariant under rotations (and translations). This is because when we rotate an image, *the* (v, w) *frames rotate with it,* because the v and w vectors are based on the local image structure. This means that all derivatives computed in the v or w directions are also invariant. Put differently, the (v, w) frame captures some of the intrinsic differential structure of the image, independent of how you have chosen your (x, y) coordinate system.

As we denote the derivative of an image in the x direction by f_x, we adopt a similar notation for derivatives in the v or w direction. For example, the first derivative of f in the w direction is denoted by f_w, the second by f_{ww}, *etc.*

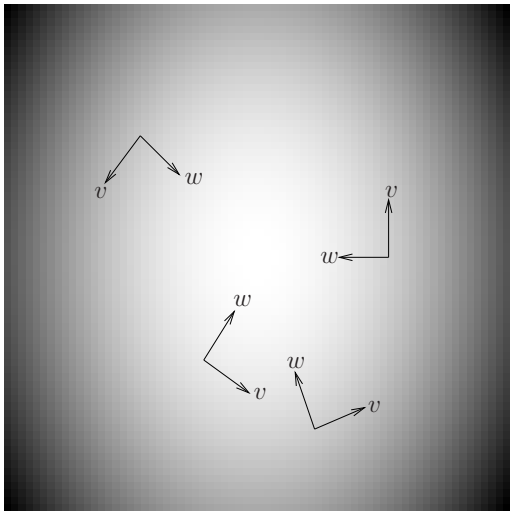


Figure 9.6 The (v, w) coordinate frame drawn at four random points in an image. The w vector equals the local gradient.

For the actual computation of expressions such as f_w, f_{vw}, f_{www}, *etc.* we usually need to rewrite them in a Cartesian form (using f_x, f_y, f_{xy}, *etc.* which we know how to compute). The intermezzo below shows how this 'translation' can be achieved.

Intermezzo*

Derivatives of a function f specified using v and w subscripting can be translated to a Cartesian form (using only partial derivatives to x and y) by using the definition of the *directional derivative:*

The derivative of f in the direction of a certain row vector a is defined as

$$f_a = \frac{1}{||a||}(a \cdot \nabla)f,$$

where ∇ is the Nabla vector defined by $\nabla = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}$, and the dot $(\cdot)$ denotes the inner product.

For example, if we take a to be the vector $(1, 2)$, then f_a equals:

$$\begin{aligned} f_a &= \tfrac{1}{\sqrt{5}} \left((1 \;\; 2) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \right) f = \tfrac{1}{\sqrt{5}} \left(\tfrac{\partial}{\partial x} + 2\tfrac{\partial}{\partial y} \right) f = \tfrac{1}{\sqrt{5}} \left(\tfrac{\partial f}{\partial x} + 2\tfrac{\partial f}{\partial y} \right) = \\ &= \tfrac{1}{\sqrt{5}}(f_x + 2f_y). \end{aligned}$$

Note that this directional derivative definition is consistent with the definition of the Cartesian derivatives. For example, f_x, *i.e.*, the directional derivative of f in the direction $(1, 0)$:

$$f_x = \frac{1}{1}\left((1 \;\; 0) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}\right) f = \left(\frac{\partial}{\partial x}\right) f = \frac{\partial f}{\partial x}$$

We can now compute the derivatives f_v and f_w:[2]

$$f_w = \frac{1}{\sqrt{f_x^2 + f_y^2}}\left((f_x \;\; f_y) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}\right) f = \frac{f_x f_x + f_y f_y}{\sqrt{f_x^2 + f_y^2}} = \sqrt{f_x^2 + f_y^2}.$$

$$f_v = \frac{1}{\sqrt{f_x^2 + f_y^2}}\left((f_y \;\; -f_x) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}\right) f = \frac{(f_y f_x - f_x f_y)}{\sqrt{f_x^2 + f_y^2}} = 0.$$

Perhaps not surprisingly, the derivative f_w equals the gradient magnitude, and f_v is zero. The latter is because v is in the direction of the local isophote, *i.e.*, the curve where f does not change value.

Higher order directional derivatives can be computed using the following extension:

$$f_{\underbrace{a \ldots a}_{n \text{ times}}} = \frac{1}{||a||^n}(a \cdot \nabla)^n f.$$

For example, the Cartesian form of f_{vv} is

$$\begin{aligned} f_{vv} &= \tfrac{1}{||v||^2}\left((f_y \;\; -f_x) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}\right)^2 f \\ &= \tfrac{1}{f_x^2+f_y^2}\left(f_y \tfrac{\partial}{\partial x} - f_x \tfrac{\partial}{\partial y}\right)^2 f \\ &= \tfrac{1}{f_x^2+f_y^2}\left(f_y^2 \tfrac{\partial^2}{\partial x^2} - 2 f_x f_y \tfrac{\partial^2}{\partial x \partial y} + f_x^2 \tfrac{\partial^2}{\partial y^2}\right) f \\ &= \tfrac{f_y^2 f_{xx} - 2 f_x f_y f_{xy} + f_x^2 f_{yy}}{f_x^2+f_y^2}. \end{aligned}$$

[2] Note that we compute the directional derivative along a straight line; the derivative along the v or w vector. We do not compute the derivative along the integral *curves* of the local v or w vectors. The latter form gives distinctly different results. Also, note that in our expressions we take the derivatives f_x and f_y to be constants, so the differential operators affect only f itself.

Edges. In chapter 5 we introduced the gradient $\nabla f(b)$ in a certain point b as a vector that always points in the most uphill direction if we view the image as an intensity landscape. The gradient is also perpendicular to the local isophote. The norm of the gradient $||\nabla f(b)|| = \sqrt{f_x^2(b) + f_y^2(b)}$ is a measure for the edgeness in b.

Another edgeness measure is based on the second derivatives of f: the Laplacian, $\Delta f = f_{xx} + f_{yy}$. In this case, edges are assumed to be located at the zero crossings of the measure. Both the gradient norm and the Laplacian are computed using first or second order partial derivatives of the image f. We can now compute these derivatives in a scaled way, using a certain scale σ. The figures 9.7 and 9.8 show some examples of scaled edgeness images. Note the results of high-scale operators in noisy environments in figure 9.8.

Ridges. If we view an image as an intensity landscape, one of the most characteristic features are the landscape's *ridges*. The definition of a ridge in terms of image derivatives still sparks some debate, apparently because different people usually have different ideas as to what exactly the ridges in an image *are*[3]. In fact, many different definitions can be found, each defining similar yet different ridge-like structures in an image.

