CITY
UNIVERSITY OF LONDON
EST 1894

Computer Vision
INM460 / IN3060

Dr Sepehr Jalali

Lecture 2: Images and image processing

Admin updates

- New due date for coursework?
  - Sunday 28th April 2019
  - OR
  - Wednesday 1st May 2019
    - 5pm.
    - Both UG and PG
  - ????
- Lecture capture is now on Moodle.

Recap from last time

Module introduction

Human visual system

- Architecture
- Perception
  - Colour
  - Grouping (Gestalt)
  - Depth
  - Motion
- Illusions to illustrate concepts

⇨ We'll see many of these themes again in upcoming lectures

Overview of today's lecture

Images

- Formation
- Pinhole camera model
- Digital representation

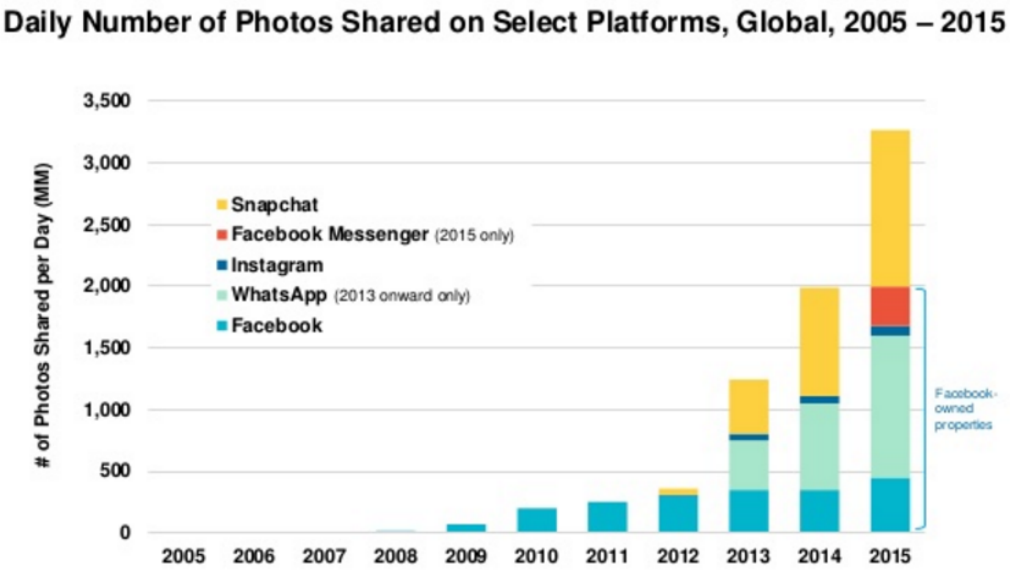
Digital image processing

- Colour and intensity transformations
- Geometric transformations
- *Filtering*

Images, and more images…

1990s

Today



http://www.kpcb.com/internet-trends

Images, and more images…

Media usage in an internet minute as of June 2018

Forecast requests received by The Weather Channel	18,055,555
Text messages sent	12,986,111
Videos watched by YouTube users	4,333,560
Google searches conducted	3,788,140
GB of internet traffic generated by Americans	3,138,420
Snaps shared by Snapchat users	2,083,333
GIFs servived by GIPH	1,388,889
Songs streamed on Spotify	750,000
Tweets sent by Twitter users	473,400
Calls made by Skype users	176,220
Hours of video streamed on Netflix	97,222
Posts published by Tumblr users	79,740
Dollars processed via Venmo P2P transactions	68,493

https://www.statista.com/statistics/195140/new-user-generated-content-uploaded-by-users-per-minute/

Consumer imaging devices

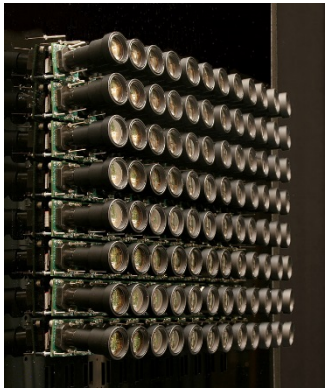
Digital cameras

Webcams

Go Pro

Nest

360° cameras



Camera arrays

Smart phones

What is an image?

An image is multi-dimensional signal that measures a physical quantity.

Multi-dimensional

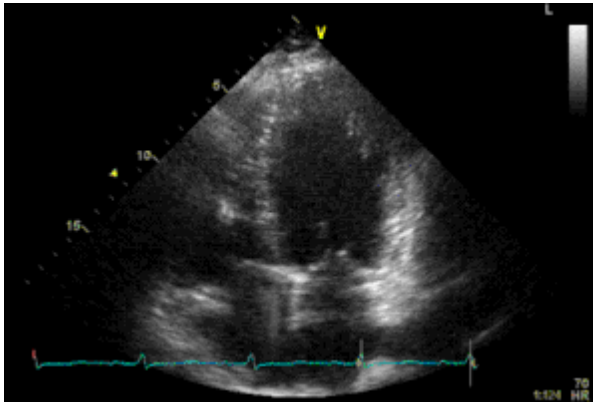
- 2D: Image (as a function of *x* and *y*)
- 3D: Video (as a function of *x*, *y*, and *t*)
- (others, e.g. CT, MRI)

Physical quantity

- Typically brightness for standard photographs
- Many other possibilities (temperature, depth, acoustic properties, etc.)

Digital photo

Thermal image



Ultrasound video

Multi-dimensional function

Mathematically, an image is a multi-dimensional function. For example, a grayscale image can be expressed a mapping from the set of real numbers in 2D space to the set of real numbers (brightness).

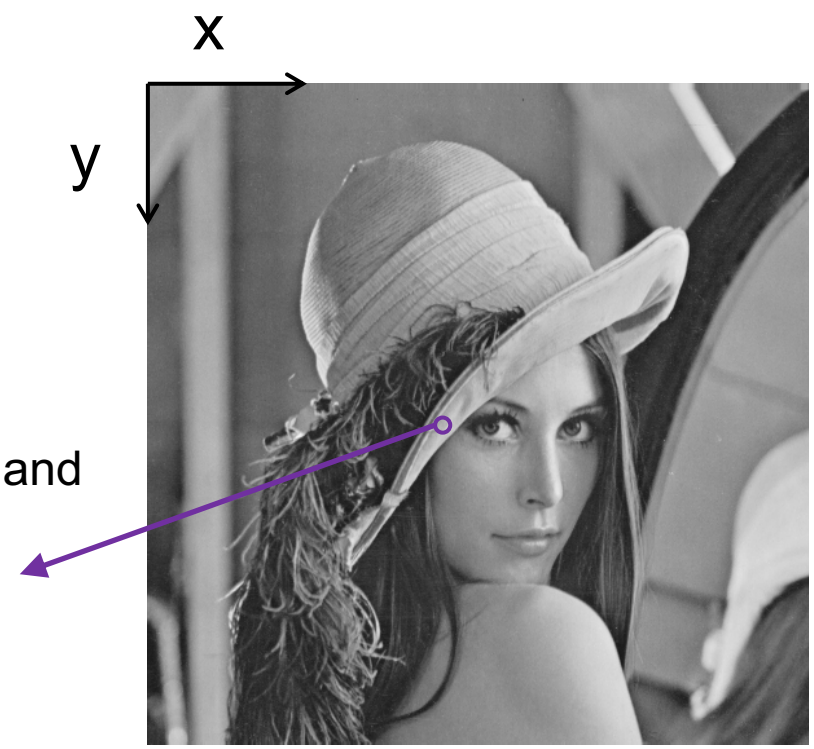
We could write

$$I : \mathbb{R}^2 \rightarrow \mathbb{R}$$

Or simply

$$I(x, y)$$

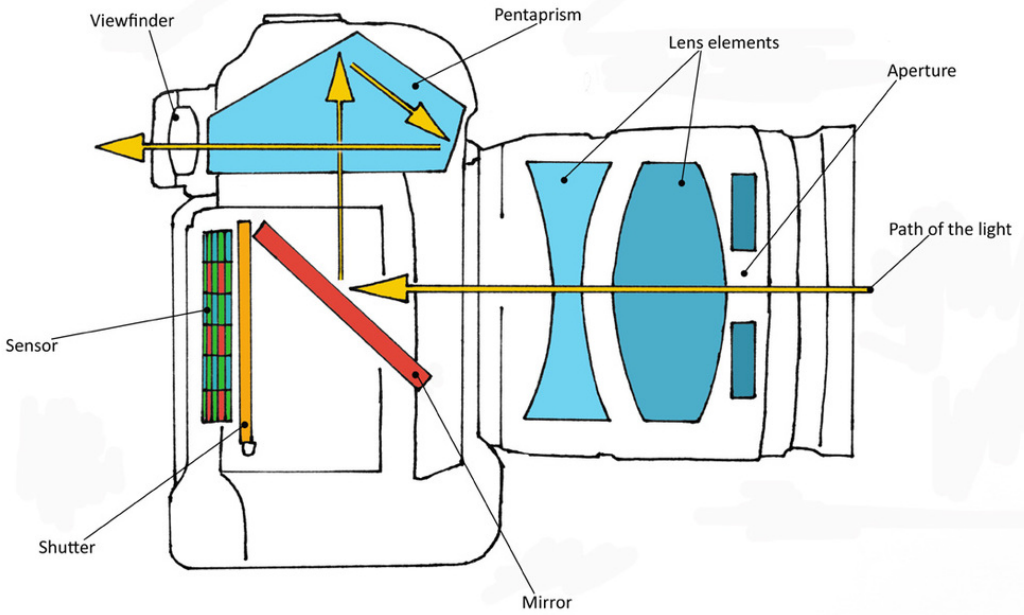
Where *x* and *y* are the spatial variables, and *I* is the intensity



