Computing Gaussian Blur

Robert Hummel and David Lowe

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

Abstract

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. It is important to evaluate the elements in these large kernels by integration of the continuous kernel in a region about the corresponding location, especially if the continuous function globally integrates to zero. We further consider zero-crossings of filtered images, and look at methods for selecting significant zero-crossing curves. Finally, we make some remarks on how to handle borders.

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 can be 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 tricks needed for efficient, accurate implementation of these kernels and interpretation of the results. There is little of any novelty in these tricks, and we use very little theory to justify comparative claims of effectiveness. Instead, we are simply concerned with a compendium of observations in implementing large kernels based on Gaussians. A much more thorough treatment, together with many further results, can be found in [2].

A Gaussian convolution is defined, in continuous variables, by

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

where

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}.$$

Here $f(x,y)$ is the input image, and $h(x,y)$ is the desired filtered image. We address the following questions: (1) How can the computation be discretized? (2) How can the discretized equation be computed most efficiently? (3) How should the borders be handled? That is, since $f(x,y)$ is not necessarily defined over all of $\mathbb{R}^2$, the convolution must be converted to a finite domain.

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. This can lead to serious inaccuracies, however. A particularly nasty consequence is that the effective (discrete) kernel, given by grid evaluations of the Gaussian $G(x,y)$, will no longer sum to one.

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:

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

where

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

Thus the discrete kernel elements should be obtained from block averages of the continuous kernel, and not be point evaluations. In particular, the resulting elements will much more nearly sum to one, in the case of Gaussian convolution.

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

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

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 need for averaging the continuous kernel rather than point evaluations is even more critical when Laplacian-of-Gaussian convolutions are attempted. We will shortly discuss alternate methods of performing these convolutions, but sometimes it is desired to perform a full image convolution against the analytically calculated kernel. In the case of Laplacian-of-Gaussian convolution, the kernel is given by:

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

Here C is a constant, which if a precise Laplacian-of-Gaussian is desired is equal to $1/(2\pi\sigma^4)$. The continuous kernel integrates to zero over $\mathbb{R}^2$, and it is highly desirable that the discrete kernel also sum to zero, to avoid any bias in subsequent

CH2342-4/86/0000/0910$01.00 © 1986 IEEE

zero-crossing operators. Accurate results are obtained only when the kernel is block averaged or weighted averaged, by formulas similar to those given above.

Analytic evaluation of the averages of the kernel are sometimes possible, but often lead to less fundamental functions. For example, block averages of the Gaussian are given by:

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

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

where erf is the error function associated with the normal distribution.

When analytic evaluation is impossible, or simply too much of a bother, the averaged kernel can 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. More sophisticated techniques can be used.

2. Iterated Discrete Convolution

Two computational tricks permit rapid discrete Gaussian convolution. The first trick, almost always used in practice, takes advantage of the fact that the Gaussian is a separable kernel. This has the consequence that a two-dimensional convolution can be performed by iterating two one-dimensional convolutions. That is, when the kernel $K(i,j)$ is separable, given say by

$$K(i,j) = k_1(i)\cdot k_2(j),$$

then the convolution can be performed according to the formula

$$\sum_{i'}\sum_{j'} K(i-i',j-j')f(i',j') = \sum_{i'} k_1(i-i')\left[\sum_{j'} k_2(j-j')f(i',j')\right].$$

Thus first the horizontal convolution against k_2 is performed first, and the result is convolved against the vertical kernel k_1. If the kernel is m by n, then this observation reduces the complexity from $O(m\cdot n)$ to $O(m+n)$. For the Gaussian kernel, both the continuous and discrete versions are separable.

The second trick applicable to Gaussian convolution makes use of the fact that the binomial coefficients form a rapid approximation to a normal distribution. It is well-known that the binomial coefficients, given by

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

approximate a normal distribution

$$\frac{1}{\sqrt{2\pi}\sigma}e^{-x^2/2\sigma^2},$$

where $x = k-n/2$, and $\sigma = (\sqrt{n}/2)$. Binomial coefficients are most easily calculated using Pascal's triangle, and lead to the design of an iterated convolution kernel, formed as follows. The initial data is $f(i)$, $i = \cdots,-1,0,1,\cdots$, (one dimension), which we set equal to $h_0(i)$. We define $h_N(i)$, for $N>0$, by

$$h_N(i) = \frac{h_{N-1}(i-1)+2h_{N-1}(i)+h_{N-1}(i+1)}{4}.$$

It follows that

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

This is a close approximation (for large N) of convolution against the Gaussian with $\sigma = \sqrt{N/2}$.

This method of Gaussian convolution has the advantage of yielding a finite computation, using simple arithmetic. Without the iterative method, the full convolution has to be performed using floating point or large-field fixed point arithmetic, and must be truncated at some arbitrary width. However, the asymptotics of the iterative approach are not as good, since roughly $3\cdot N$ operations per pixel are required, as compared to a convolution operation of $O(\sqrt{N})$ per pixel if the full convolution truncated to some fixed multiple of σ is performed.

