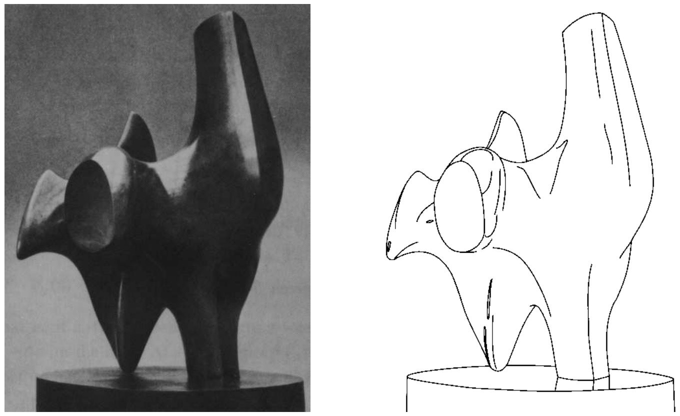
University of Cambridge
Engineering Part IIB & EIST Part II

Module I12: Computer Vision and Robotics

Handout 2: Image Structure



Roberto Cipolla and Andrew Gee
October 1999

Image interpretation

We can represent a monochrome image as a matrix $I(x,y)$ of intensity values. The size of the matrix is typically 512×512 and the intensity values are usually sampled to an accuracy of 8 bits (256 **grey levels**).

$I(x,y)$ is a function of many variables, including:

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

Typically, we aim to deduce (1) and (3) from the image, occasionally we are also interested in (5). We generally want to deduce this information independent of (2) and (4).

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
12 MBytes/s		5 KBytes/s
(mono CCD)		

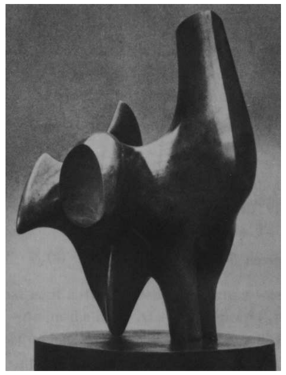
All subsequent interpretation is performed on the generic representation, not the original image. We aim to:

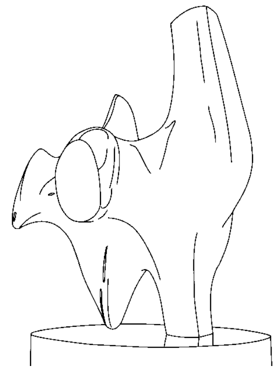
- Dramatically reduce the amount of data.
- Preserve the useful information in the images (such as the position of the camera and the shape of objects in the scene).
- Discard the redundant information in the images (such as the lighting conditions).

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

Salient features

The human visual system and the world of art give us strong clues about what sort of processing is required. It seems possible to interpret images using a small amount of **edge** and **corner** data.


The Archer, Henry Moore. 8-bit greyscale image, 591 KBytes.

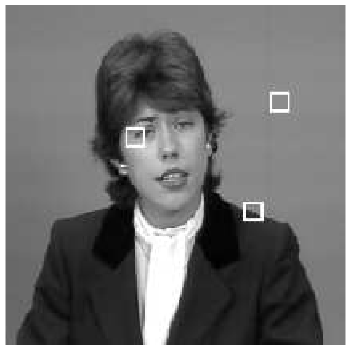

Artist's line drawing of *The Archer*. Perhaps 200 bytes of information.

Edges and corners can be matched in stereo views or tracked over time to deduce the scene structure. By making assumptions about the world, it is also possible to successfully interpret single images containing only edges.

But how can we automatically and efficiently extract these features in images?

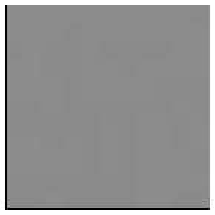
Image structure

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.

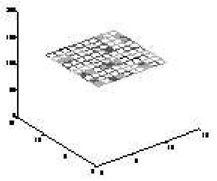


0D

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

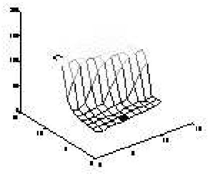


Edges and corners

1D

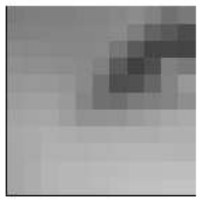
The patch containing the edge reveals an intensity discontinuity in one direction.

