Image Differentiation and Image Pyramids

Carlo Tomasi

February 4, 2021

This note discusses image *differentiation*, which is useful for edge detection, image motion analysis, and other purposes; and the analysis of images at multiple scales through image *pyramids*.

1 Image Differentiation

Many image operations, including edge detection and motion analysis in video, require computing the derivatives of image intensity with respect to the horizontal (x) or vertical (y) direction. This stands to reason: Edges are curves on the image plane across which image intensity changes rapidly (the derivative is large); Motion is computed by comparing changes over time (time derivative) with changes over image space (spatial derivatives).

However, since images are defined on discrete domains, "image differentiation" is undefined. To give this notion some meaning, we think of images as sampled versions of continuous[1] distributions of brightness. This indeed they are: the distribution of intensities on the sensor is continuous, and the sensor integrates this distribution over the active area of each pixel and then samples the result at the pixel locations.

> "Differentiating an image" means to compute the samples of the derivative of the continuous distribution of brightness values on the sensor surface.

Since we only have access to the digital image, differentiation involves, at least conceptually, undoing sampling (that is, computing a continuous image from a discrete one), differentiating, and sampling again. The process of undoing sampling is called *interpolation* if the continuous function is required to pass through the samples, and *fitting* otherwise. Figure 1 shows a conceptual bloc diagram for the computation of the image derivative in the x (or c) direction.

Interpolation or fitting can be expressed as a hybrid-domain convolution, that is, a convolution of a discrete image with a continuous kernel. This is formally analogous to a discrete convolution, but has a very different meaning:[2]

$$C(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) P(x - j, y - i)$$

[1] "Continuous" here refers to the domain: we are talking about functions of real valued variables.

[2] For simplicity, we assume that the x and y axes have the same origin and direction as the j and i axes: to the right and down, respectively. Functions of discrete variables have the row argument first, and the column argument second. Functions of continuous variables have the horizontally varying argument first, and the vertically varying argument second.

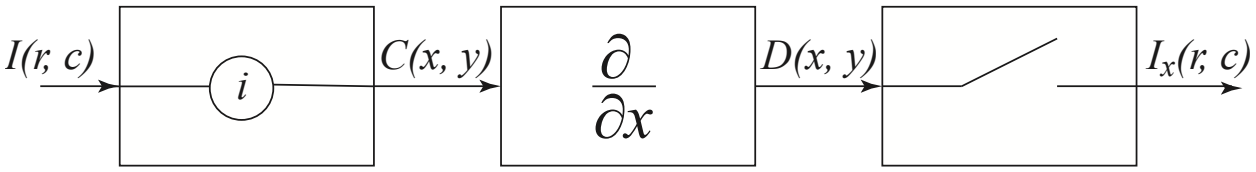


Figure 1: Conceptual bloc diagram for the computation of the derivative of image $I(r, c)$ in the horizontal (c) direction. The first block interpolates or fits the samples from a discrete to a continuous domain. The second computes the partial derivative in the horizontal (x) direction. The third block samples from a continuous domain back to a discrete one.

where x, y are real variables and $P(x, y)$ is called the *interpolation* (or *fitting*) *kernel*. The key difference with respect to the convolution we are familiar with is that now P is a function of real variables, $P : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, rather than a function of integer variables, $H : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. As a consequence, the output C is now also a function of real variables.

This hybrid convolution seems hard to implement: how can we even represent the output, a function of two real variables, on a computer? However, the chain of the three operations depicted in Figure 1 goes from discrete-domain to discrete-domain. As we now show, the transformation performed by the whole chain (but not its individual links) can be implemented easily and without reference to continuous-domain variables.

Since both interpolation (or fitting) and differentiation are linear, instead of interpolating (or fitting) the image and then differentiating we can interpolate (or fit) the image with the derivative of the interpolation (or fitting) function. Formally,

$$\begin{aligned} D(x, y) &= \frac{\partial C}{\partial x}(x, y) = \frac{\partial}{\partial x} \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) P(x - j, y - i) \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) P_x(x - j, y - i) \end{aligned}$$

where

$$P_x(x, y) = \frac{\partial P}{\partial x}(x, y)$$

is the partial derivative of $P(x, y)$ with respect to x. This change in the position of the differentiation operator is crucial, as it changes the problem from one of differentiating an arbitrary signal $C(x, y)$ to one of differentiating a known, simple kernel function $P(x, y)$.

