# CHAPTER 8

# Digital images and image formats

An important type of digital media is images, and in this chapter we are going to review how images are represented and how they can be manipulated with simple mathematics. This is useful general knowledge for anyone who has a digital camera and a computer, but for many scientists, it is an essential tool. In astrophysics data from both satellites and distant stars and galaxies is collected in the form of images, and information extracted from the images with advanced image processing techniques. Medical imaging makes it possible to gather different kinds of information in the form of images, even from the inside of the body. By analysing these images it is possible to discover tumours and other disorders.

## 8.1 What is an image?

Before we do computations with images, it is helpful to be clear about what an image really is. Images cannot be perceived unless there is some light present, so we first review superficially what light is.

### 8.1.1 Light

**Fact 8.1** (What is light?)**.** *Light is electromagnetic radiation with wavelengths in the range 400–700 nm (1 nm is* $10^{-9}$ *m): Violet has wavelength 400 nm and red has wavelength 700 nm. White light contains roughly equal amounts of all wave lengths.*

Other examples of electromagnetic radiation are gamma radiation, ultraviolet and infrared radiation and radio waves, and all electromagnetic radiation travel at the speed of light ($3 \times 10^8$ m/s). Electromagnetic radiation consists of waves and may be reflected and refracted, just like sound waves (but sound waves are not electromagnetic waves).

We can only see objects that emit light, and there are two ways that this can happen. The object can emit light itself, like a lamp or a computer monitor, or it reflects light that falls on it. An object that reflects light usually absorbs light as well. If we perceive the object as red it means that the object absorbs all light except red, which is reflected. An object that emits light is different; if it is to be perceived as being red it must emit only red light.

### 8.1.2 Digital output media

Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a rectangular array of small dots which emit light. In most technologies, each dot is really three smaller dots, and each of these smaller dots emit red, green and blue light. If the amounts of red, green and blue is varied, our brain merges the light from the three small light sources and perceives light of different colours. In this way the colour at each set of three dots can be controlled, and a colour image can be built from the total number of dots.

It is important to realise that it is possible to generate most, but not all, colours by mixing red, green and blue. In addition, different computer monitors use slightly different red, green and blue colours, and unless this is taken into consideration, colours will look different on the two monitors. This also means that some colours that can be displayed on one monitor may not be displayable on a different monitor.

Printers use the same principle of building an image from small dots. On most printers however, the small dots do not consist of smaller dots of different colours. Instead as many as 7–8 different inks (or similar substances) are mixed to the right colour. This makes it possible to produce a wide range of colours, but not all, and the problem of matching a colour from another device like a monitor is at least as difficult as matching different colours across different monitors.

Video projectors builds an image that is projected onto a wall. The final image is therefore a reflected image and it is important that the surface is white so that it reflects all colours equally.

The quality of a device is closely linked to the density of the dots.

**Fact 8.2** (Resolution)**.** *The resolution of a medium is the number of dots per inch (dpi). The number of dots per inch for monitors is usually in the range 70–120, while for printers it is in the range 150–4800 dpi. The horizontal and vertical densities may be different. On a monitor the dots are usually referred to as* pixels *(picture elements).*

### 8.1.3 Digital input media

The two most common ways to acquire digital images is with a digital camera or a scanner. A scanner essentially takes a photo of a document in the form of a rectangular array of (possibly coloured) dots. As for printers, an important measure of quality is the number of dots per inch.

**Fact 8.3.** *The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the colour is represented with up to 48 bits per dot.*

For digital cameras it does not make sense to measure the resolution in dots per inch, as this depends on how the image is printed (its size). Instead the resolution is measured in the number of dots recorded.

**Fact 8.4.** *The number of pixels recorded by a digital camera usually varies in the range* $320 \times 240$ *to* $6000 \times 4000$ *with 24 bits of colour information per pixel. The total number of pixels varies in the range 76 800 to 24 000 000 (0.77 megapixels to 24 megapixels).*

For scanners and cameras it is easy to think that the more dots (pixels), the better the quality. Although there is some truth to this, there are many other factors that influence the quality. The main problem is that the measured colour information is very easily polluted by noise. And of course high resolution also means that the resulting files become very big; an uncompressed $6000 \times 4000$ image produces a 72 MB file. The advantage of high resolution is that you can magnify the image considerably and still maintain reasonable quality.

### 8.1.4 Definition of digital image

We have already talked about digital images, but we have not yet been precise about what it is. From a mathematical point of view, an image is quite simple.

(a) (b) (c)

**Figure 8.1**. Different version of the same image; black and white (a), grey-level (b), and colour (c).

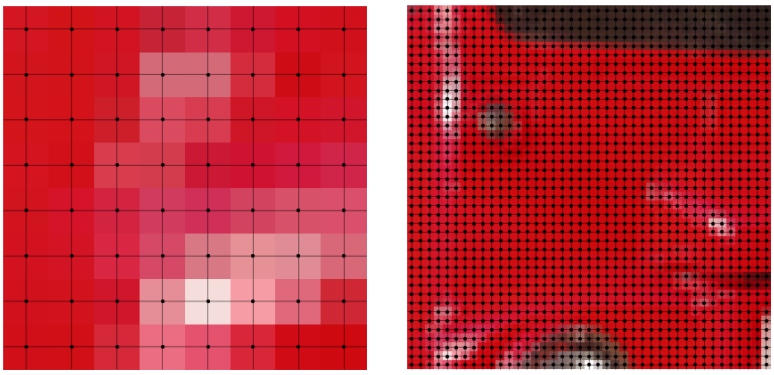
**Fact 8.5** (Digital image)**.** *A digital image $P$ is a rectangular array of* intensity values $\{p_{i,j}\}_{i,j=1}^{m,n}$. *For grey-level images, the value $p_{i,j}$ is a single number, while for colour images each $p_{i,j}$ is a vector of three or more values. If the image is recorded in the rgb-model, each $p_{i,j}$ is a vector of three values,*

