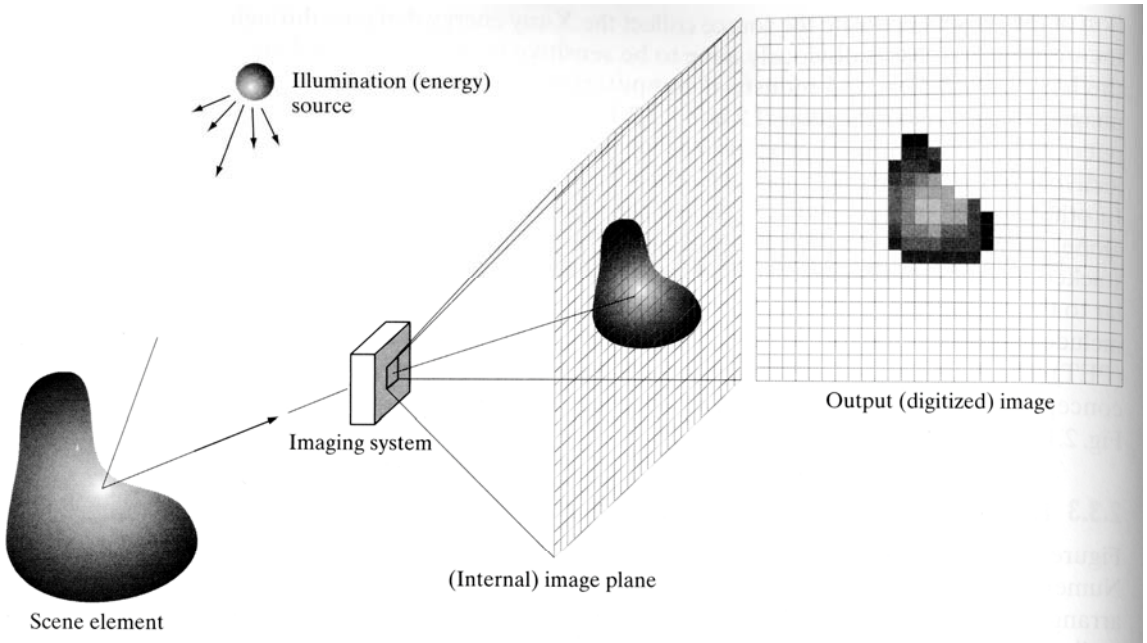
CS_255: Computer Graphics I

Mark W Jones



a b c d e

FIGURE 2.15 An example of the digital image acquisition process. (a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

2.4 Image Sampling and Quantization

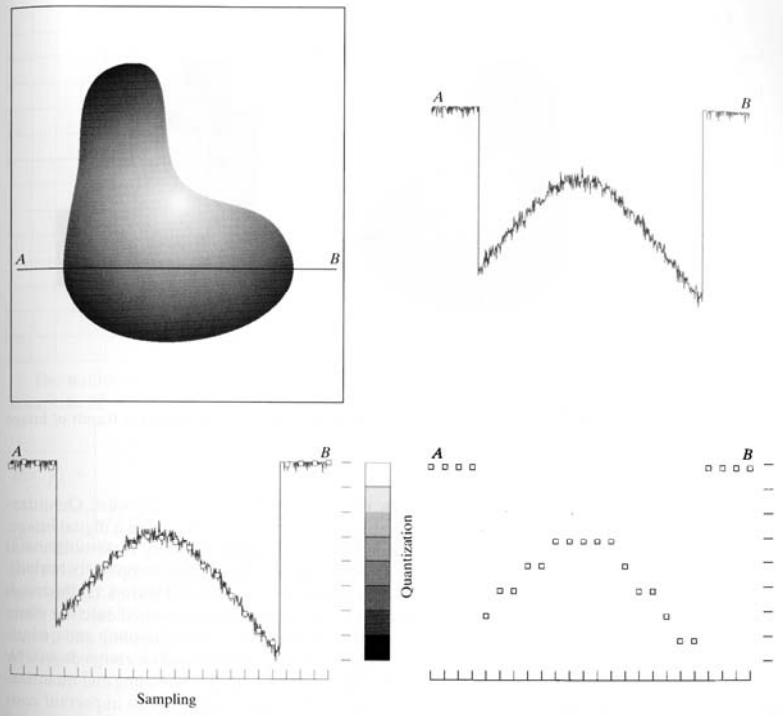
From the discussion in the preceding section, we see that there are numerous ways to acquire images, but our objective in all is the same: to generate digital images from sensed data. The output of most sensors is a continuous voltage waveform whose amplitude and spatial behavior are related to the physical phenomenon being sensed. To create a digital image, we need to convert the continuous sensed data into digital form. This involves two processes: *sampling* and *quantization*.

2.4.1 Basic Concepts in Sampling and Quantization

The basic idea behind sampling and quantization is illustrated in Fig. 2.16. Figure 2.16(a) shows a continuous image, $f(x, y)$, that we want to convert to digital form. An image may be continuous with respect to the x- and y-coordinates, and also in amplitude. To convert it to digital form, we have to sample the function in both coordinates and in amplitude. Digitizing the coordinate values is called *sampling*. Digitizing the amplitude values is called *quantization*.

The one-dimensional function shown in Fig. 2.16(b) is a plot of amplitude (gray level) values of the continuous image along the line segment AB in Fig. 2.16(a). The random variations are due to image noise. To sample this function, we take equally spaced samples along line AB, as shown in Fig. 2.16(c). The location of each sample is given by a vertical tick mark in the bottom part of the figure. The samples are shown as small white squares superimposed on the function. The set of these discrete locations gives the sampled function. However, the values of the samples still span (vertically) a continuous range of gray-level values. In order to form a digital function, the gray-level values also must be converted (*quantized*) into discrete quantities. The right side of Fig. 2.16(c) shows the gray-level scale divided into eight discrete levels, ranging from black to white. The vertical tick marks indicate the specific value assigned to each of the eight gray levels. The continuous gray levels are quantized simply by assigning one of the eight discrete gray levels to each sample. The assignment is made depending on the vertical proximity of a sample to a vertical tick mark. The digital samples resulting from both sampling and quantization are shown in Fig. 2.16(d). Starting at the top of the image and carrying out this procedure line by line produces a two-dimensional digital image.

Sampling in the manner just described assumes that we have a continuous image in both coordinate directions as well as in amplitude. In practice, the method of sampling is determined by the sensor arrangement used to generate the image. When an image is generated by a single sensing element combined with mechanical motion, as in Fig. 2.13, the output of the sensor is quantized in the manner described above. However, sampling is accomplished by selecting the number of individual mechanical increments at which we activate the sensor to collect data. Mechanical motion can be made very exact so, in principle, there is almost no limit as to how fine we can sample an image. However, practical limits are established by imperfections in the optics used to focus on the



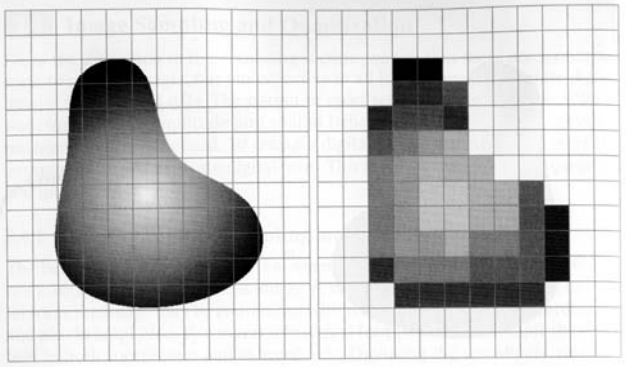
a b c d

FIGURE 2.16 Generating a digital image. (a) Continuous image. (b) A scan line from A to B in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

sensor an illumination spot that is inconsistent with the fine resolution achievable with mechanical displacements.

When a sensing strip is used for image acquisition, the number of sensors in the strip establishes the sampling limitations in one image direction. Mechanical motion in the other direction can be controlled more accurately, but it makes little sense to try to achieve sampling density in one direction that exceeds the

Digital Image Processing 2nd Edition, Rafael Gonzalez and Richard Woods, Prentice Hall, 2002, Chapter 2, pages 50, 52-56.



a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

sampling limits established by the number of sensors in the other. Quantization of the sensor outputs completes the process of generating a digital image.

When a sensing array is used for image acquisition, there is no motion and the number of sensors in the array establishes the limits of sampling in both directions. Quantization of the sensor outputs is as before. Figure 2.17 illustrates this concept. Figure 2.17(a) shows a continuous image projected onto the plane of an array sensor. Figure 2.17(b) shows the image after sampling and quantization. Clearly, the quality of a digital image is determined to a large degree by the number of samples and discrete gray levels used in sampling and quantization. However, as shown in Section 2.4.3, image content is an important consideration in choosing these parameters.

2.4.2 Representing Digital Images

The result of sampling and quantization is a matrix of real numbers. We will use two principal ways in this book to represent digital images. Assume that an image $f(x, y)$ is sampled so that the resulting digital image has M rows and N columns. The values of the coordinates (x, y) now become *discrete* quantities. For notational clarity and convenience, we shall use integer values for these discrete coordinates. Thus, the values of the coordinates at the origin are $(x, y) = (0, 0)$. The next coordinate values along the first row of the image are represented as $(x, y) = (0, 1)$. It is important to keep in mind that the notation $(0, 1)$ is used to signify the second sample along the first row. It does *not* mean that these are the actual values of physical coordinates when the image was sampled. Figure 2.18 shows the coordinate convention used throughout this book.

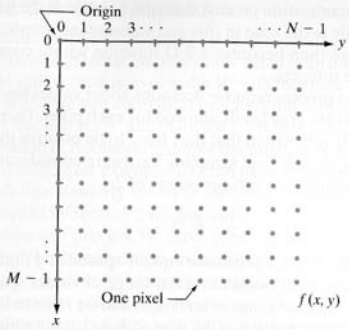


FIGURE 2.18 Coordinate convention used in this book to represent digital images.

The notation introduced in the preceding paragraph allows us to write the complete $M \times N$ digital image in the following compact matrix form:

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}. \quad (2.4\text{-}1)$$

The right side of this equation is by definition a digital image. Each element of this matrix array is called an *image element*, *picture element*, *pixel*, or *pel*. The terms *image* and *pixel* will be used throughout the rest of our discussions to denote a digital image and its elements.

In some discussions, it is advantageous to use a more traditional matrix notation to denote a digital image and its elements:

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix}. \quad (2.4\text{-}2)$$

Clearly, $a_{ij} = f(x = i, y = j) = f(i, j)$, so Eqs. (2.4-1) and (2.4-2) are identical matrices.

Expressing sampling and quantization in more formal mathematical terms can be useful at times. Let Z and R denote the set of real integers and the set of real numbers, respectively. The sampling process may be viewed as partitioning the xy plane into a grid, with the coordinates of the center of each grid being a pair of elements from the Cartesian product Z^2, which is the set of all ordered pairs of elements (z_i, z_j), with z_i and z_j being integers from Z. Hence, $f(x, y)$ is a digital image if (x, y) are integers from Z^2 and f is a function that assigns a gray-level value (that is, a real number from the set of real numbers, R) to each distinct pair of coordinates (x, y). This functional assignment

obviously is the quantization process described earlier. If the gray levels also are integers (as usually is the case in this and subsequent chapters), Z replaces R, and a digital image then becomes a 2-D function whose coordinates and amplitude values are integers.

This digitization process requires decisions about values for M, N, and for the number, L, of discrete gray levels allowed for each pixel. There are no requirements on M and N, other than that they have to be positive integers. However, due to processing, storage, and sampling hardware considerations, the number of gray levels typically is an integer power of 2:

$$L = 2^k. \quad (2.4\text{-}3)$$

We assume that the discrete levels are equally spaced and that they are integers in the interval $[0, L - 1]$. Sometimes the range of values spanned by the gray scale is called the *dynamic range* of an image, and we refer to images whose gray levels span a significant portion of the gray scale as having a high dynamic range. When an appreciable number of pixels exhibit this property, the image will have high contrast. Conversely, an image with low dynamic range tends to have a dull, washed out gray look. This is discussed in much more detail in Section 3.3.

The number, b, of bits required to store a digitized image is

$$b = M \times N \times k. \quad (2.4\text{-}4)$$

When $M = N$, this equation becomes

$$b = N^2 k. \quad (2.4\text{-}5)$$

Table 2.1 shows the number of bits required to store square images with various values of N and k. The number of gray levels corresponding to each value of k is shown in parentheses. When an image can have 2^k gray levels, it is common practice to refer to the image as a "k-bit image." For example, an image with 256 possible gray-level values is called an 8-bit image. Note that storage requirements for 8-bit images of size 1024×1024 and higher are not insignificant.

TABLE 2.1
Number of storage bits for various values of N and k.