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

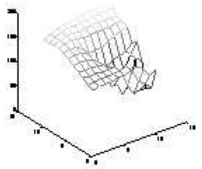


2D

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



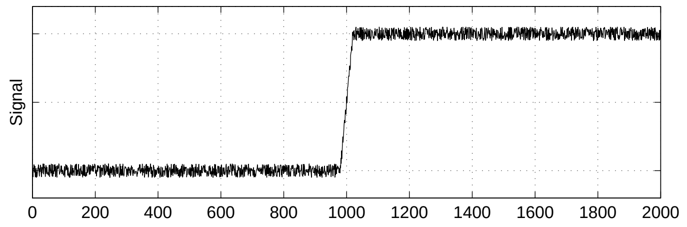
124	127	128	128	131	312	135	140	130	113	121
129	130	132	132	133	135	139	125	99	89	76
138	136	137	135	135	136	120	89	68	71	69
144	142	143	141	139	129	92	64	78	128	121
150	152	151	148	138	113	79	81	102	131	152
158	160	160	154	133	113	111	123	127	131	143
163	165	166	158	145	146	147	155	160	166	171
168	171	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



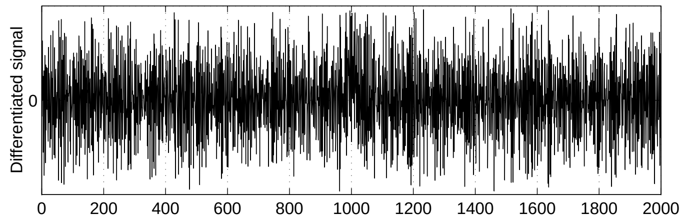
Note that an edge or corner representation imparts a desirable invariance to lighting: the intensity discontinuities are likely to be prominent, whatever the lighting conditions.

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 $I'(x)$.

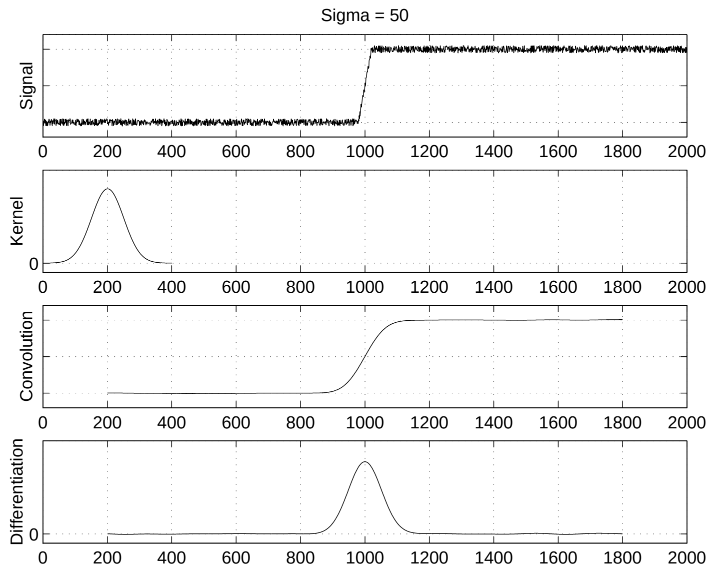


This simple strategy is defeated by noise. For this reason, all edge detectors start by smoothing the signal to suppress noise. The most common approach is to use a Gaussian filter.

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:

✍

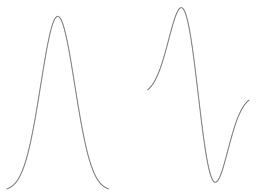
$$s(x) = g_\sigma(x) * I(x) = \int_{-\infty}^{+\infty} g_\sigma(u) I(x-u)\, du = \int_{-\infty}^{+\infty} g_\sigma(x-u) I(u)\, du$$

