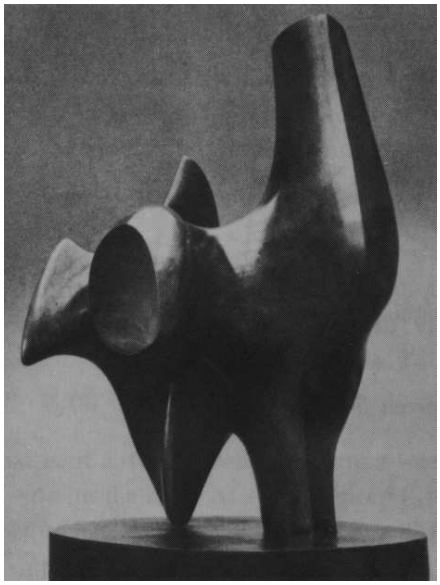
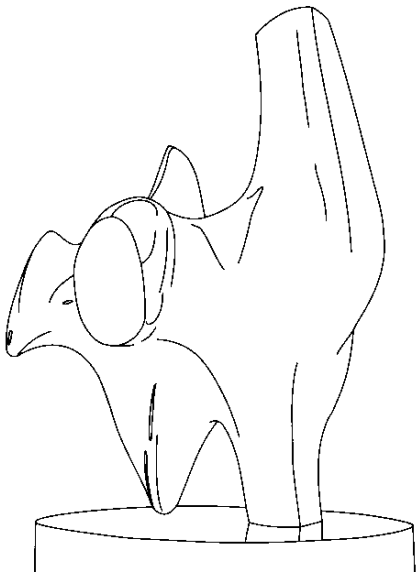
University of Cambridge
Engineering Part IB

Paper 8 Information Engineering [10mm]
Handout 2: Feature Extraction





Roberto Cipolla
May 2017

Image Intensities

We can represent a monochrome image as a matrix $I(x, y)$ of intensity values. The size of the matrix is typically 320×240 (QVGA), 640×480 (VGA) or 1280×720 (HD) and the intensity values are usually sampled to an accuracy of 8 bits (256 **grey levels**). For colour images each colour channel (e.g. RGB) is stored separately.

If a point on an object is visible in view, the corresponding pixel intensity, $I(x, y)$ is a function of many geometric and photometric variables, including:

1. The position and orientation of the camera;
2. The geometry of the scene (3D shapes and layout);
3. The nature and distribution of light sources;
4. The reflectance properties of the surfaces: specular $\leftrightarrow$ Lambertian, albedo 0 (black) $\leftrightarrow$ 1 (white);
5. The properties of the lens and the CCD.

In practice the point may also only be partially visible or its appearance may be also be affected by occlusion.

Data reduction

With current computer technology, it is necessary to discard most of the data coming from the camera before any attempt is made at real-time image interpretation.

images	$\rightarrow$	generic salient features
10 MBytes/s (mono CCD)		10 KBytes/s

All subsequent interpretation is performed on the generic representation, not the original image. We aim to:

- Dramatically reduce the amount of data.
- Preserve the useful information in the images (such as the albedo changes and 3D shape of objects in the scene).
- Discard the redundant information in the images (such as due to the lighting conditions).

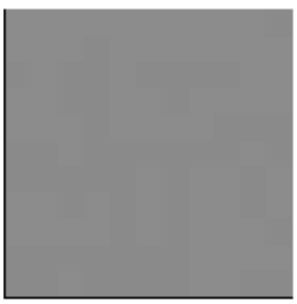
We would also like to arrive at a *generic* representation, so the same processing will be useful across a wide range of applications.

Image structure

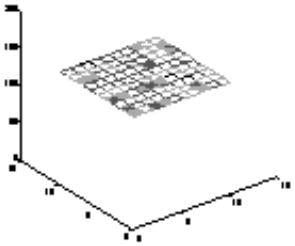
The answer becomes apparent if we look at the structure of a typical image. In this photo of "Claire", we'll examine the pixel data around several patches: a featureless region, an edge and a corner.

0D

The featureless region is characterized by a smooth variation of intensities.



140 141 141 140 141 141 140 141 141 141 139
140 141 141 139 140 140 140 140 140 140 140
139 140 139 139 140 139 139 138 139 140 140
140 140 139 139 140 140 138 140 140 140 140
140 140 141 139 141 141 140 140 139 139 138
139 139 141 139 139 139 139 139 139 140 139
139 139 139 139 139 140 139 140 140 139 141
140 140 139 140 139 141 139 140 140 139 140
140 140 140 140 139 140 139 140 131 131 139
139 139 138 139 139 139 139 140 141 140 140
139 139 140 139 139 139 139 140 140 139 139

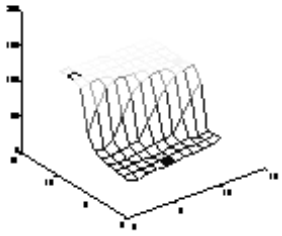


Edges and corners

1D

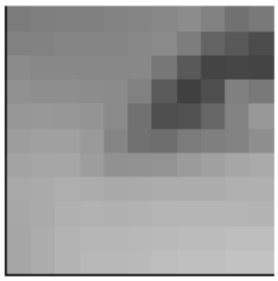
The patch containing the edge reveals an intensity discontinuity in one direction.

136	136	135	134	135	134	134	135	135	136	135
136	135	134	134	136	135	133	133	135	135	136
136	135	134	134	133	133	134	134	134	134	135
130	133	134	133	132	132	132	133	133	132	133
73	103	127	135	134	133	134	133	132	133	134
54	52	60	88	114	127	133	134	132	133	132
49	48	50	53	56	76	99	117	130	133	135
46	45	45	48	50	53	55	57	77	99	118
44	44	45	46	45	47	50	52	54	49	54
42	43	43	44	46	47	50	51	50	52	55
41	40	44	44	44	46	49	48	50	51	56

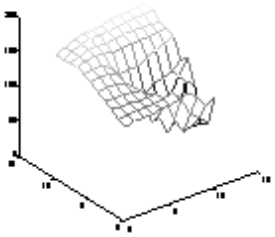


2D

The patch containing the corner reveals an intensity discontinuity in two directions.



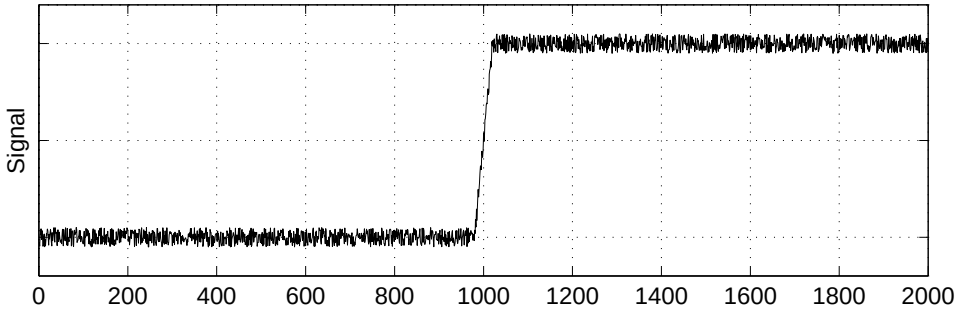
124	127	128	128	131	312	135	140	130	113	121
129	130	132	132	133	135	139	125	99	89	76
138	136	137	135	135	136	130	89	68	71	69
144	142	143	141	139	129	92	64	78	128	121
150	152	151	148	138	113	79	81	102	131	152
158	160	160	154	133	113	111	123	127	131	143
163	165	166	158	145	146	147	155	160	166	171
168	171	174	173	169	171	172	173	173	176	176
167	171	176	177	176	178	180	181	181	180	179
166	173	179	182	182	184	185	187	188	189	190
166	173	179	183	183	185	189	191	191	194	194



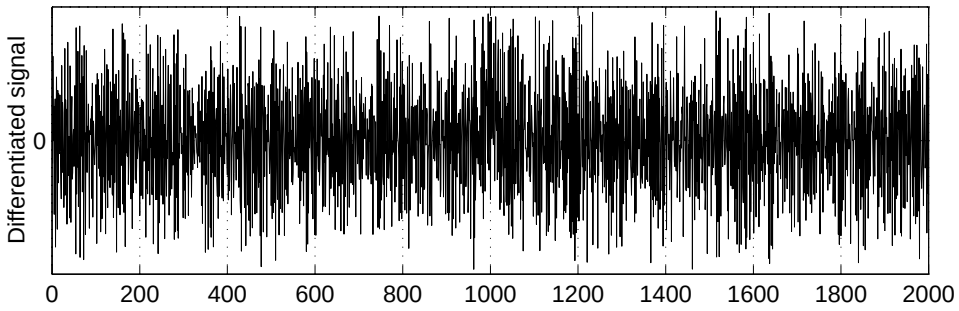
Note that an edge or corner representation imparts a desirable invariance to lighting: the intensity discontinuities are likely to be prominent, whatever the lighting conditions.