Digital Camera

Typical components

- Aperture to let light in
- Lens
- Pentaprism
- Photosensitive image plane
- Housing to keep stray light out



http://murrayparkphotography.weebly.com/inside-a-dslr.html

https://www.youtube.com/watch?v=Ic0czeUJrGE

Pinhole camera

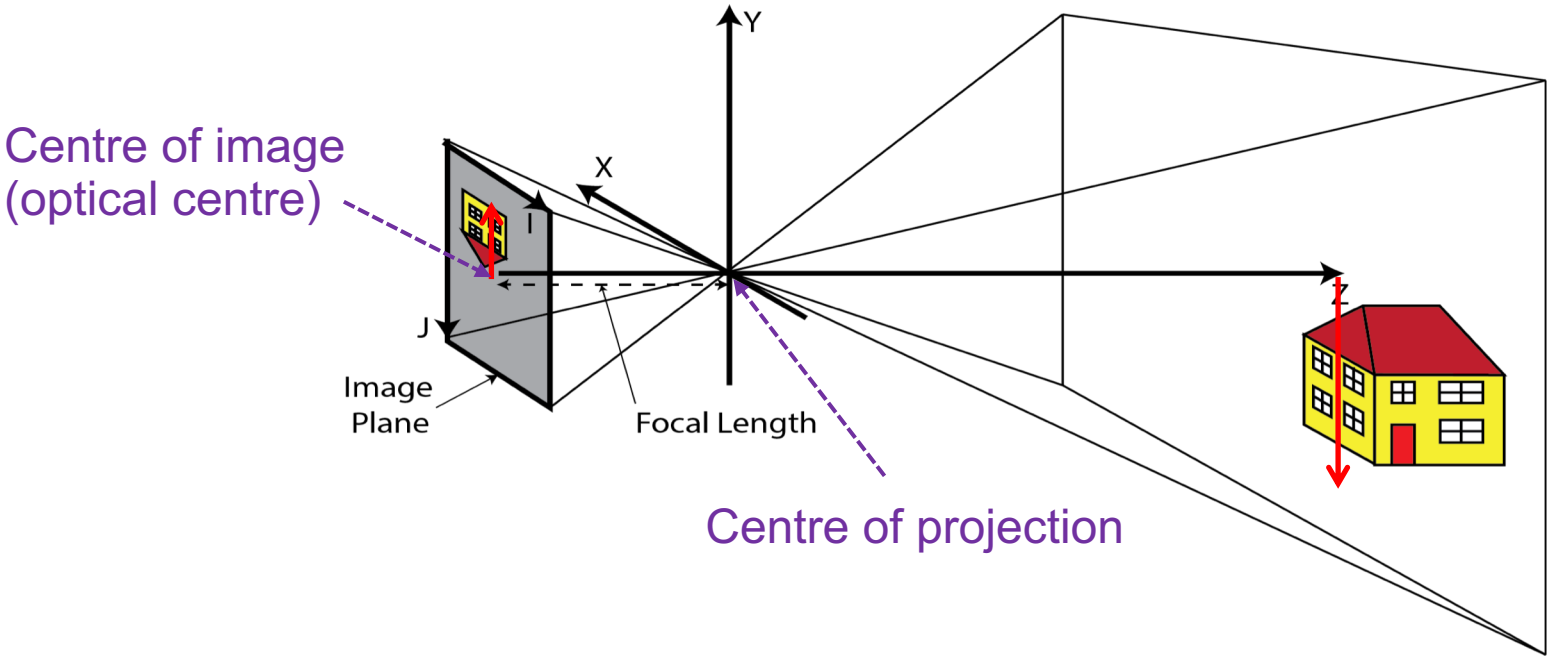
- All light passes through a single point, known as the *centre of projection*.
- On the other side of the camera is film that captures impinging light.
- This creates a *perspective projection*.

http://petapixel.com/2013/03/26/how-to-create-a-homemade-large-format-pinhole-camera-using-a-shoebox/

Pinhole camera model

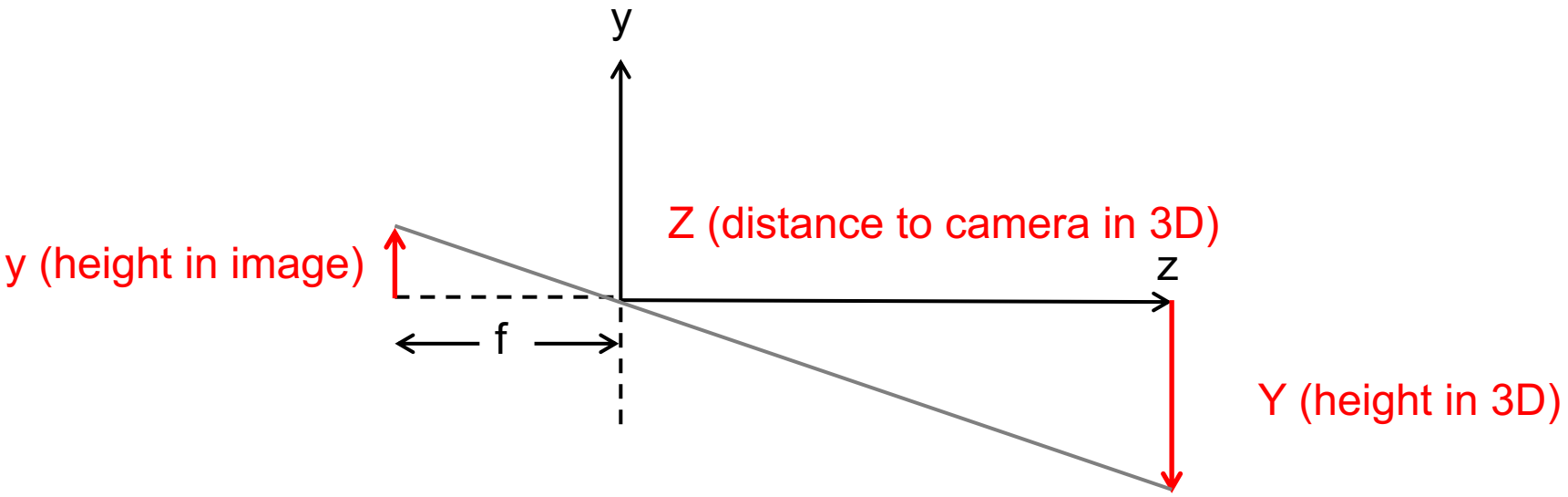
This can be modelled mathematically

- Whilst simple, the model is reasonably realistic
- Deviations occur due to lenses that create distortions



Camera coordinates

Pinhole camera model



By similar triangles, $\frac{y}{f} = \frac{Y}{Z}$ or $y = \frac{fY}{Z}$

⇨ As Z increases, y decreases (i.e., farther away objects appear smaller!)

Pinhole camera model

More generally, we can encode this projection using matrix / vector notation,

$$\mathbf{x} = K\mathbf{X}$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

where f_x, f_y are the focal lengths in x and y
c_x, c_y are the centre of the image, and
s is a skew (typically 0)

K is known as the ***intrinsic*** *camera matrix*

Let's multiply it out…

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} f_x X + sY + c_x Z \\ f_y Y + c_y Z \\ Z \end{bmatrix}$$

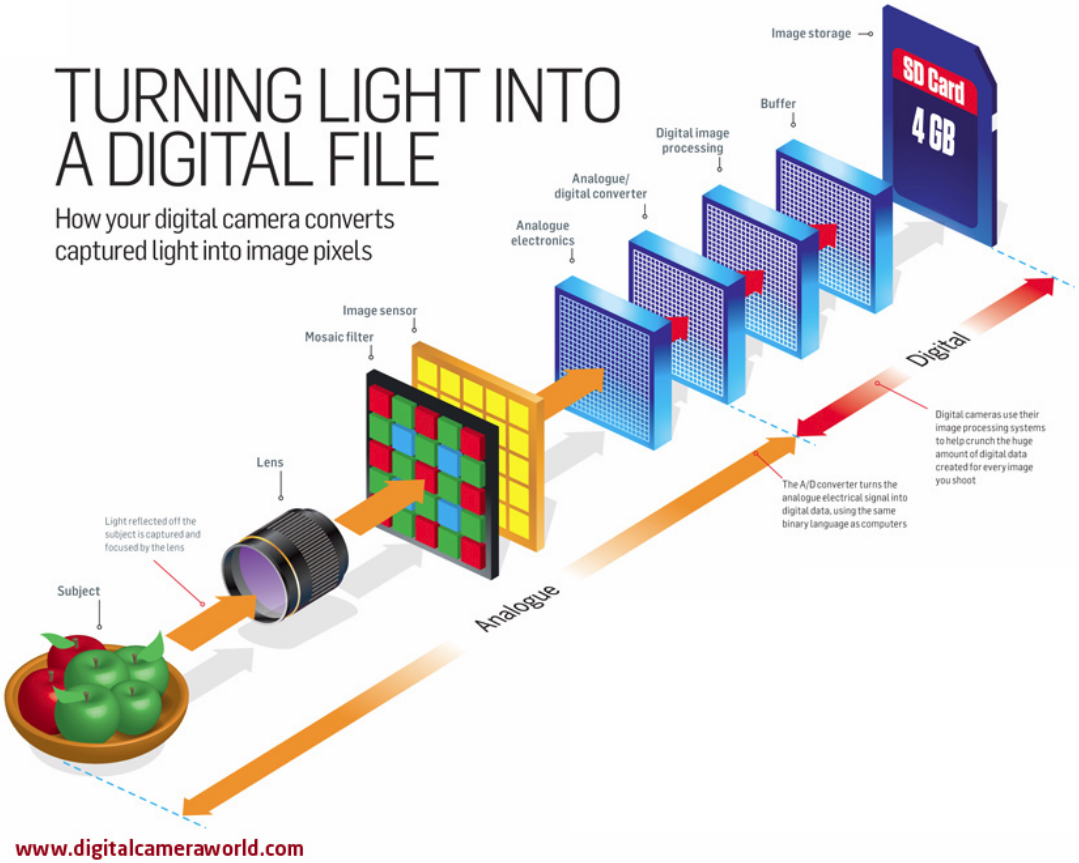
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \frac{f_x X + sY + c_x Z}{Z} \\ \frac{f_y Y + c_y Z}{Z} \\ 1 \end{bmatrix}$$

⇨ Fundamentally, perspective projection is a "divide by Z (depth)"

Nice reference (with a WebGL app): http://ksimek.github.io/2013/08/13/intrinsic/

From light to pixels

The light impinging on the image sensor is a continuous signal. This is converted to a *digital* signal through *sampling* and *quantisation*, to form a set of *pixels* comprising the image.



