Introduction to Computer Vision

Edge Detection

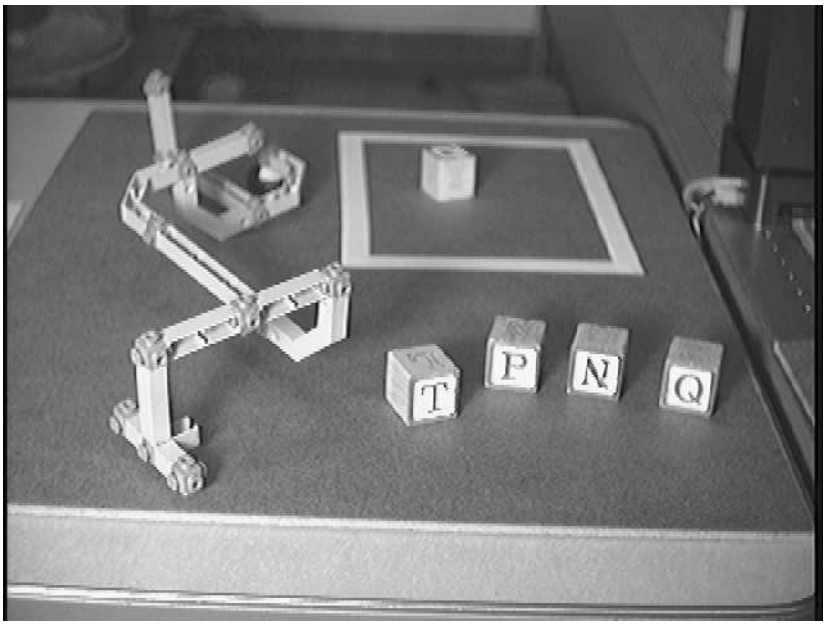
1 Introduction

The goal of edge detection is to produce something like a line drawing of an image. In practice we will look for places in the image where the intensity changes quickly. Observe that, in general, the boundaries of objects tend to produce suddent changes in the image intensity. For example, different objects are usually different colors or hues and this causes the image intensity to change as we move from one object to another. In addition, different surfaces of an object receive different amounts of light, which again produces intensity changes. Thus, much of the geometric information that would be conveyed in a line drawing is captured by the intensity changes in an image. Unfortunately, there are also a large number of intensity changes that are *not* due to geometry, such as surface markings, texture, and specular reflections. Moreover there are sometimes surface boundaries that do not produce very strong intensity changes. Therefore the intensity boundary information that we extract from an image will tend to indicate object boundaries, but not always (in some scenes it will be truly awful).

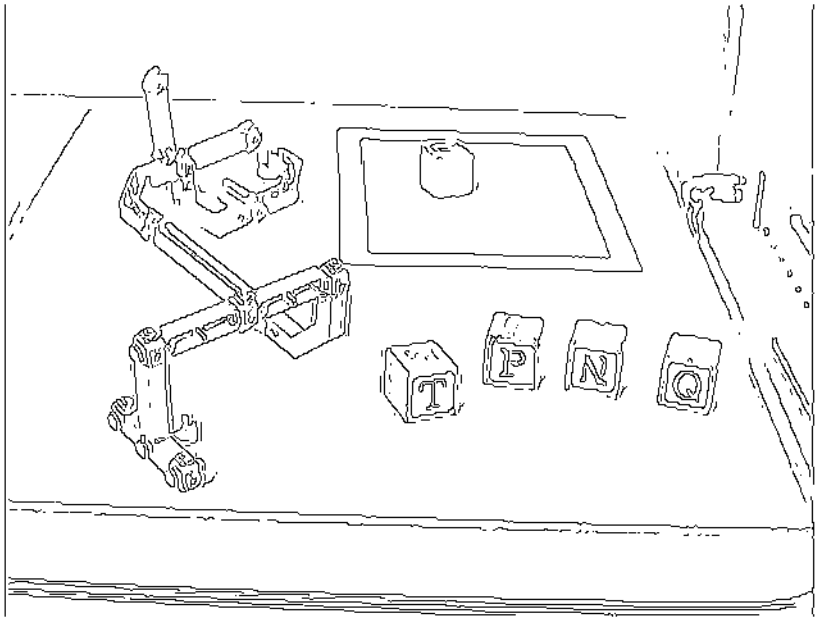
Figure 1a is an image of a simple scene, and Figure 1b shows the output of a particular edge detector on that image. We can see that in this case the edge detector produces a relatively good 'line drawing' of the scene. However, say we change the lighting conditions, as illustrated in Figure 2a. Now the edge detector produces an output that is substantially less like a line drawing, as shown in Figure 2b. Our first lesson is that the illumination conditions can make a significant difference in the information that an edge detector extracts from an image. Figure 3 shows some more edge detection results.

