MANCHESTER 1824
The University of Manchester

COMP27112

Introduction to Visual Computing

Image Processing
Lecture 4.1

Terence Morley
School of Computer Science
The University of Manchester

MANCHESTER 1824
The University of Manchester

Objectives for this Lecture

To develop an understanding of

Point-processing techniques

- Image histograms
- Grey-level mapping
- Thresholding
- Geometrical transformations

Image Histogram

Region of a greyscale image

152	158	166	165	163	171	175	169
159	160	160	164	173	172	169	173
153	160	155	158	172	169	164	175
151	156	157	154	161	165	167	174
150	150	156	157	157	164	171	171
154	155	157	158	160	161	166	165
154	153	150	154	161	159	158	161
152	156	152	153	159	160	155	161

What is its histogram?

Image Histogram

Histogram of the previous image region

I	Freq
150	3
151	1
152	3
153	3
154	4
155	3
156	3
157	4
158	4
159	3
160	5
161	5
162	0
163	1
164	3
165	3
166	2
167	1
168	0
169	3
170	0
171	3
172	2
173	2
174	1
175	2

Image Histogram

Pseudocode to generate histogram

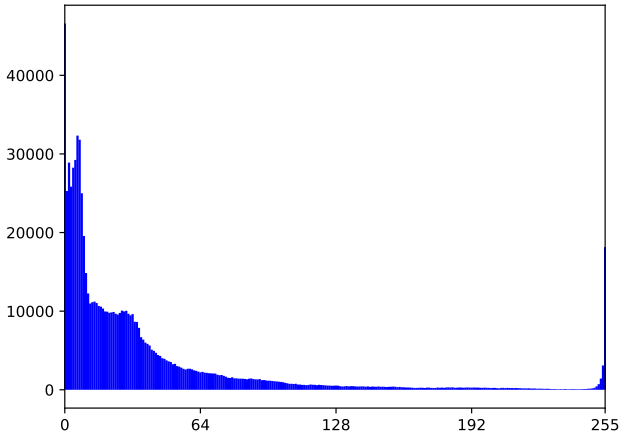
```
int histogram[256] = {0};

for(int i = 0; i < pixels in img; i++)
{
   histogram[ img[i] ]++;
}
```

Image Histogram

A *mostly* dark image

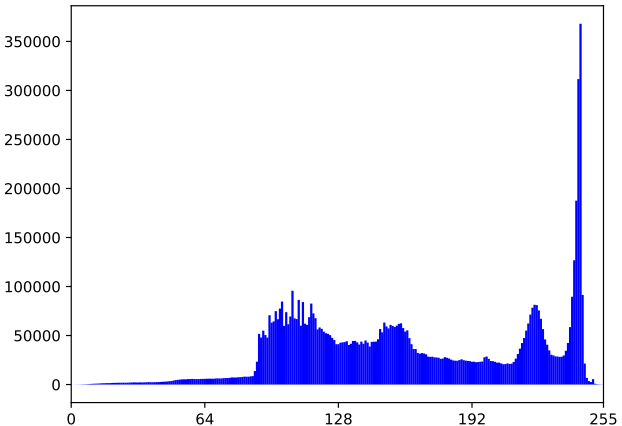
Image Histogram



Dark image

Histogram of dark image

Image Histogram



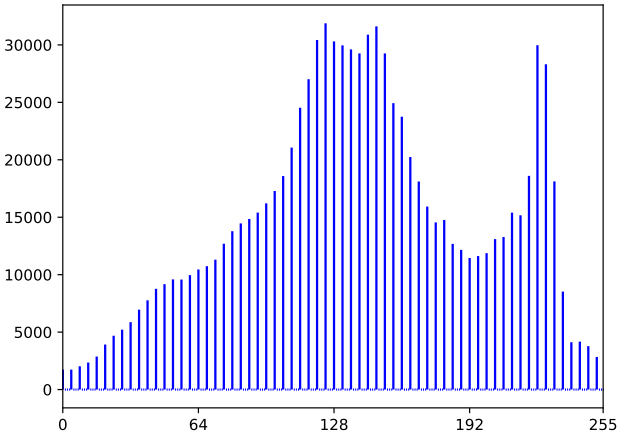
Light image

Histogram of light image

Image Histogram

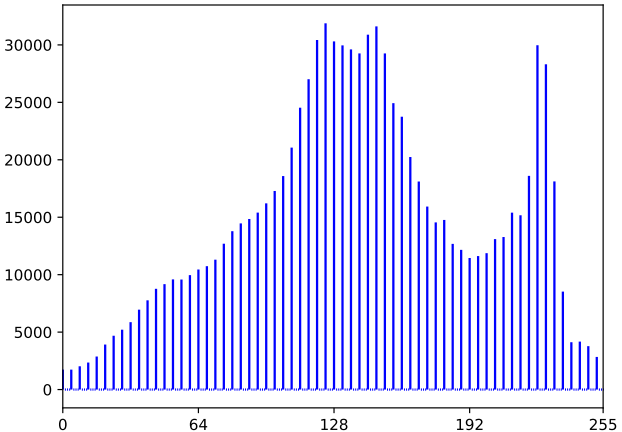
City scene

Image Histogram



Qu. Why does the histogram look like this?

Image Histogram



Histogram shows that only 64 grey levels exist in this image

Image Histogram

Incidentally, notice that the image with only 64 levels seemed perfectly fine.

