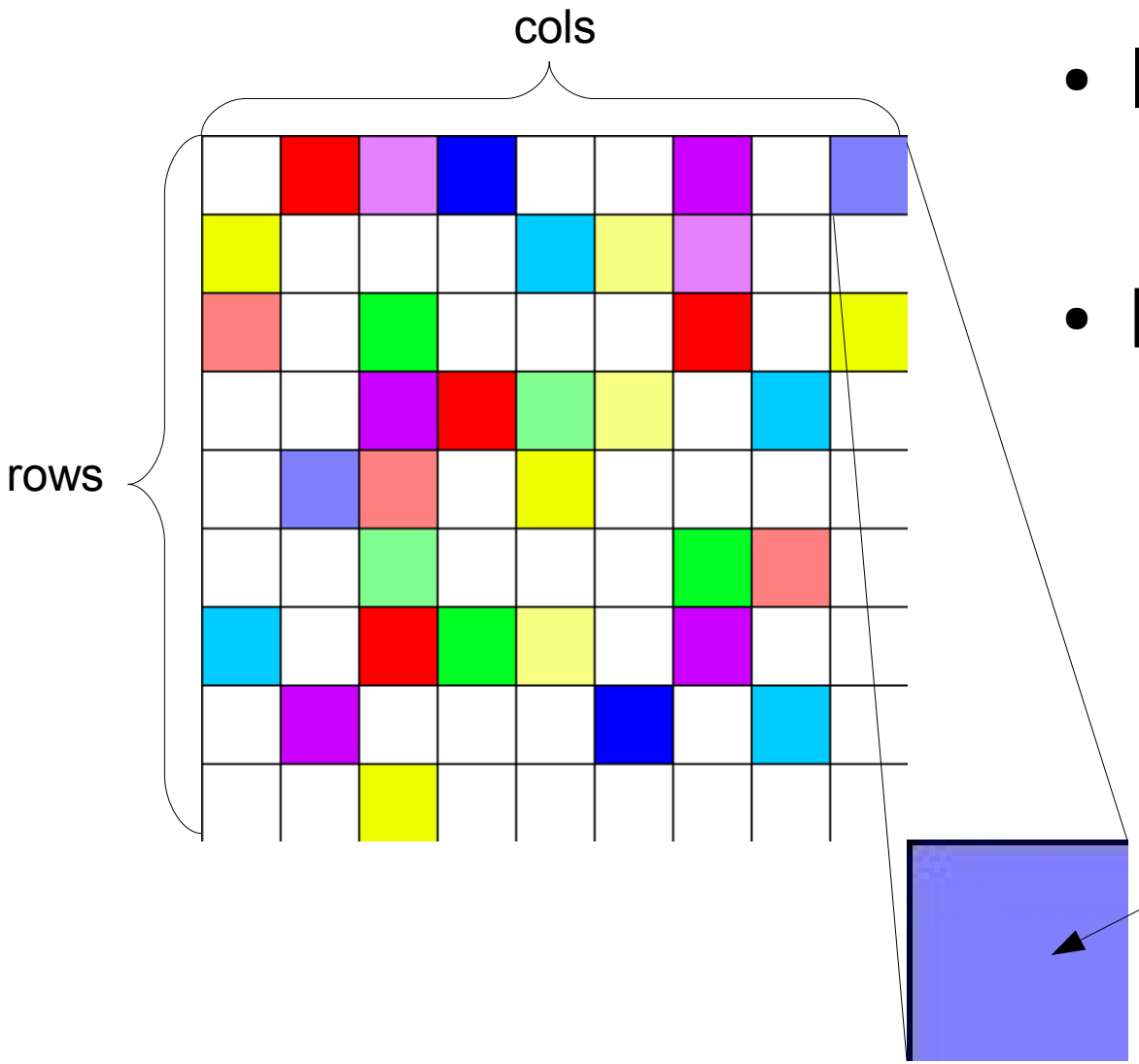
Digital Image Representation

Defining an Image



- Image Grid
  - cols * rows
- Pixels
  - Contains a single **code value**

may be a vector

(R,G,B) = (128, 128, 255)
“light purple”

Image Model

- A function $f : \boldsymbol{R} \rightarrow \boldsymbol{V}$
  - **R** is the rectangular image grid
  - **V** is the code value of a pixel (scalar or vector)
  - Maps an image grid to pixel values
- Thus can define a pixel uniquely with:

$$(x, y, f(x, y))$$

Pixel Values

- Data Types (bit-depth)
  - Binary
  - Unsigned integers
  - Signed integers
  - Floating points

Data type	Memory	# of Values	Range
Binary	<= 1 byte	2	[false - true]
8-bit u integer	**1 byte**	**256**	**[0 - 255]**
16-bit u integer	2 bytes	65536	[0 - 65535]
32-bit floating point	4 bytes	“lots”	[0.0 - 1.0]
64-bit floating point	8 bytes	“lots lots”	[0.0 - 1.0]

- Channels
  - 1 channel (scalar code value) = binary or grayscale image
  - 3 channel (vector code value) = colour image (e.g. RGB)
  - Operation on channels are independent

Defining an Image

- Colour
  - Represented by separate, independent channel values
  - Different representations (colour spaces) possible, e.g.
    - RGB (or BGR ordering)
    - HSV (or HSL)
    - Lab
    - CMYK
- Resolution
  - Number of pixels
    - 800 cols * 600 rows
  - Spatial - Number of pixels per measurement unit
    - pixels (dots) per inch, pixels per mm

