Computer Vision

Vision is a process which produces a description from images of the external world that is useful to the viewer and not cluttered with irrelevant information.
Hierarchy of representations: bottom-up from an image to a 3D world model: pixel array (the image), raw primal sketch (edge representation), primal sketch (edge groupings), 2.5D sketch (depth information), 3D world model.
Common tasks: digitisation, feature (edges, corners, regions) detection, feature grouping, characterisation of parts, object recognition.

Images

An **image** is a description of the environment from the viewpoint of an observer, with information in 2D only, producing a 2D array of picture elements (indexed horizontal first, from top-left to bottom right). The meaning of value of each pixel depends on the sensor of the camera. The image can be represented in the **frequency domain** with **Fourier transform** (more details to be discussed).
Example of sensors: optical (CCD, photodiodes, etc.), infra-red (thermal), SAR (radar), range sensors (laser), PET/CAT (magnetic or radio, with Fourier analysis generating images).

Aperture

Light consists of electromagnetic waves which come in a whole spectrum defined by the wavelength of waves. The shorter the wavelength, the better resolution is obtained by the sensor.
The **aperture** is a medium through which the light waves can travel, such as a lens or an air gap. As a wave, light is subject to diffraction (lit up ordinarily dark areas). The smaller the aperture, the larger the diffraction spreading (subject to a fundamental limit).

Synthetic Aperture Radar (SAR)

Radio waves transmitted actively from SAR, the reflection of which are then picked up by an antenna, received by the synthetic aperture and falsely coloured based on time of flight and azimuth phase shift. Similar devices include sonar and ultrasound. Footprint of waves on the land defines the resolution, dependent on antenna size.

Simple Filter

A **filter** is an operation on the image that loses information to create a new image with the required information in it. It is usually denoted by a grid or mask (often 3x3 size, but varies).
Simple **shift-invariant** or **stationary filter**: The grid is placed over the image at each pixel, and the *original* neighbourhood pixel values are each multiplied with the corresponding value in the grid. Summing up all multiplied values and dividing by the number of pixels in the neighbourhood (e.g. 9) to replace the pixel value, creating a pixel in the filtered image at the same place.
Edge handling: various methods for dealing with mask lying partly off the edges, including wrapping around to other side, assuming zero pixels, and ignoring edge pixels.
Example 3x3 filters: **smoothing** (all 1/9), **blurring** (corners 0, non-corner sides 1/8, centre 1/4), **Sobel Edge** (sharp transition) **Finding** (-1 -2 -1; 0 0 0; 1 2 1 in either column (rightwards-x) or row (downwards-y) direction), **Smoothed Edge Finding** (-1 -1 -1; 0 0 0; 1 1 1 similar with Sobel).

Describing and Modelling Images

At the lowest level, pixels exist without context but are clearly supposed to be related to each other which is further complicated by noises. Therefore, the best way is to treat pixels as random variables and use statistics — creating a **discrete random field**.
Common measurements: mean, covariance between two images.
Vectorisation: use matrix formula to compute statistical quantities. 2D to 1D position: *i* = y * width + x. For dot-product and matrix multiplications:

$$\mathbf{a}\cdot\mathbf{b}=\sum_i a_i b_i=\sum_x\sum_y a_{x,y}b_{x,y}$$

$$(\mathbf{AB})_{i,j}=\sum_k A_{i,k}B_{k,j}=\sum_a\sum_b A(\boxed{x,y},a,b)B(\boxed{a,b},\boxed{v,w})$$

We often consider images as spatially invariant (stationary), the probability distributions are the same regardless of position.

$$\mu(i, j) = \text{constant} = \mu$$
$$r(i, j; a, b) = r(i - a, j - b)$$
$$r(i, j) = r(i, j; 0, 0)$$
$$P(\mathbf{X}_{i,j}) \equiv P(\mathbf{X}_{a,b})$$
and
$$P(\mathbf{X}_{i,j}, \mathbf{X}_{i+k,j+l}) \equiv P(\mathbf{X}_{a,b}, \mathbf{X}_{a+k,b+l})$$

Linear Orthogonal Transform

Essentially replace the axes of the image to allow better compression or information retrieval, achieving image transformation.

Delta Functions

Functions that return zero for every input except one, in which case returns 1.

Kronecker δ: for discrete maths, returns 1 at a defined x = *a*.

Dirac δ: for continuous maths, returns ∞ at a defined x = *a*, integrates to 1.

Orthogonal Basis Sets

With basis vectors (represented as *i*, *j*, *a*, *b*, etc.), we find the coefficients of these vectors. If $\{x_1, x_2\}$ is the basis set, then a point **p** is given by $\mathbf{p} = a_1x_1 + a_2x_2$, and compute dot products $\mathbf{p}x_1 = x_1\, a_1x_1 + x_1\, a_2x_2$, $\mathbf{p}x_2 = x_2\, a_1x_1 + x_2\, a_2x_2$, creating two simultaneous equations.

If we then find the special case of basis set $\{x_1, x_2\}$ that $x_1x_2 = \delta(x_1 - x_2)$, i.e. unit length vectors at right angles to each other, then the two vectors are orthogonal or othornomal. Coefficients $px_1 = a_1$, $px_2 = a_2$ because the other term is zeroed by orthogonality, allowing much easier coefficient computation.

Describing Every Point

For a *n*-point image, it is possible to construct a point **p** from standard basis vectors $\{e_1, e_2\}$, i.e. $\mathbf{p} = \mathbf{p}_1e_1 + \mathbf{p}_2e_2$. For completeness, if we express coefficients $a_1 = e_k\, x_1 = x_{1k}$, $a_2 = e_k\, x_2 = x_{2k}$, then we can construct e_k with an orthonormal basis set: $e_k = a_1x_1 + a_2x_2 = x_{1k}\, x_1 + x_{2k}\, x_2$, which can be written in components as $e_{ki} = \delta(i - k) = \sum_j x_{jk}x_{ji}$.

Linear Transformation

In addition to computing the vector of coefficients, if we know a vector of coefficients, along with the basis set, we can compute **p**. This is a bidirectional lossless mapping.

Matrix form: This process can also be represented in the matrix form, which is a *n* by *n* matrix with basis vectors as rows, where *n* is the size of the basis vector. Then $\mathbf{p} = T^Ta$, thus $a = (T^T)^{-1}\mathbf{p}$. T is a matrix which rotates the vector space.

Orthonormal transforms: to be able to describe every point as required above, a basis set is said to be orthonormal, that is, $TT^T = I$, $T^{-1} = T^T$. So $a = (T^T)^{-1}\mathbf{p} = (T^{-1})^{-1}\mathbf{p}$ can be expressed as $a = T\mathbf{p}$.

Transforming Images

The **p** mentioned in the previous sections represents an individual point in an image. In **image transformation**, we convert an image from one representation to another with a set of basis images. To express the aforementioned properties with regarding to an image with dimensions (N, M) on each point (x, y) instead of using vectorisation, modifications are needed as followed:

Image basis set: $\{B_{1,1}(x, y), B_{1,2}(x, y), \ldots B_{N,M}(x, y)\}$, $I(x, y) = a_{1,1}B_{1,1}(x, y) + a_{1,2}B_{1,2}(x, y) + \ldots + a_{N,M}B_{N,M}(x, y)$, and we need to have the same number of basis images as there are pixels (N, M).

Image orthonormal condition: $\sum_x\sum_y B_{i,j}(x, y)B_{k,l}(x, y) = \delta(i - k, j - l)$.

Calculating image coefficients: $a_{i,j} = \sum_x\sum_y B_{i,j}(x, y)I(x, y)$.

Image coefficient (*k* of them) **completeness**: $\sum_k B_k(x, y)B_k(a, b) = \delta(x - a, y - b)$.