Finally, we need to sample the result $D(x, y)$ at the grid points (r, c) to obtain a discrete image $I_c(r, c)$. This yields the final, *discrete* convolution that computes the derivative of the underlying continuous image with respect to the horizontal variable:[3]

$$J(r, c) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) P_x(c - j, r - i) \; .$$

Note that all continuous variables have disappeared from this expression: this is a standard, discrete-domain convolution, so we can implement this on a computer without difficulty. In other

[3]Again, c and r are assumed to have the same origin and orientations as x and y.

words, while some of the steps in the chain of operations in Figure 1 are potentially problematic in terms of implementation, the transformation performed by the chain as a whole is a simple, discrete convolution of the type we are familiar with.

The correct choice for the function $P(x, y)$ is outside the scope of this course, but it turns out that the truncated Gaussian function is adequate, as it smooths the data (through fitting, not interpolation) and therefore improves the signal-to-noise ratio. We therefore let $\tilde{P}$ be the (unnormalized) Gaussian function of two continuous variables x and y:

$$\tilde{P}(x, y) = G(x, y)$$

and $\tilde{P}_x$, $\tilde{P}_y$ its partial derivatives with respect to x and y (Figure 2). We then sample $\tilde{P}_x$ and $\tilde{P}_y$ over a suitable interval $[-n, n]$ of the integers and normalize them by requiring that their response to a linear ramp (a function that is linearly increasing in one direction and constant in the other) yield the slope of the ramp itself.

The ramp is the simplest function with nonzero derivative, and we use it as a "probing function" since we know the desired derivative. This normalization ensures that derivatives are not scaled up or down: The derivative of x is 1, not just some constant. A unit-slope, discrete ramp in the j direction is represented by

$$u(i, j) = j$$

and we want to find a constant u_0 such that

$$u_0 \sum_{i=-n}^{n} \sum_{j=-n}^{n} u(c - j, r - i)\tilde{P}_x(j, i) = 1$$

for all r, c so that

$$P_x(x, y) = u_0 \, \tilde{P}_x(x, y) \quad \text{and} \quad P_y(x, y) = u_0 \, \tilde{P}_y(x, y) \; .$$

The derivative of $u(i, j)$ is the same everywhere, so we only need to test the output of the convolution at one point. In particular, for $r = c = 0$ (the simplest choice) we obtain

$$u_0 = -\frac{1}{\sum_{i=-n}^{n} \sum_{j=-n}^{n} j\tilde{P}_x(j, i)} \; . \tag{1}$$

Since the partial derivative $G_x(x, y)$ of the Gaussian function with respect to x is negative for positive x (or j), the product $jG_x(j, i)$ is either zero or positive and the constant u_0 is positive. By symmetry, the same constant normalizes G_y.

Of course, since the two-dimensional Gaussian function is separable, so are its two partial derivatives:

$$I_c(r, c) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} I(r - i, c - j)G_x(j, i) = \sum_{j=-n}^{n} d(j) \sum_{i=-n}^{n} I((r - i, c - j)g(i)$$

where

$$d(x) = \frac{dg}{dx} = -\frac{x}{\sigma^2} g(x)$$

is the ordinary derivative of the one-dimensional Gaussian function $g(x)$. A similar expression holds for $I_r(r, c)$ (see below).