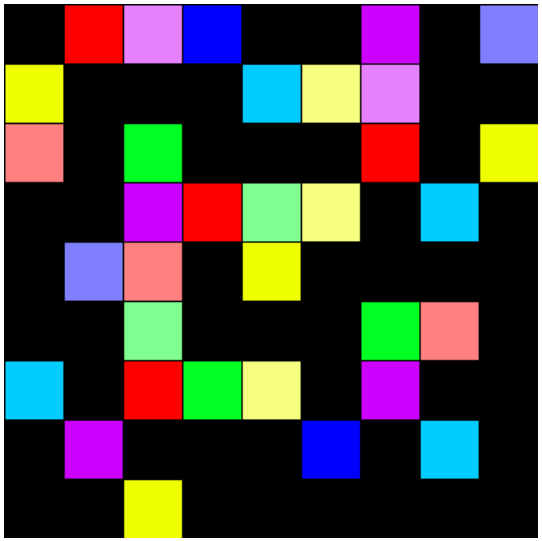
More Image Types

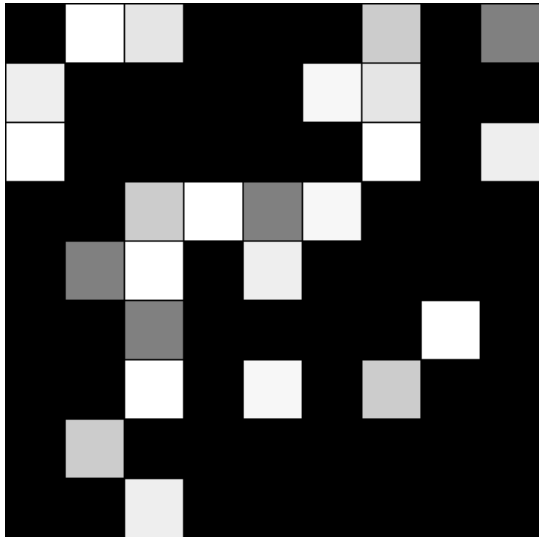
- Videos
  - A sequence of images
- 3D images
  - The image grid is extended to the 3rd coordinate
  - Time varying images (similar to videos)
- 4D videos
  - Time varying 3D images

Example

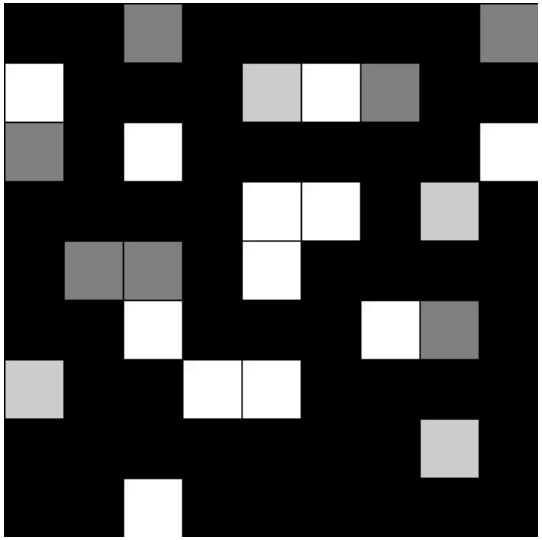
- Defining an image
  - 1600 cols * 1200 rows
  - 16 bits unsigned integer per channel
  - 3 channels
    - RGB colour model
    - Data stored in BGR order
    - *(48 bits required per pixel)*
  - Displayed at 72 dots per inch (dpi)
    - i.e. physical size = 22.22 in * 16.67 in

RGB Colour

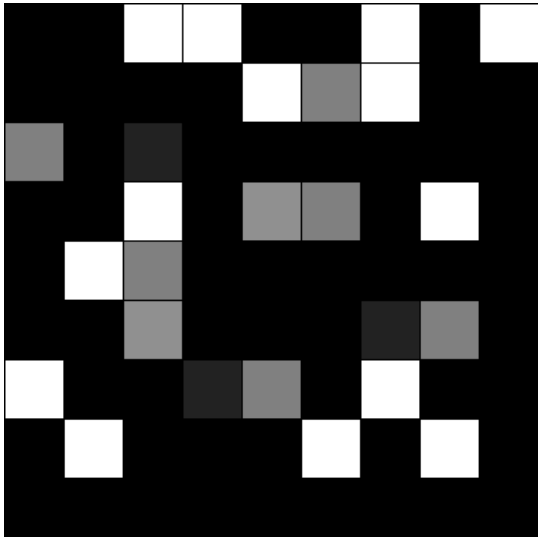




R



G



B

Thresholding and Linear Mapping

General Idea

For every pixel's intensity as input, apply a (linear) function to generate an output intensity for that pixel

for each pixel (x, y)

$$output\_value_{(x,y)} = func\ (\ input\_value_{(x,y)}\ )$$

Threshold Functions

- Thresholding is like an if-else statement
- Different types possible:
  - Threshold to binary

```
if (input > threshold)
  output = max // 255 for 8-bit images, or 1 for binary
else
  output = 0
```