2 Local Operators

In this section we will look at some simple local operators for identifying edges, and consider some of their limitations. Lets start with a one-dimensional example. A simple model of an edge is illustrated in Figure 4a. This signal has low constant value up to a point and a higher constant value after the edge location. To identify the edge location we can consider

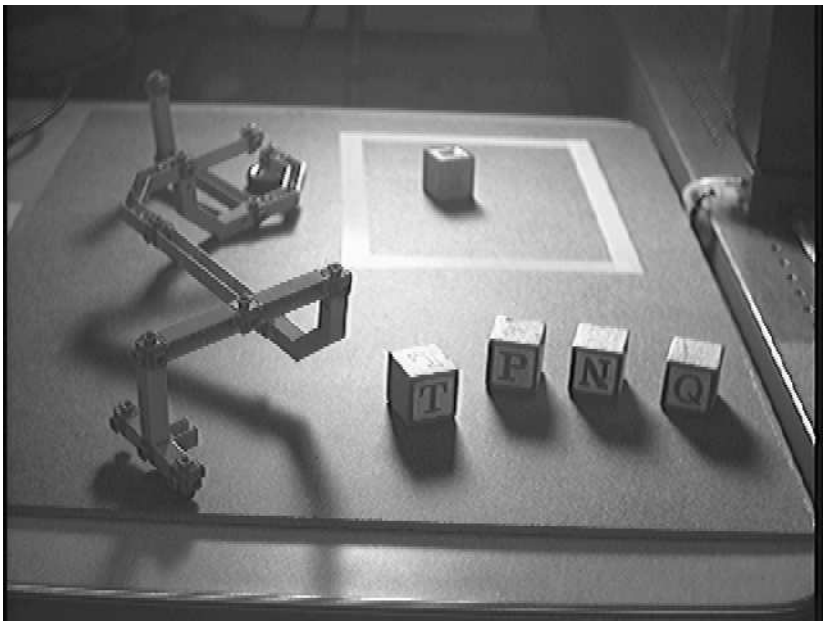


(a)

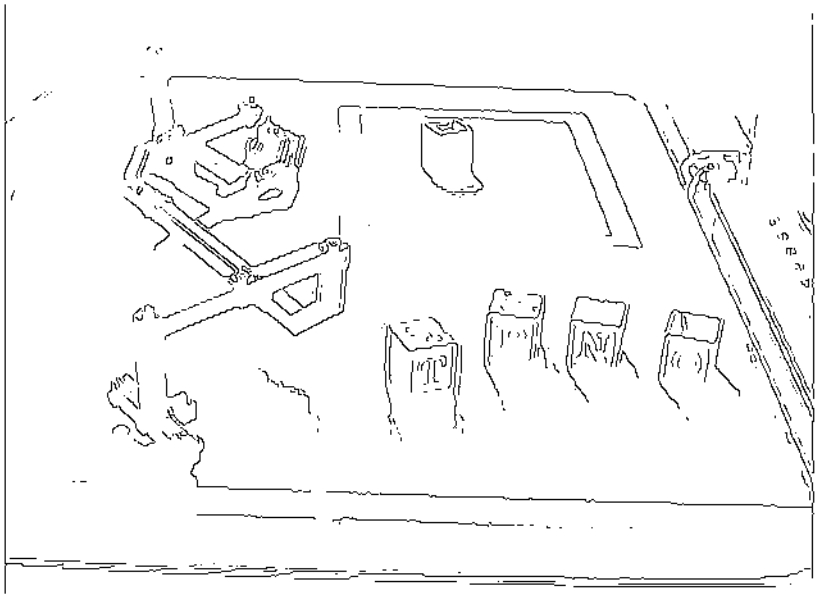


(b)

Figure 1: An image of a simple scene and the edges extracted.



(a)



(b)

Figure 2: A less good image of the simple scene and the edges extracted.