Compare that with the same image having 32 levels (look at the sky):

Brightness Adjustment

$I(x, y)$

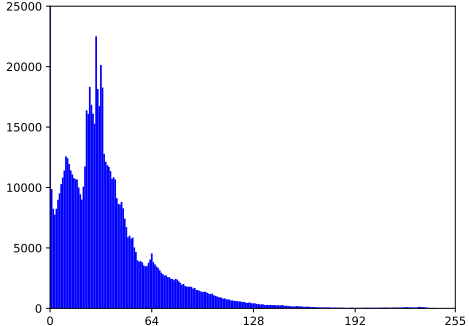
g

$I'(x, y) = I(x, y) + k$

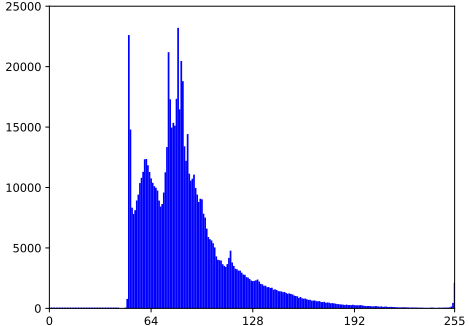
$g' = g + k$

In this case $k = 50$

Brightness Adjustment



$I(x, y)$



$I'(x, y) = I(x, y) + k$

g

$g' = g + k$

In this case $k = 50$

Contrast Adjustment

$I(x, y)$

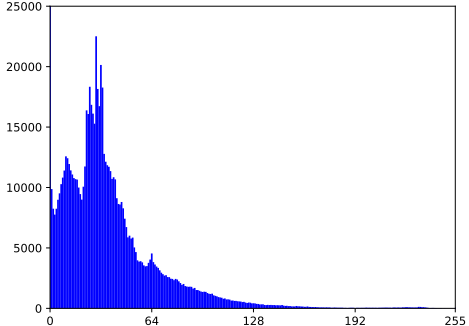
g

$I'(x, y) = I(x, y) \times k$

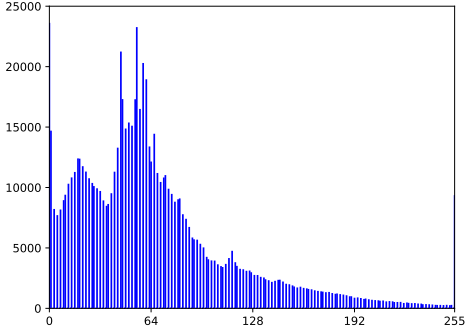
$g' = g \times k$

In this case $k = 1.8$

Contrast Adjustment



$I(x, y)$



$I'(x, y) = I(x, y) \times k$

g

$g' = g \times k$

In this case $k = 1.8$

MANCHESTER 1824
The University of Manchester

Clipping

Qu. What is $255 + 1$ when dealing with unsigned bytes?

Clipping

Qu. What is $255 + 1$ when dealing with unsigned bytes?

$$255 + 1 = 0$$
$$255 + 2 = 1$$
etc.

Clipping

Qu. What is $255 + 1$ when dealing with unsigned bytes?

$$255 + 1 = 0$$
$$255 + 2 = 1$$
etc.

	1	1	1	1	1	1	1	1
+								1
1	0	0	0	0	0	0	0	0

(**1** = overflow)

Clipping

Qu. What is $255 + 1$ when dealing with unsigned bytes?

$$255 + 1 = 0$$
$$255 + 2 = 1$$
etc.

Qu. What effect will this have when adding 50 to an image to increase its brightness?

Clipping

Original

No clipping

Clipping

Clipping

When you process pixel values, you need to check for the result going over 255 and below 0

Example pseudocode

```
uint8 px = 250;
...
int32 p = px;
p += 50; // Increase brightness
if(p > 255) p = 255;
if(p < 0) p = 0;
px = p;
```

MANCHESTER 1824
The University of Manchester

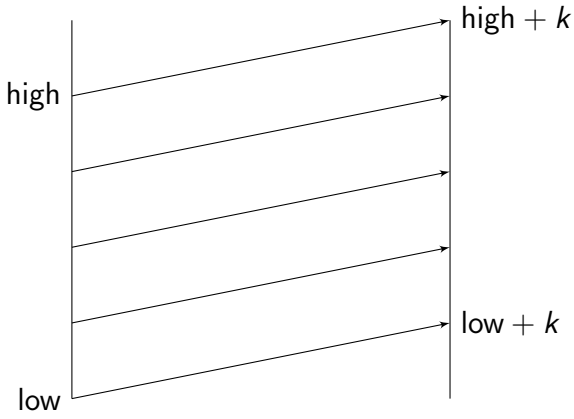
COMP27112

Introduction to Visual Computing

Image Processing
Lecture 4.2

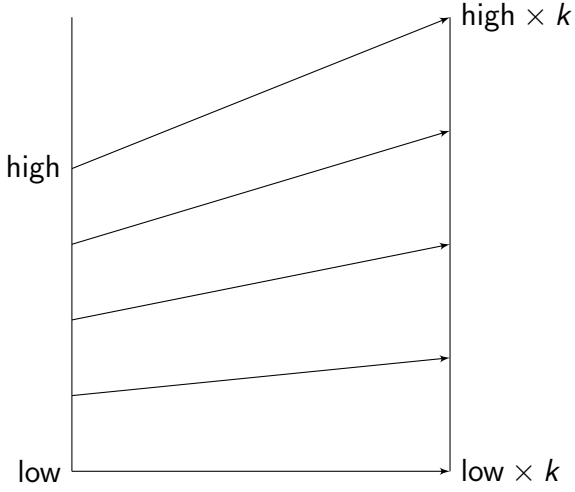
Terence Morley
School of Computer Science
The University of Manchester

