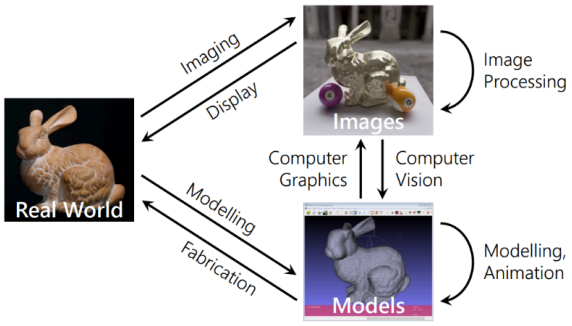
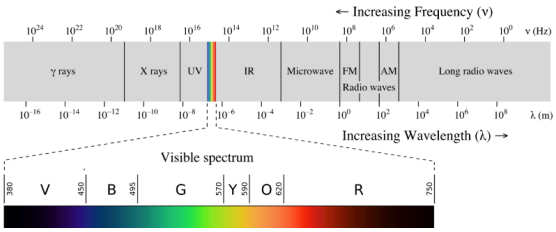
Introduction to Visual Computing



Key Terms

- **Imaging:** Capturing photos.
- **Display:** Making an image visible using emissive (displays) or reflective (paper) media.
- **Fabrication:** Turning models into physical objects.
- **Processing:** Performing operations on digital images to produce another image.
- **Graphics:** Generating an image from a model.
- **Vision:** Reconstructing a scene virtually from real-world image and video.

Images & Colours



Visible Light Spectrum

Visible colours are a **subset of the wavelengths** in the electromagnetic spectrum.

Spectral Power Distribution

An incident spectrum details the **wavelengths** that make up a colour. A reflectance distribution dictates which colours are **reflected**. So, the two determine the overall colour.

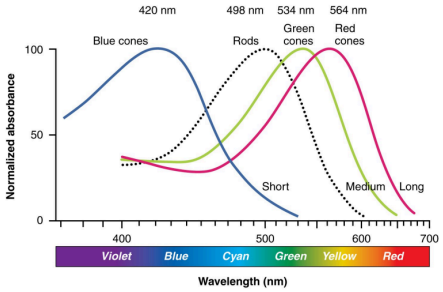
The radiance of a wavelength is $R(\lambda) = E(\lambda) * S(\lambda)$. *Memorise!*

- λ - the wavelength.
- E - the incident radiance.
- S - the reflectance.

Cones & Displaying Colour

Humans have **trichromatic vision**, meaning we can see three cone types and are most sensitive to short (blue), medium (green) and long (red) wavelengths.

So, our displays only need to show distributions for our cone responses!

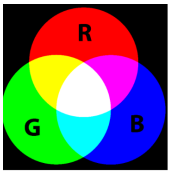


Cones are for **colour** vision and rods are for **night** vision!

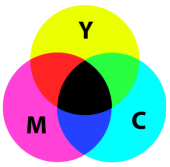
Additive & Subtractive Primary Colours

This inspired the creation of the **additive** RGB system and the **subtractive** CMY system.

- Suitable system depends on background colour.



Mixing lights
e.g. LCD monitors

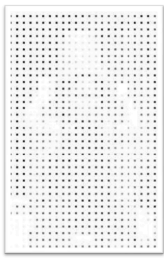


Filters and paints
e.g. printers

Other Colour Spaces

- **CMYK:** Adds black to save ink when printing grey.
- **HSL (Hue, Saturation, Lightness):** User-friendly so common in colour pickers,
- **CIEXYZ:** Specifies colours and the ranges.
- **Greyscale**

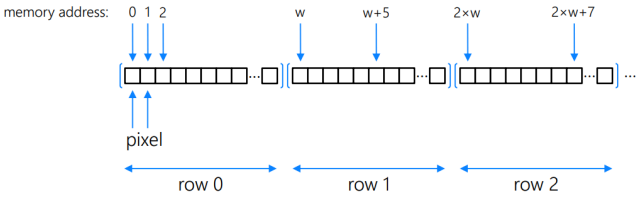

continuous function


discrete samples

Representing Images

Digital images are **2D representations** of a continuous function stored as **pixels** in an array.

- Pixels are single sampled colours.



w is the width of the image.

Storing Greyscale Images

Digital images are stored as **intensity values** in **scanline order** in memory.

Storing Colour Images

Stored similarly to greyscale images but with three RGB values for each pixel.

$$Index = 3 * ((row * wdth) + col)$$

Memorise!

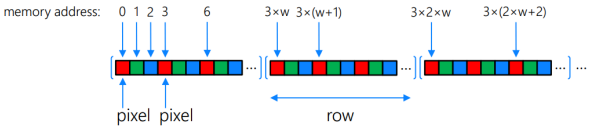


Image Compression

- **Lossless Compression:** PNGs/GIFs use LZW encoding to enumerate frequent strings.
- **Lossy Compression:** JPGs use DFT to encode high frequencies with fewer bits.

Camera Imaging

Relates to **how a sensor captures an image**. There are two main types:

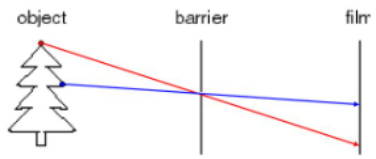
CCD: Pixels read out one-by-one.

CMOS: Amplifiers increase readout speed.

Sensors record photons using **arrays** made of **cells**. Each cell records the photons that reach it.

Aperture

Controls how much light passes through to the camera's sensor or film.



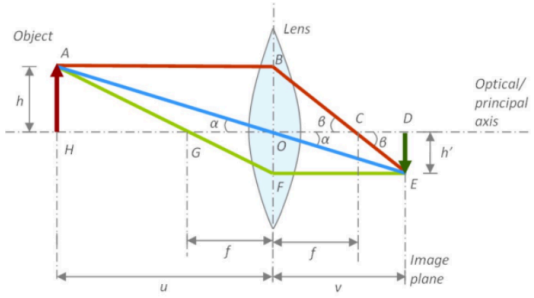
This is called the **pinhole camera model**. It enables all light to be captured from a **focal point**.

- **Image Plane:** The sensor where the image is formed
- **Optical Center/Focal Point/Center of Projection:** The point light rays cross.

Changing the aperture changes the **depth of field** and thus blurriness. Achieving the perfect aperture is difficult. Too wide lets through too many light rays and creates a blurry image, but too small leads to **diffraction** (bending of rays).

Thin Lens Camera Model

Lenses **focus rays** onto a single focal point to mitigate diffraction.



$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$ *Memorise!*

Denotes a relationship between the variables:

- f - The focal distance (specific to the lens).
- u - The object distance from the lens.
- v - The image distance from the lens.

Display Types

Active	***Passive***
• Most common. • Expensive. • More power-hungry.	• Traditional. • More common in mass-produced products.

e.g. LCDs, OLED, plasma.	*e.g. laser & ink-jet printers.*

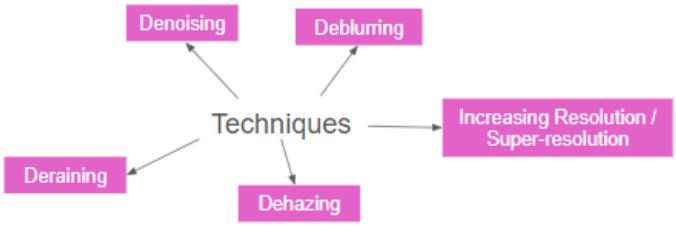
Liquid Crystal Displays (LCDs)

Utilise liquid crystals to twist the polarisation of light, with the voltage controlling how much light can pass through.

RGB to Greyscale

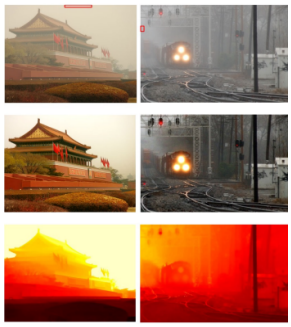
$l(r, g, b) = 0.2126r + 0.7152g + 0.0722b$

Image Enhancement



Dehazing

Improves image quality by removing the haze in an image using the **dark channel prior**. In haze-free outdoor images, one colour channel in a patch has very low intensity. The dark channel is the channel with the lowest intensity in the patch, and is recalculated for each pixel.



$$I_{\text{dark}}(x) = \min_{y \in \Omega(x)} \left(\min_{c \in \{R,G,B\}} I^c[y] \right)$$

Memorise!

- $I^c(y)$ - intensity of pixel y in channel c.
- Ω - the local patch centered around pixel x.

Transmission Map

$t(x) = 1 - \omega * I_{dark}(x)$ *Memorise!*

- ω is a constant which keeps some haze to avoid artefacts.

General Image Formula

$I(x) = J(x)t(x) + A(1 - t(x))$ *Memorise!*

- $J(x)$ - the scene radiance (the clear image we want to recover).

- $t(x)$ - the transmission map (the level of haze).
- A - the atmospheric light (the highest intensity pixel of all the dark channel pixels).
- x - a pixel in the image.

Recovery Equation

$$J(x) = \frac{I(x) - A}{max(t(x), t_0)} + A$$ *Memorise!*

- t_0 - a small constant to prevent division by zero.

1. Compute the dark channels for each pixel in the hazy image.
2. Find the atmospheric light from these.
3. Estimate the transmission map.
4. Calculate the scene radiance to get the dehazed image.

Noise: Unwanted **pixel variations** that **degrade image quality** caused by sensor imperfections, low light conditions, etc.

Thermal Noise *a.k.a Additive Gaussian Noise*
Caused by random electron fluctuations when the photo sensor is amplified.

- Related to ISO.
- Simulate using Gaussian Noise.

Shot Noise *a.k.a Poisson Noise*
Caused by photons randomly hitting the sensor.

- Related to the photodiode.

Replacement Noise *a.k.a Salt & Pepper Noise*

- Dead pixels = too dark.
- Hot pixels = too bright.

Speckle Noise
Two types: objective & subjective.

- Common in medical imaging and radar systems.

Signal-Noise Ratio (SNR)

Measures noise severity by comparing the desired signal level to the background noise.

- Expressed in decibels.
- Works for signal types other than images.

$$SNR = \frac{P_{signal}}{P_{noise}}$$ where P is the power (pixel intensity).

Convolutions & Image Filtering

Modifies an image's appearance by changing the colours of its pixels. This is done **per-pixel**.

Quantifying Colours

A pixel's colour is represented by a set of numbers indicating the **intensity of each colour channel**. A byte for each channel can represent a wide range of colours, but more is even better! *e.g. three channels makes* $2^8 * 2^8 * 2^8 = 16,777,216$

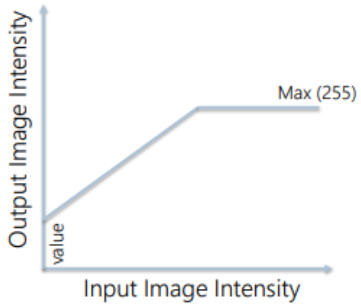
Point-based Filters

Calculates modified pixels using functions.

Increasing Brightness

$I(x,y)\ +=\ value\ (s.t.\ I(x,y)\ <=\ 255)$ *Memorise!*

Increase each pixel's intensity by a fixed value, with a cap.



