Image Differentiation

Carlo Tomasi

September 4, 2017

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.

Thus, 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.

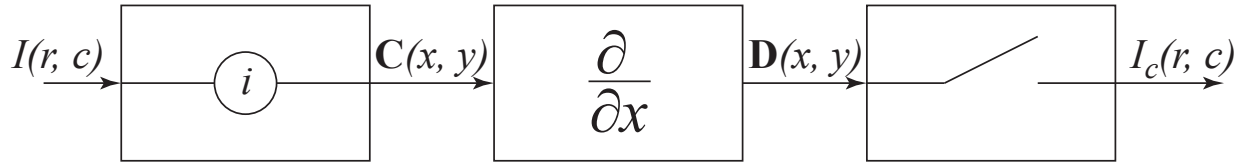


Figure 1: Conceptual bloc diagram for the computation of the derivative of image $I(r, c)$ in the horizontal (c) direction. The first block interpolates or fits the samples from a discrete to a continuous domain. The second computes the partial derivative in the horizontal (x) direction. The third block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

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

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

different meaning:[2]

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

where x, y are real variables and $P(x,y)$ is called the *interpolation* (or *fitting*) *kernel*.

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[3] and differentiation are linear, instead of interpolating the image and then differentiating we can interpolate the image with the derivative of the interpolation function. Formally,

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

where

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

is the partial derivative of $P(x,y)$ with respect to x.

Finally, we need to sample the result $\mathbf{D}(x,y)$ 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:[4]

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

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.

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) while differentiating. 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 ramp yield the slope of the ramp itself. 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

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

[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.

[3]For simplicity, we henceforth say "interpolate" to mean "interpolate or fit."

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

and we want to find a constant u_0 such that

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

for all r, c so that

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

In particular for $r = c = 0$ we obtain

$$u_0 = -\frac{1}{\sum_{i=-n}^{n} \sum_{j=-n}^{n} j G_x(j,i)} \ . \tag{1}$$

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

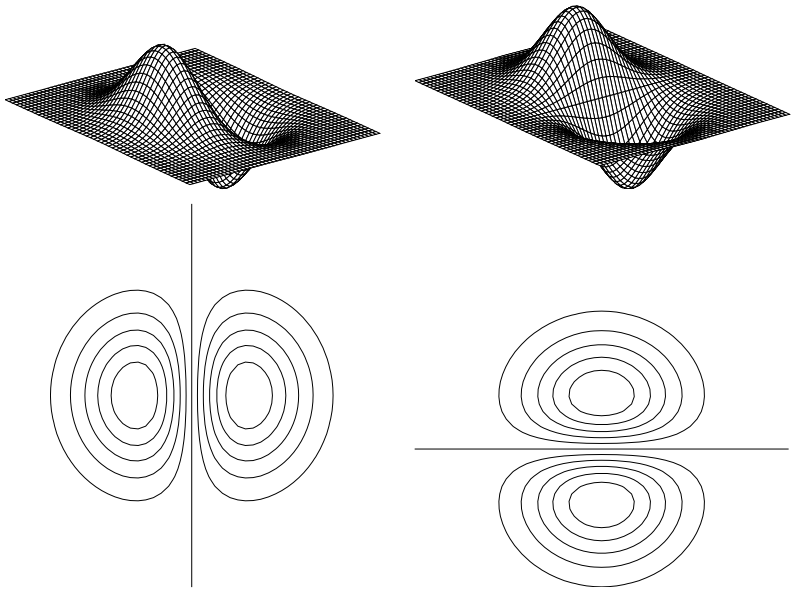


Figure 2: The partial derivatives of a Gaussian function with respect to x (left) and y (right) represented by plots (top) and isocontours (bottom). In the isocontour plots, the x variable points horizontally to the right, and the y variable points vertically down.

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

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

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).

Thus, the partial derivative of an image in the x direction is computed by convolving[5] 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:

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

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):

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

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

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

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.