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

However, the derivative theorem of convolution tells us that

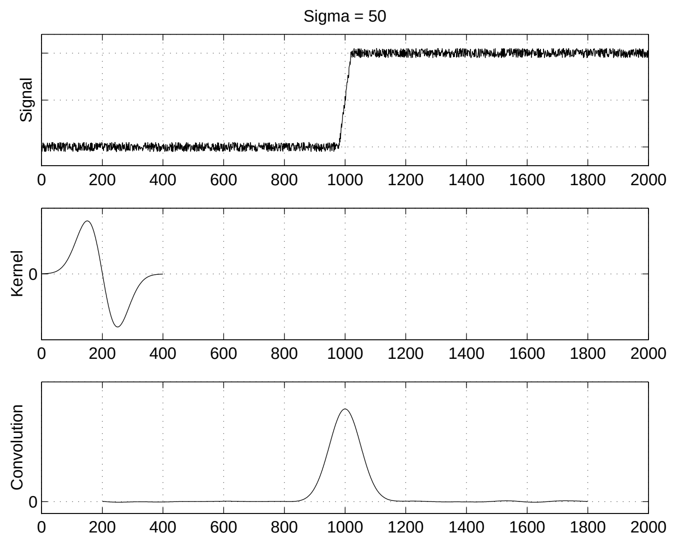
$$s'(x) = \frac{d}{dx}\left[g_\sigma(x) * I(x)\right] = g'_\sigma(x) * I(x)$$

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



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

1D edge detection

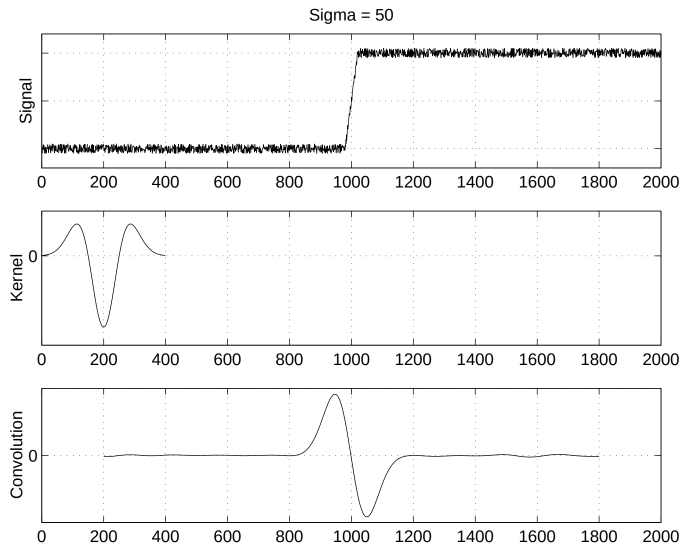


Having obtained the convolved signal $s'(x)$, interpolation can be used to locate any maxima or minima to sub-pixel accuracy. Finally, an edge is marked at each maximum or minimum whose magnitude exceeds some threshold.

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, $g''_\sigma(x)$:

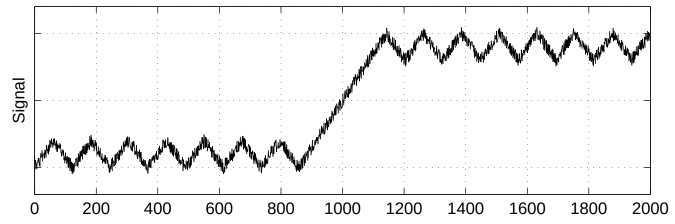
$$s''(x) = g''_\sigma(x) * I(x)$$

Zero-crossings



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

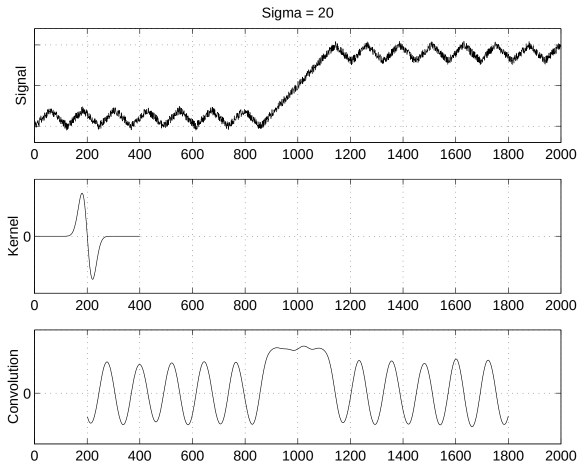
We have not yet addressed the issue of what value of σ to use. Consider this signal:



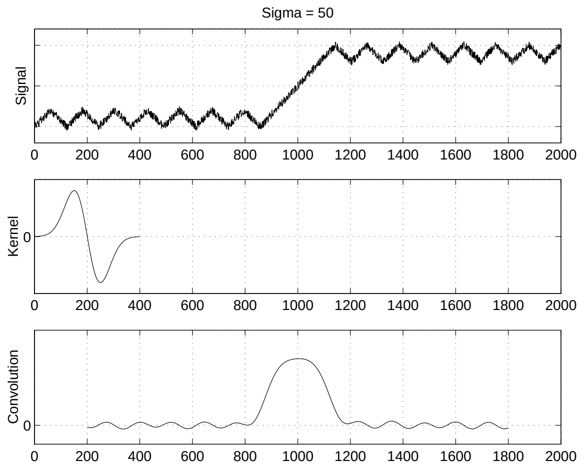
Does the signal have one positive edge or a number of positive and negative edges?

Multi-scale edge detection

Using a small σ brings out all the edges.



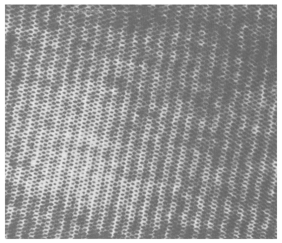
As σ increases, the signal is smoothed more and more, and only the central edge survives.



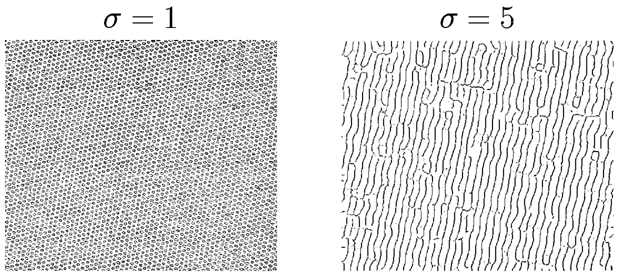
Multi-scale edge detection

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.



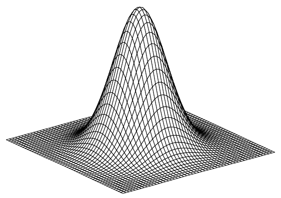
This is an image of a dish cloth. After edge detection, we see different features at different scales.



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

2D edge detection

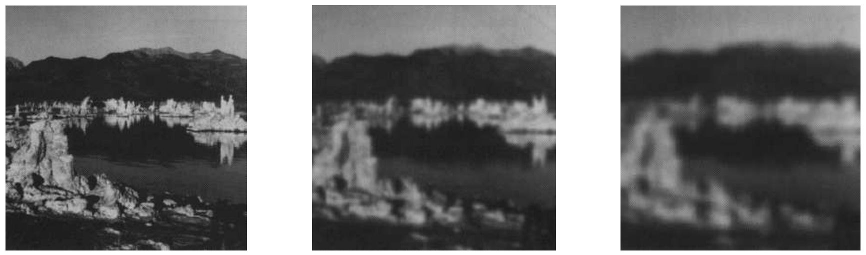
The 1D edge detection scheme can be extended to work in two dimensions. First we smooth the image $I(x, y)$ by convolving with a 2D Gaussian $G_\sigma(x, y)$:



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

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

The effects of this blurring on a typical image:



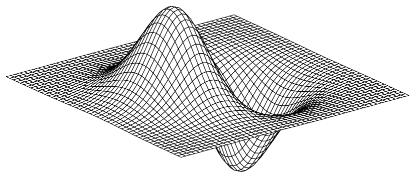
Unsmoothed $\sigma = 3$ pixels $\sigma = 4$ pixels

2D edge detection

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

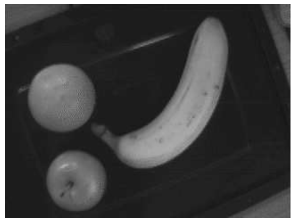
$$\nabla S = \nabla(G_\sigma * I)$$

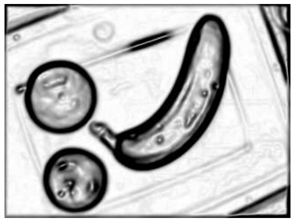
$$= \begin{bmatrix} \frac{\partial(G_\sigma * I)}{\partial x} \\ \frac{\partial(G_\sigma * I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix}$$



$$\frac{\partial G_\sigma(x,y)}{\partial x} = \frac{-x}{2\pi\sigma^4} \exp - \left(\frac{x^2+y^2}{2\sigma^2}\right)$$