$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$

*that denote the amount of red, green and blue at the point $(i, j)$.*

The value $p_{i,j}$ gives the colour information at the point $(i, j)$. It is important to remember that there are many formats for this. The simplest case is plain black and white images in which case $p_{i,j}$ is either 0 or 1. For grey-level images the intensities are usually integers in the range 0–255. However, we will assume that the intensities vary in the interval $[0,1]$, as this sometimes simplifies the form of some mathematical functions. For colour images there are many different formats, but we will just consider the rgb-format mentioned in the fact box. Usually the three components are given as integers in the range 0–255, but as for grey-level images, we will assume that they are real numbers in the interval $[0,1]$ (the conversion between the two ranges is straightforward, see section 8.2.3 below). Figure 8.1 shows an image in different formats.

**Fact 8.6.** *In these notes the intensity values $p_{i,j}$ are assumed to be real numbers in the interval $[0,1]$. For colour images, each of the red, green, and blue*



(a) (b)

**Figure 8.2**. Two excerpt of the colour image in figure 8.1. The dots indicate the position of the points $(i, j)$.

*intensity values are assumed to be real numbers in* $[0, 1]$.

If we magnify a small part of the colour image in figure 8.1, we obtain the image in figure 8.2 (the black lines and dots have been added). A we can see, the pixels have been magnified to big squares. This is a standard representation used by many programs — the actual shape of the pixels will depend on the output medium. Nevertheless, we will consider the pixels to be square, with integer coordinates at their centres, as indicated by the grids in figure 8.2.

**Fact 8.7** (Shape of pixel)**.** *The pixels of an image are assumed to be square with sides of length one, with the pixel with value* $p_{i,j}$ *centred at the point* $(i, j)$.

### 8.1.5 Images as surfaces

Recall from chapter 7 that a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ can be visualised as a surface in space. A grey-level image is almost on this form. If we define the set of integer pairs by

$$\mathbb{Z}_{m,n} = \{(i, j) \mid 1 \leq i \leq m \text{ and } 1 \leq j \leq n\},$$

we can consider a grey-level image as a function $P : \mathbb{Z}_{m,n} \mapsto [0, 1]$. In other words, we may consider an image to be a sampled version of a surface with the intensity value denoting the height above the $(x, y)$-plane, see figure 8.3.

**Figure 8.3**. The grey-level image in figure 8.1 plotted as a surface. The height above the $(x, y)$-plane is given by the intensity value.

**Fact 8.8** (Grey-level image as a surface)**.** *Let $P = (p)_{i,j=1}^{m,n}$ be a grey-level image. Then $P$ can be considered a sampled version of the piecewise constant surface*

$$F_P : [1/2, m+1/2] \times [1/2, n+1/2] \mapsto [0,1]$$

*which has the constant value $p_{i,j}$ in the square (pixel)*

$$[i-1/2, i+1/2] \times [j-1/2, j+1/2]$$

*for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.*

What about a colour image $P$? Then each $p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j})$ is a triple of numbers so we have a mapping

$$P : \mathbb{Z}_{m,n} \mapsto \mathbb{R}^3.$$

If we compare with definition 7.9, we see that this corresponds to a sampled version of a parametric surface if we consider the colour values $(r_{i,j}, g_{i,j}, b_{i,j})$ to be $x$-, $y$-, and $z$-coordinates. This may be useful for computations in certain settings, but visually it does not make much sense, see figure 8.4

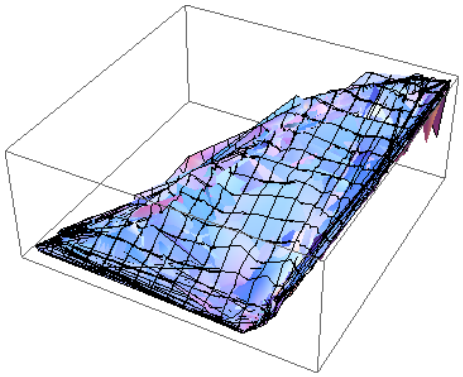


**Figure 8.4**. A colour image viewed as a parametric surface in space.

## 8.2 Operations on images

When we know that a digital image is a two-dimensional array of numbers, it is quite obvious that we can manipulate the image by performing mathematical operations on the numbers. In this section we will consider some of the simpler operations.

### 8.2.1 Normalising the intensities

We have assumed that the intensities all lie in the interval $[0,1]$, but as we noted, many formats in fact use integer values in the range 0–255. And as we perform computations with the intensities, we quickly end up with intensities outside $[0,1]$ even if we start out with intensities within this interval. We therefore need to be able to *normalise* the intensities. This we can do with the simple linear function in observation 5.23,

$$g(x) = \frac{x-a}{b-a}, \quad a < b,$$

which maps the interval $[a,b]$ to $[0,1]$. A simple case is mapping $[0,255]$ to $[0,1]$ which we accomplish with the scaling $g(x) = x/255$. More generally, we typically perform computations that result in intensities outside the interval $[0,1]$. We can then compute the minimum and maximum intensities $p_{\min}$ and $p_{\max}$ and map the interval $[p_{\min}, p_{\max}]$ back to $[0,1]$. Several examples of this will be shown below.