One ridge definition that appeals to the intuition is one based on the (v, w) gradient-based coordinate frame: figure 9.9 shows part of an image depicted as an intensity landscape. At a ridge point (1), and two other randomly chosen points, we have drawn the gradient vector w. If we plot the grey values around the ridge point in the v and the w direction (left two plots of the four), then only the v direction plot shows a concave profile. This is because the v direction is the direction crossing the ridge. The same plots at a non-ridge point do not show any significant concavity. This is the basis of our definition for ridgeness: the concavity of the grey-value profile in the v direction. Since the second derivative is a measure for concavity, our ridgeness measure is simply f_{vv}. For the Cartesian form of this measure see the intermezzo above. Figure 9.10 shows some examples of f_{vv} images at different scales. The measure not only detects ridges (low f_{vv} values), but also 'valleys' (high f_{vv} values), see figure 9.11. Figure 9.9 can also be used to define another ridgeness operator: In any non-ridge point, it can be observed that the local gradient is pointing roughly toward the ridge. In a ridge point, the gradient is directed *along* the ridge. This means that if we travel from the left point to the right one via the ridge point, the gradient will turn swiftly as we cross the ridge. The gradient turning speed when traveling in the v direction through an image may therefore be another ridgeness operator. We can measure this turning speed as follows: the orientation of the gradient can be characterized by an angle θ with the positive x-axis, *i.e.*, $\theta = \arctan(f_y/f_x)$. Its rate of change (turning speed of the gradient) equals its first derivative, here taken in the v direction. This results in the expression $\frac{1}{||v||}(v \cdot \nabla)\theta$, which

[3] Unfortunately, this holds true to some extent for most types of image features.

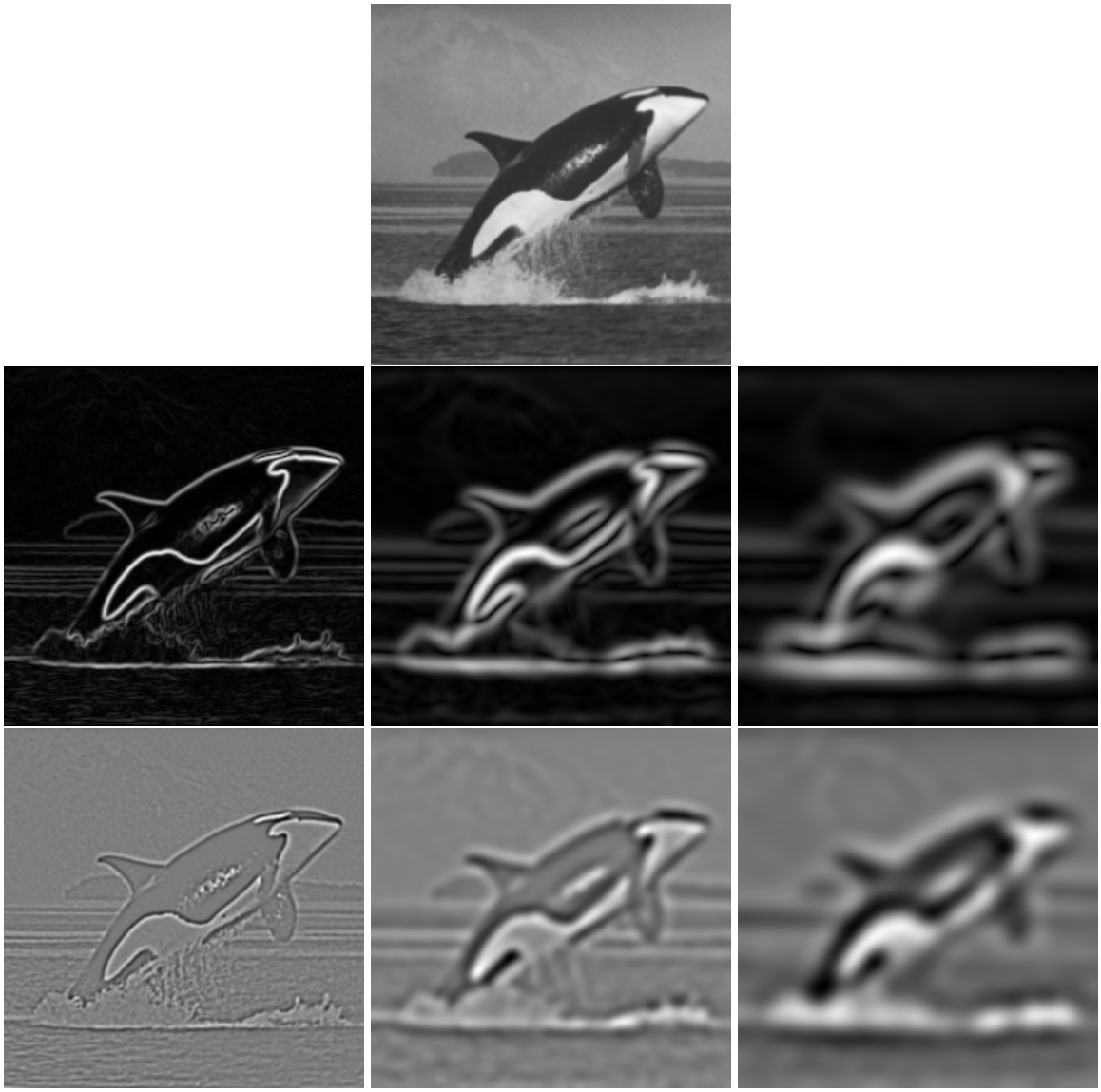


Figure 9.7 Edgeness images at different scales. Top row: original, 256×256 image. Middle row: gradient norm at scales 1, 4 and 8 pixels. Bottom row: Laplacian images at scales 1, 4, and 8 pixels. Note that details vanish from the edgeness images as the scale increases. However, the localization certainty decreases with increasing scale; the 'resolution' of the edge is lowered.