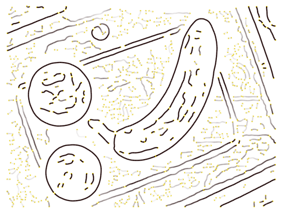
The following example shows $|\nabla S|$ for a fruity image:


(a) Original image

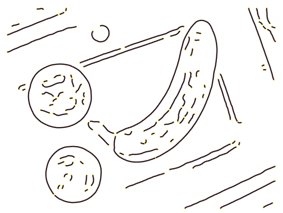

(b) Edge strength $|\nabla S|$

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

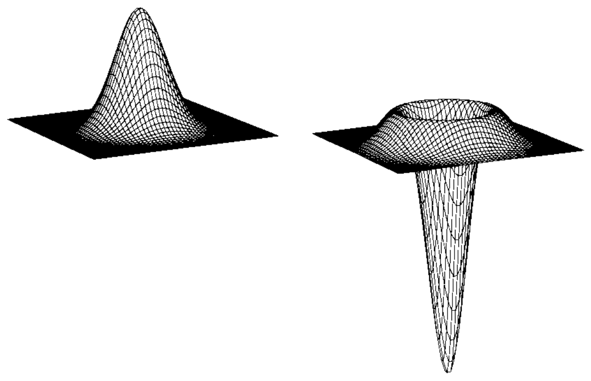
Next, the edgels are **thresholded**, so that only those with $|\nabla S|$ above a certain value are retained. Finally, weak edgels, which have been deleted, are revived if they span gaps between strong edgels, in a process known as **hysteresis**.


(d) Thresholding with hysteresis

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 ($\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$).

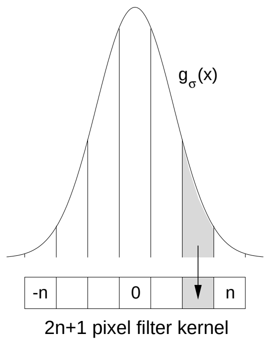


G_σ $\nabla^2 G_\sigma$

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	1.5	3	6
$2n+1$	11	23	45

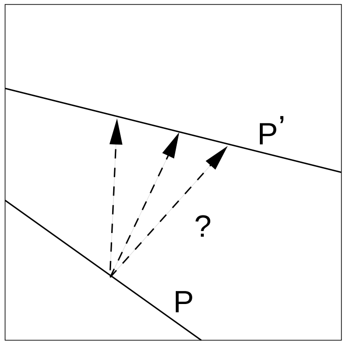
The 2D convolutions would appear to be computationally expensive. However, they can be decomposed into two 1D convolutions:

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

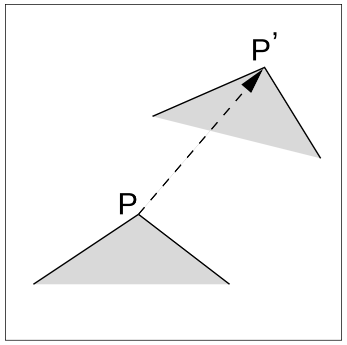
The computational saving is $(2n+1)^2/2(2n+1)$.

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 matches other portions of the image 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:

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

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

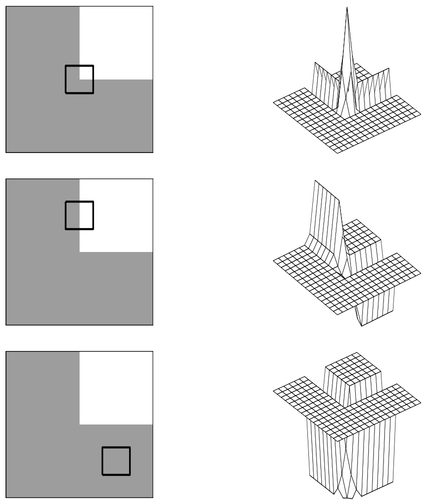


Image & patch Correlation

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:

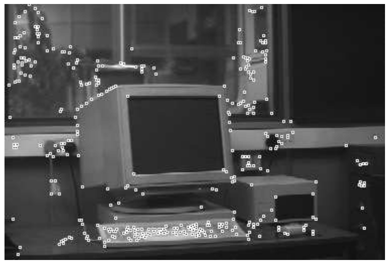
No structure: (smooth variation) $\lambda_1 \approx \lambda_2 \approx 0$