N/k	1 (L = 2)	2 (L = 4)	3 (L = 8)	4 (L = 16)	5 (L = 32)	6 (L = 64)	7 (L = 128)	8 (L = 256)
32	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
64	4,096	8,192	12,288	16,384	20,480	24,576	28,672	32,768
128	16,384	32,768	49,152	65,536	81,920	98,304	114,688	131,072
256	65,536	131,072	196,608	262,144	327,680	393,216	458,752	524,288
512	262,144	524,288	786,432	1,048,576	1,310,720	1,572,864	1,835,008	2,097,152
1024	1,048,576	2,097,152	3,145,728	4,194,304	5,242,880	6,291,456	7,340,032	8,388,608
2048	4,194,304	8,388,608	12,582,912	16,777,216	20,971,520	25,165,824	29,369,128	33,554,432
4096	16,777,216	33,554,432	50,331,648	67,108,864	83,886,080	100,663,296	117,440,512	134,217,728
8192	67,108,864	134,217,728	201,326,592	268,435,456	335,544,320	402,653,184	469,762,048	536,870,912

Key Concepts:

The idea of a continuous world represented digitally.

Sampling at pixel locations to obtain amplitude (intensity or colour).

Quantizing into discrete values for digital representation.

Representing within memory as a matrix (array) of the quantized values.

Dynamic range as the number of levels (range of quantized values).

Values as integers in range [0, L-1] where L is usually an integer power of 2 ($L=2^k$)

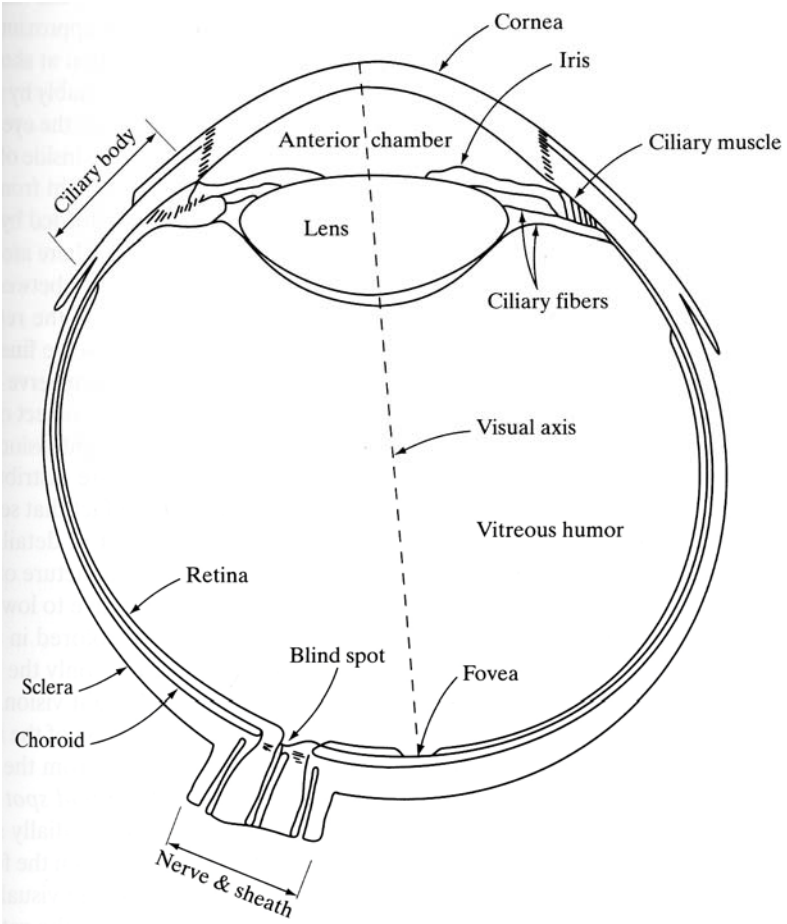
e.g. 1 byte=8 bits gives $256=2^8$ levels.

Black and white images (2 levels) use 1 bit per pixel

Grey level images (where there are 256 grey levels) use 8 bits per pixel

Typical colour images have 8 bits for each of Red, Green and Blue = 24 bits per pixel

The impression of the size of memory required to store images



Human Vision System

"In terms of image acquisition, the eye is totally superior to any camera system yet developed. The retina, on which the upside-down image is projected, contains two classes of discrete light receptors – cones and rods. There are between 6 and 7 million cones in the eye, most of them located in the central part of the retina, called the fovea. These cones are highly sensitive to colour, and the eye muscles rotate the eye so that the image is focused primarily on the fovea. The cones are sensitive to bright light and do not operate in dim light. Each cone is connected, by its own nerve, to the brain.

There are at least 75 million rods in the eye distributed across the surface of the retina. They are sensitive to light intensity but not colour. They share nerve endings and only serve to give an overall picture of the image. These work in dim and bright light conditions, but as they are the only sensors operating in dim light, only a monochromatic, multi-grey-level image is obtained when it is dark.

The range of intensities to which the eye can adapt is of the order of 10^{10}, from the lowest visible light to the highest bearable glare. In practice the eye performs this amazing task by altering its own sensitivity depending on the ambient level of brightness. While it can cope with this wide range of light levels, it cannot simultaneously discriminate between different dim and different bright levels. Given one job or the other it can cope, but not at the same time.

There is considerable evidence to suggest that the subjective brightness (i.e. the brightness perceived by the eye) is a logarithmic function of intensity. Thus a small increase in dim light intensity is perceived as equal to a large increase in bright light intensity. This suggests that when setting up pallets for grey level displays the number of shades with grey level values in the lower half of the range should be greater than the number of shades with grey level values in the upper half of the range."

extract from Introductory Computer Vision and Image Processing, Adrian Low, McGraw Hill, 1991, section 2.4, page 13,

Image credit: Digital Image Processing 2nd Edition, Rafael Gonzalez and Richard Woods, Prentice Hall, 2002, Chapter 2, (page 35, figure of cross section from human eye from), (page 37, distribution of rods and cones in the retina)

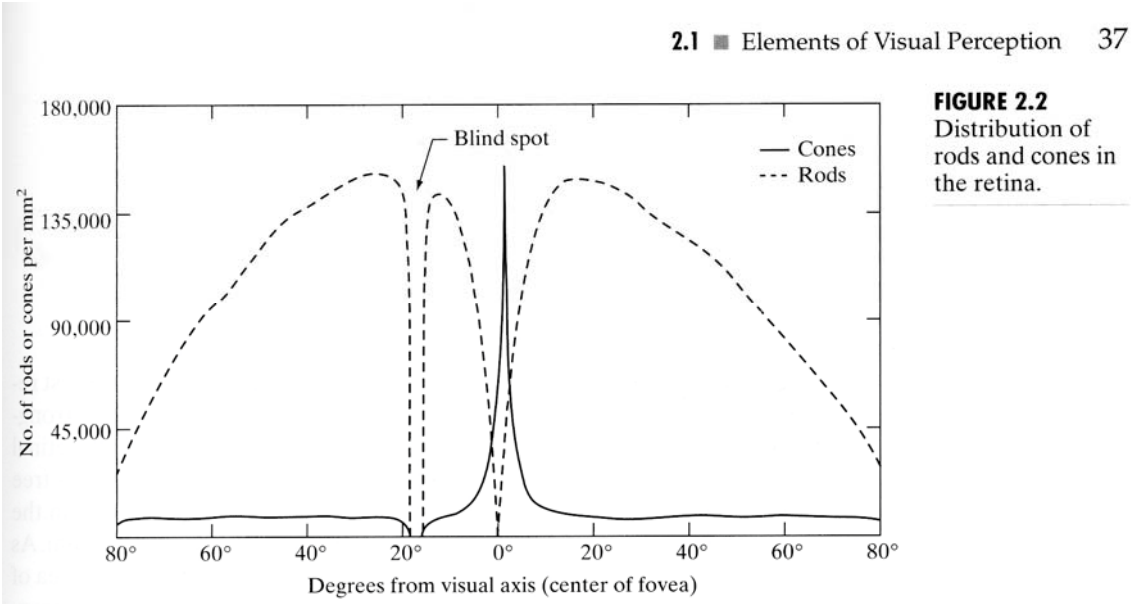


FIGURE 2.2 Distribution of rods and cones in the retina.

Human Vision System

- Cones – colour
- Rods – intensity
- High intensity range – 10^{10}, achieved by sensitivity adaption
- Logarithmic sensitivity to light

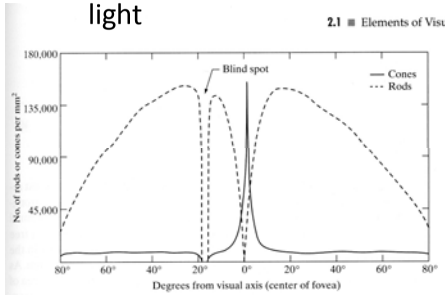


FIGURE 2.2 Distribution of rods and cones in the retina.

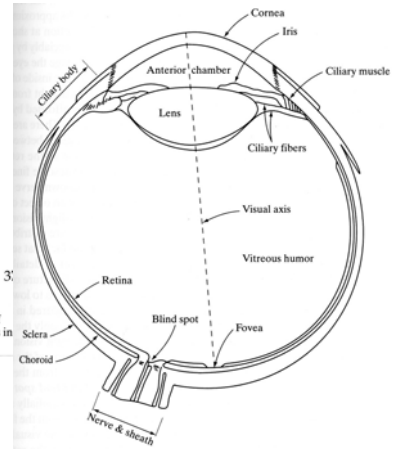


Image credit: Digital Image Processing 2nd Edition, Rafael Gonzalez and Richard Woods, Prentice Hall, 2002, Chapter 2, (page 35, figure of cross section from human eye from), (page 37, distribution of rods and cones in the retina)

Distributing Intensity Levels

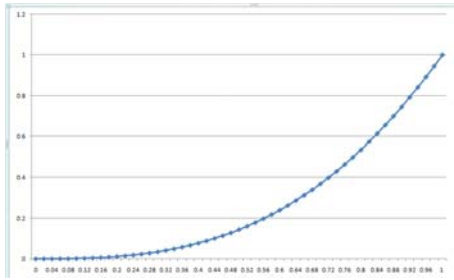
- Logarithmic sensitivity – e.g. perceived brightness increase of 20w-22w is same as 100w-110w
- Ratio important (in above example r=1.1)
- To display n+1 intensity levels (I_0-I_n) we should have: $I_1/I_0=I_2/I_1=...=I_n/I_{n-1}=r$ (the ratios, r, of each successive intensity are equal)
- Therefore, $I_k=r^kI_0$ and since $I_n=1.0$, $r=(1.0/I_0)^{1/n}$

Distributing Intensity Levels

- e.g. (from book)
- For 3 levels, and $I_0=1/8$, r=2
- $I_1=rI_0=1/4$
- $I_2=r^2I_0=1/2$
- $I_3=1.0$

Gamma Correction

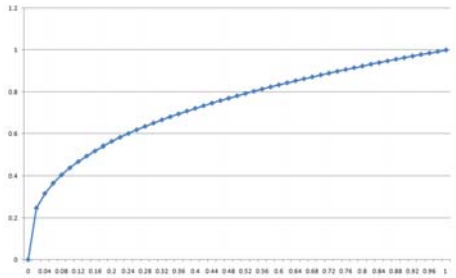
- Monitor response curve:
- $I=aV^{\gamma}$
- e.g. PAL specification has gamma=2.8



- RGB=(64, 64, 64) should be half the intensity of (128, 128, 128), but under PAL is not.
- Need gamma correction

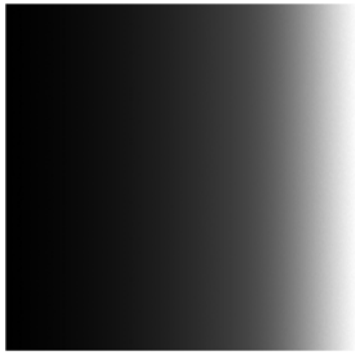
Gamma Correction Equation

- $V=(I/a)^{1/\gamma}$ (assume a=1 for this course)
- Implementation: divide pixel by 255 (get I, a number between 0 and 1)



- Find power(I, 1.0/gamma)
- Multiply by 255 to get new intensity
- (Do for each colour channel, for each pixel)

Example



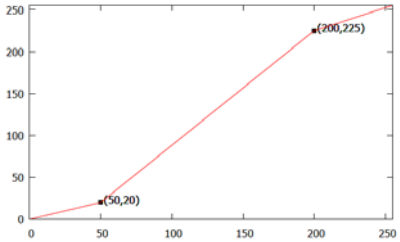
- Input: Grey Scale
- Monitor displays
- This matches gamma=2.8, so correct with gamma=2.8 (i.e. $I=V^{1/2.8}$)

Contrast Stretching

- Is a Piecewise Linear Transformation Function
- Gamma correction is a non-linear transformation function
- Aims to increase the apparent dynamic range of an image by increasing contrast in some intervals of the domain and decreasing in others
- Transform requires user input consisting of 2 points (r_1, s_1) and (r_2,s_2) that control the transformation

Contrast Stretching

Example transform



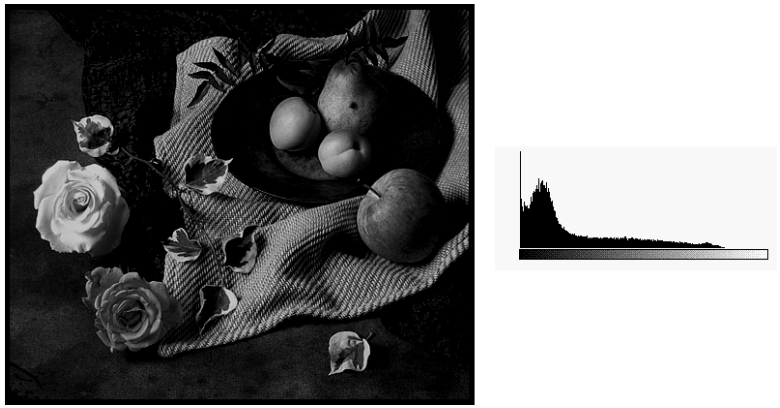
- Levels below 50 and above 200 are compressed to allow more levels between 50 and 200

Equation

- $out = \begin{cases} \frac{s_1}{r_1} \times in & if\ in < r_1 \\ \frac{(s_2-s_1)}{(r_2-r_1)} \times (in - r_1) + s_1 & if\ r_1 \le in \le r_2 \\ \frac{(255-s_2)}{(255-r_2)} \times (in - r_2) + s_2 & if\ in \ge r_2 \end{cases}$
- Given the input value, in, the output value, out, is calculated using the above equation.
- Calculate for each colour channel.