Input-Output Mapping



Brightness adjustment

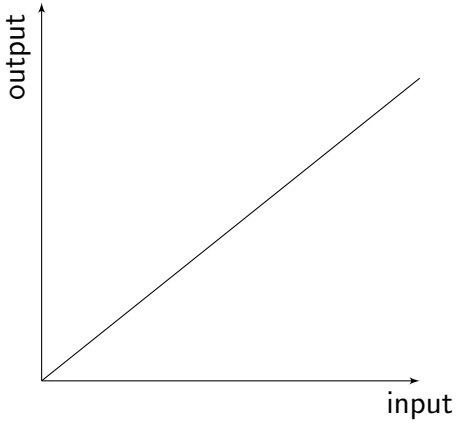
Input-Output Mapping



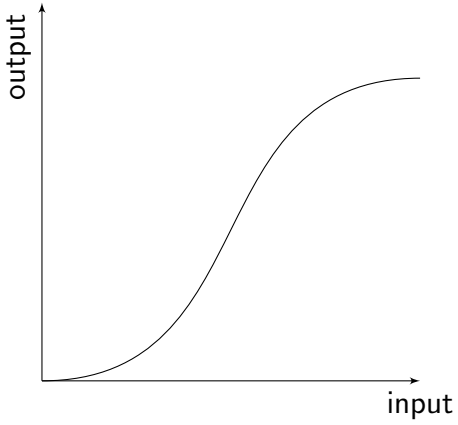
Contrast adjustment

Input-Output Mapping

Creating lookup tables to relate an input grey level to an output grey level



Linear



Non-linear

Input-Output Mapping

Mappings can be created in code with a mathematical formula:

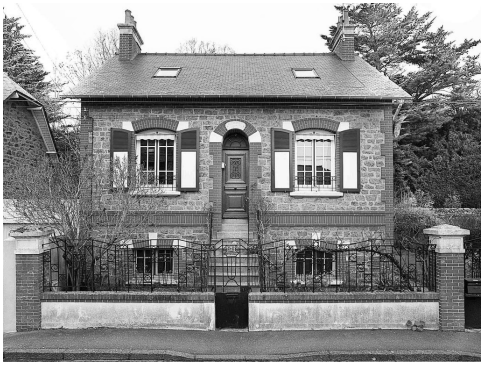
$$\text{img[r][c]} = \text{img[r][c]} * 1.5$$

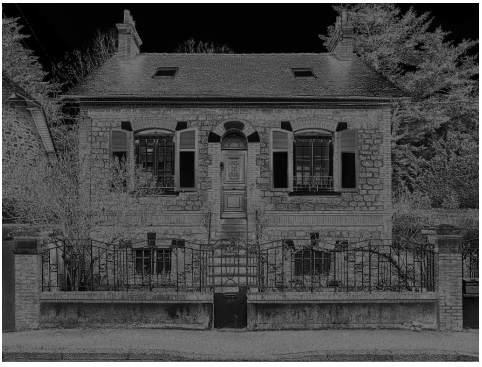
Or with a lookup table:

$$\text{map[256]} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 2 & 3 & 5 & 6 & 8 & 9 & 11 & 12 & 14 & \cdots \\ \hline \end{array}$$

$$\text{img[r][c]} = \text{map[img[r][c]]}$$

Solarise



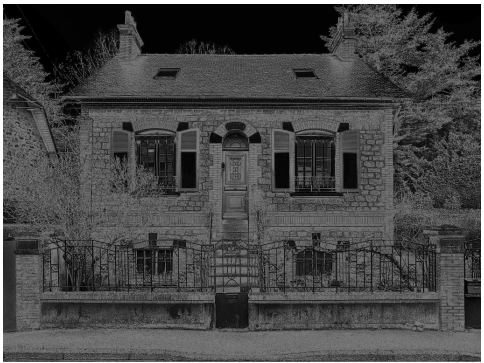


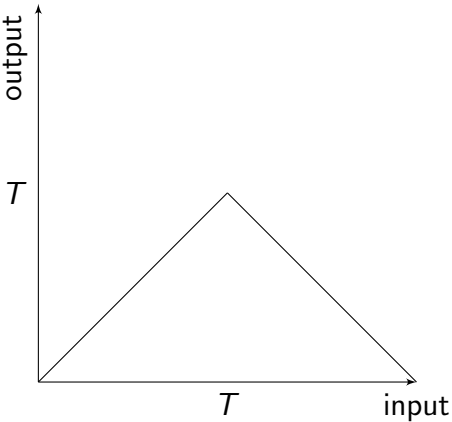
$I(x, y)$

$$I'(x, y) = \begin{cases} 255 - I(x, y) & \text{if } I(x, y) > T \\ I(x, y) & \textit{otherwise} \end{cases}$$