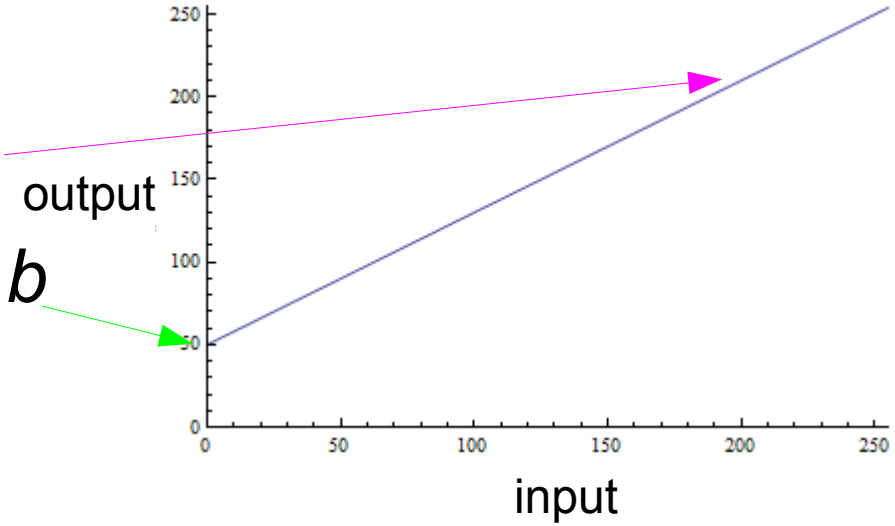
  - Threshold below

```
if (input < threshold)
  output = 0
else
  output = input
```

Linear Functions

- Change intensity of a pixel linearly
- In general: $\text{out}(x, y) = a \cdot \text{in}(x, y) + b$

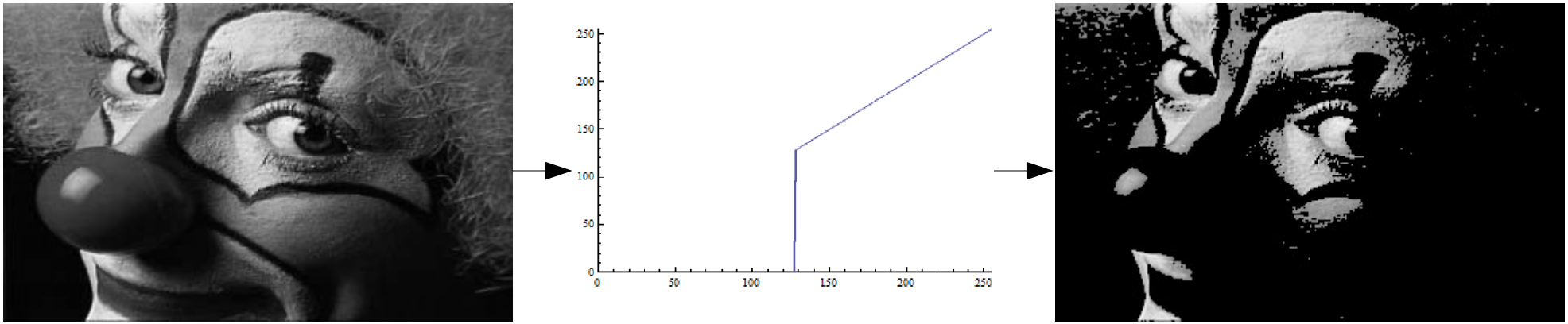
  - Gain (gradient) *a*
  - Bias (y-intercept) *b*



- In image processing, the output is clamped to the allowed pixel values (e.g. [0 - 255]).