Histograms

- A histogram is a graph representing number of pixels at each intensity level



Histograms

- We can make non-uniformly brighter using gamma correction, or uniformly brighter by adding constant intensity to each pixel

Histogram Calculation

```
int[][] histogram;

histogram=new int[256][3];

for (j=0; j<h; j++)
   for (i=0; i<w; i++)
      for (c=0; c<3; c++)
         histogram[data[c+3*i+3*j*w]&255][c]++;

for (i=0; i<256; i++)
    System.out.println(i+" r="+histogram[i][2]+
                       " g="+histogram[i][1]+
                       " b="+histogram[i][0]);
```

pixel value

Histograms

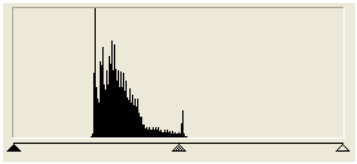
Typical Output

```
233 r=790 g=491 b=566
234 r=783 g=672 b=1214
235 r=704 g=796 b=1308
236 r=577 g=970 b=227
237 r=542 g=982 b=1278
238 r=614 g=1045 b=190
239 r=614 g=1093 b=1134
240 r=572 g=1065 b=240
241 r=700 g=844 b=828
```

- e.g. 790 pixels have a red value of 233, 491 pixels have a green value of 233, and 566 pixels have a blue value of 233

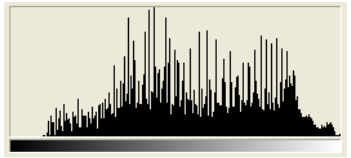
Histograms

- Reduced contrast in Photoshop
- Displayed histogram (combined RGB into grey level image)



Histograms

- Used Histogram Equalization
- Note how the histogram is more spread out



Histogram Equalization

- Contrast can be improved by spreading distribution of intensity evenly over the pixels
- Total number of pixels in image is height*width (h*w)
- Number of grey levels is g_levels
- Ideal histogram is flat
- Ideal number of pixels at each level:
- p_num=(h*w)/g_levels

Histogram Equalization (code)

For grey level images only:

```
int[] histogram, mapping;
int red, green, blue, grey, t_i=0;

int g_levels=256;

histogram=new int[g_levels];
mapping=new int[g_levels];
```

Histogram Equalization

- Mapping F is defined as:
- *If t(i) is the actual number of pixels at all old grey levels up to and including grey level i, then*

F(i)=max{0,round((g_levels*t(i))/(h*w))-1}

t(i) is known as the cumulative distribution function

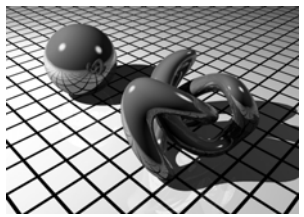
Histogram Equalization

- In code (after making image grey):

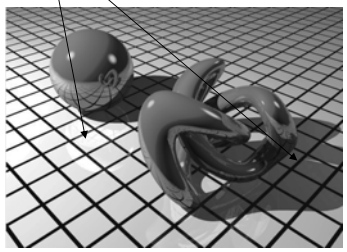
```
for (i=0; i<g_levels; i++) {
  t_i+=histogram[i];
  mapping[i]=
max(0,Math.round((g_levels*t_i)/(h*w))-1);
}
```

Histogram Equalization

110 115
111 116
112 116
113 117
114 117
115 118
116 119
117 119
118 120
119 121
120 121
121 122
122 122
123 123
124 124
125 124

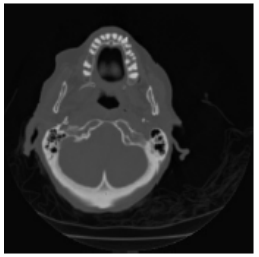


- e.g. old grey pixels of value 112 map to new grey pixel of value 116
- see contrast in shadows and reflections



Histogram Equalization

- Lowering the dynamic range in images
- e.g. 12 bit CT scanner – display on 8 bits?
- naïve – remove lowest 4 bits



- Better – use histogram equalization with g_levels=max_intensity-min_intensity=3365 (for this CT head)

Colour Histogram Equalization

- Convert RGB to HSV
- Build histogram on V (brightness)
- Equalize V
- Convert HSV back to RGB
- Display

Histogram Equalization

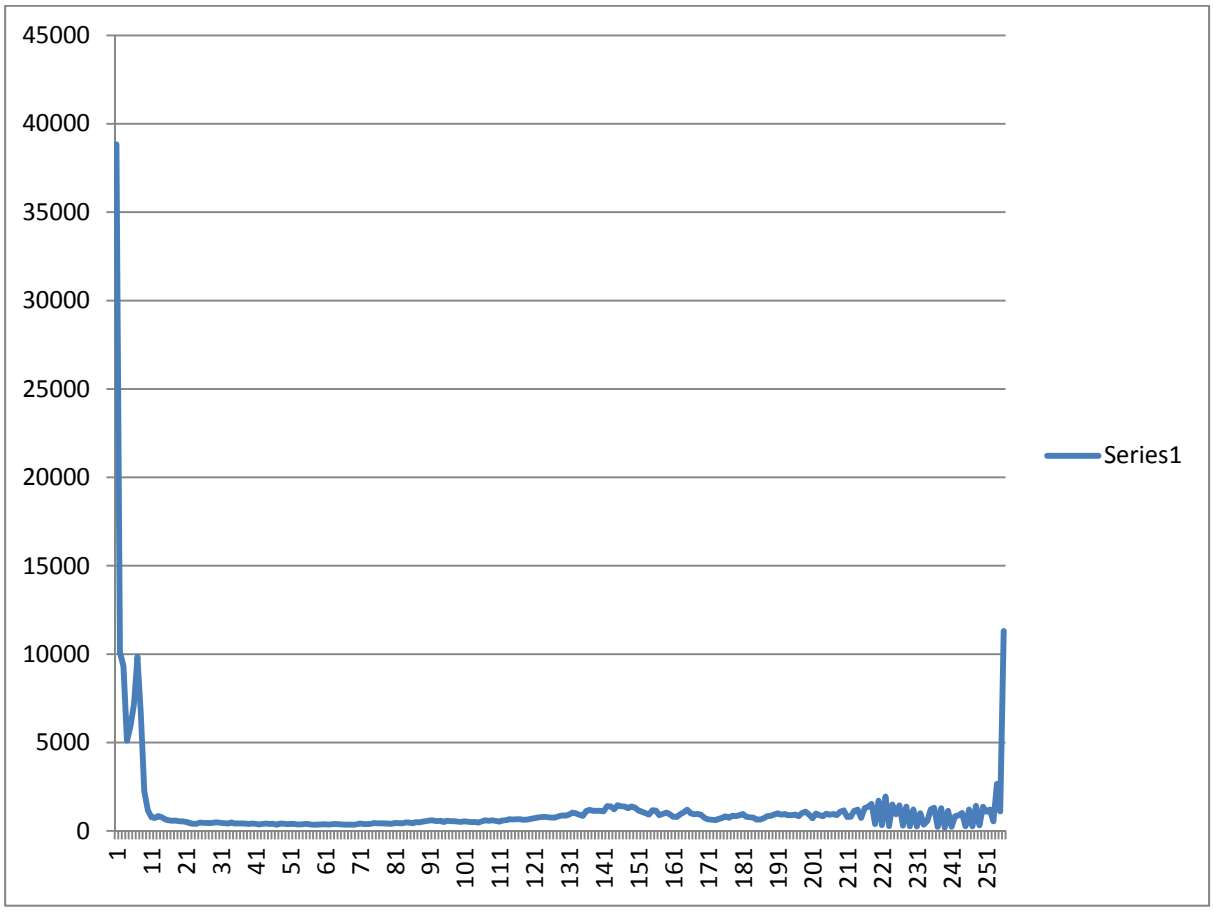
Stage 1 Create the histogram

For a single colour channel from an image with intensities from 0 to 255, there will be 256 values in the histogram (and the other arrays we will create in the next stages).

The histogram is an array that counts how many elements of the single colour channel of the image have that value.
So histogram[v]=p indicates p pixels have the value v, where v takes values from 0 to 255.

To create a histogram, loop through every pixel of the image, obtaining the intensity value, v, of the colour channel in question, and doing
histogram[v]++;
(make sure you have previously initialised histogram[i] to zero for all i.)

Here is an example output. This is the histogram from the red channel of the assignment image.



Stage 2 Create the cumulative distribution function t

t[i] is the number of pixels in the colour channel that have values up to and including i.

So

t[0]=histogram[0]

t[1]=histogram[0]+histogram[1]=t[0]+histogram[1]

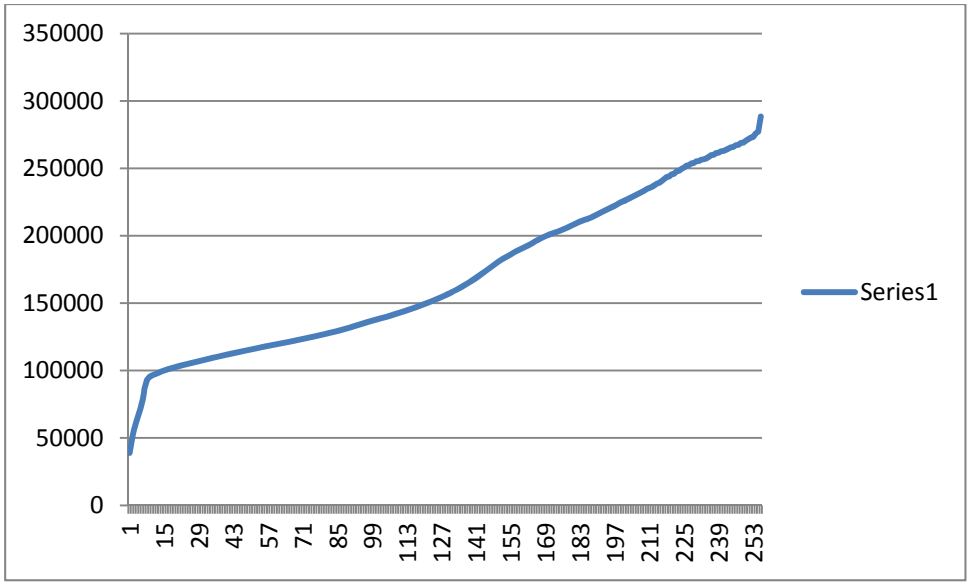
t[2]=histogram[0]+histogram[1]+histogram[2]=t[1]+histogram[2]

In general

t[n]=t[n-1]+histogram[n]

i.e. t can be calculated using a single loop.

The next graph is a plot of the t[i] function from the above histogram. You can see that as we encounter a peak, the cumulative function jumps up. As we encounter lots of lower values, the function gets steadily larger. For those who have done AS/A maths, this function at i is the integral of the histogram from 0 to i.



Here are the last few values printed to the screen:

268879

269198

270560

271650

272859

273401

276060

277169

288470

Note that the image size, Size=634x455=288470. The last number should be the total number of elements in the data set as all elements in the data set have values up to and including the maximum value.