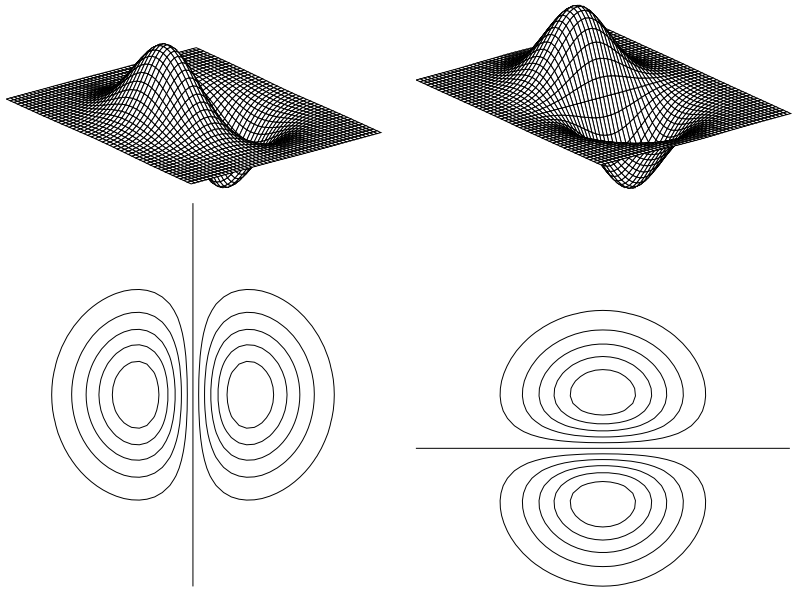


Figure 2: The partial derivatives of a Gaussian function with respect to x (left) and y (right) represented by plots (top) and isocontours (bottom). In the isocontour plots, the x variable points horizontally to the right, and the y variable points vertically down.

Thus, the partial derivative of an image in the x direction is computed by convolving[4] with $d(j)$ and $g(i)$. The partial derivative in the y direction is obtained by convolving with $d(i)$ and $g(j)$. In both cases, the order in which the two one-dimensional convolutions are performed is immaterial, because convolution commutes:

$$\begin{aligned} I_c(r,c) &= \sum_{i=-n}^{n} g(i) \sum_{j=-n}^{n} I(r-i,c-j)d(j) = \sum_{j=-n}^{n} d(j) \sum_{i=-n}^{n} I(r-i,c-j)g(i) \\ I_r(r,c) &= \sum_{i=-n}^{n} d(i) \sum_{j=-n}^{n} I(r-i,c-j)g(j) = \sum_{j=-n}^{n} g(j) \sum_{i=-n}^{n} I(r-i,c-j)d(i) \ . \end{aligned}$$

Normalization can also be done separately: the one-dimensional Gaussian g is normalized as shown in a previous note, and the one-dimensional Gaussian derivative d is normalized by the one-dimensional equivalent of equation (1):

$$\begin{aligned} \tilde{d}(u) &= u e^{-\frac{1}{2}\left(\frac{u}{\sigma}\right)^2} \\ k_d &= \frac{1}{\sum_{v=-n}^{n} v\tilde{d}(v)} \\ d(u) &= -k_d \tilde{d}(u) \ . \end{aligned}$$

[4]Consistently with our definition of the reference axes, functions of j are row vectors, and functions of i are column vectors.

We can summarize this discussion as follows.

The "derivatives" $I_c(r,c)$ and $I_r(r,c)$ of an image $I(r,c)$ in the horizontal (to the right) and vertical (down) direction, respectively, are approximately the samples of the derivative of the continuous distribution of brightness values on the sensor surface. The images I_c and I_r can be computed by the following convolutions:

$$\begin{aligned} I_c(r,c) &= \sum_{i=-n}^{n} g(r-i) \sum_{j=-n}^{n} I(i,j)d(c-j) = \sum_{j=-n}^{n} d(c-j) \sum_{i=-n}^{n} I(i,j)g(r-i) \\ I_r(r,c) &= \sum_{i=-n}^{n} d(r-i) \sum_{j=-n}^{n} I(i,j)g(c-j) = \sum_{j=-n}^{n} g(c-j) \sum_{i=-n}^{n} I(i,j)d(r-i) \ . \end{aligned}$$

In these expressions,

$$g(u) = k_g \tilde{g}(u) \quad \text{where} \quad \tilde{g}(u) = e^{-\frac{1}{2}\left(\frac{u}{\sigma}\right)^2} \quad \text{and} \quad k_g = \frac{1}{\sum_{v=-n}^{n} \tilde{g}(v)}$$

and

$$d(u) = k_d \tilde{d}(u) \quad \text{where} \quad \tilde{d}(u) = u e^{-\frac{1}{2}\left(\frac{u}{\sigma}\right)^2} \quad \text{and} \quad k_d = -\frac{1}{\sum_{v=-n}^{n} v\tilde{d}(v)} \ .$$

The constant σ determines the amount of smoothing performed during differentiation: the greater σ, the more smoothing occurs. The integer n is proportional to σ, so that the effect of truncating the Gaussian function is independent of σ.