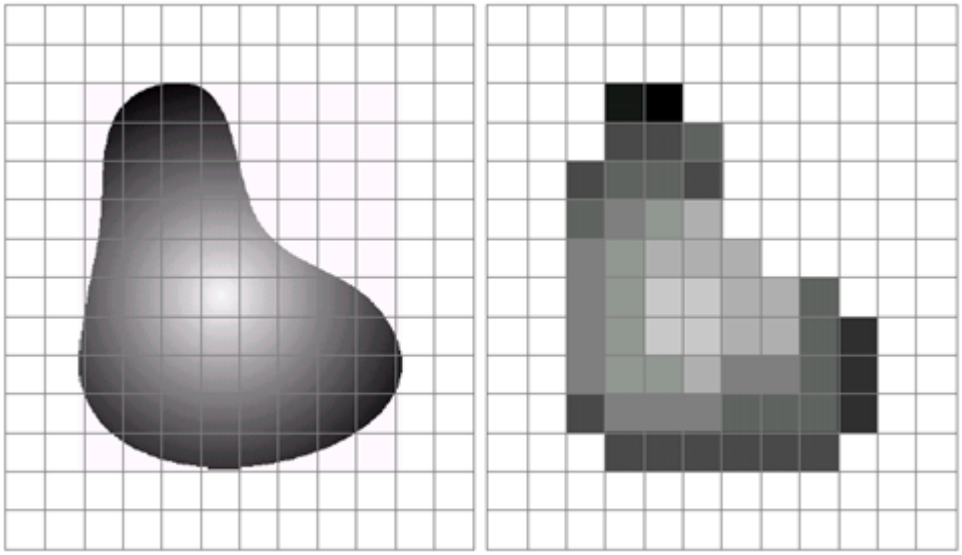
Digital image

A digital image is a collection of pixels, arranged on a 2D grid. Each pixel stores colour information comprising the image. The image *resolution* is the size of the image, in pixels (in width and height).

http://www.g2cs.org/media/interactives/image-compression/00.html

Sampling

Sampling converts a continuous signal into a discrete one. The light impinging the image plane is a continuous signal in *x* and *y*. In a digital camera, this signal is sampled on a regular grid. The sampling rate determines the resolution.



⇨ Should we always maximise resolution?

Well-known devices

iPhone
7, x

Rear **camera**
12 MP

Samsung Galaxy S7

Rear **camera**
12 MP

High Definition TV, HDTV

display
1920x1080 ~ 2 MP

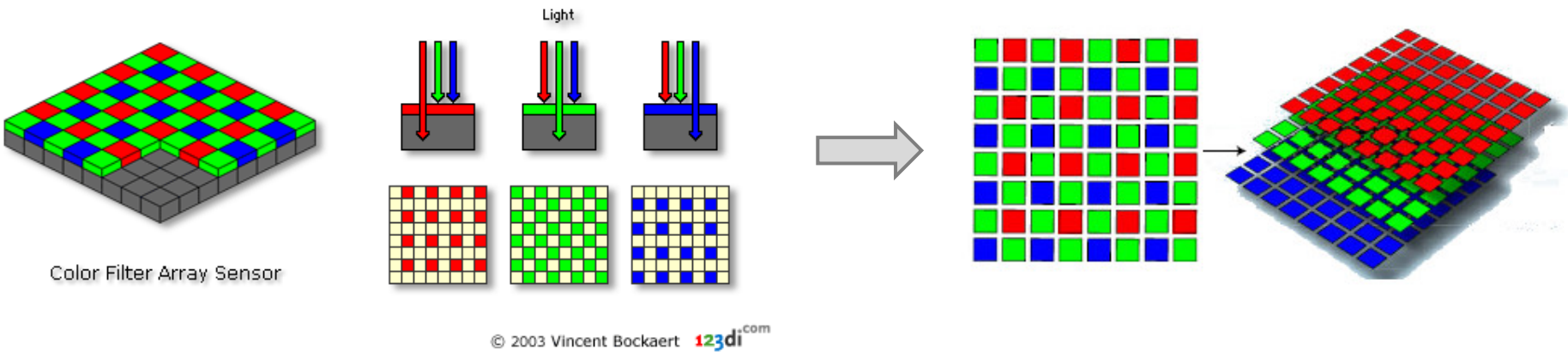
4K Ultra HD TV
3840x2160 = 8.3MP

Colour filter array and demosaicing

Light captured by digital cameras is usually binned into separate red, green, and blue values using a *colour filter array*, often made of a GRBG pattern called the Bayer pattern. A *demosaicing* algorithm interpolates the data so that each pixel has a red, green, and blue (RGB) value.

Mosaic

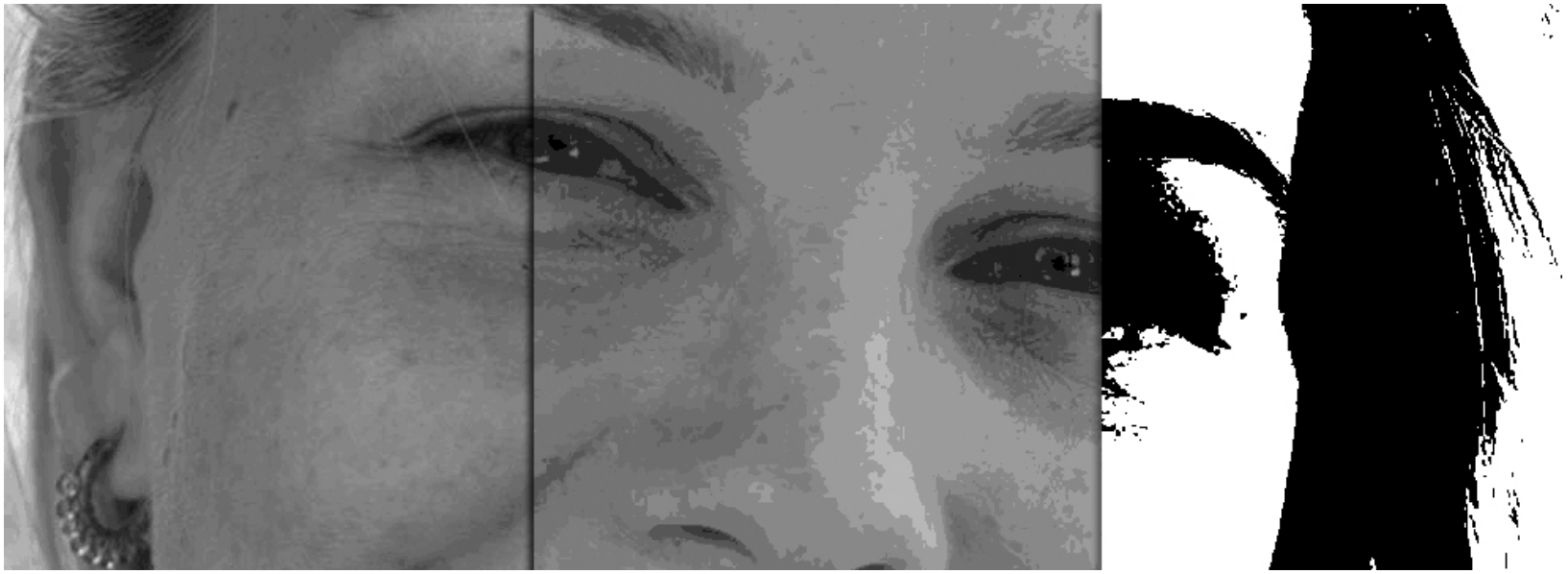
Demosaiced image



https://www.youtube.com/watch?v=2-stCNB8jT8

Quantisation

Quantisation limits the values a pixel can have to a finite set. For example, in a greyscale image, one normally uses an unsigned byte (8 bits) per pixel, so there are $2^8 = 256$ different intensities, normally represented in the range [0, 255].



8 bits per pixel
$2^8 = 256$ shades

4 bits per pixel
$2^4 = 16$ shades

1 bit per pixel
$2^1 = 2$ shades
binary image

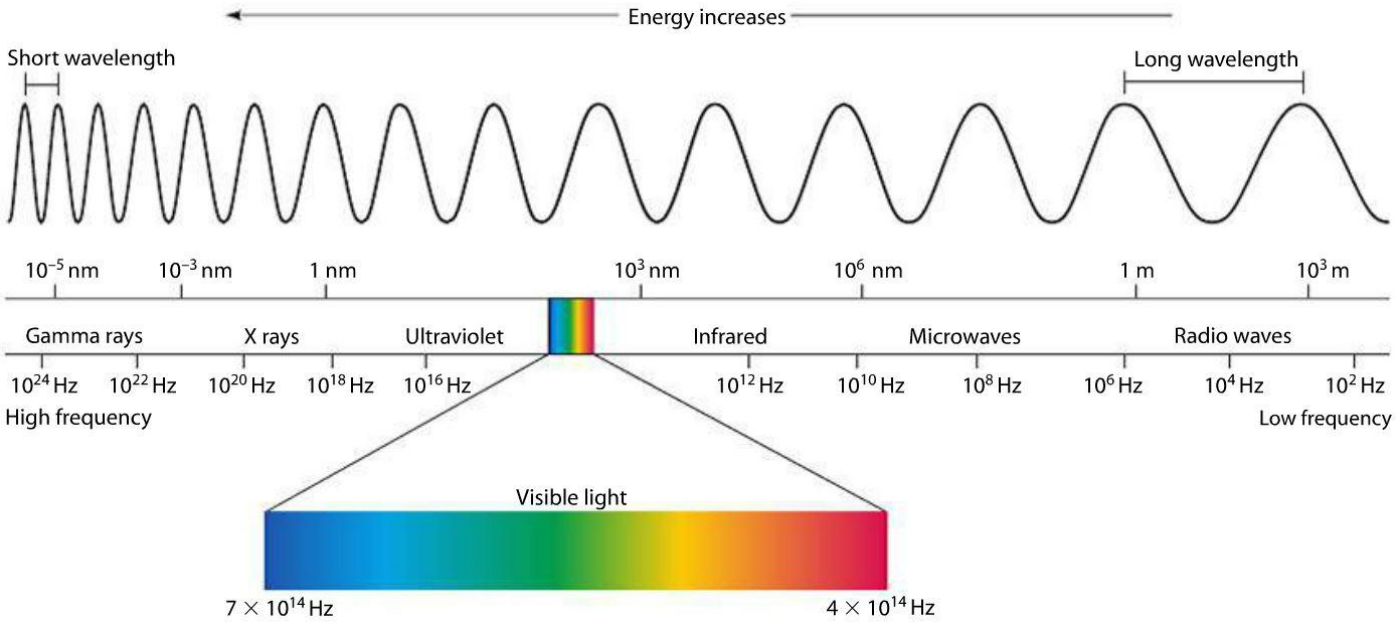
https://songbirdsings13.wordpress.com/year-12/new-media/unit-16/pmd1-unit-19/

Quantisation

In standard colour photographic images, one uses an 8 bits for *each* colour channel (red, green, blue), or 24 bits per pixel. That means there are 2^{24} possible colours for a pixel. This is roughly 16.7 million colours!