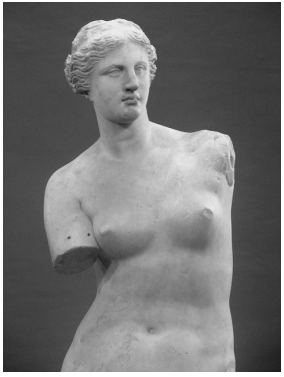
In this case $T = 127$

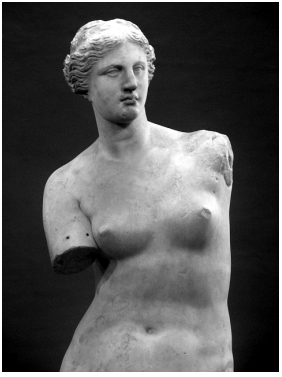
Solarise





Min-Max Linear Stretch

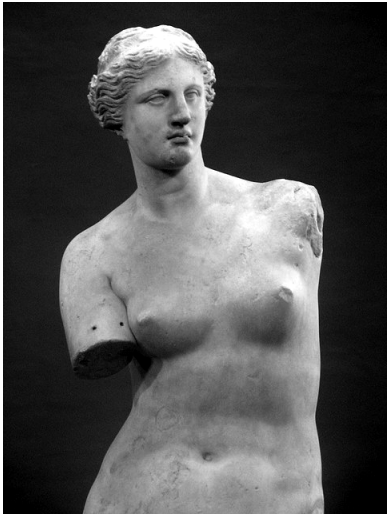


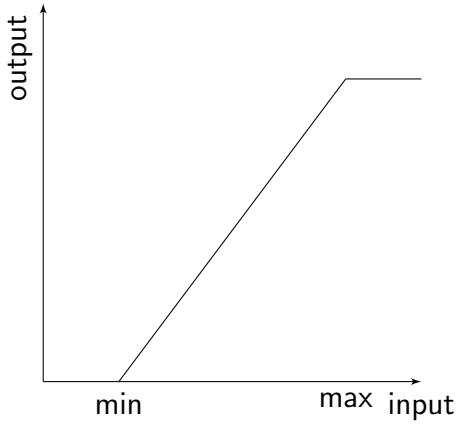


$$g' = \begin{cases} 0 & g < \text{min} \\ \frac{g - \text{min}}{\text{max} - \text{min}} 255 & \text{min} \leq g \leq \text{max} \\ 255 & g > \text{max} \end{cases}$$

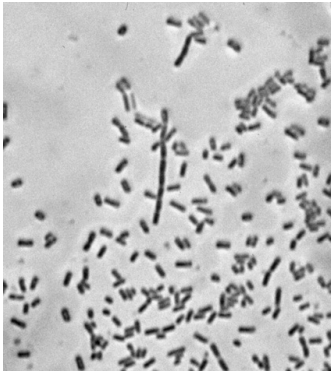
In this case, min $= 50$ and max $= 225$

Min-Max Linear Stretch





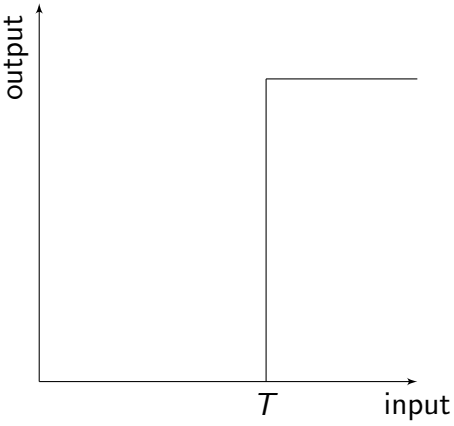
Thresholding



$$g' = \begin{cases} 255 & g \geq T \\ 0 & \text{otherwise} \end{cases}$$

Soil bacteria thresholded at $T = 170$

Thresholding

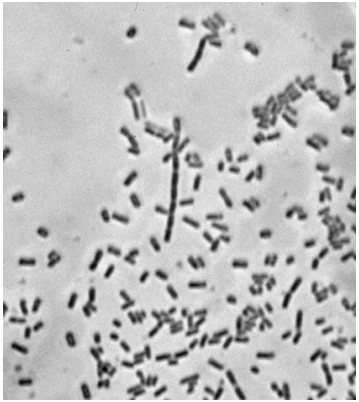


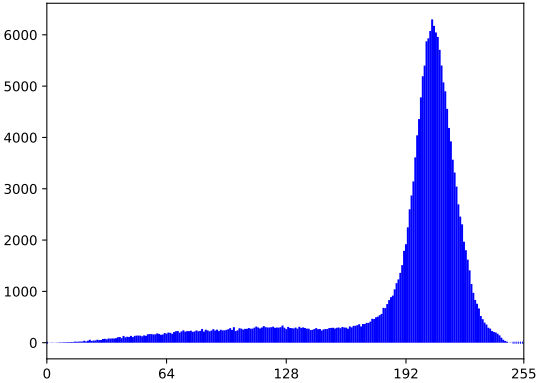
Thresholding

But how do we choose a value for the threshold, T?

Thresholding

Manual selection of T

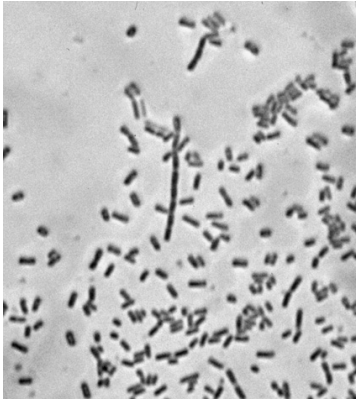


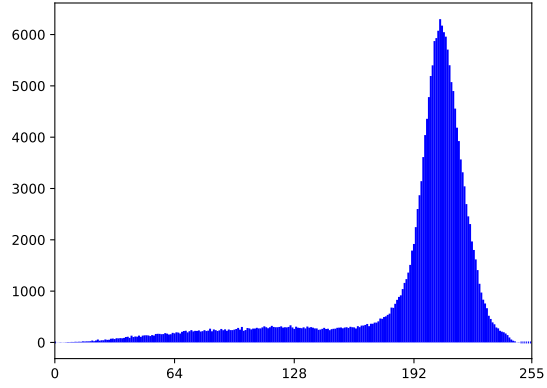


Qu. Where should we choose the threshold?