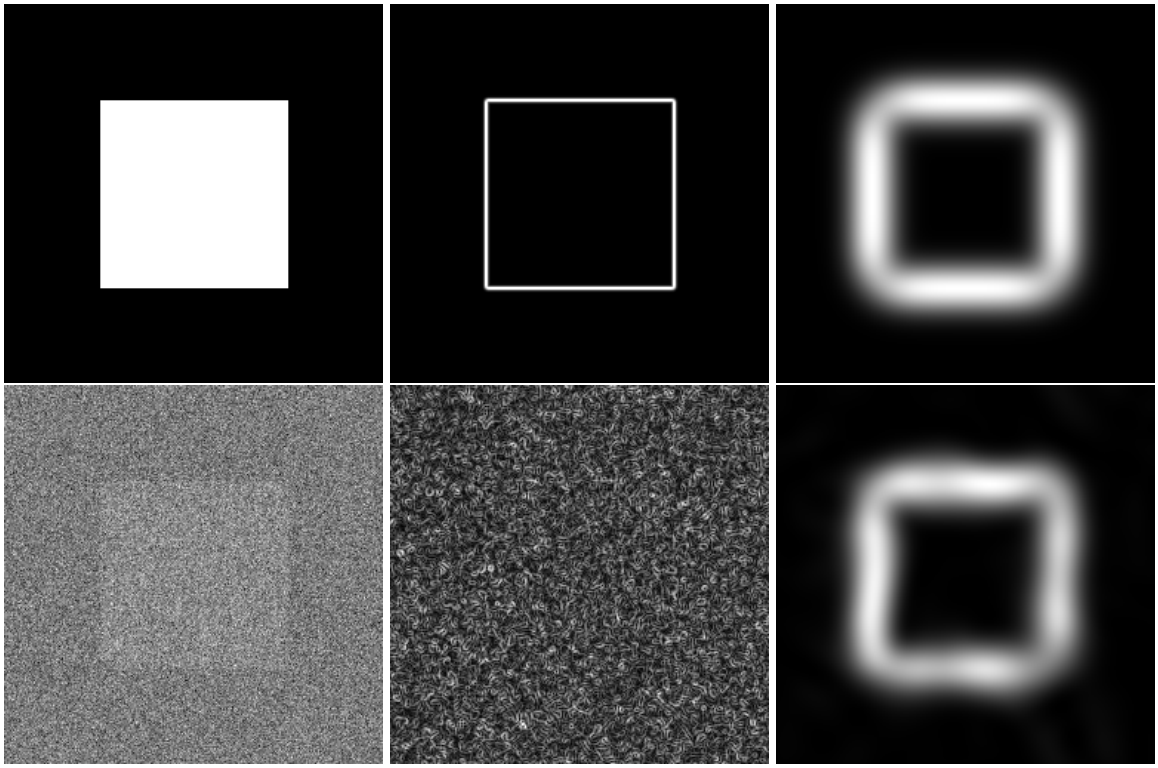


Figure 9.8 Gradient norm edgeness images of a 256×256 artificial image (top left), and of the same image with added noise (bottom left). The middle and right column show the edgeness images at scales of 1 and 16 pixels respectively. Note that only the 16 pixel scale performs well in the noisy image.

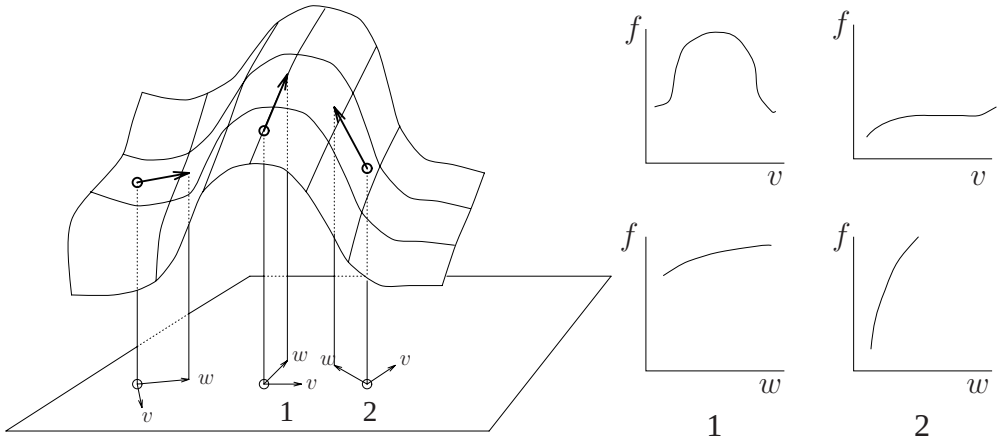


Figure 9.9 The grey-value profiles in the v and w (gradient) directions at a ridge point (1), and a non-ridge point (2). Note that only the v-profile in the ridge point is markedly concave.

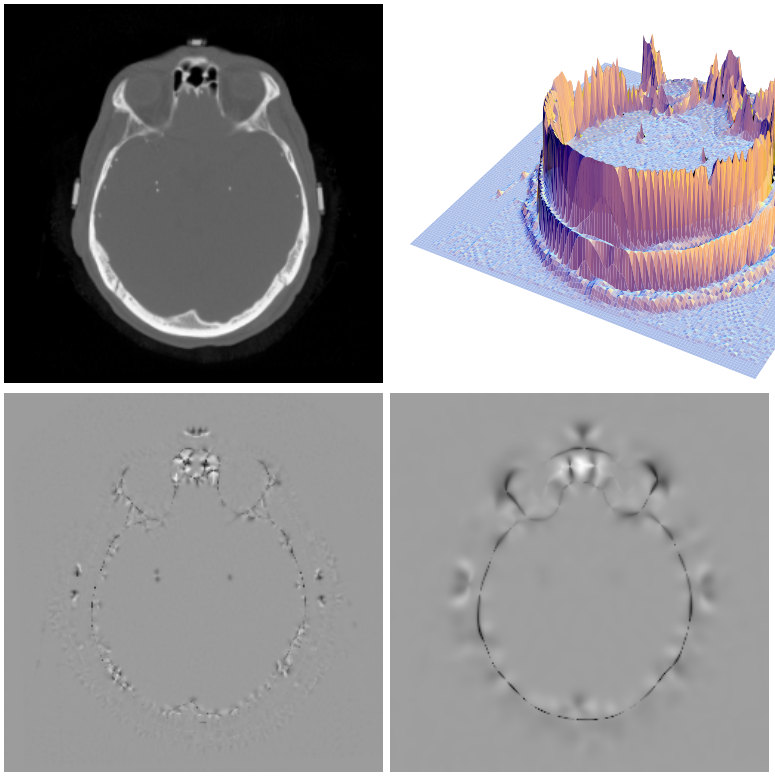


Figure 9.10 The top row shows an original image (256×256 pixels) and its landscape version. The landscape image shows that a roughly circular ridge is present in the image. However, the ridge is locally very jagged, so the ridgeness operator f_{vv} will not show much at a low scale of one pixel (bottom left). At a scale of four pixels, the f_{vv} image nicely shows the ridge curve.

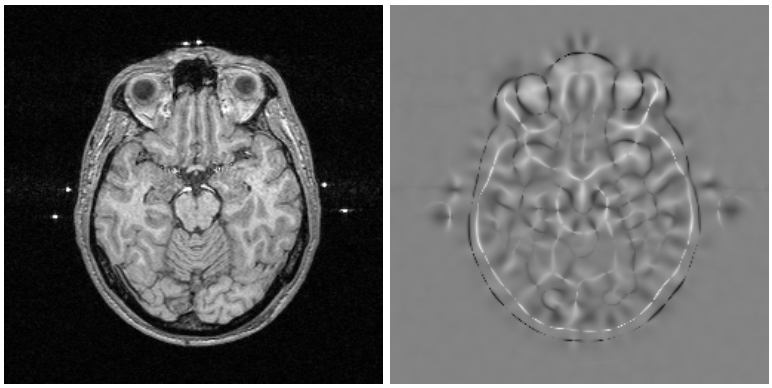


Figure 9.11 Example of the f_{vv} ridgeness operator. Left: original image (256×256 pixels). Right: f_{vv} image at a scale of 4 pixels. Note that ridges in the original form dark curves, whereas valleys from bright curves.

can be worked out in the same manner as before:

$$\frac{1}{||v||}(v \cdot \nabla)\theta = \frac{2f_x f_y f_{xy} - f_y^2 f_{xx} - f_x^2 f_{yy}}{(f_x^2 + f_y^2)^{\frac{3}{2}}}.$$

It turns out this expression is very similar to the f_{vv} ridgeness operator. In fact, it equals $-\frac{f_{vv}}{f_w}$, so the only difference is the minus sign and a normalization with respect to the gradient magnitude f_w. This latter aspect makes that the $\frac{f_{vv}}{f_w}$ responds more in low-contrast image areas than f_{vv}. The measure $\frac{f_{vv}}{f_w}$ is also known in literature as the *isophote curvature*. Figure 9.12 shows some examples of $-\frac{f_{vv}}{f_w}$ images.