Colour

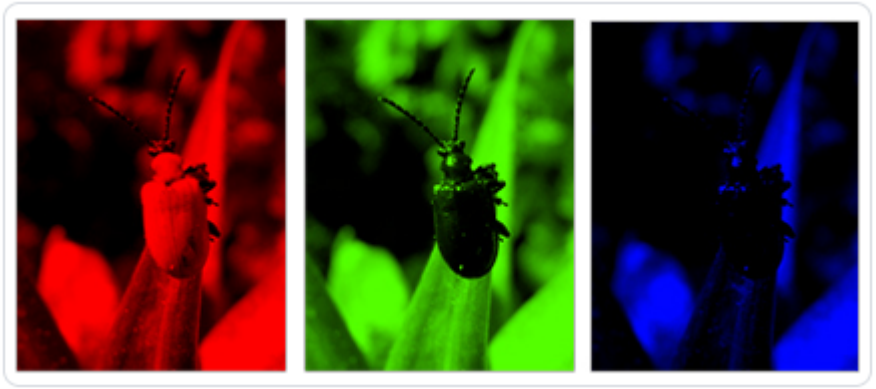
A narrow section of the electromagnetic spectrum is visible to the human eye.



http://classconnection.s3.amazonaws.com/255/flashcards/4013255/jpg/electromagneticspectrum-141B490BAC872789434.jpg

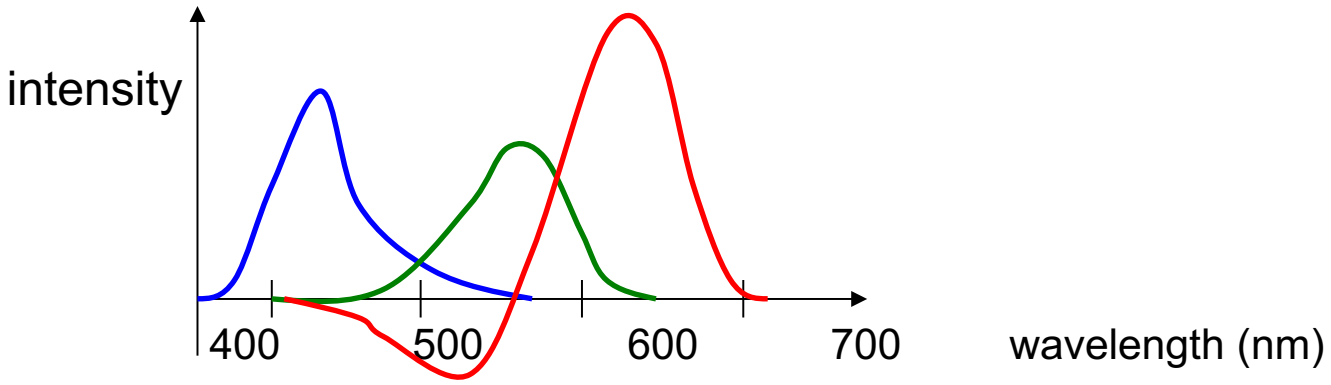
Colour

Colour images typically have three colour *channels* corresponding to the amount of red, green, and blue present at each pixel. Combining them together produces the final colour.



Colour Theory: Human Retina

- Tri-stimulus theory of vision:
  - Human eyes perceive colour through stimulation of three visual pigments in cones of retina, due to L, M, and S cones.
  - One pigment has peak sensitivity to red, the other green, the other blue.
- Below shows the intensity of the combination of “pure” R, G and B light that can appear to a human like each wavelength/frequency of actual light.

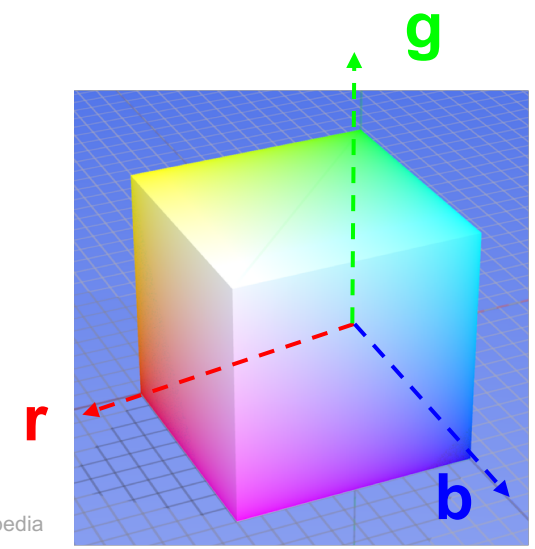


- RGB colour: three colours are related to the physiology of the human eye

$$\mathbf{C} = \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

RGB colour space

- Has its origin in colour television, now used in displays (flat panels, phones)
- *Additive* mixing or red, green, and blue light form the final colour
- RGB is a colour space
  - Red axis, green axis, blue axis
  - Values typically in the range from 0 (none) to 255 (full colour) along each axis
  - A colour is a point in this space, represented as a vector $[r, g, b]^T$

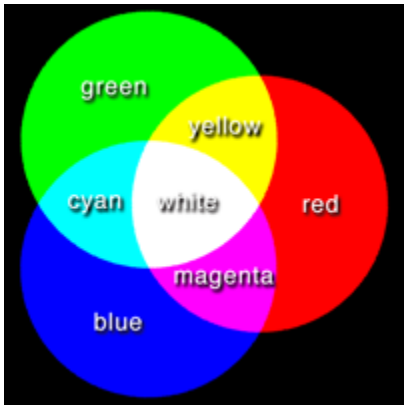


Source: wikipedia

RGB colour

RGB uses additive colour mixing, because it describes what kind of *light* needs to be *emitted* to produce a given colour. Light is added together to create form from out of the darkness.

Colour Component			Colour Common Name
R	G	B	
0	0	0	Black
0	0	255	Blue
0	255	0	Green
0	255	255	Cyan
255	0	0	Red
255	0	255	Magenta
255	255	0	Yellow
255	255	255	White

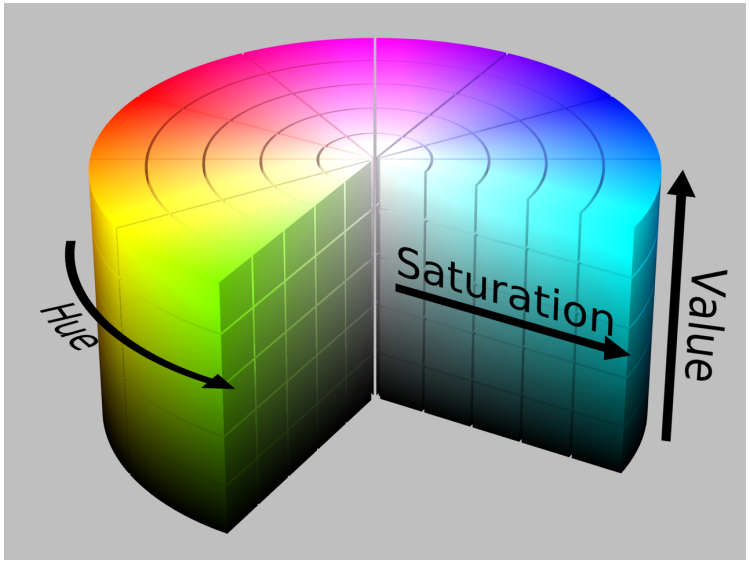


⇨ Of course, one can have values, like [255, 127, 0] -- orange

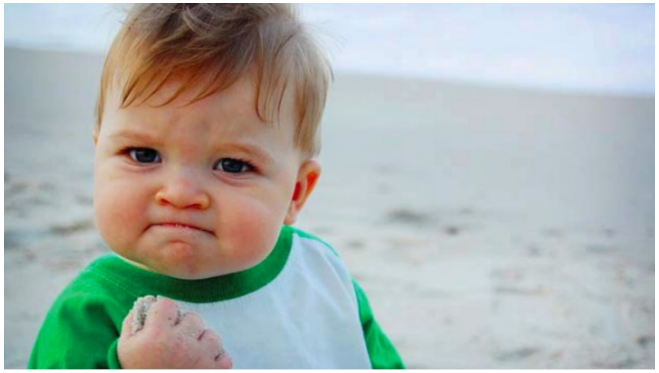
HSV

- Transforms the colour space to be more intuitive and perceptually relevant than RGB, using a cylindrical coordinate system.
  - H (hue) corresponds to the discernible colour based on the dominant wavelength (ROYGBIV)
  - S (saturation) corresponds to the vividness of the colour. Low saturation means a grayscale colour in the centre of the cylinder.
  - V (value) corresponds to brightness.

⇨ For example, a bright red colour has a red hue, full saturation, and high value.