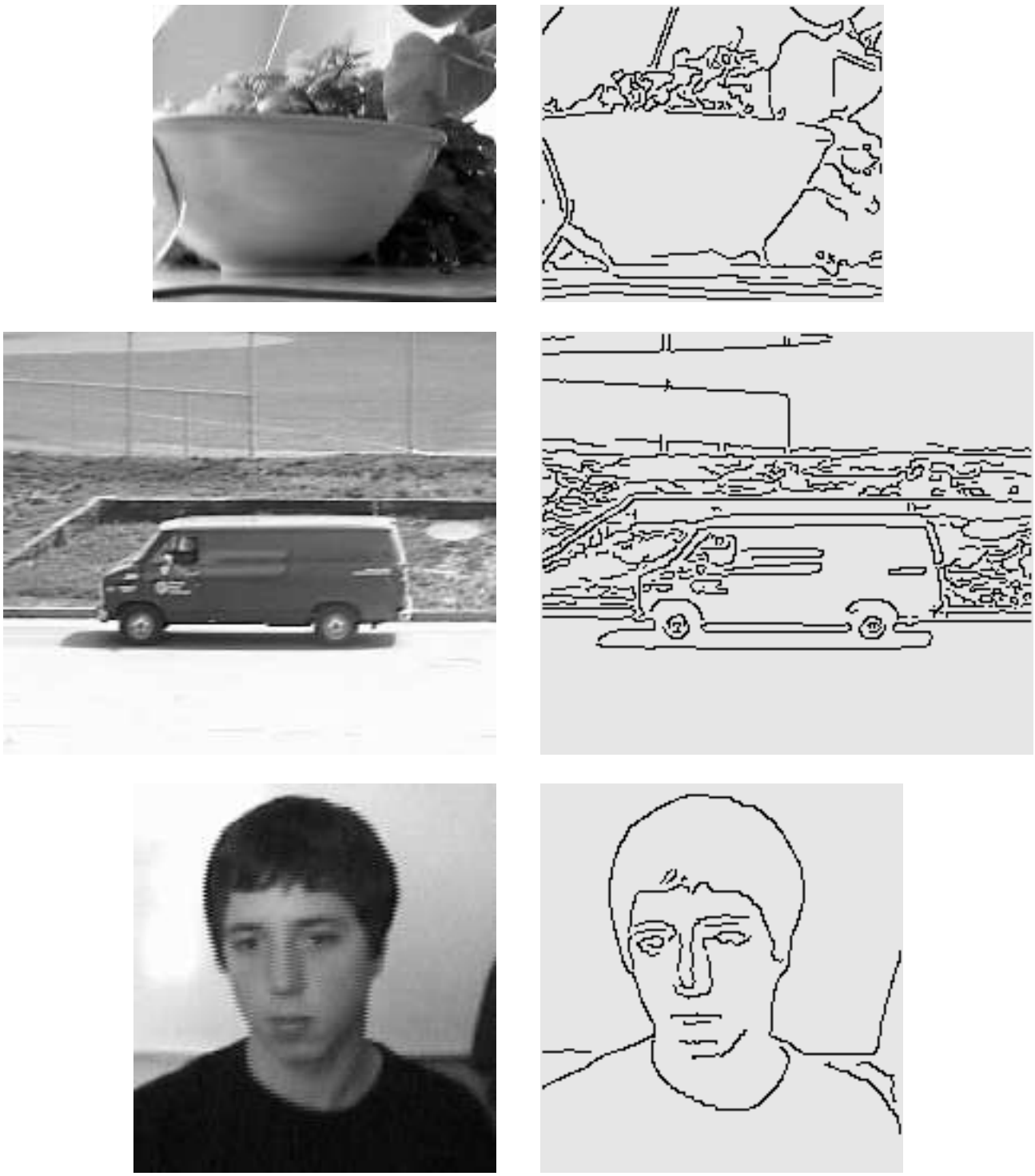
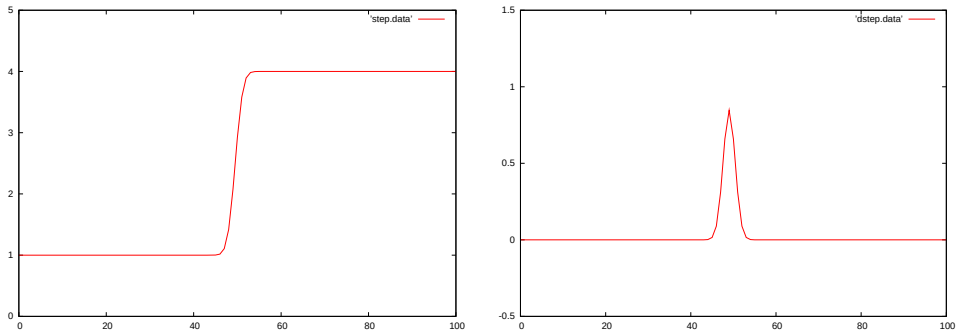


Figure 3: Example edge detection results.



(a) (b)

Figure 4: One-dimensional step edge and its derivative.

the first derivative of the signal as shown in Figure 4b. Note how the edge shows up as a peak in the first derivative.

Now consider an idealized continuous image $I(x, y)$, which denotes intensity as a function of location. In order to identify edges, we are interested in finding regions of the image where the intensity changes fast. It is natural to use the *gradient* of the image intensity,

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right).$$

This is a vector valued quantity which can be thought of intuitively as pointing in the direction of steepest increase in intensity at each location.

The magnitude of the gradient reflects how rapidly the intensity is changing in the direction of steepest increase. Actually we usually use the squared gradient magnitude, to avoid computing square roots,

$$\|\nabla I\|^2 = \left(\frac{\partial I}{\partial x} \right)^2 + \left(\frac{\partial I}{\partial y} \right)^2.$$

One nice property of the gradient magnitude is that it is rotationally symmetric, or *isotropic*. This means that it is insensitive to the orientation of the gradient vector, which is often desirable. In other words, an edge is an edge – regardless of the orientation of that edge with respect to the Cartesian coordinate system. Two less desirable properties of the gradient magnitude are that it is a *nonlinear* operator, and it loses information about which side of an edge is brighter (due to the squaring which loses sign information).

Finite Approximations

In computer vision we do not have continuous image functions $I(x, y)$ that we can differentiate in order to compute quantities such as the gradient magnitude. An image is digitized

both in space and in intensity, producing an array $I[x, y]$ of 'intensity values'. These intensities are generally integers in some range, and their magnitude usually reflects the brightness level (0 is darkest and the maximum value is brightest).

In practice we can use finite difference approximations to estimate derivatives. For example, for a discrete one-dimensional sampled function, represented by a sequence of values (an array) $F[x]$, we know that

$$\frac{dF}{dx} \approx F[x+1] - F[x],$$

and

$$\frac{d^2F}{dx^2} \approx F[x-1] - 2F[x] + F[x+1].$$

Directly estimating the image gradient using the finite difference approximation shown above has the undesirable effect of causing the horizontal and vertical derivatives to be estimated at different points in the image. Instead we can use

$$\frac{\partial I}{\partial \hat{x}} \approx I[x+1, y] - I[x-1, y],$$

and

$$\frac{\partial I}{\partial \hat{y}} \approx I[x, y+1] - I[x, y-1].$$

So the squared gradient magnitude is

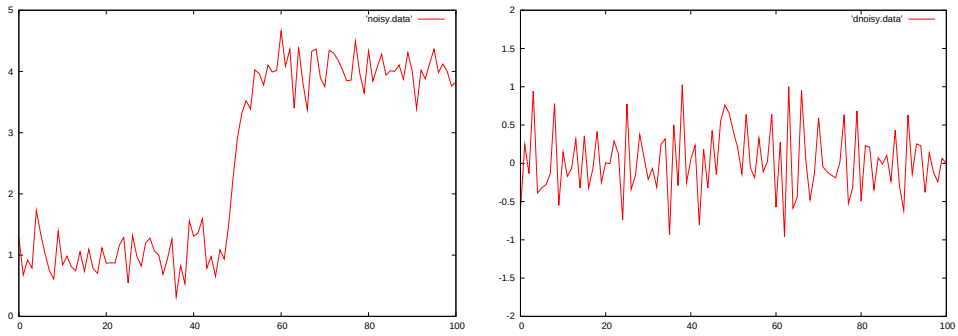
$$\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2 \approx (I[x+1, y] - I[x-1, y])^2 + (I[x, y+1] - I[x, y-1])^2. \qquad (1)$$