Increasing Brightness Filter

Inverting Colours

$O(x,y)\ =\ 255\ -\ I(x,y)$ *Memorise!*

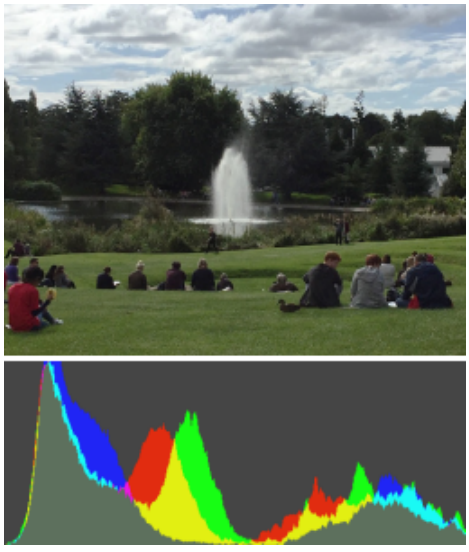
Increasing Contrast

$$O(x,y) = \left[\frac{I(x,y) - 51}{0.6}\right]_0^{255}$$

51 and 0.6 are the parameters. *Memorise!*

Other Filters

Darkening, emphasise shadows & lights, compress to darks.

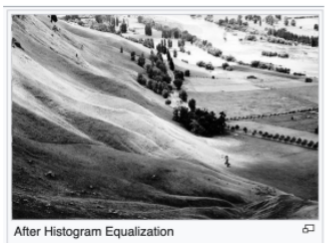


Histograms

Histograms summarise the distribution of colour intensities in an image.

Using Histograms to Alter Colours

Histogram equalisation spreads the distribution equally.



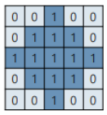
Off Point-based/Neighbourhood Filters

Enhance features or suppress noise by considering the **variations around a pixel**.

Median Filtering

Smooths images by replacing each pixel with the **median value** of all pixels in its neighbourhood.

3px median filter | 10px median filter



disc/diamond

Morphological Filtering

Filters images using **set theory** (*i.e. max and min rather than using weights*) and uses a **structuring element** to select which pixels to apply this to (left).

Erosion

Reduces the width of bright objects.

$$(I \ominus B)(x,y) = \min_{(i,j)\in B} I(x-i, y-j)$$

where B is the structuring element.

Memorise!

Dilation

Increases the width of bright objects.

$$(I \oplus B)(x,y) = \max_{(i,j)\in B} I(x-i, y-j)$$

where B is the structuring element.

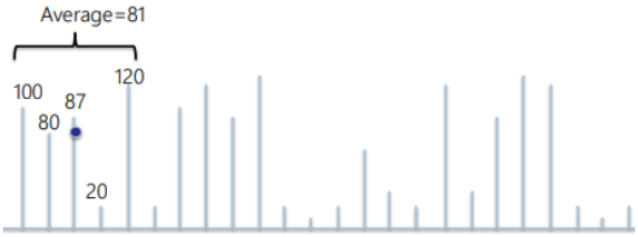
Memorise!

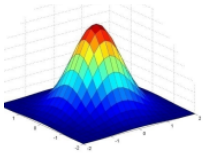
Opening: Dilating after erosion to avoid boundary shrinkage.

Closing: Eroding after dilating to avoid boundary expansion.

Box Filter

Smooths images by calculating the average value of pixels in an area, with equal weights, and assigns that to the main pixel.





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

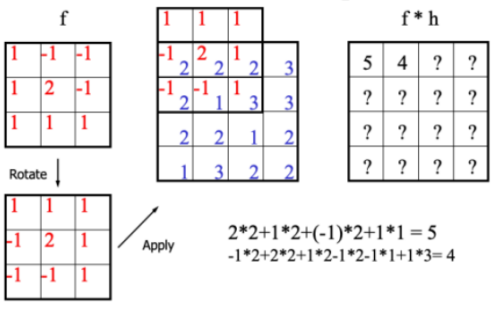
0.003	0.013	0.022	0.013	0.003
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

Gaussian / Low Pass Filter

Also smooths images but puts **more weight on the center** pixel. Changing the **standard deviation** alters the blurriness.

These two are just convolution!



Convolution

Asymmetric kernels must be **flipped**!

Slides a **kernel** over an image to compute a new value for each pixel. The center element of the kernel is placed over it and replaced by a **weighted sum** of itself and the **neighbouring** pixels.

Uneven kernels cause **calculation difficulties** and are **computationally expensive**.

Stride

How much the kernel moves at each step.

Larger Stride = Smaller Output Dimensions

$$Output\ Size = (\frac{Input\ Size\ -\ Kernel\ Size}{Stride}) + 1$$

Don't memorise!

Padding

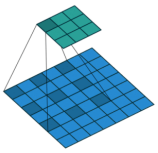
Adding pixels around the input's edges to control the output size or preserve dimensions.

$$Output\ Size = (\frac{Input\ Size\ -\ Kernel\ Size + 2\ *\ Padding}{Stride}) + 1$$

Don't memorise!

Dilation

Places gaps between kernel elements, allowing them to cover a larger field without increasing the number of parameters.



Convolutional Filters/Kernels

- Motion Blur.

$$\frac{1}{5} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Sharpening.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

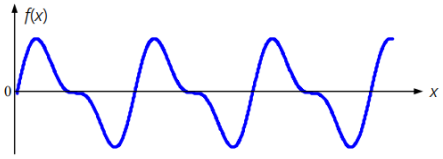
The Fourier Transform

Analogue Signals

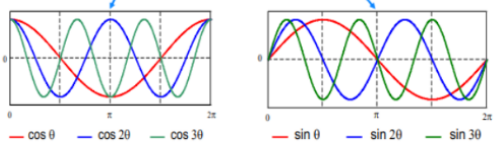
A smooth, continuous signal which varies over time and carries repeated information.

Periodic Functions

A function is periodic if it is defined for all real x and there is some positive number N s.t. $f(x + N) = f(x)$.



$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$$



Don't memorise!

The Fourier Series

A formula for any **periodic function** to be written as a **weighted sum of sine and cosine functions of different frequencies**.

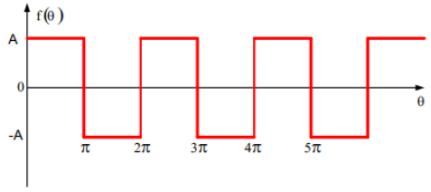
a_0, a_n and b_n are found using formulae which will be given in the exam!

Example

Find the fourier series of the periodic function seen left.

$$f(x) = \begin{cases} -A & \text{for } -\pi \le x < 0 \\ +A & \text{for } 0 \le x \le \pi \end{cases}$$

$$f(x + 2\pi) = f(x)$$



$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)\,dx$$
$$= \frac{1}{\pi}\left(\int_{-\pi}^{0}(-A)\cos(nx)\,dx + \int_{0}^{\pi} A\cos(nx)\,dx\right)$$

case n=0: $$a_0 = \frac{1}{\pi}\left(\int_{-\pi}^{0}(-A)\,dx + \int_{0}^{\pi} A\,dx\right) = 0$$

otherwise: $$a_n = \frac{1}{\pi}\left[(-A)\frac{\sin(nx)}{n}\right]_{-\pi}^{0} + \frac{1}{\pi}\left[A\frac{\sin(nx)}{n}\right]_{0}^{\pi} = 0$$

Don't memorise, just understand!

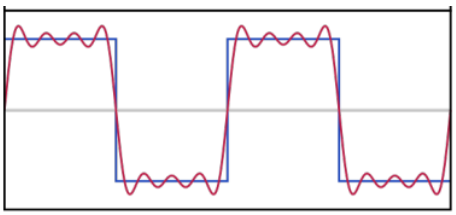
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,dx$$
$$= \frac{1}{\pi}\left(\int_{-\pi}^{0}(-A)\sin(nx)\,dx + \int_{0}^{\pi} A\sin(nx)\,dx\right)$$
$$= \frac{1}{\pi}\left[(-A)\frac{-cos(nx)}{n}\right]_{-\pi}^{0} + \frac{1}{\pi}\left[A\frac{-cos(nx)}{n}\right]_{0}^{\pi}$$
$$= \frac{A}{n\pi}(\cos(0) - \cos(-n\pi) - \cos(n\pi) + \cos(0))$$
$$= \frac{A}{n\pi}(2\cos(0) - 2\cos(n\pi))$$

$$b_n = \begin{cases} \frac{4A}{n\pi} & \text{when } n \text{ is odd} \\ 0 & \text{when } n \text{ is even} \end{cases}$$

Solution:

Fourier series:

$$\frac{4A}{\pi}\left(\sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x) + \frac{1}{7}\sin(7x) + \ldots\right)$$



The Fourier Transform

Extends the Fourier series to **split any function into its frequency components**. Translates an image in the **spatial domain**, a grid of pixel intensities, into the **frequency domain** where it is represented as a combination of **sinusoidal frequencies**. For 2D images, it converts data into a **2D frequency spectrum**, where each point represents the contribution of a specific **sinusoid** to the **overall signal/image**.

The **Fourier transform** of a function $f : \mathbb{R} \to \mathbb{R}$ is given by:

$$F(\omega) = \mathcal{F}[f](\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-i\omega x} \, dx$$

The **inverse Fourier transform** restores f from F:

$$f(x) = \mathcal{F}^{-1}[F](x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\omega) e^{i\omega x} \, d\omega$$

The Discrete Fourier Transform

A specific version of the fourier transform for **finite sampled signals**, like **audio** and **images**. Should be applied to each colour channel. Used for audio and image compression (JPEGs).

Discrete Fourier transform (DFT):

$$F[k] = \sum_{n=0}^{N-1} f[n] e^{-2\pi i n k / N} \qquad (k = 0, 1, \ldots, N-1)$$

Inverse discrete Fourier transform:

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{2\pi i n k / N} \qquad (n = 0, 1, \ldots, N-1)$$

Properties

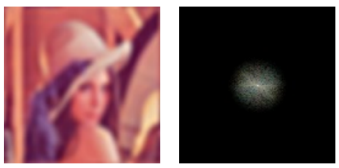
- **Linearity:** Two functions can be mixed. $\mathcal{F}[af + bg](\omega) = a\mathcal{F}[f](\omega) + b\mathcal{F}[g](\omega)$
- **Shifting:** Shifting a signal in time or space doesn't change the magnitude of frequencies but alters the phase. for a constant a, if $g(x) = f(x - a)$, then: $\mathcal{F}[g](\omega) = e^{-ia\omega} \mathcal{F}[f](\omega)$
- **Modulation:** Two functions can interact to form a new one. for a constant a, if $g(x) = e^{iax} f(x)$, then: $\mathcal{F}[g](\omega) = \mathcal{F}[f](\omega - a)$
- **Scaling:** The size and duration of a signal can be changed. for a constant a, if $g(x) = f(ax)$, then: $\mathcal{F}[g](\omega) = \frac{1}{|a|} \mathcal{F}[f]\left(\frac{\omega}{a}\right)$
- **Periodicity (DFT Only):** Results repeat, making the frequency spectrum circular.
- **Convolution (DFT Only):** Convolution in the spatial domain is **multiplication** in the frequency domain. So, converting, multiplying and inversing is **more efficient**!

The Frequency Spectrum

Maps the different frequencies that make up the signal. So, removing some manipulates the image!

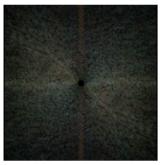
Low-Pass Filter

Blurs an image by removing signals with a **frequency above** a threshold.



High-Pass Filter

Highlights edges by removing signals with **frequencies below** a threshold.



Noise Removal

Remove patterned noise by discarding regions associated with it.