A full transformation is done by first calculating the basis set of the image on each point (pixel) in the image, reconstructing them by multiplying the basis images with their standard basis vectors, and then sum the products together, to get the transformed image.

If we arrange the basis images (vectorised) into rows in a matrix, then pre-multiplying the image vector by T will calculate the coefficients and therefore the transformed image.

The Karhunen-Loeve Transform

A method to make pixels uncorrelated with each others, similar to **principal component analysis (PCA)**. First calculate a symmetric and positive-definite covariance matrix, then take the complete and orthonormal eigenvectors to form basis images. This creates new coefficients that are **uncorrelated**, with maximum amount of image energy in the smallest number of coefficients.

$$\mathbf{Y}^* = \begin{bmatrix} (\mathbf{T}\vec{y}_0)^{\mathrm{T}} \\ (\mathbf{T}\vec{y}_1)^{\mathrm{T}} \\ \vdots \\ (\mathbf{T}\vec{y}_{n-1})^{\mathrm{T}} \end{bmatrix} = \mathbf{Y}\mathbf{T}^{\mathrm{T}}$$

$$\Sigma^* = \frac{1}{n}\mathbf{Y}^{*\mathrm{T}}\mathbf{Y}^* = (1/n)\left(\mathbf{Y}\mathbf{T}^{\mathrm{T}}\right)^{\mathrm{T}}\mathbf{Y}\mathbf{T}^{\mathrm{T}} = (1/n)\mathbf{T}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\mathbf{T}^{\mathrm{T}}$$
$$= (1/n)\mathbf{T}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\mathbf{T}^{\mathrm{T}} = \mathbf{T}\Sigma\mathbf{T}^{\mathrm{T}}$$

$$\Sigma^* = \begin{bmatrix} \vec{e}_0^{\,\mathrm{T}} \\ \vec{e}_1^{\,\mathrm{T}} \\ \vdots \\ \vec{e}_{K-1}^{\,\mathrm{T}} \end{bmatrix} \cdot \Sigma \begin{bmatrix} \vec{e}_0 & \vec{e}_1 & \cdots & \vec{e}_{K-1} \end{bmatrix} = \begin{bmatrix} \vec{e}_0^{\,\mathrm{T}} \\ \vec{e}_1^{\,\mathrm{T}} \\ \vdots \\ \vec{e}_{K-1}^{\,\mathrm{T}} \end{bmatrix} \cdot \begin{bmatrix} \lambda_0\vec{e}_0 & \lambda_1\vec{e}_1 & \cdots & \lambda_{K-1}\vec{e}_{K-1} \end{bmatrix} = \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & \lambda_{K-1} \end{bmatrix}$$

Other Possible Transformations

Hadamard: Binary basis images for fast hardware implementation, but hard to analyse.

Discrete Cosine Transform (DCT): fast with real number operations only, close to optimal for highly correlated images. Used for compression. It is orthonormal. Algorithms as shown.

Discrete Sine Transform (DST): even faster than DCT, also with real number operations only. Used in block processing and filtering with good energy compaction.

1D $$b_k(i) = \sqrt{\frac{2}{n}}\cos\left[\frac{\pi}{n}k(i+\frac{1}{2})\right]$$ Indices run from 0 to n-1

2D $$B_{kl}(i,j) = \sqrt{\frac{4}{NM}}\cos\left[\frac{\pi}{N}k(i+\frac{1}{2})\right]\cos\left[\frac{\pi}{M}l(j+\frac{1}{2})\right]$$

Discrete Fourier Transform (DFT, $F(u, v)$): used in convolution and filtering with high energy compaction, but requires complex arithmetic. It is an important signal processing algorithm. It puts DCT and DST together with complex numbers (imaginary possible in transformed image). It moves processing from the image domain to the frequency domain. Inverse ($I(x, y)$)is possible.

A continuous version of DFT is available for working with functions.

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y)e^{-2\pi iux/N}e^{-2\pi ivy/M},$$

$$I(x,y) = \frac{1}{NM}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{2\pi iux/N}e^{2\pi ivy/M}$$

Convolution and Correlation Filter

A **convolution filter** is a linear and shift-invariant filter (same operation on each pixel). It will result in information loss.

$$g \otimes I = I'(x,y) = \sum_{a,b} g(a,b)I(x-a, y-b)$$

Any number of passes can be applied to an image, and for an image with dimensions (N, M), N^2M^2 number of operations need to be performed at each pass.

With Fourier Transform, convolution can be calculated in $NM\log[MN]$ as shown. This requires both N and M being power of 2.

$$f \otimes g = \mathcal{F}^{-1}\left[\mathcal{F}(f)\mathcal{F}(g)\right]$$

A **correlation filter** is of similar form to the correlation filter, except that it does *not* apply a rotation of 180 degrees to the filter applied, therefore it is *not* associative, which

$$g \oplus I = \sum_{a,b} g(a,b)I(x+a, y+b)$$

$$\mathcal{F}(f \oplus g) = \mathcal{F}(f)^*\mathcal{F}(g)$$

makes it not suitable for use on asymmetric filters.
With Fourier Transform, correlation can be written as:
The * represents the complex conjugate, reversing the signs of all complex numbers in the expression denoted.

Linear Filtering
Based on the convolution theorem, various types of linear filters can be implemented: noise reduction, low/high pass, edge enhancement and de-blurring. Examples:
Spatial Averaging Filter: Each pixel is replaced by a weighted average of its neighbourhood pixels, useful for sub-sampling and noise reduction. Requires a limited window for accurate constant value approximation, reducing noise from variance σ^2 to σ^2/N.
Gaussian Filter: Popular noise reduction filter. It is a weighted neighbourhood averaging filer that is symmetric and separable.

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

High-pass Filter: Removes low frequency components from the spectrum, useful for removing a slowly varying background. This emphasises textured regions or edges.
Low-pass Filter: Removes high frequency components from the spectrum, useful for removing high frequency details. This emphasises uniform regions.
Median Filter: Replace the pixel value with the median of neighbourhood, useful for removing limited quantities of noisy pixels.
Edge Filtering: Enhance the image where there is an **edge** (a transition from one value to another), either by approximating difference between adjacent values (increasing noise), or by differentiating a noise-smoothed image with a difference operator. The Gaussian filter is a good noise-resistant edge detector.
Laplacian Operator: By taking gradients in the x and y directions we can find the maximum gradient of the edge and its direction, this allows us to detect edges. It is isotropic (direction does not matter), with zero crossings at edge locations. We apply Laplacian to a noise-smoothing filter such as Gaussian to achieve effective edge detection.

$$\nabla^2 I=\frac{\partial^2 I}{\partial x^2}+\frac{\partial^2 I}{\partial y^2}$$

$$\nabla^2 G=\frac{1}{\sigma^2}\left[1-\frac{x^2+y^2}{\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Noise Modelling
A common way of modelling the imaging system as a whole is as a **linear system with noise**.
If *I*(x, y) is the 'true' image, the output of the system is *I'*(X, Y) = T(*I*) + n, where T is the linear transform by the image capture system, and n is random noise. Also possible to assume T is stationary and n is uncorrelated: *I'*(X, Y) = T *convolute I* + n.
Types of noises:
CCD: non-additive **Poisson**, with mean μ.
Several unconnected noise processes: additive Gaussian (common).
SAR: multiplicative Raleigh-Bessel.

$$P(I(x,y)=a)=\frac{\mu^a}{a!}e^{-\mu}$$

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

Covariance: If the noise is identically distributed with no bias or pattern, it is possible to calculate statistics from the (random) samples. The formula for calculation of covariance coincides with the correlation of the noise with the image.
Noise spectral power: SP(n) = F(n *correlate* n). n *correlate* n is zero except at (0, 0) for random noise due to zero covariance between independent variables. At zero, covariance is noise variance Var(n), which is also the value of spectral power SP(n).

