Tutorial 8 - Hough Transform and Colour

Gidon Rosalki

2025-12-10

1 Hough Transform

The Hough Transform is a method of finding parametric shapes in an image. Parametric shapes are things like straight lines, rectangles, circles, and so on. More generally, it is a voting scheme on shape parameters. It is slightly similar to RANSAC.
The applications include finding signs in an image (through detecting rectangles and circles), 3D reconstruction of buildings, the eye in a face, the ball in a scene, and so on. The challenge is that the image usually contains lots of irrelevant information (edges unrelated to the shape or due to noise), and partial occlusions.

One can differentiate the image, set a threshold, and then the remaining pixels are part of the edges. Given an edge image, we would like to group sets of edge pixels to identify dominant lines in an image. One possibility is to perform template matching in the edge image. Here we generate a hypothetical line "kernel", and convolve with the image. This requires a large set of masks to accurately localize lines, which is computationally expensive. We will instead perform an edge image transformation where edge pixels vote for possible lines. This eventually reduces the problem to vote thresholding.

Recall, the **parameter space** is a space of a, b. So each line in the image, is a point in the parameter space, since a line is composed of $y = ax + b$, so the (a, b) are the coordinates of the point in parameter space. In contrast, a line in parameter space, is a **point** in the image. We can transfer sides, and get

$$y = ax + b \tag{1}$$

However, remember for a point, we know x, y so we may:

$$b = -xa + y$$

So for a point (x, y) there are an infinite number of lines $b = -xa + y$ passing through it.

1.1 Use a voting scheme

We will apply an *edge detector* on the image, either through the derivative with a threshold, or using Canny's method. We will then prepare a table $h(a, b)$, for a, b in a certain range. We will then loop over the image, for each pixel (x, y) on an edge *vote* for all the cells (a, b) which satisfy the exation $y = ax + b$. This means that we increase the cell counter $h(a, b)$. We then choose the cell which has the maximum value (or the cells which pass a given threshold).

1.2 Example

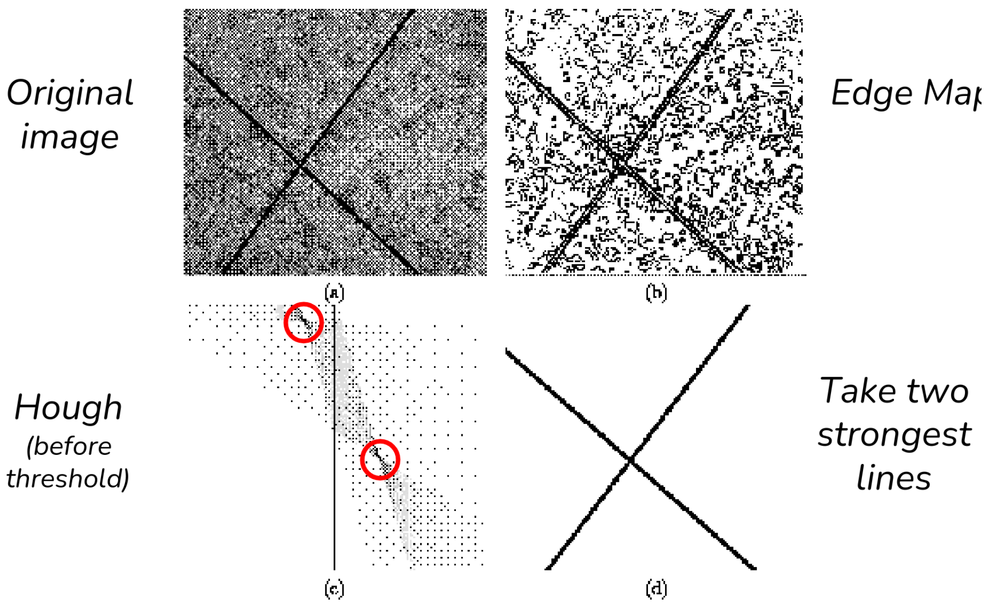

Figure 1: Example: Slope intercept representation

However, this representation has some issues. Vertical lines are not covered by the representation $y = ax + b$. Additionally, there is no way to select the range of values for the table. There are infinite values for a, b that may be within the image, so the table must be prepared to handle these infinite values. Additionally, we cannot quantise a, b. They are both values in $\mathbb{R}$.