1D edge detection

We start with the simple case of edge detection in one dimension. When developing an edge detection algorithm, it is important to bear in mind the invariable presence of image noise. Consider this signal $I(x)$ with an obvious edge.



An intuitive approach to edge detection might be to look for maxima and minima in $I'(x)$.



This simple strategy is defeated by noise. For this reason, all edge detectors start by smoothing the signal to suppress noise. The most common approach is to use a Gaussian filter.

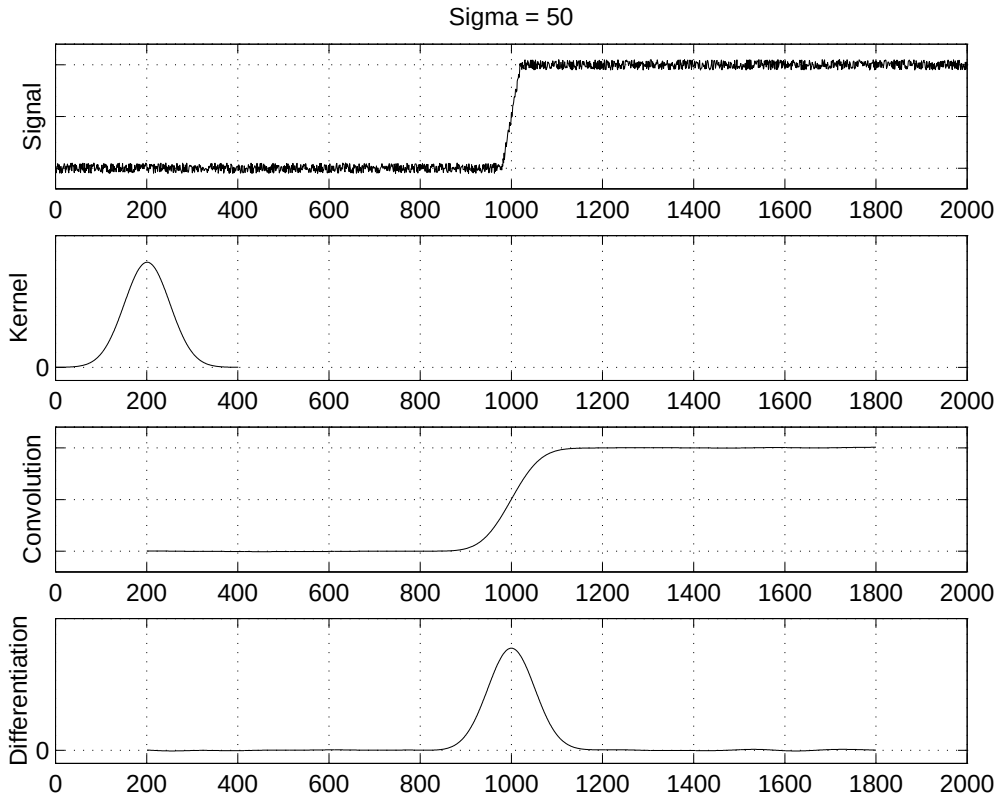
1D edge detection

A broad overview of 1D edge detection is:

1. Convolve the signal $I(x)$ with a Gaussian kernel $g_\sigma(x)$. Call the smoothed signal $s(x)$.

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

2. Compute $s'(x)$, the derivative of $s(x)$.
3. Find maxima and minima of $s'(x)$.
4. Use thresholding on the magnitude of the extrema to mark edges.



1D edge detection

The smoothing in step (1) is performed by a 1D convolution:

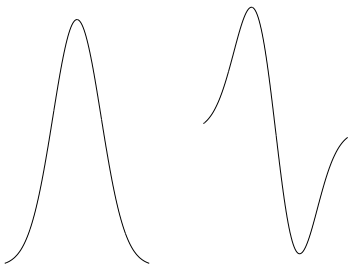
$$s(x) = g_\sigma(x) * I(x) = \int_{-\infty}^{+\infty} g_\sigma(u) I(x-u)\, du = \int_{-\infty}^{+\infty} g_\sigma(x-u) I(u)\, du$$

For discrete signals, the differentiation in step (2) is also performed by a 1D convolution. Thus edge detection would appear to require two computationally expensive convolutions.

However, the derivative theorem of convolution tells us that

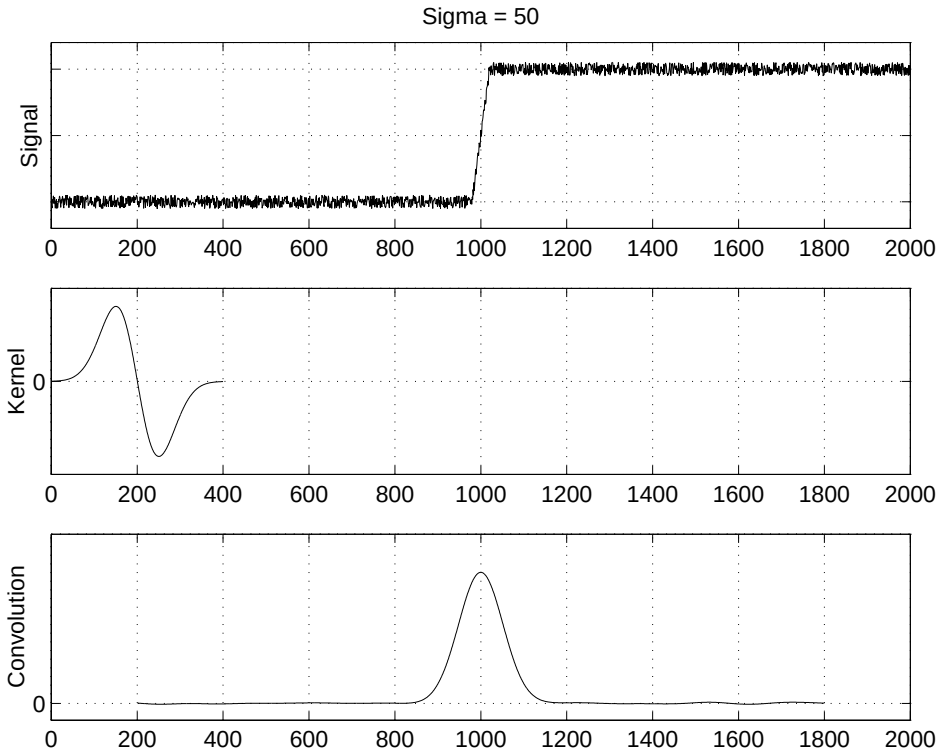
$$s'(x) = \frac{d}{dx}\left[g_\sigma(x) * I(x)\right] = g'_\sigma(x) * I(x)$$

so we can compute $s'(x)$ by convolving only once — a considerable computational saving.