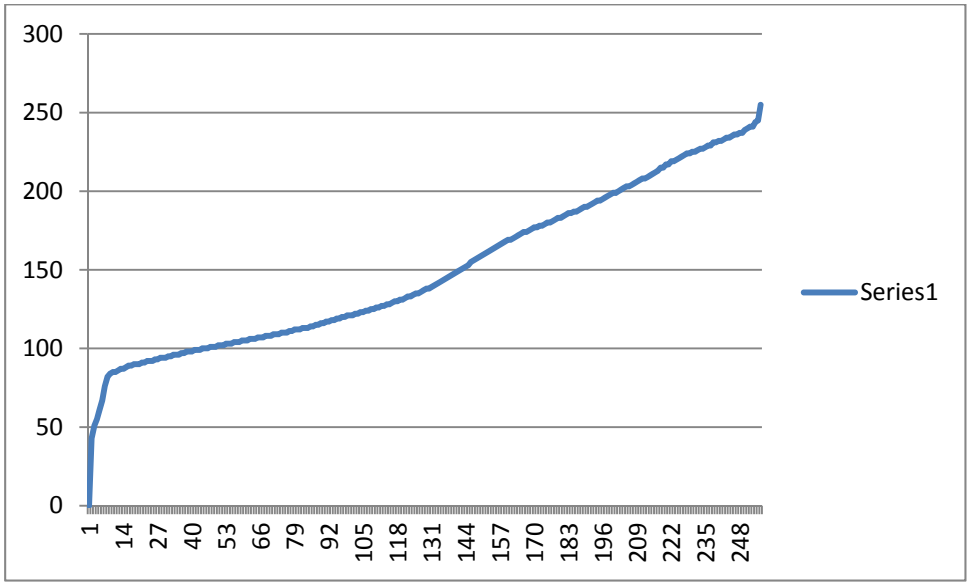
Stage 3 Create the mapping

We need to map the elements of the data set to a new range $[i_1, i_2]$:
mapping[i]=((i2-i1)*(t[i]/Size))+i1
Where we want to map **to** an image with $i_1=0$ and $i_2=255$, this becomes:
mapping[i]=255.0*(t[i]/Size), where Size is the number of pixels.

Stage 4 Create the image

Loop through the image as usual. Get the value from the appropriate colour channel as usual and find its mapping to new intensity:
new_intensity=mapping[old_intensity]
And assign new_intensity to each of the rgb values of the pixel as usual to get a new grey scale, equalised image.

Here is the plot of the mapping function. You can see that given an input data value, i, mapping[i] gives an intensity between 0 and 255.



Stages 2 and 3 combined

Since mapping[i] only requires t[i] **and** t[i] only requires t[i-1] and histogram[i], it is possible to combine stages 2 and 3.

Grey Scale and Colour Images

All of the above refers to getting the intensity from one colour channel, performing histogram creation and equalisation and copying the mapped value back to 3 colour channels to produce a grey scale image.

To change a colour image to grey scale, you should rather take the red, green and blue values, sum them and divide by 3. This gives you a single brightness value that you can do the above to, and then copy into the 3 colour channels to give a new equalised grey scale image.

To perform histogram equalisation on a colour image (not required, but some of you may fancy a challenge), you need to get some code to convert RGB to HSV. Then use the Value channel to create the histogram and do equalisation to create a new Value channel. Then convert that new HSV back to RGB. This is probably how Photoshop carries out colour histogram equalisation.

Spatial Linear Filtering - Why?

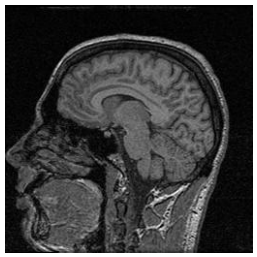
- Image processing
- Remove noise from images (e.g. poor transmission (from space), measurement (X-Rays))

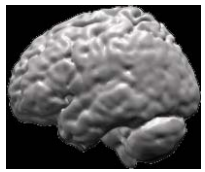
Original Image

Corrupted Image

Filtered Image

Edge Detection

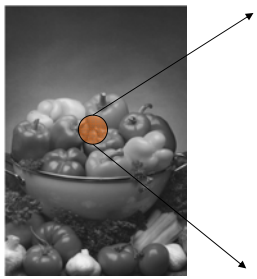


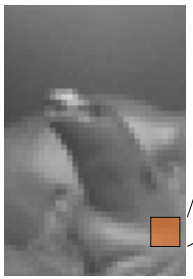


- e.g. could digitally extract brain for display
- could check if machined parts are correct

How does it work?

- Lets use a grey image for ease
- Each pixel has a single value between 0 (black) and 255 (white)





$$\begin{bmatrix} 100 & 110 & 120 & 130 & 140 \\ 120 & 120 & 130 & 130 & 140 \\ 130 & 140 & 150 & 150 & 160 \\ 140 & 150 & 150 & 160 & 170 \\ 150 & 160 & 170 & 180 & 190 \end{bmatrix}$$

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

$$\begin{bmatrix} 100 & 110 & 120 & 130 & 140 \\ 120 & 120 & 130 & 130 & 140 \\ 130 & 140 & 150 & 150 & 160 \\ 140 & 150 & 150 & 160 & 170 \\ 150 & 160 & 170 & 180 & 190 \end{bmatrix}$$

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

100	110	120	130	140
120	120	130	130	140
130	140	150	150	160
140	150	150	160	170
150	160	170	180	190

Sum=1x100+3x110+1x120+
3x120+9x120+3x130+
1x130+3x140+1x150
= 3080

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

100	110	120	130	140
120	120	130	130	140
130	140	150	150	160
140	150	150	160	170
150	160	170	180	190

Sum=1x110+3x120+1x130+
3x120+9x130+3x130+
1x140+3x150+1x150
= 3260

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

100	110	120	130	140
120	120	130	130	140
130	140	150	150	160
140	150	150	160	170
150	160	170	180	190

Sum=1x120+3x130+1x140+
3x130+9x130+3x140+
1x150+3x150+1x160
= 3390

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

100	110	120	130	140
120	120	130	130	140
130	140	150	150	160
140	150	150	160	170
150	160	170	180	190

Sum=1x120+3x120+1x130+
3x130+9x140+3x150+
1x140+3x150+1x150
= 3450

Filter Kernel

- Now select a filter kernel
- e.g. Gaussian blur (removes noise)
- For each pixel in the image
- Can the kernel be centred over the pixel?
- If so, calculate the sum of the products

1	3	1
3	9	3
1	3	1

100	110	120	130	140
120	120	130	130	140
130	140	150	150	160
140	150	160	160	170
150	160	170	180	190

Result

*	*	*	*	*
*	3080	3260	3390	*
*	3450	3620	3740	*
*	3720	3870	4060	*
*	*	*	*	*

Cross Correlation

In principle, given a 3×3 subimage of pixels:

$$I_{ij} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

and a 3×3 filter kernal:

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

operating on pixel p_{22}, the new value p'_{22} is given by:

$$p'_{22} = m_{11}p_{11} + m_{12}p_{12} + \cdots + m_{32}p_{32} + m_{33}p_{33}$$

*What about the edges?

- * indicates filter could not be centred
- 3x3 = 1 pixel edge all the way around
- 5x5 = 2 pixels edge all the way around
- nxm = ? Exercise!

*	*	*	*	*
*	3080	3260	3390	*
*	3450	3620	3740	*
*	3720	3870	4060	*
*	*	*	*	*

*What about the edges?

- Reduce size of image – but could be counter intuitive for users.
- Put a single colour border or pad the image beforehand.
- Use old pixel values (not sensible for e.g. edge detectors)
- Use known neighbours colour.
- Use some fudge (e.g. have some filters reduced in size so they can fit against the edges when needed)

*	*	*	*	*
*	3080	3260	3390	*
*	3450	3620	3740	*
*	3720	3870	4060	*
*	*	*	*	*

Aren't those numbers too big?

- Yes, they are no longer in the range 0-255
- We could have negative numbers (for a high pass filter)
- So we need to map the new numbers onto our range 0-255
- First calculate the max and min values
- max=4060, min=3080
- Then for each intermediate value I, calculate **I'=((I-min)*255)/(max-min)**

*	*	*	*	*
*	0	47	81	*
*	96	141	172	*
*	167	206	255	*
*	*	*	*	*

Normalisation

- The last step:
- **I'=((I-min)*255)/(max-min)**
- Is called *Normalisation*
- As used, it maps pixels values to the range 0..255. Pixels could be mapped to other ranges (e.g. You might have a 12 bit display, or you might want to map back to the same range the image currently uses)

And colour?


Calculate sum of products for each colour channel independently Normalise as before, but replace red with new red, etc.

140	130	150	128	130
130	190	130	120	120
120	130	150	140	145
147	168	165	162	160
150	170	167	167	165

120	130	130	108	110
130	195	87	80	78
90	85	81	82	81
80	82	85	86	89
92	95	100	101	102

14	35	15	12	13
32	91	13	12	12
12	33	15	14	14
20	21	22	23	26
19	18	16	20	21

What about the filters?

1	1	1
1	1	1
1	1	1

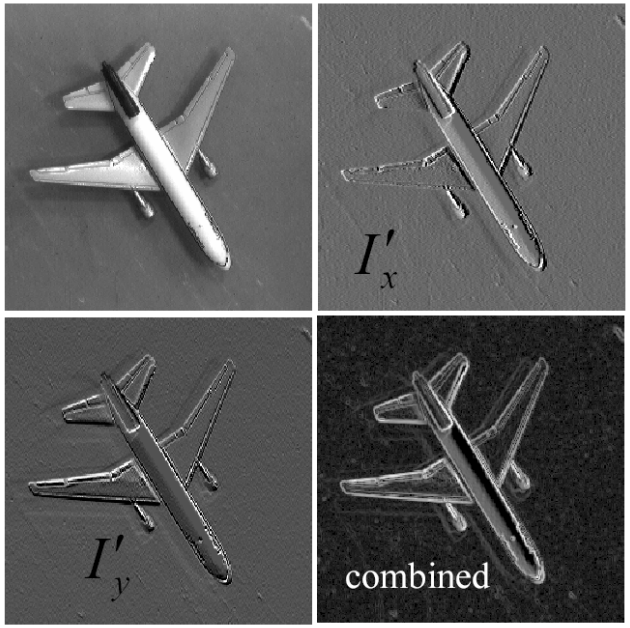
-1	-1	-1
-1	8	-1
-1	-1	-1

- Low pass 3x3 box filter (nxm box filter has size nxm with all 1's). The bigger the filter, the larger the blur (and time – think about that)
- High pass filter (like *sharpen* in Photoshop

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

- Sobel edge detectors – X-direction and Y-direction



Low Pass Filter - Gaussian

7x7

1	4	7	10	7	4	1
4	12	26	33	26	12	4
7	26	55	71	55	26	7
10	33	71	91	71	33	10
7	26	55	71	55	26	7
4	12	26	33	26	12	4
1	4	7	10	7	4	1

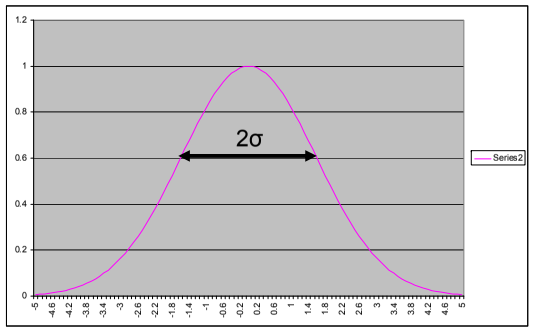
3x3

1	3	1
3	9	3
1	3	1

Where do these numbers come from?

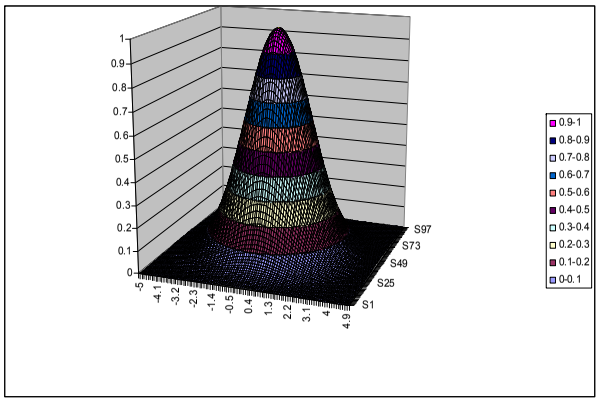
Gaussian

- 1 dimensional
- $G(x)=e^{-(x^2/2\sigma^2)}$
- In Excel power(exp(1), -(x*x/2*σ*σ))



Gaussian

- 2D
- $G(x,y)=e^{-((x^2+y^2)/2\sigma^2)}$



1 Low-Pass Filters

The aim of low-pass filters is to *average* out local differences in order to remove *noise* from the image. It is also considered to be a *smoothing* process. Another low-pass filter is shown below. In this filter all elements are set to 1, which means that a pixel does become an average of itself and all of its neighbours. Noise occurs when an image is the result of some capturing process, e.g. a planetary probe taking a photo and radioing it back to Earth; there will be some loss of signal, and in those places erroneous values will be filled in. The lower the *signal to noise ratio*, the more errors that appear in the image. By applying a low-pass filter these pixel values are adjusted according to those of their neighbours, and the large jumps due to errors are reduced. Low-pass filters remove high frequencies in an image (sudden changes in pixel values) while retaining (or passing through) the low frequencies (gradual changes in pixel values). See figure 1 for an example of a low-pass filter.