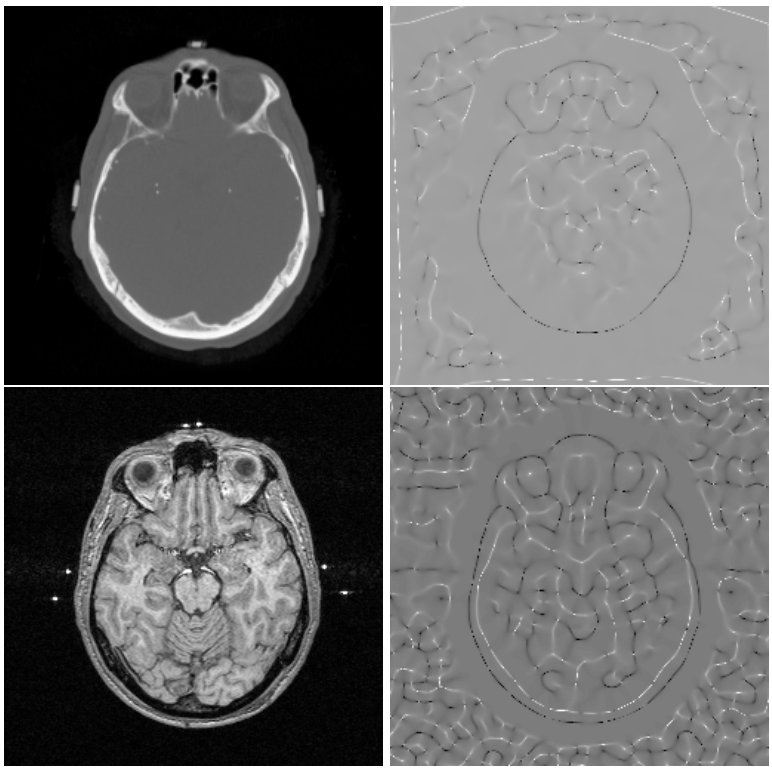


Figure 9.12 Examples of the $\frac{f_{vv}}{f_w}$ ridgeness operator. Left column: original 256×256 images. Right column: $\frac{f_{vv}}{f_w}$ images at a scale of four pixels. Note that the operator also responds in low contrast image areas.

Other differential geometric structures. It would be beyond the scope of this book to treat in detail other differential geometric structures. It is however illustrative to show some examples of other operators, to give some idea of their capabilities in relation to image processing. Table 9.1 lists a number of operators (all invariant under rotation and translation). Note that names for expressions (such as *cornerness*) are descriptive at best, and should not be taken in too literal a sense. The figures 9.13 through 9.16 show some example images.

Name (expression)	Cartesian form
Edgeness, f_w	$\sqrt{f_x^2 + f_y^2}$
Ridgeness, f_{vv}	$\frac{f_y^2 f_{xx} - 2 f_x f_y f_{xy} + f_x^2 f_{yy}}{f_x^2 + f_y^2}$
Isophote curvature; ridgeness, $-\frac{f_{vv}}{f_w}$	$-\frac{f_y^2 f_{xx} - 2 f_x f_y f_{xy} + f_x^2 f_{yy}}{(f_x^2 + f_y^2)^{\frac{3}{2}}}$
Cornerness, $f_{vv} f_w^2$	$f_y^2 f_{xx} - 2 f_x f_y f_{xy} + f_x^2 f_{yy}$
Flowline curvature, $-\frac{f_{vw}}{f_w}$	$\frac{f_x f_y (f_{yy} - f_{xx}) + f_{xy}(f_x^2 - f_y^2)}{(f_x^2 + f_y^2)^{\frac{3}{2}}}$
Isophote density, f_{ww} (or $\frac{f_{ww}}{f_w}$)	$\frac{f_{xx} f_x^2 + 2 f_{xy} f_x f_y + f_{yy} f_y^2}{f_x^2 + f_y^2}$
Umbilicity	$\frac{2(f_{xx} f_{yy} - f_{xy}^2)}{f_{xx}^2 + 2 f_{xy}^2 + f_{yy}^2}$
Unflatness	$f_{xx}^2 + 2 f_{xy}^2 + f_{yy}^2$
Checkerboard detector	$-(f_{xxxx} f_{yyyy} - 4 f_{xxxy} f_{xyyy} + 3 f_{xxyy} f_{xxyy})^3 + 27(-f_{xxxy}(f_{xxxy} f_{yyyy} - f_{xyyy} f_{xxyy}) + f_{xxyy}(f_{xxxy} f_{xyyy} - f_{xxyy} f_{xxyy}) + f_{xxxx}(f_{xxyy} f_{yyyy} - f_{xyyy} f_{xyyy}))^2$

Table 9.1 Examples of differential invariants.

9.2 Scale space and diffusion*

The physical processes of diffusion and heat conduction offer a new view of scale space. At a first glance, these processes and scale space appear to be unrelated, but consider figure 9.17. Suppose the left graph represents the temperature as a function of length along a metal rod. After a certain amount of time, the temperature distribution will have evolved to the state represented by the right graph. As you may have guessed from the shape of the graphs, the right graph is nothing but the left graph convolved with a Gaussian. The temperature distribution at any time after the initial state can be computed by convolving the initial distribution with a Gaussian, where the width σ is directly related to the time a certain state occurs. Figure 9.18 shows some evolution states. The collection of all evolution states equals the scale space of the original state.