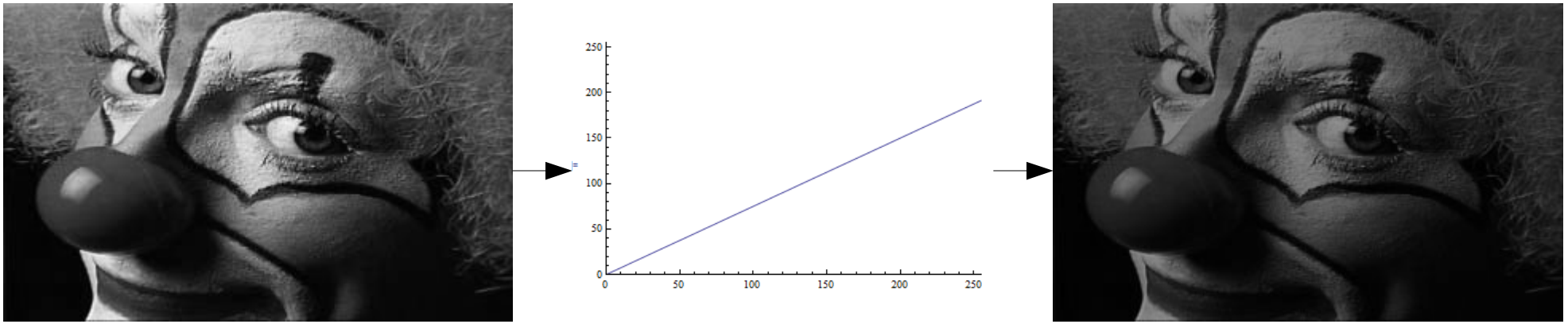
Examples

- Threshold below 128
  out = (in < 128) ? 0 : in



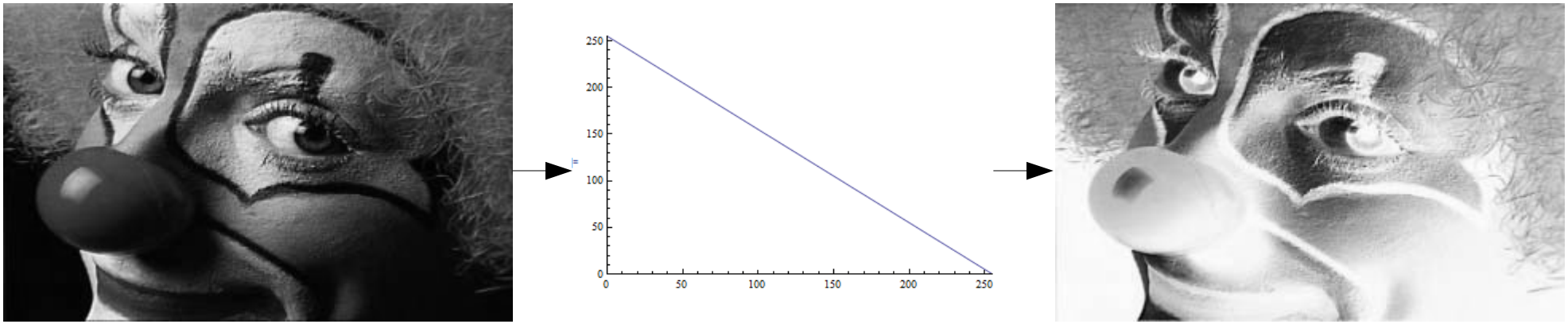
Examples

- Reducing contrast
  out = 0.60 * in



Examples

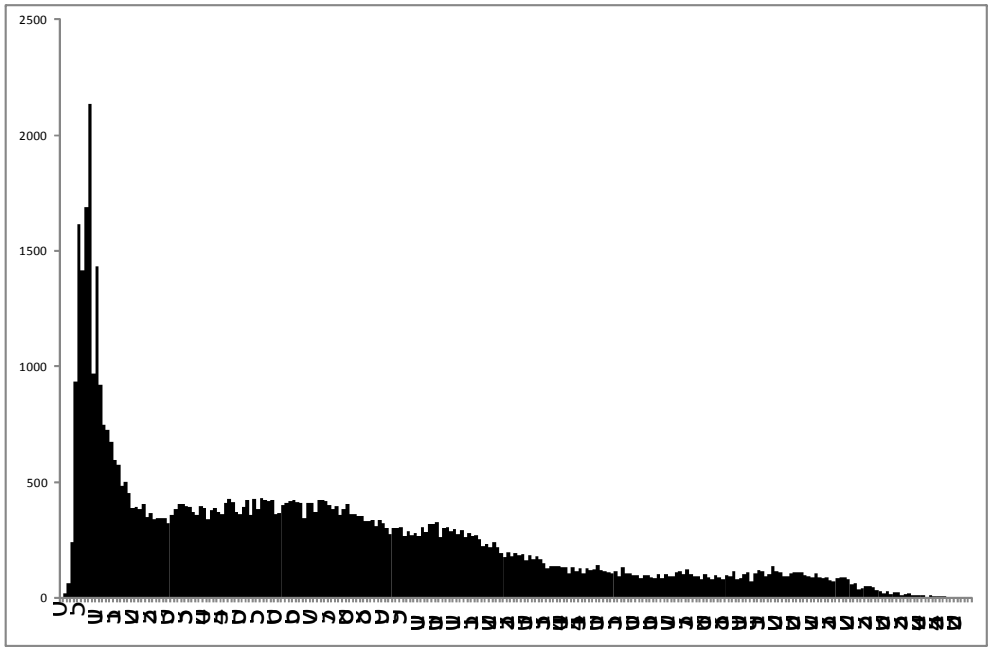
- Negation (8-bit)
  out = -1 * in + 255
  out = 255 - in



