From Pixels to Features, J.C. Simon (ed.)
© Elsevier Science Publishers B.V. (North-Holland), 1989

Computational Considerations in Convolution and Feature-extraction in Images

Robert Hummel and David Lowe[1]

Courant Institute of Mathematical Sciences
New York University
251 Mercer Street, New York, NY 10012 USA

Blurring by Gaussian convolution, or by a Laplacian of a Gaussian kernel, is a common image processing technique used for edge detection, multiresolution representation, motion analysis, matching, and other early visual processing tasks. We consider methods for computing Gaussian convolutions on discrete grids, and also consider difference-of-Gaussian and Laplacian-of-Gaussian convolutions. To properly approximate the continuous theory, it is important to evaluate the elements in these large kernels by integration of the continuous kernel in a region about the corresponding location. We further consider zero-crossings of filtered images, and look at methods for selecting significant zero-crossing curves. Finally, we discuss how to handle borders. We note difficulties and instabilities in dealing with zero-crossings, and discuss briefly ways to deal with these problems.

1. Continuous Convolution

Many image processing operations rely on convolutions against the image intensity data. While many operators are local, and can be computed using convolutions with three-by-three pixel kernels, there are some operators that require much larger convolution kernels. For example, the Marr-Hildreth edge operator [1] requires convolution kernels that are sometimes as large as fifty by fifty pixels in extent. Most of the large kernel filters are based on Gaussian blurring of the image data either to compute a blurred, smoothed version of the image data, or to compute a difference-of-Gaussian image, or a Laplacian-of-Gaussian image.

We will give a number of computational techniques needed for efficient, accurate digital implementation of these kernels and interpretation of the results. Many of the computational methods discussed here are well-known, and we are mainly concerned with bringing together a compendium of observations in implementing large kernels based on Gaussians. A treatment of large kernel convolutions, together with many further results, can be found in [2]. An article by Huertas and Medioni [3] contains many relevant suggestions for implementation of the types of kernels discussed here, and the Grimson and Hildreth discussion [4] of Haralick's paper on the second directional derivative [5] gives some details of their implementations.

A Gaussian convolution is defined, in two continuous variables, by

$$h(x,y) = \int_{\mathbb{R}^2} G(x-x',y-y')f(x',y')dx'dy',$$

where $G(x,y)=(1/2\pi\sigma^2)e^{-(x^2+y^2)/2\sigma^2}$. Here $f(x,y)$ is the input image, and $h(x,y)$ is the desired filtered image.

The need for discretization is imposed since the input function is in fact given at a discrete grid of points, say $f(i,j)$, where i and j take on integer values. The temptation is to convert the integral into a sum, replacing the integrand by point evaluations at the grid points, i.e.,

[1]The second author is now with the Computer Science Dept., University of British Columbia, Vancouver, B.C. V6T 1W5, Canada.

$$h(i,j) \approx \sum_{i'}\sum_{j'} G(i-i',j-j')f(i',j') \ .$$

This can lead to inaccuracies, however.

Instead, for all convolutions, including Gaussian convolution, the integration should be discretized by modeling the continuous $f(x,y)$ from the given discrete data $f(i,j)$. The easiest method is to assume that $f(x,y)$ is obtained from $f(i,j)$ by nearest-neighbor interpolation. Then the continuous integral can be computed from the discrete sum in the form:

$$h(i,j) = \sum_{i'}\sum_{j'} \overline{G}(i-i',j-j')f(i',j'),$$

where

$$\overline{G}(i,j) = \int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} G(x,y)dxdy.$$

Thus the discrete kernel elements should be obtained from block averages of the continuous kernel, and not by point evaluations.

If bilinear interpolation is used to model $f(x,y)$ instead of nearest neighbor interpolation, the formula becomes:

$$\overline{G}(i,j) = \int_{i-1}^{i+1}\int_{j-1}^{j+1} G(x,y)\cdot(1-|i-x|)\cdot(1-|j-y|)dxdy.$$

Higher order interpolants can be treated similarly. In general, the discrete kernel elements are obtained by integrating the continuous kernel against the basis functions of the interpolation method.

The result of the block averaging procedure is that the kernel values can be changed somewhat. Figure 1 shows a one-dimensional Gaussian curve with dots at the integer points, superimposed on a bar graph representing the block average values. The bar graph gives, in a sense, a more accurate discrete Gaussian convolution. Further, the (infinite) sum of the averaged values $\overline{G}(i,j)$ are guaranteed to sum to one. Interestingly, the infinite sum of the evaluates of the Gaussian $G(i,j)$ also very nearly sum to one, as long as σ is larger than about 0.5 pixels. This occurs because the errors tend to cancel, due to the fact that the Gaussian has a region of negative curvature, and another region with positive curvature. The first row of Table 1 gives the sum of the kernel values for different values of σ for a two-dimensional Gaussian kernel. However, the fact that the point evaluates of the Gaussian very nearly sum to one doesn't justify their use for Gaussian