$g_\sigma(x)$ $\quad$ $g'_\sigma(x)$

1D edge detection

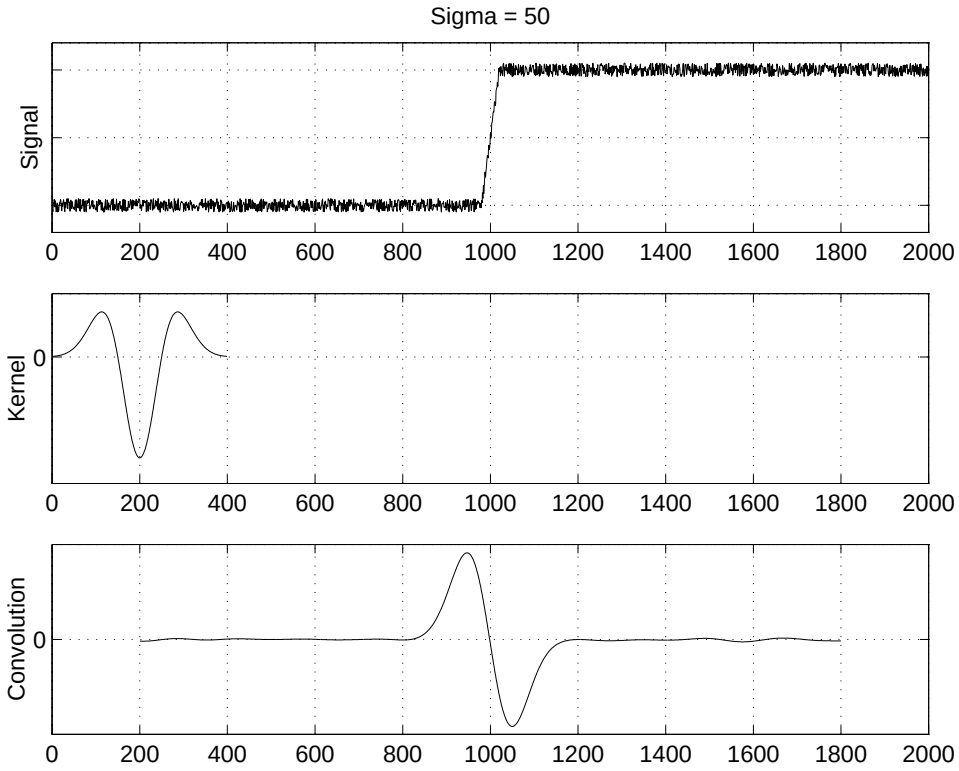


Having obtained the convolved signal $s'(x)$, interpolation can be used to locate any maxima or minima to sub-pixel accuracy. Finally, an edge is marked at each maximum or minimum whose magnitude exceeds some threshold.

Looking for maxima and minima of $s'(x)$ is the same as looking for zero-crossings of $s''(x)$. In many implementations of edge detection algorithms, the signal is convolved with the *Laplacian* of a Gaussian, $g''_\sigma(x)$:

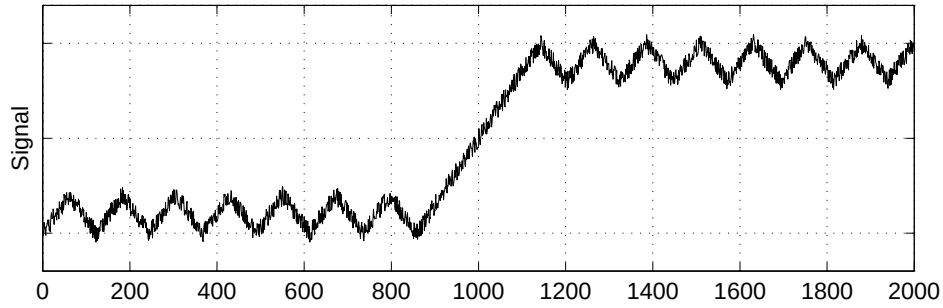
$$s''(x) = g''_\sigma(x) * I(x)$$

Zero-crossings



The zero-crossings of $s''(x)$ mark possible edges.

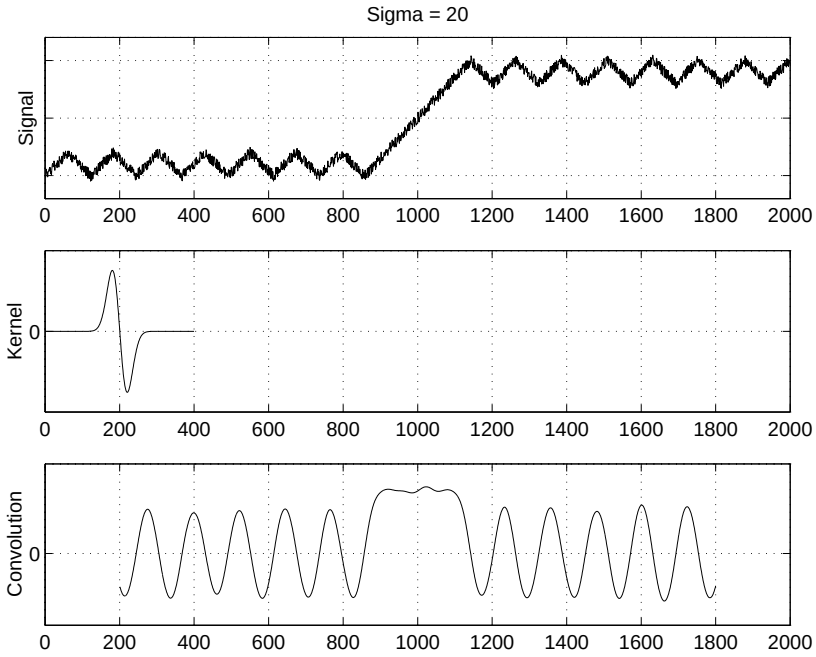
We have not yet addressed the issue of what value of σ to use. Consider this signal:



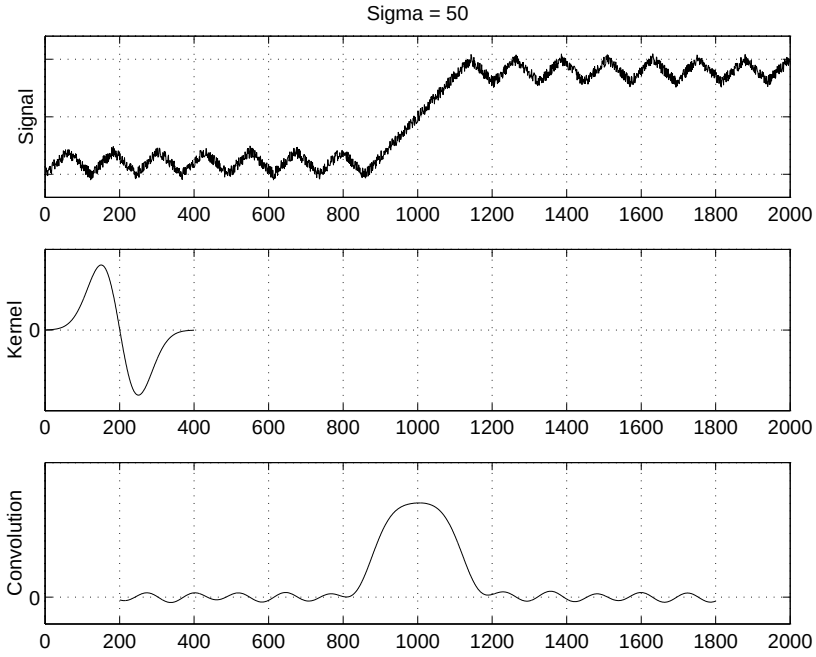
Does the signal have one positive edge or a number of positive and negative edges?