Histograms

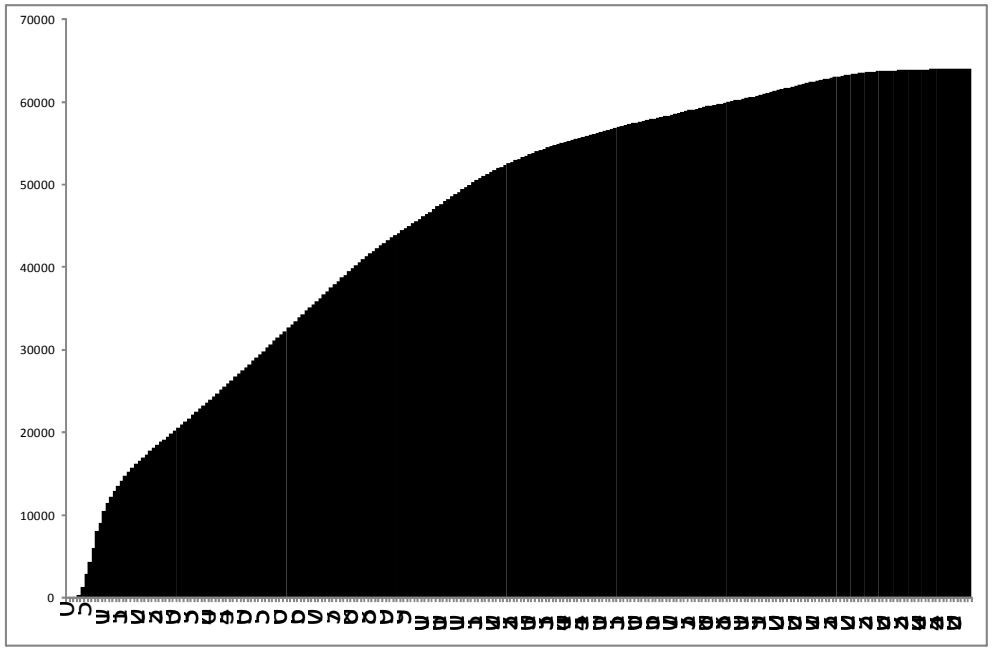
Definition

- A histogram of an image is the frequency of each possible intensity value of the image.
- $m_i = \text{count}\{f(x, y) = i\} \quad \text{for} \quad 0 \leq i \leq I_{max}$



Cumulative Histograms

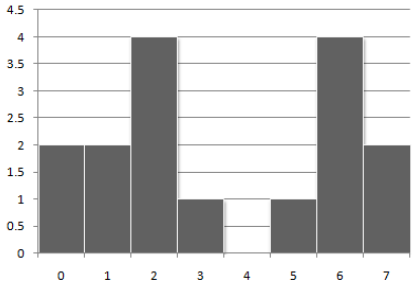
- The cumulative histogram is the frequency of all pixels with intensity lower than a value.
- $M_i = \underset{i}{\text{count}}\{f(x, y) \leq i\}$ for $0 \leq i \leq I_{max}$
- $M_i = \sum_{j=0} m_j$



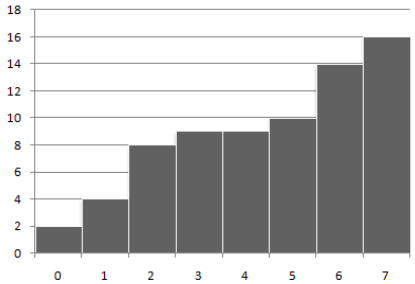
Example

2	6	2	7
2	6	1	2
6	0	1	3
5	0	6	7

Value	Count
0	2
1	2
2	4
3	1
4	0
5	1
6	4
7	2
Total	16



m



M

Normalised Histograms

- Express the histogram as observed probability.
  - $\tilde{m}_i = \frac{m_i}{N}$

    where *N* = *cols* * *rows* is the number of pixels

- For the cumulative histogram, the normalised version is called the cumulative distribution function (cdf).
  - $cdf_i = \sum_{j=0}^{i} \tilde{m}_j = \frac{M_i}{N}$

Histograms

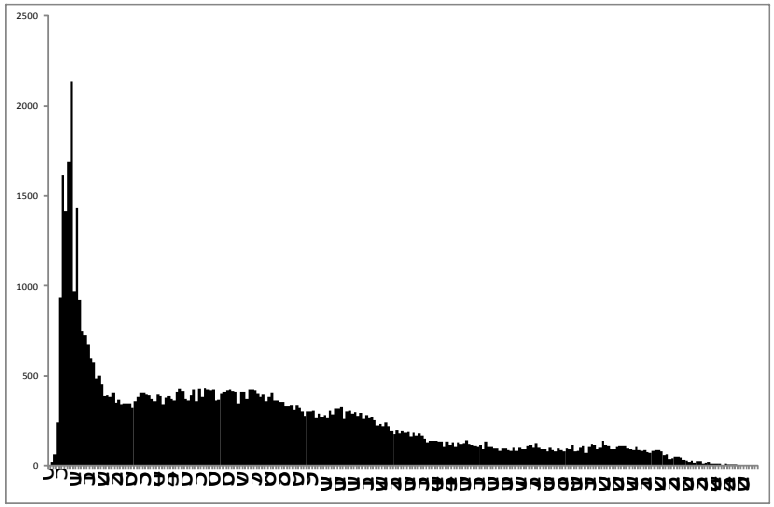
- Note:
  - $M(I_{max}) = N$
  - *cdf* has range [0.0 - 1.0]
  - $cdf(I_{max}) = 1.0$

Histograms

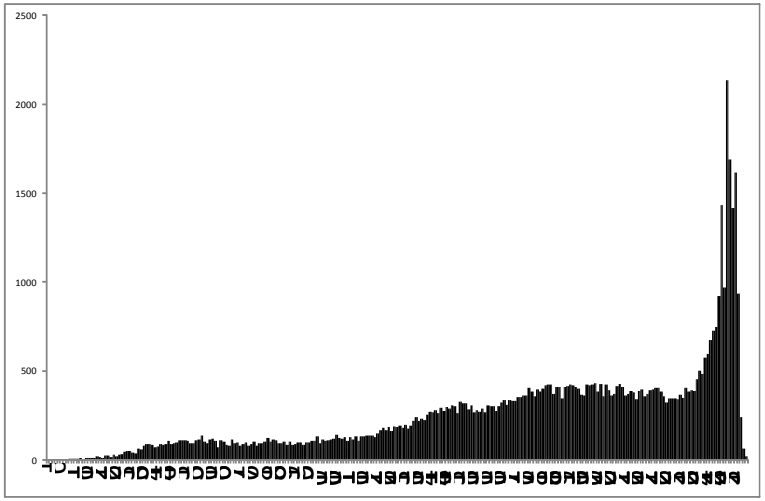
- Thought exercise:
  - What is the histogram of the clown image flipped upside down?