The Image Gradient

A value of the partial derivative I_x and one of I_y can be computed at every image position[5] (x,y), and these values can be collected into two images. Two sample images are shown in Figure 3 (c) and (d).

A different view of a detail of these two images is shown in Figure 4 in the form of a *quiver diagram*. For each pixel (x,y), this diagram shows a vector with components $I_x(x,y)$ and $I_y(x,y)$. The diagram is shown only for a detail of the eye in Figure 3 to make the vectors visible.

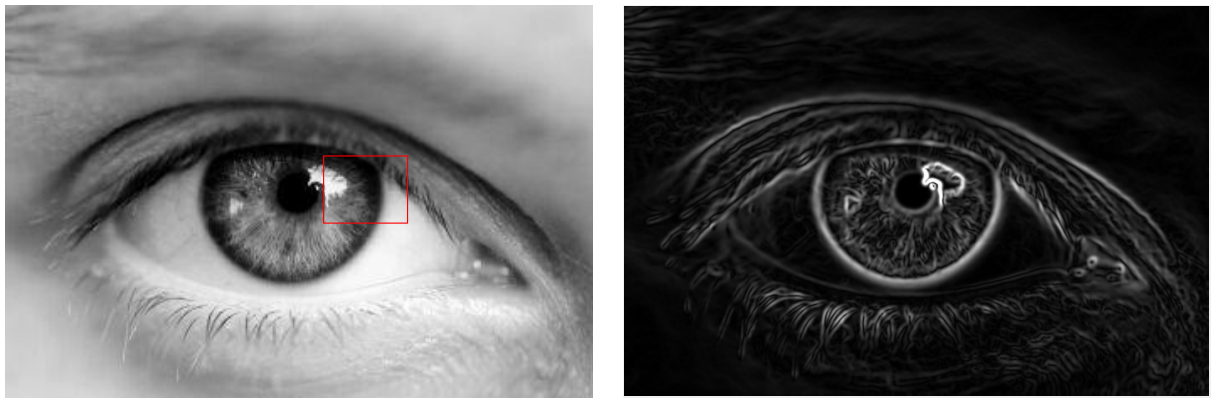
The two components $I_x(x,y)$ and $I_y(x,y)$ considered together as a vector at each pixel form the *gradient* of the image,

$$\mathbf{g}(x,y) = (I_x(x,y), I_y(x,y))^T \ .$$

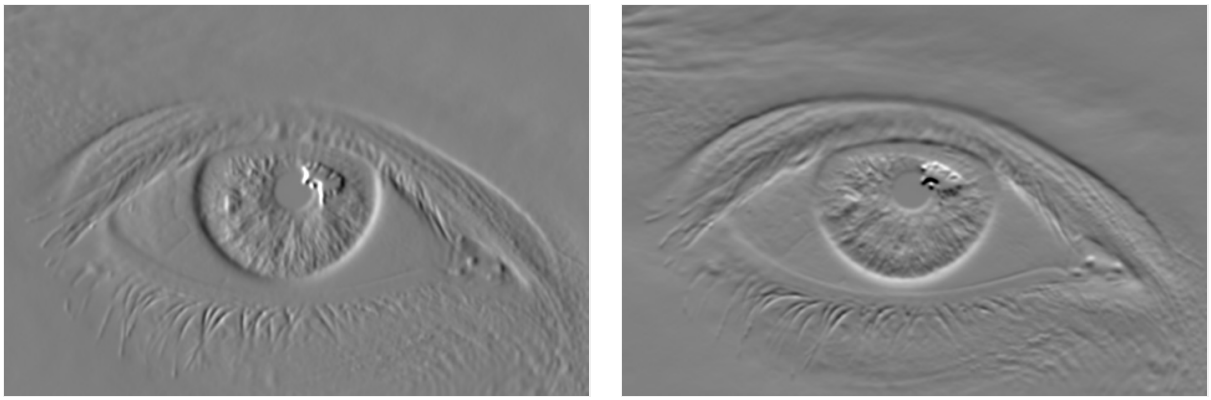
The gradient vector at pixel (x,y) points in the direction of greatest change in I, from darker to lighter. The magnitude

$$g(x,y) = \|\mathbf{g}(x,y)\| = \sqrt{I_x^2(x,y) + I_y^2(x,y)}$$

[5] We assume to pad images by replication, rather than with zeros, beyond their boundaries to prevent large spurious derivatives there.