The physical process of heat conduction is usually modeled by the following partial differential equation:

$$f_t = \alpha^2 f_{xx},$$

where $f = f(x, t)$ represents the temperature at position x at time t, and α is a material-dependent constant. This equation is known as the *heat* or *diffusion* equation (it can be used as a model for either process). It models how the temperature distribution evolves: the change in time (f_t) depends only on the current temperature distribution, and some

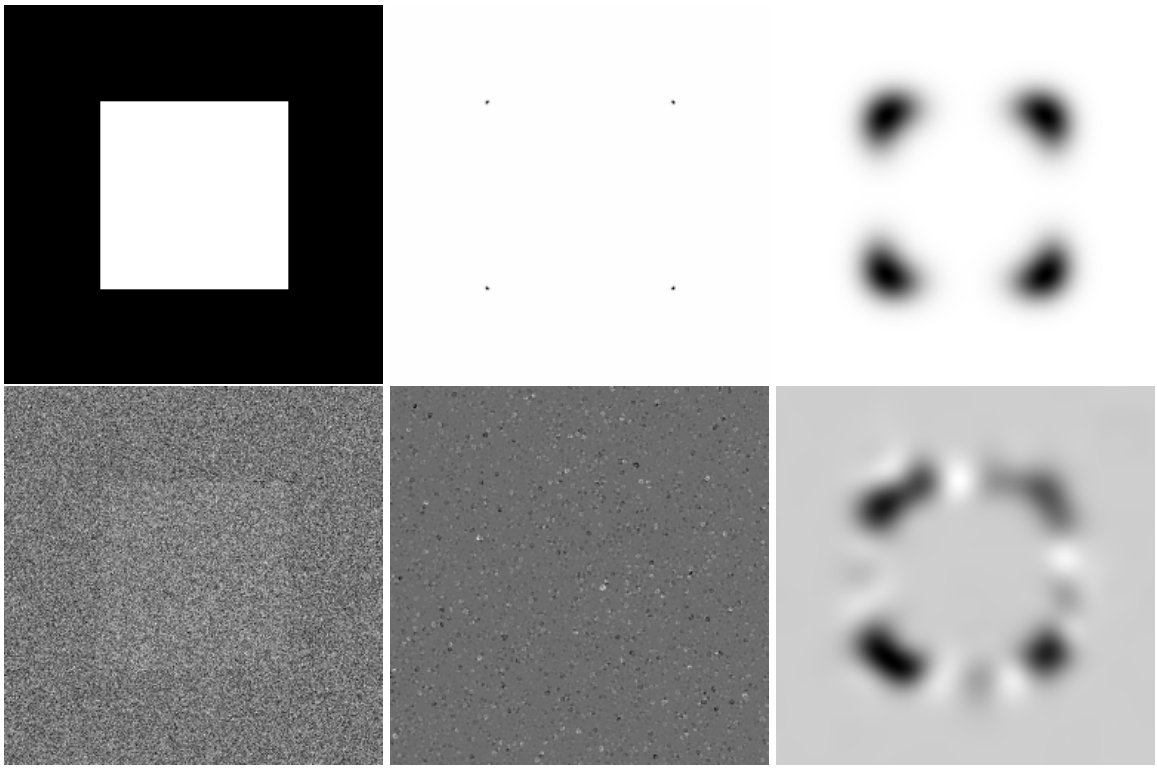


Figure 9.13 Top row: original (256×256) image, and two cornerness images at scales of 1 and 16 pixels. Bottom row: same original image, with added noise, and the cornerness images at the same two scales. Note that only the large scale can cope with the (small scale) noise.

nifty modeling that we won't report here shows that the change in time is proportional to the second spatial derivative of f, as the equation shows. The two-dimensional form of this equation is

$$f_t = \alpha^2 \Delta f,$$

where Δf equals the Laplacian $\Delta f = \nabla \cdot (\nabla f) = f_{xx} + f_{yy}$. If we assume the function f has an infinite spatial domain, it can be shown that a solution $f(x, \tau)$ at a certain time τ can be found by convolving the initial state $f(x, 0)$ with a Gaussian function with width $\sigma = \sqrt{2\alpha^2 t}$.

In terms of images, this means that we can regard a scaled image $f_\sigma(x, y)$ as the solution of the heat equation at a time related to σ, where the original image $f = f_0(x, y)$ is the initial 'temperature distribution', if we think of a grey value as a temperature.

The heat conduction model can be extended to include some common phenomena. For example, in practice we may need to model the heat conduction in situations where

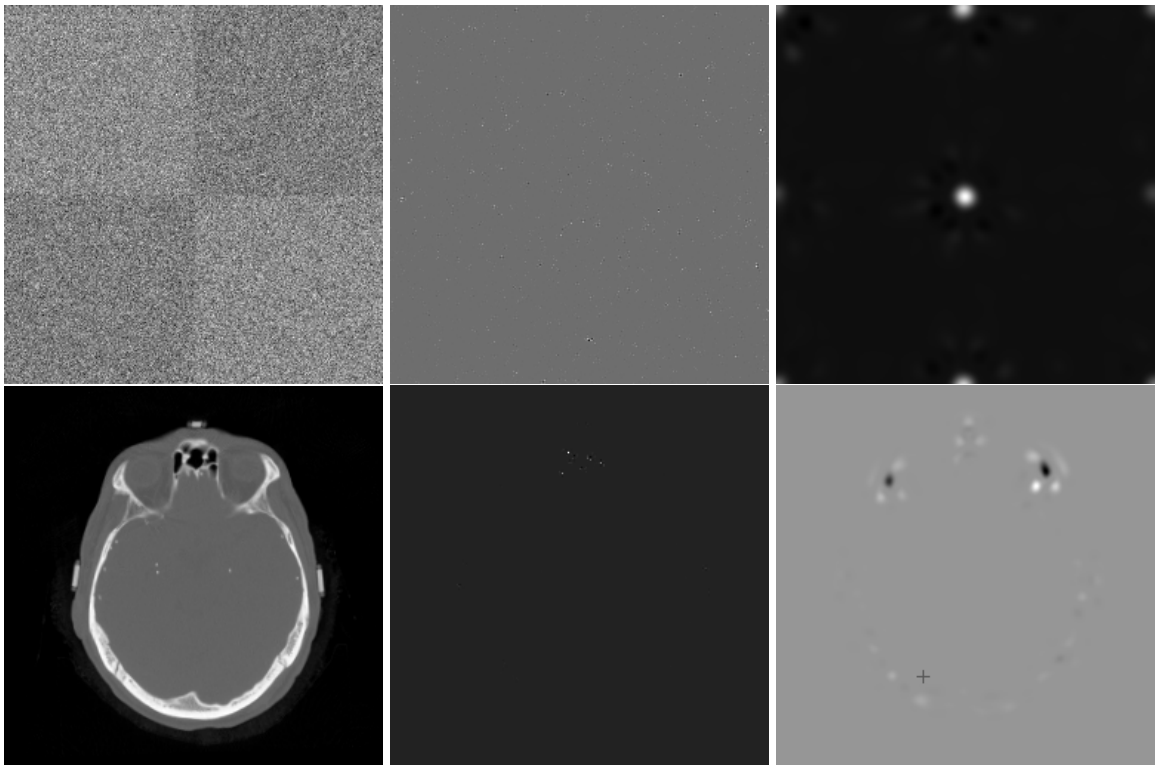


Figure 9.14 Top row: original noisy (256×256) image of part of a checkerboard, and two checkerboard detector images at scales of 1 and 16 pixels. Bottom row: 256×256 CT image and checkerboard detector images at scales of 1 and 8 pixels