1	1	1
1	1	1
1	1	1

Figure 1: An example of a low-pass filter - the box filter.

2 Edge Detection

High-Pass filters are designed to do the reverse. Their intended use is to accentuate small scale structures such as edges. They enhance edges and remove gradual changes. In designing a high-pass filter we must take into account the fact that we wish to enhance pixels that are different to their neighbours. This is best accomplished by having negative contributions from pixels surrounding the central pixel under the mask as the mask moves pixel-by-pixel through the image. See figure 2 for an example of a high-pass filter.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 2: An example of a high-pass filter.

Another example would be the popular edge detection filters known as the Sobel Edge Detection filters (figure 3). The 3×3 filters are normally given as:

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Figure 3: Sobel edge detection filters (a) X-direction. (b) Y-direction.

The X-direction filter is used for detecting vertical edges, and the Y-direction filter is used for detecting horizontal edges.

The larger the filter used, the more contribution the surrounding pixels make to the final image. It is useful to use these 3×3 matrices when detecting edges, since smaller matrices would fail to find edges in noisy images, or where change between one area and another is gradual.

The results of using both filters are added together and then thresholding is applied to obtain the edges in the image.

3 Aside : Thresholding

Thresholding normally refers to setting all the grey levels below a certain level to zero; or above a certain level to a maximum brightness level. The maximum brightness will be 255 on an 8-bit grey-scale.

```
for row := 1 to rowmax do
        for col := 1 to colmax do
                if (image[row][col]>threshold) then
                        image[row][col]=MAX;
                else
                        image[row][col]=MIN;
```

An example of its use would be in *optical character recognition*. If some document is scanned in, then it is easier to recognise text if it is black on a white background. By assumming the text will be darker than some threshold value, all values lighter than this are set to white, and all value darker than this are set to black using the above code.

4 Embossing

The filter in figure 4 has the effect of *embossing* the image.

-1	-1	-1
-1	1	1
-1	1	1

Figure 4: Filter for embossing an image.

It works by removing gradual changes, or uniform areas of colour. Edges are enhanced to give some feeling of depth. This is achieved by making some edges brighter and some edges darker. The result is something that looks as if it has been embossed onto the background, hence the name. Embossing is usually used for effects in images and documents - see xv's emboss function.

Spatial Linear Filtering

- Correlation, as introduced, is a form of spatial linear filtering
- An image of size MxN is filtered with a filter mask of size mxn

$$g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)\, f(x+s, y+t)$$

- w is the kernel filter (e.g. 3x3 box matrix)
- Assume odd matrix size, and 0,0 is the centre
- Therefore the above expression loops over the filter

Image Credit: All images (unless otherwise noted) in these 2 lectures are by Jason Xie (Swansea)

Spatial Linear Filtering

$$g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)\, f(x+s, y+t)$$

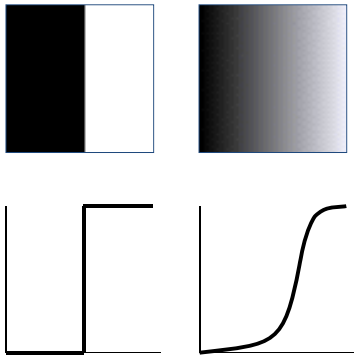
- to describe the equation: e.g. in pseudo code

```
for (s=-a; s<=a; s++) {
      for (t=-b; t<=b; t++) {
            sum=sum+w[s][t]*f[x+s][y+t];
      }
}
```

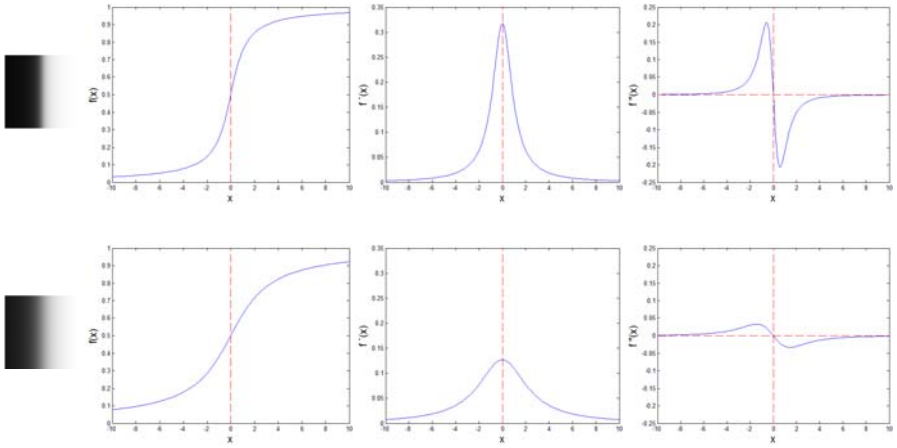
- where w is a 2D filter array, and f is a 2D image (does not take into account colour)

Edge Detection

- What are edges?
- Usually a perceivable change in intensity
- e.g. see the abrupt change on the left (graphed bottom left)
- Compared to the change on the right (graphed bottom right)
- Can we create edge detectors?
- Edges can be spotted where the "rate of change" of intensity is high.
- Recall "rate of change" is the derivative of a function



Edge Detection: Derivatives



Edge profile | 1st order derivative | 2nd order derivative

Edge Detection: Derivatives

- In the previous images, the vertical dotted line is the perceived edge
- This is difficult to compute on the edge profile
- Its where the rate of change of intensity is highest
- We can see this in the 1st derivative where the function has a maximum value
- This is harder to spot for smoother edges
- The maximum of a function can be found where its derivative equals zero
- Therefore see the 2nd derivative, where the edge is obvious as the function crosses from +ve to -ve

Edge Detection: Derivatives

- Edges can be detected by considering the first derivative (2nd column of graphs)
- Notation, given a function f its derivative is denoted f'
- Defining, using limits,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- that is, as h tends toward zero (limit definition), the right hand side defines the derivative.

Edge Detection: Derivatives

- The derivative can be approximated by evaluating the function with a small h
- But, if h does not tend to zero, the derivative will have some error (that is why its approximating)
- One discrete approximation is central differences:

$$f'(x) \approx f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

- where h is small

Edge Detection: Derivatives

$$f'(x) \approx f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

- If we make h twice the distance between pixels
- The above equation simple states that the derivative of the image gradient at a pixel, is the next (right) pixel's value minus the previous (left) pixel's value
- The *Prewitt* operator takes this idea and creates a correlation filter which calculates the discrete derivative

Edge Detection: Finite Differences

- Prewitt operator

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- This is sensitive to noise, so it was combined with a Gaussian smoothing filter to give the:
- Sobel operator

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Edge Detection: Finite Differences

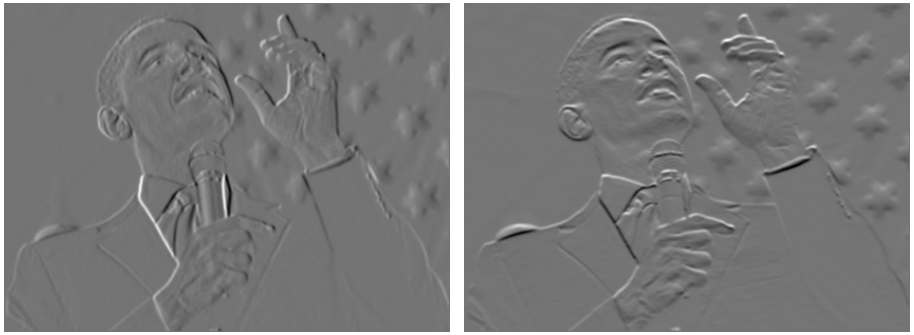
- The Sobel operator uses the first filter to give image Ix, and the second to give Iy
- Then the magnitude of the gradient is calculated along with its direction, thus:

$$\text{magnitude} = \sqrt{I_x^{\,2} + I_y^{\,2}} \qquad \text{gradient direction} = \arctan\left(\frac{I_y}{I_x}\right)$$

- think of any operation on image Ix as applying the operation to every pixel in Ix
- e.g. the magnitude of pixel (i,j) is

$$magnitude(i, j) = \sqrt{I_x(i, j)^2 + I_y(i, j)^2}$$

Edge Detection: Sobel

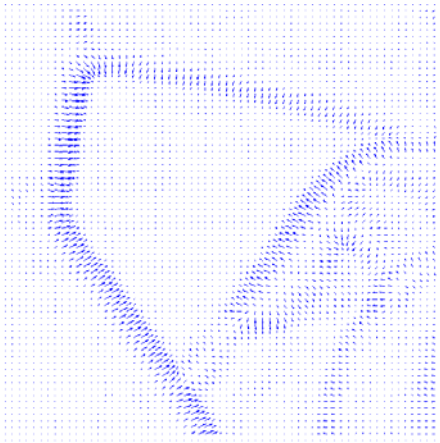


Ix

Iy

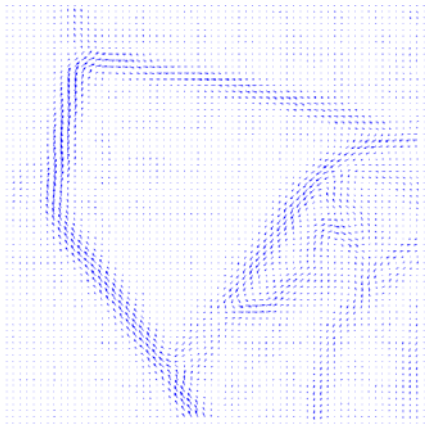
Edge Detection: Sobel – Gradient and Magnitude

- Each arrow direction is gradient direction, and the length is the magnitude
- Note how the unvarying regions have small arrows, and the edges have long arrows pointing across the edge



Edge Detection: Sobel Gradient Normal and Magnitude

- Each arrow direction is at right angles to the gradient direction, and the length is the magnitude
- Note how the unvarying regions have small arrows, and the edges now have long arrows pointing in the direction of the edge



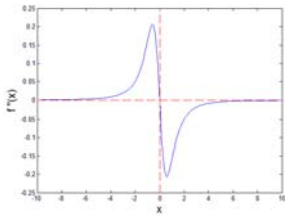
Sobel: Finding edges

- Taking the previous information
- If the gradient is greater than some value, map the angle onto a grey value (e.g. 0 deg (up)=white, 180 deg (down)=black

Sobel: Finding edges

- Or, high magnitude=black, low=white (left image), then all pixel values above a certain magnitude are set to black (thresholding – right image)
- Note, not all edges are present – poor connectivity

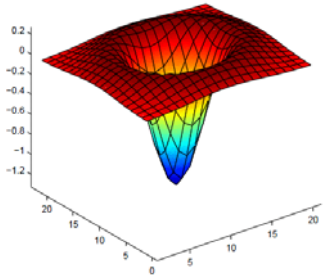
2nd order operator



- Sobel was the first derivative of a Gaussian (blur)
- The 2nd derivative can be used, and we then look for any pixel that is above zero that has a neighbour below zero – this is the crossing, and therefore the edge
- The following 5x5 (high pass) filter can be used

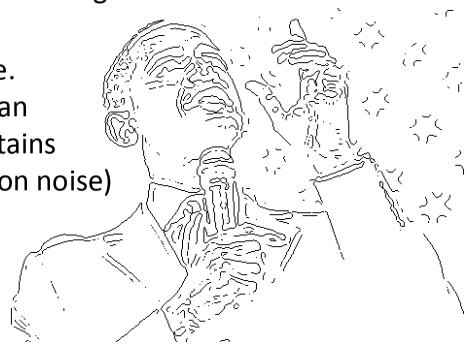
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- This is the 2nd derivative of the Gaussian and when plotted looks like:



2nd order operator

- Edges are found by looking for zero crossings – pixels above zero next to pixels below zero
- It produces good connectivity at low processing cost
- Finding edges using zero crossing is more accurate than the 1st order operators
- It is sensitive to noise (i.e. produces more edges than needed if the image contains noise – see next lecture on noise)



Summary

- Linear spatial filtering described (equation)
- Edges as high changes of intensity
- Rate of change of intensity (1st derivative) – edge spotted as maximum
- 2nd derivative shows edge at zero crossings
- Use correlation (Prewitt / Sobel) to find 1st derivative
- Use correlation (2nd order operator) to find 2nd derivative (then look for zero crossings)

Removing Noise

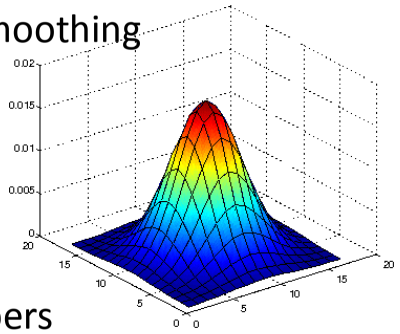
- Images can be noisy (sensors imperfect, transmission noise, compression artifacts)