De-convolution and Wiener Filters
To reconstruct the origin image from the pixels of the image after degradation by convolution of the sensor, we want to remove the transformation T and possibly remove noise n. Process as shown.

$$F(I)=\frac{F(I')}{F(T)}=\frac{F^*(T)F(I')}{F^*(T)F(T)}$$

$$I=F^{-1}\left(\frac{T^*}{|T|^2}\right)\otimes I'$$

Simple application (de-convolution) of the Fourier Transform of the inverse filter is unstable, especially in the presence of image noise. The **Wiener Filter** is designed to work with image noise.
The Wiener Filter is a linear filter minimising the squared error between the original and estimated images. Expressed as shown below.

$$G(u,v)=\frac{H^*(u,v)S_{II}(u,v)}{|H(u,v)|^2 S_{II}(u,v)+S_{nn}(u,v)}$$

where S_{ll} is the spectral power of the original image without noise, S_{nn} is the spectral power of the noise in the image. When S_{nn} is small, this is just the inverse filter.
The Wiener Filter mathematically returns the best results in image de-blurring.

Advanced Edge Detectors
Simple edge detectors such as Sobel have several disadvantages, such as missing corners due to their reliance on one-dimensional gradients. The user must adjust the threshold. To prevent errors such as missing or false edges, particularly in textured or noisy images (possibly with variations in illumination), more advanced edge detectors are required.
Canny Edge Detector (CED): good positive detection, good localisation, and unique response for each edge. The steps of CED:
- Applying 2D Gaussian smoothing to the image with a 5x1 or 7x1 mask.
- Compute gradients along both axes by calculating derivatives.
- Compute gradient magnitude and direction of each pixel based on the derivatives, track along the directions in the neighbourhood, and linearly interpolate gradient magnitudes. If the magnitude of central pixel of the neighbourhood greater than both interpolated magnitudes, an edge has been detected.
- Store the edge's sub-pixel position (by fitting a parabola), gradient strength and orientation/ direction.
- Link edges together into strings via their orientations.
- Perform thresholding with hysteresis, using two thresholds: a higher threshold for segmenting clearly-separate edge segments, and a lower threshold for connecting segments.

Statistical Edge Detectors (SED): edge detection with machine learning techniques, e.g. maximum log-likelihood edge detector. It works by determining the narrow area with near equal probabilities between pixels from two characteristics regions: $p_1(X|\Theta_1, k) = p_2(X|\Theta_2, M-k)$. $p(x|\mu, \sigma)$ for Gaussian distributions. The two regions are modelled as probability density functions (PDFs), which are assumed to segment the region correctly into two parts with equal statistical probability at all pixels of the edges we are after (of a certain brightness value).

$$\prod_{i=1}^{k} p_1(x_i \mid \Theta_1) = \prod_{j=1}^{M-k} p_2(x_j \mid \Theta_2)$$

Assumptions of SED: all pixels in one region are characterised by the same probability — independent and identically distributed; pixels are uncorrelated. Hypothesis testing:
Log-likelihood Algorithm:
Local estimate of grey-level for each region. In detecting the edge, compare the log-likelihood of having only one region with that of having two regions, with a threshold such as $\lambda = 0$ when the k value becomes the most appropriate for segmenting the two regions along a new edge.

$$\lambda = k \ln\left(p_1(\hat{x}_1 \mid \Theta_1)\right) - (M-k)\ln\left(p_2(\hat{x}_2 \mid \Theta_2)\right)$$

Practical solution of log-likelihood: in a practical problem, we often assume the two regions are equal in size i.e. k = M - k. The set of parameters Θ is represented as the mean and variance of the region data, and with Gaussian distribution of the PDF, $p_1(X|\Theta_1)$ and $p_2(X|\Theta_2)$ can be replaced by the formula shown, and a quadratic equation can be solved for $p_1 = p_2$ to solve for x as the log-likelihood estimate.

$$p(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{\sigma^2}\right]$$

Comparison of the two edge detection methods: Edge techniques using local gradient are appropriate in the case when images display sharp contrasts while statistical decisions are more appropriate when when images are noisy and display local variance. Local gradient techniques will fail in the case when the image display significant noise or statistical variation. Local gradient is fast calculated using masks (usually 3×3), while local statistical decisions take much computational power, requiring the knowledge of local statistics model and the calculation of various parameters. Local gradient methods are appropriate for images taken under visible illumination conditions while edge techniques employing statistical decisions are more appropriate frequency domain images like sonar images.

Corner Detection

A simple corner detector: pick a small block of pixels, determine the auto-correlation around each pixel position, then rotate the block and recalculate auto-correlation. Pixels where the two auto-correlations are both maximum can be marked as a corner.

This can be done more cheaply with change of intensity: $E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v)-I(x,y)]^2$, where w is the window function (binary or Gaussian), $I(x+u, y+v)$ is the shifted intensity and $I(x,y)$ is the original intensity.

With a Taylor series, we can approximate the image at each pixel with local image gradients, and then approximate change of image intensity by neglecting the higher derivatives.

The result is that a small shift [u, v] can be expressed with a bilinear approximation:

$E(u,v) \sim= [u,v]\ M\ [u; v]$, where $M = \sum_{x,y} w(x,y)[I^2_x\ I_x \cdot I_y; I_x \cdot I_y\ I^2_y]$ with image derivatives.

With eigenvalues λ_1, λ_2, we can analyse the properties of each point by observing the characteristics of the two eigenvalues: if the two differ by much, it is likely an **edge**; if the two are both large but similar in value, it is likely a **corner**; otherwise, it is likely a "flat" region.

Harris Corner Detector (HCD): a detector measuring corner response. $R = det\ M - k(trace\ M)^2$; where $det\ M = \lambda_1\lambda_2$ and $trace\ M = \lambda_1 + \lambda_2$ by adding or multiplying the two eigenvalues. k is an empirical constant, k = 0.04-0.06. In the above classification system: large positive R for a corner; large negative R for an edge; small magnitude for a flat region.

Process of HCD: compute derivatives of the image along both axes, compute products of the derivatives at every pixel (I_{xx}, I_{xy}, I_{yy}), then compute the sums of the products of the derivatives at each pixel $[S_{xx}\ S_{xy}; S_{xy}\ S_{yy}]$ (convolution with weighting function at this step). Compute M based on the sums, and apply M to calculate response value R. Essentially $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ due to the eigenvalue properties. Threshold R to identify corners. Pixels with a neighbour with higher R value will not be considered again as a corner has already been identified.

Possible use of HCD: matching views of different orientation like epipolar lines; motion estimation; camera calibration.

Boundary Representation

Boundary representation is the process of describing boundaries in image data. One of the easiest methods is **contour following**, which uses a chain coding algorithm with numbers to represent each step of the boundary relative to the previous step, with special codes for bifurcations and crosses.

An alternative method is to model a boundary by splitting it into segments with certain local curvature characteristics, e.g. *a* for a half-circle and *b* for a straight line, so a finger can be approximated as *bab*.

Yet another alternative method is to calculate the centre of the object, and map distance from the centre to locations on the boundary contour. This produces a plot identifying the distances to each point on the boundary, with peaks and troughs indicating the corners.

A more generic method is the **Line Split/Merge Algorithm**: fit a straight line to all edges in string, if root mean square (RMS) error less than threshold then we accept the fitting for that edge and stop. Otherwise, find the point with the highest curvature on this line, split the line into two there, and repeat for each sub-line. The merge algorithm works similarly by checking whether merging adjacent pairs of lines fitting consecutive edges results in an RMS below threshold. If below threshold then merge and repeat until no such pairs can be found.

However, a straight line obtained by solving linear equations of the line between two endpoints of an edge segment is likely not the best representation (unless edge happens to be a straight line). It is usually better to use **Least Squares Fitting** (LSF) to fit all points along the edge segment. This is done by minimising the polynomial error function.