1.3 A better representation

We can instead use polar representation, where:

$$d = x\cos(\theta) + y\sin(\theta)$$

Here, d is the distance from the origin $(0, 0)$, and θ is the angle relative to the x axis. So now, a point in the image will be translated to a sinusoidal function in $r - \theta$ space. Points of intersections of these sinusoidal functions will estimate the line equation.

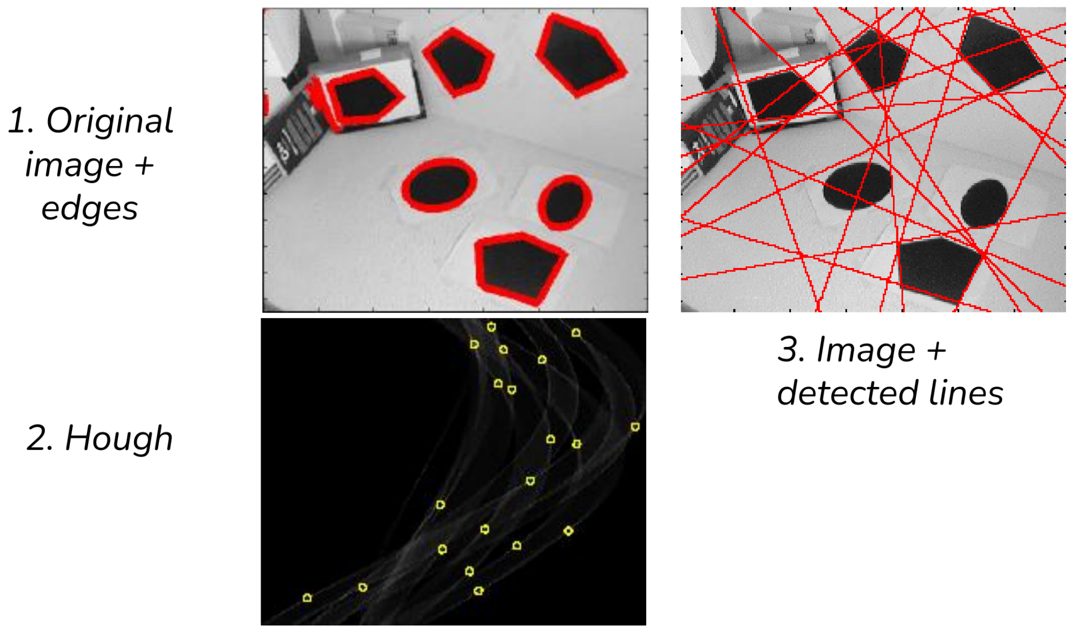

Figure 2: Hough transform with polar representation

We can note immediate improvements, this can handle the vertical lines that could not be handled, we can limit the values to specific ranges (r is in $[0, 359]$, d is less clear, so it is unlikely to appear on an exam, but could just be the size of the image), and we can quantise them to $\mathbb{N}$.

1.4 Smoothing

Without smoothing then a point will refer to a line, but when we smooth, then the point is expanded in size, and thus the line is expanded into a valley:

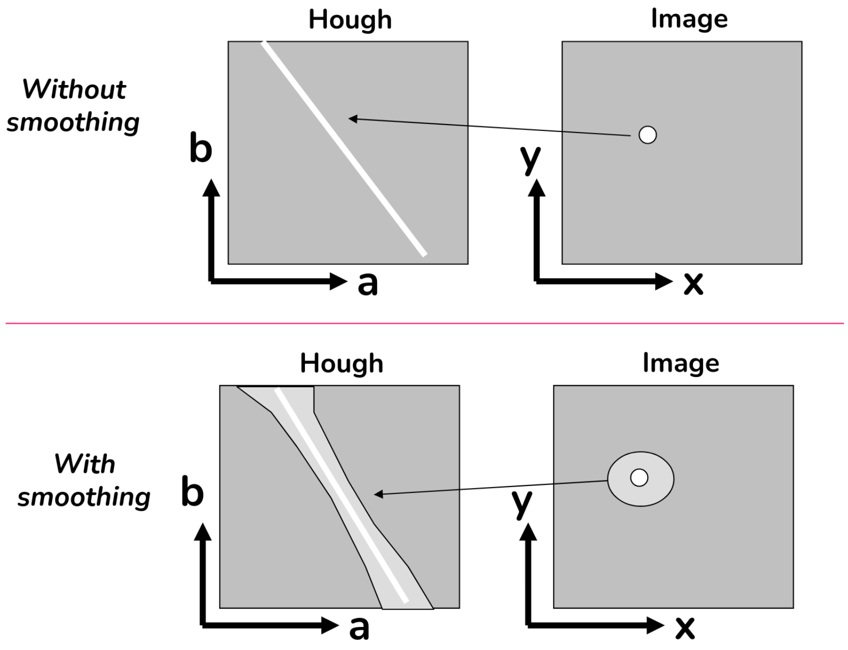

Figure 3: Smoothing

This helps resolve the issue of noise in the image, since it removes individual pixel noise lines.

1.5 Making a decision

So now we want to find the two dominant lines in the image. The problem is that if (r, θ) is the cell with the maximal value, then $(r + \varepsilon, \theta + \varepsilon)$ will also get high scores. To resolve this we only search for local maxima.

We can also add in the local gradient. If a line is the transition between 2 colours, then taking a point on this line has the gradient that is *perpendicular* to the line of the transition. Therefore, when adding in the local gradient, a point on the line will also point to a point in the Hough transform. Taking all the points of this line will, thanks to noise, point to a very tight cluster of points in the Hough transform. This reduces the problem of infinite lines, since we know from the tight cluster which (approximate) line we want.

1.6 Complex shapes

1.6.1 Circles

Perhaps we also want to detect circles in the image, which are not just constructed of straight lines (I mean, that is quite literally their definition). A naïve solution would be to construct a table $h\left(a, b, r\right)$, and for each pixel (x, y) which is an edge in the image, vote for all the cells satisfying

$$(x - a)^2 + (y - b)^2 = r^2$$

However, the size of the table is $O\left(n^3\right)$, and the voting for each pixel takes $O\left(n^2\right)$. The solution is to use the gradient information, so you vote only for centres (a, b) in the gradient direction. We could also use a more advanced solution, and vote with complex numbers.

With no orientation, each token (point) votes for all possible circles that pass through that point (infinite). With orientation, then each token can vote for a smaller number of circles. Realistically, only 2, since we have limited the size of the circle, and stated the direction of the circle at that point, so it is tangent to only 2 circles.

Consider this example for finding circles:

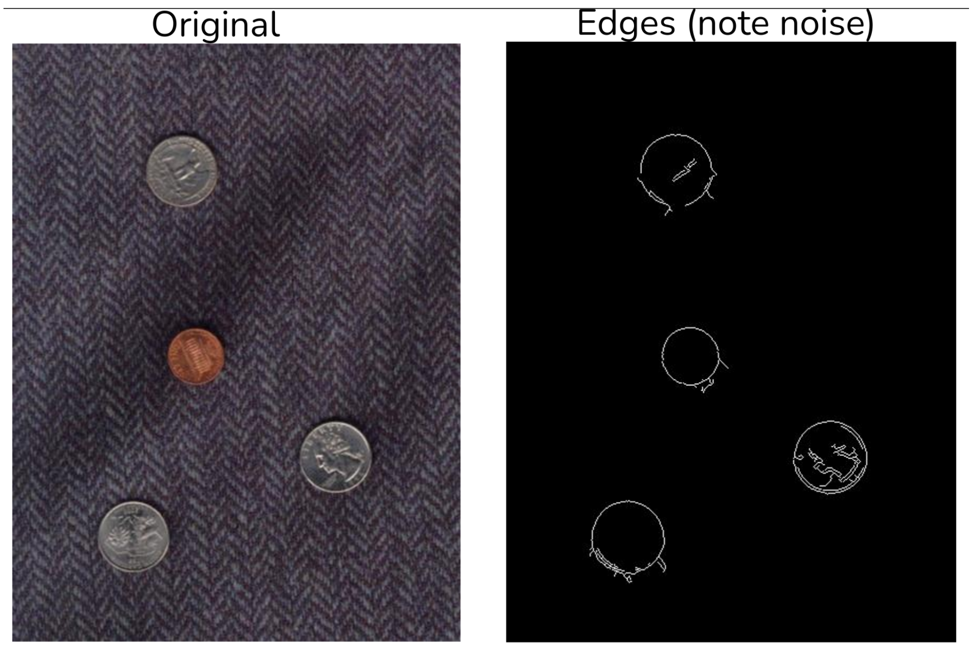

Figure 4: Penny

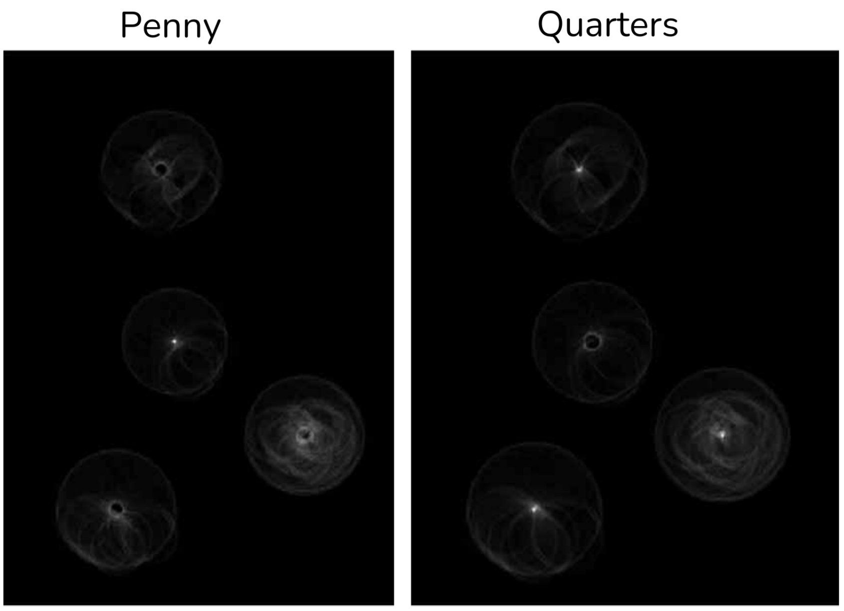

Figure 5: Hough transforms for finding the penny

Here we can see that there are 2 different transforms for the different sizes, and the penny only shows up in the image looking for circles the right size, and in the right image where we are using larger radii, the penny does not appear, only the quarters.

2 Colour

Systems need colour. An incomplete list of reasons why includes

- To tell what food is edible
- To distinguish between material changes from shading changes
- To group parts of one object together in a scene
- Check whether a person's appearance is normal / healthy
- And so on

Newton did some experiments with prisms, where he decomposed sunlight into individual components (remarkably rainbow like), and also where he used a second prism to recombine them into a single beam of white. The visible electromagnetic spectrum goes from around 390nm (violet) to 770nm (red).

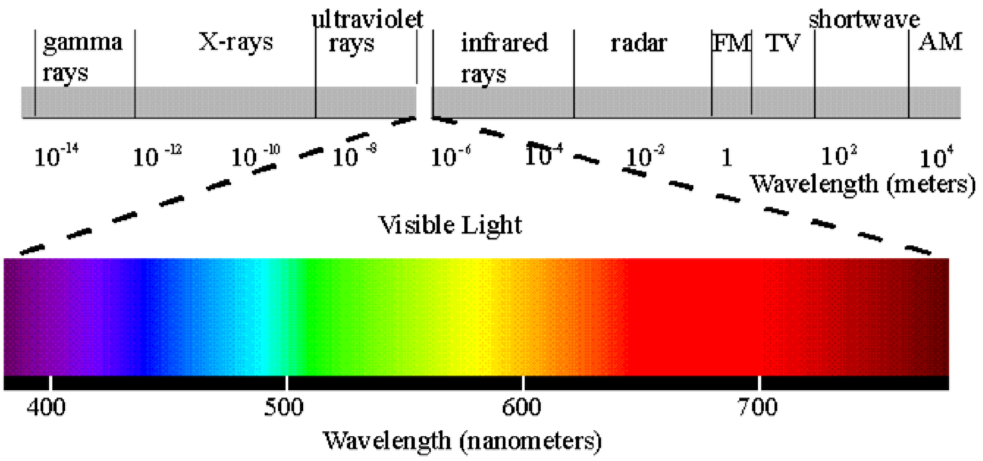

Figure 6: EM spectrum

Humans have 2 types of cells in our eyes for seeing. We have rods, and cones. The **rods** are intensity receptors, that are very insensitive, and can only see in black and white. Great for night vision, and there are around 125 million of them, primarily focused in ones peripheral vision. There are also **cones**, which can receive colour. There are 3 types, one for each of red, green, and blue, and they are densely packed in the retina.