Multi-scale edge detection

Using a small σ brings out all the edges.



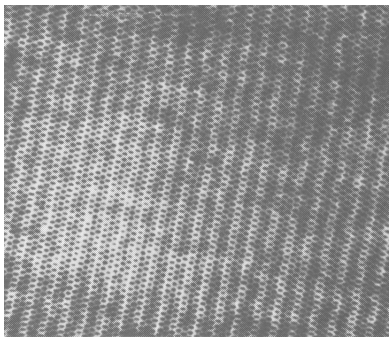
As σ increases, the signal is smoothed more and more, and only the central edge survives.



Multi-scale edge detection

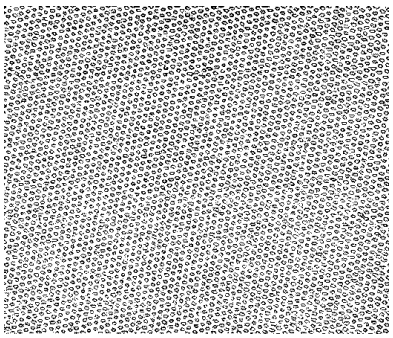
The amount of smoothing controls the **scale** at which we analyse the image. There is no right or wrong size for the Gaussian kernel: it all depends on the scale we're interested in.

Modest smoothing (a Gaussian kernel with small σ) brings out edges at a fine scale. More smoothing (larger σ) identifies edges at larger scales, suppressing the finer detail.

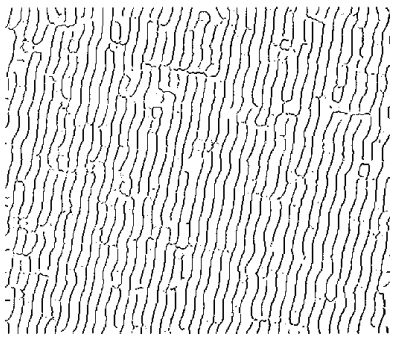


This is an image of a dish cloth. After edge detection, we see different features at different scales.

$\sigma = 1$



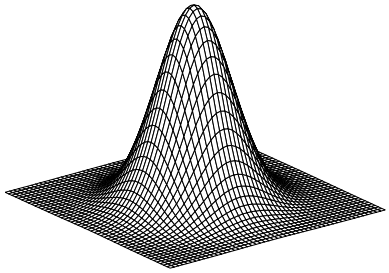
$\sigma = 5$



Fine scale edge detection is particularly sensitive to noise. This is less of an issue when analysing images at coarse scales.

2D edge detection

The 1D edge detection scheme can be extended to work in two dimensions. First we smooth the image $I(x, y)$ by convolving with a 2D Gaussian $G_\sigma(x, y)$:



$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp - \left(\frac{x^2 + y^2}{2\sigma^2} \right)$$

$$\begin{aligned} S(x, y) &= G_\sigma(x, y) * I(x, y) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_\sigma(u, v) I(x - u, y - v)\, du\, dv \end{aligned}$$

The effects of this blurring on a typical image:

Unsmoothed

$\sigma = 3$ pixels

$\sigma = 4$ pixels

2D edge detection

The next step is to find the gradient of the smoothed image $S(x, y)$ at every pixel:

$$\nabla S = \nabla(G_\sigma * I)$$

$$= \begin{bmatrix} \frac{\partial (G_\sigma * I)}{\partial x} \\ \\ \frac{\partial (G_\sigma * I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix}$$

The following example shows $|\nabla S|$ for a fruity image:

(a) Original image

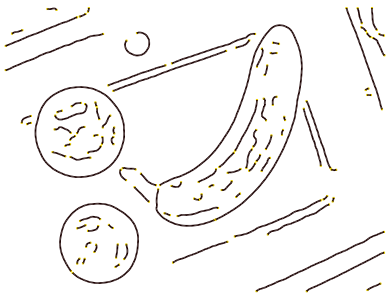
(b) Edge strength $|\nabla S|$

2D edge detection

The next stage of the edge detection algorithm is **non-maximal suppression**. Edge elements, or **edgels**, are placed at locations where $|\nabla S|$ is greater than local values of $|\nabla S|$ in the directions $\pm\nabla S$. This aims to ensure that all edgels are located at ridge-points of the surface $|\nabla S|$.

(c) Non-maximal suppression

Next, the edgels are **thresholded**, so that only those with $|\nabla S|$ above a certain value are retained.

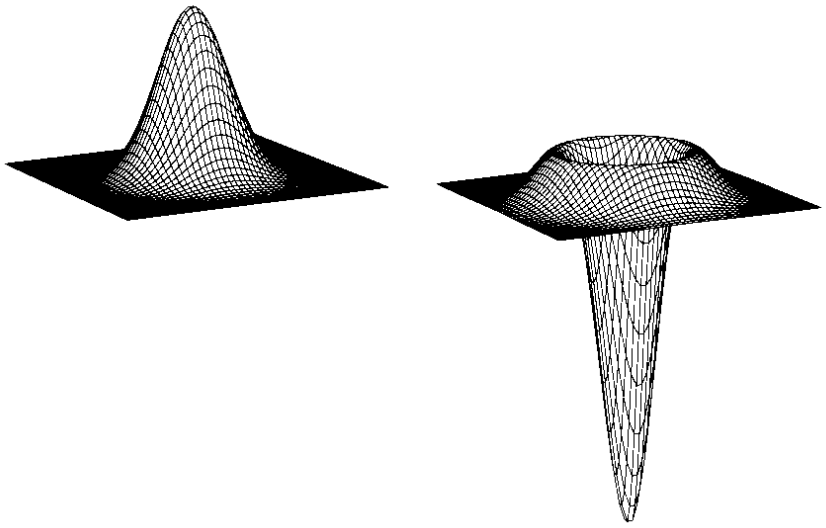


(d) Thresholding

2D edge detection

The edge detection algorithm we have been describing is due to Canny (1986). The output is a list of edgel positions, each with a strength $|\nabla S|$ and an orientation $\nabla S / |\nabla S|$.

An alternative approach to edge detection was developed by Marr and Hildreth (1980). While the Canny detector is a *directional* edge finder (both the gradient magnitude and direction are computed), the Marr-Hildreth operator is *isotropic*. It finds zero-crossings of $\nabla^2 G_\sigma * I$, where $\nabla^2 G_\sigma$ is the **Laplacian** of G_σ ($\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$).

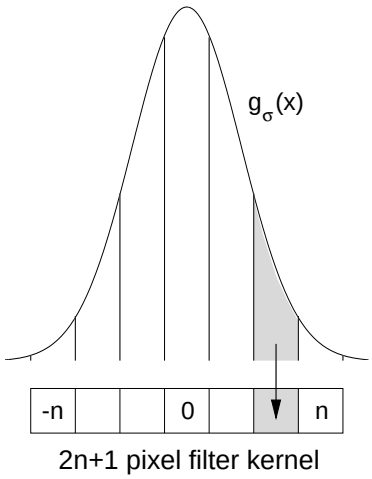


G_σ $\nabla^2 G_\sigma$

Implementation details

In practice, the image and filter kernels are discrete quantities and the convolutions are performed as truncated summations:

$$S(x,y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} G_\sigma(u,v) I(x-u, y-v)$$



For acceptable accuracy, kernels are generally truncated so that the discarded samples are less than 1/1000 of the peak value.

σ	1.0	1.5	3	6
$2n+1$	7	11	23	45

The 2D convolutions would appear to be computationally expensive. However, they can be decomposed into two 1D convolutions:

$$G_\sigma(x,y) * I(x,y) = g_\sigma(x) * [g_\sigma(y) * I(x,y)]$$

The computational saving is $(2n+1)^2/2(2n+1)$.

Implementation details

Differentiation of the smoothed image is also implemented with a discrete convolution.