An alternative to LSF is to use eigenvectors, minimising the error in the direction perpendicular to the fitted (initially straight) line. An error function representing the perpendicular distance is minimised by moving the point in either perpendicular direction at fixed intervals along the line.

Hough Transform

Instead of using a linear function, **Hough Transform** represents a straight line as a point (s,θ) in the polar coordinates system $s = x\cos\theta + y\sin\theta$ as a parametric representation of borders and edges.

Hough Transform can be used to map straight lines: get a list of boundary points from the image, map each point to a (s,θ) polar coordinate. Count the number of edge points mapping into a certain location, if $x_i\cos\theta + y_i\sin\theta = s_k$ for $\theta = \theta_l$, then Count(s_k, θ_l) += 1— the intersection count at that polar point increments as one more line intersects at that discrete bin. A peak in this accumulator array represents a line (common direction and location). Multiple peaks can be found if there are multiple straight lines, with the dominant peak corresponding to the longest line.
Similar techniques can be used to map circles, using the centre of the circle and the radius r. The process essentially maps out straight lines that are normals to the circle, which intersect in certain locations. Peak intersection points in the accumulator array suggest locations of centres of circles. Distances from such centres of circles to locations on edges accumulate in the space of circle radii r. Choose the highest peak in the space for each circle centre location and radius.

Curve Interpolation with Contour Models
Contour models interpolate curves, that is, the approximation of the curve using piecewise linear approximation and cubic spline approximation.
We divide curves into curve segments, each defined by endpoints, tangent vectors and continuity requirements, and represent each segment in a parametric format. We need to know the curve at N points when using a N-dimensional polynomial to express a curve segment. In the cubic case (four-dimensional polynomial), the **perfect interpolation** can be calculated by taking four points and express the model in a matrix form as shown. The parametric matrix should be invertible.

$$\overline{p}_x = \begin{bmatrix} p_x(1) \\ p_x(2) \\ p_x(3) \\ p_x(4) \end{bmatrix} = \overline{T} \cdot \overline{C} = \begin{bmatrix} 1 & t_1 & (t_1)^2 & (t_1)^3 \\ 1 & t_2 & (t_2)^2 & (t_2)^3 \\ 1 & t_3 & (t_3)^2 & (t_3)^3 \\ 1 & t_4 & (t_4)^2 & (t_4)^3 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\overline{C} = \overline{T}^{-1} \cdot \overline{p}_x$$

Cubic B-splines
Perfect interpolation may not be desirable due to insufficiently smooth resulting curves. An alternative is to use control points to control the curve through blending polynomials. Assuming first and second derivative continuity, four control points can define a **convex hull** (to be covered later) to contain curve segments. A few knots (sample points on the curve) control the modelling along with the control points. The blending functions are $(1-t)^3$, $4-6t^2+3t^3$, $1+3t+3t^2-3t^3$, t^3 respectively. Sharp corners in the curve can be modelled by repeating the control points.

Active Contour Modelling
Active contour models (“Snakes”) are used to define contours of objects and regions in images with parametric curves. It requires a certain contour initialisation to begin with, and uses an energy form with the curve. This allows it to find the object boundary with an energy minimisation problem.
Each contour has an energy $E_{snake} = E_{internal} + E_{external}$, defined in a way such that the final contour position will have a minimum energy.
$E_{internal}$ is the sum of elastic and bending energy, depends on the intrinsic properties of the curve.
The **elastic energy** measures the elastic potential of the curve as if it’s a rubber band, discouraging over-stretching by introducing tension. It is responsible for shrinking the contour. α is an adjustable weight to control elastic energy.
The **bending energy** encourages the stiffness of the curve as if it’s a thin metal strip, and is modelled with another adjustable weight β. It tries to smooth out the curve under elastic energy.
$E_{internal} = E_{elastic} + E_{bending} = \int_s 0.5(\alpha\, |v_s|^2 + \beta\, |v_{ss}|^2)ds$, where v_s and v_{ss} are the first and second derivatives of the contour v(s) (i.e. the gradient and curvature respectively).
$E_{external}$ is derived from the image, minimal at boundaries. $E_{external} = \int_s E_{image}(v(s))ds$. E_{image} can be defined as the gradient at each point, or the gradient of the image convoluted with a Gaussian kernel.
With the full energy model $E_{snake} = \int_s 0.5(\alpha\, |v_s|^2 + \beta\, |v_{ss}|^2)ds + \int_s E_{image}(v(s))ds$, we want to find the correct value of v(s) (and thus its first and second derivatives) to minimise the energy function. With Euler-Lagrange differential equation, this is when $\alpha v_{ss} - \beta v_{ssss} - \nabla E_{image} = 0$, when applied to a set of specific points on the contour.

Under Gaussian noise, results of the modelling is better in regions where initialisation is close to object boundary, worse elsewhere. Elastic energy may dominate in areas with weak edges, leading to errors in the final contour.

Binary Image Analysis

Binarization: the process of using a threshold to convert a grayscale image into a black-or-white image with two levels, useful for extracting shape information.

Selection of binarization threshold: it can be fixed if image characteristics are known from acquisition or pre-analysis; it can be set to a predetermined fraction between the min and max intensity level values; it can be set based on the shape of the histogram — best using local minima between two peaks. The optimal way to set the threshold however, is to use **probability density functions** (PDFs) of a certain model. Usually the model is Gaussian.

With two PDFs crossing at a centre location, the point where the two PDFs are equal is chosen as the threshold. This is identical to the process of determining the area of highest probability in **Statistical Edge Detectors** discussed earlier ($p_1(X \mid \Theta_1) = p_2(X \mid \Theta_2)$ where $\Theta = \{\mu, \sigma\}$).

Entity Labelling: the process of attaching a value (label) to each blob of connected pixel, uniquely naming an entity. The kinds of entity named can be edges, corners, segmented objects/regions, etc. Assign the same labels to all pixels in a neighbourhood if they are connected.

Region Adjacency: associations between objects of different labels based on adjacency, which can be expressed with a connected adjacency graph. The background (neighbouring every other object) may be a node in the graph.

Shape Characterisation: descriptors describing simple shapes, such as area (number of pixels), elongatedness (ratio between length and width of the bounding triangle, may not be relevant in curvy, thin objects), direction (orientation), and compactness (ratio of square of the border length and its area).

Convex Hull: the smallest region which contains the object such that any two points of the region can be connected by a straight line such that all points on which belong to the region.

Projections: sum of number of pixels along each row (horizontal projection) or column (vertical projection) of the image.

Moments: represents characterisation of shapes. Zero moment (i, j = 0) represents the area. Normalising with **central moments** provide translation invariance. This is done by dividing the moment obtained from displacing points relative to the centroid with the zero moment. The centroid coordinates x_c and y_c can be calculated by dividing first order moments $i\ f(i,j)$, $j\ f(i,j)$ with the zero moments.

$$m_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} i^p j^q f(i,j)$$

The first order elements (i + j = 1 => 01, 10) also show the centre of mass, while the second order (i + j = 2 => 02, 20, 11) elements show the axis of orientation.

With binary images, f(i,j) is 1 when location is inside the shape, and 0 otherwise. The second order moments show how spread is the shape in a certain direction (e.g. horizontal, vertical or direction of orientation). Other metrics include skewness (third order moments, level of distortion from symmetry) and Kurtosis (fourth order moments, peakedness of shape).

The direction of orientation can be calculated from the second order moments $\theta = 0.5\tan^{-1}(2\mu_{11}/\mu_{20} - \mu_{02})$.

Mathematical Morphology Analysis

Mathematical **morphology** studies the objects with simple operators employing structural elements, used to smooth shapes or describe them from binary images. This can also be extended to grey-level images.

A mathematical morphology is defined by transformation functions (operators) and structuring elements (square, rhombus, circle, etc.). Simple transformations include erosion and dilation.