(a) (b)



(c) (d)

Figure 3: (a) Image of an eye. See Figure 4 for a different view of the detail in the box. (b) The gradient magnitude of the image in (a). Black is zero, white is large. (c), (d) The partial derivatives in the horizontal (c) and vertical (d) direction. Gray is zero, black is large negative, white is large positive. Recall that a positive value of y is downwards.

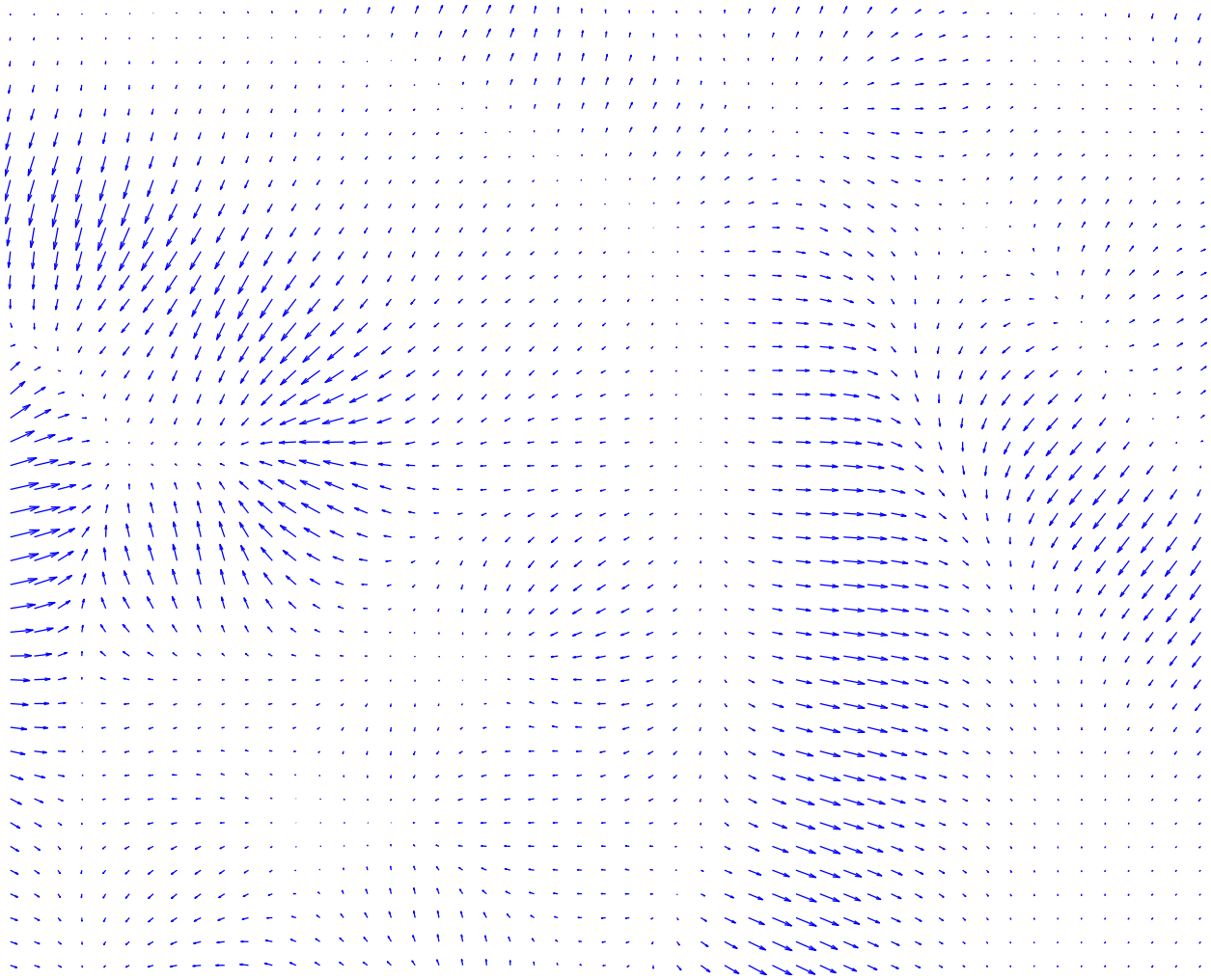


Figure 4: Quiver diagram of the gradient in the detail from the box of Figure 3 (a). Note the long arrows pointing towards the bright glint in the eye, and those pointing from the pupil and from the eyelid towards the eye white.