HSV



Original image

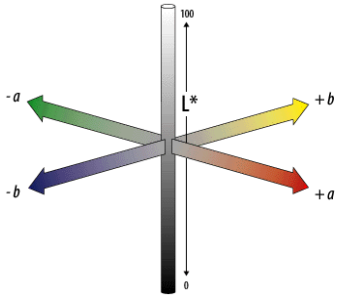
H

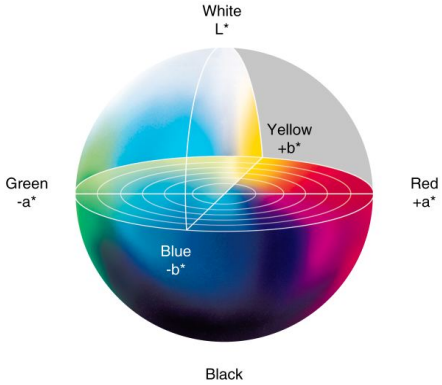
S

V

LAB

- RGB and HSV are not *perceptually uniform,* that is, the distance between colours in the colour space do not match human visual perception of colour difference
- CIE LAB attempts to be perceptually uniform
- It is an *opponent* colour system
  - Based on the observation that for humans, retinal colour stimuli are translated into distinctions between
    - Light and dark
    - Red and green
    - Blue and yellow
  - Non-linear transformation from RGB

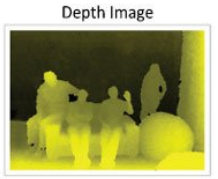




Other common image types

RGBD

- In addition to a colour image (RGB), acquires a depth image (D), which represents the distance from each pixel from the camera



Volumetric images

- Image data stored as voxels (small cubes, or 3D pixels). Examples include computed tomography (CT) or magnetic resonance (MRI)

Video

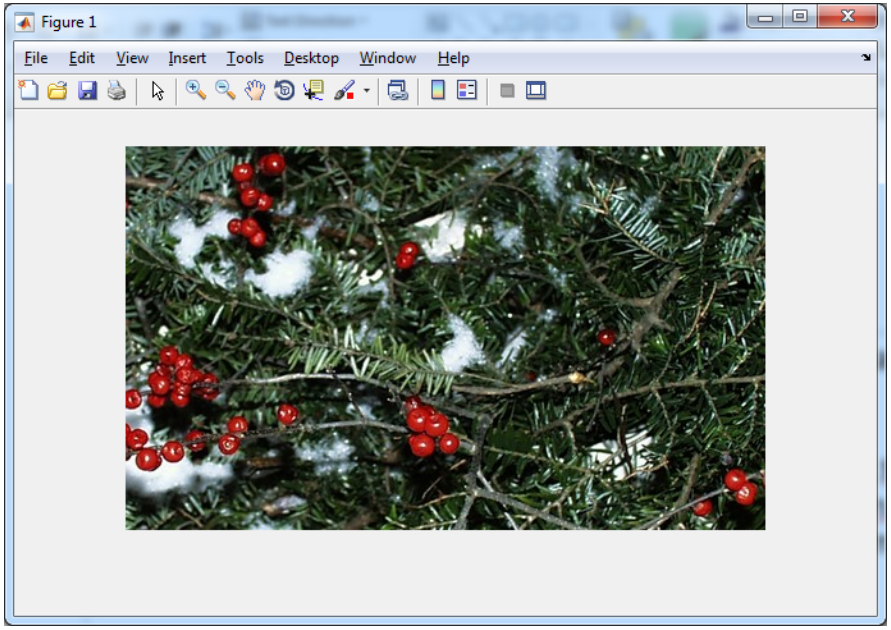
- A sequence of images over time

Images in Matlab

Matlab (aka **Mat**rix **Lab**oratory) is well suited for images, as it was originally designed to manipulate 2D arrays (matrices).

Matlab can load images of a variety of common formats (.tif, .jpg, .bmp, .png). It also comes with some images built-in (e.g., 'cameraman.tif', 'coins.png'). The command to load an image is `imread`. To show an image, use the command `imshow`.

```
I = imread('greens.jpg');
imshow(I);
```



Images in Matlab

In the Workspace, you'll see *I* is a variable of size 300 x 500 x 3. This means the image has a height of 300, width of 500, and three colour channesl (corresponding to red, green, and blue).

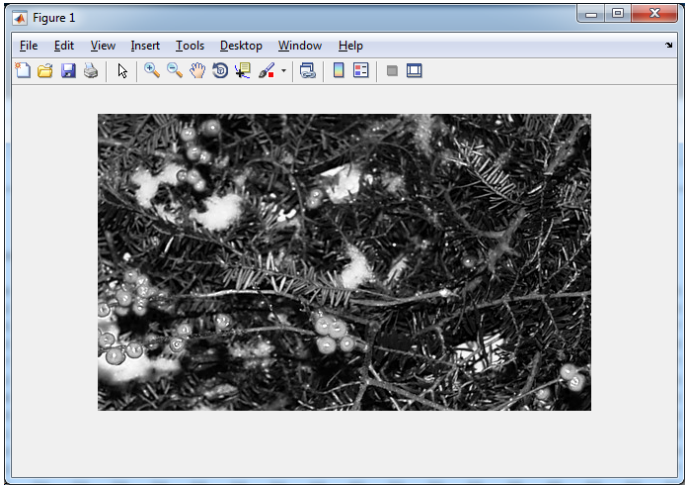
It's type is uint8, meaning each value in each colour channel is an unsigned integer with 8 bits of precision. This means there are $2^8 = 256$ values possible, in the range [0, 255].

To look at the red channel, show only the first colour channel

red channel

```
imshow(I(:, :, 1));
```

all rows all columns

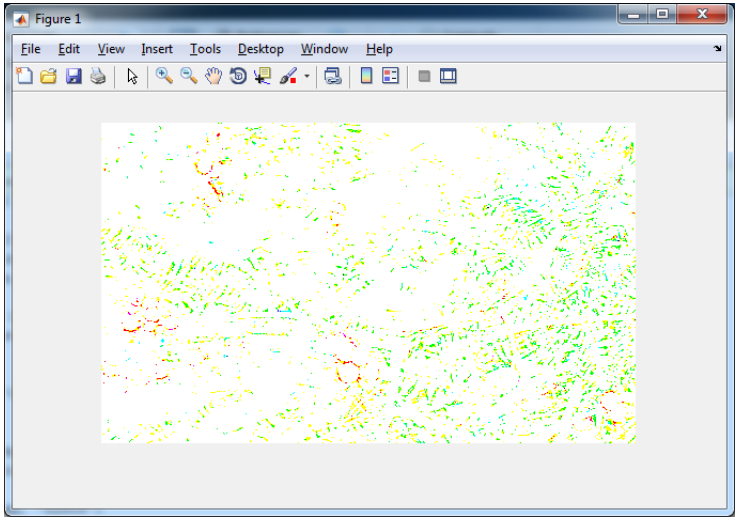


uint8 and double

Images are typically uint8 when loaded. However, sometimes it may be convenient to convert them to double. Matlab supports type conversions:

```
I = imread('greens.jpg');
J = double(I);
```

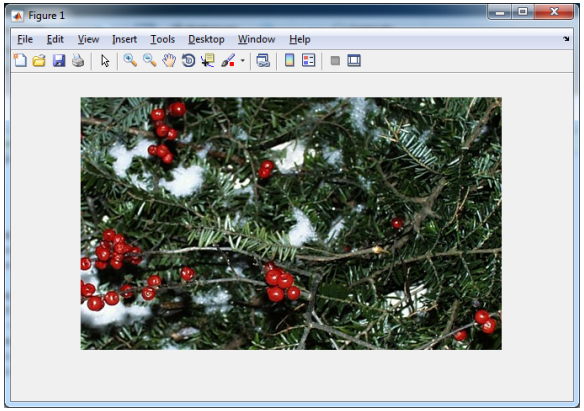
The values of `I` (going from 0 to 255 in each colour channel) are converted to double. However if you call imshow on `J`, a funny image is shown, since for double data, Matlab expects values between 0 and 1 (everything else *clipped*).




```
imshow(J);
```

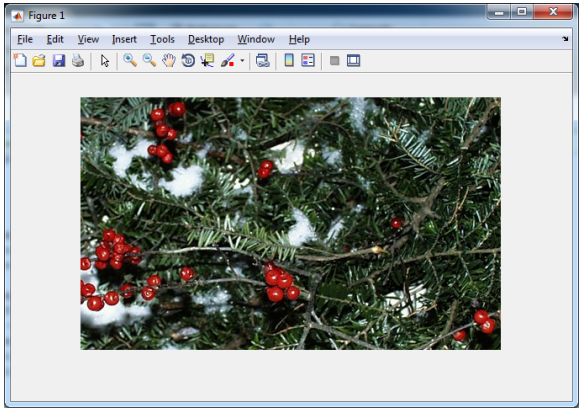
Visualizing double data

This can be resolved by visualising the data divided by 255.




```
I = imread('greens.jpg');
J = double(I);
imshow(J/255);
```

The function `im2double(I)` does the division for you.




```
I = imread('greens.jpg');
J = im2double(I);
imshow(J);
```

Colour space conversions: greyscale

In Matlab, it is easy to convert from one colour space to another.

Conversion from RGB to greyscale can be done with `rgb2gray`

```
I = imread('Holiday.png');
imshow(I);
I1 = rgb2gray(I);
imshow(I1);
```

Other colour space conversions

Conversion from RGB to HSV can be done with `rgb2hsv`

```
I = imread('Holiday.png');
imshow(I);
J = rgb2hsv(I);
imshow(J(:,:,1)); % H
figure;
imshow(J(:,:,2)); % S
figure;
imshow(J(:,:,3)); % V
```

H

S

V

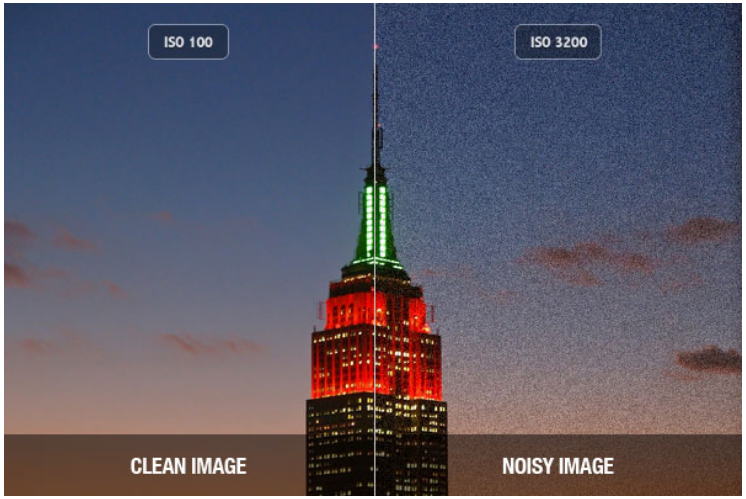
and back with `hsv2rgb`

```
K = hsv2rgb(J);
```

There is also `rgb2lab`, and `lab2rgb`.

Imperfections in imaging

Limited resolution



Noise

Bloom

Imperfections in imaging

Motion blur

Poor contrast

Compression artefacts

Lens distortion

CITY
UNIVERSITY OF LONDON
EST 1894

Digital image processing

Digital image processing is the use of computer algorithms to transform an image. Example transformations include

- Colour transformations
- Geometric transformations
- Filtering (*next lecture*)

Brightness transformations

Brightness transformations are *position-independent* and take the form

$$J = f(I)$$

where

$I(x, y)$ is the original image,
$J(x, y)$ is the transformed image
and f is a function that transforms colour I to colour J.

The function f is independent of position in the image, and therefore the (x, y) arguments are dropped in the equation $J = f(I)$ above; f operates on the pixel values.