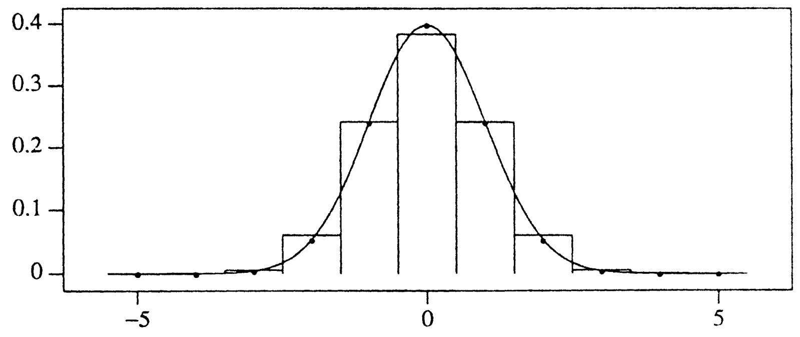


Figure 1. A one-dimensional Gaussian, with σ=1.0. Dots mark point evaluates of the Gaussian at each integer. The heights of the bars in the bar graph give the mean values of the Gaussian in a block of width 1 centered at each integer.

	σ								
	.50	.60	.75	1.0	1.5	2.0	3.0	4.0	5.0
$\sum\sum G(i,j)$	1.02897	1.00328	1.00006	1.0	1.0	1.0	1.0	1.0	1.0
$\sum\sum \lvert G(i,j)-\bar{G}(i,j)\rvert$	0.31213	0.17204	0.09651	0.058	0.026	0.015	0.0068	0.0038	0.0024
$\sum\sum \Delta G(i,j)$	-1.15203	-0.12971	-0.00238	~0.0	~0.0	~0.0	~0.0	~0.0	~0.0
$\sum\sum \lvert \Delta G(i,j)-\overline{\Delta G}(i,j)\rvert$	3.84728	1.86594	1.17351	0.768	0.414	0.225	0.1039	0.0593	0.0382

Table 1. The sum of point-evaluate kernel elements, where $G(x,y)$ is a Gaussian in two variables with standard deviation σ and total integral 1.

convolution — the averaged values $\bar{G}$ form a better approximation to the continuous theory.

Indeed, assume that the image data is bounded, say $|f(x,y)| \le M$. We have denoted the convolution using block average kernel values $\bar{G}(i,j)$ by $h(i,j)$. If point evaluates $G(i,j)$ are used instead, let us denote the result by $h_P(i,j)$:

$$h_P(i,j) = \sum_{i'}\sum_{j'} G(i-i',j-j')f(i',j') .$$

We can then bound the difference:

$$|h(i,j)-h_P(i,j)| \le \sum_{i'}\sum_{j'} |G(i',j') - \bar{G}(i',j')| \cdot M .$$

The bound can be attained, and thus the sum $\sum\sum |G(i,j) - \bar{G}(i,j)|$, measures a proportionality factor for the maximum error in using point evaluates. These values are listed in the second row of Table 1 for various values of σ.

The need for averaging the continuous kernel rather than point evaluations is more critical when Laplacian-of-Gaussian convolutions are attempted. Accuracy is often very important; for example, the locations of zero-crossings can depend critically on small changes in values of the filtered image. In the case of Laplacian-of-Gaussian convolution, the kernel is given by:[2]

$$\Delta G(x,y) = C \cdot \left[\frac{x^2+y^2}{\sigma^2} - 2 \right] e^{-(x^2+y^2)/2\sigma^2} .$$

Here C is a constant, which if a precise Laplacian-of-Gaussian is desired is equal to $1/(2\pi\sigma^4)$. Once again, point evaluations of the kernel lead to different values than block averages. Figure 2 depicts the one-dimensional situation with the Laplacian of the Gaussian, with σ=1.0 . This time, kernel elements should sum to zero. Once again, the block-averaged kernel elements, $(\overline{\Delta G})(i,j)$, are guaranteed to sum to zero, while the point evaluates $\Delta G(i,j)$ serendipitously very nearly sum to zero, as long as σ is sufficiently large (greater than 1.0 suffices). The third row of entries in Table 1 gives the values for the infinite sum of kernel elements. The error bound for using point evaluates of the Laplacian-of-Gaussian as opposed to block averages can be measured by the sum of absolute differences: $\sum\sum |\overline{\Delta G}(i,j) - \Delta G(i,j)|$. These values are listed in the fourth row of Table 1 for varying values of σ. We see that point evaluates can lead to rather large errors, for example, 70% relative errors at $\sigma = 1.0$ pixels, and remaining as high as 10% relative error for $\sigma = 3$ pixels.

[2]We use the notation " Δ " to denote the Laplacian operator, as is common in mathematical analysis. In physics, the notation " ∇^2 " is often used for the Laplacian operator in $\mathbb{R}^3$ (three-dimensional space), as a shorthand for the divergence of the gradient. The notation is explained by the fact that the gradient of a function f is generally denoted by ∇f, and the divergence of the result is denoted $\nabla \cdot \nabla f$. The notation " ∇^2 " has been borrowed and applied in $\mathbb{R}^n$ for more general n by many disciplines.