By considering the Taylor-series expansion of $S(x, y)$ it is easy to show that a simple finite-difference approximation to the first-order spatial derivative of $S(x, y)$ is given by:

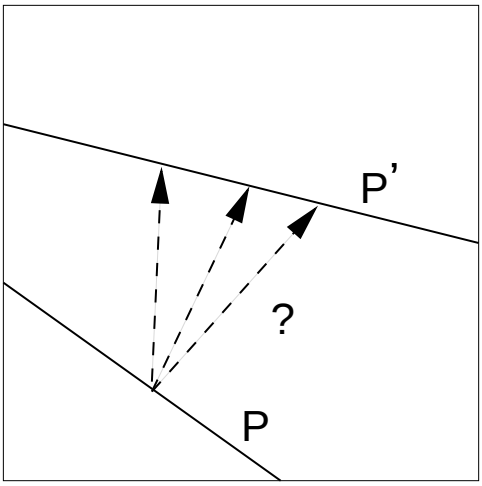
$$\frac{\partial S}{\partial x} = \frac{S(x+1, y) - S(x-1, y)}{2}$$

This is equivalent to convolving the rows of image samples, $S(x, y)$, with the kernel

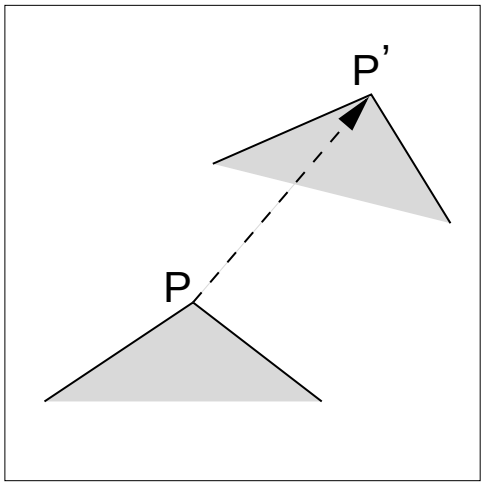
1/2	0	-1/2

Corners

While edges are a powerful intermediate representation, they are sometimes insufficient. This is especially the case when image *motion* is being analysed. The motion of an edge is rendered ambiguous by the **aperture problem**: when viewing a moving edge, it is only possible to measure the motion *normal* to the edge.



Edge



Corner

To measure image motion completely, we really need to look at **corner** features. We saw earlier that a corner is characterized by an intensity discontinuity in two directions. This discontinuity can be detected using **correlation**.

Correlation

The normalized **cross-correlation** function measures how well an image patch, $P(x, y)$, matches other portions of the image, $I(x, y)$, as it is shifted from its original location. It entails sliding the patch over the image, computing the sum of the products of the pixels and normalizing the result:

$$c(x,y) = \frac{\sum_{u=-n}^{n}\sum_{v=-n}^{n} P(u,v)\, I(x+u, y+v)}{\sqrt{\sum_{u=-n}^{n}\sum_{v-n}^{n} P^2(u,v) \sum_{u=-n}^{n}\sum_{v-n}^{n} I^2(x+u, y+v)}}$$

A patch which has a well-defined peak in its correlation function can be classified as a "corner".

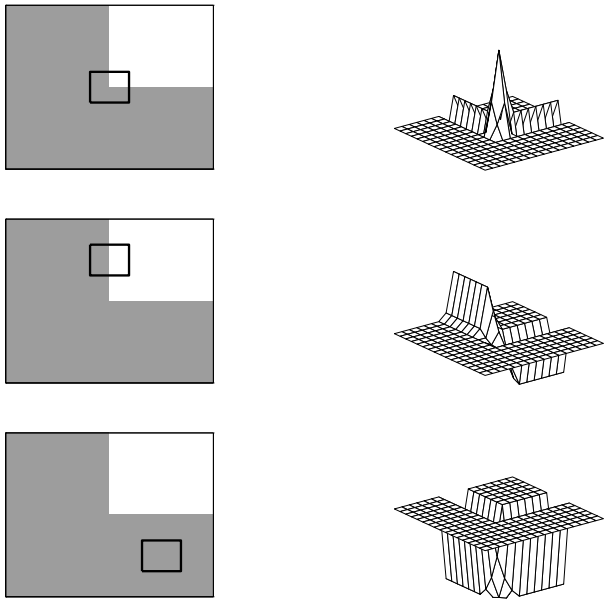


Image I & patch$P$ Correlation $c(x, y)$

Corner detection

A practical corner detection algorithm needs to do something more efficient than calculate correlation functions for every pixel!

1. Calculate change in intensity in direction $\mathbf{n}$:

$$I_n \equiv \nabla I(x,y).\hat{\mathbf{n}} \equiv \begin{bmatrix} I_x & I_y \end{bmatrix}^T .\hat{\mathbf{n}}$$

$$I_n^2 = \frac{\mathbf{n}^T \nabla I \nabla I^T \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

$$= \frac{\mathbf{n}^T \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

where $I_x \equiv \partial I/\partial x$ and $I_y \equiv \partial I/\partial y$.

2. Smooth I_n^2 by convolution with a Gaussian kernel:

$$C_n(x,y) = G_\sigma(x,y) * I_n^2$$

$$= \frac{\mathbf{n}^T \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

where $\langle\ \rangle$ is the smoothed value.

Corner detection

The smoothed change in intensity around (x,y) in direction $\mathbf{n}$ is therefore given by

$$C_n(x,y) = \frac{\mathbf{n}^T \mathrm{A} \mathbf{n}}{\mathbf{n}^T \mathbf{n}}$$

where A is the 2×2 matrix

$$\begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

Elementary eigenvector theory tells us that

$$\lambda_1 \leq C_n(x,y) \leq \lambda_2$$

where λ_1 and λ_2 are the eigenvalues of A. So, if we try every possible orientation $\mathbf{n}$, the maximum smoothed change in intensity we will find is λ_2, and the minimum value is λ_1.

We can therefore classify image structure around each pixel by looking at the eigenvalues of A:

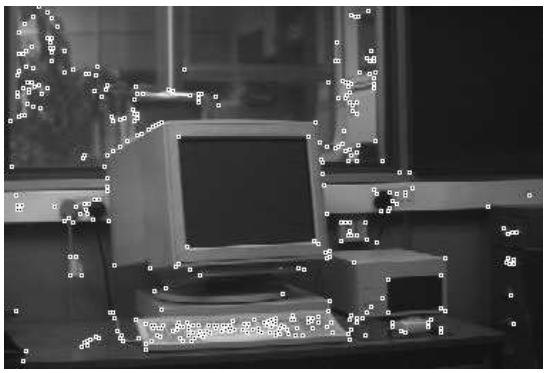
No structure: (smooth variation) $\lambda_1 \approx \lambda_2 \approx 0$

1D structure: (edge) $\lambda_1 \approx 0$ (direction of edge), λ_2 large (normal to edge)

2D structure: (corner) λ_1 and λ_2 both large and distinct

Corner detection

The corner detection algorithm we have been describing is due to Harris (1987). It is necessary to calculate A at every pixel and mark corners where the quantity $\lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2$ exceeds some threshold ($\kappa \approx 0.04$ makes the detector a little "edge-phobic"). Note that $\det \mathrm{A} = \lambda_1\lambda_2$ and trace A$= \lambda_1 + \lambda_2$.



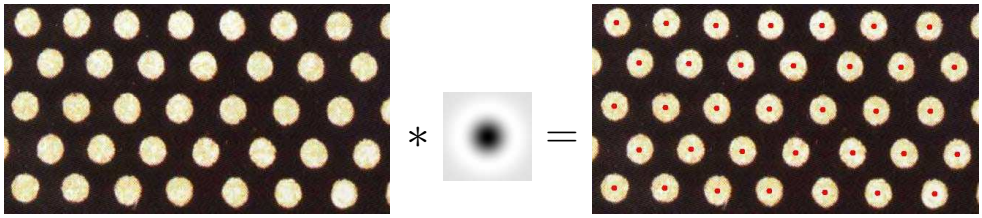
Low threshold

High threshold

Corners are most useful for **tracking** in image sequences or **matching** in stereo pairs. Unlike edges, the displacement of a corner is not ambiguous. Corner detectors must be judged on their ability to detect the *same* corners in similar images. Current detectors are not too reliable, and higher-level visual routines must be designed to tolerate a significant number of **outliers** in the output of the corner detector.