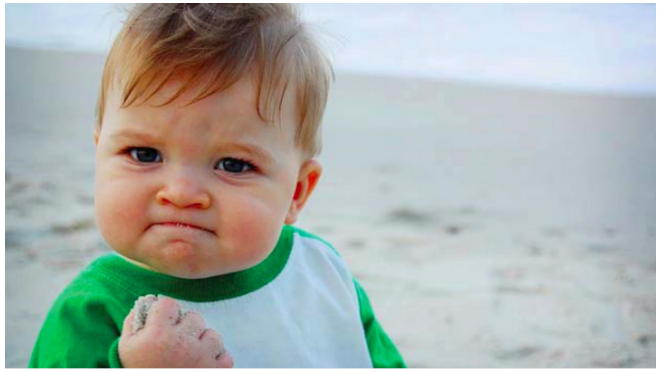
Negative

Recall a standard 8 bit image has intensities in the range [0, 255] in each colour channel. The negative of the image can be formed simply as

$$J = 255 - I$$

In Matlab, this can be implemented as

```
I = imread('success-Kid.jpg');
figure; imshow(I);
J = 255 - I;
figure; imshow(J);
```



Original image

Negative

Tinting

Tinting (and colour balancing) applies an adjustment to the colours, normally as a multiplication.

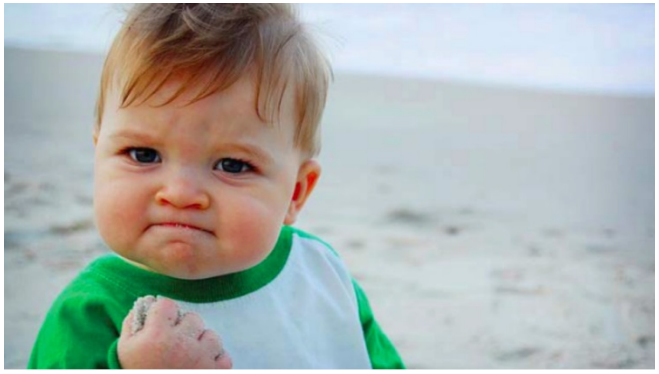
$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} s_r & 0 & 0 \\ 0 & s_g & 0 \\ 0 & 0 & s_b \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} s_r r \\ s_g g \\ s_b b \end{bmatrix}$$

Here, s_r, s_g, and s_b scale the red, green, and blue colour of each pixel.

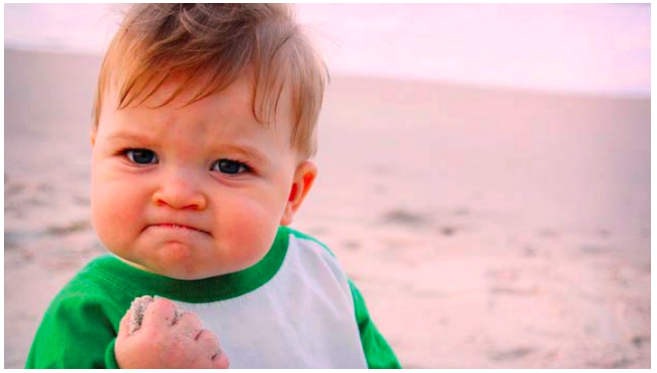
Example

Example: s_r = 1.25, s_g = 1, s_b = 1

```
I = imread('success-Kid.jpg');
R = I(:, :, 1);
G = I(:, :, 2);
B = I(:, :, 3);
s_r = 1.25; s_g = 1; s_b = 1;
J(:, :, 1) = s_r * R;
J(:, :, 2) = s_g * G;
J(:, :, 3) = s_b * B;
figure; imshow(J);
```



Original image

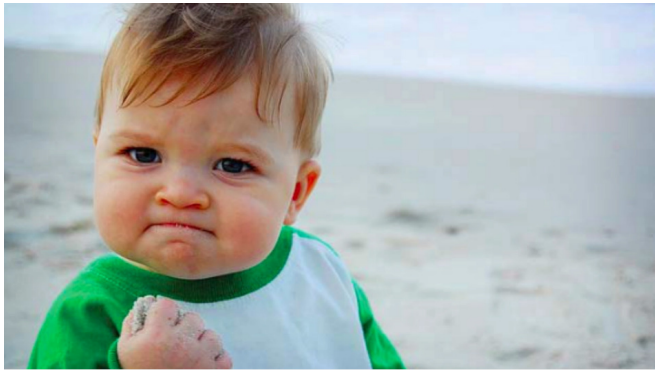


Transformed image

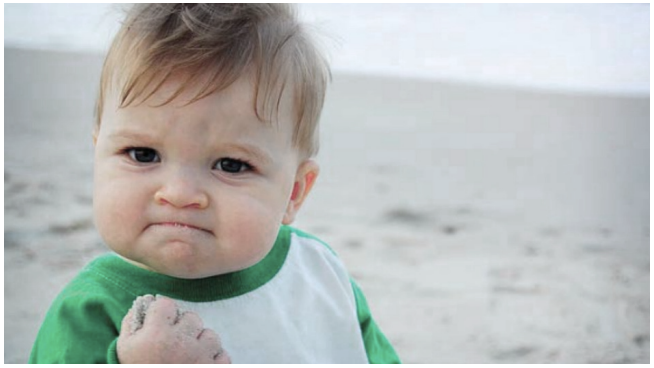
Transformations in HSV

Of course you can do this in other colour spaces. Here is an example of decreasing saturation.

```
I = imread('success-Kid.jpg');
HSV = rgb2hsv(I);
H = HSV(:, :, 1);
S = HSV(:, :, 2);
V = HSV(:, :, 3);
s_h = 1; s_s = 0.5; s_v = 1;
J(:, :, 1) = s_h * H;
J(:, :, 2) = s_s * S;
J(:, :, 3) = s_v * V;
J = hsv2rgb(J);
figure; imshow(J);
```



Original image



Transformed image

⇨ How might you darken the image?

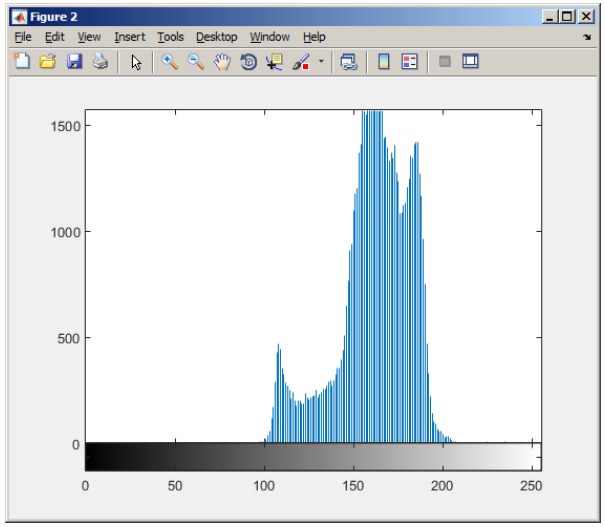
Histogram

A histogram gives a *count* of pixel values in an image. One can compute histograms in each colour channel. However, for simplicity, let's consider greyscale images now. The function `imhist` will compute the histogram.

```
I = imread('cars.png');
figure;
imshow(I);
figure;
h = imhist(I, 256);
```

256 bins

Image

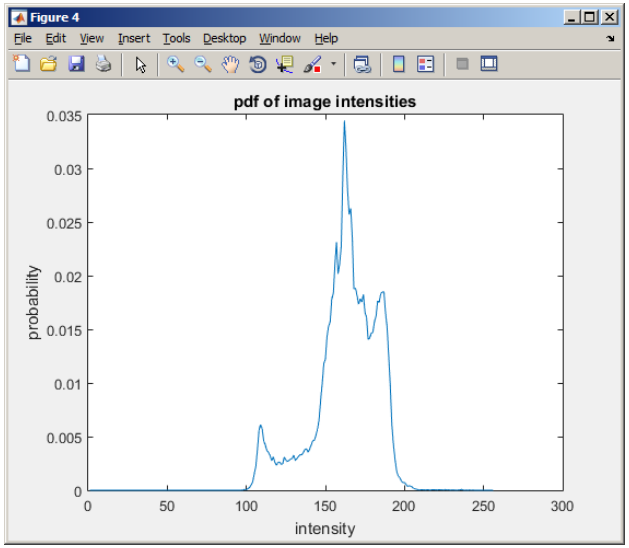


Histogram

Histogram as a pdf

A histogram can be normalised and interpreted as a probability density function (pdf), representing the probability of a pixel having a particular brightness.

```
pdf = h / sum(h);
figure;
plot(pdf);
title('pdf of image intensities');
xlabel('intensity');
ylabel('probability');
```



PDF

Here, we could say in this image, a pixel has a higher probability of having an intensity of 160 than an intensity of 200.

Contrast stretching

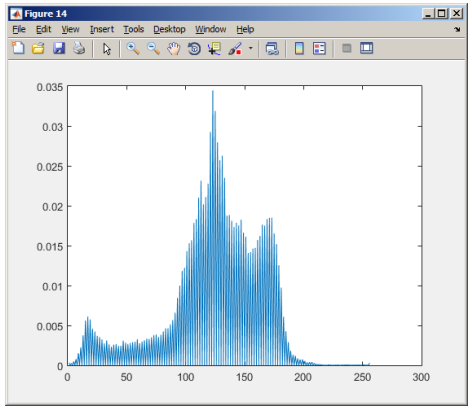
Looking at this histogram, you can see that most of the intensities are clustered in a range of [100, 200]. This is why the image looks “washed out” as the image is lacking darker values (<100) and brighter values (>200).

Similar to tinting, we can transform the values, this time using a function

$$J = \alpha I + \beta$$

For example, with $\alpha = 2$ and $\beta = -200$, we spread the intensities to achieve

```
alpha = 2;
beta = -200;
J = uint8(alpha * double(I) + beta);
figure; imshow(J);
```



Gamma correction

Gamma correction applies a non-linear transformation of pixel values. It tries to take advantage of the non-linear manner in which humans perceive colour, with greater sensitivity to differences between darker tones than lighter ones.