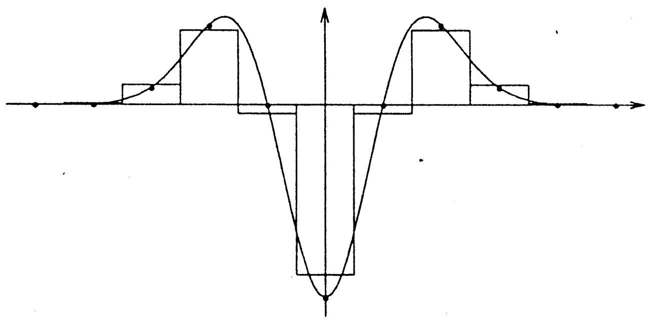


Figure 2. The Laplacian of a Gaussian in one dimension, i.e., the second derivative of a Gaussian, with σ=1.0. The point evaluates and mean values at each integer are shown as in Figure 1.

We conclude that the use of averaged kernel elements as opposed to point evaluates is important, especially for the Laplacian-of-Gaussian kernel.

Analytic evaluation of the averages of the kernels is possible in terms of the error function *erf*. For example, block averages of the Gaussian are given by:

$$\overline{G}(i,j) = \int_{i-1/2}^{i+1/2} \int_{j-1/2}^{j+1/2} G(x,y)\,dxdy =$$

$$\frac{1}{4}\left[\operatorname{erf}\left[\frac{i+1/2}{\sqrt{2}\sigma}\right] - \operatorname{erf}\left[\frac{i-1/2}{\sqrt{2}\sigma}\right]\right] \cdot \left[\operatorname{erf}\left[\frac{j+1/2}{\sqrt{2}\sigma}\right] - \operatorname{erf}\left[\frac{j-1/2}{\sqrt{2}\sigma}\right]\right],$$

where *erf* is the error function associated with the normal distribution, i.e., $\operatorname{erf}(x) = \int_0^x (2/\sqrt{\pi})e^{-t^2}dt$. (*Erf* is a built-in function on most Fortran compilers). Similarly, block averages of the Laplacian-of-Gaussian kernel are given by

$$\overline{\Delta G}(i,j) = \int_{i-1/2}^{i+1/2} \int_{j-1/2}^{j+1/2} \Delta G(x,y)\,dxdy = \frac{1}{2}\left[g'(i+1/2) - g'(i-1/2)\right]\left[\operatorname{erf}\left[\frac{j+1/2}{\sqrt{2}\sigma}\right] - \operatorname{erf}\left[\frac{j-1/2}{\sqrt{2}\sigma}\right]\right]$$

$$+ \frac{1}{2}\left[g'(j+1/2) - g'(j-1/2)\right]\left[\operatorname{erf}\left[\frac{i+1/2}{\sqrt{2}\sigma}\right] - \operatorname{erf}\left[\frac{i-1/2}{\sqrt{2}\sigma}\right]\right],$$

where $g'(x) = (-x/\sqrt{2\pi}\sigma^3)e^{-x^2/2\sigma^2}$.

For more general continuous kernels, analytic evaluation may be impossible or simply too much of a bother. The averaged kernel can then be computed (off-line) by numerical quadrature methods, using standard numerical integration package software. Simpson's method for one-dimensional integration usually suffices for the kernels under consideration.

A separate issue when implementing these convolutions concerns quantization and truncation of the kernel. In practice, the kernel elements are typically approximated by rational numbers, so that integer arithmetic may be used. Depending on the number of bits used to represent the kernel elements, the kernel will be effectively truncated to a finite domain, since elements outside of some radius will be approximated by zero. Further truncation is also possible. When using these kernels, errors come from two separate sources: from the quantization error due to the approximation of the kernel elements by rational values, and the truncation error due to the omission of kernel elements outside of a given radius. The first error is controlled by the number of bits used in the representation of the kernel elements. The truncation error is dependent on the radius used for the implementation of the kernel. For bounded image data, the truncation error is bounded by an amount proportional to the integral of the kernel outside of the truncation radius.

Hildreth notes that the central negative region in $\Delta G(x,y)$ has radius $w=\sqrt{2}\sigma$, and then suggests that the Laplacian-of-Gaussian kernel be implemented with a radius of $4w\approx 5.66\sigma$ [6]. In Figure 3a, we plot the values of the integrals of the continuous kernels $G(x,y)$ and $\Delta G(x,y)$ outside of the truncation radii $t\cdot\sigma$. (Actually, the table gives the integrals of the kernels outside a box of size $2t\sigma$ by $2t\sigma$). Note that the y-axis is logarithmic. It should also be noted that these plots are independent of σ (the function $G(x,y)$ depends on σ). The truncation error drops as the truncation location $t\cdot\sigma$ is increased. We see that for Gaussian convolution, the truncation errdr is less, ranging from roughly 10% at 2σ down to less than one part in 10^7 at 5.6σ. Laplacian-of-Gaussian convolution has larger truncation errors, (the dashed curve), with roughly 5% error at 3σ, and one part in 10^6 accuracy at 5.6σ.