Thresholding

Manual selection of T





A threshold value of $T = 170$ was chosen after observation of the histogram

Automatic Thresholding

Manual selection of T requires a person to examine the histogram.

How can this be done automatically?

Automatic Thresholding

P-tile (percentile) calculation of T

If we know that an object should occupy a certain percentage of the pixels in an image, we can automatically select T

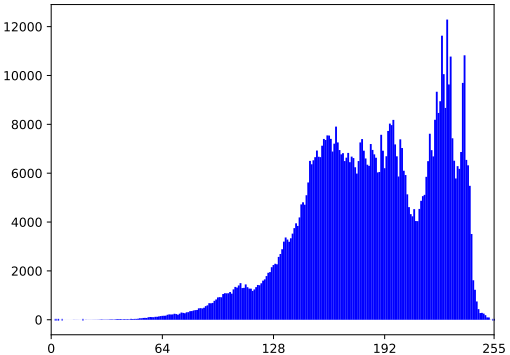
Example: a biscuit on a conveyor line should occupy 41% of the image.

Algorithm

- Calculate the number of pixels that should be object (% of image size)
- Create image histogram
- Accumulate frequencies until total exceeds number of expected object pixels
- Return current grey level as T

Automatic Thresholding

P-tile (percentile) calculation of T



Threshold calculated as $T = 176$ for object occupying 41% of the image

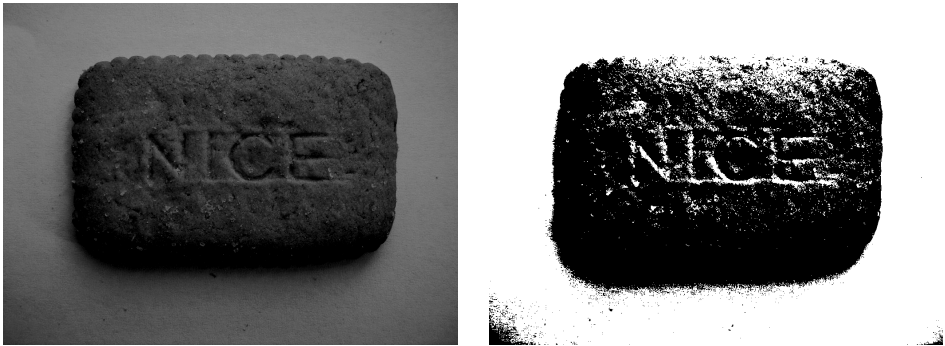
Automatic Thresholding

P-tile (percentile) calculation of T

Threshold calculated as $T = 176$ for object occupying 41% of the image

Automatic Thresholding

P-tile – Biscuit in low light

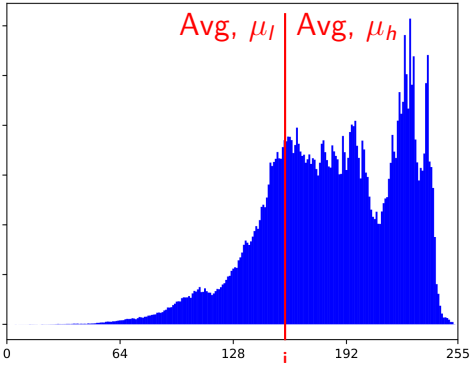


Threshold calculated as $T = 64$ for object occupying 41% of the image

Note Algorithm was not affected by the reduced lighting

Automatic Thresholding

Finding a threshold between the means of lower and upper parts of histogram



Algorithm

For $i \in [1, 255]$):

μ_l = Average intensity below i

μ_h = Average intensity above, and including, i

Midpoint$_i = (\mu_l + \mu_h)/2$

Let $T = i$ where Midpoint$_i$ is closest to i

Automatic Thresholding

Simple method from previous slide

```python
# Read image file (as greyscale, not colour)
filename = sys.argv[1]
img = cv.imread(filename, cv.IMREAD_GRAYSCALE)

# Find histogram - count of each intensity level 0 to 255
hist = cv.calcHist([img],[0],None,[256],[0,256])

min_diff = sys.float_info.max # a large value
T = 0
#
for i in range(1,256):
    w0 = 0
    w1 = 0
    wt0 = 0
    wt1 = 0
    for a in range(0, i):
        w0 += hist[a]
        wt0 += (a * hist[a])
    for b in range(i, 256):
        w1 += hist[b]
        wt1 += (b * hist[b])

    if w0 == 0 or w1 == 0: continue

    mw = 0.5 * ( (wt0/w0) +  (wt1/w1))
    diff = abs(mw - i)
    if diff < min_diff:
        min_diff = diff
        T = i

print('Threshold =', T)
```

Python snippet

Automatic Thresholding

Otsu's method – similar to the previous method

Otsu's method [edit]

The algorithm exhaustively searches for the threshold that minimizes the intra-class variance, defined as a weighted sum of variances of the two classes:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t ,and σ_0^2 and σ_1^2 are variances of these two classes.