The mathematical form is

$$J = A\, I^{\gamma}$$

Where A is a constant and γ is parameter.
Normally $A = 255^{1-\gamma}$ for an image in the range [0, 255]

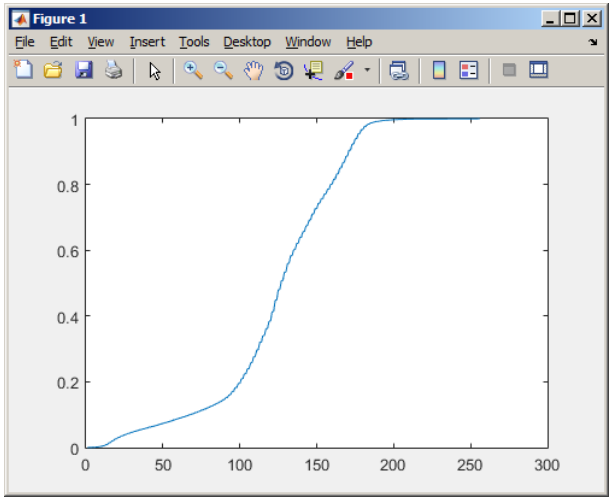
```
gamma = 2;
J = 255^(1-gamma)*double(I).^gamma;
figure;
imshow(uint8(J));
```

Histogram equalisation

Histogram equalisation tries to achieve a flat histogram. This way all image intensities are used in equal amounts.

This uses the cumulative histogram, computed as

```
h = imhist(I, 256);
[rows, cols] = size(I);
cdf = cumsum(h) / (rows*cols);
plot(cdf);
```



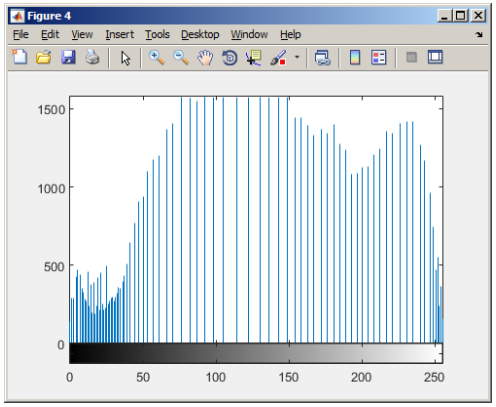
https://uk.mathworks.com/help/images/ref/adapthisteq.html

Histogram equalisation

With the cdf in place, we can transform the image as

```
J = uint8(255*cdf(I+1));

figure;
imshow(J);
figure;
imhist(J, 256);
```

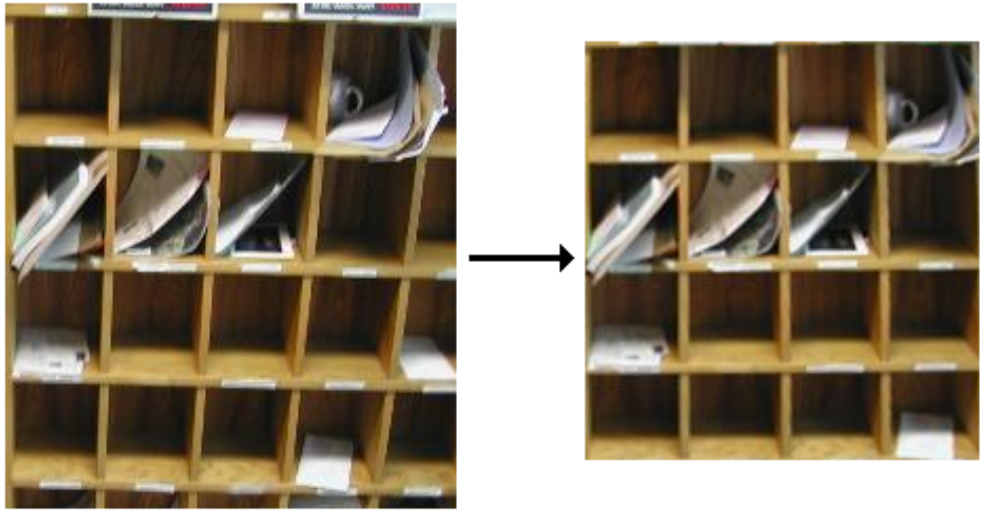
Note: Matlab has a function `histeq` that you can use as well.



Geometric transformations

Geometric transformations change the spatial position of pixels in the image. They are also known as *image warps*. Geometric transformations have a variety of practical uses, including

- Registration: bringing multiple images into the same coordinate system
- Removing distortion
- Simplifying further processing



Distorted image

Corrected image

Geometric transformations

In a geometric transformation, the positions of pixels in the image are transformed. Mathematically, this is expressed (in a general form) as

$$\mathbf{x}' = \mathbf{T}(\mathbf{x})$$

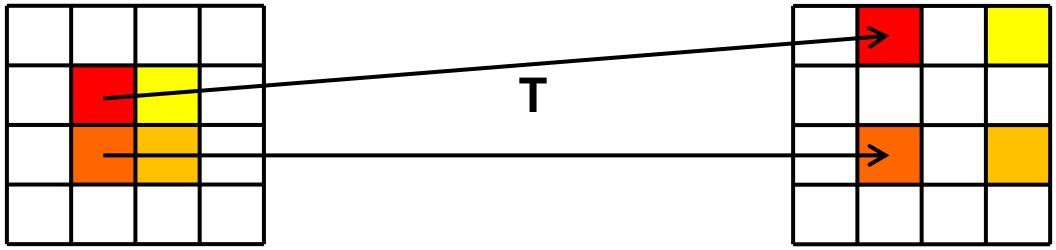
where **x** is the position of the pixel in the distorted image
x' is the position of the pixel in the corrected image
T(**x**) is a function that maps **x** to **x**'

An image warp (warping image `I` to produce `J`) is normally implemented as follows:

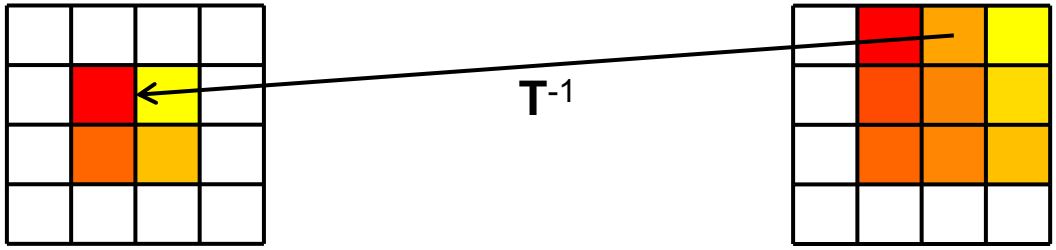
for every pixel position **x'** in the corrected image:
  Using $\mathbf{T}^{-1}$, determine **x**, where **x'** came from in the distorted image
  Interpolate a value from `I`(**x**) to produce `J`(**x'**)
end

Geometric transformations

You may notice this is applied somewhat *backwards*, rather than using a (forward) mapping **T** to transform pixels from the distorted image to the corrected image, we use an (inverse) transform $\mathbf{T}^{-1}$.



Forward mapping may result in gaps



Inverse mapping ensures no gaps

⇨ Using an inverse mapping ensures all the pixels in the corrected image will be filled. However, it's necessary to *interpolate* pixels from the distorted image.

Affine transformation

An affine transformation takes the form $\mathbf{x}' = A\mathbf{x}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

In Matlab, you can make an **affine transformation** using

```
A = affine2d([a11 a12 a13; a21 a22 a23; 0 0 1]');
```

and apply the transformation to an image using `imwarp`

Matlab wants a transposed input

Special cases

There are several common special cases, including

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Translation

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

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix}$$

Scaling

Since these are common transformations, there are special Matlab functions for them, `imtranslate`, `imrotate`, and `imresize`.

Examples of using imtranslate, imrotate, and imresize

```
I = imread('Holiday.png');
imshow(I);
```

```
J = imtranslate(I, [100, 0]);
imshow(J);
```

```
J = imrotate(I, 45);
imshow(J);
```

```
J = imresize(I, 0.5);
imshow(J);
```

Affine transformation

Another special case includes *skew*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \tan\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + y\tan\theta \\ y \\ 1 \end{bmatrix}$$

or

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + sy \\ y \\ 1 \end{bmatrix}$$

Example

```
I = imread('LicensePlate.png');
figure; imshow(I);
```


```
% Rotate clockwise 15 degrees to align base
J1 = imrotate(I, -15, 'bilinear');
figure; imshow(J1);
```

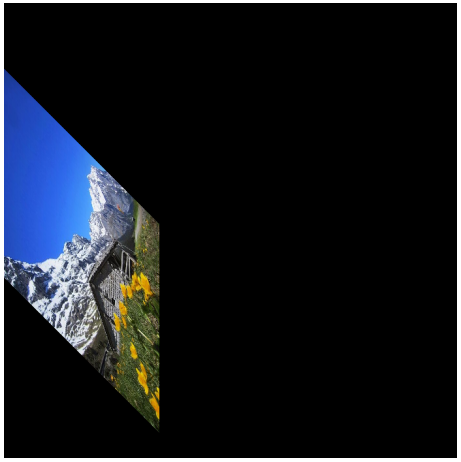

```
% Now apply a skew
tform = affine2d([1 .3 0; 0 1 0; 0 0 1]');
J2 = imwarp(J1, tform);
figure; imshow(J2);
```

Exercise: Match code to transformed image

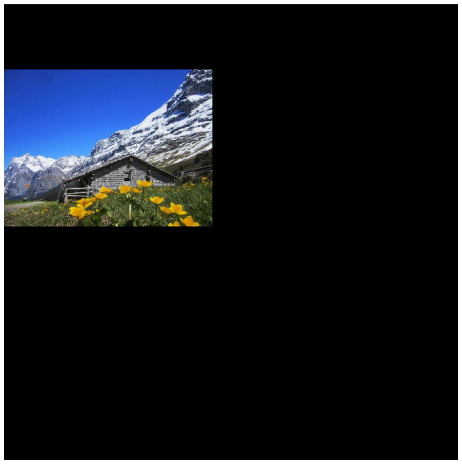
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

```
R = imref2d([700,700]);

% Transformation 1
A = affine2d([.5 0 0; 0 .5 100; 0 0 1]');
[J, RJ] = imwarp(I, A, 'OutputView', R);
imshow(J, RJ);

% Transformation 2
A = affine2d([.5 0 100; 0 .5 0; 0 0 1]');
[J, RJ] = imwarp(I, A, 'OutputView', R);
imshow(J, RJ);
```