25% noise

Removing Noise: Spatial Linear Filtering

- Already seen Gaussian smoothing

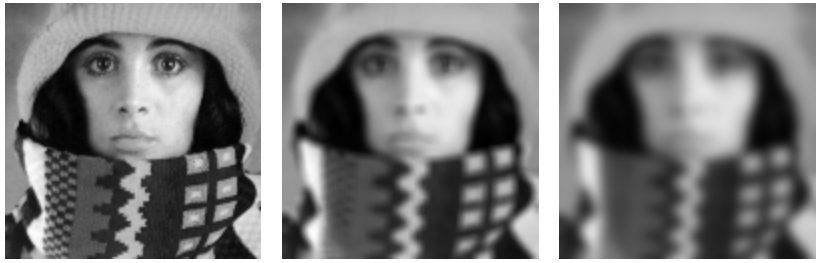
$$\begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$



- Here is a Gaussian, but this time using real numbers

Removing Noise: Gaussian Smoothing

- The box filter was all 1's – this is an average of all pixels local to the filter
- The Gaussian multiplies pixels by values – its called a *weighted average*
- It smoothes all parts of an image – including across edges
- Therefore sometimes known as a blurring filter
- Test it out in your assignment – note it does not blur as much as the box filter (because of the high weight on the central pixel)



Removing Noise: Non-linear Filtering

- Median filtering
- This is a non-linear filter because it no longer applies a filter kernel (or weighting) to pixel values
- Its different to all the previously presented filter techniques in these lectures
- Instead of weighting, the median (middle) value of the local neighbourhood is used

Removing Noise: Non-linear Filtering

- Using a small image (left), take the median filter size (e.g. 3x3)

80	89	78	80	81
82	88	86	90	85
81	83	52	85	83
78	80	78	82	80
76	77	77	75	73

Create a list of those pixels, and sort it

[88, 86, 90, 83, 52, 85, 80, 78, 82]

[52, 78, 80, 82, 83, 85, 86, 88, 90]

Find the median pixel (i.e. always the 5th pixel in the case of 3x3 region). Use it as the new value

80	89	78	80	81
82	88	86	90	85
81	83	83	85	83
78	80	78	82	80
76	77	77	75	73

Would have been 64, if using a 3X3 Gaussian kernel:

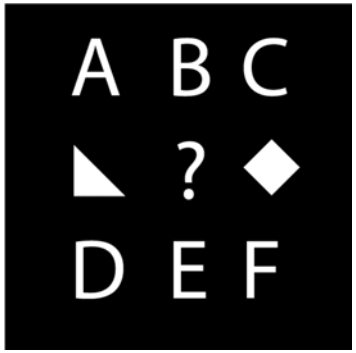
$$\begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$

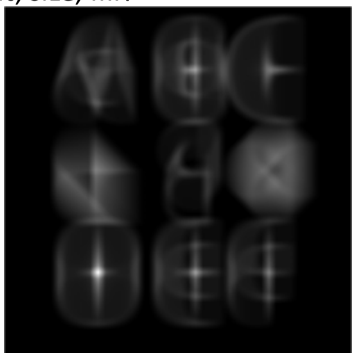
Removing Noise: Non-linear Filtering

- It preserves edges better than smoothing operators
- Its good at removing dots and lines (scratches)
- Not so affected by outliers (see previous slide)
- But sharp corners are smoothed (see below images)

Template Matching

- Detection and recognition by matching patterns
- Simple example
  - Filtering based upon correlation of template with image
  - Brightest spot indicates best match
- But what if its rotated? Different font, size, ...?





Template Matching

- Do we use templates for all possible differences?

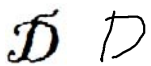
Scale changes

Intensity variations

Transformations

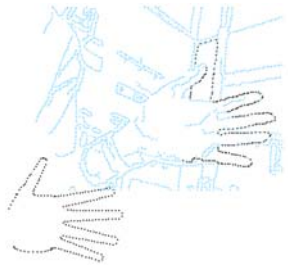
Noise

Partial occlusion



Computer Vision

- This introduces the topic of computer vision
- Real world problems are hard
- Identifying objects/people in images and video is a big research problem



[2]

[1]

[1] Schneiderman and Kanade, CVPR 2000

[2] Stenger et al. ICCV 2003

Resizing Images

- Recall images are a 2D array of RGB values
- I[y][x] gives access to pixel (x,y)
- I[y][x][c] gives access to colour c of pixel (x,y)
- See code in assignment framework
- To resize image Ia from size (Xa,Ya) to image Ib with size (Xb,Yb) we need to find a value for every colour component of every pixel in image Ib
- In other words, we need to find Ib[y][x][c] for all x,y,c where x in [0,Xb-1], y in [0,Yb-1], c in [0,1,2]=[R,G,B] (or [B, G, R] depending on which way around the bytes are stored

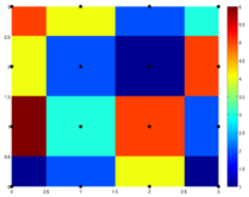
Resizing Images: Nearest Neighbour

- Use a loop:

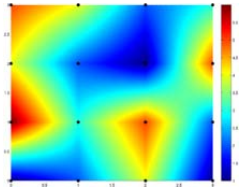
```
for j=0 to Yb-1
   for i=0 to Xb-1
      for c=0 to 2
         y=j*Ya/Yb <- make sure this is done using floats
         x=i*Xa/Xb <- same
         Ib[j][i][c]=Ia[y][x][c]
```

- Advantages:
  - Easy to code
  - Fast to compute (only look up 1 old pixel for each new pixel)
- Disadvantage:
  - Poor quality because each pixel is effectively made bigger

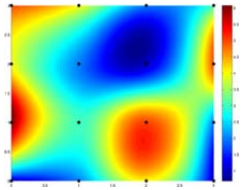
Comparing Interpolation Algorithms [source wikipedia]



- Dots have single colour which is used for nearest neighbouring pixels



- Bilinear interpolation used
- This function is smooth between points, but the derivative is not smooth at boundaries

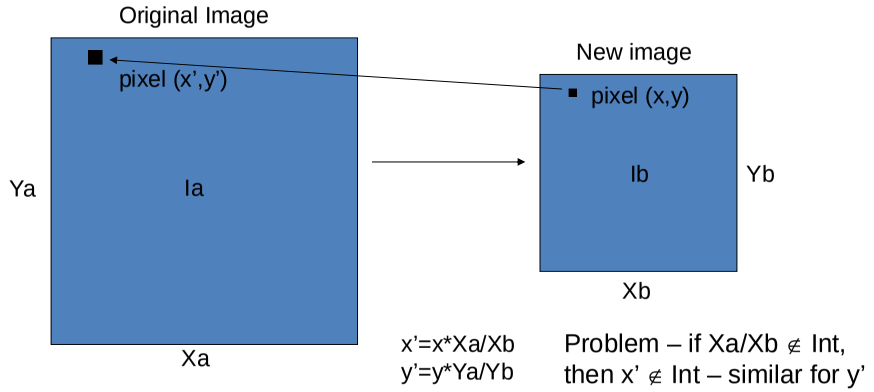


- Bicubic interpolation used
- This function is smooth between points and the derivative is smooth at the boundaries

All these techniques are demonstrated in Photoshop

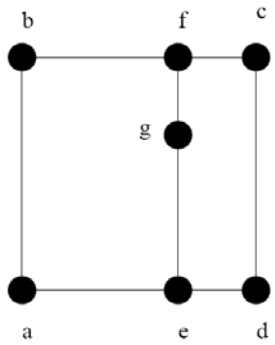
Resizing Images: Bilinear Interpolation

- Scaling images



Finding Pixel Colour

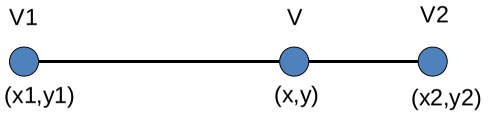
- How do we find a colour for a pixel (x',y') where x',y' ∈ Real?



- e.g. we know the colours at integer pixels a, b, c and d.
- We want to find out the colour at non-integer pixel g.
- Solution – we find out colours at e and f using LINEAR INTERPOLATION, and then g the same

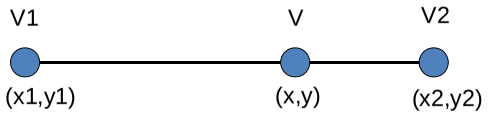
Linear Interpolation Equations

- To find value (e.g. colour) given position (on the y-axis)
- $v = v_1 + (v_2 - v_1)\frac{(y-y_1)}{(y_2-y_1)}$
- To find value (e.g. colour) given position (on the x-axis)
- $v = v_1 + (v_2 - v_1)\frac{(x-x_1)}{(x_2-x_1)}$

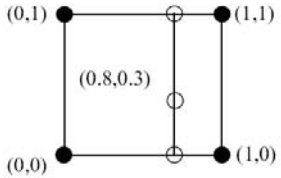


Linear Interpolation Equations

- To find position given colour
- x= $x_1 + (x_2 - x_1)\frac{(v-v_1)}{(v_2-v_1)}$
- $y = y_1 + (y_2 - y_1)\frac{(v-v_1)}{(v_2-v_1)}$



Bilinear Interpolation Exam Question May 2007



Point	Intensity
(0, 0)	0
(1,0)	120
(0,1)	80
(1,1)	140

Describe linear and bilinear interpolation (giving equations where appropriate) and demonstrate their use to calculate the intensity at positions (0.8, 0) and (0.8, 0.3) in the above square.

equations for linear interpolation and, bilinear interpolation and application
[3 marks]. (0.8,0)=96, (0.8,1)=128, (0.8,0.3)=105.6 [2 marks]

Dithering (including Error Diffusion Dithering/ Floyd-Steinberg methods)

(Reducing dynamic range)

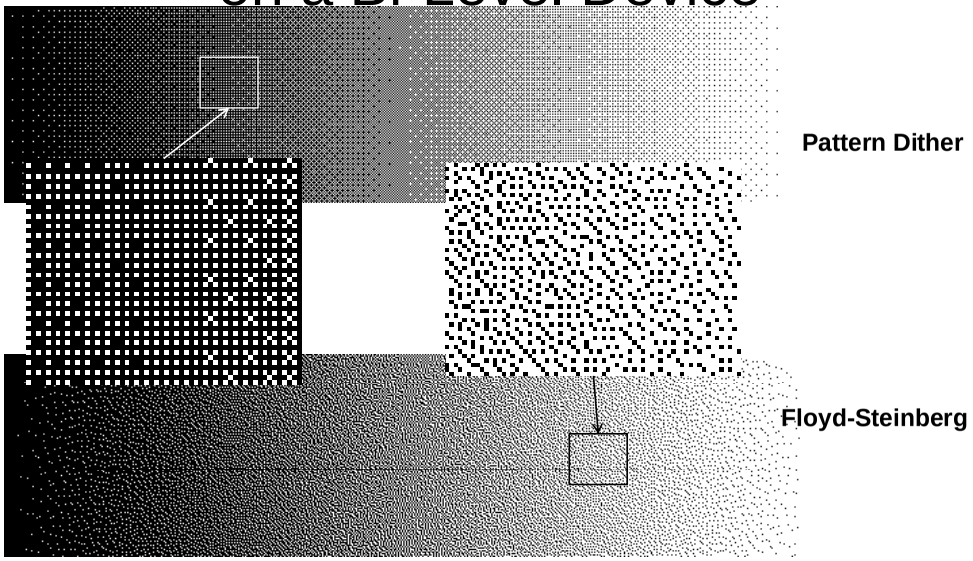
p585-591 of Hearn & Baker (3rd)

Displaying Grey-Scale on a Bi-Level Device

256 Greys

Thresholding

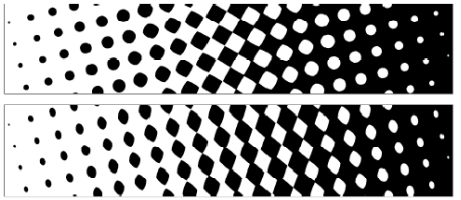
Displaying Grey-Scale on a Bi-Level Device