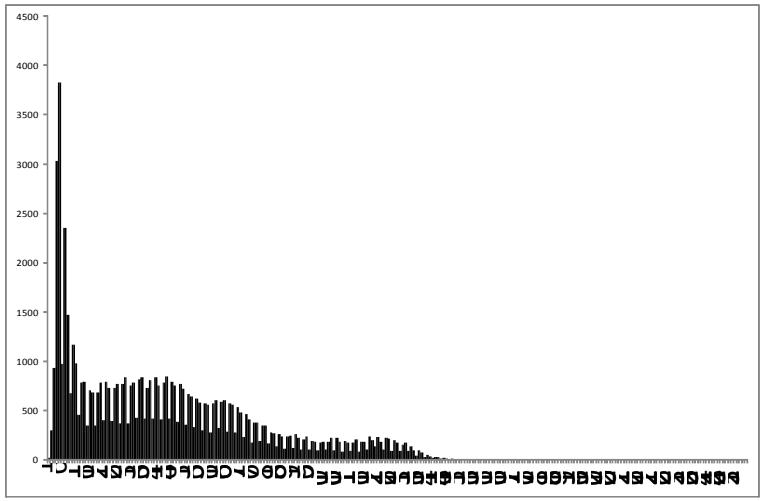
- An image is not uniquely defined by its histogram.
- Any permutation (rearrangement) of pixels of an image will give an identical histogram.
  - And thus, an identical cumulative histogram.