Now we can write a simple program to compute the squared gradient magnitude:

```
for x from 1 to xmax − 1
  do for y from 1 to ymax − 1
   do M[x, y] = (I[x + 1, y] − I[x − 1, y])^2 + (I[x, y + 1] − I[x, y − 1])^2
```

This yields an array $M[x, y]$ with 'large' values where the intensity is changing quickly. Note that the value of $M[x, y]$ is undefined for $x = 0$, $x = xmax$, $y = 0$ or $y = ymax$ because otherwise we would access beyond the boundaries of $I[x, y]$.

We can *threshold* the array $M[x, y]$ to locate edges. This is the first of many 'sounds good but doesn't work' stories in computer vision. The problem is that derivatives are very sensitive to local variations. For example, Figure 5a shows a noisy version of a one-dimensional edge, and Figure 5b shows the first derivative of this noisy signal. Note how the peaks in the derivative are completely meaningless.



(a) (b)

Figure 5: Noisy version of a one-dimensional step edge and its derivative.

The central problem with local operators is that there is substantial local variation in a digitized image, and much of this variation is 'noise' rather than information regarding the scene. In practice this local variability in an image causes edges to appear nearly everywhere. Figure 6 shows the result of running a local gradient magnitude edge detector (similar to the simple program given above) on the image shown in Figure 1a. Contrast this with the edges from Figure 1b to see that we can do much better (with some non-local, or at least less local, edge operators).

3 Convolution and Smoothing

We saw in the previous section that to detect edges in an image, we can use spatial derivatives. In particular we considered the squared gradient magnitude,

$$\|\nabla I\|^2 = \left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2.$$

One problem, however, is that images contain a lot of high-frequency 'noise'. We saw the effects of this in Figure 6, where the application of a local gradient magnitude edge detector yielded many small edges.

This problem can be abstracted with the following example. Consider an array of numerical values,

5	6	7	6
6	5	6	5
5	5	6	7
7	6	5	6

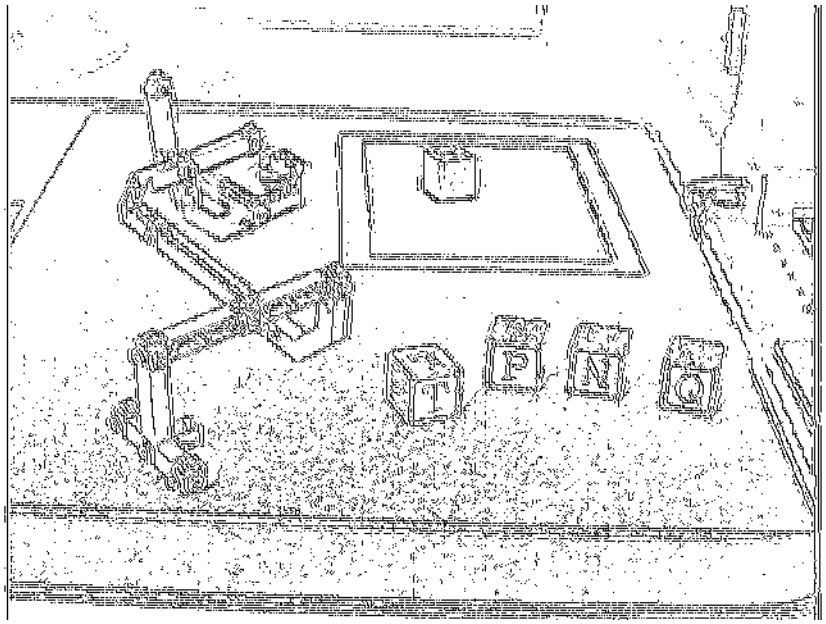


Figure 6: Local edges extracted from the image of the simple scene.

This array has local 'boundaries' all over the place – nearly none of the neighboring values are the same. Yet, the changes are very small, and there are not 'regions' of similar value. What we would like is to produce a different array, such as

6	6	6	6
6	6	6	6
6	6	6	6
6	6	6	6

from the original array. This process of removing spatial high frequencies (small local changes) is known as *lowpass filtering*. We will discuss a particular method of lowpass filtering, which is done by *convolving* an array (in this case the image) with a filter (generally a Gaussian operator, or normal distribution). Note that the noisy array should be distinguished from an array of values such as

5	5	5	6
5	5	6	6
5	5	6	6
5	6	6	6

where the differences are still relatively small, but there are two distinct regions (of value 5 and value 6). We don't want the filtering operation to remove these sorts of distinctions,

and in fact we will be able to do this by appropriately selecting a *scale* or spatial extent for the lowpass filtering operation.