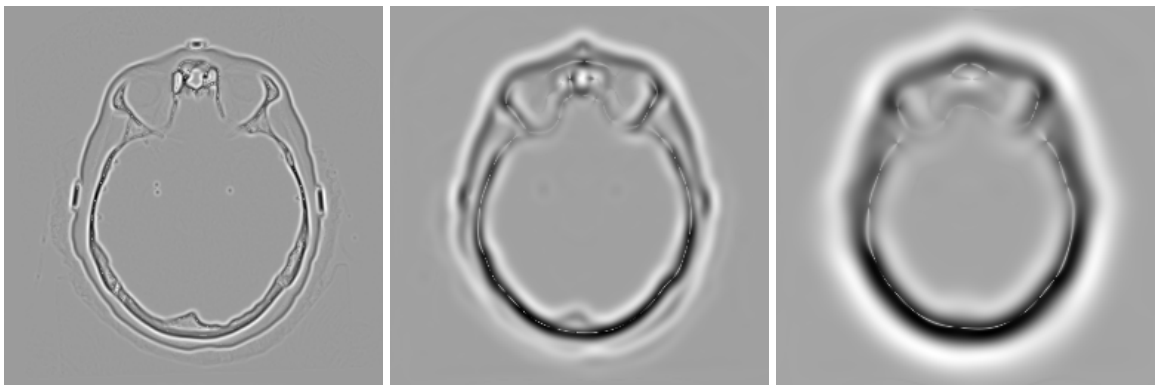


Figure 9.15 f_{ww} isophote density images at scales of 1, 4, and 8 pixels of the CT image in the previous figure.

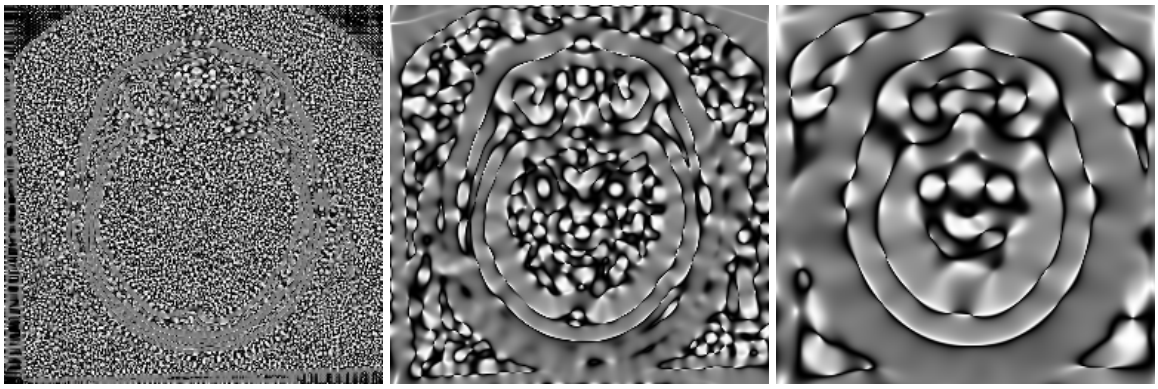


Figure 9.16 Umbilicity images at scales of 1, 4, and 8 pixels of the CT image used in the previous figures. The umbilicity operator distinguishes elliptical (positive value) and hyperbolic (negative value) areas of the image intensity landscape.

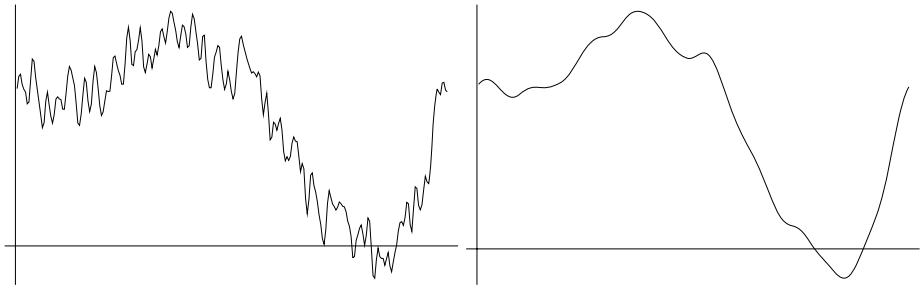


Figure 9.17 Scale space and the physical process of heat conduction. Left: temperature as a function of length of a metal rod. Right: the same function, after some time has elapsed and normal heat conduction has taken place. These functions are related to scale space: the right graph is in fact the left graph convolved with a Gaussian.

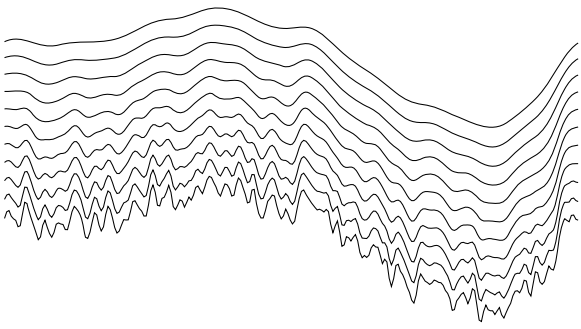


Figure 9.18 Some temperature evolution states of the function in the previous figure (plotted vertically above each other for clarity). The space of the temperature evolution equals the scale space of the original state.

different materials –with different heat conduction speeds– are involved. An extended heat equation that can handle heat conduction that changes in space and time is

$$f_t = \nabla \cdot (c \nabla f),$$

where $c = c(x, y, t)$ is a measure for the local heat conduction. If c is very small at a certain location, then almost no heat will flow across that location when time passes. This is where the analogy of scale space vs. heat conduction becomes interesting: it is often a drawback of scale space is that edges of objects become very blurred (diffused!) at high scales. Sometimes, it is useful if we could somehow preserve the edges of objects at a scale σ or larger. We can do this by making $c(x, y, t)$ small if (x, y) is located at an edge. To determine if (x, y) is located at an edge, we can use the gradient magnitude f_w (computed at the right scale) as an edgeness measure. A suitable formula for c (one of many, really) is

$$c = \frac{1}{1 + \left(\frac{f_w}{K}\right)^2},$$

where K is a positive constant. The heat equation with this c substituted is known as the *Perona-Malik* equation. Applying this equation to an image creates a totally different scale space than when $c = 1$, as in the 'normal' scale space.

We can create other scale spaces by modifying the c function so that diffusion around some type of geometrical structures (edges, ridges, *etc.*) is encouraged or discouraged. The generating equations can then, *e.g.*, be $f_t = f_w$, $f_t = f_{vv}$, $f_t = f_{vv}^{1/3} f_w^{2/3}$, *etc.* , to respectively encourage diffusion at edges, at ridges (diffuse in the center of objects rather than at the edges), and at corners[4]. Figure 9.19 shows some examples. Figure 9.20 shows a noise reduction application.

While computing a scaled image in the Gaussian scale space is relatively easy –it requires only convolution of the original image with a Gaussian– images in the other scale spaces mentioned cannot usually be computed directly, because the necessary convolution kernel is different in each location of the image. In practice, scaled images are computed by letting the underlying differential equation evolve to the desired scale (time), using a numerical method (such as Euler's method) to approximate the solution of the equation.

[4]Evolution of an image according to these three equations are also known as normal flow, Euclidean shortening flow, and affine shortening flow respectively.