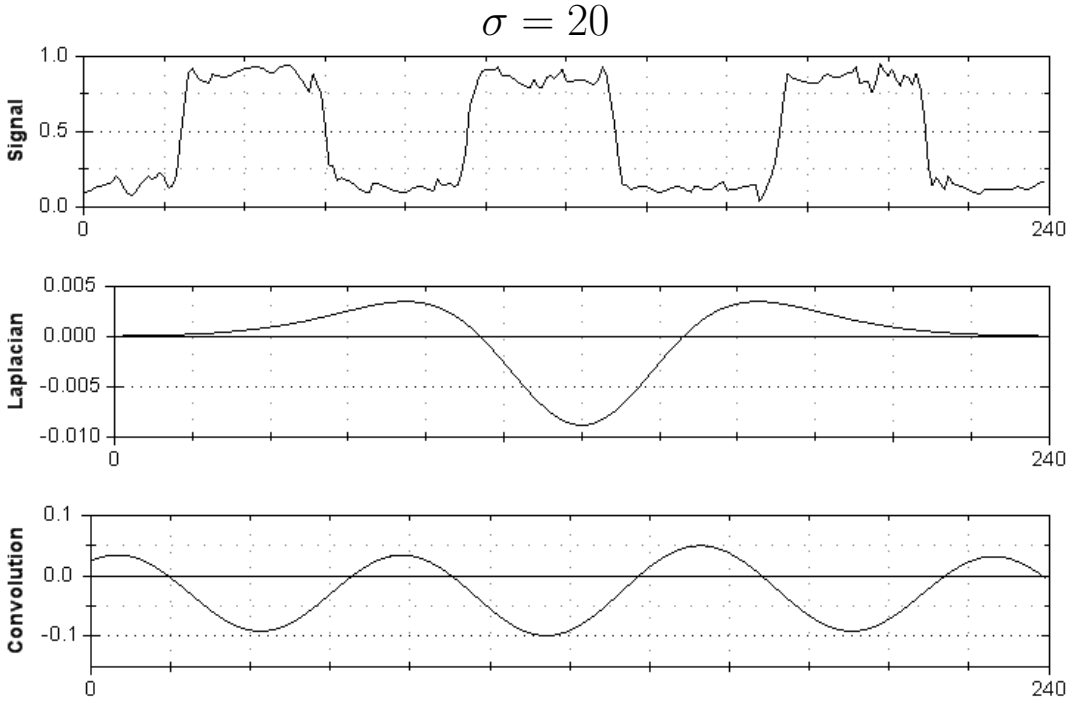
Interest Point Detection - Blobs

A *blob* is a region of pixels with intensities higher (or lower) than surrounding pixels. Whereas edges and corners are features which are found at discontinuities, blobs are localised in the middle of areas of similar intensity which are surrounded by pixels of a different intensity on their boundaries.



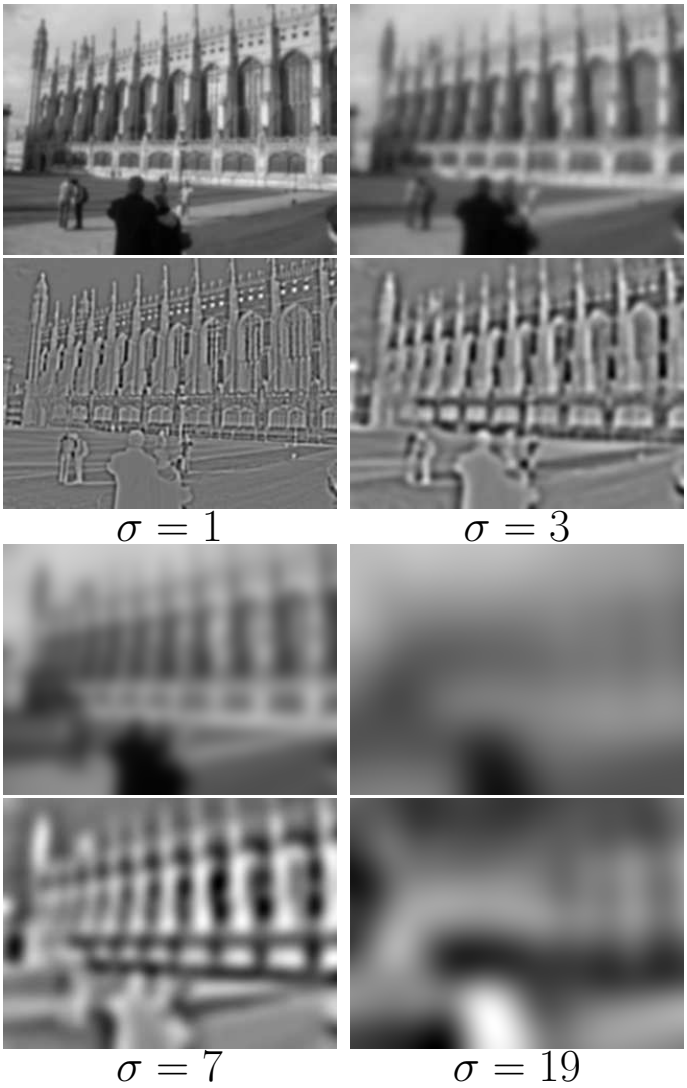
Polka Dots Detected Blobs

The 1D signal is a scan line running across one of the polka dots above. The result shows how, even though the signal is quite noisy, the minimum of the resulting response from the **Laplacian** of the Gaussian places the centre of the dot perfectly.



Blobs and Band-pass filtering

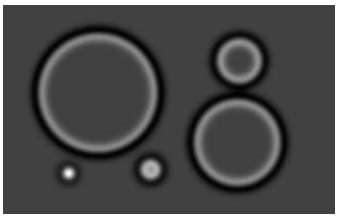
The size of the blob detected depends on the sigma of the detector used. As the sigma is increased, larger and larger image features are detected, ranging from small boxes to entire buildings. Each time the blob detector will fire on the center of the blob in question, making it ideal for extracting texture from the inside of an object or for fixing location of an object in the scene.



Scale Space

Corners and especially blobs have a range of scales over which they will be detected. The Laplacian of a Gaussian as recorded at a particular location is a monotonic function over scale, with definite peaks or troughs. These maxima and minima lie in the middle of the ranges at which blobs and corners will be detected at that point, and thus are considered ideal places to examine the surroundings of the feature point for use in later processes.