The class probability $\omega_{0,1}(t)$ is computed from the L bins of the histogram:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

For 2 classes, minimizing the intra-class variance is equivalent to maximizing inter-class variance.[2]

$$\begin{aligned}\sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2\end{aligned}$$

which is expressed in terms of class probabilities ω and class means μ, where the class means $\mu_0(t)$, $\mu_1(t)$ and μ_T are:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

The following relations can be easily verified:

$$\begin{aligned}\omega_0\mu_0 + \omega_1\mu_1 &= \mu_T \\ \omega_0 + \omega_1 &= 1\end{aligned}$$

The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm.

Algorithm [edit]

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1, \ldots$ maximum intensity
    1. Update ω_i and μ_i
    2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$

Algorithm from Wikipedia – Looks complicated?

MANCHESTER 1824
The University of Manchester

Automatic Thresholding

Otsu's method

```python
# Find histogram - count of each intensity level 0 to 255
hist = cv.calcHist([img],[0],None,[256],[0,256])

hist /= (img.shape[0] * img.shape[1])

max_sigma_squared = 0
T = 0
#
for i in range(1,256):
    sigma_squared = 0
    w0 = 0
    w1 = 0
    wt0 = 0
    wt1 = 0
    for a in range(0, i):
        w0 += hist[a]
        wt0 += (a * hist[a])
    for b in range(i, 256):
        w1 += hist[b]
        wt1 += (b * hist[b])

    if w0 == 0 or w1 == 0: continue

    sigma_squared = w0 * w1 * ( (wt0/w0) - (wt1/w1) )**2

    if sigma_squared > max_sigma_squared:
        max_sigma_squared = sigma_squared
        T = i

print('Threshold =', T)
```

Python code for Otsu's method – Not complicated

Aside – Mathematical Notation

Complicated Mathematics

If you look at an image processing algorithm, it may sometimes look complicated

This is necessary to describe the method precisely

Spend some time working carefully through an algorithm
and it will usually turn out to be simpler than it seems

Automatic Thresholding

Otsu's method

```
24    // Perform thresholding with a supplied threshold value
25    threshold(inputImage, outputImage, 128, 255, THRESH_BINARY);
26    // Save new image
27    imwrite("supplied_threshold.png", outputImage);
28
29    // Perform thresholding with Otsu threshold value
30    threshold(inputImage, outputImage, 0, 255, THRESH_BINARY | THRESH_OTSU);
31    // Save new image
32    imwrite("otsu_threshold.png", outputImage);
```

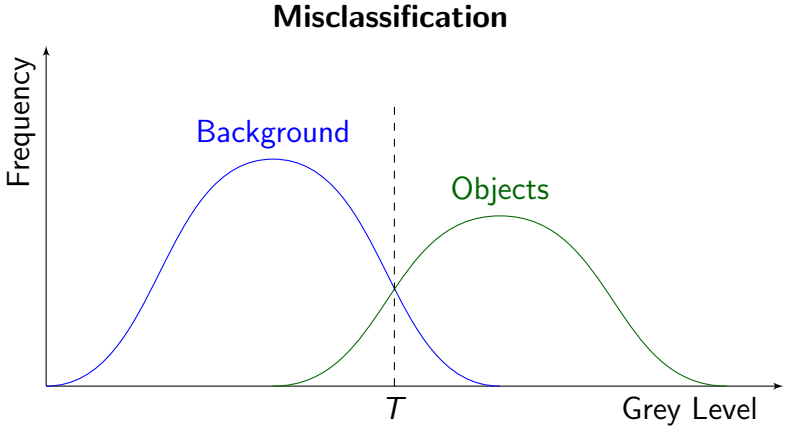
C code for thresholding with Otsu's method in OpenCV

Automatic Thresholding

Using p-tile method (with 41%)

Using Otsu's method

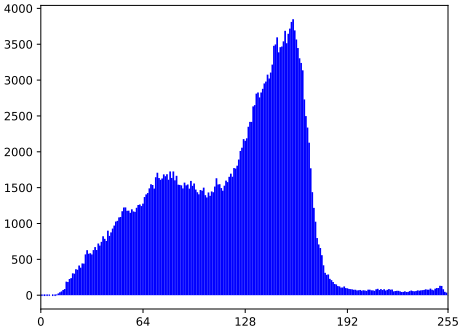
Thresholding



The histograms of background and objects overlap so some background pixels will be classified as objects and vice versa

Thresholding

Misclassification



MANCHESTER 1824
The University of Manchester

COMP27112

Introduction to Visual Computing

Image Processing
Lecture 4.3

Terence Morley
School of Computer Science
The University of Manchester

Geometrical Transformations

Spiralling succulent

Geometrical Transformations

Transformations Demo...

Geometrical Transformations

In the next few slides we will see affine transformations on this image from inside Manchester Museum

Scaling

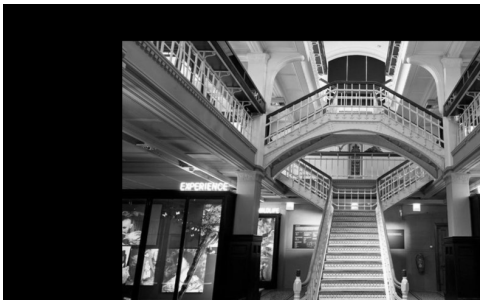
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

e.g. $s_x = 2, s_y = 2$
$(1, 0) \rightarrow (2, 0)$
$(2, 0) \rightarrow (4, 0)$

(We will discuss interpolation later)

Translation



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

$$= \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

e.g. $t_x = 160, t_y = 60$
$(1, 0) \rightarrow (161, 60)$
$(2, 0) \rightarrow (162, 60)$

Scaling and Translation

From the previous two slides we might perform a scaling followed by a translation as follows:

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

$$= \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \end{bmatrix}$$

But this can be achieved with a single matrix multiplication...