Quantization error is a consequence of representing the kernel elements imprecisely. If each kernel element is accurate to with ε, and the image data is bounded by M, then each term in the convolution can theoretically contribute $M\varepsilon$ in error. In practice, however, quantization errors in the representation of the kernel values tend not to accumulate, so that we may assume that the total

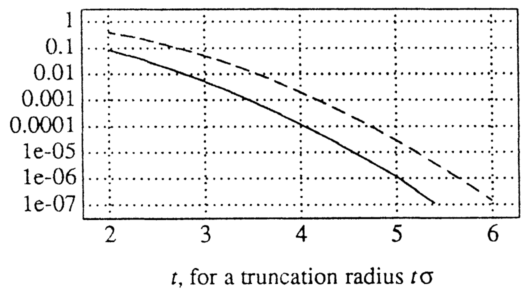


Figure 3a. Truncation errors due to clipping of the kernel within a box, for the block averaged Gaussian kernel (solid curve) and the Laplacian-of-Gaussian (dashed curve). The size of the box is $2t\cdot\sigma$ by $2t\cdot\sigma$, where t is the abcissa value. The values plotted represent the absolute value of the integral of the kernel outside the box.

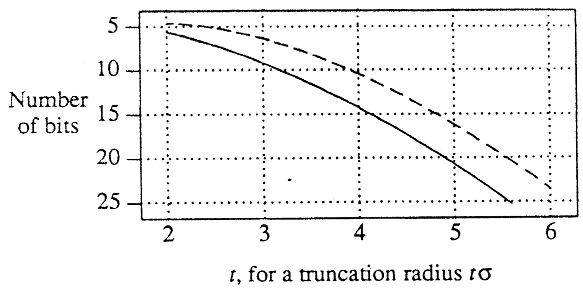


Figure 3b. The number of bits required to represent kernel elements at the extreme points of a specified radial cutoff. The solid curve gives the number of bits needed so that the effective truncation location of the Gaussian is at a radius of $t\cdot\sigma$, while the dashed curve is the same for the Laplacian-of-Gaussian kernel.

error due to quantization is of order $O(M\varepsilon)$, independent of the size of the kernel. The accuracy ε is usually determined by the number of bits b used to represent fixed-point versions of the kernel elements: $\varepsilon \approx 2^{-b}$. There is also a relationship between the kernel truncation size and ε. A maximum truncation radius is determined by ε, since once the kernel decays below ε, the kernel is effectively set to zero. Put another way, for a given truncation radius, a minimum number of bits are required to represent the kernel elements to that radius, and more bits may be used at the expense of some computation costs.

In Figure 3b, we assume that the truncation radius is set, and that ε is thereby chosen as the smallest kernel element. We then plot the number of bits required to represent the smallest kernel element, for both the Gaussian kernel (solid line) and the Laplacian-of-Gaussian (dotted). We see, for example, that roughly 25 bits of precision suffices for both kernels at a radius of 5.6σ. This strategy, setting the number of bits so that the quantization of kernel elements effects the truncation at the desired radius, seems to be a sensible method for balancing quantization and truncation error with computational effort.

Of course, high accuracy is an issue only if one believes that it is important that the appropriate kernel (Gaussian or Laplacian-of-Gaussian) be implemented accurately. For example, using point evaluates for the kernel elements leads to potentially different results, and thus is not necessarily an accurate implementation. However, it might happen that for the intended application, the differences are inconsequential, and the Gaussian or Laplacian-of-Gaussian can be replaced by less accurate kernels or more general kernels. We reiterate that the determination of locations of zero-crossings is an application that *does* require high accuracy.

In order to implement Gaussian convolution efficiently, two computational tricks can be used. One is to make use of the fact that the Gaussian kernel is separable, and to thus perform the convolution first in one direction, and then in the orthogonal direction. It should be noted that the integrated Gaussian kernels, given above, are also separable. Making use of the separability of the Gaussian kernel does not compromise the accuracy of the calculation. A second trick that is especially important with certain pipeline architectures is that the Gaussian may be approximated by the iterated convolution against a local discrete kernel. In one dimension, if a discrete signal $f(i)$ is iteratively convolved against the three tap kernel [1/4 1/2 1/4] N times, then the result is the same as the convolution

$$h_N(i) = \sum_{k=-N}^{N} \frac{1}{2^{2N}} \binom{2N}{k+N} f(i+k).$$

This is an approximation to convolution against a Gaussian with $\sigma = \sqrt{N/2}$. The approximation improves as N increases. In two dimensions, the iterated convolution, performed N times, against the three-by-three mask

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} .$$

yields an approximation to the two-dimensional convolution against a Gaussian with $\sigma = \sqrt{N/2}$.