(a) (b) (c)

**Figure 8.5**. The red (a), green (b), and blue (c) components of the colour image in figure 8.1.

### 8.2.2 Extracting the different colours

If we have a colour image $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ it is often useful to manipulate the three colour components separately as the three images

$$P_r = (r_{i,j})_{i,j=1}^{m,n}, \quad P_r = (g_{i,j})_{i,j=1}^{m,n}, \quad P_r = (b_{i,j})_{i,j=1}^{m,n}.$$

These are conveniently visualised as grey-level images as in figure 8.5.

### 8.2.3 Converting form colour to grey-level

If we have a colour image we can convert it to a grey-level image. This means that at each point in the image we have to replace the three colour values $(r, g, b)$ by a single value $p$ that will represent the grey level. If we want the grey-level image to be a reasonable representation of the colour image, the value $p$ should somehow reflect the intensity of the image at the point. There are several ways to do this.

It is not unreasonable to use the largest of the three colour components as a measure of the intensity, i.e, to set $p = \max(r, g, b)$. The result of this can be seen in figure 8.6a.

An alternative is to use the sum of the three values as a measure of the total intensity at the point. This corresponds to setting $p = r + g + b$. Here we have to be a bit careful with a subtle point. We have required each of the $r$, $g$ and $b$ values to lie in the range $[0, 1]$, but their sum may of course become as large as 3. We also require our grey-level values to lie in the range $[0, 1]$ so after having computed all the sums we must normalise as explained above. The result can be seen in figure 8.6b.

(a) (b) (c)

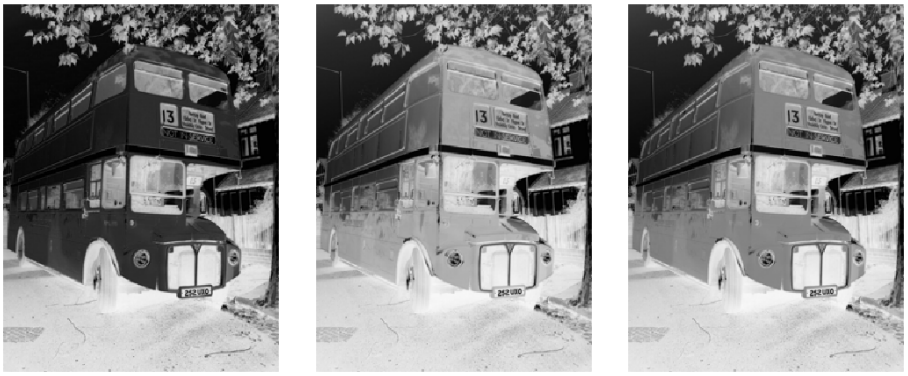
**Figure 8.6**. Alternative ways to convert the colour image in figure 8.1 to a grey level image. In (a) each colour triple has been replaced by its maximum, in (b) each colour triple has been replaced by its sum and the result mapped to $[0,1]$, while in (c) each triple has been replaced by its length and the result mapped to $[0,1]$.

A third possibility is to think of the intensity of $(r,g,b)$ as the length of the colour vector, in analogy with points in space, and set $p = \sqrt{r^2+g^2+b^2}$. Again we may end up with values in the range $[0,3]$ so we have to normalise like we did in the second case. The result is shown in figure 8.6c.

Let us sum this up as an algorithm.

**Algorithm 8.9** (Conversion from colour to grey level)**.** *A colour image* $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ *can be converted to a grey level image* $Q = (q_{i,j})_{i,j=1}^{m,n}$ *by one of the following three operations:*

1. *Set* $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$ *for all* $i$ *and* $j$*.*
2. (a) *Compute* $\hat{q}_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}$ *for all* $i$ *and* $j$*.*
   (b) *Transform all the values to the interval* $[0,1]$ *by setting*
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$
3. (a) *Compute* $\hat{q}_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$ *for all* $i$ *and* $j$*.*
   (b) *Transform all the values to the interval* $[0,1]$ *by setting*
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$



(a) (b) (c)

**Figure 8.7**. The negative versions of the corresponding images in figure 8.6.

In practice one of the last two methods are usually preferred, perhaps with a preference for the last method, but the actual choice depends on the application.

### 8.2.4 Computing the negative image

In film-based photography a negative image was obtained when the film was developed, and then a positive image was created from the negative. We can easily simulate this and compute a negative digital image.

Suppose we have a grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ with intensity values in the interval $[0,1]$. Here intensity value 0 corresponds to black and 1 corresponds to white. To obtain the negative image we just have to replace an intensity $p$ by its 'mirror value' $1-p$.

**Fact 8.10** (Negative image)**.** *Suppose the grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ is given, with intensity values in the interval $[0,1]$. The negative image $Q = (q_{i,j})_{i,j=1}^{m,n}$ has intensity values given by $q_{i,j} = 1 - p_{i,j}$ for all $i$ and $j$.*

### 8.2.5 Increasing the contrast

A common problem with images is that the contrast often is not good enough. This typically means that a large proportion of the grey values are concentrated in a rather small subinterval of $[0,1]$. The obvious solution to this problem is to