The Fast Fourier Transform (FFT)

Uses **divide and conquer** to reduce the complexity from $O(N^2)$ to $O(Nlog_2(N))$, but may require **padding inputs**.

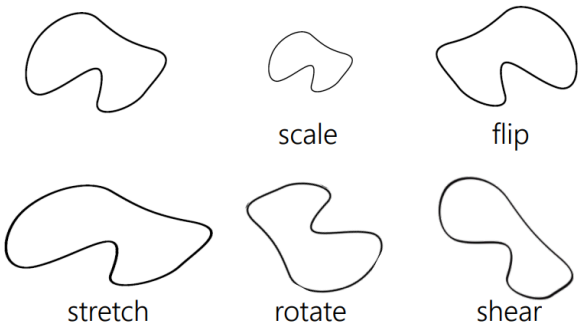
2D Transformations

There are three types of transformations:

Rigid
All points on an object move with the same speed and rotation.

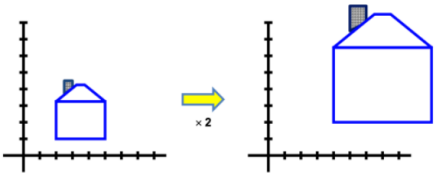
Articulated
Several components which each are rigid but operate at different speeds and rotations.

Non-Rigid
All points on an object can move freely. *E.g. a rubber band*

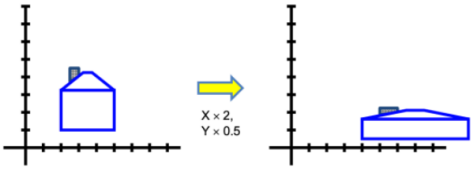


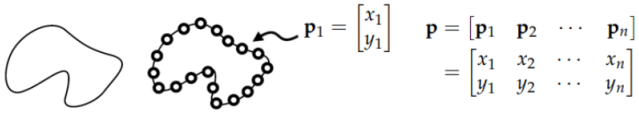
Scaling

Uniform: The scalar is the same for all components.



Non-Uniform (Stretching): The scalar isn't the same for all components.





Transformation Matrices

A 2D shape can be represented by a matrix containing the set of points along the shape.

To transform the shape, we multiply this matrix by a transformation matrix!

Uniform Scaling

$$x' = \lambda x \quad y' = \lambda y \implies \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Where λ is the scale factor.

Horizontal Flip

$$x' = -x \quad y' = y \implies \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous Coordinates

Translation moves a shape in a horizontal or vertical direction, homogeneous coordinates **enable transformation and translation in one linear matrix**, called an **affine transformation**!

$$\begin{bmatrix} p'_1 \\ p'_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \longrightarrow \begin{bmatrix} p'_1 \\ p'_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

You can stack multiple of them ($p' = T_3 T_2 T_1 p$) and represent a wide range of transformations because they provide **six degrees of freedom**: translation (2), rotation, scale, stretch & shear.

Affine transformations **preserve lines** and **parallelism**.

Example Affine Transformations *Memorise!*

Stretch	Rotation	Translation
$\begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

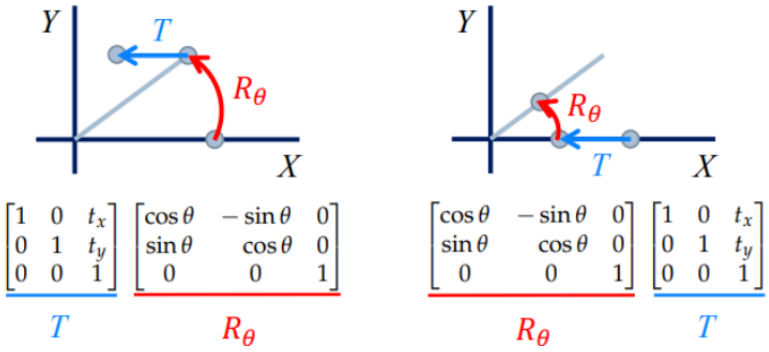
Compound Transformations

Possible because matrix multiplication is **associative!**

Combinations of multiple transformations one after another, including but not limited to affine transformations. To combine two matrices, multiply them from **last to first** applied: $M = S * T$

Order Matters!

In general, for two matrices $AB \neq BA$.



$$\underbrace{\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{T} \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_\theta} \qquad \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_\theta} \underbrace{\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{T}$$

Rotation About an Arbitrary Point

1. Translate the coordinates to origin.
2. Rotate them around it.
3. Translate them back so it appears like they were rotated around that point.

Row VS Column Representation

Coordinates can be represented as row or column vectors. If using row vectors, **transpose** everything and **multiply in reverse** order!

$$\mathbf{p}' = \mathbf{T}_1\,\mathbf{T}_2\,\mathbf{T}_3\,\mathbf{p} \qquad \mathbf{p}'^{\top} = \mathbf{p}^{\top}\,\mathbf{T}_3^{\top}\,\mathbf{T}_2^{\top}\,\mathbf{T}_1^{\top}$$

Column Vectors | *Row Vectors* *Memorise!*

Projective Transformations *Memorise!*

Fills the bottom row of an affine transformation matrix to enable **changes in perspective, perspective transformations**! Can be still used for affine transformations.

- **Preserve straight lines** but not necessarily parallelism or distances.
- **8 degrees of freedom!**

Scale factor s to ensure coordinates stay constant!

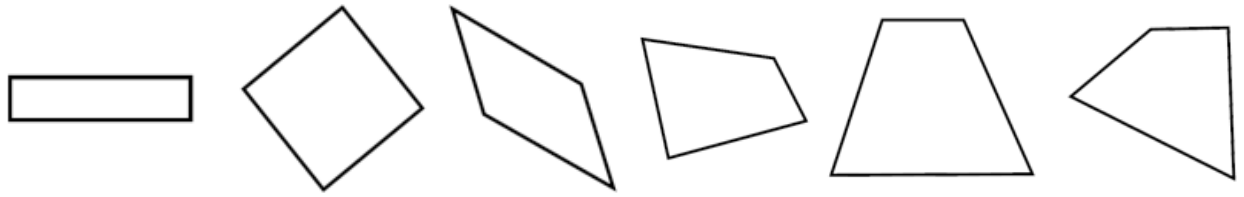
The transformation equation becomes:

$$s\begin{bmatrix} p_1' \\ p_2' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

which is just 3 linear equations:

$$sp_1' = m_{11}p_1 + m_{12}p_2 + m_{13}$$
$$sp_2' = m_{21}p_1 + m_{22}p_2 + m_{23}$$
$$s = m_{31}p_1 + m_{32}p_2 + m_{33}$$

Example Projective Transformations



Homographies
An instance of a projective transformation on a plane.

Example: Image Alignment

To align two images, we find **common points between** them, and identify a **projective transformation** to fit one with the other. The number of points needed is dictated by the type of transformation used:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Affine transform
6 degrees of freedom
3 points to specify

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

Homography
8 degrees of freedom
4 points to specify

Using an Affine transform, rearrange the equation so the unknowns are in a vector and combine the equations for the three point pairs!

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}$$

This gives an equation $Ma = x$, so we just solve for a using Gaussian elimination.

Input image | Forward mapping $\mathbf{p}' = \mathbf{T}\mathbf{p}$ | Backward mapping $\mathbf{p} = \mathbf{T}^{-1}\mathbf{p}'$

Image Warping

Applying a transformation to an image using forward mapping leads to computers rounding the coordinates, **resulting in gaps**! So, you should use **backward mapping** which finds the original pixels for each pixel in the resulting image!

However, that backward mapping gives sub-pixel coordinates! *e.g. 132.45, 80.76* **Interpolation** selects a neighbouring pixel to copy!

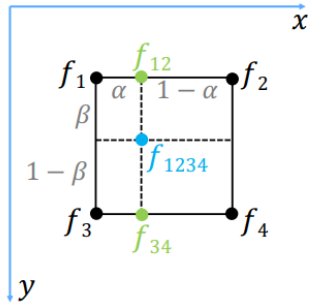
Nearest-neighbour Interpolation

Rounds to the nearest integer.

Bilinear Interpolation

Calculates a value based on the nearest four pixels, based on their closeness.

Memorise!



$$f_{12} = (1-\alpha)f_1 + \alpha f_2$$

$$f_{34} = (1-\alpha)f_3 + \alpha f_4$$

$$\begin{aligned} f_{1234} &= (1-\beta)f_{12} + \beta f_{34} \\ &= (1-\alpha)(1-\beta)f_1 + \\ &\quad \alpha(1-\beta)f_2 + \\ &\quad (1-\alpha)\beta f_3 + \alpha\beta f_4 \end{aligned}$$

Scale-Invariant Feature Transform (SIFT)

Detects and describes keypoints for image stitching and object recognition.

Keypoints/Interest Points/Features/Blobs

Image locations with **distinctive texture/contrast/scale**, useful for identifying matches between images. Difficult to detect because of **variations in size, orientation, etc**, which SIFT compensates for. Ideally, they should have:

Unique **representation** for matching.

Rich **image content** within a window.

Well-defined **position**.

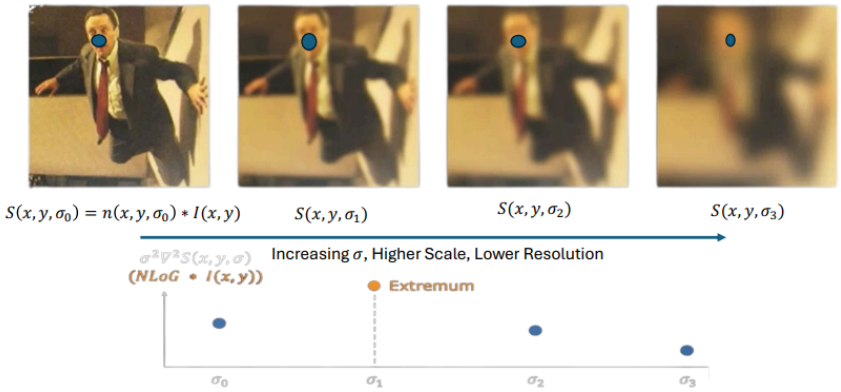
Rotation and scaling **invariant**.

Insensitive to lighting changes.

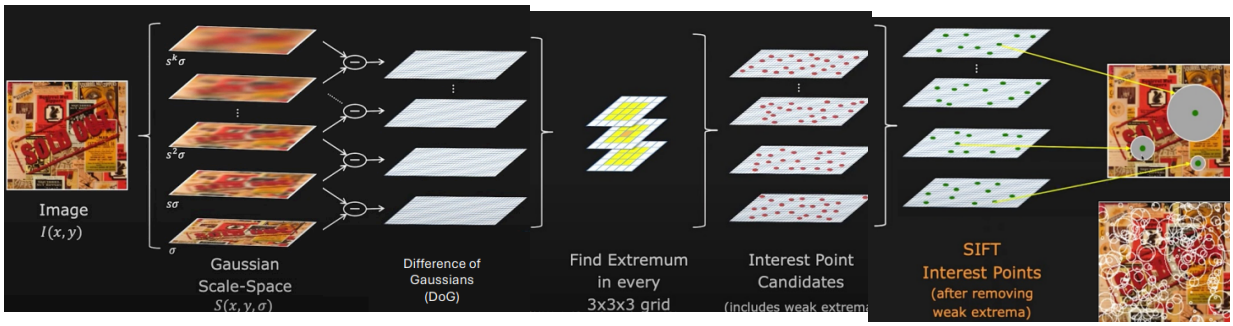
Extracting Blobs

Blob-like features have **fixed positions**, **definite size** and are detected by applying different levels of Gaussian filter to the image to create the **scale space**, then applying the **Normalised Laplacian of Gaussian (NLOG)** to each and looking for **local extrema**.