Dilation: the **Minkovski set addition** (adding each vector in one set to its corresponding vector in another) of the structuring element to the set X. $X \oplus B = U_{b \in B} X_b$. It is an expanding operator, creating a uniform expansion layer of elements with width given by the structuring element. Dilation of a set X includes all translations of X on structuring elements B that have common points with the initial set X.

Dilation is commutative, associative, distributive, and increasing (maintaining subset relations).
Erosion: the **Minkovski set subtraction** (subtracting each vector in one set from its corresponding vector in another) of the structuring element from the set X. $X \ominus B = \cap_{b \in B} X_b$. It is a shrinking operator, iteratively peeling a set with a uniform layer of elements having the width of the structuring element. Erosion of a set X includes all the translations of X on the structuring elements B which are included in the original set X.
Erosion followed by dilation is called **opening** ($X_B = (X \ominus B) \oplus B$, removing some sharp edges), while dilation followed by erosion is called **closing** ($X^B = (X \oplus B) \ominus B$, filling some gaps). Both combinations perform noise reduction and smoothing, but opening is a reduction operator while closing is an extension operator.
Thinning: a process of eroding pixels iteratively, stopping at the desired thickness by observing the neighbourhood.
Skeleton: the locus of the centres of the maximal discs indescribable inside an image object, created by thinning down to a single pixel in width. It gives a description of the object structure. To be discussed in more detail later.
By using simple mathematical morphology operations such as dilation and erosion, more complex functions such as **shape morphing** can be constructed. Shape morphing make two sets approach each other in shape and size, expanding one of the sets in regions where the other is defined, and eroding in regions where the other is not defined. It stops when the transformed shapes morphed from the two sets become identical. This is the same as the skeleton of the difference shape between the two sets.

Image Segmentation
Segmentation is a process of gathering features that belong together, all defining a certain region in the image or a certain object. Associating a model with observed features is called **fitting**. **Top-down** and **bottom-up** segmentation possible, the former of which groups pixels from the same object together, while the latter groups pixels together if they look similar.
Segmentation is a method of visual grouping, which may also include grouping by homogeneity (grey-level, colour, etc.), proximity, and continuity.
Segmentation can be **supervised** (with *a priori* knowledge) or **unsupervised**. Two main categories of methods exist for image segmentation: **clustering** and **partitioning**.

Clustering Segmentation
Image clustering is the process of clustering similar pixels (with features such as colour) together. The purpose is to achieve high **intra-class similarity** and low **inter-class similarity**. We infer the class labels and the number of classes directly from the data, in contrast to a supervised classification.
The intra-class similarity can be measured by summing up the squared distances of each node of each class to the class' centre, and summing up the sum for each class to get the objective function.
K-means Algorithm: a method to achieve clustering. First decide on a value for k (the number of clusters), then either randomly or with priori knowledge initiate k cluster centres. Then we repeat the process of deciding class memberships of N objects by assigning them to the nearest cluster centres, re-estimating the k cluster centres by assuming the membership assignment correct, and checking again whether all N objects are still assigned to the correct cluster centre. If any requires reassignment due to distance to nearest cluster centre has changed, then the process continues. Otherwise, the clustering is complete.
K-means clustering based on intensity or colour is essentially vector quantisation of the image attributes. The resulting clusters may not be spatially coherent.
Advantages of K-means: simple to implement, and it converges to a local minimum of the error function.
Disadvantages of K-means: it is memory intensive, requires explicit choice of K clusters. In addition, it is sensitive to non-optimal initialisation and outliers. It can only find spherical clusters.
Mean shift algorithm: an alternative method to K-means. It seeks a mode/local maximum of density of a given distribution. This works by first choosing a search window of certain width and

location, the computing the centre by using data located inside the window assuming a certain distribution (e.g. Gaussian). Then it locates a new search window around the new centre, and consider the data located inside the new window to repeat the process. This repeats until no centre change is required. The algorithm as represented as shown, where ψ(X) is the Gaussian cost function in a neighbourhood of the centre X. σ measure the window size, while d is the number of dimensions considered.

$$M_\sigma(X) = \frac{\sigma^2}{d+2}\frac{\nabla\psi(X)}{\psi(X)} = \frac{\sum_{i=1}^{N} X_i \exp(-\frac{||X-X_i||^2}{2\sigma^2})}{\sum_{i=1}^{N} \exp(-\frac{||X-X_i||^2}{2\sigma^2})} - X$$

With the application of mean shift, all data points will be clustered in the attraction basin of a mode/ local maximum. The attraction basin is the region for which all trajectories lead to the same mode.
In feature finding, windows can be initiated at individual pixel locations. Mean shift is performed for each window until convergence. Windows ending up near the same mode can be merged.
Advantages of mean shift: does not assume spherical clusters like K-means. It requires a single parameter (window size) and finds variable number of clusters. It is robust in dealing with outliers.
Disadvantages of mean shift: the output is dependent on window size. The process is computationally expensive, and does not scale well with higher dimensions of feature space.

Partitioning Segmentation (Graph Cuts)
Partitioning is an alternative to clustering in segmenting images. We consider pixels or block of pixels, and represent features and their relationships using a graph of affinities between nodes. We draw an edge between every pair of pixels, and weight the edges by the affinity (similarity) of the two pixels in the pair. Finally, we cut graphs into subgraphs with strong **interior links** and weak **exterior links**.
A similarity matrix W = floor($W_{i,j}$) where the edge weight between nodes i and j $w_{i,j} = \exp(-||x_i - x_j||^2 / \sigma^2_X)$ is constructed. For each node i with j number of connecting nodes, the sum of its weight is $d_i = \sum_j w_{i,j}$. We can then make cuts in a graph to separate partitions separates by a common set of low-weight (low affinity) edges: *cut*(A, A') = $\sum_{i\in A,\ j\in A'} w_{i,j}$. This results in similar pixels being in the same segment, and dissimilar pixels being in different segments.
Features such as intensity, distance and texture can be represented by $w_{i,j}$, which could be Fourier coefficients or filter outputs.
In calculating $w_{i,j}$, a low σ value means only nearby points are grouped, while a high σ value means far-away points can be grouped together. Items can be placed into blocks by eigenvectors, allowing clear clustering regardless of row ordering.
When selecting the edges to cut, minimum cuts could penalise large segments, which can be fixed by normalising the cut by component size (dividing *cut* by sum of inverse of edge sums). Its determination can be done by computing the affinity matrix (W) and degree matrix (D) between each pair of nodes, and eigen-solve (D-W)y = λDy. Then use the eigenvector with the second smallest eigenvalue to bipartition the graph. After partitioning, decide whether repartitioning is necessary by choosing a threshold on the eigenvalues to decide the number of regions.

Motion Estimation
Motion estimation is the process of estimating 3D variations represented by image sequences, with the variation of time that can be represented as the fourth dimension in a 4D data sequence.
Several aspects may be of interest under a time variation, such as motion, the segmentation and tracking of moving objects, and coding of objects.
Optical flow is the apparent motion of brightness patterns in the image. It is supposed to be the same as the motion field, but not necessarily, as lightning changes could generate apparent motion that does not actually exist.
Brightness consistency assumption: an assumption is made that the brightness of a small region remains the same despite a possible change in the region's location.
Spatial coherence assumption: an assumption is made that neighbouring points in the scene tend to belong to the same surface and thus have similar motions, and since they also project to nearby points in the image, we expect spatial coherence in the image flow.

Temporal persistence assumption: an assumption is made that the image motion of a surface patch changes gradually over time.

Optical Flow Estimation: two possible methods: **feature-based** (detect features and match them) or **gradient-based** (consider the gradient of the temporal variation from one image frame to another). We can split each frame into blocks, and estimate the displacement of each block from one frame to another, known as the **Block Matching Algorithm**.