3.1 Discrete Convolution

Consider the sampled function $h[x, y]$ defined in terms of two other sampled functions $f[x, y]$, and $g[x, y]$,

$$h[x, y] = \sum_i \sum_j f[i, j]g[x - i, y - j]. \tag{2}$$

Intuitively each value $h[x, y]$ is obtained by 'placing' g shifted by (x, y) over f, and summing the products of the individual entries of f and g. Usually both f and g are non-zero over a finite domain so h is defined in terms of finite sums. We say that h is the convolution of f and g, which is written as,

$$h = f \otimes g.$$

Convolution is *commutative* and *associative*,

$$a \otimes b = b \otimes a.$$

$$(a \otimes b) \otimes c = a \otimes (b \otimes c).$$

These two properties are very useful because they allow us to rearrange computations in whatever fashion is most convenient (or efficient).

3.2 Smoothing Using Convolution

Now we want to use the convolution operator to smooth an image. A simple way to smooth an image is to average together neighboring values. This can be accomplished by convolution with an $n \times n$ mask that has the value $1/n^2$ at each location. For example, a four by four version of such a mask would be

1/16	1/16	1/16	1/16
1/16	1/16	1/16	1/16
1/16	1/16	1/16	1/16
1/16	1/16	1/16	1/16

Convolving an image with this mask computes the average value of the image over every possible 4×4 neighborhood (by simply summing 1/16 of each of 16 values). However, the sudden truncation of the mask has unfortunate frequency domain effects — it causes high-frequency noise. While a formal derivation of this fact is beyond the scope of these notes,

intuitively one can see why this is true. As the mask is shifted across the image, noise at the edges of a given mask position will produce significant noise in the output – because the noisy value will be part of the sum at one position of the mask, and then completely absent from the sum at the next position of the mask. One way to avoid this problem is to weight the contributions of values farther from the center of the mask by less.

We use a Gaussian to do this weighting (other functions work too). In one dimension, the Gaussian is given by

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}.$$

This is the canonical 'bell-shaped' or 'normal' distribution as used in statistics. The maximum value is attained at $G_\sigma(0)$, the function is symmetric about 0, and $\int G_\sigma(x)dx = 1$ (the area under the function is 1). The parameter σ controls the 'width' of the curve – the larger the value of σ the slower the function approaches 0 as $x \to \infty$ (and $x \to -\infty$).

In two dimensions, the Gaussian is defined as

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

In the discrete case we wish to form some sort of mask, analogous to the 'averaging' mask considered above that had values of $1/n^2$ at each location. While the one-dimensional Gaussian is defined across the entire domain $[-\infty, \infty]$, in practice we can truncate the function after the value of G_σ becomes very small. In general we use $[-4\sigma, 4\sigma]$, because the value of $G_\sigma(4\sigma)$ is very close to zero. Note that if we truncate the Gaussian too quickly (say at $\pm 2\sigma$), then we get the same effect that occurs when we simply average – the values near the edge of the mask are relatively large and thus a good deal of high frequency noise is produced when the mask is shifted from one position to the next.

The discretely sampled function, or mask, should sum to 1 just like in the continuous case (where the integral over the entire domain is 1). The simplest way to do this is to compute the un-normalized Gaussian

$$\mathcal{G}_\sigma[x, y] = e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

for each grid point (x, y), where x and y range over $\pm 4\sigma$ (i.e., the origin $(0, 0)$ is at the center of the mask). Then each element is divided by the sum of all the elements, in order to yield a normalized mask which sums to 1. That is, $G_\sigma[x, y] = \mathcal{G}_\sigma[x, y]/S$, where $S = \sum_x \sum_y \mathcal{G}_\sigma[x, y]$.

For example, if $\sigma = 0.5$ then we obtain a 5×5 mask (with center at $(0, 0)$ and out to $\pm 4\sigma = \pm 2$). The values sum to 1 and the entries are symmetric about the origin:

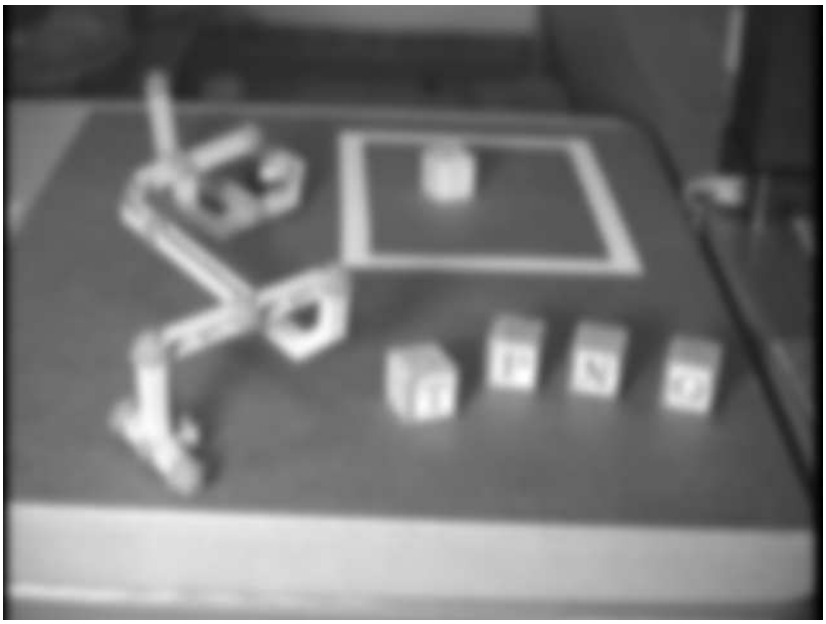


Figure 7: lowpass filtering (smoothing) an image with a Gaussian filter.