Linear Mapping and Histograms



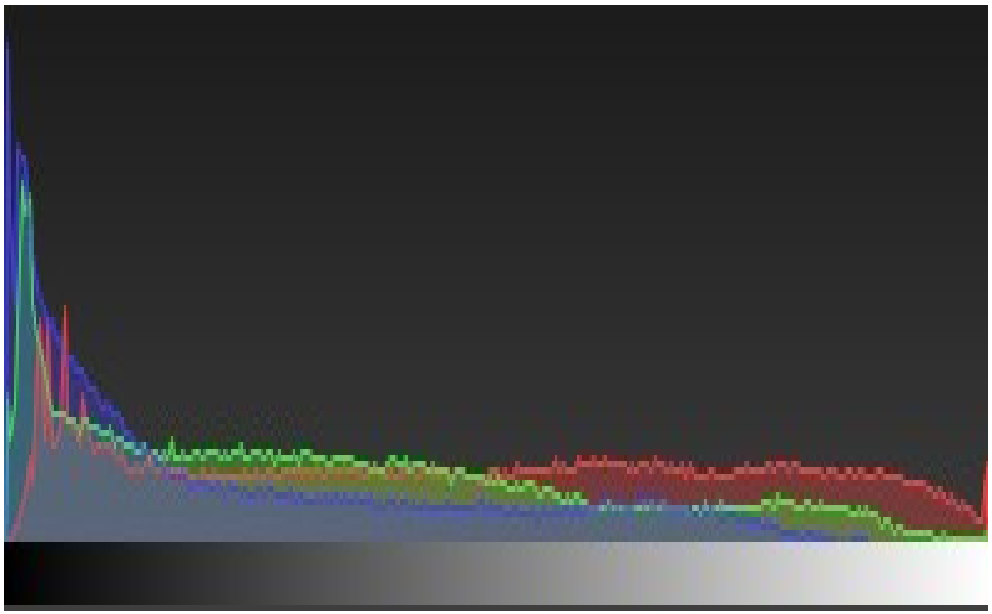
Linear Mapping and Histograms





Colour Histogram

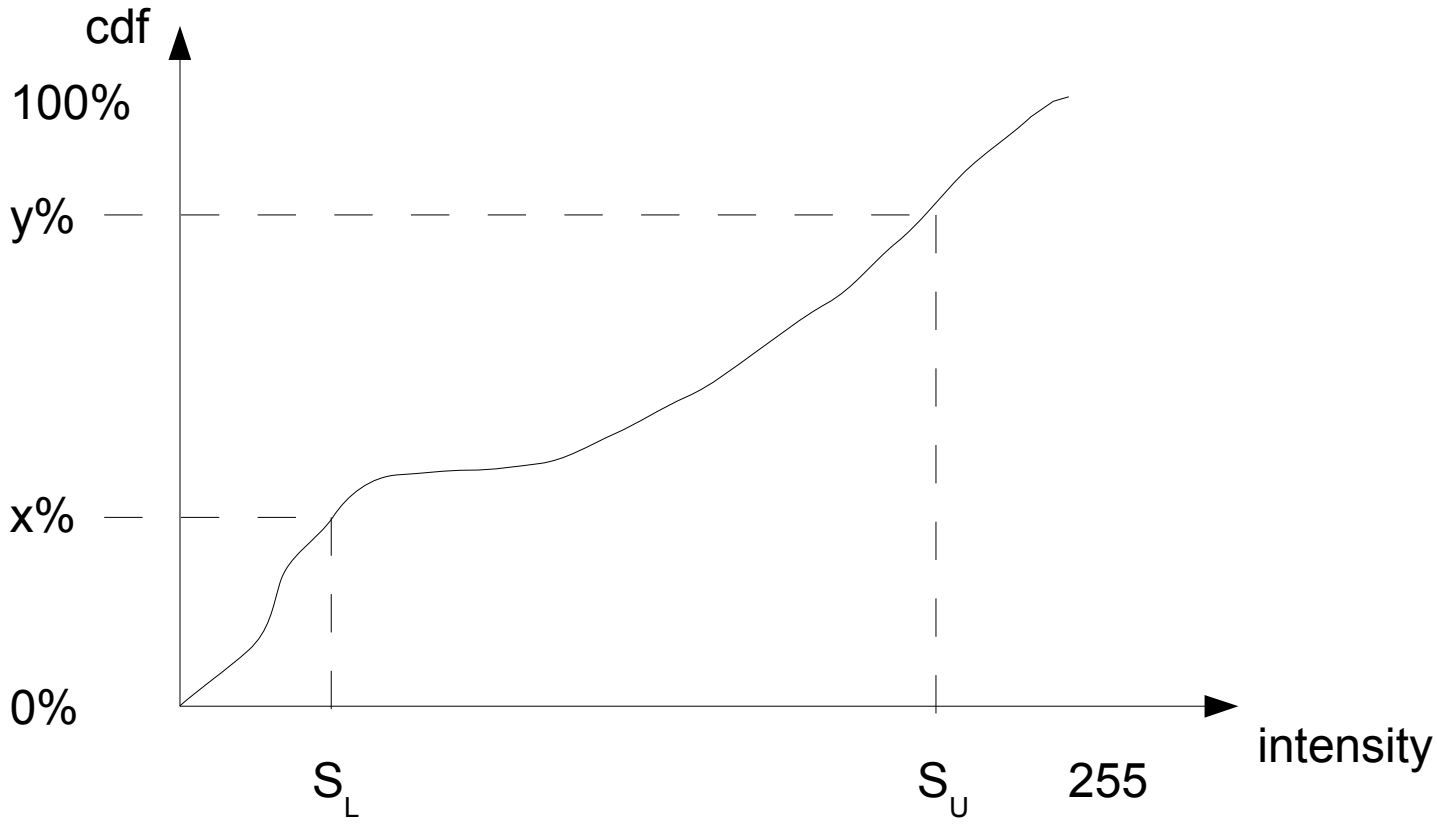
- Different ways to express colours in histogram:
  - Convert to grey intensities
  - Graph each colour separately
  - Graph in 3D



Histogram Stretching

- Use linear mapping to enhance the contrast of the image keeping the middle x% to y% of the original image.
  - Apply the cumulative histogram or cdf
  - Find the intensity value of x% and y%
  - Use linear mapping to map the intensity value at x% and y% to 0 and 255

Histogram Stretching



- Derivation of linear mapping formula
  - We want to map a range of intensities linearly to another range using gain a and bias b
  - source lower and upper bound $S_L < S_U$
    target lower and upper bound $T_L < T_U$
  - Solve simutaneous equations for gain a and bias b
  - For histogram stretching set target range to [0, 255]

$$T_L = a * S_L + b$$
$$T_U = a * S_U + b$$

$$b = T_L - a * S_L$$
$$T_U = a * S_U + T_L - a * S_L$$
$$T_U - T_L = a(S_U - S_L)$$

$$a = \frac{T_U - T_L}{S_U - S_L}$$

$$b = T_L - \frac{T_U - T_L}{S_U - S_L} * S_L$$

Histogram Equalisation

Idea

- Stretch histogram such that frequencies are equally distributed.
  - The histogram is flat. $m_i \approx K$
  - The integral of the histogram (the cumulative histogram) increases linearly. $M_i \approx i \cdot K$

How to do it

- Apply the *CH* or *cdf* as a Look Up Table (LUT).
  - $\text{result}(x, y) = cdf\,(\text{input}(x, y)) \cdot I_{max}$
- However, we need to normalize the result to the maximum range available. Use contrast stretching.
  - $\text{result}(x, y) = \dfrac{M(\text{input}(x, y)) - M_{min}}{M_{max} - M_{min}} \cdot I_{max}$

  - When M*(input)* = M_{min}, *result = 0*
  - When M*(input)* = M_{max}, *result* = I_{max}

How to do it

- It does not matter whether we use the normalized version (cdf) or not.
- M_{min} is the first non-zero value.
- M_{max} is always $N = cols * rows$.
- Round the result to the nearest integer in the case of integer data types.