Block Matching Algorithm: each block can consist of one or more pixels, and is surrounded by a larger search region around the given block. In the next frame, we search for the most correlated block in the same search region. A correlation filter or block-pixel differences calculations can be used. Block size should be properly calibrated to contain sufficient level of features of a small region, but not too large that no good match can be found.

Brightness constancy: no change in brightness of a possibly-moved block, represented as $I(x+u, y+v, t+l) - I(x, y, t) = 0$. A Taylor series decomposition in space and time gives: $I(x, y, t) + uI_x(x, y, t) + vI_y(x, y, t) + I_t(x, y, t) - {}_I(x, y, t) \sim= 0$, that is, $uI_x(x, y, t) + vI_y(x, y, t) + I_t(x, y, t) = 0$. The first two parts are spatial derivatives, while the third part is the temporal derivative. Together, the spacial-temporal optical flow constrains that $I_x u + I_y v + I_t = 0$, with each of I_x, $I_y v$ and I_t being the partial derivative of I with respect to the parameter.

Potential issues in block matching: optimal flow estimation may fail if regions of images have constant colour or grey-level; or if random or regular patterns exist and interfere with brightness constancy detection; or if occlusion occur such that covering and uncovering of moving objects affect the traceability of background or other objects. These issues could be mitigated with the use of a reliability factor or a hierarchical approach like pyramidal Lucas-Kanade as shown below.

Lucas-Kanade Algorithm: a differential method for motion estimation, assuming that the flow is essentially constant in a local neighbourhood of each pixel (or feature point). At each new frame, assume the same location and open a search area. Solve the basic optical flow equations for all pixels in the neighbourhood with least squares solution as shown.

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

With the displacements u and v solved, they are applied to the pixel determining the neighbourhood, thus shifting the window by (u, v), producing a new I_t value.This process is repeated until the change is small enough.

The LHS of the equation is equal to A^TA [u; v], and $A^TA = M$, the second moment matrix also used in **Harris Corner Detector** and **Binary Image Analysis**. The eigenvector associated with the larger eigenvalue of M points in the direction of fastest intensity change, with the other eigenvector orthogonal to it.

Different regions under Lucas-Kanade: low-textured region (small magnitude gradients, with two small eigenvalues), edges (either very large or very small gradients, with one large and one small eigenvalues), high-textured region (large gradients of different magnitudes, with two large eigenvalues).

Use of pyramids in Lucas-Kanade: The use of temporal gradients often fails to detect large moments. This problem is mitigated by the use of pyramids which gradually increase the sampling window for detecting large movements.

It may be possible to track the boundaries of moving objects from one frame to another, by using clustering on different features in each frame, such as motion vector, grey level and location. This involves tracking moving objects by flooding their movements with segmentation and corresponding motion vectors. Frame predictions based on difference between two consecutive frames are used in compression algorithms such as MPEG2. Some regions may be **uncovered** (occluded and undetermined), require 3D object shape information to further clarify.

Higher Level Vision

The **intermediate** and **high level computer vision** allows the handling of structure extraction, reasoning tasks (scene labelling, object recognition, etc.), and towards image understanding.

Examples: pattern invocation, motion sensation, illusory surfaces and shape-from-texture.

Computer vision is a visual perception process based on automated and integrated functions, with algorithms and representations organised hierarchically. It encompasses elements of imager processing and patter recognition.

Levels of vision: low level (**pixel** operations, intensity features — edges and corners, and frequency domain analysis as discussed earlier); intermediate level (2D/3D structure finding on pixels — regions, blobs, lines, circles, shape from shading, etc., perceptual organisation); higher level (relational structure operations — objects and their recognition, scenes). These levels can be organised into a hierarchy of processing levels, each consisting of algorithms, representation and a defined goal or output. Inter-level control required (information flow, initiate processing, etc.)

Visual Control

Control paradigms: **pyramidal** (low and intermediate level, coarse to fine resolution processing with increased resolution only in interesting regions, human-inspired); **hierarchical** (can be top-down with model-driven, bottom-up with data-driven, or mixed).

Control structures: **open loop** (predefined and static model); **closed loop** (adaptive model refinement — e.g. Kalman filtering).

Human models: **pre-attentive** — eye movements ("robot head" model, foveal vision, peripheral vision, saccade — random eye movements, and vergence); **attentive** — feature/detail focus; **non-attentive** — purposive or reflexive (specialised, evolved or goal-directed).

Marr's Theory

Marr's Theory is an established framework of levels of vision, from primal sketch to 3D models. Like most frameworks, it studies the computational theory of the image processing, the representation and algorithms involved in processing the inputs, and the hardware implementation for image processing.

Levels in Marr's framework:

images: grid of raw intensity values as captured by sensors;

primal sketch: series of intrinsic images (multi-scale zero crossings, reflectance map, stereo disparities, optimal flow, edge segments, geometric relationship between features, etc.) N.B. generally raw primal sketch is combined into primal sketch for Marr's;

2.5D sketch: derived representations from primal sketch, such as local surface orientations from reflectance, depth from stereopsis, shape from motion/shading and depth discontinuities — simulate human capability for depth understanding.

Both the primal and 2.5D sketches perform the separation of four factors in visual systems: geometry (shape and placement), reflectance of visible surfaces, illumination, and view points.

3D model: hierarchical organisation of surface patches of 3D shape primitives, relational structure (spatial or contextual relationships). This allows the association of a new description with the appropriate one in the collection.

Each level should have algorithms operating on input data, an appropriate representation of data or model and a defined goal or output. Inter-level control of information flow and initial process must be exerted.

Shape-from-shading

Image formation process: the process of combining **intrinsic features** (depth, texture/albedo, surface reflectance, etc.) and **extrinsic parameters** (position/colour/direction of the light, viewpoint and viewing direction) into an image.

Computer graphics and vision are the two opposite ways of applying the image formation process: graphics select albedo map and reluctance function to improve realness of the synthesis of a scene, while vision analysis image content to recover shape and texture data based on an assumed reflectance model.

Shape-from-shading, a specialised version of **photometric stereo**, attempts to retrieve surface information from the shading of an object. It is a photoclinometry or inverse-rendering process. It requires a single image to estimate the 3D shape of the object, which includes surface orientation and material properties. The human vision system performs shape-from-shading naturally, however it suffers from similar **convex-concave ambiguity** issues as computer vision.

Based on the **Lambertian reflectance model** (assuming the surface is a perfect defuse reflector which scatters the incident light in all directions), shape-from-shading recovers surface normals by seeking agreement between the Lambertian estimation and the smoothness. By integrating the field of surface normals, surface height functions can be reconstructed, with two exceptions:

specular effects when the surface normal bisects the light source and viewer directions; and **roughness effects** and **microfacet** (small, smooth bits) **distribution** near the occluding limb.
The surface height function is $z = f(x, y)$, and the unit surface normal **n** of a point on the surface can be calculated as shown (the formula for source **s** is essentially the same, but with S_x and S_y instead of their partial derivatives on z):

$$n = \frac{1}{\sqrt{1+(\frac{\partial z}{\partial x})^2+(\frac{\partial z}{\partial y})^2}} \begin{pmatrix} \partial z/\partial x \\ \partial z/\partial y \\ 1 \end{pmatrix}$$

Under a Lambertian reflectance model, the amount of light reaching an observer is independent of the view direction and depends only on the albedo of the surface, i.e. $I_{ref} = k\rho(\mathbf{n}\cdot\mathbf{s})$, where ρ is the diffuse coefficient and k is the strength of incident light. (N.B. k is used for a different notation in comparison with VIGR's $I_{ref} = L_d k_d(\mathbf{n}\cdot\mathbf{s})$!)
The **image brightness** is normalised as $E = \mathbf{n}\cdot\mathbf{s}$. E is often a measurement from which we want to recover the surface normal **n**, hence deduce its shape. However, because there are two degrees of freedom: **n** and **s**, it is impossible to recover both from a single E value. This requires:
Shape-from-shading as an optimisation problem: establish two constraints to this optimisation problem — small variation of surface normals in direction (smooth surface); surface normals perpendicular to the edge of the object and in image plane when at occluding boundaries (for **occlusion**, see VIGR notes). There are two main approaches to achieve this optimisation: the **Horn and Brooks Variational Approach**, and the **Worthington and Hancock Algorithm**.