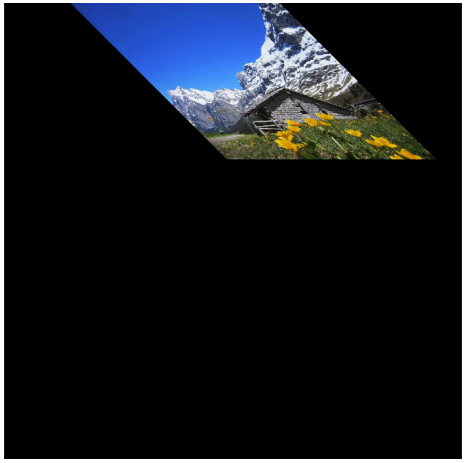
```
% Transformation 3
A = affine2d([.5 .5 100; 0 .5 0; 0 0 1]');
[J, RJ] = imwarp(I, A, 'OutputView', R);
imshow(J, RJ);

% Transformation 4
A = affine2d([0 .5 0; .5 .5 100; 0 0 1]');
[J, RJ] = imwarp(I, A, 'OutputView', R);
imshow(J, RJ);
```



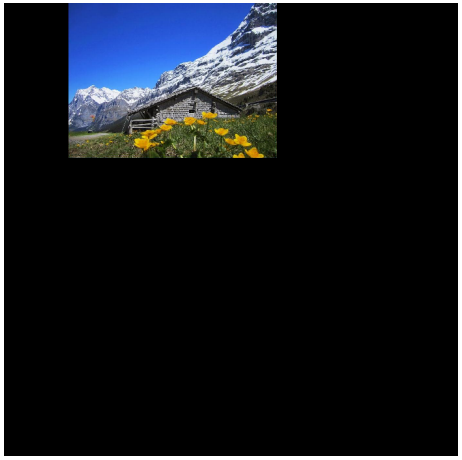
A



B



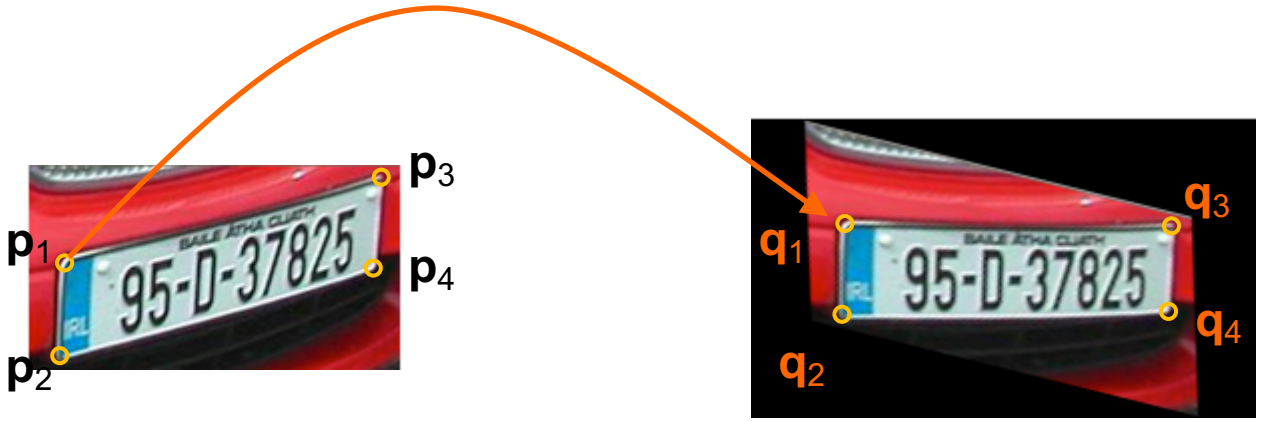
C



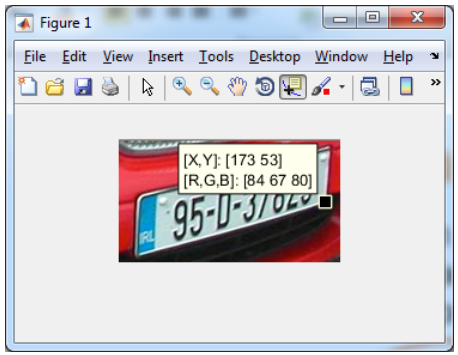
D

Estimation through correspondences

If the affine transformation is not known in advance, it can be estimated. The goal is to determine the six coefficients in the *A* matrix. This can be achieved by finding at least **three** *correspondences*, or matching points.



For example, we could say $\mathbf{p}_1$ corresponds to $\mathbf{q}_1$. What we would like to do is estimate the affine transformation that best aligns the correspondences.



$\mathbf{p}_1 = [18, 47]^T$
$\mathbf{p}_2 = [15, 100]^T$
$\mathbf{p}_3 = [178, 6]^T$
$\mathbf{p}_4 = [173, 53]^T$

$\mathbf{q}_1 = [48, 50]^T$
$\mathbf{q}_2 = [48, 100]^T$
$\mathbf{q}_3 = [212, 50]^T$
$\mathbf{q}_4 = [212, 100]^T$

Estimation through correspondences

Noting that

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

If we have three correspondences we can write

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}$$

Or $q = Ma$ which can be solved as $a = M^{-1}q$

This gives the coefficients needed for the affine transformation. What can you do if you have more than three correspondences?

Use the pseudo-inverse instead of the inverse!

Matlab code

```
% Estimate affine transformation through correspondences
I = imread('LicensePlate.png');
p1 = [18, 47]; p2 = [15, 100]; p3 = [178, 6];
q1 = [48, 50]; q2 = [48, 100]; q3 = [212, 50];

q = [q1(1), q1(2), q2(1), q2(2), q3(1), q3(2)]';
M = [p1(1), p1(2), 1, 0, 0, 0; 0 0 0 p1(1), p1(2), 1; ...
    p2(1), p2(2), 1, 0, 0, 0; 0 0 0 p2(1), p2(2), 1; ...
    p3(1), p3(2), 1, 0, 0, 0; 0 0 0 p3(1), p3(2), 1];
a = inv(M)*q;
tform = affine2d([a(1) a(2) a(3); a(4) a(5) a(6); 0 0 1]');

% Now apply the warp to the image
J = imwarp(I, tform);
figure; imshow(J);
```

estimateGeometricTransform

Matlab has a function, estimateGeometricTransform that simplifies estimation of (similarity, affine, and projective) geometric transformations.

```
tform = estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType)
```

```
% Estimate affine transformation through correspondences
I = imread('LicensePlate.png');
p1 = [18, 47]; p2 = [15, 100]; p3 = [178, 6];
q1 = [48, 50]; q2 = [48, 100]; q3 = [212, 50];

P = [p1; p2; p3];
Q = [q1; q2; q3];
tform = estimateGeometricTransform(P, Q, 'affine');

% Now apply the warp to the image
J = imwarp(I, tform);
figure; imshow(J);
```

Projective transformation

Images normally acquired by photographic cameras are formed by perspective projection, as described earlier. If we view a planar surface not parallel to the image plane, then an affine transformation will not map the shape to a rectangle.

Instead, we must use a ***projective* transformation** of the form

$$\begin{bmatrix} x'w \\ y'w \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11}x + p_{12}y + p_{13} \\ p_{21}x + p_{22}y + p_{23} \\ p_{31}x + p_{32}y + 1 \end{bmatrix}$$

To estimate a projective transformation, at least **four** 2D correspondences are needed (due to the eight unknowns, also called degrees of freedom (DOF)).

Estimation through correspondences

In a similar way as in the affine case, we can write

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \end{bmatrix}$$

Or $q = Mp$ which can be solved as $p = M^{-1}q$

This gives the coefficients needed for the transformation. In Matlab, one can use the `estimateGeometricTransform` function with `'projective'` as the third argument to solve the equation shown above.

Projective transformation

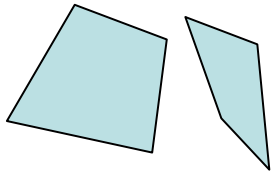
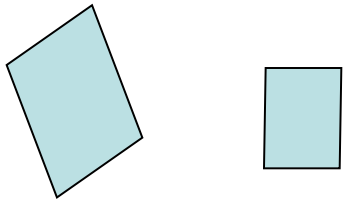
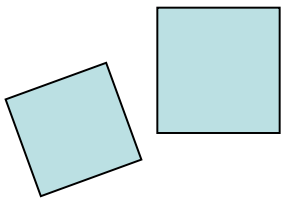
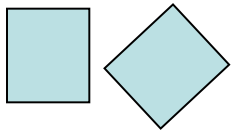
Example: Find the best fitting warp to transform the game area to a given rectangle

original

affine

projective

Hierarchy of 2D transformations

		Transformed squares	Preserves
Projective 8 DOF	$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix}$		Collinearity
Affine 6 DOF	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallellism of lines
Similarity 4 DOF	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratios of lengths, angles
Rigid-body 3 DOF	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Lengths, areas

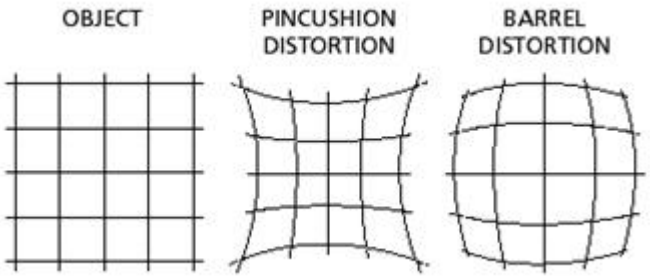
Slide source: Marc Pollefeys

Non-linear transformations

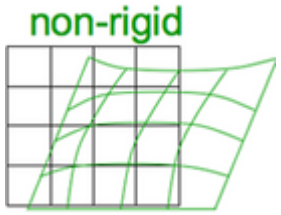
Other transformations are possible, including those that do not involve a matrix, and instead a more general function that transforms pixel locations. What's needed is a way to describe the transformation

$$\mathbf{x}' = \mathbf{T}(\mathbf{x})$$

Examples of other common transformations include



Radial lens distortion



Non-rigid transformation

Coursework

The coursework has been released

- UG students: Wednesday 1st May 2019
- PG students: Wednesday 1st May 2019

Task(s): *Face detection and recognition;*

- *UG students: creative mode*
- *PG students: OCR in images and videos*

CITY
UNIVERSITY OF LONDON
EST 1894

Next session

- Filtering Images
- Feature extraction
- Convolution
- Edge detection
- Corner detection