of the gradient is the local amount of change in the direction of the gradient, measured in gray levels per pixel. The *(total) derivative* of image intensity I along a given direction with unit vector $\mathbf{u}$ is the rate of change of I in the direction of $\mathbf{u} = (u, v)$. This can be computed from the gradient by noticing that the coordinates $\mathbf{p} = (x, y)^T$ of a point on the line through $\mathbf{p}_0 = (x_0, y_0)^T$ and along $\mathbf{u}$ are

$$\mathbf{p} = \mathbf{p}_0 + t\mathbf{u} \quad \text{that is,} \quad \begin{array}{l} x = x_0 + tu \\ y = y_0 + tv \end{array} ,$$

so that the chain rule for differentiation yields

$$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial t} = I_x u + I_y v \; .$$

In other words, the derivative of I in the direction of the unit vector $\mathbf{u}$ is the projection of the gradient onto $\mathbf{u}$:

$$\frac{dI}{dt} = \mathbf{g}^T \mathbf{u} \; . \tag{2}$$

2 The Gaussian Image Pyramid

It is often useful to analyze an image $I(\mathbf{x})$ at different scales. One can then form a stack of images obtained by repeatedly blurring the input image:

$$\begin{array}{rcl} B_0 & = & I \\ B_\ell & = & B_{\ell-1} * S_\sigma \quad \text{for} \quad \ell = 1, \ldots, L \end{array} \tag{3}$$

where $S_\sigma(\mathbf{x})$ is a smoothing kernel, typically a Gaussian with width σ, and the symbol '$*$' denotes convolution. The larger σ, the more high-frequency information (that is, fine detail) is suppressed at every level of smoothing, and analysis of B_ℓ reveals finer or coarser structures in the image depending on the value of the *level* ℓ. Figure 5 shows the result of blurring an input image $L = 7$ times with a Gaussian kernel with $\sigma = 2$ pixels. Note the loss of detail at higher levels ($\ell > 0$) of the stack.

The convolution of a Gaussian with parameter σ_1 with another Gaussian with parameter σ_2 is a Gaussian with parameter $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$. Because of this, convolving an image ℓ times with a Gaussian with parameter σ is the same as convolving the same image once with a Gaussian with parameter

$$\sigma_\ell = \sqrt{\ell}\sigma$$

for each ℓ. So we can also write

$$\begin{array}{rcl} B_0 & = & I \\ B_\ell & = & B_0 * S_{\sigma_\ell} \quad \text{for} \quad \ell = 1, \ldots, L \; . \end{array}$$

Of course, the iterative smoothing procedure (3) is more efficient, because the kernels are smaller.

The pixel resolution of the images in the stack is high when compared to the spatial frequencies contained in the images for $\ell > 0$: There are many pixels even if the image brightness changes slowly over space. Because of this, the blurred images B_ℓ can be sampled after filtering without significant loss of information. Without getting into the quantitative aspects of sampling and image

Figure 5: A Gaussian stack.

bandwidth, it turns out that most of the image information is preserved if every time the image is blurred with a Gaussian filter with parameter σ, the image is subsampled by a factor of about $\sigma/1.6$.

When $s = \sigma/1.6$ is an integer number, it is clear what this sampling means: Filter with a Gaussian with parameter σ, then retain every s-th pixel in each dimension. When s is not an integer, on the other hand, sampling "every s pixels" entails retrieving image values between the values available in the image array. This can be done by *sub-pixel interpolation*, which requires a model for the continuous image that the array values are samples of. One of the simplest such models is the *bilinear* one, in which the underlying continuous image $I(\mathbf{x})$ is assumed to be separately linear in x and y, the two components of $\mathbf{x}$, between integer values of the coordinates. This model leads to *bilinear interpolation*: Let $\mathbf{x} = (x, y)$, and (with $\lfloor \cdot \rfloor$ denoting the floor function)

$$\begin{aligned} \xi = \lfloor x \rfloor \quad &, \quad \eta = \lfloor y \rfloor \\ \Delta x = x - \xi \quad &, \quad \Delta y = y - \eta \; . \end{aligned}$$