2. Difference-of-Gaussians

A number of operations are based on difference-of-Gaussian (DoG) or Laplacian-of-Gaussian (LoG) operations. The Marr-Hildreth edge operator is one such; another is the DOLP [7]. First, it should be noted that the DoG is an approximation to the LoG, in the sense that

$$\Delta G(x,y) = \lim_{\sigma_2 \to \sigma_1} \frac{1/2}{\sigma_1^2 - \sigma_2^2} \left[G_{\sigma_1}(x,y) - G_{\sigma_2}(x,y) \right] .$$

If σ_1 and σ_2 are very different, then the approximation is not very good. On the other hand, if σ_1 and σ_2 are very close, implementing the difference quotient may well lead to numerical precision problems. In many schemes, such as the Burt Laplacian pyramid [8], the effective DoG is formed with the ratio of σ_1 and σ_2 fixed. In fact, Marr and Hildreth suggested a separation according to $(\sigma_1/\sigma_2) = 1.6$. This suggestion is often misinterpreted as a statement that the LoG is best approximated by a DoG with $\sigma_1/\sigma_2 = 1.6$, which is clearly not true.

Image 2 shows the zero-crossings for the difference-of-Gaussians applied to the image showed in Image 1, where the ratio of the σ's, $\sigma_1/\sigma_2 = 1.6$, while Image 3 shows the zero-crossings using the Laplacian of the Gaussian. For Image 2, we used $\sigma_1 = 6.4$ and $\sigma_2 = 4.0$, while the LoG example in Image 3 is based on $\sigma = 4.9$. In this example, we find that the DoG provides slightly better correlation of the zero-crossings with edges and significant features in the original image. We might also compare the DoG with an LoG with a different intermediate value of σ, but our experience with such comparisons is similar to the one shown. Namely, we find that the location and density of zero-crossings is slightly more stable when a DoG is used.

If a pyramid scheme is desired, then the Burt method is generally the one of choice. The effective ratio of scales, σ_2/σ_1, is equal to 2 in the standard scheme where pyramid levels decrease in size by a factor of 2 between levels. Other schemes, however, are possible [2,9]. If a DoG is desired at a fixed maximum resolution, then for greatest efficiency, the two Gaussian-convolved images should be computed, and some care should be exercised in computing the difference, since numerical precision may be difficult.

If the LoG is desired, despite advantages of the DoG, it can be calculated by any of the following four methods:

(1) Use the Laplacian-of-Gaussian kernel, averaged in blocks, to convolve against the data $f(i,j)$, as discussed in the previous section. Note that the kernel is not separable, so that the resulting convolution will be expensive.

(2) First compute a Laplacian of the image data, using a four-point Laplacian, i.e.,

$$L(i,j) = f(i-1,j) + f(i+1,j) + f(i,j+1) + f(i,j-1) - 4f(i,j),$$

and then Gaussian-blur the result. In this method, numerical precision in the blurring of $L(i,j)$ is a major concern, since if insufficient bits for the representation are provided, the result of any substantive amount of blurring will be zero.

(3) Compute blurred versions h_N of the data $f(i,j)$ using Nth iterated convolutions against a local kernel, but using full accuracy by retaining all necessary bits. The LoG is approximated using $h_{N+1} - h_N$.

(4) Use the Huertas-Medioni decomposition of the Laplacian-of-Gaussian into the sum of two separable kernels [10]. Specifically, the decomposition is given by

$$\Delta G(x,y) = C \cdot h_1(x) \cdot h_2(y) + C \cdot h_2(x) \cdot h_1(y),$$

where

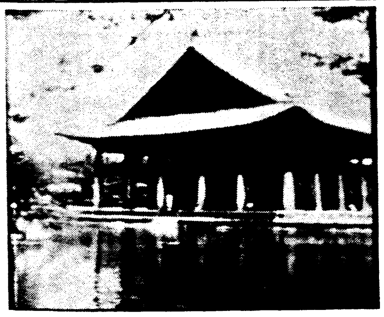
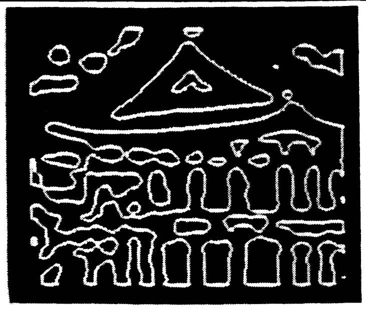



Image 1 and 2. An original image, and the zero-crossings of the Difference-of-Gaussian filtered version of the image in Image 1, where the Gaussians have standard deviation σ's corresponding to 4 pixels and 6.4 pixels.

$$h_1(x) = \left[\frac{x^2}{\sigma^2}-1\right]\cdot e^{-x^2/2\sigma^2}, \quad h_2(y) = e^{-y^2/2\sigma^2}.$$

Note that in each of the two separable kernels, one of the two dimensions uses a Gaussian convolution. The other convolution in each of the two kernels is a second derivative of a Gaussian in one dimension. The decomposition arises since $\Delta(g(x)g(y)) = g''(x)g(y) + g(x)g''(y)$.

For the examples in Image 2 and 3, we used double-precision floating-point representations of the image and kernel elements, for maximum accuracy. We used extremely large kernels to essentially eliminate truncation errors. When we used any of the approximation methods or computational tricks discussed above on a limited-precision processor, the results of applying the zero-crossing operation were frequently disconcertingly different. The difficulty is that the location of the zero-crossings can be extremely sensitive to small changes in the filtered images. There were always a number of zero-crossings (with high gradients in the filtered data) that were not affected by the approximation methods, but many of the zero-crossings belonging to texture edges and weak edges were considerably changed.