6.96E-8	2.80E-5	2.07E-4	2.80E-5	6.96E-8
2.80E-5	0.0113	0.0837	0.0113	2.80E-5
2.07E-4	0.0837	0.618	0.0837	2.07E-4
2.80E-5	0.0113	0.0837	0.0113	2.80E-5
6.96E-8	2.80E-5	2.07E-4	2.80E-5	6.96E-8

The convolution of an image with the mask G_σ is given by,

$$I_s[x,y] = \sum_{i=-4\sigma}^{4\sigma} \sum_{j=-4\sigma}^{4\sigma} I[x-i, y-j] G_\sigma[i,j]. \tag{3}$$

This resulting array $I_s[x,y]$ is a lowpass filtered (or smoothed) version of the of the original image I. Figure 7 shows a smoothed version of the image from Figure 1, where the image has been convolved with a Gaussian of $\sigma = 4$. Compare it with the original image.

There is a tradeoff between the size σ of G_σ and the ability to spatially locate an event. If σ is large, we no longer really 'know' where some event (such as a change in intensity) happened, because that event has been smoothed out by a factor related to σ.

Efficiently computing $G_\sigma \otimes I$

A 'direct' implementation of discrete convolution, as shown in equation (3) requires $O(m^2n^2)$ operations for an $m \times m$ mask and an $n \times n$ image. The mask is positioned at each of the n^2

image locations and m^2 multiply and add operations are done at each position. In the case of Gaussians, the operator is *separable* and we can use this fact to speedup the convolution to $O(mn^2)$. This is a significant savings, both theoretically and in practice, over the direct implementation. Any smoothing method (or edge detector) that uses separable filtering operators should be implemented in this manner.

We note that

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

where

$$G_x(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}},$$

and

$$G_y(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-y^2}{2\sigma^2}}.$$

In other words, the rotationally symmetric Gaussian, G, is just the product G_xG_y of two one-dimensional Gaussians. How do we use this fact to speed up the computation? We further note that $G(x,y) = G_x^*(x,y) \otimes G_y^*(x,y)$, where

$$G_x^*(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-1}{2}\left(\frac{x^2}{\sigma^2}\right)} \delta(y)$$

and

$$G_y^*(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-1}{2}\left(\frac{y^2}{\sigma^2}\right)} \delta(x)$$

and δ is the unit impulse (or Dirac delta function). In effect we are using the delta function as a technical trick to turn the product G_xG_y into the convolution $G_x^* \otimes G_y^*$. Recall that convolution integrates the product of two functions in order to produce one element of the output function, so the delta function can be thought of as selecting a single element of this product (by multiplying all the others by zero).

We see that,

$$G \otimes I = (G_x^* \otimes G_y^*) \otimes I = G_x^* \otimes (G_y^* \otimes I),$$

by the fact that $G = G_x^* \otimes G_y^*$ and the associativity of the convolution operator. This means that in order to compute the convolution of G with I, we can instead first convolve the image with G_x^* and then convolve the result with G_y^*. Why is this an advantage? Because both G_x^* and G_y^* are essentially 'one-dimensional' (due to the delta function in the definition of these functions). It is easiest to see this in the discrete case, so we now turn to that.

When g is an $m \times m$ mask forming a discrete sampling of G, we can write $g = g_x^* \otimes g_y^*$ where g_x^* is a discrete version of G_x^* and similarly g_y^* is a discreve version of G_y^*. The array

g_x^* has one row of nonzero entries g_x, and g_y^* has one column of nonzero entries g_y. Both g_x and g_y correspond to a one-dimensional gaussian mask. Diagrammatically,

$$\begin{bmatrix} 0 \\ [\ g_x\ ] \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & \begin{bmatrix} \\ g_y \\ \\ \end{bmatrix} & 0 \end{bmatrix} = \begin{bmatrix} \\ g \\ \\ \end{bmatrix}.$$

Thus, the convolution $G \otimes I$ can be done by first convolving I with a column vector g_y and then convolving the result with a row vector g_x. The two-dimensional convolution is replaced by two one-dimensional convolutions. Each of the one-dimensional convolutions involves $O(mn^2)$ operations, and thus the overall time to compute the convolution has been reduced from $O(m^2n^2)$ to $O(mn^2)$ by this method.

In practice, the method of smoothing by two successive convolutions with one-dimensional masks is much faster than convolution with a two-dimensional mask. For example, when $\sigma = 1$ the two-dimensional mask (out to $\pm 4\sigma$) is 9×9, and thus the savings is a factor of about 5 (two 1×9 masks as opposed to one 9×9 mask). For $\sigma = 2$, it is about 10 times faster. Recall, however, that this method only works for functions that can be decomposed into the product of two one-dimensional functions (such as the Gaussian).

4 Canny Edge Operator

The Canny [1] edge detector is based on the gradient magnitude of a smoothed image: local maxima of the gradient magnitude that are high are identified as edges. The motivation for Canny's edge operator was to derive an 'optimal' operator in the sense that it,

- Minimizes the probability of multiply detecting an edge.
- Minimizes the probability of failing to detect an edge.
- Minimizes the distance of the reported edge from the true edge

The first two of these criteria address the issue of *detection*, that is, given that an edge is present will the edge detector find that edge (and no other edges). The third criterion addresses the issue of *localization*, that is how accurately the position of an edge is reported. There is a tradeoff between detection and localization – the more accurate the detector the less accurate the localization and vice versa.