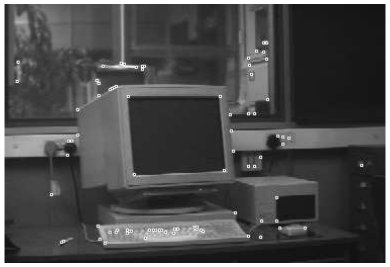
1D structure: (edge) $\lambda_1 \approx 0$ (direction of edge), λ_2 large (normal to edge)

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 $\det A = \lambda_1\lambda_2$ and trace A$= \lambda_1 + \lambda_2$.

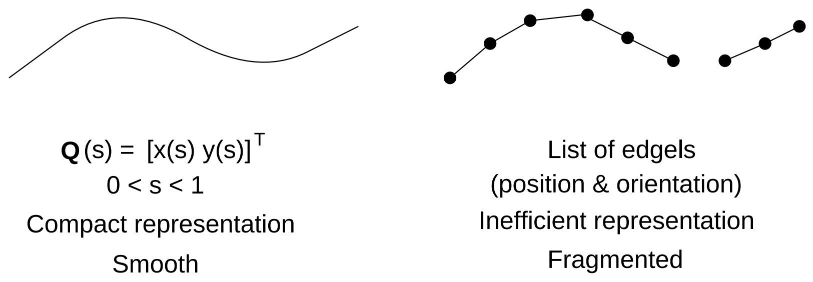

Low threshold


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.

Algebraic edge representations

Compared with raw images, edgel representations offer significant data reduction while preserving most of the useful image structure. However, they are not as compact a representation as they could be (a typical image will generate thousands of edgels), and current edge detection algorithms often produce fragmented edgel chains along strong edges.



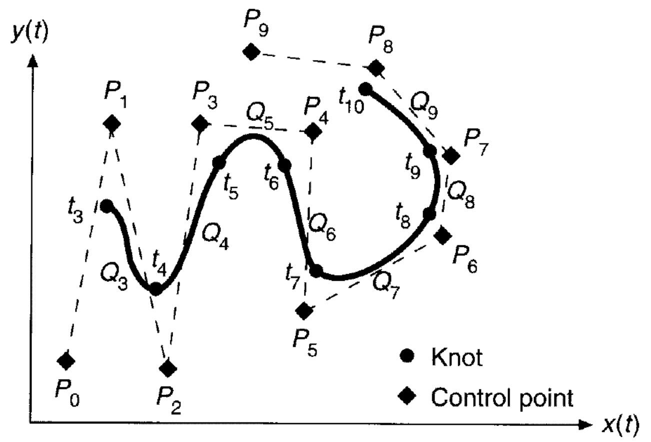
An alternative approach is to attempt to automatically fit an algebraic, parameterized curve to an edge of interest. The representation is now extremely compact (only the coefficients of the curve's equation need be stored) and continuity of the edge is implicit.

B-splines

A natural choice for the curve parameterization is the **B-spline**, which is widely used in computer graphics. A cubic B-spline is specified by $m+1$ **control points** $\mathbf{p}_0, \mathbf{p}_1 \ldots \mathbf{p}_m$ and comprises $m-2$ cubic polynomial curve segments $\mathbf{Q}_3, \mathbf{Q}_4 \ldots \mathbf{Q}_m$. The joining points between each curve segment are known as **knots**. The equation of each curve segment is

$$\mathbf{Q}_i(s) = \frac{1}{6}\left[s^3\ s^2\ s\ 1\right]\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_{i-3} \\ \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \end{bmatrix}$$

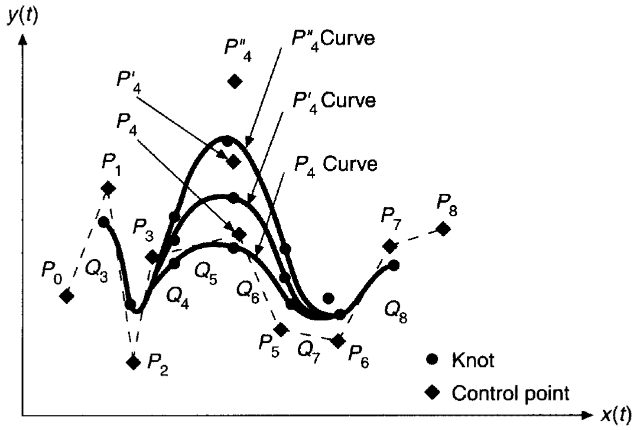
for $0 \le s < 1$ and $3 \le i \le m$.