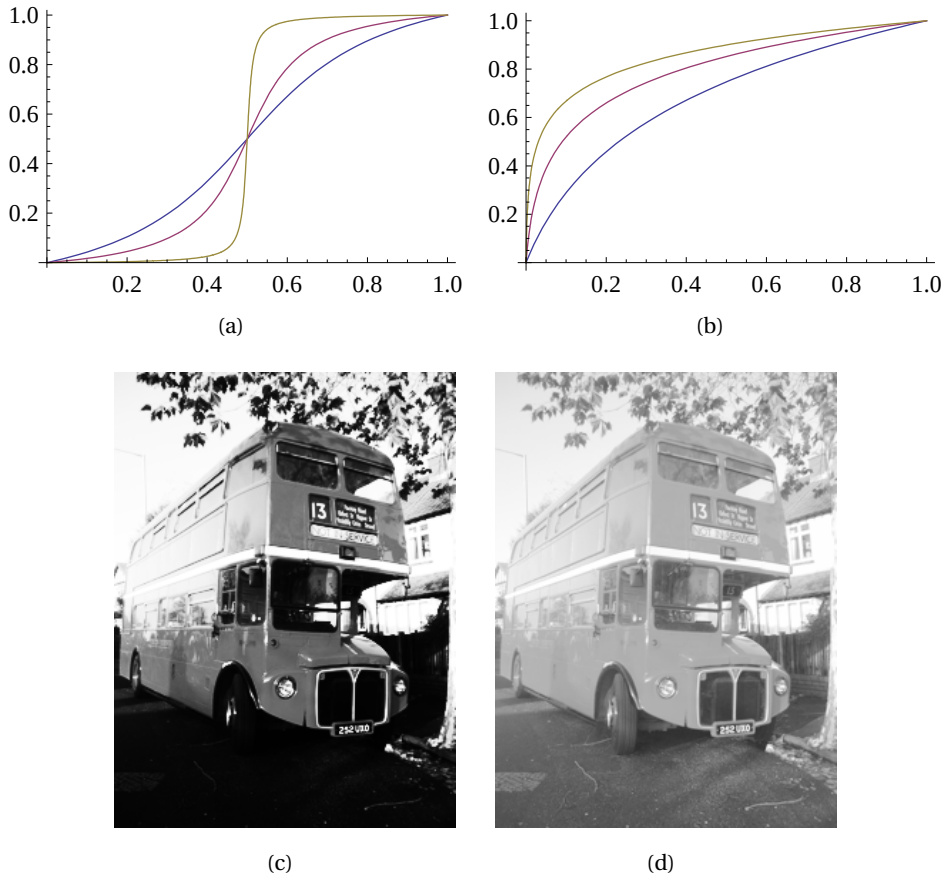



**Figure 8.8**. The plots in (a) and (b) show some functions that can be used to improve the contrast of an image. In (c) the middle function in (a) has been applied to the intensity values of the image in figure 8.6c, while in (d) the middle function in (b) has been applied to the same image.

somehow spread out the values. This can be accomplished by applying a function $f$ to the intensity values, i.e., new intensity values are computed by the formula

$$\hat{p}_{i,j} = f(p_{i,j})$$

for all $i$ and $j$. If we choose $f$ so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect.

Figure 8.8 shows some examples. The functions in the left plot have quite large derivatives near $x = 0.5$ and will therefore increase the contrast in images with a concentration of intensities with value around 0.5. The functions are all on the form

$$f_n(x) = \frac{\arctan\big(n(x-1/2)\big)}{2\arctan(n/2)} + \frac{1}{2}. \tag{8.1}$$

For any $n \neq 0$ these functions satisfy the conditions $f_n(0) = 0$ and $f_n(1) = 1$. The three functions in figure 8.8a correspond to $n = 4$, 10, and 100.

Functions of the kind shown in figure 8.8b have a large derivative near $x = 0$ and will therefore increase the contrast in an image with a large proportion of small intensity values, i.e., very dark images. The functions are given by

$$g_\epsilon(x) = \frac{\ln(x+\epsilon) - \ln\epsilon}{\ln(1+\epsilon) - \ln\epsilon}, \tag{8.2}$$

and the ones shown in the plot correspond to $\epsilon = 0.1$, 0.01, and 0.001.

In figure 8.8c the middle function in (a) has been applied to the image in figure 8.6c. Since the image was quite well balanced, this has made the dark areas too dark and the bright areas too bright. In figure 8.8d the function in (b) has been applied to the same image. This has made the image as a whole too bright, but has brought out the details of the road which was very dark in the original.

**Observation 8.11.** *Suppose a large proportion of the intensity values $p_{i,j}$ of a grey-level image $P$ lie in a subinterval $I$ of $[0,1]$. Then the contrast of the image can be improved by computing new intensities $\hat{p}_{i,j} = f(p_{,j})$ where $f$ is a function with a large derivative in the interval $I$.*

We will see more examples of how the contrast in an image can be enhanced when we try to detect edges below.

### 8.2.6 Smoothing of an image

When we considered filtering of digital sound in section 4.4.2 of the Norwegian notes, we observed that replacing each sample of a sound by an average of the sample and its neighbours dampened the high frequencies of the sound. We can do a similar operation on images.

Consider the array of numbers given by

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix}. \tag{8.3}$$

We can smooth an image with this array by placing the centre of the array on a pixel, multiplying the pixel and its neighbours by the corresponding weights, summing up and dividing by the total sum of the weights. More precisely, we would compute the new pixels by

$$\hat{p}_{i,j} = \frac{1}{16}\big(4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}\big).$$

Since the weights sum to one, the new intensity value $\hat{p}_{i,j}$ is a weighted average of the intensity values on the right. The array of numbers in (8.3) is in fact an example of a computational molecule, see figure 7.3. For simplicity we have omitted the details in the drawing of the computational molecule. We could have used equal weights for all nine pixels, but it seems reasonable that the weight of a pixel should be larger the closer it is to the centre pixel.