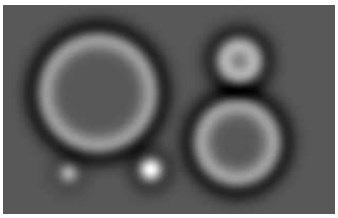
Original



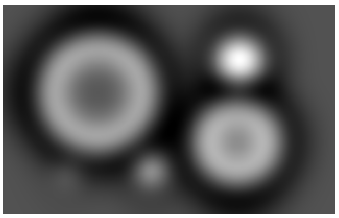
Level 0

$\sigma = 5$

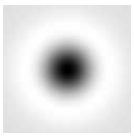
$\sigma = 10$



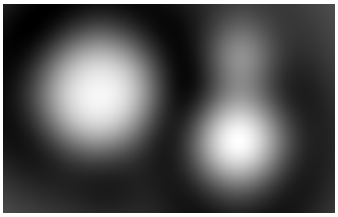
Level 1



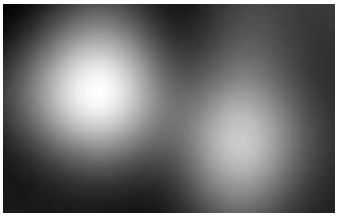
Level 2



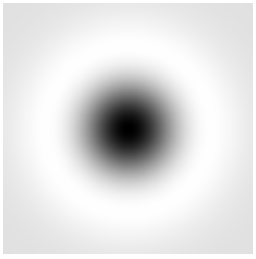
$\sigma = 20$



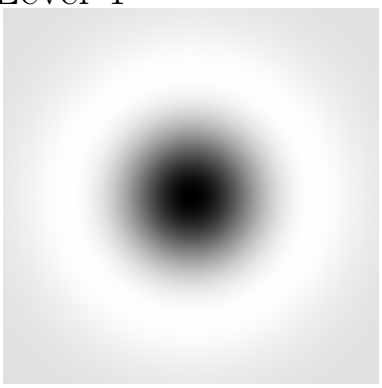
Level 3



Level 4



$\sigma = 40$



$\sigma = 80$

Scale Space, cont.

We achieve scale independence by looking at the different resolutions of an image. There are an infinite number of possible resolutions for any image, a three-dimensional function of intensity over location and scale known as the **scale space** of the image, denoted $S(x, y, t)$. This can be calculated by convolving the original image $I(x, y)$ with Gaussians of arbitrary scale t, thus the scale space function is often written as

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

It is impractical to examine all possible resolutions, and indeed impossible to do so when we are restricted by digital image representation. Thus, we sample the space by choosing particular resolutions to examine.

Scale Space, cont.

We produce a discrete set of low-pass filtered images with scale satisfying

$$\sigma_i = 2^{\frac{i}{s}}\sigma_0$$

so that after that it doubles after s intervals. The s images in each octave are spaced logarithmically with the scale of neighbouring images satisfying

$$\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$$

Blurring with large scales is avoided in 2 ways: subsampling the image after each octave (i.e. scale has doubled) and by using a finite set of incremental blurrs.

The resulting sampling of spaces is called an **image pyramid**, for which we compute scales ranging from σ to 2σ (an octave), and then subsample the image for the next octave.

Scale Space, cont.

Within each octave, as we convolve images repeatedly with Gaussian filters, the resulting image has a σ calculated as

$$G(\sigma_1) * G(\sigma_2) = G\left(\sqrt{\sigma_1^2 + \sigma_2^2}\right)$$

At every interval i of the pyramid, we want $\sigma_i = 2^{\frac{i}{s}}\sigma_0$ (so that it doubles after s intervals). We need to achieve this incrementally, thus

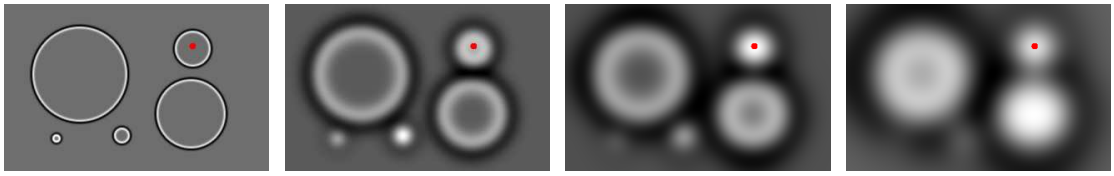
$$G(\sigma_{i+1}) = G(\sigma_i) * G(\sigma_k)$$

So, what is σ_k? 

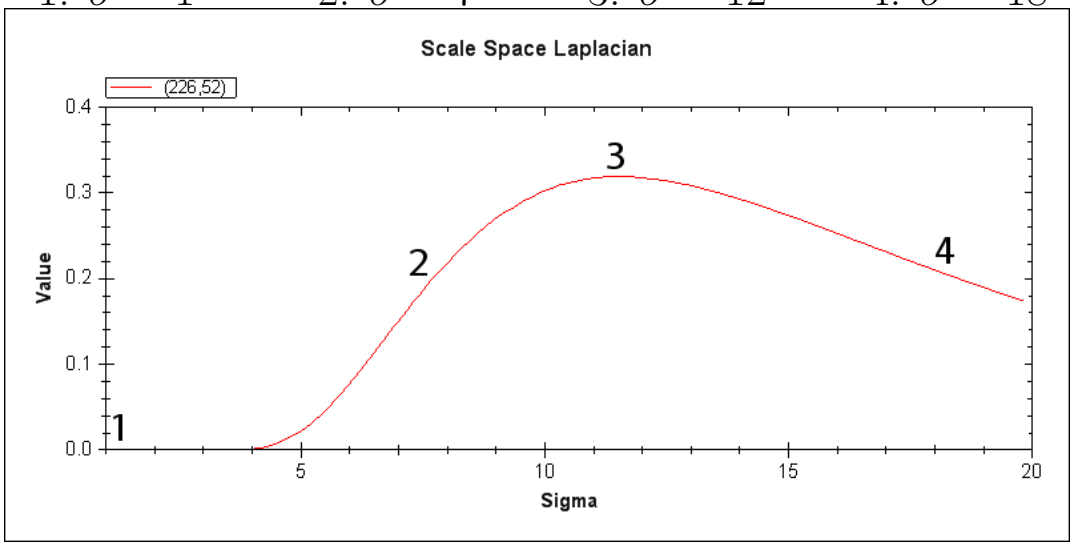
$$\begin{aligned}
\sigma_k &= \sqrt{\sigma_{i+1}^2 - \sigma_i^2} \\
\sigma_{i+1} &= 2^{\frac{1}{s}}\sigma_i \\
\sigma_k &= \sqrt{2^{\frac{2}{s}}\sigma_i^2 - \sigma_i^2} \\
\sigma_k &= \sigma_i\sqrt{2^{\frac{2}{s}} - 1}
\end{aligned}$$

Scale Space, cont.

Different scales are ideal for interest points of different sizes, and all the interest points which are ideal at particular scale are used to describe the image at that scale, thus providing scale invariance to the overall system. The ideal scale for a keypoint is located at the maximum of the scale space function at that point. For example, with a blob, we would want to find the maximum of the Laplacian of a Gaussian over scale.



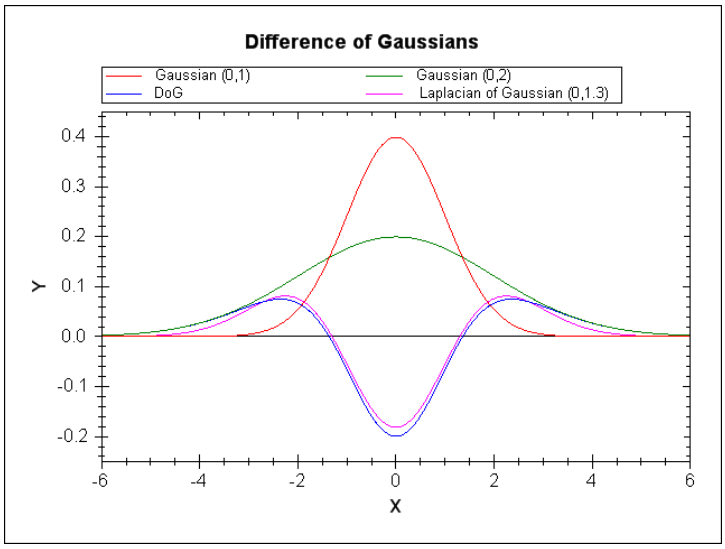
1: $\sigma = 1$ 2: $\sigma = 7$ 3: $\sigma = 12$ 4: $\sigma = 18$



Since we cannot efficiently find the exact point, the largest value of the samples in the pyramid is found by comparing 26 neighbours and then the exact point interpolated.