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

The Image Gradient

A value of the partial derivative I_x and one of I_y can be computed at every image position[6] (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

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

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

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

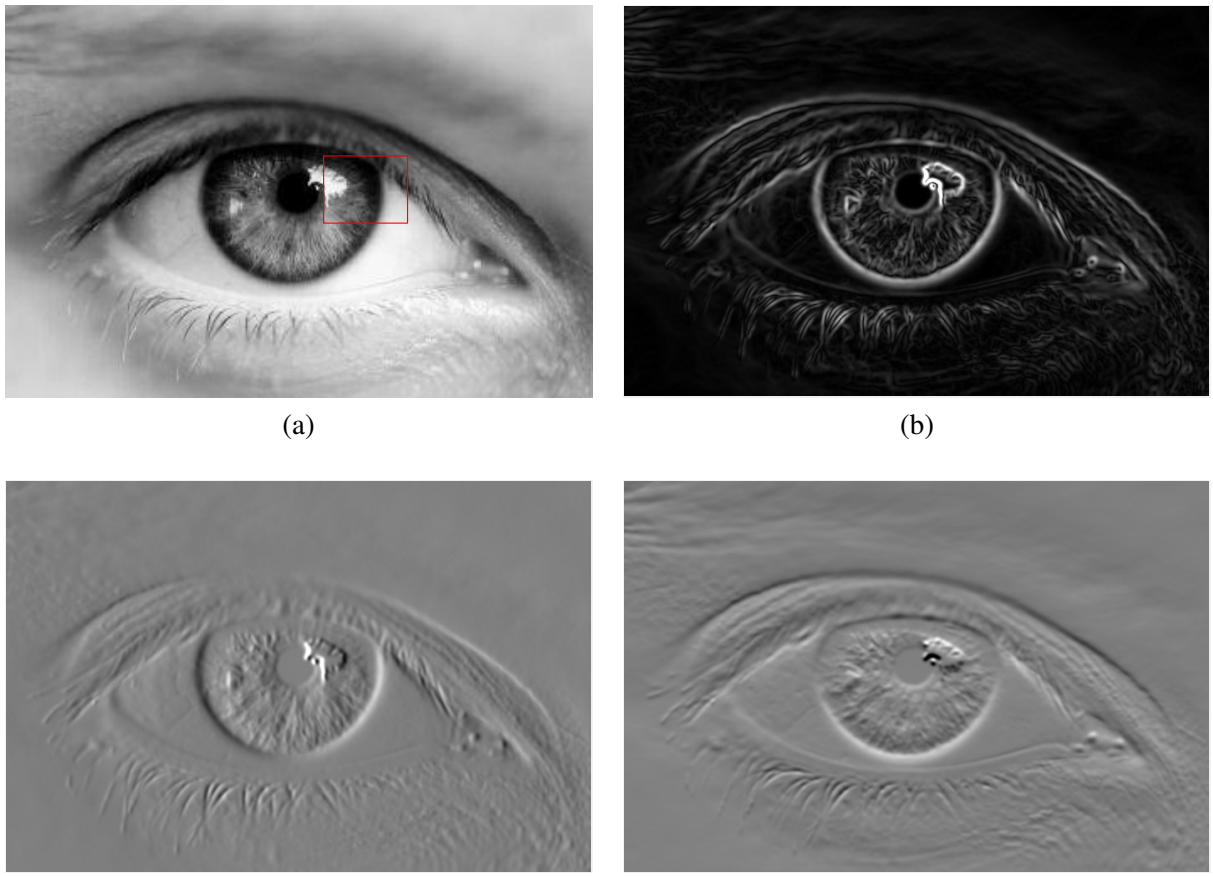
so that the chain rule for differentiation yields

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

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

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

[6] We assume to pad images by replication 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.