As for audio, the values used are taken from Pascal's triangle, since these weights are known to give a very good smoothing effect. A larger filter is given by the array

$$\frac{1}{1024}\begin{pmatrix}1 & 6 & 15 & 20 & 15 & 6 & 1\\ 6 & 36 & 90 & 120 & 90 & 36 & 6\\ 15 & 90 & 225 & 300 & 225 & 90 & 15\\ 20 & 120 & 300 & 400 & 300 & 120 & 20\\ 15 & 90 & 225 & 300 & 225 & 90 & 15\\ 6 & 36 & 90 & 120 & 90 & 36 & 6\\ 1 & 6 & 15 & 20 & 15 & 6 & 1\end{pmatrix}. \tag{8.4}$$

These numbers are taken from row six of Pascal's triangle. More precisely, the value in row $k$ and column $l$ is given by the product $\binom{6}{k}\binom{6}{l}$. The scaling $1/4096$ comes from the fact that the sum of all the numbers in the table is $2^{6+6} = 4096$.

The result of applying the two filters in (8.3) and (8.4) to an image is shown in figure 8.9 (b) and (c) respectively. The smoothing effect is clearly visible.

(a) (b) (c)

**Figure 8.9**. The images in (b) and (c) show the effect of smoothing the image in (a).

**Observation 8.12.** *An image $P$ can be smoothed out by replacing the intensity value at each pixel by a weighted average of the intensity at the pixel and the intensity of its neighbours.*

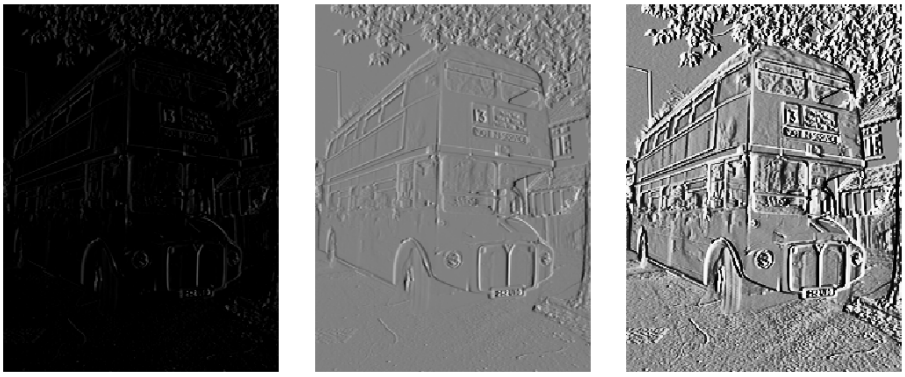
### 8.2.7 Detecting edges

The final operation on images we are going to consider is edge detection. An edge in an image is characterised by a large change in intensity values over a small distance in the image. For a continuous function this corresponds to a large derivative. An image is only defined at isolated points, so we cannot compute derivatives, but we have a perfect situation for applying numerical differentiation. Since a grey-level image is a scalar function of two variables, the numerical differentiation techniques from section 7.2 can be applied.

**Partial derivative in $x$-direction.** Let us first consider computation of the partial derivative $\partial P/\partial x$ at all points in the image. We use the familiar approximation

$$\frac{\partial P}{\partial x}(i,j) = \frac{p_{i+1,j} - p_{i-1,j}}{2}, \tag{8.5}$$

where we have used the convention $h = 1$ which means that the derivative is measured in terms of 'intensity per pixel'. We can run through all the pixels in the image and compute this partial derivative, but have to be careful for $i = 1$ and $i = m$ where the formula refers to non-existing pixels. We will adapt the simple convention of assuming that all pixels outside the image have intensity 0. The result is shown in figure 8.10a.



(a) (b) (c)

**Figure 8.10**. The image in (a) shows the partial derivative in the $x$-direction for the image in 8.6. In (b) the intensities in (a) have been normalised to $[0,1]$ and in (c) the contrast as been enhanced with the function $f_{50}$, equation 8.1.

This image is not very helpful since it is almost completely black. The reason for this is that many of the intensities are in fact negative, and these are just displayed as black. More specifically, the intensities turn out to vary in the interval $[-0.424, 0.418]$. We therefore normalise and map all intensities to $[0,1]$. The result of this is shown in (b). The predominant colour of this image is an average grey, i.e, an intensity of about 0.5. To get more detail in the image we therefore try to increase the contrast by applying the function $f_{50}$ in equation 7.6 to each intensity value. The result is shown in figure 8.10c which does indeed show more detail.

It is important to understand the colours in these images. We have computed the derivative in the $x$-direction, and we recall that the computed values varied in the interval $[-0.424, 0.418]$. The negative value corresponds to the largest average decrease in intensity from a pixel $p_{i-1,j}$ to a pixel $p_{i+1,j}$. The positive value on the other hand corresponds to the largest average increase in intensity. A value of 0 in figure 8.10a corresponds to no change in intensity between the two pixels.

When the values are mapped to the interval $[0,1]$ in figure 8.10b, the small values are mapped to something close to 0 (almost black), the maximal values are mapped to something close to 1 (almost white), and the values near 0 are mapped to something close to 0.5 (grey). In figure 8.10c these values have just been emphasised even more.

Figure 8.10c tells us that in large parts of the image there is very little variation in the intensity. However, there are some small areas where the intensity

changes quite abruptly, and if you look carefully you will notice that in these areas there is typically both black and white pixels close together, like down the vertical front corner of the bus. This will happen when there is a stripe of bright or dark pixels that cut through an area of otherwise quite uniform intensity.

Since we display the derivative as a new image, the denominator is actually not so important as it just corresponds to a constant scaling of all the pixels; when we normalise the intensities to the interval $[0,1]$ this factor cancels out.

We sum up the computation of the partial derivative by giving its computational molecule.

**Observation 8.13.** *Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial x$ of the image can be computed with the computational molecule*

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

As we remarked above, the factor $1/2$ can usually be ignored. We have included the two rows of 0s just to make it clear how the computational molecule is to be interpreted; it is obviously not necessary to multiply by 0.