The 3 different types of cones have different **spectral response curves**. Blue is **small** wavelengths, green is **medium** wavelengths, and red is **large** wavelengths. We will note that there is a lot of overlap between red and green response curves, which can result in R/G colour blindness.

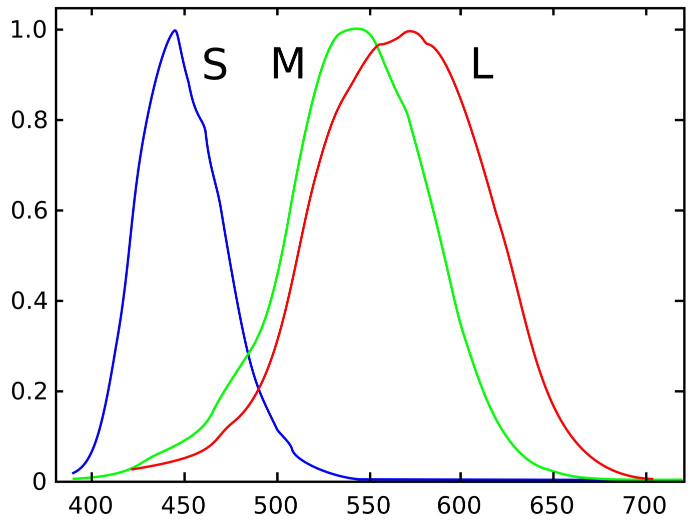

Figure 7: Spectral response curves

Colour matching experiments show that for a typical person no more than three spectra are required to reproduce all perceived colours. For example, RGB. Each colour may be given as

$$T = \sum_{i=1}^{3} w_i P_i$$

So for example,

$$P_1 = 645nm\ (R) P_2 = 526nm\ (G) P_3 = 444nm\ (B) \tag{2}$$

Grassman laid down some laws, for the linearity of colour matching:

1. Colour matching is additive:

$$C_1 + C_2 = (R_1 + R_2) + (G_1 + G_2) + (B_1 + B_2)$$

2. Scaling the colour and the primaries b the same factor preserves the match:

$$aC = aR + aG + aB$$

These statements are true as any biological law. They mean that people behave like **linear** systems in the colour matching experiment.

We cannot quite represent **all** the colours in the world with just RGB, but we can represent enough of them that most people cannot notice the difference. Some professional printers thus have more colours from which to choose, sometimes even up to 9, rather than the standard 4 in a home printer, to help them have more precise colour matching.

We can say in general that

$$C = \begin{pmatrix} c_1(\lambda_1) & \dots & C_1(\lambda_N) \\ c_2(\lambda_1) & \dots & C_2(\lambda_N) \\ c_3(\lambda_1) & \dots & C_3(\lambda_N) \end{pmatrix}$$

Why do we have so many wavelengths for each colour? This is because we do not see only one wavelength per major colour, but rather a range, so this represents that neatly.
Let the spectral signal be described by the vector

$$t = \begin{pmatrix} t(\lambda_1) \\ \vdots \\ t(\lambda_N) \end{pmatrix}$$

Then the amounts of each primary colour needed to match t are Ct.
Two spectra t, and s will perceptually match if $Ct = Cs$, where C is the colour matching functions in a set of primaries. This is **colour metamerism**. Here, despite seeing possibly very different spectra of colours, we will *perceive* the same colour of light. Metameric lights are lights with different spectral power distributions, but appear identical to most observers. Consider for example the light of a tungsten bulb, and the light of a phosphor CRT television. The light origin is completely different, and the spectra are incredibly different, but to most people, the colour of emitted light will be very similar.