Theoretically, if $f(i)$ is integer data represented by a fixed number of bits, then each successive computation of $h_N(i)$ should require two additional bits for accurate representation. However, as long as nearest-neighbor rounding is used, the errors are not too great when the iterative approach is used with fixed precision arithmetic at all levels of N.

3. 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 [3]. 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. In many schemes, such as the Burt Laplacian pyramid [4], 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 $(\sigma_1/\sigma_2) = 1.6$.

The fact is that for zero-crossings and many other applications, the DoG is better than the LoG. If a pyramid scheme is desired, then the Burt method is by far the one of choice. However, if a DoG is desired at a fixed resolution, then for maximum 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 the 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 $f(i,j)$. 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 blurring will quickly become zero

(3) Compute blurred versions of the data $f(i,j)$ using the functions $h_N(i,j)$ as described before, 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 [5]. 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

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

Note that in each of the two separable kernels, one of the two dimensions uses a Gaussian convolution.

4. Zero-crossings

To compute the zero crossings of a two-dimensional function $h(i,j)$, the image should be thresholded at zero, and border points identified in the resulting binary image. A border point can either be defined as a pixel that contains a pixel of a different binary value among its four (or eight) neighbors (in which case edges will be two pixels thick), or a "1" pixel with a "0" pixel among its eight 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 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.

Zero-crossings of the second directional derivative, where the direction of differentiation is the direction of the local gradient [6], has been suggested as a possible alternative for 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:

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

where a subscript denotes a partial derivative. The partial derivatives can be calculated using local three-by-three convolutions. The the discretized $h(i,j)$ is a sum of products of local convolutions of the image data. Haralick recommends that the partial derivatives of f be computed using the "facet model," although f could represent a smoothed version of the image data, smoothed by a Gaussian blur or some other operation.

Even when a DoG 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 by a 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 can be computed by taking a Sobel gradient magnitude of the Laplacian-of-Gaussian filtered image, and evaluating the result at the zero-crossings. 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. Of course, thresholding the results in this manner means that the resulting curves no longer form closed contours.

Alternatively, it is possible to "and" the results of finding zero-crossing contours at several successive scales of resolution. Providing the contours are thickened in all but one of the scales before the "and-ing," the result again is a collection of curves, not necessarily closed, but associated with prominent edges. This technique is essentially equivalent to the Sobel magnitude of the filtered data discussed above.

Finally, it is possible to combine a zero-crossing operator with a Sobel edge operator applied to the original image data, by simply "and-ing" the results of the zero-crossing operator with a thresholded Sobel magnitude image. The results again are similar to the techniques mentioned above.

5. Borders

All of the convolutions discussed so far assume that the original data $f(x,y)$ is defined for $(x,y) \in \mathbb{R}^2$ and that the grid of points (i,j) is an infinite lattice. Generally, however, images are defined on a finite rectangular grid of points, or sometimes on an irregularly shaped grid. Suppose that $f(i,j)$ is defined for $(i,j) \in \mathcal{D}$, where $\mathcal{D}$ is a domain of grid points. We define the boundary points of the domain as points in $\mathcal{D}$ with at least one of the eight neighbors outside of $\mathcal{D}$, and denote those points by $\partial\mathcal{D}$. How should the infinite convolutions be defined?

A usual tactic is to simply extend $f(i,j)$ to be zero for pixels outside of $\mathcal{D}$. For Gaussian convolution, however, there is a better approach, motivated by regarding Gaussian convolution as a means of solving the Heat equation [7]. Specifically, we use the formulas for iterative blurring:

$$\begin{aligned} 16h_{N+1}(i,j) = {} & h_N(i-1,j-1) + 2h_N(i-1,j) + h_N(i-1,j+1) \\ & + 2h_N(i,j-1) + 4h_N(i,j) + 2h_N(i,j+1) \\ & + h_N(i+1,j-1) + 2h_N(i+1,j) + h_N(i+1,j+1) \end{aligned}$$

for $(i,j) \in \mathcal{D} \setminus \partial\mathcal{D}$, and $h_{N+1}(i,j) = h_N(i,j)$ for $(i,j) \in \partial\mathcal{D}$. Although this yields something different than Gaussian blurring, the result is an appropriate blurring of the image data. Difference-of-Gaussian filters can then be computed. Note that the DOG filters, when defined this way, will always give zero values on the boundary.

Acknowledgements

This research was supported by Office of Naval Research Grant N00014-85-K-0077 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] J. Crowley, "A representation for visual information," CMU Robotics Institute, Ph.D. Thesis (1982).

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

[5] 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 Recognition*, pp. 293-298 IEEE, (June, 1986).

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

[7] 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).