**Partial derivative in $y$-direction.** The partial derivative $\partial P/\partial y$ can be computed analogously to $\partial P/\partial x$.

**Observation 8.14.** *Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial y$ of the image can be computed with the computational molecule*

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

The result is shown in figure 8.12b. The intensities have been normalised and the contrast enhanced by the function $f_{50}$ in (8.1).

(a) (b) (c)

**Figure 8.11**. Computing the gradient. The image obtained from the computed gradient is shown in (a) and in (b) the numbers have been normalised. In (c) the contrast has been enhanced with a logarithmic function.

**The gradient.** The gradient of a scalar function is often used as a measure of the size of the first derivative. The gradient is defined by the vector

$$\nabla P = \left( \frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right),$$

so its length is given by

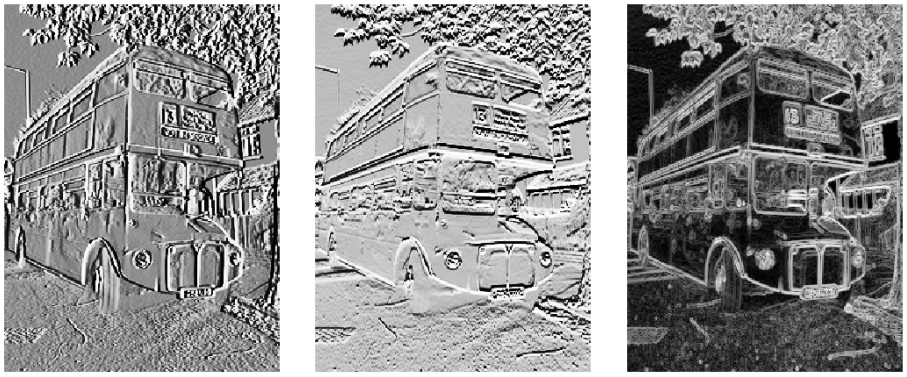
$$|\nabla P| = \sqrt{\left( \frac{\partial P}{\partial x} \right)^2 + \left( \frac{\partial P}{\partial y} \right)^2}.$$

When the two first derivatives have been computed it is a simple matter to compute the gradient and its length, and the resulting image is shown in figure 8.11c.

The image of the gradient looks quite different from the images of the two partial derivatives. The reason is that the numbers that represent the length of the gradient are (square roots of) sums of squares of numbers. This means that the parts of the image that have virtually constant intensity (partial derivatives close to 0) are coloured black. In the images of the partial derivatives these values ended up in the middle of the range of intensity values, with a final colour of grey, since there were both positive and negative values.

Figure 8.11a shows the computed values of the gradient. Although it is possible that the length of the gradient could become larger than 1, the maximum value in this case is about 0.876. By normalising the intensities we therefore increase the contrast slightly and obtain the image in figure 8.11b.

To enhance the contrast further we have to do something different from what was done in the other images since we now have a large number of intensities near 0. The solution is to apply a function like the ones shown in figure 8.8b



(a) (b) (c)

**Figure 8.12**. The first-order partial derivatives in the $x$-direction (a) and $y$-direction (b), and the length of the gradient (c). In all images, the computed numbers have been normalised and the contrast enhanced.

to the intensities. If we use the function $g_{0.01}$ defined in equation(8.2) we obtain the image in figure 8.11c.

### 8.2.8 Comparing the first derivatives

Figure 8.12 shows the two first-order partial derivatives and the gradient. If we compare the two partial derivatives we see that the $x$-derivative seems to emphasise vertical edges while the $y$-derivative seems to emphasise horizontal edges. This is precisely what we must expect. The $x$-derivative is large when the difference between neighbouring pixels in the $x$-direction is large, which is the case across a vertical edge. The $y$-derivative enhances horizontal edges for a similar reason.

The gradient contains information about both derivatives and therefore emphasises edges in all directions. It also gives a simpler image since the sign of the derivatives has been removed.

### 8.2.9 Second-order derivatives

To compute the three second order derivatives we apply the corresponding computational molecules which we described in section 7.2.

**Observation 8.15** (Second order derivatives of an image)**.** *The second order derivatives of an image $P$ can be computed by applying the computational*

(a) (b) (c)

**Figure 8.13**. The second-order partial derivatives in the $x$-direction (a) and $xy$-direction (b), and the $y$-direction (c). In all images, the computed numbers have been normalised and the contrast enhanced.

*molecules*

$$\frac{\partial^2 P}{\partial x^2}: \quad \begin{pmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\frac{\partial^2 P}{\partial y \partial x}: \quad \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix},$$

$$\frac{\partial^2 P}{\partial y^2}: \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

With the information in observation 8.15 it is quite easy to compute the second-order derivatives, and the results are shown in figure 8.13. The computed derivatives were first normalised and then the contrast enhanced with the function $f_{100}$ in each image, see equation 8.1.

As for the first derivatives, the $xx$-derivative seems to emphasise vertical edges and the $yy$-derivative horizontal edges. However, we also see that the second derivatives are more sensitive to noise in the image (the areas of grey are less uniform). The mixed derivative behaves a bit differently from the other two, and not surprisingly it seems to pick up both horizontal and vertical edges.