Properties of B-splines

B-splines are ideal for fitting to image edges. They may be open or closed as required, and are defined with continuity properties at each knot. The flexibility of the curve increases as more control points are added: each additional control point allows one more inflection. It is also possible to use multiple knots to reduce the continuity at knots.

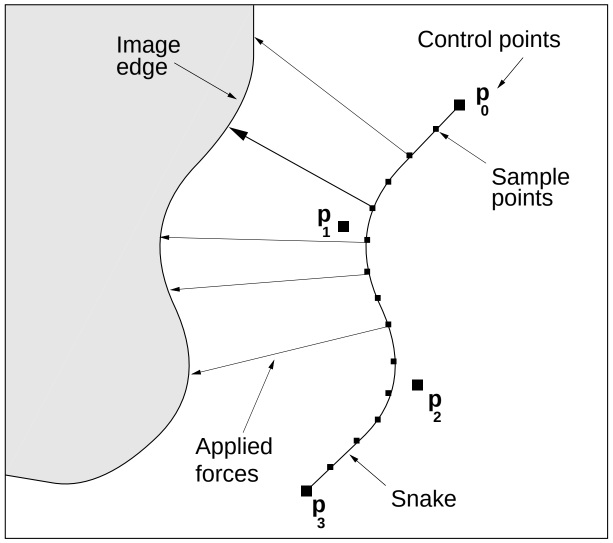
What makes B-splines especially suited to edge fitting is that they exhibit *local control*: modifying the position of one control point causes only a small part of the curve to change.



Active contours — "Snakes"

B-splines can be fitted to image edges using a technique called **active contours** or **snakes**.

1. Initialise the B-spline near the edge.
2. Select a number of evenly spaced sample points along the B-spline.
3. From each sample point, search normal to the spline for an edge in the image (using standard edge detection techniques).



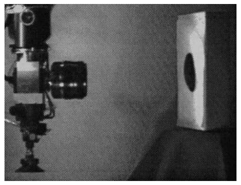
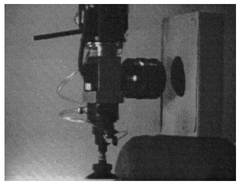
4. Apply an elastic "force" at each sample point, proportional to the distance to the edge.

Active contours — "Snakes"

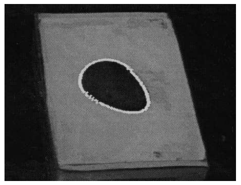
5. Calculate the elastic energy $E(I(x,y),\mathbf{Q})$ associated with the forces.
6. Move the control points to minimize E (least squares).
7. Repeat from (3).

The algorithm converges to produce a spline which closely follows the edge in the image.

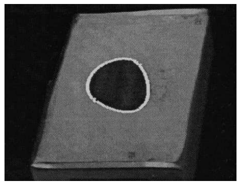
By running the algorithm continuously, if the edge moves, then the spline moves with it. Thus we have an extremely useful **contour tracker**. Tracking curves in image sequences allows us to reconstruct shape from contour, or estimate time to contact for navigation or visual docking.

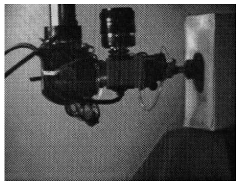
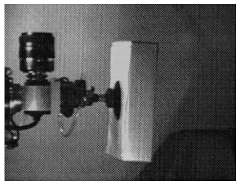
Exploratory movements



Contour …



tracking

Docking and grasping

Bibliography

Some of the illustrations in this handout were taken from the following publications, which also make good further reading.

Edge detection

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

Corner detection

C. Harris. Geometry from visual motion. In A. Blake and A. Yuille, editors, *Active Vision*, pages 263–284. MIT Press, Cambridge MA, 1992.

Snakes

M. Kass, A. Witkin and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

B-spline snakes and docking application

R. Cipolla. *Active Visual Inference of Surface Shape*. Lecture notes in Computer Science (1016), Springer-Verlag, 1995.