Horn and Brooks Variational Approach
Horn and Brooks establish a cost function as shown, with the first term measuring how well observed brightness matches the Lambertian prediction (*data closeness*); the second term measuring the *smoothness* of the recovered surface with directional derivatives of the field of surface normals. The approach ends when the field of surface normals minimise the cost function. This cost function can be adapted into an iterative process for minimisation. For numerical stability, in practice the closer $\epsilon^2/2\lambda$ is to zero, the more dominant smoothing effect is on the shape-from-shading process, and the poorer data closeness is of the resulting data. This is the key trade-off in shape-from-shading.

$$C = \iint \left(E - \mathrm{n}\cdot\mathrm{s}\right)^2 + \lambda\left(\left|\frac{\partial \mathbf{n}}{\partial x}\right|^2 + \left|\frac{\partial \mathbf{n}}{\partial y}\right|^2\right) dxdy$$

Iterative process of Horn and Brookes: surface normals are initialised to be perpendicular to the occluding boundary and on the image plane. Then the normal direction is iteratively updated inwards from the occluding boundary.

$$\mathrm{n}_{ij}^{k+1} = \hat{n}_{ij}^{k} + \frac{\varepsilon^2}{2\lambda}\left(E_{ij} - \mathrm{n}_{ij}^{k}\cdot\mathrm{s}\right)\mathrm{s}$$

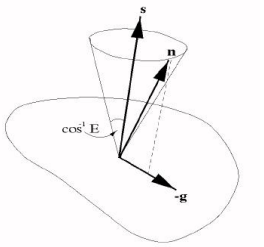
Local Neighbourhood Mean Normal — **Brightness Error**

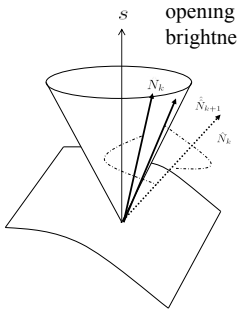
Averaging **+** ***Step parallel to light-source direction***

Disadvantages of Horn and Brooks: a stable and uniform convergence requires small $\epsilon^2/2\lambda$ values, which causes the smoothness term to dominate the data closeness term (hence normal hardly moving in direction of light source), causing the solution to be in poor agreement with the Lambertain reflectance model, thus eroding the surface detail. The method becomes numerically unstable if smoothing term is artificially reduced. Horn and Brooks also assume matte surface with no albedo variations on surface or surface highlights, and a single light source; which may not be the case in practice.
The improved **Worthington and Hancock Algorithm** seeks to address these issues.

Worthington and Hancock Algorithm
To address the data closeness issue in **Horn and Brooks**, a geometric method is developed so that the surface normals reside on a cone whose axis is in light source direction, and whose opening angle is equal to the inverse cosine (in this case, arccos) of the normalised image brightness $E = \mathbf{n}\cdot\mathbf{s}$. The constraint thus becomes a geometric one: the normal **n** must fall on a cone of ambiguity $\arccos(E)$ centred on the direction of **s**. This guarantees data closeness by constraining the normal **n** within a certain angle to **s**.

Iterative process of Worthington and Hancock: Initialise normals on cone, and align image plane in the negative direction of the Canny edge gradient. Then apply smoothness constraints to take normals away from the cone, possibly by local averaging; and project the smoothed normals back onto their closest positions on the cone, and repeat. This process cause surface normals to rotate about the cone centred on the light source with increasing iteration number.





Advantage of Worthington and Hancock: With this method, brightness E dictates the angle to the light source direction, while smoothness dictates the position on the cone, separating the two factors therefore circumventing the otherwise necessary trade-off between smoothness and data closeness.
Under **topographical labelling** based on shape index, Worthington and Hancock renders a more detailed and accurate probability graph than Horn and Brooks.

Surface Integration
Surface integration is the process of reconstructing surface height from field of surface normals. This requires the field of surface normals to be integrable/ developable (height dependent on path otherwise). The occluding contour of objects must provide locations of the same relative height.

$$\frac{\partial n_x}{\partial y} = \frac{\partial n_y}{\partial x}$$

Path-based method: pick a point on the boundary, search for the next pixel with the smallest change in surface normal direction, and increment height by a fixed amount.
Better methods: advance front from boundary (**Bruckstein**) and using inverse Fourier transform of surface normal field (**Frankot and Chellappa**).

Principle Component Analysis (PCA) for Shape Variation Modelling
Most natural objects exhibit shape variability, which may be a result of differences between object classes, or variations within an object class. A statistical model can be constructed to model this type of variation.
PCA: If we model each shape as a point in a high dimensional space, the distribution of these shapes can be captured by the distribution of points in the space. With the use of **principle component analysis** (**PCA**), we are able to reduce the dimensionality of data, by choosing directions along the maximum spread in dataset projected at a higher dimension. This is done by estimating the **covariance matrix** of points, and calculating to find the direction along which the variance of the data points is maximised.
The covariance between two individual variables is expressed as shown, while a covariance matrix ∑ containing covariance between each pair of elements from two sets of variables is shown below it. In computing the variance along a line, move the centre of the datapoint to the origin by zero-meaning the datapoints, then the position of a point along direction **u** is **x**·**u**. The variance of the points along direction u is given by $\mathbf{u}^T\sum\mathbf{u}$, where ∑ is the covariance matrix.

$$\mathrm{cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

$$\Sigma = {}^{1}\!/_{n}\mathbf{X}\mathbf{X}^{\mathbf{T}} = {}^{1}\!/_{n}\begin{pmatrix}\mathbf{x}_1 & \dots & \mathbf{x}_n\end{pmatrix}\begin{pmatrix}\mathbf{x}_1^T \\ \\ \mathbf{x}_n^T\end{pmatrix}$$

Thus, to maximise variance along a direction, $\mathbf{u}^T\sum\mathbf{u}$ needs to be maximised. Based on linear algebra, the principal eigenvector is in the direction of maximum variance ($\lambda_0 = \mathbf{u}$). Each subsequent eigenvector is orthogonal to the previous, and in the direction of maximum variance. This way, a new set of axes defined by eigenvectors can be found.
Because the covariance matrix ∑ is symmetric and positive semidefinite (all eigenvalues ≥ 0), it is possible to eigendecompose ∑ as $\sum = U\Lambda U^T$, where U is the set of maximising directions, while Λ is a diagonal matrix of eigenvalues. To rotate the datapoints into the new coordinate system, simply multiply with U^T: $X' = U^TX$. The covariance matrix of the new datapoints $\sum' = U^T\sum U = D$, and is diagonal containing the previously-calculated eigenvalues $\lambda_0, \lambda_1, \dots$

Point Distribution Models

To characterise shapes, long-vectors and their covariance matrices can be used. Pick 2D shapes from the same class, and place landmark points at corresponding/identifying features on these shapes based on deformations. For each landmarked shape, concatenate coordinates of the landmarks into a long-vector, and store the long-vectors into a set, which requires no reordering, addition or omission during the operations. A covariance matrix ∑ can then be calculated as shown.

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \hat{X}\right)\left(X_i - \hat{X}\right)^T$$

It is then possible to eigendecompose each covariance matrix $\sum = U\Lambda U^T$, where U is the matrix of eigenvectors and Λ is a diagonal matrix of ordered corresponding eigenvalues. We can then deform mean-shape in directions of principal eigenvectors using parameter vector b:

$$\widetilde{X} = \hat{X} + \widetilde{U}b$$

Deformable Model

With an established representation of features, we can deform the features to create new shapes. In the directions of principle eigenvectors, multiply the truncated eigenvector matrix with a parameter vector b (with the same number elements), and add the result to the original image data to achieve the deformed model.