In performing the separable convolutions in the Huertas-Medioni decompositions of the LoG, one should use kernel elements obtained from block averages, as discussed in Section 1. The block averages have analytical expressions, namely:

$$\int_{i-1/2}^{i+1/2} h_1(x)dx = g'(i+1/2) - g'(i-1/2),$$

where $g'(x) = -xe^{-x^2/2\sigma^2}$, and

$$\int_{i-1/2}^{i+1/2} h_2(x)dx = \frac{1}{2}\left[\operatorname{erf}\left[\frac{i+1/2}{\sqrt{2}\sigma}\right] - \operatorname{erf}\left[\frac{i-1/2}{\sqrt{2}\sigma}\right]\right].$$

3. Zero-crossings

To compute the zero-crossings of a function of two variables, $h(i,j)$, the data should be thresholded at zero, and border points identified in the resulting binary image. There have been some attempts at obtaining subpixel accuracy in the determination of zero crossings; in this case we can view the situation as the same, with a high-resolution interpolated image obtained from the filtered data. We mention two possible definitions for a border point. A border point can be defined as a pixel that contains a pixel of a different binary value among its four (or eight) neighbors. In this case, edges will be two pixels thick. Alternatively, a border point can be defined as a "1" pixel with a "0" pixel among its neighbors. One way of finding border pixels using the second definition, appropriate when fast three-by-three convolution hardware is available, is to blur the

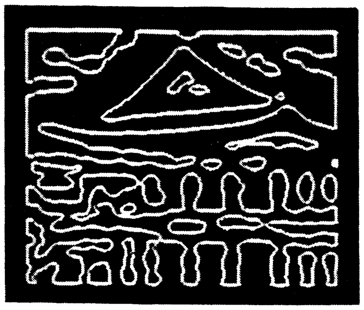


Image 3. The zero-crossings of the Laplacian-of-Gaussian filtered version of the image in Image 1, where the Gaussian has standard deviation σ corresponding to 4.9 pixels.

binary image using the mask with 1's in the eight neighbors and 9 in the center, and then thresholding the result so that values 9 and above, but less than 17, are identified as border pixels. (Here we have used eight pixel neighborhoods.)

Haralick has suggested that zero-crossings of the second directional derivative, where the direction of differentiation is the direction of the local gradient [5], will tend to more accurately locate edges as compared to the Marr-Hildreth edge operator. It should be noted that this operator also yields closed contours for the zero-crossings, and can be computed by finding zero-crossings of:

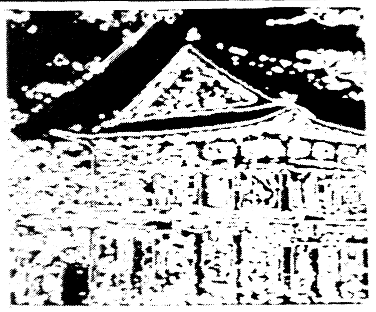
$$h(x,y) = f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy},$$

where a subscript denotes a partial derivative. The nonlinear second order partial differential operator on the right hand side is in fact the directional derivative scaled by the square of the gradient, and is thus defined even when the gradient is zero. Canny [11] notes that the operator can be formulated as $(1/2)\nabla f \cdot \nabla |\nabla f|^2$. We used a different computation, formulating the partial derivatives by using local three-by-three convolutions. Specifically we have used:

$$f_x \sim \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad f_y \sim \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{bmatrix},$$

$$f_{xx} \sim \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad f_{xy} \sim \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}, \quad f_{yy} \sim \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The discretized $h(i,j)$ is a sum of products of local convolutions of image data as given by these (or related) kernels. The image data can be the original image, or an appropriately blurred version of the image data. Haralick recommends that the partial derivatives of f be computed using the "facet model," so that the smoothing is implicit in the representation of the parameters of the local facet for f. Other methods of blurring can be used, including Gaussian blur, although this method of computation does not work well with images that have been substantially blurred by a Gaussian due to the small magnitude of the local gradient of the blurred image. In Image 4a, we show the zero-crossings of the second directional derivative, after the image has been only slightly blurred ($\sigma \approx 0.7$). This should be compared with zero-crossings of the Laplacian-of-Gaussian of the same image at a comparable scale (Image 4b).



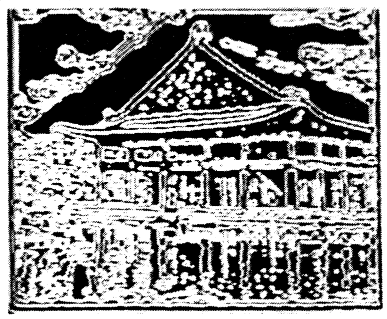


Image 4a. The zero-crossings of the second directional derivative, computed in the direction of the gradient (the Haralick operator). The original image was blurred by a Gaussian, with $\sigma \approx 0.7$.
Image 4b. The zero-crossings of a Laplacian-of-Gaussian convolution of the image, taken at approximately the same scale of resolution as Image 4a. We note that the Laplacian-of-Gaussian tends to introduce a number of extraneous zero-crossings, and tends to be slightly less accurate near corners.