- Different Gaussian filters generated from different sigma values.
- Blurring smooths out patterns, making larger patterns stand out.
- **Difference of Gaussian** (DoG) is an approximate faster version.



Find the extrema in every 3x3x3 grid and **filter out the weakest** to get the SIFT interest points!



Scale Invariance

$$\frac{\sigma_1}{\sigma_2}$$

Normalises blob size across different scales, using their ratio, ensuring keypoints are detected consistently regardless of image resizing.

Rotation & Brightness Invariance

Compute the **principal orientation** of features by identifying and plotting a normalised histogram of their **image gradient directions**. Use these to undo rotation.

Comparing SIFT Descriptors

Then, **compare the two arrays of image gradient directions** using a distance metric to match features.

L2 Distance

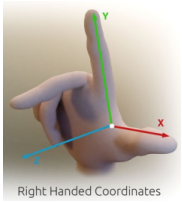
Smaller the better.

Intersection

Larger the better.

3D Transformations

We use right-handed coordinates, so you can use the right-hand rotation rule: thumb in direction or rotation axes and fingers indicate positive rotation direction.



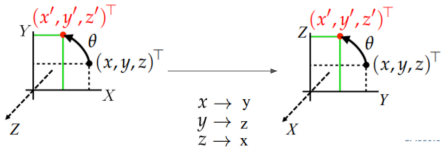
Rotations

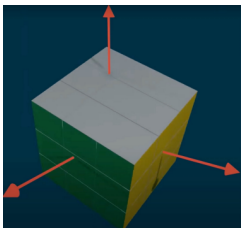
This formula rotates around the z-axis. This **isn't homogenous** because it doesn't incorporate translation!

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

To calculate other rotation matrices from this:

1. Identify new axes values depending on axis of rotation and thumb rule.
2. Use this to rearrange the coordinate vectors.
3. Rearrange the input columns into the correct order.
4. Rearrange the output rows into the correct order.





Euler Angles *Memorise!*

Three values representing rotations about different axes describe **any orientation** of a rigid body. An **ordered combinatio**n generates a rotation about an **arbitrary axis** passing through the origin.

- 27 possible combinations for rotation.

$$\mathbf{p}' = \mathbf{R}_z(\psi)\mathbf{R}_y(\phi)\mathbf{R}_x(\theta)\mathbf{p}$$

Compound Rotations

Calculate from right to left using the rotation matrices.

Order matters!

$$R_{\text{combined}} = R_x(90^\circ) \cdot R_y(90^\circ) \cdot R_x(-90^\circ)$$

1. **Compute** $R_y(90^\circ) \cdot R_x(-90^\circ)$:

$$R_y(90^\circ) \cdot R_x(-90^\circ) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}$$

2. **Compute** $R_x(90^\circ) \cdot (R_y(90^\circ) \cdot R_x(-90^\circ))$:

$$R_x(90^\circ) \cdot \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

The result is an $R_z(90)$ rotation! Work this out by applying the compound rotation to $[1, 0, 0]$ to get $[0, \ -1, \ 0]$. Rotation on any axis can be made by **rotation around any two other axes**.

Gimbal Lock

Gimbal: A rotation frame in one axis.

Each Euler angle has a gimbal, which are mounted in each other following the order of rotation.

Gimbal lock occurs when one axis of rotation lines up with another, preventing movement in one. For example, if R_x lines up with R_z, then no rotation about R_x.

$$\begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \Rightarrow \begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 + t_1 \\ p_2 + t_2 \\ p_3 + t_3 \\ 1 \end{bmatrix}$$

3D Homogeneous Coordinates

Adding a 1 to the coordinates makes translation linear.

3D Affine Transformations

Add an extra row of values. Recall that off-diagonal elements control **rotation** and **shearing**.

Rotation, scaling, shearing — Translation

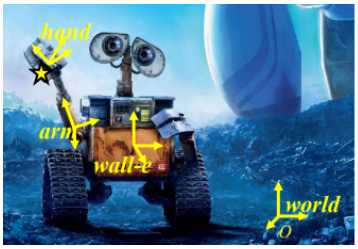
$$\begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{p}}_{\text{world}} = \mathbf{R}_1 \tilde{\mathbf{p}}_{\text{wall-e}} + \mathbf{t}_1$$
$$\mathbf{p}_{\text{world}} = \mathbf{T}_1 \mathbf{p}_{\text{wall-e}}$$

3D Local/Global Coordinates

Represent a local coordinate system (wall-e) in a global one (world) using homogeneous coordinates.

With even more compound coordinate systems, you can combine the transformations for each. If the hand moves, only the T_3 matrix needs changing.



$$\mathbf{p}_{\text{arm}} = \mathbf{T}_3 \mathbf{p}_{\text{hand}}$$
$$\mathbf{p}_{\text{wall-e}} = \mathbf{T}_2 \mathbf{p}_{\text{arm}}$$
$$\mathbf{p}_{\text{world}} = \mathbf{T}_1 \mathbf{p}_{\text{wall-e}}$$
$$\mathbf{p}_{\text{world}} = \mathbf{T}_1 \mathbf{T}_2 \mathbf{T}_3 \mathbf{p}_{\text{hand}}$$

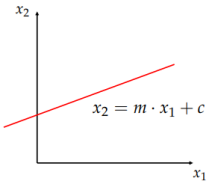
Order of Transformations

2D Curves

There are three ways to represent 2D curves.

Explicit Representation

Represents **one axis is represented as a function of another,** giving a relationship between dimensions. i.e. $x_2 = f(x_1)$



e.g. a line is represented as $x_2 = m * x_1 + c$.

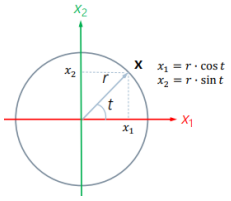
Can't represent vertical lines or output two values! *I.e. can't draw a circle*

Parametric Representation

Maps from a **parameter t to points on the curve** using vectors a and v. Generally used.

$f(t) = a + t * v$

e.g. a circle is $f(t) = r \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}$

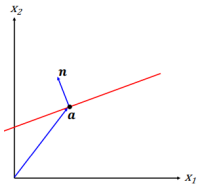


Like describing pen position at time t!

Easily extended to 3D!

Implicit Representation

A **function of points in space, where $f(x) = 0$ when the point is on the curve**.

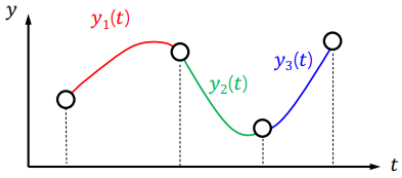


e.g. $f(x) = n * (x - a)$ *is a line and* $f(x) = ||x|| - r$ *is a circle!*

Tells you how close points are to the curve!

Piecewise Modelling

Represent a complex curve by breaking it up into tiny segments and modelling each with a different function.



C^n Continuity

A **curve/function is C^n continuous if its n^{th} derivative is continuous**. Determines how two curves connect to each other.

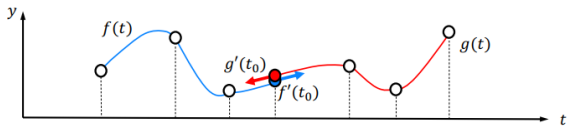
- C^n continuity implies C^{n-1} continuity.

C^0 Ensures Connections are Continuous!

Given the connection is at t_0, if $f(t_0) = g(t_0)$, the connection is continuous!

C^1 Ensures Connections are Smooth!

Given the connection is at t_0, if $f'(t_0) = g'(t_0)$, the connection is smooth!



Cubic Curves

Parametric curves defined by a **third-degree polynomial**: $p(t) = x_0 + tx_1 + t^2x_2 + t^3x_3$

Rewritten in matrix notation:

$$\mathbf{p}(t) = \underbrace{\begin{bmatrix}\mathbf{x}_3 & \mathbf{x}_2 & \mathbf{x}_1 & \mathbf{x}_0\end{bmatrix}}_{\text{Shape matrix } \mathbf{C}} \underbrace{\begin{bmatrix}t^3\\t^2\\t\\1\end{bmatrix}}_{\text{Parametrisation matrix } \mathbf{Q}(t)} = \mathbf{C}\,\mathbf{Q}(t)$$

The tangent indicates **direction and speed** at the point of drawing.

$$\frac{d}{dt}p(t) = x_1 + 2x_2t + 3x_3t^2 = C[0, 1, 2t, 3t^2]$$

Hermite Curves *Memorise!*

A type of cubic curve **defined by the values and gradients at its endpoints**. Factor out the shape matrix C into a **shape/geometry** matrix G and a **blending matrix** M.

G holds the **customisable** endpoints and derivatives.

M and Q(t) are constant across all Hermit curves!

$$\mathbf{p}(t) = \underbrace{\begin{bmatrix}\mathbf{p}(0) & \mathbf{p}(1) & \mathbf{p}'(0) & \mathbf{p}'(1)\end{bmatrix}}_{\text{Shape/ Geometry matrix } \mathbf{G}} \underbrace{\begin{bmatrix}2 & -3 & 0 & 1\\-2 & 3 & 0 & 0\\1 & -2 & 1 & 0\\1 & -1 & 0 & 0\end{bmatrix}}_{\text{Blending matrix } \mathbf{M}} \underbrace{\begin{bmatrix}t^3\\t^2\\t\\1\end{bmatrix}}_{\text{Parametrisation } \mathbf{Q}(t)}$$

$$p(t) = GMQ(t)$$

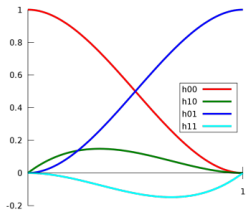
MQ(t) are the **basis functions**!

Changing G changes the shape of the curve!

The Basis Functions

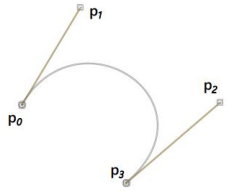
Multiplying M and $Q(t)$ gives us the basis functions, which define a **B-spline**.

$$\begin{aligned}\mathbf{p}(t) = {} & \mathbf{p}(0)(1 - 3t^2 + 3t^3) + \\ & \mathbf{p}(1)(3t^2 - 2t^3) + \\ & \mathbf{p}'(0)(t - 2t^2 + t^3) + \\ & \mathbf{p}'(1)(-t^2 + t^3)\end{aligned}$$



Bézier Curves *Memorise!*

Ensure that the gradients take specific values to satisfy the **convex hull property**, that the curve will never pass outside of the space formed by the four control points:

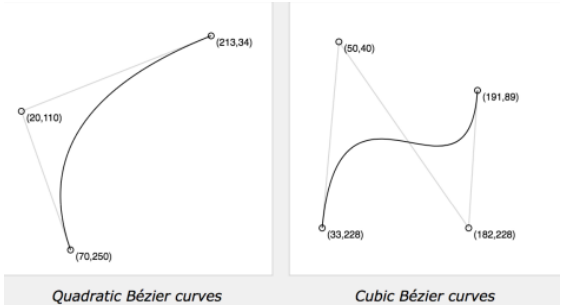


$$\frac{d\mathbf{p}}{dt}(0) = 3(\mathbf{p}_1 - \mathbf{p}_0)$$
$$\frac{d\mathbf{p}}{dt}(1) = 3(\mathbf{p}_3 - \mathbf{p}_2)$$

This constraint creates the **Bézier blending matrix** and **basis functions**.

$$= \begin{bmatrix}\mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\end{bmatrix} \begin{bmatrix}-1 & 3 & -3 & 1\\3 & -6 & 3 & 0\\-3 & 3 & 0 & 0\\1 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}t^3\\t^2\\t\\1\end{bmatrix}$$

Types: Linear, quadratic, cubic or quartic.