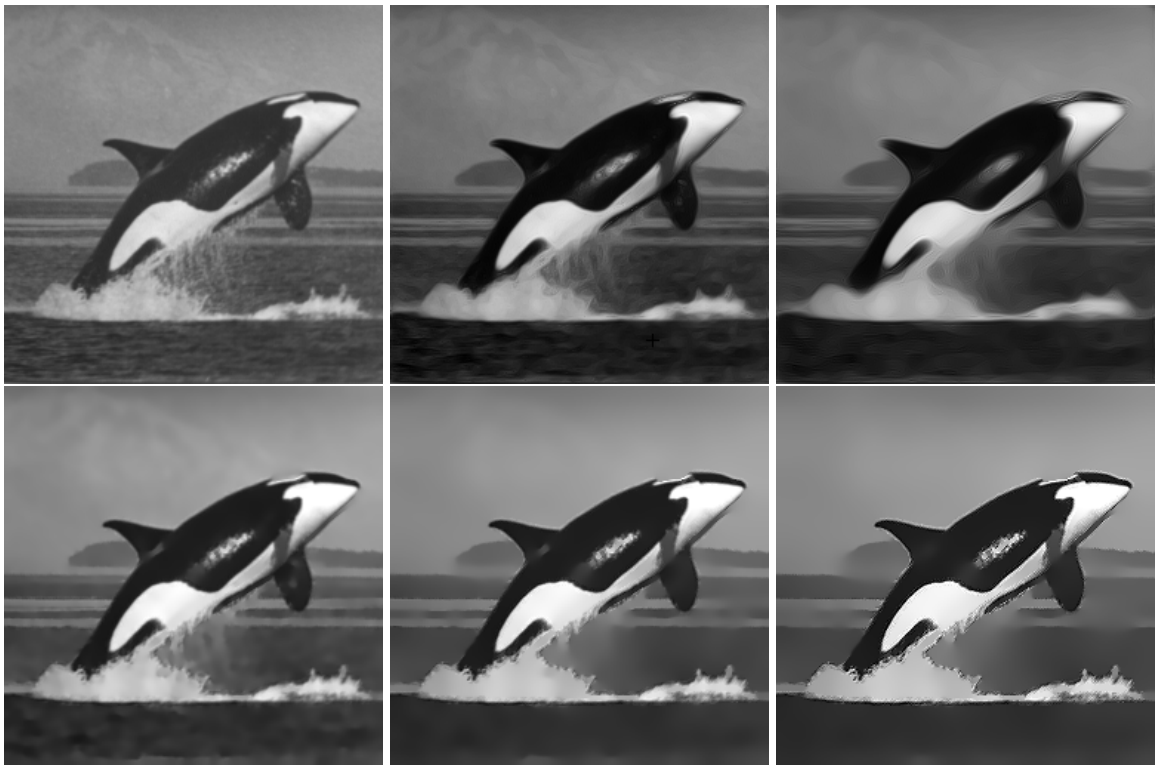


Figure 9.19 Top row: original image and two scaled versions from a scale space generated by the equation $f_t = f_{vv}$. Bottom row: three scaled images from a scale space generated by the Perona-Malik equation.

9.3 The resolution pyramid

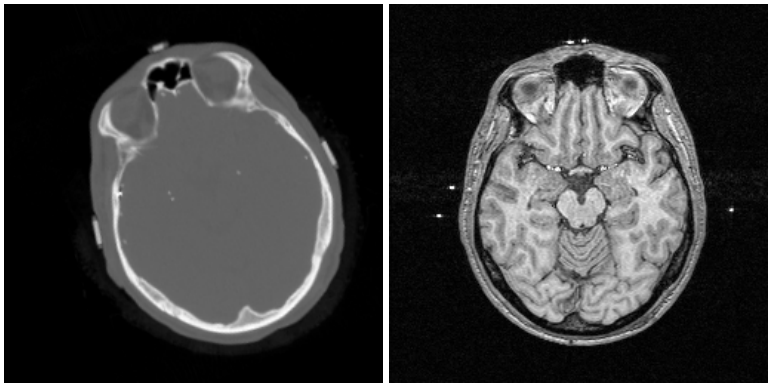
At the start of this chapter, we mentioned a simple method for lowering the resolution of an image: divide the image in regular blocks of pixels (say 2×2 pixels each), and merge the pixels in each block to a single new pixel. This merging can be done by, *e.g.*, averaging the pixel values, taking (*e.g.*) the top-left pixel value, or taking the maximum or minimum pixel value. We can apply the same operation to the results, and again to its results, *etc.* creating a series of images –each image with a lower resolution than the image before it– called a *resolution pyramid*. Figure 9.21 shows an example.

This method of resolution lowering lacks most of the useful mathematical properties that scale space has, and it is not straightforward how to lower the resolution by a factor that is not integer. Nevertheless, the resolution pyramid is extremely useful in its own right in many different applications. This is because an image in the pyramid retains much of the global structure of the original image (exactly how much depending on the pyramid level), while taking less memory bytes to store. In general, processing of a pyramid image is significantly faster than processing the original image. A common

way to employ the pyramid is to do fast and coarse processing on the top level image (the lowest resolution image), and use its results to guide processing at the level below it, continuing until the original image (the bottom level of the pyramid) is reached. For an example, we show the use of a pyramid in an image registration application.

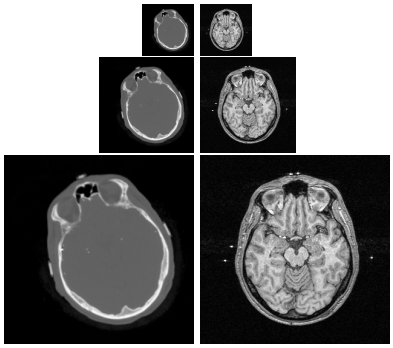
Example: image registration using a multiresolution pyramid

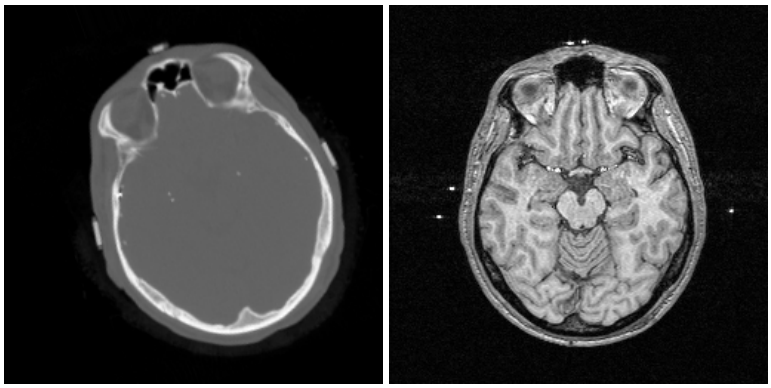
Suppose we have two 256×256 images that need to be brought into registration by rotation and translation of one of the two:



We want the registration error to be no more than one pixel size anywhere in the image. The straightforward 'brute force' approach to find the correct registration is to examine all possible transformations, computing a suitable registration measure in each case, in finally picking the transformation for which the measure is maximum. Unfortunately, this approach results in over 60 million configurations to check, and so is not usually a feasible approach to take.