Pattern Dither

Floyd-Steinberg

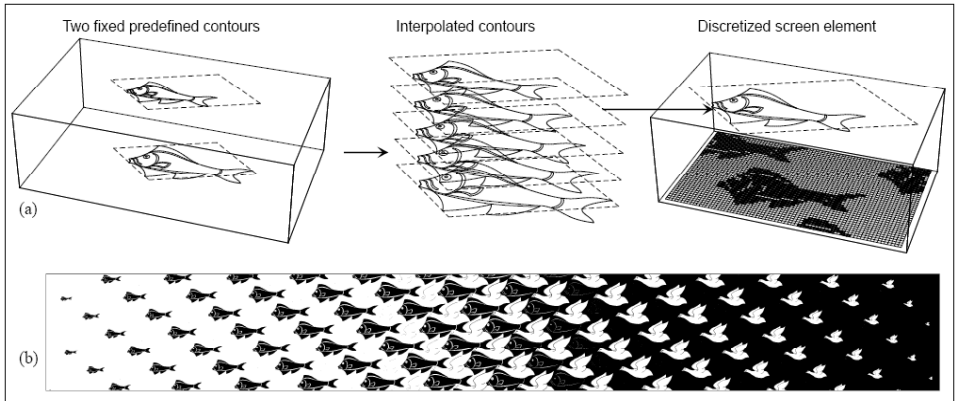
Other Grey Scales



- (Top) Print industry
- (Bottom) Computer generated

Victor Ostromoukhov and Roger D. Hersch, Artistic screening, Siggraph, 219 - 228, 1995

Escher Inspired



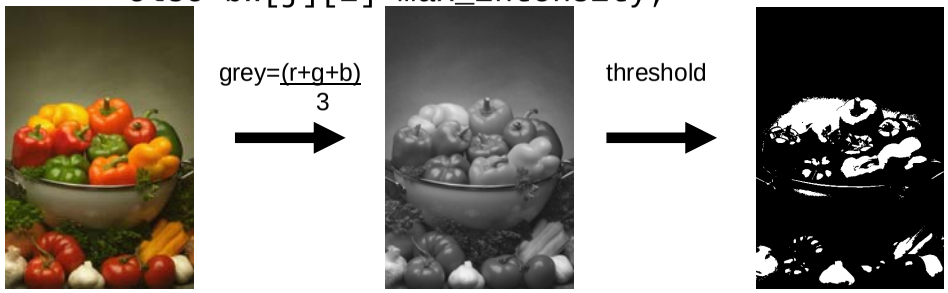
Victor Ostromoukhov and Roger D. Hersch, Artistic screening, Siggraph, 219 - 228, 1995

image->adjustments->posterise and threshold

Thresholding

Code:

```
threshold=max_intensity/2;
for j=1 to height
  for i=1 to width
     if (grey[j][i]<threshold) bw[j][i]=0
     else bw[j][i]=max_intensity;
```



Calculating Error

original grey image (0-255)

100	100	120	140
110	110	130	150
120	150	170	200
140	170	200	250

Total intensity=
100+100+120+…+250=2360
Average=
2360/16=147.5

threshold image

0	0	0	255
0	0	255	255
0	255	255	255
255	255	255	255

Total intensity=
0+0+0+255+…+255=2550
Average=
2550/16=159.375
Error (Total)=190 (Average)=11.875

Pathological Case

original grey image (0-255)

127	127	127	127
127	127	127	127
127	127	127	127
127	127	127	127

Total intensity=
127+127+127+…+127=2032
Average=
2032/16=127

threshold image

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Total intensity=
0+0+0+…+0=0
Average=
0/16=0
Error (Total)=-2032 (Average)=-127

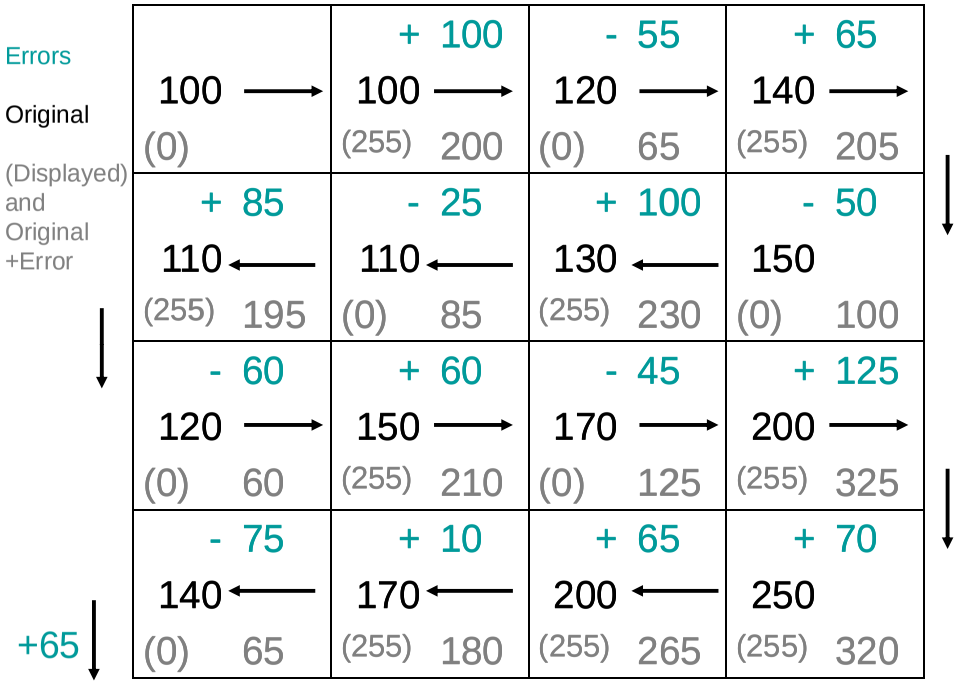
Error Diffusion Dithering

Code at each pixel:

```
grey[j][i]=grey[j][i]+error
if (grey[j][i]<threshold) {
        bw[j][i]=0
        error=grey[j][i]
}
else {
        bw[j][i]=max_intensity
        error=grey[j][i]-max_intensity
}
```

Error=0

100	100	120	140
110	110	130	150
120	150	170	200
140	170	200	250



Calculating Error

original grey image (0-255)

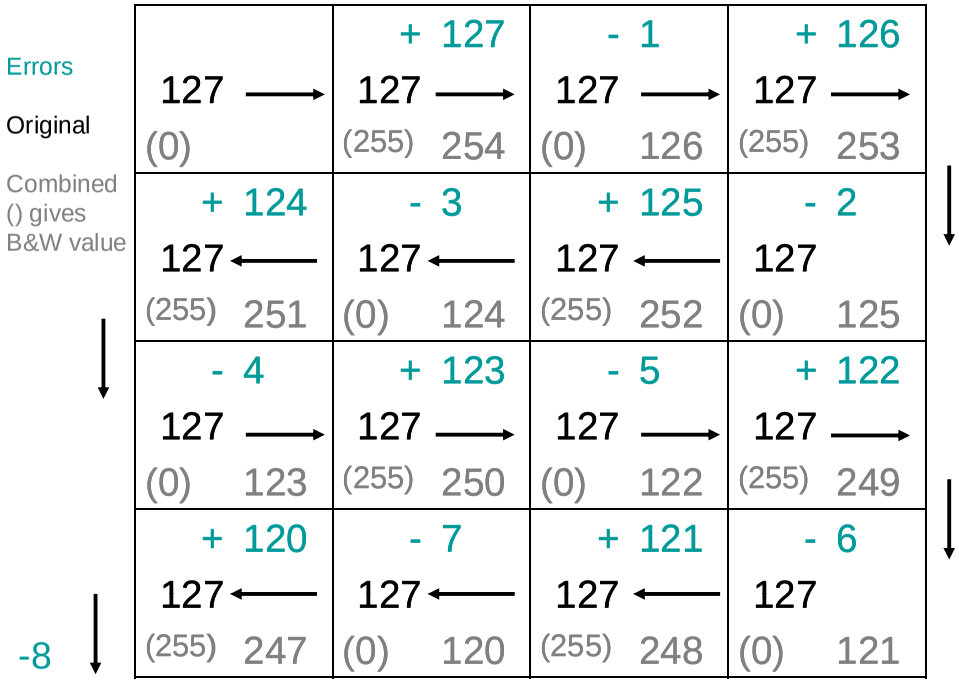
100	100	120	140
110	110	130	150
120	150	170	200
140	170	200	250

Total intensity=
100+100+120+…+250=2360
Average=
2360/16=147.5

error diffusion image

0	255	0	255
255	0	255	0
0	255	0	255
0	255	255	255

Total intensity=
0+255+0+255+…+255=2295
Average=
2295/16=143.4375
Error (Total)=-65 (Average)=-4.0625



Pathological Case

original grey image (0-255)

127	127	127	127
127	127	127	127
127	127	127	127
127	127	127	127

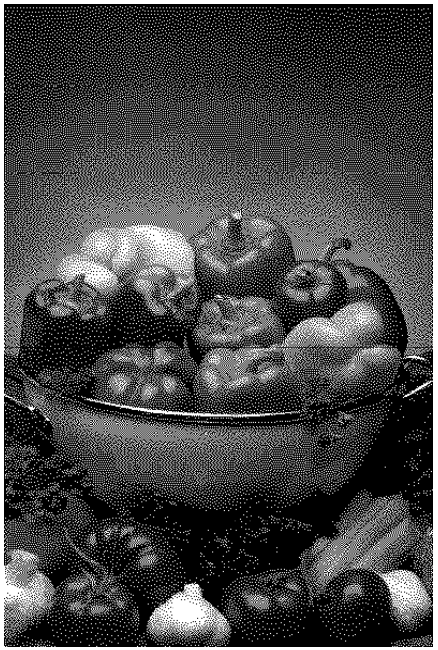
Total intensity=
127+127+127+…+127=2032
Average=
2032/16=127

error diffusion image

0	255	0	255
255	0	255	0
0	255	0	255
255	0	255	0

Total intensity=
0+255+0+…+0=2040
Average=
2040/16=127.5
Error (Total)=8 (Average)=0.5

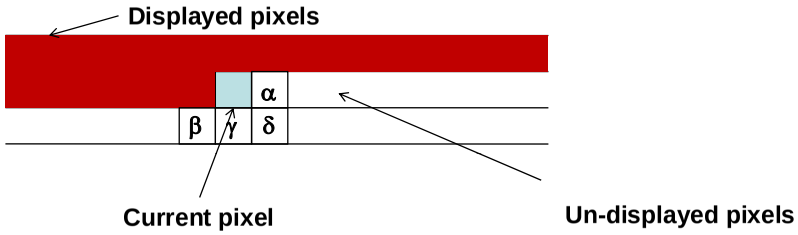
True colour



2 colour (B&W)

Floyd-Steinberg Error Diffusion Dithering

- Instead of passing 100% of the error to the next pixel, distribute it to the 4 “un-displayed” neighbours of the current pixel.
- $\alpha+\beta+\gamma+\delta=1$, e.g. $(\alpha,\beta,\gamma,\delta)=(^{7}/_{16}, ^{3}/_{16}, ^{5}/_{16}, ^{1}/_{16})$



Floyd-Steinberg Error Diffusion Dithering

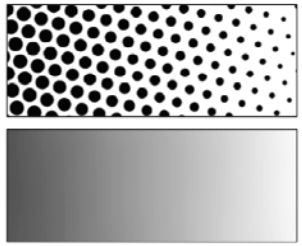
Code at each pixel:

```
if (grey[j][i]<threshold) {
        bw[j][i]=0
        error=grey[j][i]
}
else {
        bw[j][i]=max_intensity
        error=grey[j][i]-max_intensity
}
grey[j+1][i-1]=grey[j+1][i-1]+error*β;
grey[j+1][i]=grey[j+1][i]+error*γ;
grey[j+1][i+1]=grey[j+1][i+1]+error*δ;
grey[j][i+1]=grey[j][i+1]+error*α;
```

- Passing error
- Assumes grey[j][i] corresponds to pixel (i,j)
- Care must be taken at edges
- Grey must be type integer

Halftoning (Print industry)

- B&W images produced using differently sized black circles
- Diameter proportional to darkness of region
- Colour halftones use dots of different sizes in each colour channel (usually CMY=Cyan, Magenta, Yellow)
- Reduces dynamic range (use fewer colours, but still give the impression of continuous tone)



Halftoning (Computer Graphics)

- Use halftone approximation patterns (halftone patterns)
- These are example templates that produce the patterns that shall be used in the following sections (3x3, 4x4, 5x5):

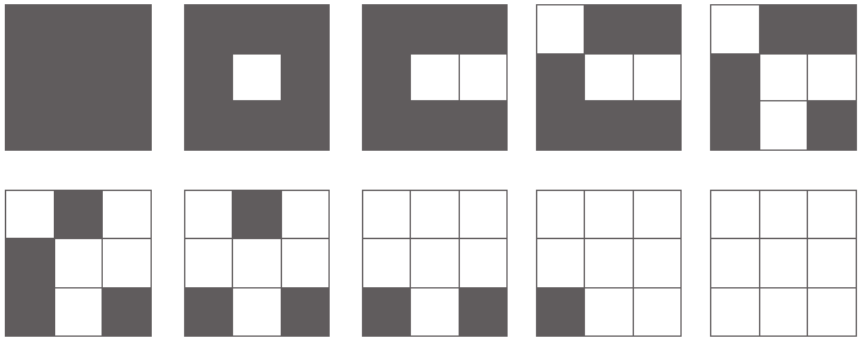
3	7	5
6	1	2
9	4	8

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

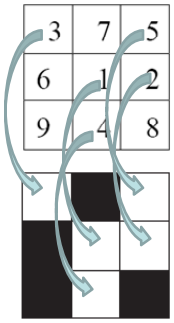
22	11	18	15	24
16	3	7	5	10
21	6	1	2	19
13	9	4	8	14
25	17	20	12	23