Scaling and Translation

Using single matrix multiplication:

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

Or, using **homogeneous** coordinates throughout which will allow us to multiply transformation matrices together:

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

Multiplication of Transformation Matrices

For example, we have a matrix for a translation and one for a scaling

We want to scale and then translate

We can multiply the matrices together (in reverse order) to obtain a single matrix to apply to the image

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Notice that the result is the same as the matrix on the previous slide

OpenCV Functions

Functions for performing geometrical transformations in OpenCV include:

warpAffine()
and
warpPerspective()

The former takes a 2×3 matrix and the latter a 3×3 matrix

Reflection/Flipping

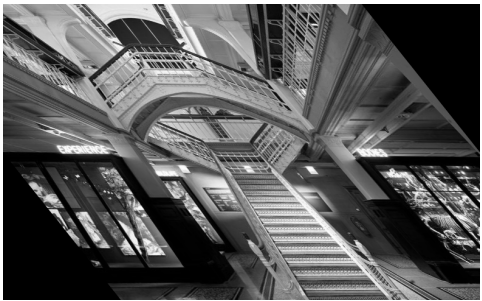
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x \\ -y \\ 1 \end{bmatrix}$$

Note It is usual to translate the image centre to the origin, perform the transformation and then translate back. In this case for an image with height 480:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 480 \\ 0 & 0 & 1 \end{bmatrix}$$

Shear



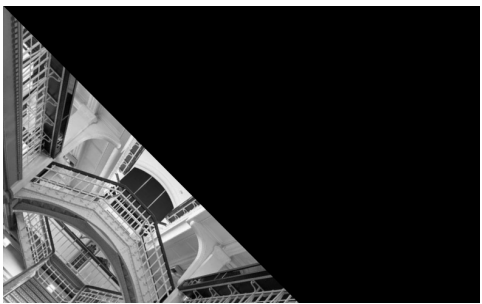
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \lambda_x & 0 \\ \lambda_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x + 0.8y \\ y \\ 1 \end{bmatrix}$$

In this case $\lambda_x = 0.8$, $\lambda_y = 0$

Remember It is usual to create a matrix to translate the image centre to the origin, perform a transformation and then translate back

Rotation



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

For $\theta = 45°$:

$$T = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation

To rotate about image centre, create a matrix that translates centre to origin, rotates and translates back

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Single matrix example for $\theta = 45°$, image $= 640 \times 480$:

$$T = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 263.5 \\ \sqrt{2}/2 & \sqrt{2}/2 & -156.0 \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Transformations

Scale, Shear, Rotate, Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

As we have seen on previous slides, the different matrix entries are responsible for the following transformations:

a, e	$\rightarrow$	scaling
b, d	$\rightarrow$	shearing
a, b, d, e	$\rightarrow$	rotation
c, f	$\rightarrow$	translations

Camera Lens Distortion

Original

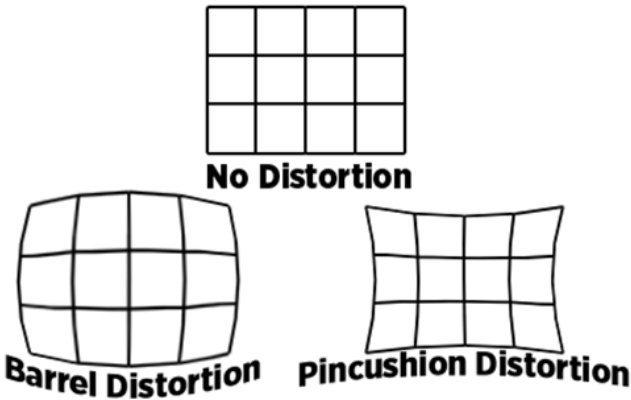
Barrel distortion

Camera Lens Distortion

Original

Pin-cushion distortion

Camera Lens Distortion



Camera Lens Distortion

Radial and Tangential distortions

Radial distortion

(r is the radius from the image centre, assuming here that the origin is the image centre)

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots)$$
$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots)$$

Tangential distortion

$$x_{\text{corrected}} = x + (2p_1 xy) + p_2(r^2 + 2x^2)$$
$$y_{\text{corrected}} = y + p_1(r^2 + 2y^2) + (2p_2 xy)$$

Source: https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html

Camera Lens Distortion

Radial and Tangential distortions

From the previous slide, OpenCV uses the constants in the **distortion matrix**:

$$\begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix}$$

Distortions can be corrected with the OpenCV function

undistort()

Camera Lens Distortion

Lens Distortion Demo...

Wide-angle Lens

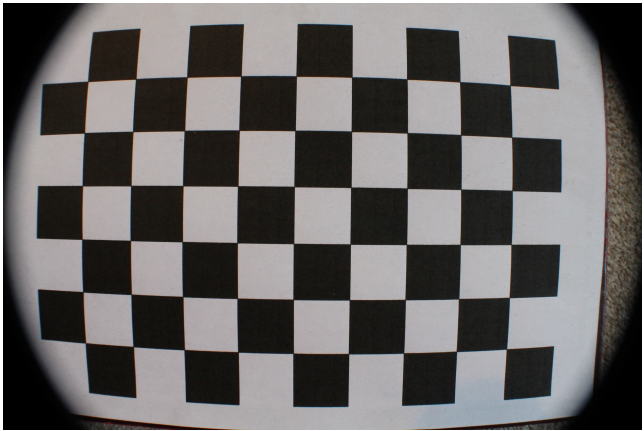
Petronas Towers, Malaysia. Image taken with a wide-angle lens