Example

2	6	6	2
5	7	3	7
2	6	3	5
6	7	2	4

4-bit image (max val = 7)

Value	m	M
0	0	0
1	0	0
2	4	4
3	2	6
4	1	7
5	2	9
6	4	13
7	3	16

Example

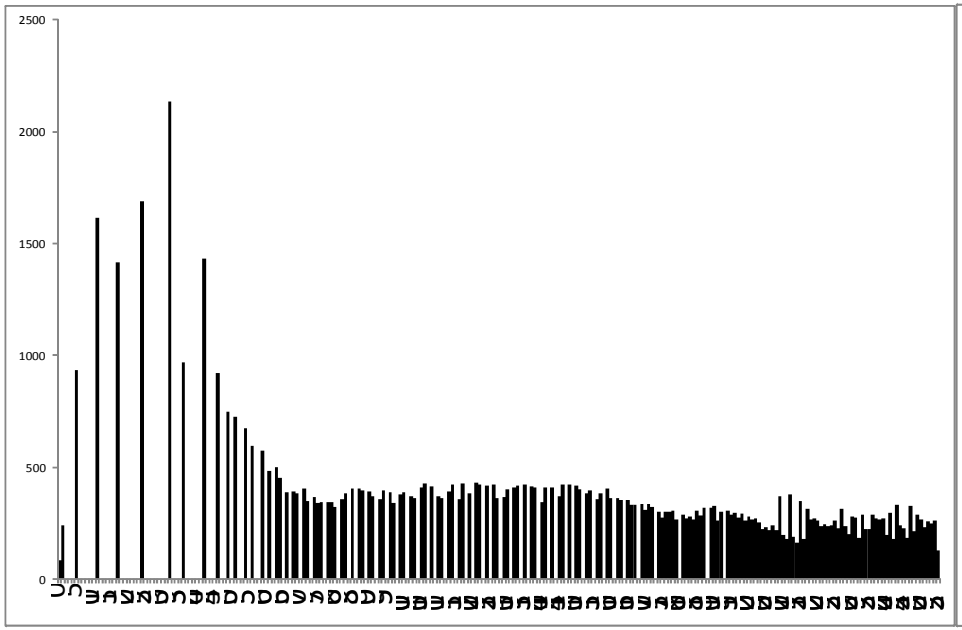
Value	M	out
0	0	N/A
1	0	N/A
2	4	0
3	6	1.17
4	7	1.75
5	9	2.92
6	13	5.25
7	16	7

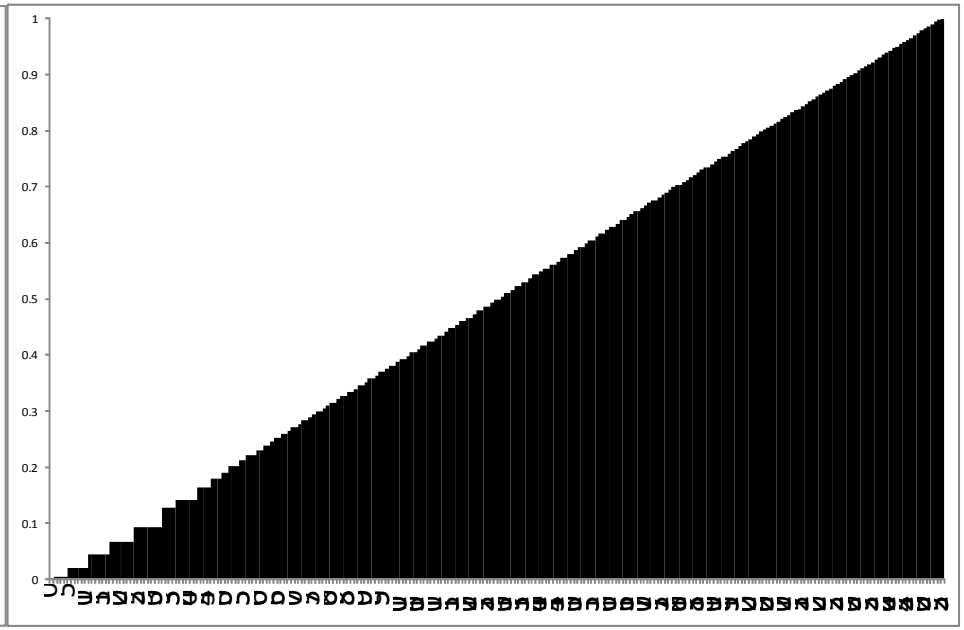
$$\text{out}(x,y) = \frac{M(\text{in}(x,y)) - M_{min}}{M_{max} - M_{min}} \cdot I_{max}$$

$$\text{out}(x,y) = \frac{M(\text{in}(x,y)) - 4}{16 - 4} \cdot 7$$

0	5.25	5.25	0
2.92	7	1.17	7
0	5.25	1.17	2.92
5.25	7	0	1.75

Example





m M

Demos

- Building histograms with ImageJ macro

```
width = getWidth();
height = getHeight();
histo = newArray(256);

for (y = 0; y < height; y++) {
  for (x = 0; x < width; x++) {
    v = getPixel(x, y);
    histo[v]++;
  }
}

for (i = 0; i < 256; i++) {
  print(i, histo[i], "\n"); // equivalent to print(i + " " + histo[i] + " " + "\n");
}
```

Extras

LUT Linear Mapping

- Linear Mapping can be inefficient.
  - Recompute linear equation for every pixel.
- Use a Look Up Table (LUT)
  - One input value can only have one output, no matter which pixel it is.
  - Build a LUT of all possible input values (e.g. [0 – 255]) and their output.
- LUT is faster
  if possible input values << number of pixels

LUT Linear Mapping

```
width = getWidth();
height = getHeight();
lut = newArray(256);

// build LUT for gain 0.95 and bias 25
for (i = 0; i < 256; i++) {
  lut[i] = round(0.95 * i + 25);
}

for (y = 0; y < height; y++) {
  for (x = 0; x < width; x++) {
    v = getPixel(x, y);
    setPixel(x, y, lut[v]);
  }
}
```