Halftoning

- nxn pixels can represent n^2+1 intensity levels (on a bi-level display)
- 3x3 displays 10 intensity levels distributed from black to white – halftone patterns:



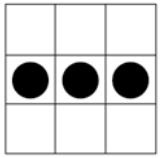
Halftoning



- e.g. intensity level 5, all pixels up to and including 5 are set to white
- Given original intensity $I \in [0,1]$ and an nxn halftone template, intensity I is represented by pattern p, where
- $p=\min(\text{floor}(I*(n^2+1)), n^2)$
- (Intensity levels are labelled $0..n^2$ e.g. 0..9 in case of 3x3)

Halftoning Patterns

- The patterns must form a growth sequence so that any pixel on for intensity level j is also on for all levels k>*j. (Avoids some visual effects)*
- Avoid single pixels (i.e. try to set neighbouring pixels). *(Avoids problems with inaccurate printers)*
- Avoid patterns developing, e.g. this 3x3 dither pattern for intensity level 3 would result in stripes appearing for large regions of this intensity



Pattern Dither

e.g. 4x4 template

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

100	100	120	140
110	110	130	150
120	150	170	200
140	170	200	250

Process – get total scaled intensity=0.578
Find pattern template to use (p=9)
Compare with 4x4 template
Set pixel white if p >= template number

1	9	3	
	5		7
4		2	
	8		6

- p=min(floor(I*(n^2+1)), n^2)
- To get total scaled intensity I∈[0,1]:
- Total intensity=100+100+120+…+250=2360
- Average intensity=Total intensity/number of pixels: 2360/16=147.5
- Total Scaled intensity=average intensity/255: 147.5/255=0.578
- 4x4 Pattern p=min(floor(0.578*(4^2+1)), 4^2)=9

Ordered Dither

e.g. 4x4 template

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

100	100	120	140
110	110	130	150
120	150	170	200
140	170	200	250

Process – get pattern number for each pixel.

6	6	8	9
7	7	8	10
8	10	11	13
9	11	13	16

- p=min(floor(I*(n^2+1)), n^2)
- To get scaled intensity I∈[0,1]:
- Top left pixel=100
- Scaled intensity=intensity/255: 100/255=0.39
- 4x4 Pattern p=min(floor(0.39*(4^2+1)), 4^2)=6
- For 110, p=7, for 120, p=8, for 130, p=8, for 140, p=9, for 150, p=10, for 170, p=11, for 200, p=13, for 250, p=16

Ordered Dither

e.g. 4x4 template

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

6	6	8	9
7	7	8	10
8	10	11	13
9	11	13	16

Now compare pattern number with template. If pattern number is >= template number, set pixel to white

6	6	8	9
7	7	8	10
8	10	11	13
9	11	13	16

Question 1

(a) **Image Processing** Given a 3x3 sub-image of pixels:

$$I_{ij} = \begin{bmatrix} p_{11}\ p_{12}\ p_{13} \\ p_{21}\ p_{22}\ p_{23} \\ p_{31}\ p_{32}\ p_{33} \end{bmatrix}$$

and a 3×3 filter kernel:

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

what is the equation for calculating the new intermediate value at pixel p_{22}?

Given the following 5×5 image (grey-level 0-255 image):

$$\begin{bmatrix} 100 & 110 & 120 & 130 & 140 \\ 120 & 120 & 130 & 130 & 140 \\ 130 & 140 & 150 & 150 & 160 \\ 140 & 150 & 150 & 160 & 170 \\ 150 & 160 & 170 & 180 & 190 \end{bmatrix}$$

and 3x3 filter kernel:

-1	-1	-1
-1	8	-1
-1	-1	-1

what is the result of applying the filter kernel to the image using the process of cross-correlation? Your answer should cover the following items: give the intermediate values for all possible pixels, indicate what choices could be made at the pixels the filter cannot be operated on, calculate the maximum and minimum intermediate values, show the normalisation equation, and use the normalisation equation to create the final pixel values.

What differences occur when a 4×4 filter is used, and when a 5×5 filter is used? What difference is made when a colour image needs to be cross-correlated?

Name some applications of cross-correlation. What is a hi-pass filter and what is a low-pass filter and when should they be used? Identify the filter given in the question.

[16 marks]

[Bookwork, understanding, practical application of topic]

$$p_{22} = \sum_{j=1}^{3} \sum_{i=1}^{3} p_{ij} m_{ij}$$

*	*	*	*	*
*	*-40*	*-10*	*-80*	*
*	*30*	*70*	*10*	*
*	*10*	*-60*	*-40*	*
*	*	*	*	*

[5 marks]

=could not be applied. Could reduce size of image, make black, keep old colour, use nearest known neighbour's colour, use smaller filters at the image edges.* *[1 marks]***

max=70, min=-80. ***[1 marks]***

Normalisation equation $P = (I - min) \times \frac{255}{(max-min)}$ ***[1 marks]***

In this case:

$P = (I + 80) \times \frac{255}{(150)}$

Gives

*	*	*	*	*
*	*68*	*119*	*0*	*
*	*187*	*255*	*153*	*
*	*153*	*34*	*68*	*
*	*	*	*	*

[3 marks]

A 4×4 *will have an extra row and column unknown on the top and one side (or bottom and one side). A* 5×5 *will have 2 rows unknown all the way round.* ***[1.5 marks]***

Also, increasing the size of the filter will result in the need for more calculation. ***[0.5 marks]***

A colour image will require the above operation to be carried out for each colour channel. The min and max are for all colour channels, and not each individually. ***[1 marks]***

Applications medical imaging and astronomy (low-pass to remove noise). Object recognition, facial recognition (hi-pass to enhance edges). ***[1.5 marks]***

This filter is an edge detector. ***[0.5 marks]***

Part ***a*** *subtotal=****[16 marks]***

Some other example images and filters (for you to practice and check your calculations):

Example 1

Filter:

1	3	1
3	9	3
1	3	1

Image:

80	100	110	130	140
80	100	110	130	140
100	120	150	150	160
120	150	150	160	170
150	160	170	180	200

Result: Intermediate result:

*	*	*	*	*
*	2570	2960	3320	*
*	3040	3480	3720	*
*	3570	3850	4070	*
*	*	*	*	*

Final result (given to 1d.p.)

*	*	*	*	*
*	0	66.3	127.5	*
*	79.9	154.7	195.5	*
*	170.	217.6	255.	*
*	*	*	*	*

Example 2

Filter:

1	3	1
3	9	3
1	3	1

Image:

40	50	50	80	100
60	70	70	90	100
80	70	80	150	160
100	100	90	150	170
120	120	130	150	170

Result: Intermediate result:

*	*	*	*	*
*	1630	1850	2400	*
*	1940	2270	3220	*
*	2450	2680	3570	*
*	*	*	*	*

Final result (given to 1d.p.)

*	*	*	*	*
*	0	28.9	101.2	*
*	40.7	84.1	209.	*
*	107.8	138.	255.	*
*	*	*	*	*

Example 3

Filter:

1	3	1
3	9	3
1	3	1

Image:

0	0	0	255	255
0	0	0	255	255
0	0	0	255	255
0	0	0	255	255
0	0	0	255	255

Result: Intermediate result:

*	*	*	*	*
*	0	1275	5100	*
*	0	1275	5100	*
*	0	1275	5100	*
*	*	*	*	*

Final result (given to 1d.p.)

*	*	*	*	*
*	0	63.8	255.	*
*	0	63.8	255.	*
*	0	63.8	255.	*
*	*	*	*	*

Additional question: Can you comment on the image and result?

The initial image has a black left side and a white right side - it contains very high frequency (the edge). The correlated image introduces a grey intermediate column of pixels into the harsh edge. i.e. it blurs the edge.

Example 4

Filter:

1	1	1
1	-8	1
1	1	1

Image:

0	0	0	255	255
0	0	0	255	255
0	0	0	255	255
0	0	0	255	255
0	0	0	255	255

Result: Intermediate result:

*	*	*	*	*
*	0	765	-765	*
*	0	765	-765	*
*	0	765	-765	*
*	*	*	*	*

Final result (given to 1d.p.)

*	*	*	*	*
*	127.5	255.	0	*
*	127.5	255.	0	*
*	127.5	255.	0	*
*	*	*	*	*

Additional question: Can you comment on the image and result?

The initial image has a black left side and a white right side - it contains very high frequency (the edge). The correlated image will keep unvarying areas grey, but will accentuate edges in the image. Overall the effect will be to highlight all the edges.

Error Diffusion

Example 1

Try error diffusion on the following pixels (part question from 2007–2008):

80	90	100	110	120
95	110	120	140	170

The resulting image should be:

0	255	0	0	255
0	0	255	0	255

The errors passed to each pixel are:

80	-85	15	125	-10
115	20	-90	45	-95

(i.e. the first error is passed from the first pixel and added to the second pixel, and the last error on the first row is passed down and added to the last pixel of the second row. The first error on the second row is the overall error for these two lines)

The errors, when added to the original pixel values give:

80	170	15	125	245
115	20	165	45	160

which are compared to 128 to determine if the pixel is displayed using black (0) or white (255).

Example 2

Image:

50	60	70	70	80
40	50	60	70	70

The resulting image should be:

0	0	255	0	0
0	0	0	0	255

The errors passed to each pixel are:

50	110	-75	-5	75
110	70	20	-40	-110

The errors, when added to the original pixel values give:

50	110	180	-5	75
110	70	20	-40	145

Note the pixel values summed with the error become negative in some cases. This means the integer data type should be used rather than byte when implementing error diffusion.

Example 3

Image:

180	200	220	230	240
160	180	190	200	240

The resulting image should be:

255	0	255	255	255
255	0	255	255	255

The errors passed to each pixel are:

-75	125	90	65	50
0	95	-85	-20	35

The errors, when added to the original pixel values give:

180	125	345	320	305
255	95	170	235	290

Note the error exceeds 255 in this case. This means the integer data type should be used rather than byte when implementing error diffusion.

Example question in full:

(b) This is a grey-level image containing pixel values between 0 and 255.

80	90	100	110	120
95	110	120	140	170

(i) What would be the results after *thresholding* to display on a bi-level device? **[2 marks]**

(ii) Calculate the total and average error between the image and the thresholded image. **[2 marks]**

(iii) Demonstrate *standard error-diffusion* on the image. **[3 marks]**

(iv) Calculate the total and average error between the image and the error-diffusion image. **[2 marks]**

Using the image below, show what the results would be if we tried to display it on a bi-level device using *thresholding*. Calculate the error between the original and the bi-level images. What are the advantages and disadvantages of using such a method?

Demonstrate the results of using the *standard error-diffusion* method. Calculate the error between the original and the new image. What are the advantages and disadvantages of this method? What feature of the standard error-diffusion process retains error in the local area? Name another method that retains error in the local area, and describe how it achieves that. (In all cases assume a 0-255 grey-level image).

[10 marks]

[Practical application of theory]

Thresholding becomes:

0	0	0	0	0
0	0	0	1	1

[1 marks]

Original total = 1135. Bi-level image = $2 \times 255 = 510$*. Error=625 too dim. Advantage fast to compute, easy to implement. Disadvantage very inaccurate. Examples were given in lecture of thresholding which demonstrated that most image detail was lost.*

[2 marks]

(Error calculation - also see example 1 previously)

Note error diffusion works by zig-zagging through the image passing error to the next available (undrawn) pixel.

Error diffusion 80→0, err=80, 90+80=170→1, err=170-255=-85, 100-85=15→0, err=15, 110+15=125→0, error=125, 120+125=245→1, err=245-255=-10, 170-10=160→1, err=160-255=-95, 140-95=45→0, err=45, 120+45=165→1, err=165-255=-90, 110-90=20→0, err=20, 95+20=115→0, err=115.

0	1	0	0	1
0	0	1	0	1

[3 marks]

Error for new dither = $4 \times 255 = 1020$. *Error=115 too dim (Aside: the magnitude of the error matches the error passed by the last pixel above, so we know this is probably correct). Advantages this is much better representation of the image, it is not too difficult to implement. Disadvantage better methods exist (see next part).*

[2 marks]

e.g. Floyd Steinberg passes error to 4 undrawn neighbouring pixels in the ratio of (7/16 to right pixel, 1/16 to bottom left pixel, 5/16 to below pixel, 3/16 to bottom right pixel (assuming left to right operation)). Error is retained in local area by passing error to neighbourhood (as in the above ratio). At the image edge, error is passed to the undrawn 2 pixels below the current one. And on the last row, all of the error is passed to the undrawn pixel. (Aside: This is a very full answer for the FS part).

[2 marks]

*Part **b** subtotal=**[10 marks]***