Fisheye Lens

Barrel distortion from a fisheye lens

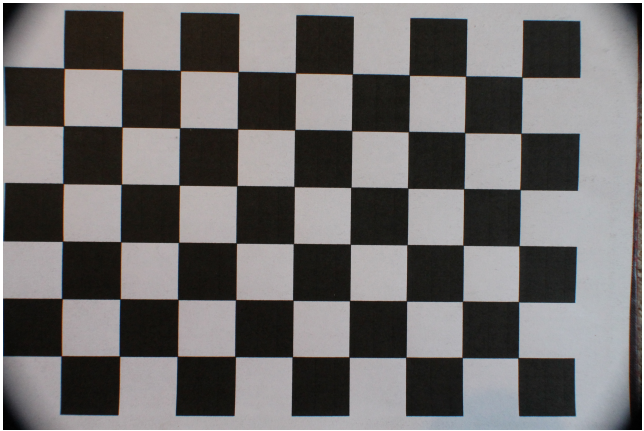
(This was taken with a cheap, low-quality, add-on fisheye lens)

Camera Calibration



The **chessboard pattern** that is used by OpenCV to allow you to correct for camera distortion

Camera Calibration



Chessboard image after calibration
(Used OpenCV to calculate a distortion matrix from multiple images)

Fisheye Lens

Barrel distortion from a fisheye lens

Fisheye Lens

Undistorted using matrix found by the chessboard calibration

MANCHESTER 1824
The University of Manchester

COMP27112

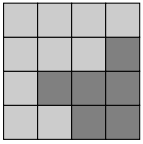
Introduction to Visual Computing

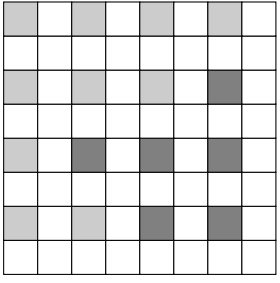
Image Processing
Lecture 4.4

Terence Morley
School of Computer Science
The University of Manchester

Interpolation

Scaling an image by a factor of 2

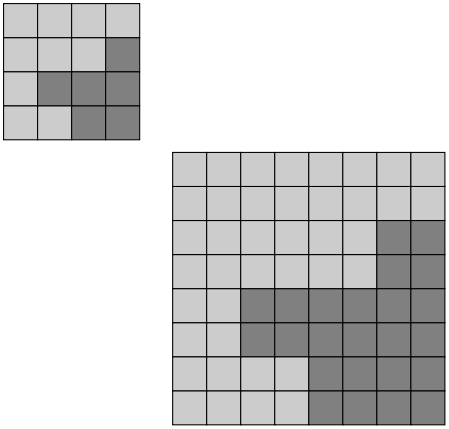




What about the missing pixels?

Interpolation

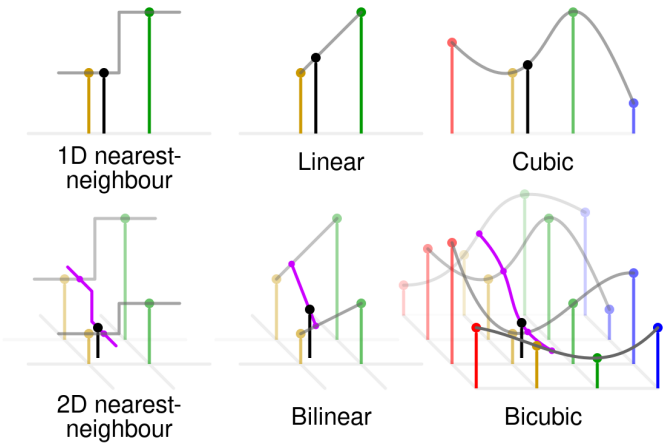
Scaling an image by a factor of 2 - Nearest neighbour interpolation



The result is a bit blocky/jagged.

Interpolation

Some interpolation methods



Forward/Reverse Mapping

In the previous **scaling** example, each pixel in the source image had somewhere to go in the destination image

For a **rotation**, source pixels might not 'land' in a destination pixel (that is, coordinates can be non-integer)

If we round the coordinates, we will create holes in the destination image

Any ideas?

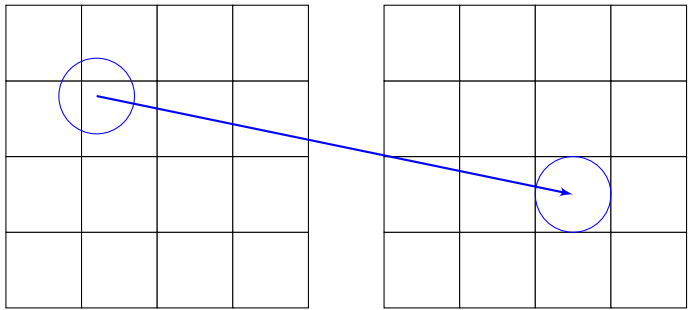
Forward/Reverse Mapping

Solution

Rather than cycling through source pixels and calculating a destination pixel,

cycle through destination pixels and calculate their values from source pixels...

Forward/Reverse Mapping



For each destination pixel, calculate where it came from and then use interpolation to calculate the pixel value

Practical Problem

How might we (find and) level the horizon?

Summary

What have we discussed?

- Image histograms
- Grey-level mapping
- Thresholding
- Geometrical transformations