Given an image I, the Canny edge detector considers the squared gradient magnitude of the smoothed image,

$$m(x,y) = \|\nabla(G \otimes I)\|^2 = \left(\frac{\partial I_s}{\partial x}\right)^2 + \left(\frac{\partial I_s}{\partial y}\right)^2.$$

Instead of thresholding $m(x,y)$ in order to identify edges, we should look for peaks in the magnitude and then thresholds the peaks. The justification for this is that edge locations will have a higher gradient magnitude than non-edges (i.e., will be local peaks in the gradient). However, rather than comparing the gradient magnitude at a given location to the values at all neighboring locations, Canny observes that we are only concerned that the gradient be a 'peak' with respect to its neighbors in the gradient direction, and not in other directions. To help understand this, consider the analogy of being on a ridge of a mountain range. The gradient direction points down from the 'top' of the ridge toward the valleys on either side. The ridge itself may move up and down such that at a given location we can be standing on the ridge but not be at a local peak in all directions. In the gradient direction, however, we are at a peak whenever we are on the ridge.

The unit vector in the gradient direction is given by

$$\hat{\nabla}(x,y) = \frac{\nabla(I_s)}{||\nabla(I_s)||}.$$

If we denote $\hat{\nabla}(x,y)$ by the unit vector $(\delta x, \delta y)$, then $m(x,y)$ is defined to be a local peak in the gradient direction when

$$m(x,y) > m(x+\delta x, y+\delta y),$$

and

$$m(x,y) > m(x-\delta x, y-\delta y).$$

That is, we say that $m(x,y)$ is a local peak whenever it is greater than the values in the gradient direction and the opposite of the gradient direction. Canny calls this local peak detection operation *non-maximum suppression* (NMS). Note that in practice it is generally better to use $>$ in one direction and $\geq$ in the other, rather than $>$ in both directions, to allow for detection of 'wide' edges as peaks.

The NMS operation still leaves many local 'peaks' that are not very large. These are then thresholded based on the gradient magnitude (or strength of the edge) to remove the small peaks. The peaks that pass this threshold are then classified as edge pixels. Canny uses a thresholding operation that has two thresholds, lo and hi. Any local maximum for which

$m(x, y) >$ hi is kept as an edge pixel. Moreover, any local maximum for which $m(x, y) >$ lo and some neighbor is an edge pixel is also kept as an edge pixel. Note that this is a recursive definition — any pixel that is above the low threshold and adjacent to an *edge* pixel is itself an *edge* pixel. This form of using two thresholds allows the continuation of weaker edges that are connected to strong edges, and is a form of hysteresis.

To summarize the steps of the Canny edge detector, for a discrete image $I[x, y]$,

1. Smooth the image using a 2D Gaussian, $I_s = G_\sigma \otimes I$.

2. Compute the gradient and squared gradient magnitude of the smoothed image,

$$\nabla I_s = \left(\frac{\partial I_s}{\partial x}, \frac{\partial I_s}{\partial y} \right)$$

$$m(x, y) = \left(\frac{\partial I_s}{\partial x} \right)^2 + \left(\frac{\partial I_s}{\partial y} \right)^2$$

3. Use the unit vector $\frac{\nabla I_s}{|\nabla I_s|} = (\delta x, \delta y)$ at each point to estimate the gradient magnitude in the gradient direction and opposite of the gradient direction. This can be done by a weighted average of the neighboring pixels in each direction or simply selecting the neighboring pixel closest to each direction $(x + \delta x, y + \delta y)$ and $(x - \delta x, y - \delta y)$.

4. Let $p = m(x, y)$, $p_+ = m(x + \delta x, y + \delta y)$, $p_- = m(x - \delta x, y - \delta y)$. Define a peak as

$$(p > p_+ \wedge p \geq p_-) \vee (p > p_- \wedge p \geq p_+) \ .$$

5. Threshold 'strong' peaks in order to get rid of little peaks due to noise, etc. Use $m(x, y)$ as measure of edge strength. Use a hysteresis mechanism as described above with two thresholds on edge strength, lo and hi.

The edges shown in this handout are from the Canny edge detector. In practice, this edge operator (or variants of it) is the most useful and widely used.

5 Multiscale Processing

A serious practical problem with any edge detector is the matter of choosing the *scale* of smoothing (the value of σ to use). For many scenes, using the Canny edge detector with $\sigma = 1$ seems to produce 'good' results, but this is not very satisfactory. Clearly, as σ increases

less and less of the detailed edges in an image are preserved (and spatial localization gets worse and worse). For many applications it is desirable to be able to process an image at multiple scales, in order to determine which edges are most significant in terms of the range of scales over which they are observed to occur.

Witkin [3] has investigated more thoroughly the idea of multi-scale signals derived from smoothing a signal with a Gaussian at different scales. He calls this *scale space* — which is a function defined over the domain of the original function, plus another dimension corresponding to the scale parameter. For example, say we have an image $I(x, y)$. The corresponding scale-space function is

$$\mathcal{I}(x, y, \sigma) = I(x, y) \otimes G_\sigma(x, y),$$

where σ is the scale parameter.

A number of natural structures can be extracted from the scale-space function, the most common of which is the 'edges' at each scale. These edges can be identified as extrema of gradient magnitude (as in the Canny operator) or as zero crossings of the Laplacian (as in the Marr-Hildreth operator). In either case, the result is a binary-valued function (or space) $\mathcal{E}(x, y, \sigma)$. Figure 8 shows a grey-level image and the edges at various scales of smoothing.