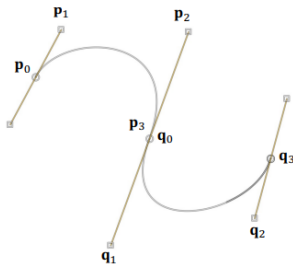
Quadratic Bézier curves *Cubic Bézier curves*

To achieve continuity (C^0):

$$\mathbf{p}(1) = \mathbf{q}(0)$$
$$\mathbf{p}_3 = \mathbf{q}_0$$

To achieve smoothness (C^1):

$$\mathbf{p}'(1) = \mathbf{q}'(0)$$
$$3(\mathbf{p}_3 - \mathbf{p}_2) = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{p}_3 - \mathbf{p}_2 = \mathbf{q}_1 - \mathbf{p}_3$$
$$\mathbf{q}_1 = 2\mathbf{p}_3 - \mathbf{p}_2$$



Surfaces, Edges & Corners

Read over PowerPoints. Only need to remember equations relating to **derivatives of gaussians & canny edge detectors**.

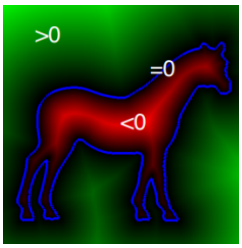
Implicit Representation

Space partitioning:

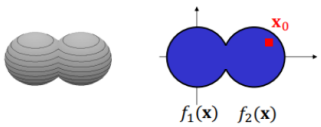
$\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) > 0\}$ Outside

$\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) = 0\}$ Curve/Surface

$\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) < 0\}$ Inside



Union of 2 spheres: $f(\mathbf{x}) = \min(f_1(\mathbf{x}), f_2(\mathbf{x}))$



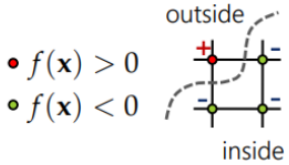
$f_1(\mathbf{x}_0) > 0$
$f_2(\mathbf{x}_0) < 0$
$f(\mathbf{x}_0) < 0$

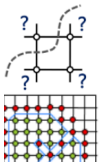
Intersection: $f(\mathbf{x}) = \max(f_1(\mathbf{x}), f_2(\mathbf{x}))$

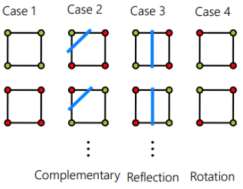
Marching Squares

for 2D curves, we have 16 (2^4) different configurations

4 equivalence classes (up to rotational and reflection symmetry + complement)



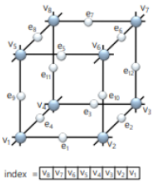


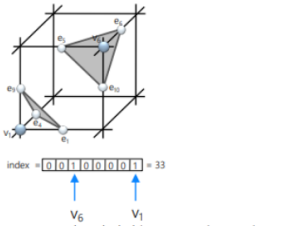


$f(\mathbf{x}) > 0$
$f(\mathbf{x}) < 0$

Marching Cubes

$2^8 = 256$ cases.

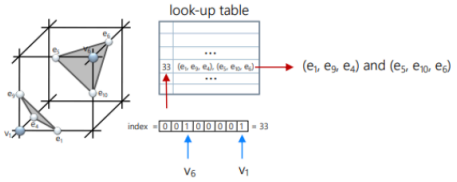




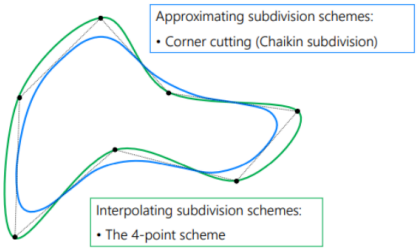
We can convert the whole binary array into an integer

Using the case index, retrieve the connectivity in the look-up table

- Example: index 33 in the look-up table indicates that the cut edges are e_1, e_4, e_5, e_6, e_9 and e_{10}; the output triangles are (e_1, e_9, e_4) and (e_5, e_{10}, e_6)



Subdivision Subsurfaces



1D Edge Detection

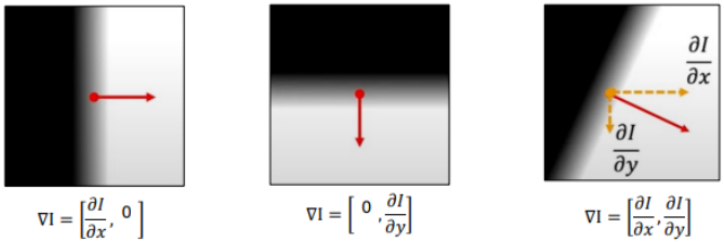
First Derivative: $\left|\frac{df}{dx}\right|$

Local Maxima
Indicate Edges

2D Edge Detection

Also calculate gradient magnitude and direction. Finite differences is used to approximate partial derivatives. Memorise!

$$\nabla \mathrm{I} = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$ Pronounced as "Del I"



Thresholding

Compares partial derivatives to a threshold. **Standard** uses one and **hysteria based** uses two:

- $\|\nabla I(x,y)\| < T_0$ Definitely Not an Edge
- $\|\nabla I(x,y)\| \geq T_1$ Definitely an Edge
- $T_0 \leq \|\nabla I(x,y)\| < T_1$ Is an Edge if a neighbouring pixel is Definitely an Edge

Canny Edge Detectors

Smooth the image with Gaussians and use the sobel operator to calculate gradients. Changing σ refines edge detection.

Smooth Image with 2D Gaussian: $n_\sigma * I$

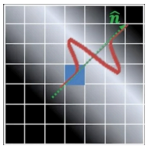
Compute Image Gradient using Sobel Operator: $\nabla(n_\sigma) * I$

Find Gradient Magnitude at each pixel: $\|\nabla(n_\sigma) * I\|$

Find Gradient Orientation at each pixel:

$$\hat{n} = \frac{\nabla(n_\sigma) * I}{\|\nabla(n_\sigma) * I\|}$$

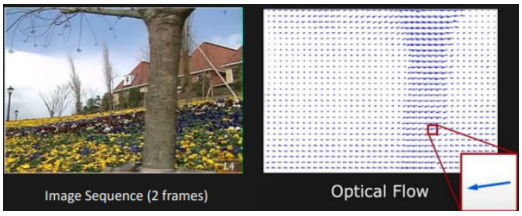
Compute 1D Laplacian along the Gradient Direction $\hat{n}$ at each pixel $\frac{\partial^2 (n_\sigma * I)}{\partial \hat{n}^2}$



Optical Flow & Motion Field

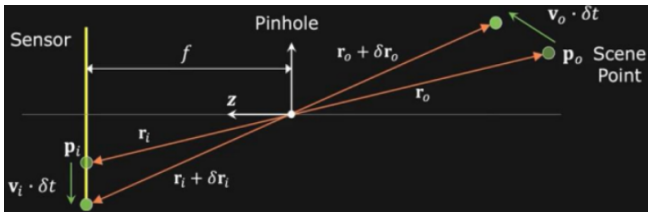
Motion Field: The actual motion of a point in 3D space projected onto a 2D camera plane.

Optical Flow: Estimates the motion of video points / changing patterns using the pixel displacement between frames.



The Pinhole Camera Model

Captures a **scene point** in 3D space as an **image point** on the 2D camera sensor.



- v_i and v_0 are the image and scene point velocities.
- r_i and r_0 are the original positions.
- ∂r_i and ∂r_0 represent displacement from movement.

$$\frac{r_i}{f} = \frac{r_o}{r_o \cdot z}$$

Projection Equation

Describes the relationship between the coordinates of a 3D points (r_0) and its projection on the 2D plane (r_i).

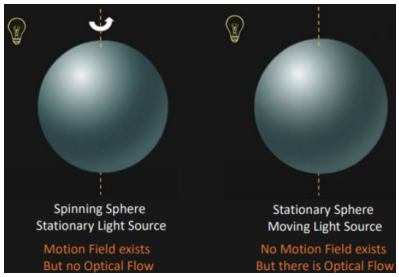
Velocities

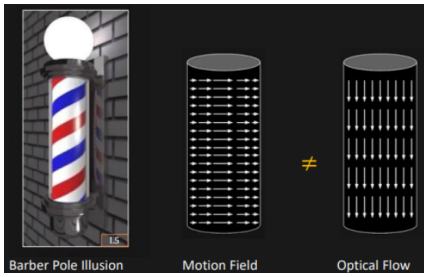
Image Point Velocity: $v_i = \frac{dr_i}{dt}$ Scene Point Velocity: $v_0 = \frac{dr_0}{dt}$

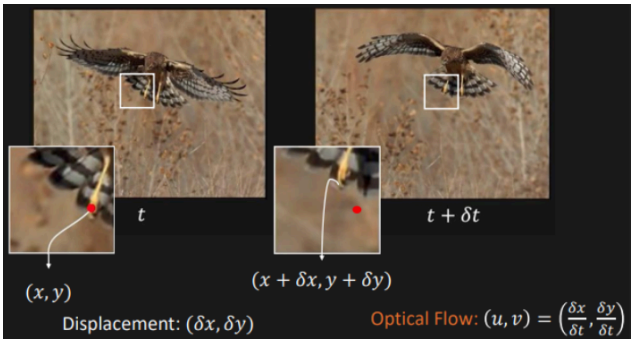
Differentiating the projection formula gives an expression for the **image point velocity** based on the scene point:

$$\mathbf{v}_i = \frac{d\mathbf{r}_i}{dt} = f\frac{(\mathbf{r}_o \cdot \mathbf{z})\mathbf{v}_o - (\mathbf{v}_o \cdot \mathbf{z})\mathbf{r}_o}{(\mathbf{r}_o \cdot \mathbf{z})^2} = f\frac{(\mathbf{r}_o \times \mathbf{v}_o) \times \mathbf{z}}{(\mathbf{r}_o \cdot \mathbf{z})^2}$$

When Optical Flow ≠ Motion Field







Estimating Optical Flow

Given two consecutive images, the **optical flow constraint equation** estimates the displacement and optical flow vector of a point.

The intensity of two consecutive points at time t is: $I(x, y, t)$ and $I(x + \partial x,\ y + \partial y,\ t + \partial t)$

The Constraint Equation

Describes how **pixel intensity changes over time**, assuming:

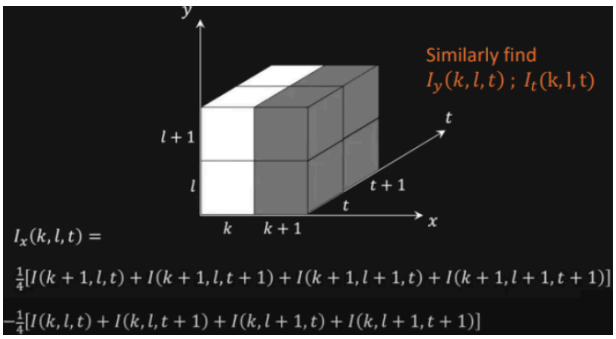
Point brightness is constant. The steps are small.

$$I_x u + I_y v + I_t = 0$$

where:

- $I_x = \frac{\partial f}{\partial x}$, etc are the partial derivatives of intensity with respect to x, etc.
- $u = \frac{\partial x}{\partial t}$ and $v = \frac{\partial y}{\partial t}$ (the displacement)

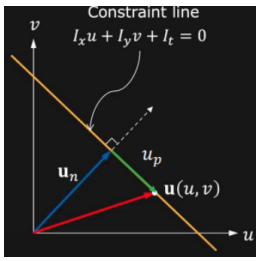
Calculating Partial Derivatives



The intensity partial derivatives are calculated using the **finite differences approximation.** Pixels from consecutive images can be represented as blocks. $I_x(k, l, t)$ is the intensity at $x = k, y = l$ and $t = 1$ in the x direction. Take the average difference over a neighbourhood in the spatial and temporal domains.