Whether a DoG, LoG, or second directional derivative is used, the zero-crossings do not necessarily track the edges accurately. Prominent edges will generally have a zero-crossing curve associated with them (at some scale of resolution), but the zero-crossings will tend to wander off of the edges and complete loops for which the main edge forms only a portion of the loop. Thus it is desired to tag pixels identified as a zero-crossing with a measure of the edge strength. One way of doing this, used frequently in practice, is to measure the slope of the filtered image at the zero-crossing. Prominent edges usually give rise to zero-crossings for which the crossing has a strong slope. Thus a measure of importance is given by $|\nabla(\Delta G * f)|(x,y)$, at points (x,y) where $\Delta G * f(x,y) = 0$. This idea, for example, is mentioned in Marr's book [12]. The slope at a zero-crossing can be computed by taking a Sobel gradient magnitude of the Laplacian-of-Gaussian filtered image. We have found that by throwing out zero-crossing points where the Sobel gradient magnitude is below a threshold, the remaining curves form a much more accurate depiction of the edges in the image.

One reason why the zero-crossings include curves that do not correspond to physical edges is that the zero-crossings of the second directional derivative, and the zero-crossings of the Laplacian as an approximation to the second directional derivative, include both gradient maxima and gradient minima. Gradient minima arise when there are two successive step edges of the same step direction lying approximately parallel in the image; James Clark considers the "phantom edge" produced by the gradient minima [13], and notes that for the second directional derivative, a zero-crossing is a gradient maxima if and only if

$$\frac{\partial f}{\partial n}\frac{\partial^3 f}{\partial n^3} < 0 ,$$

where $\partial/\partial n$ is the directional derivative in the direction of the Gradient. Berzins [14] has suggested that for the Laplacian-of-Gaussian images, only zero-crossings that satisfy

$$\frac{\partial f}{\partial n}\frac{\partial(\Delta f)}{\partial n} < 0$$

be retained, where $\partial/\partial n$ now stands for the directional derivative perpendicular to the zero-crossing. Clark notes that this is equivalent to the test

$$\nabla f \cdot \nabla(\Delta f) < 0 ,$$

and is a way of (approximately) retaining only edges corresponding to gradient maxima. Image 5 shows the zero-crossings of Image 3, retaining only those zero-crossings passing the test given above. Because of quantization of the values, edges with extremely small image gradients or extremely small gradients in the filtered data are also eliminated. We see from Image 5 that although some extraneous zero-crossings are eliminated, the zero-crossing screening is not a panacea.

The essential difficulty with the methods for "cleaning-up" zero-crossings is that there is no clear and obvious translation of the notion of a physical edge to a local condition on image intensity variations. Physical edges can be associated with both small and large gradients, with zero-crossings, or with texture or other feature boundaries. Marr and Hildreth [1] originally suggested that this problem could be addressed by combining zero-crossings from multiple scales. For example, zero-crossings that change location very slowly over a range of scales might be presumed to arise from a single physical cause. However, experiments with "and"-ing the zero-crossing contours at multiple scales (with contours in all but one of the scales suitably thickened before "and"-ing) tend not to be successful at retaining edges with low intensity gradients, even when those correspond to true physical edges. Perhaps orientation selectivity needs to be considered before combining edges at varying scales. Also, noise and local texture properties need to be evaluated to account for movement of zero-crossings. However, there is currently no generally-accepted method for combining contours across scales.

4. Borders

All of the convolutions discussed so far assume that the original data $f(x,y)$ is defined for $(x,y) \in \mathbf{R}^2$ and that the grid of points (i,j) is an infinite lattice. Generally, however, images are

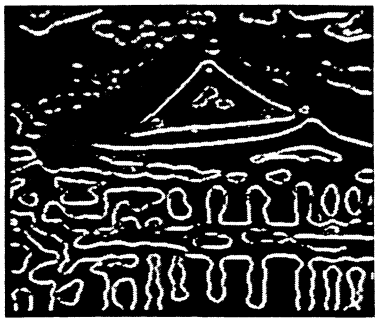


Image 5. The zero-crossings of Image 3, retaining only those zero-crossings that pass the "Berzin's test," namely those for which $\nabla f \cdot \nabla(\Delta f) < 0$.

defined on a finite rectangular grid of points, or sometimes on an irregularly shaped grid. Suppose that $f(i,j)$ is defined for $(i,j) \in D$, where D is a domain of grid points. We define the boundary points of the domain as points in D with at least one of the eight neighbors outside of D, and denote those points by ∂D. How should the infinite convolutions be defined?