Then,

$$\begin{aligned} I(\mathbf{x}) \quad &= \quad I(\xi, \eta)\,(1 - \Delta x)\,(1 - \Delta y) \\ &+ \quad I(\xi + 1, \eta)\,\Delta x\,(1 - \Delta y) \\ &+ \quad I(\xi, \eta + 1)\,(1 - \Delta x)\,\Delta y \\ &+ \quad I(\xi + 1, \eta + 1)\,\Delta x\,\Delta y \; . \end{aligned}$$

We can now sample the image I with any sampling period, integer or otherwise.

We encapsulate the operations of filtering followed by sampling into a single function

$$B = \texttt{resize}(I, \phi)$$

where the *downsampling factor* $\phi = 1/s$ is a positive real number that denotes the ratio between the size of B and that of I (in each of the two coordinate dimensions). For values $0 < \phi < 1$, the image shrinks. The filter in downsampling is Gaussian with parameter

$$\sigma = 1.6/\phi \; .$$

Figure 6 shows an example of the effects of resizing an image with and without smoothing. To make the differences more obvious, a large sampling factor was used. This Figure shows that if an image is sampled without first blurring it, neighboring pixels in the reduced-size image may come from parts of the image that have nothing to do with each other in the scene, and this leads to the jagged, disorganized appearance in panel (b) of the Figure. Blurring ensures that each sampled pixel summarizes the average values in an image neighborhood that matches the sampling factor. While the reduced-size image in (c) is blurred relative to the original in (a), whatever information is left is more clearly related to the original. The phenomenon that degrades the image in (b) is called *aliasing* in the literature.

Replacing convolution with S_σ with $\texttt{resize}(\cdot, \phi)$ where $0 < \phi < 1$ in equation (3) yields the *Gaussian pyramid*:

$$\begin{aligned} G_0 &= I \\ G_\ell &= \texttt{down}(G_{\ell-1}) \quad \text{for} \quad \ell = 1, \ldots, L \; . \end{aligned} \tag{4}$$

In the last expression, we think of fixing ϕ to some value between 0 and 1 (for instance, $\phi = 1/2$) and define

$$\texttt{down}(X) = \texttt{resize}(X, \phi) \; .$$

We will sometimes also need

$$\texttt{up}(X) = \texttt{resize}(X, 1/\phi)$$

where `down` and `up` use the same value of ϕ. When the second argument to `resize` is greater than one, as it is in `up`, the function `resize` performs no filtering. This is because there is no information to suppress when going up in resolution, and there is no way to make up information that is not there. With `up`, the image grows larger rather than smaller.

The two operations `down` and `up` are called *downsampling* and *upsampling*. There is a crucial difference between the two: Downsampling blurs the input image with a Gaussian and then samples it by bilinear interpolation to make it smaller. Upsampling merely resamples the input image on a finer grid, but it does not undo the blurring, and the extra pixel values in $\texttt{up}(X)$ are merely "made up" by bilinear interpolation. So X and $\texttt{up}(X)$ contain the same frequencies (same level of detail), but the latter has more pixels than the former.

Since the image shrinks at each level, it is no longer necessary to specify the maximum level L: once the image shrinks so much that it becomes smaller than the Gaussian kernel (about $\lfloor \log_{1/\phi}(\min(R, C)) \rfloor - 1$ steps, where R and C are the number of rows and columns of I), the procedure stops.

Figure 7 shows the Gaussian pyramid for the same input image used for the Gaussian stack in Figure 5 and for $\phi = 1/2$. The input image has $R = 365$ rows and $C = 384$ columns, and the pyramid has $L = 7$ levels (plus the input image itself).

The Gaussian pyramid is said to be a *lowpass* pyramid, in that every level contains all the image frequencies *below* some value, roughly proportional to ϕ^ℓ.

(a)

(b)

(c)

Figure 6: An illustration of aliasing. The image in (a) was just resampled in (b) and instead downsized by blurring and resampling in (c). The original image has 2448×3264 pixels, and is not shown to scale with the other two images. For both (b) and (c), the sampling factor is $\phi = 1/30$.

Figure 7: A Gaussian pyramid.