Notice: k stays the same in each because we're finding I_x.

Geometrical Interpretation & The Aperture Problem



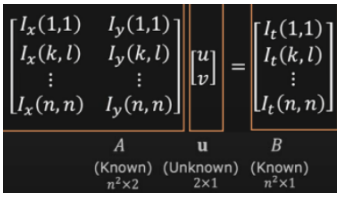
For any point (x, y), its optical flow vector (u, v) lies on the constraint line, and can be split into **normal flow** (u_n) and **parallel flow** (u_p): $u = u_n + u_p$

The direction and magnitude of normal flow can be calculated, but parallel flow is more difficult. This is called the **aperture problem**, caused by having two unknowns in the constraint equation.

If the viewpoint is restricted, you can only see normal flow.

The Lucas-Kanade Solution

Assume the motion field and optimal flow is constant within a neighbourhood around each pixel, allowing the removal of the t factor. Now, generate multiple constraint equations and express them as matrices.



Where the window size is n. Solving for u gives:

$$\boldsymbol{u} = (A^T A)^{-1} A^T B$$

Unreliable Flow Computations

$A^T A$'s eigenvalues must be **different but not massively** and both **can't be tiny**, which can be caused by:

Smooth regions: low eigenvalues. **Edges:** large difference.

Both eigenvalues are high for textured regions, meaning we can reliably compute flow!

Coarse-to-Fine Estimation Algorithm

Starting from a low resolution, iteratively increasing it, calculating optimal flow, warping the before frame and recalculating it **improves flow estimation accuracy**. Continue iterating until it converges.

Change Detection

Given a static camera, meaningful **changes can be identified by classifying pixels as foreground or background** robustly in real-time.

Frame Difference Formula

Significant differences between frames are labelled as foreground by comparing the intensity difference with a threshold $F_t = |I_t - I_{t-1}| > T$

Not Robust: Picks up background movement.

Background Modelling

Compare each frame to a **background model** B rather than the previous intensity.

Mean

Calculates the mean intensity of the first k pixels for the background.

Undynamic: First k frames unusable.

Median

Calculates the median intensity instead.

Computationally Inefficient

Moving Median

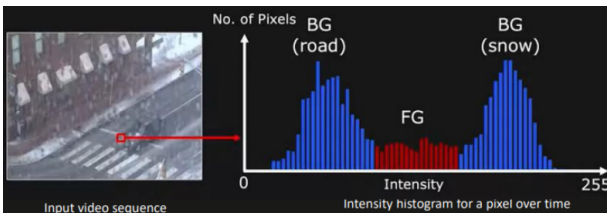
Takes the median of the **last** k **frames**, making it **more dynamic**.

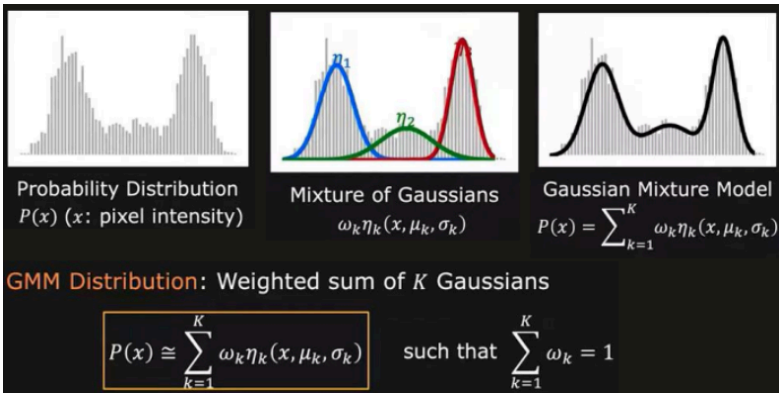
Last k frames must be stored.

Cannot handle weather, shadows, shake, etc.

Gaussian Mixture Models

Plots each pixel's **intensity over time on a histogram**. Variations are caused by static and **moving objects**. By assuming that it's comprised of several Gaussian distributions, it can be modelled by a Gaussian mixture model, which allows us to calculate the **probability of a pixel taking a specific intensity and classify it**.

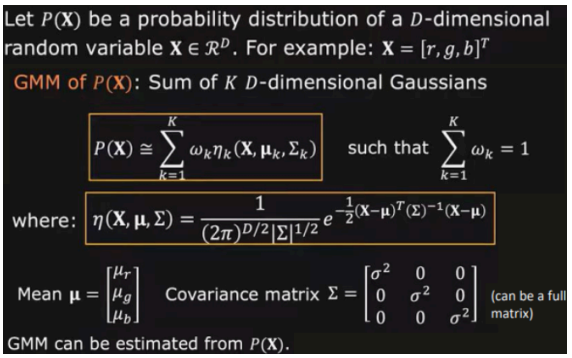




ω is the weight/scale/evidence, which represents how strongly a particular Gaussian component contributes to the overall model

High Dimensional GMMs

Increasing the number of dimensions using **multivariate Gaussian distributions** enables intensity of all **RGB channels**.

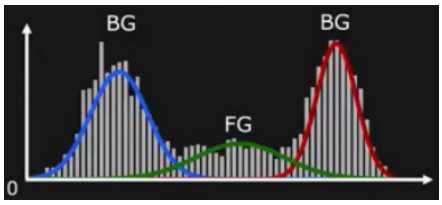


The covariance matrix represents the relationship between the channels.

- Constant σ = same width.
- Different σ = different widths.
- Only diagonals means they're independent.
- Non-diagonals means they're dependent.

Classification

Assuming that pixels are background most of the time:



Large $\frac{\omega}{\sigma}$: Background Small $\frac{\omega}{\sigma}$: Foreground

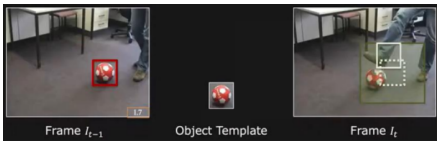
Adaptive Change Detection Algorithm

1. Compute and normalise histograms from frames.
2. Model histograms as 3-5 Gaussians.
3. For each frame:
    a. The pixel X belongs to the Gaussian distribution that its intensity values are closest to.
    b. Classify as background if $\frac{\omega}{\sigma}$, else as foreground.
    c. Update the histogram with the new intensity for the dynamic moving average and update the histogram if the difference is significant.

Object Tracking with Templates

Given the location of a target in a previous frame, there are three methods of finding it in the current one.

Appearance-based Tracking



Given a template T in frame I_{t-1}, search the neighbourhood S to find the same window in image I_t.

Sum of Absolute Differences

$$SAD(k,l) = \sum_{(i,j)\in T} |I_1(i,j) - I_2(i+k, j+l)|$$

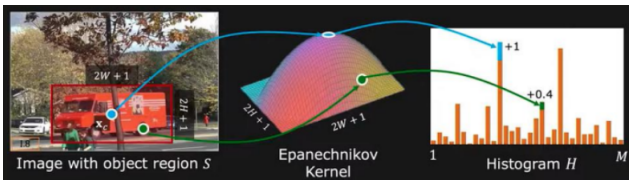
Sum of Squared Differences

$$SSD(k,l) = \sum_{(i,j)\in T} |I_1(i,j) - I_2(i+k, j+l)|^2$$

Minimum Normalised Cross-Correlation

$$NCC(k,l) = \frac{\sum_{(i,j)\in T} I_1(i,j)\, I_2(i+k, j+l)}{\sqrt{\sum_{(i,j)\in T} I_1(i,j)^2 \sum_{(i,j)\in T} I_2(i+k, j+l)^2}}$$

Histogram-based Tracking

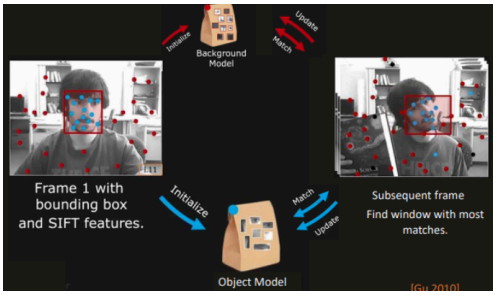


Apply a kernel/filter to the template T, which gives more importance to the pixels in the centre, to get a **weighted histogram** of the pixels.

Then, search for the window in S for a match.

- More robust to changes in object pose, scale, illumination and occlusion but struggles with common objects.

Feature-based Tracking



1. Run a frame with a bounding box through a SIFT detector and add inside to an object model and those outside to a background model.

2. Re-run on the next frame and match the points with those in the models, for each feature and descriptor v_i:
    a. Compute the total distance d_0 between it and all points in the previous object model.
    b. Repeat for the background model, generating d_b.
    c. $$C(\mathbf{v}_i) = \begin{cases} +1 & \text{if } {}^{d_O}/_{d_B} < 0.5 \ (\mathbf{v}_i \text{ may belong to object}) \\ -1 & \text{otherwise } (\mathbf{v}_i \text{ does not belong to object}) \end{cases}$$
3. Iteratively redraw the bounding box, keeping its area constant, to contain as many points as possible. Select the one with the highest **match score**.
4. Update the object model to contain the new object model points.

Graphics

OpenGL

A **computer graphics API** for rendering with **geometric primitives**, often used in **interactive applications** with 3D graphics.

- OS & window independent.

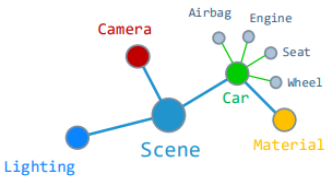
WebGL *a.k.a. web graphics library*

A **JavaScript API** for browser 3D graphics rendering, based on OpenGL.

- No compilation needed.
- Run locally.
- Supported by all major browsers.

three.js App Skeleton

three.js simplifies WebGL use.

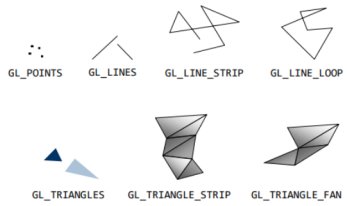


`three.js` maintains a graph for objects
[Scene is the root of a graph]

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, window.innerWidth /
    window.innerHeight, 0.1, 1000);

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

Generally use an animation function which recursively calls itself to update object positions.



Geometric Primitives

Basic building block shapes or objects, specified by their vertices.
Doesn't include points!

Shape Acquisition

3D models can be created **manually with software** or by **3D scanning with depth sensors**. But how are they stored?

Triangle Meshes

Each triangle is a set of vertices and a face.

- Simple & general.
- Render efficiently.
- Output by most scanners.