A better approach is to build a (*e.g.*) four level pyramid:





The top level images have a 32×32 resolution, and the number of possible transformations that are no more than one pixel length apart is now low enough for each to be examined separately. Each examination in itself is also much faster than examining the original, because the number of pixels is reduced by a factor of 64.

The best transformation found at the top pyramid level will be too coarse to be very accurate at the next pyramid level, but it can function as a suitable starting position at this level. At this level, we again examine a number of transformations and pick out the best one, but keep the number of transformations down by examining only a suitable range of transformations around the starting position. In this fashion, we continue down the pyramid until we reach the original image, where we can find the optimal transformation by examining only a relatively small number of transformations around the starting position.

The possibility of a coarse-to-fine strategy such as the one described above is not only an asset of the resolution pyramid, but also of any scale space. The implementation and use of the strategy is most straightforward in the case of the resolution pyramid. The pyramid, however, lacks many of the useful mathematical properties of scale space that enable a more robust approach using scale space techniques instead of the pyramid in many applications.

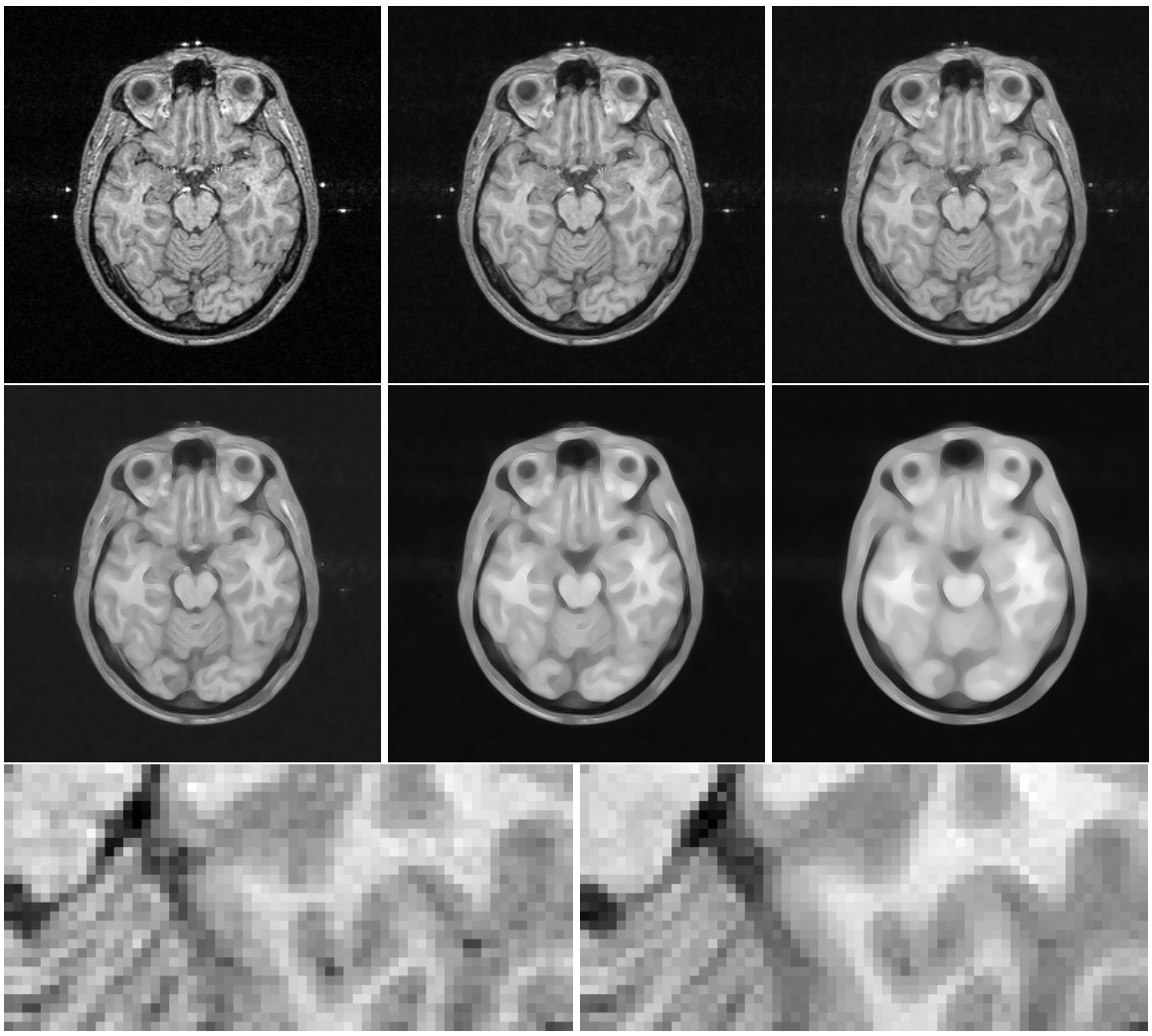


Figure 9.20 Scale space generated from the top left image using the equation $f_t = f_{vv}$. A zoom of the orginal and top right image shows that a moderate scaling of an image removes noise without touching essential image structures much.

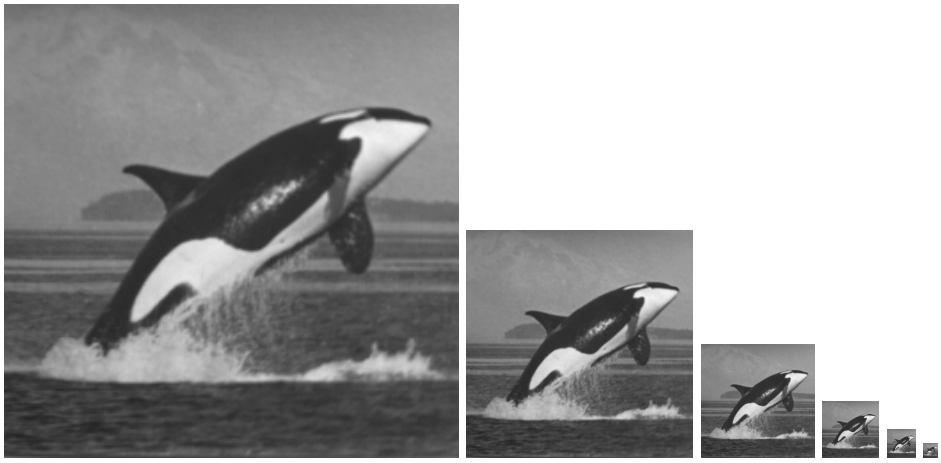


Figure 9.21 Example of a resolution pyramid. The original 256×256 image is on the left. The next image is formed by averaging each block of 4 pixels into a single pixel, forming a 128×128 image. Each following image is formed in the same manner; the final image has a resolution of 8×8 pixels.