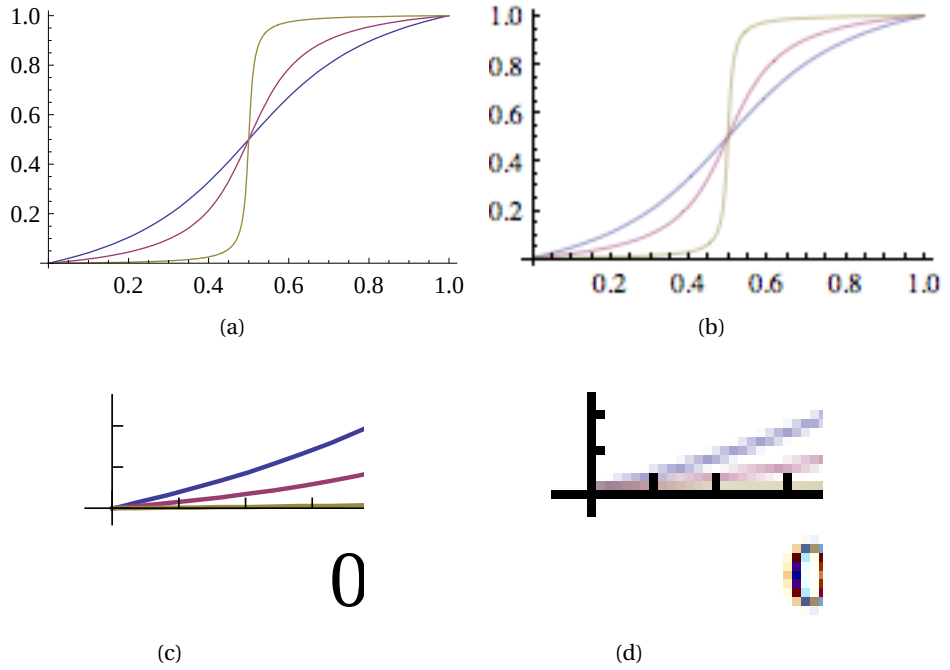



**Figure 8.14**. The difference between vector graphics ((a) and (c)) and raster graphics ((b) and (d)).

## 8.3 Image formats

Just as there are many audio formats, there are many image formats, and in this section we will give a superficial description of some of them. Before we do this however, we want to distinguish between two important types of graphics representations.

### 8.3.1 Raster graphics and vector graphics

At the beginning of this chapter we saw that everything that is printed on a computer monitor or by a printer consists of small dots. This is a perfect match for digital images which also consist of a large number of small dots. However, as we magnify an image, the dots in the image become visible as is evident in figure 8.2.

In addition to images, text and various kinds of line art (drawings) are also displayed on monitors and printed by printers, and must therefore be represented in terms of dots. There is a big difference though, in how these kinds of graphical images are stored. As an example, consider the plots in figure 8.14. In figure (c), the plot in (a) has been magnified, without any dots becoming visible.

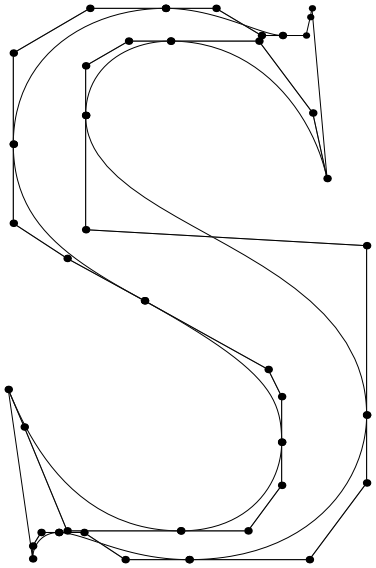


**Figure 8.15**. The character 'S' in the font Times Roman. The dots are parameters that control the shape of the curves.

In (d), the plot in (b) has been magnified, and here the dots have become clearly visible. The difference is that while the plots in (b)-(d) are represented as an image with a certain number of dots, the plots in (a)-(d) are represented in terms of mathematical primitives like lines and curves — this is usually referred to as a *vector representation* or *vector graphics*. The advantage of vector graphics is that the actual dots to be used are not determined until the figure is to be drawn. This means that in figure (c) the dots which are drawn were not determined until the magnification was known. On the other hand, the plot in (b) was saved as an image with a fixed number of dots, just like the pictures of the bus earlier in the chapter. So when this image is magnified, the only possibility is to magnify the dots themselves, which inevitably produces a grainy picture like the one in(d).

In vector graphics formats all elements of a drawing are represented in terms of mathematical primitives. This includes all lines and curves as well as text. A line is typically represented by its two endpoints and its width. Curved shapes are either represented in terms of short connected line segments or smoothly connected polynomial curve segments. Whenever a drawing on a monitor or printer is requested, the actual dots to be printed are determined from the mathematical representation. In particular this applies to fonts (the graphical shapes of characters) which are usually represented in terms of quadratic or cubic polynomial curves (so-called *Bezier curves*), see figure 8.15 for an example.

**Fact 8.16.** *In vector graphics a graphical image is represented in terms of mathematical primitives like lines and curves, and can be magnified without any loss in quality. In raster graphics, a graphical image is represented as a digital image, i,.e., in terms of pixels. As the image is magnified, the pixels become visible and the quality of the image deteriorates.*

### 8.3.2 Vector graphics formats

The two most common vector graphics formats are Postscript and PDF which are formats for representing two-dimensional graphics. There are also standards for three-dimensional graphics, but these are not as universally accepted.

**Postscript.** *Postscript* is a programming language developed by Adobe Systems in the early 1980s. Its principal application is representation of page images, i.e., information that may be displayed on a monitor or printed by a printer. The basic primitives in Postscript are straight line segments and cubic polynomial curves which are often joined (smoothly) together to form more complex shapes. Postscript fonts consist of Postscript programs which define the outlines of the shapes of all the characters in the font. Whenever a Postscript program needs to print something, software is required that can translate from the mathematical Postscript representation to the actual raster representation to be use on the output device. This software is referred to as a Postscript *interpreter* or *driver*. Postscript files are standard text files so the program that produces a page can be inspected (and edited) in a standard editor. A disadvantage of this is that Postscript files are coded inefficiently and require a lot of storage space. Postscript files have extension `.eps` or `.ps`.