Each direction represented by an eigenvector can be the direction of deformation to create different results.

With the least-squares method, we can find a best fit model to fit a deformable model to new image data. Good initialisation of the model is important to achieve a desired solution.

Limitations of the deformable model: the number of landmarks in each training sample must remain the same, the correspondences between landmark points must be known, and if correspondences are wrong, then covariance matrix measures confusion rather than shape variability.

Shape-from-shading with Shape Variation Models

Shape variation models such as a point distribution model can be used to aid surface shape reconstruction. While traditional shape-from-shading suffers from convex-concave ambiguity as discussed earlier, a statistical model can be used to perform better cone-based reconstructions from surface normals.

Equidistant azimuthal projection (EAP): a method for projecting spherical 3D orientations to 2D directions on a plane, achieving a dimensional-reducing projection. The projection preserves distances to the mean normal on the sphere, placing surface normals points on the tangent plane. It is then possible to use a point distribution model to describe statistical properties of the projected surface normals.

Use shape-from-shading to get an initial estimation of surface normals, convert them into points with EAP. Obtain a best-fit model on the points, and project it back onto the sphere with inverse EAP. Then, project best-fit normals on the sphere onto nearest locations on the irradiance cone centred at the light source, and perform iterative constraint-based fitting until convergence. This usually produces a better shape-from-shading estimation against convex-concave ambiguities.

Structural Shape/Scene Description

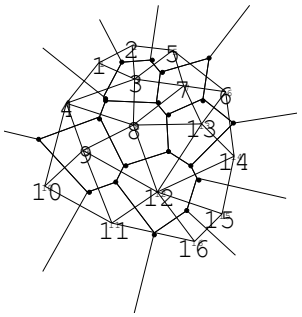
The high level vision is an **inference process**. It concerns the symbolic identity (e.g. class) of image entities such as regions. It is based upon available image data (e.g. geometric relationship between objects) and textual information. High level vision must adapt a suitable way of representing its knowledge of the image, which can be either **analogical** (coherent and continuous, attaching attributes to concepts) or **propositional** (dispersed and discrete, describing relationships between concepts). A good way to represent this kind of knowledge is with **attributed relational graphs**, with nodes as image regions with attributes, and arcs as constraint relations between regions.

Graph Matching Algorithms

In order to associate nodes in the attributed **relational graphs** with those in data models, **graph matching algorithms** (GMA) need to be applied.

Problems faced by GMA: over segmentation (too many data nodes compared to model nodes); under segmentation (too little data nodes); noise (data nodes not described by the model).
Methods of GMA: graph isomorphisms; subgraph isomorphisms; clique finding; elastic matching.
The general process of applying high level relational models is to first extract shape-primitives from raw image data (segmentations, surfaces, etc.), and abstract these primitives with a relational graph, and match abstraction with the graph to a model with a GMA.

Relational Graphs
Also described as **point and line proximity graphs**, there are a few common types of relational graphs:
Region adjacency graph: apply segmentation to image plane and segment it into connected regions and locate perimeters of regions. Represent regions with nodes, while an edge between two nodes represent a common length of perimeter between two regions.
Nearest Neighbour graph: represent a set of points as nodes, and compute distance between points (e.g. Euclidian distance or Manhattan distance). Create an edge between a node and its N-nearest points. The graph is directed, and may contain region centroids, centres of lines or corner features.
Voronoi tessellation: grow a region with uniform radial speed about each point, and halt growth when two growing regions collide. The collision front is the perpendicular bisector of the line connecting the two regions' points, and each point is contained within a polygonal region. The graph thus tessellates the image planes into these polygonal regions.
Delaunay graph: construct a region adjacency graph for the result of the above Voronoi tessellation, resulting in nodes with on average 5.5 neighbours. It is composed of triangular faces. Each point is a node, and all nodes belong to first-order cycles containing three nodes, as shown. (N.B. the numbered locations are *nodes* in first-order three-node cycles, while the solid dots are points).



Constrained Delaunay triangulation: a process of dividing the plane into triangular segments such that no triangulated point is inside the circumcircle drawn from any of the triangular segments, while preserving the directionality of the segments. This is done by finding the closest end-point distance, dividing the line into sub-segments with the closest distance as length, and placing growing Voronoi seeds/points at the centre of each segments. The image plane is thus segmented into regions with collision fronts originated from seeds.
Gabriel graph: constructed from a Delaunay graph, a circle is constructed on each edge/line connection two points, whose diameters are equal to the length of the edge. Original Delaunay edges whose circumcircles contain one or more nodes are deleted.
Relative neighbourhood graph: construct a lune on each edge of the Delaunay graph, the two circles defining the lune are entered on the two nodes defining the edges, with the edge as radius. Delete edges from the original Delaunay graph with one or more nodes in the lune.

Shock Graphs
Shocks (or **singularities**) are places where the inward evolving boundary particles collide in the boundary evolution process. This is also referred to as a Blum skeleton. The **shock points** are centres of circles contained in the interior and tangent to the border in two or more points.
Reaction-diffusion analysis: a process of smoothing the shape and inducing a scale space on the generated stocks by adding in **diffusion** at curvature dependent speed and then applying **reaction** at constant speed. The boundary is propagated inwards over time.
Skeleton extraction: a process of transforming the boundaries into a **skeleton** of an object in the image. Two possible ways to achieve this: **boundary evolution** (apply reaction and detect shocks, an direct approach but detection is unstable and computationally demanding); **thinning** (peeling off the out layer of pixels at each iteration, computationally efficient but not invariant under Euclidean transformations). Skeletons are stable under shape morphing.
The shape can be characterised by the **bending** along the skeleton. **Ligatures** may also exist on the skeleton, where noise perturbations provide possibilities for branches to develop on the skeleton. Also measured by derivatives.

$$\frac{dl}{ds} = \frac{dl_1}{ds} + \frac{dl_2}{ds}$$

Shocks or skeleton points can be labelled based on the change in radius of osculating circles, which are constructed along the skeleton between near boundary points.
Shock tree/graph is a tree where the last-formed (furtherest away from the boundary) is the root, and first-formed points are the leafs. Edges of this tree represent the connectivity of the skeleton branches.

Graph Matching
Graph matching is a process of matching shapes by matching their structural abstractions, with the use of a GMA. Correspondences are searched for between nodes of the graphs, and edges of graphs are used to check for and maintain consistency of the pattern of correspondences.
Relational graphs are generally represented with adjacency matrices with edge weight functions. The matching process match two graphs by constructing a fixed mapping relation between nodes in the two graphs, arranged in permutation matrices.
Permutation matrix: a matrix that re-orders the nodes of a graph. Each row and column of a permutation matrix contains a single one, with the rest being zero. Isomorphism is approximated between two graphs by minimising the difference in adjacency matrix under permutation.
Trace criterion: if permutation matches a pair of edges, score one; otherwise, score zero. This way, the number of consistently matched edges is counted by the criterion for matching purposes.
Problems using permutation matrices: finding permutation matrices is a difficult constrained optimisation problem. Search methods such as Hungarian method or Ullman's search with backtrack can be used, but a relaxed representation of the problem may be better. The relaxation allows the elements to be of any value, and the row-column constraint is also relaxed.
Umeyama algorithm: a graph matching algorithm applicable to graphs with the same number of nodes but different edge structures/adjacency matrices. It matches eigenvectors of the adjacency matrices with approximation.
Luo's algorithm: the approximation nature of this type of approach can also be considered as an optimisation problem, for which a correspondence indicator matrix and a cost function are used. Gradient ascent performs the optimisation, with the use of soft assign.