Difference of Gaussians

The **difference of Gaussians** interest point, or DoG as it is often called, is a blob detector. The points are taken from the minima and maxima of the DoG response over an image. It takes its name from the fact that it is calculated as the difference of two Gaussians, which approximates the Laplacian of a Gaussian.

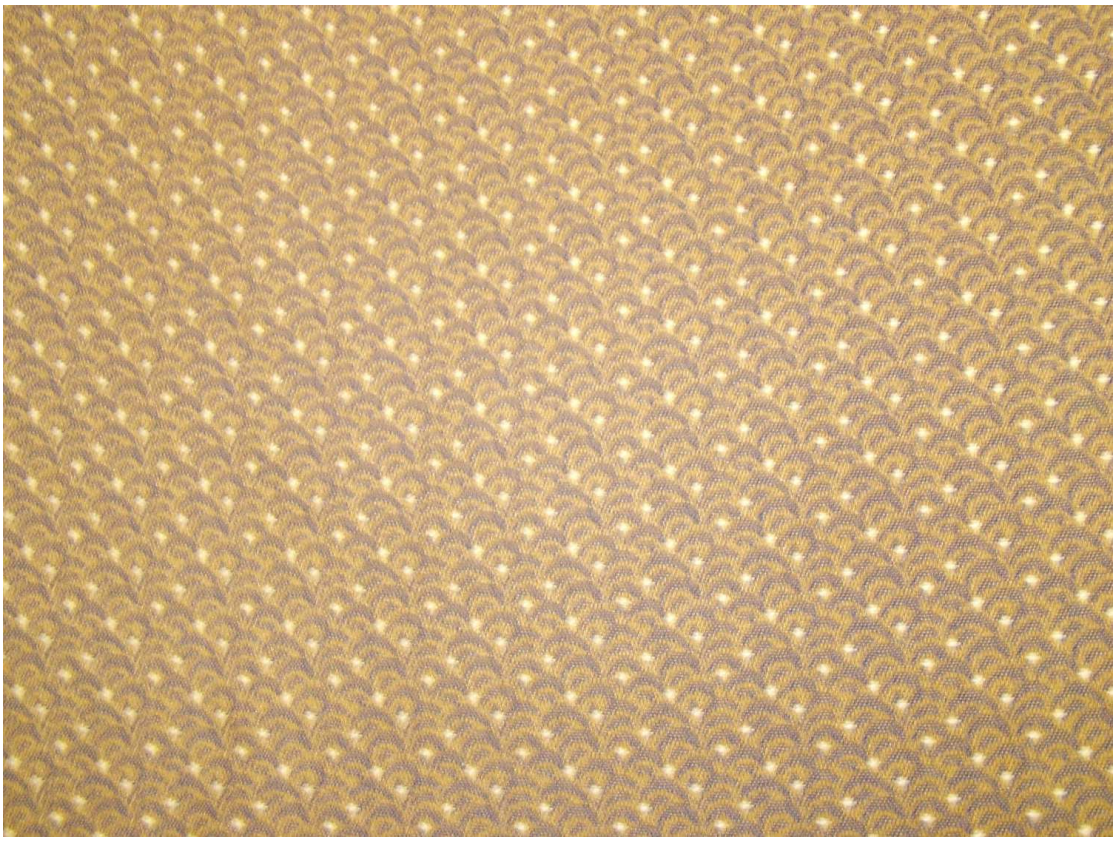


In a system which utilizes a scale space pyramid (such as the one we will consider), the DoG point is a useful entity, as a response can be computed simply subtracting one member of a pyramid level from the one directly above it. For image matching, blobs are particularly useful features to concentrate on, as they are usually found inside of objects as opposed to at their edges and thus are less likely to contain part of the background in queries, in addition to other properties such as stability, repeatability, and a definite optimal scale.

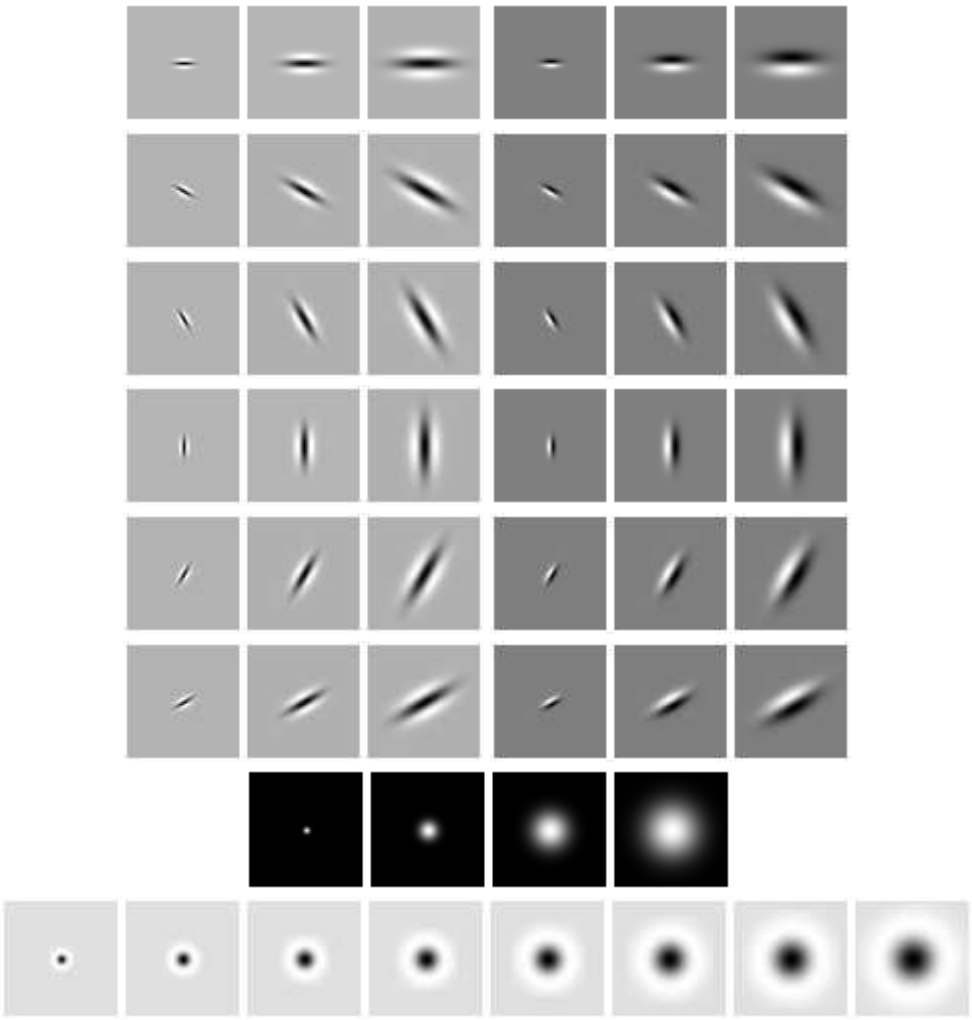
Image Structure: Texture

Image texture arises from large numbers of small objects such as grass, brush, pebbles and hair and surfaces with orderly and repetitive patterns such as the spots or stripes on animals, wood and skin. They typically consist of organised patterns of regular sub-elements called *textons*. A natural way to describe texture is to find these textons and describe how they are distributed.

Image textures can be described by their response to a collection of filters to represent the patterns of spots, bars etc.



Characterising Texture



This is an example filter bank. It consists of 8 Laplacian of Gaussian filters and 4 Gaussian filters at different scales to provided non-oriented responses, and 36 oriented filters at 6 different angles, 3 different scales, and 2 different *phases*. The two phases of oriented filters are first and second derivatives of Gaussians on the minor axis and elongated Gaussians on the major axis, and thus detect edges or bars respectively along their major axes.

The descriptor is simply the concatenated responses of all of the filters in the filter bank at a pixel.