2.1 Colour spaces

For only mixing coefficients, the CIE (Commission Internationale d'Eclairage) defined 3 new **hypothetical** light sources X, Y, and Z to replace red, green, and blue:

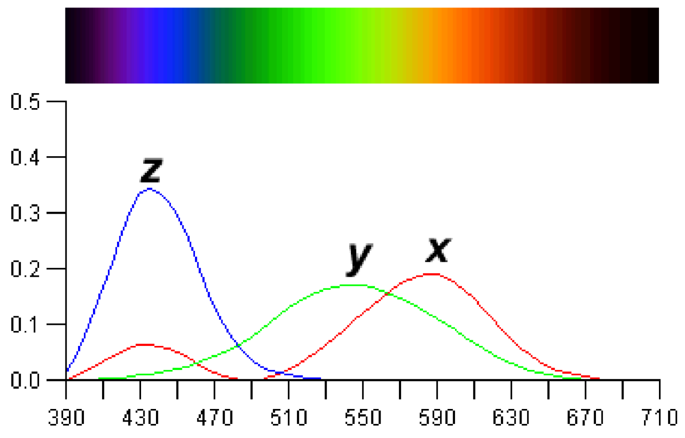

Figure 8: CIE-XYZ colour space

We can translate between this and RGB through multiplying by a matrix:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \tag{4}$$

Not all XYZ colours have positive RGB values, but *all* positive RGB values have positive XYZ values. It is often convenient to work in 2D colours space, so the 3D colour space is projected onto the $x + y + z = 1$ plane, to yield the chromaticity diagram. This diagram is presenting the three imaginary colours, X, Y, Z. All the weights are positive, and it represents all the visible colours (tha gamut of human vision). Colours are additive, and the weights of X, Y, Z form the 3D CIE-XYZ space.

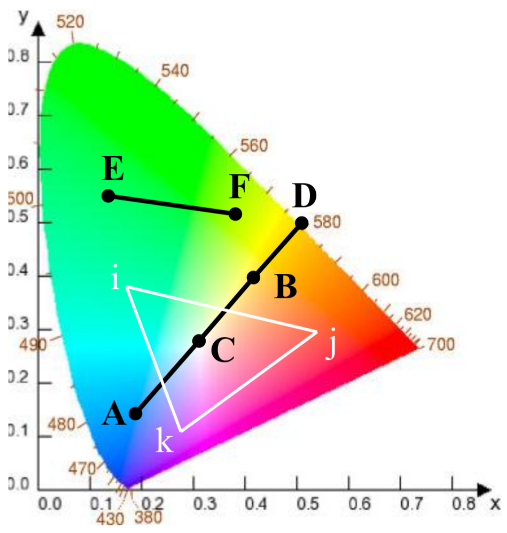

Figure 9: CIE chromaticity diagram

In this diagram, C is “white”, and A and V can be mixed to produce any colour along the line AB., including white. The same holds for EF, but without white, and for IJK, which does include white.
We can draw the MacAdam ellipses on this diagram (not shown here), where the idea is that within these ellipses, humans cannot distinguish between the colours.

2.1.1 YIQ and YUV colour models

This is a recoding of RGV for transmission efficiency, and compatibility with greyscale televisions. Recall from the first lecture

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

We can also consider RGB on a cube. RGB is additive, meaning we add different amounts of the 3 primaries to get every colour, so should we represent this on a cube, then the greyscale colours will appear on the diagonal between the black corner, and the white colour. We can thus flatten this cube onto a hexagon.

2.1.2 HSV colour space

Here we have 3 values, the *Hue*, which is the colour of the pure pigment, the *Saturation*, which is a measure of colourfulness, or the distance from the grey line, and value, which is a measure of brightness. Maximum saturation is at $V = 1$, and at $V = 0$, then the hue is undefined.

2.1.3 CMYK - subtractive colour model

A computer monitor additively mixes red, green, and blue to create colour, where white is the maximum of all 3 primaries. A CMYK printer instead uses light absorbing cyan, magenta, and yellow inks, and white is the colour of the background. This is useful, since paper is not emissive.

There are **lots** of colour spaces.

- CIE - XYZ: Linear
- CIE - $L * a * b$: Non linear
- CIE - RGB: Linear
- HSV and HSL: Nonlinear
- CMY: Linear
- CMYK: Nonlinear
- YUV, YIQ: Linear
- And more...

2.2 Colour histograms

These are usually density functions $\mathbb{R}^3 \to \mathbb{R}$. The value of the bin is the number of image pixels that have the RGB values equal to $[r_i, g_i, b_i]$. Assuming an 8 bit depth RGB image, the RGB histogram has $256^3 = 16,777,216$ bins, which means it is usually very sparse. A natural image property that contributes to the spareseness of colour histograms is the *piecewise smooth world property.*

Suppose we want to find images that are similar to a query image. Comparing pixel values is extremely sensitive to changes in the image. Comparing colour histograms is much more robust, since we are only comparing the colour palette.