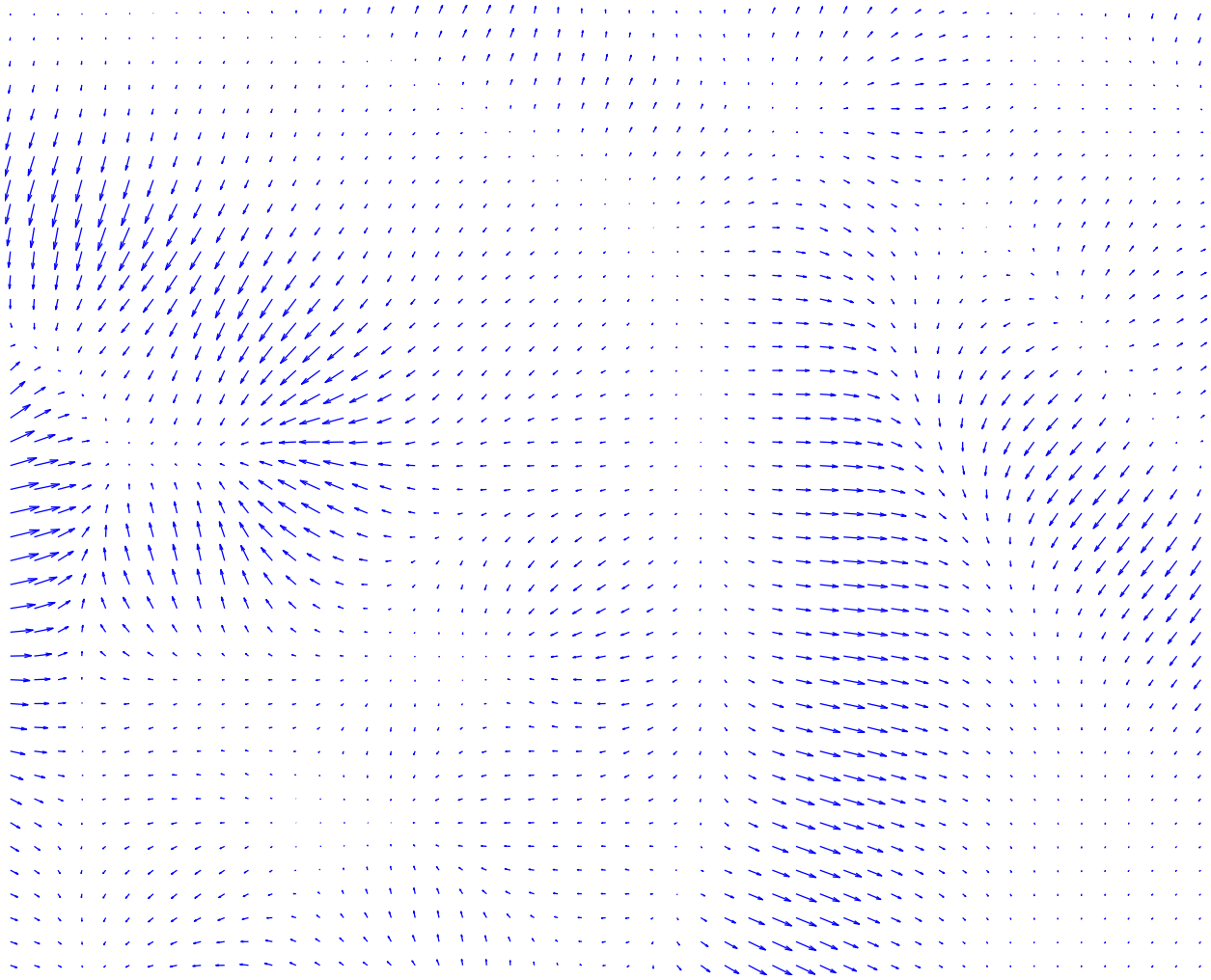


Figure 4: Quiver diagram of the gradient in the detail from the box of Figure 3 (a). Note the long arrows pointing towards the bright glint in the eye, and those pointing from the pupil and from the eyelid towards the eye white.