Storage

Should be:

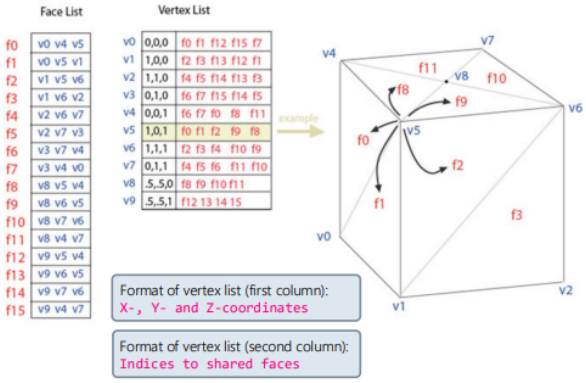
Easy to **edit**. *E.g. add/delete vertices & faces.*

Easy to **access**/instant access to neighbours.

Easy to **display**. *E.g. looping over vertices for fast drawing.*

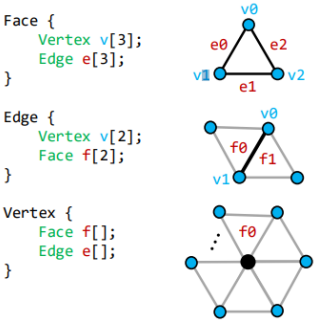
- **PLY Files:** Contains a header, list of vertices & a list of triangles made from these.
  - Triangle Format: (no. of vertices, first index, …, last index)
  - Vertex indexes start from 0. *e.g. 3 0 1 2*
- **OBJ Files:** Contains a list of vertices & triangles made from them.
  - Vertex indexes start from 1 *e.g. f 1 2 3*

The Face-Vertex Data Structure

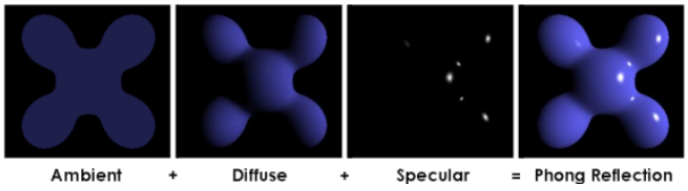


The Face-Edge-Vertex Data Structure

Highly redundant.



Lighting



Phong Lighting Model

No intra-reflections!

Most popular lighting model. Combines **specular highlights** for shiny surfaces, **diffuse highlights** for dull surfaces and **ambient** light to produce reflections.

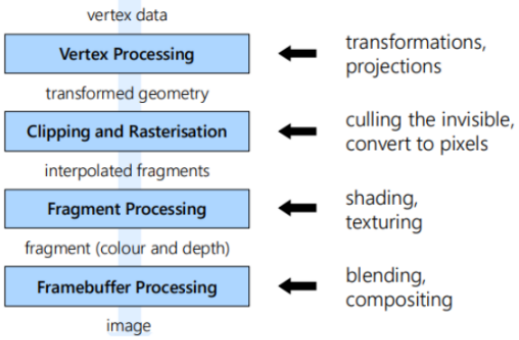
Raytracing

Computationally expensive but **physically accurate** light simulation, with reflections, shadows, motion-blur, etc but no caustics!

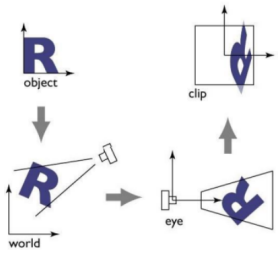
The Graphics Pipeline

A **sequence of stages** which **process a single object/surface/fragment at a time**. **Modular**, easy to optimise and implement. **Handles simple primitives**, but supports complex ones through **tessellation**.

Tessellation **approximates complex surfaces** with a triangle mesh.



Memorise!



Stage 1: Vertex Processing

Applies transformations from the object space to the clip/camera space.

- **Input:** Vertex data (position, colour, etc).
- **Output:** Transformed vertices, colours, etc.

Stage 2: Clipping & Rasterisation

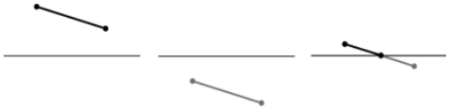
Turns vertices into primitives.

- **Input:** Vertices in clip space, colours, etc.
- **Output:** Set of **fragment descriptions** (3D positions, depth, normals, etc).

Clipping/Culling

Removes object parts not in the view frustum for efficiency with later steps.

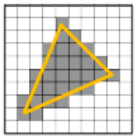
- **Back-face Culling:** Removes faces which are facing away from the cameras(inside surfaces of shapes) where $plane\ normal\ *\ viewing\ vector < 0$.
- **Point-plane Clipping:** Tests if a point is on the right side of the plane by removing points where $plane\ normal\ *\ point\ vector < 0$.
- **Segment-plane Clipping:** Tests endpoints of a plane to clip occluded sections.



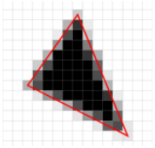
Rasterisation

Converts **continuous primitives to discrete pixels**.

The **naive approach** checks if the pixel is in the primitive or not (aliasing boundary).

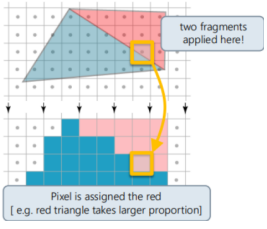


The smoothing approach/anti-aliasing:



Line rasterisation scans each column to find covered pixels and turns the closest to the line.

Triangle rasterisation converts triangles defined by vertex coordinates to fragments, pixel coordinates and shading data.

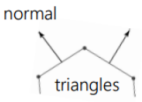


Stage 3: Fragment Processing

Processes the **fragments** to add **shading**, **textures** and **depth**.

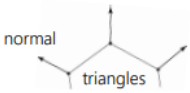
Flat Shading

Calculated from triangle normals.



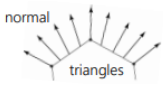
Gouraud Shading

Calculated from vertex normals.



Phong Shading

Interpolates normals and calculates shading for each pixel.



Texture Mapping

A function applies an 1D, 2D or 3D image/colour to a surface by calculating the corresponding points between the mesh and texture.

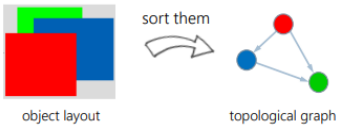
Stage 4: Framebuffer Processing

Looks at how objects overlap to render visible surfaces and composes transparent surfaces using framebuffer blending.

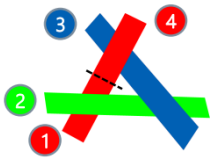
- **Input:** Fragments.
- **Output:** Final image.

Painter's Algorithm

Topologically sorts the objects by depth then fills the framebuffer by **drawing objects from back-to-front**.



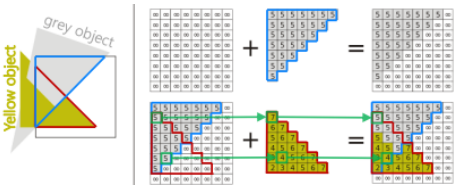
- Simplest method.
- **Inefficient:** Overpaints hidden surfaces.
- An edge from A to B means "A occludes B".
- No valid sorting order if there are cycles, so you need to **split the polygons**.



The Z-Buffer

Maintaining an ordered graph is expensive. Drawing objects in **any order** by **storing the closest depth for each pixel in the Z-Buffer** solves this. When adding a pixel, compare the depth to the value in the Z-Buffer:

- If less, update the colour framebuffer and depth framebuffer.
- If greater, discard the object.



Animation

Animations are **sequences of images** played in **rapid succession** to give the **illusion of movement**.

Keyframing

Drawing characters at key poses and filling in frames between.

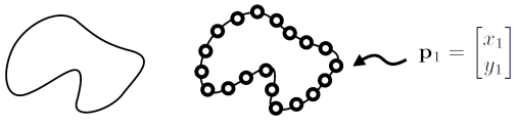
Labour intensive!

Motion Capture

Records live action and transfers motion to a virtual character.

Rigging

Determines a **character's realistic movement** and animation. Requires artistic and technical training!



Procedural Rigging

A function is applied to the points specifying the shape.

- Used for linear (*e.g. scaling*) or non-linear deformation (*e.g. twists/bends*).

Skeleton-based Rigging

Approximates the character's actual skeleton with skin, a hierarchy of components, joints and controls.

Image Matching

Applications

- **Digital Mapping:** Assigning descriptions to significant locations.
- **Environment Modelling:** Monitoring crops with drones.
- **Orthophotos:** An aerial photograph corrected for scale and distortion.
- **Medicine:** Aligning MRI scans over time.

Template Matching

Finds the location of a template in an image, **direct alignment (pixel-based matching)**.

Given an image $I_1(x)$ and template $I_0(x)$, need to find the offset $u = [\hat{u}, \hat{v}]$ between them.

Cross-Correlation (CC)

Optimally (when only translated) measures the similarity between a template and an image region's intensity values, assuming they only differ by translation, brightness and contrast. Find the best u by maximising:

Memorise!

$$E_{CC}(u) = \sum_i I_0(x_i) I_1(x_i + u)$$

$E = 1$ where there's a match!

Exhaustive Search: Checks all possible offsets.

Coarse-to-Fine: Iteratively resize an image from small to large. The max match in small gives initialisation for next search.

Feature-based Alignment

Searches for alignment where **extracted features** agree.

Least-Squares Matching

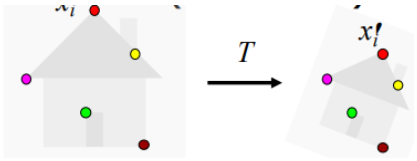
High Precision!

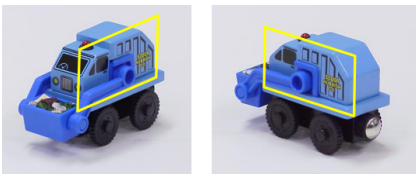
Points are also known as tie/key/interest points.

Least-squares matching can find a line to best fit a set of points by minimising the sum of squared differences, and can be applied elsewhere!

Identifying a transformation, similar, affine or projective, between pairs, and minimising the difference between the new points under the transformation and the target:

$$\sum_i residual(T(x_i), x_i')$$





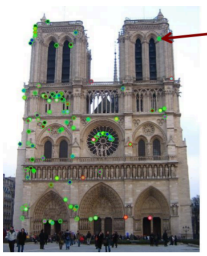
Feature Extraction

First, need to find the keypoints and identify the pairs!

Descriptor: Summarises the structure around a keypoint.

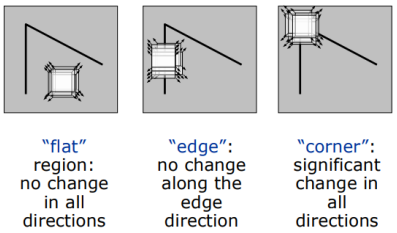
Keypoint/Feature Point: Where a description is computed.

Local features are robust to occlusion!



Harris Corner Detector

Viewing points through a window and analysing the intensity changes when we move in each direction helps identify corners. Memorise!



$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function | Shifted intensity | Intensity

The window function weights pixels flatly or Gaussian.

Reduce computational load by calculating a **bilinear approximation** for small shifts using a matrix M calculated from the image derivatives.

$$E(u,v) \cong \begin{bmatrix} u,v \end{bmatrix} \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The Eigenvalues of M can classify points:

- **Flat Region:** λ_1 and λ_2 are small.
- **Corner:** Both are large.
- **Edge:** One is large, one is small.

$$R = \lambda_1 \lambda_2 - k\left(\lambda_1 + \lambda_2\right)^2$$

The **measure of corner response** outputs large values for corners. Memorise!