Note that as we would expect, for larger values of σ there are fewer edges (extrema of the first derivative of the smoothed function), and the spatial localization of the edges becomes poorer. Another interesting observation is that the edges do not simply appear or disappear at random as the scale parameter changes. Edges tend to form 'tree' structures, with the root at the largest value of sigma for which a boundary appears. This in principle allows a given edge to be tracked across scales (from coarse to fine), although this edge tracking problem is not easy. Note that scale space edges still do not solve the problem of what scale to pick! One option is to do further processing using the entire scale space tree, and thus avoid the problem of ever picking a scale of processing.

The scale space approach does partially address the issues of the tradeoff between detectability and localization. As σ gets smaller, the localization gets better and the detection gets worse. With the scale space representation we in part get the best of both worlds because it is possible to pick a value of σ where the detection is good, and then follow the edge contour down in scale until the localization is also good. The problem with this approach is when a given edge contour splits into two disjoint edges as σ decreases, because then we are left with the issue of which path to choose. Over ranges of σ where a contour does not split, however, this is a good technique for overcoming the detection/localization tradeoff.

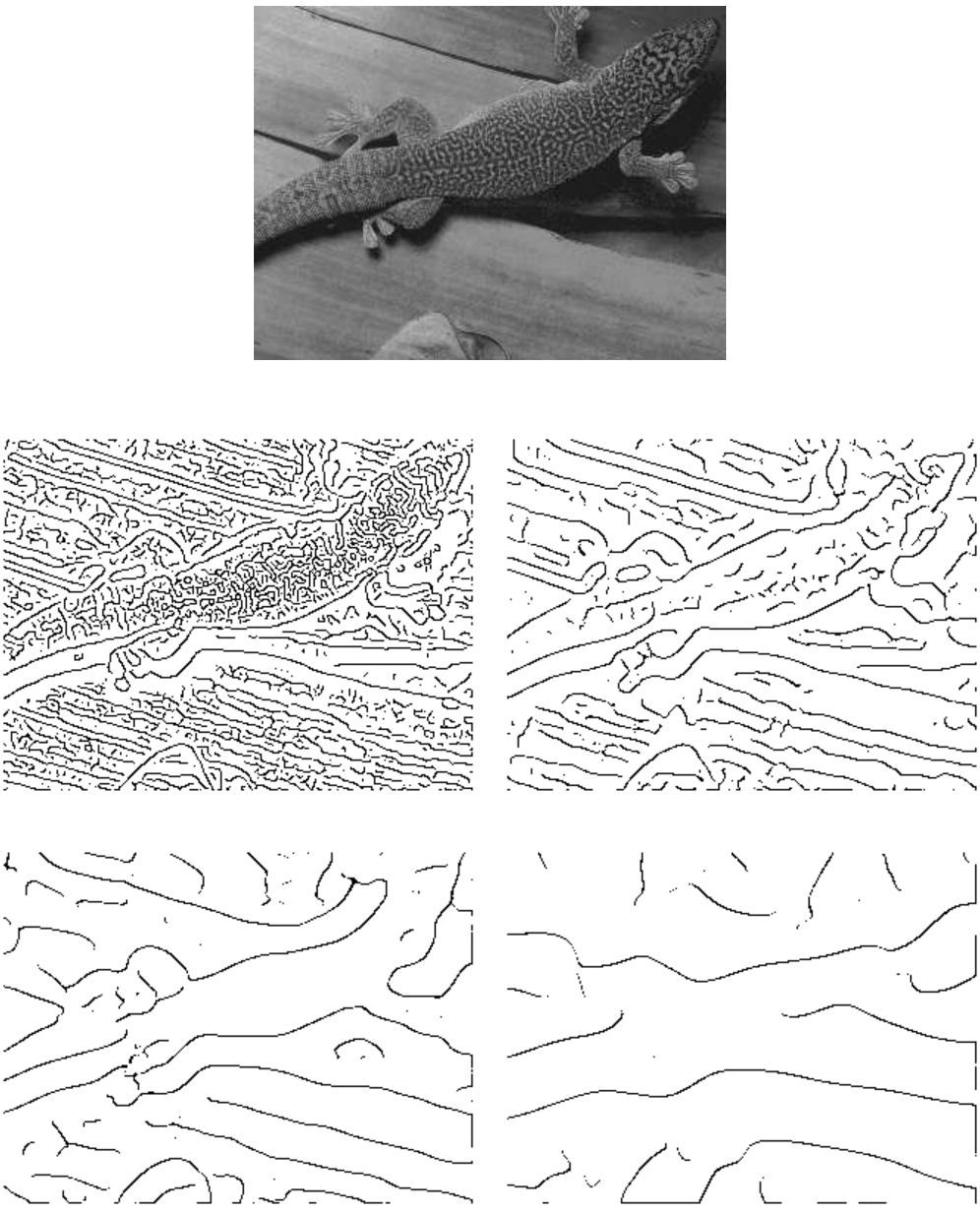


Figure 8: Canny edges at multiple scales of smoothing, $\sigma = 2, 4, 8, 16$. The scale-space edges are a 'stack' of these images (for all values of σ, not just those shown here) forming a three-dimensional binary function $\mathcal{E}(x, y, \sigma)$.

References

[1] J.F. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, pp. 34-43, 1986.

[2] D. Marr and E. Hildreth, "Theory of Edge Detection", *Proc. of the Royal Society of London B*, Vol. 207, pp. 187-217, 1980.

[3] A.P. Witkin, "Scale Space Filtering", *Proc. of International Joint Conference on Artificial Intelligence*, pp. 1019-1022, August 1983.