A usual tactic is to simply extend $f(i,j)$ to be zero for pixels outside of D. For Gaussian convolution, however, there are alternate approaches, motivated by regarding Gaussian convolution as a means of solving the Heat equation [15]. Specifically, we can use the formulas for iterative blurring in conjunction with fixed boundary data. In this approach, $h_{N+1}(i,j)$ is an image obtained from $h_N(i,j)$ as follows. For interior points, $(i,j) \in D - \partial D$, the $h_{N+1}(i,j)$ values are obtained from the convolution of h_N against the three by three kernel given at the end of Section 1. For border points, set $h_{N+1}(i,j) = h_N(i,j)$. Although this yields something different from Gaussian blurring, the result is an appropriate blurring of the image data. An analog to difference-of-Gaussian filters can then be computed. Note that such difference filters, when defined this way, will always give zero values on the boundary.

Another possibility for handling borders is to, in essence, insist that the normal derivative at the borders be zero. In terms of the discrete blurring kernels, this condition can be implemented (for a convex grid of points D, and when the kernel is no bigger than three-by-three) by the following rule: *Every access to a pixel outside of the grid D should substitute for that pixel the value of the closest pixel in the grid D.* We call the result *adiabatic blurring.*

Thus, for example, suppose that D consists of the rectangular array of pixels $\{(i,j) \mid 0 \le i \le 511, 0 \le j \le 511\}$. Then to update a pixel on the left edge, $(i,0)$, with $1 \le i \le 510$ (not a corner point), the formula is

$$h_{N+1}(i,0) = (\frac{1}{16}) \cdot \left[3h_N(i-1,0) + h_N(i-1,1) + 6h_N(i,0) + 2h_N(i,1) + 3h_N(i+1,0) + h_N(i+1,1) \right] .$$

Similarly, in the upper left corner, the update rule is

$$h_{N+1}(0,0) = (\frac{1}{16}) \cdot \left[9h_N(0,0) + 3h_N(0,1) + 3h_N(1,0) + h_N(1,1) \right] .$$

For such a rectangular grid D, the rule can be seen to be equivalent to first blurring the rows by the formulas using the masks [1/4 1/2 1/4] at interior points, [0 3/4 1/4] at left edges, and [1/4 3/4 0] at right edges, and then blurring the resulting columns by similar masks oriented vertically.

This method of updating has the property that the global mean is constant, i.e., $\sum\sum h_N(i,j)$ is independent of N. Once again, differences of levels can be formed to construct analogs to the DoG and LoG structure. The results will be the same as if the image were extended by zeros or some other fashion to an infinite domain, and then blurred by a Gaussian, except possibly very the borders. Indeed, at the Nth level of blur by an iterated blurring procedure, only points within N

pixels of the border will show any difference. If the image is extended by zeros, then the borders will tend to be "darker" if Gaussian blurring is used as opposed to adiabatic blurring. If instead the image is extended by flipping the image left-to-right and top-to-bottom, to form a doubly periodic image with period twice the size of the original image in each dimension, then Gaussian blurring results in precisely the same image as adiabatic blurring.

Acknowledgements

This research was supported by Office of Naval Research Grant N00014-85-K-0077, Work Unit NR 4007006, and NSF grants DCR-8403300 and DCR-8502009.

References

[1] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings Royal Society London (B)*, p. 187 (1980).

[2] P. Burt, "Fast filter transforms for image processing," *Computer Graphics and Image Processing* **16**, p. 20 (1981).

[3] A. Huertas and G. Medioni, "Detection of intensity changes with subpixel accuracy using Laplacian-Gaussian masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**, pp. 651-664 (1986).

[4] W.E.L. Grimson and E.C. Hildreth, "Comments on "Digital step edges from zero-crossings of second directional derivatives"," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **7**, pp. 121-124 (1985).

[5] R. Haralick, "Digital step edges for zero crossings of second directional derivatives," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, p. 58 (1984).

[6] E. C. Hildreth, "Implementation of a theory of edge detection," MIT Technical Report AI-TR-579 (April, 1980). Master's Thesis.

[7] James L. Crowley and Alice C. Parker, "A representation for shape based on peaks and ridges in the difference of low-pass transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, pp. 156-169 (1984).

[8] P. Burt and T. Adelson, "The laplacian pyramid as a compact image code," *IEEE Trans. on Communications*, p. 532 (1983).

[9] Shmuel Peleg, Orna Federbusch, and Robert Hummel, "Custom-made pyramids," in *Parallel Computer Vision*, ed. Len Uhr, Academic Press (1987).

[10] J. S. Chen, A. Huertas, and G. Medioni, "Very fast convolution with Laplacian-of-Gaussian masks," *Proceedings of the Conference on Computer Vision and Pattern Recogntion*, pp. 293-298 IEEE, (June, 1986).

[11] J. F. Canny, "Finding edges and lines in images," MIT AI Lab Technical Report 720 (1983).

[12] D. Marr, *Vision*, W. H. Freeman and Company (1982).

[13] James J. Clark, "Authenticating edges produced by zero crossing algorithms," Div. of Applied Sciences, Harvard University, Technical Report (Aug, 1986).

[14] V. Berzins, "Accuracy of Laplacian edge detectors," *Computer Vision, Graphics, and Image Processing* **27**, pp. 195-210 (1984).

[15] Robert Hummel, "Representations based on zero-crossings in scale-space," *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pp. 204-209 (June, 1986).