1. Find points where the corner response is larger than a threshold to find corners.
2. Take the local maxima of these points.

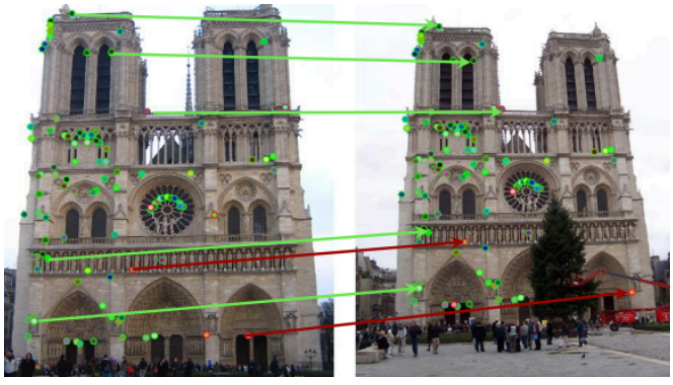
SIFT (Scale-Invariant Feature Transform)

HOG (Histogram of Oriented Gradients)

Descriptors

SURF (Speed-up Robust Features)

GLOH (Gradient Location & Orientation Histogram)



Identifying Pairs / Feature Matching

Given a feature I_1, find the best match I_2 using a distance function to find the one with min distance.

There are **outliers**!

Calculate the **sum of squared differences** for both descriptors. You can assess the number of true and false positives using a **maximum distance threshold** which flags matches over a certain distance as false.

Correspondence Problem: Similar local areas between two incorrect points leads to false matches!

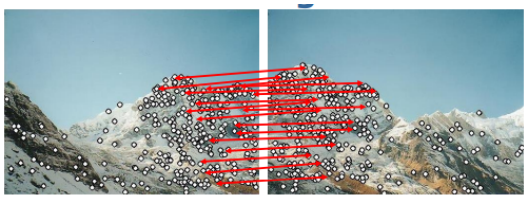
RANSAC (RANdom SAmple Consensus)

Identifies outliers by finding the best partition of inlier set points and outlier set points, allowing the model, the pairs, to be made of only inliers. It adopts a **trial-and-error approach**:

1. **Sample** the number of data points needed to fit the model.
2. **Compute** model parameters using the samples.
3. **Score** using the fraction of inliers within a threshold of the model.
4. **Select** the model with the most inliers.

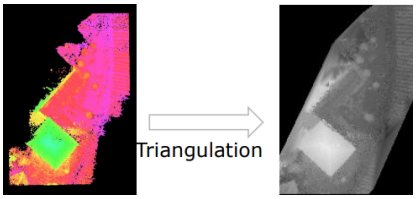
In the context of finding pairs, this means:

1. Extract features.
2. Compute **putative matches**.
3. Repeatedly until you find a pair with the most inliers:
    a. Pick a putative pair.
    b. Hypothesize transformation T from it.
    c. Verify it by checking the other pairs are consistent.



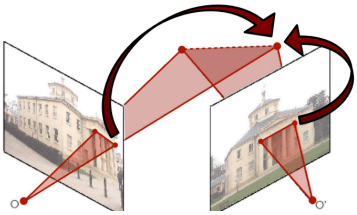
Point Clouds

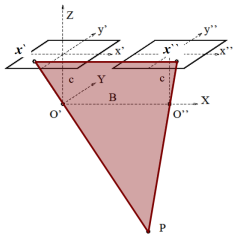
Point clouds are a collection of points in 3D space, where each is a match, found using stereo vision. Can be made into a mesh using **Delauny triangulation**. Dense point clouds are more effective because **sparse point clouds lead to detail loss**.



Stereo Vision

The **locations of points in 3D** can be calculated from the corresponding 2D points and camera orientations.





The **Stereo Normal Case!**

Two or more **identically oriented** cameras can work together to perceive depth by capturing images **in the same plane** from slightly different perspectives. The **disparity** is used to estimate depth. The cameras should be **parallel** to each other and **perpendicular** to the scene.

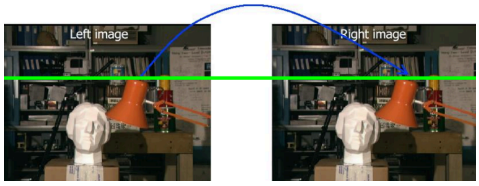
So, given a pair of images and the point pairs between them, we can perform a 3D reconstruction.

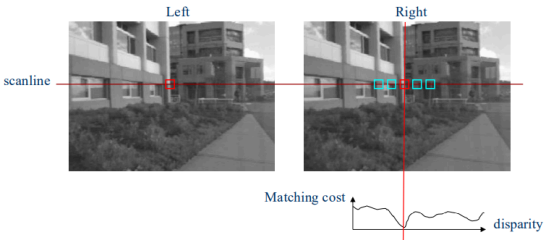
The Stereo Problem

After identifying sparse, robust key point pairs with SIFT or the Harris Corner detector, need to collect more using **window-based matching** to make a dense point cloud. Then, their **disparity** (difference in position) needs to be calculated to estimate the depth.

Pre-processing: Rectification

Images from different perspectives are **reprojected** so the corresponding pairs lie on a horizontal scanline.





Window-based Matching

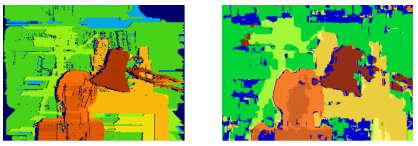
Pick a window in the left, slide a window along the right scanline, comparing the matching cost (**Sum of Absolute Differences**) to find the minimum. Doesn't enforce global smoothness.

Struggles with photometric variations, specularities, textureless regions, repetitive textures and large occlusions.

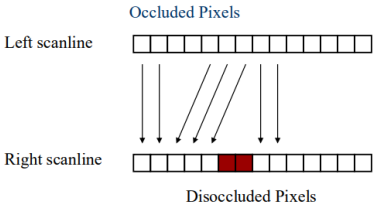
Scanline Stereo

Matches pixels on the scanline, looking at the entire row together, making sure it's smooth and looks at rows **independently**.

Prone to streaking!



Scanline matching | Window-based matching



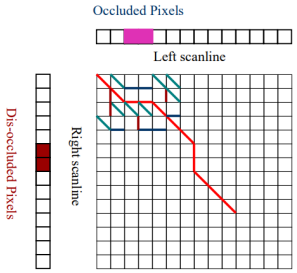
Occluded Areas

Occluded and **disoccluded** areas/pixels result in no match and a large matching cost! **Sequential (paired)** add a small cost if their intensities are close!

Dynamic Programming

Given these options, can represent finding the lowest matching cost with **scanline stereo** as an **optimal path problem** over a **cost grid** which uses **dynamic programming**.

- At each step, identify the optimal match and take it.
- Backtrack to get the optimal path.
- Travel horizontally/vertically at occlusions and disocclusions.

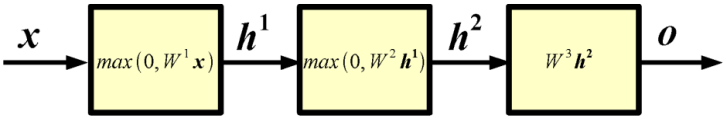


Deep Learning

A Brief History

- **Backpropagation (1986):** Efficient gradient calculation enabled gradient-based networks.
- **Long Short-Term Memory (1997):** Used for sequence modelling.
- **Convolutional Neural Networks (1998):** Learns spatially invariant image patterns using backpropagation. Reduced parameters.
- **ImageNet (2010):** A large dataset of labelled images.
- **AlexNet (2011):** A very successful CNN trained with GPUs.
- **Synthetic Datasets (2012):** Datasets created using 3D assets.
- **Mask R-CNN (2016):** Object tracking.

Neural Networks Memorise!



$x \in R^D \quad W^1 \in R^{N_1 \times D} \quad b^1 \in R^{N_1} \quad h^1 \in R^{N_1}$

$$h^1 = max(0, W^1 x + b^1)$$

x input
h^1 1-st layer hidden units
h^2 2-nd layer hidden units
o output

W^1 1-st layer weight matrix or weights
b^1 1-st layer biases

Assessing Network Performance

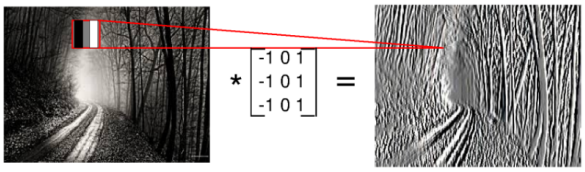
The loss function is calculated per sample, like the negative log-loss likelihood for classification.

Training

The network learns by minimising the loss using **backpropagation** to compute loss gradients w.r.t. the parameters and updating them.

Convolutional Networks

Used with images. **Passes a kernel over the pixels, allowing them to share parameters** (the kernel values) so they have less than normal networks.



They **learn the best filters** to capture the most important image **features**. Each layer has multiple features!

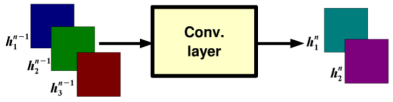
One-to-one (Image input -> class label)

The Maths Memorise!

Uses ReLU and replaces the multiplication with convolution.

$$h_j^n = max(0, \sum_{k=1}^{\wedge} h_k^{n-1} * w_{kj}^n)$$

output feature map | input feature map | kernel



Number of output layers = Number of filters

Local Contrast Normalisation

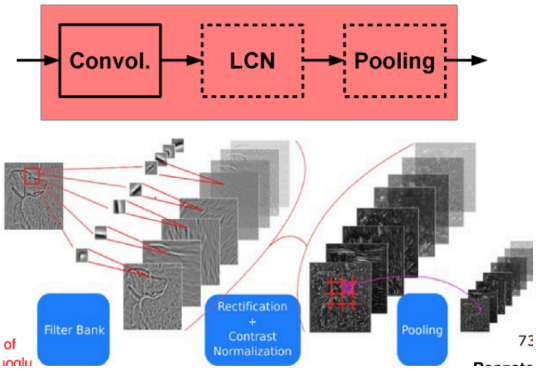
Ensures the network identifies the **same features regardless of contrast**. Improves invariance, optimisation and sparsity.

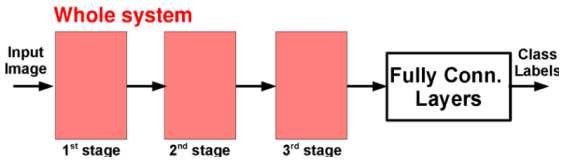
$$h^{i+1}(x,y)=\frac{h^i(x,y)-m^i(N(x,y))}{\sigma^i(N(x,y))}$$

Max Pooling

Takes the maximum of an area, reducing the number of features and making the network **robust to their exact locations**.

Typical Architecture





Use Cases

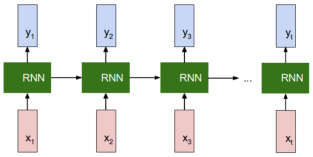
Performs better than SIFT with SVM.

- Optical character recognition.
- Classification.
- Object detection.
- Scene parsing.

Recurrent Neural Networks

Process sequences, enabling:

- One-to-many: image captioning.
- Many-to-one: action prediction.
- Many-to-many: video captioning.



They have an internal state which updates as a sequence is processed and is passed between the layers.

Image & Video Captioning

A **CNN captures each frame's key features** which are **processed by a recurrent neural network**.