There exist some different ways of calculating the distance between two histograms:

- Euclidean distance:
$$d^2(h,g) = \sum_A \sum_B \sum_C (h(a,b,c) - g(a,b,c))^2$$
- Intersection score
$$s(h,g) = \sum_A \sum_B \sum_C \min(h(a,b,c), g(a,b,c))$$
- EMD - "Earth Mover's Distance"

In the RGB space, an object with a uniform colour, and varying light reflectance forms a 3D line through the origin.

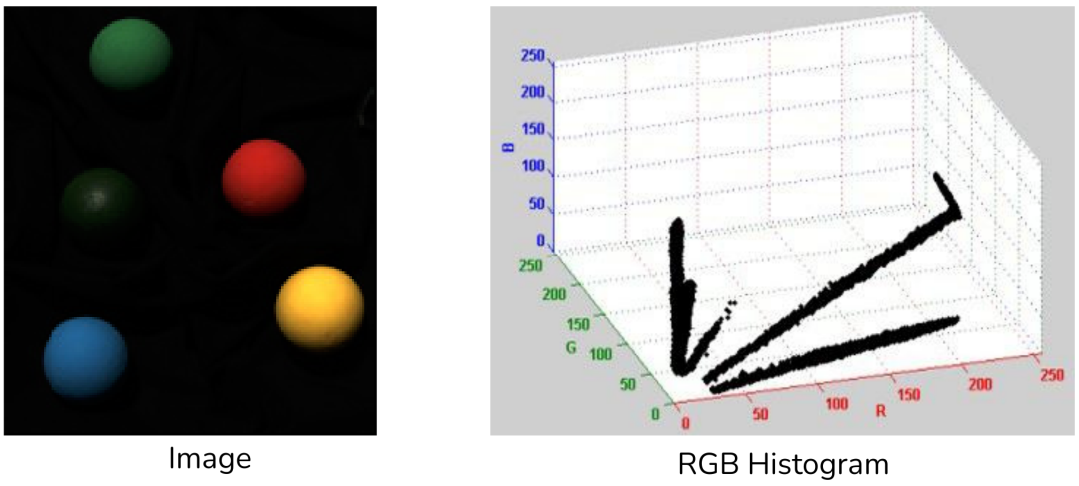

Figure 10: Colour lines in colour histograms

In bright image regions, the pixel values may be clipped to a maximum value. This causes a loss of details, and colour distortion. The idea is to use the unclipped values of the line to recover clipped ones. Practically: identify a saturation (or breaking) point, fit a 3D line to the unclipped part, and predict the clipped pixel values.

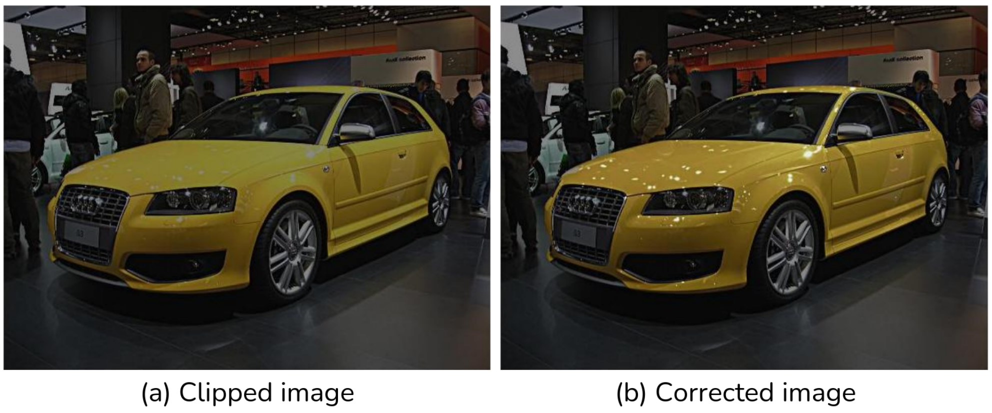

Figure 11: Correction of a clipped colour image of a car