Since many pages contain images, Postscript also has support for including raster graphics within a page.

**PDF.** *Portable Document Format* is a standard for representing page images that was also developed by Adobe. In contrast to Postscript, which may require external information like font libraries to display a page correctly, a PDF-file contains all the necessary information within itself. It supports the same mathematical primitives as Postscript, but codes the information in a compact format. Since a page may contain images, it is also possible to store a digital image in PDF-format. PDF-files may be locked so that they cannot be changed. PDF is in wide-spread use across computer platforms and is a preferred format for exchanging documents. PDF-files have extension `.pdf`.

### 8.3.3 Raster graphics formats

There are many formats for representing digital images. We have already mentioned Postscript and PDF; here we will mention a few more which are pure image formats (no support for vector graphics).

Before we describe the formats we need to understand a technical detail about representation of colour. As we have already seen, in most colour images the colour of a pixel is represented in terms of the amount of red, green and blue or $(r, g, b)$. Each of these numbers is usually represented by eight bits and can take integer values in the range 0–255. In other words, the colour information at each pixel requires three bytes. When colour images and monitors became commonly available in the 1980s, the file size for a 24-bit image file was very large compared to the size of hard drives and available computer memory. Instead of storing all 24 bits of colour information it was therefore common to create a table of 256 colours with which a given image could be represented quite well. Instead of storing the 24 bits, one could just store the table at the beginning of the file, and at each pixel, the eight bits corresponding to the correct entry in the table. This is usually referred to as eight-bit colour and the table is called a *look-up* table or *palette*. For large photographs, 256 colours is far from sufficient to obtain reasonable colour reproduction.

Images may contain a large amount of data and have great potential for both lossless and lossy compression. For lossy compression, strategies similar to the ones used for audio compression are used. This means that the data are transformed by a DCT or wavelet transform (these transforms generalise easily to images), small values are set to zero and the resulting data coded with a lossless coding algorithm.

Like audio formats, image formats usually contain information like resolution, time when the image was recorded and similar information at the beginning of the file.

**GIF.** *Graphics Interchange Format* was introduced in 1987 as a compact representation of colour images. It uses a palette of at most 256 colours sampled from the 24-bit colour model, as explained above. This means that it is unsuitable for colour images with continuous colour tones, but it works quite well for smaller images with large areas of constant colour, like logos and buttons on web pages. Gif-files are losslessly coded with a variant of the Lempel-Ziv-Welch algorithm. The extension of GIF-files is `.gif`.

**TIFF.** *Tagged Image File Format* is a flexible image format that may contain multiple images of different types in the same file via so-called 'tags'. TIFF sup-

ports lossless image compression via Lempel-Ziv-Welch compression, but may also contain JPEG-compressed images (see below). TIFF was originally developed as a format for scanned documents and supports images with one-bit pixel values (black and white). It also supports advanced data formats like more than eight bits per colour component. TIFF-files have extension `.tiff`.

**JPEG.** *Joint Photographic Experts Group* is an image format that was approved as an international standard in 1994. JPEG is usually lossy, but may also be lossless and has become a popular format for image representation on the Internet. The standard defines both the algorithms for encoding and decoding and the storage format. JPEG divides the image into $8 \times 8$ blocks and transforms each block with a Discrete Cosine Transform. These values corresponding to higher frequencies (rapid variations in colour) are then set to 0 unless they are quite large, as this is not noticed much by human perception. The perturbed DCT values are then coded by a variation of Huffman coding. JPEG may also use arithmetic coding, but this increases both the encoding and decoding times, with only about 5 % improvement in the compression ratio. The compression level in JPEG images is selected by the user and may result in conspicuous artefacts if set too high. JPEG is especially prone to artefacts in areas where the intensity changes quickly from pixel to pixel. The extension of a JPEG-file is `.jpg` or `.jpeg`.

**PNG.** *Portable Network Graphics* is a lossless image format that was published in 1996. PNG was not designed for professional use, but rather for transferring images on the Internet, and only supports grey-level images and rgb images (also palette based colour images). PNG was created to avoid a patent on the LZW-algorithm used in GIF, and also GIF's limitation to eight bit colour information. For efficient coding PNG may (this is an option) predict the value of a pixel from the value of previous pixels, and subtract the predicted value from the actual value. It can then code these error values using a lossless coding method called DEFLATE which uses a combination of the LZ77 algorithm and Huffman coding. This is similar to the algorithm used in lossless audio formats like Apple Lossless and FLAC. The extension of PNG-files is `.png`.

**JPEG 2000.** This lossy (can also be used as lossless) image format was developed by the Joint Photographic Experts Group and published in 2000. JPEG 2000 transforms the image data with a wavelet transform rather than a DCT. After significant processing of the wavelet coefficients, the final coding uses a version of arithmetic coding. At the cost of increased encoding and decoding times,

JPEG 2000 leads to as much as 20 % improvement in compression ratios for medium compression rates, possibly more for high or low compression rates. The artefacts are less visible than in JPEG and appear at higher compression rates. Although a number of components in JPEG 2000 are patented, the patent holders have agreed that the core software should be available free of charge, and JPEG 2000 is part of most Linux distributions. However, there appear to be some further, rather obscure, patents that have not been licensed, and this may be the reason why JPEG 2000 is not used more. The extension of JPEG 2000 files is `.jp2`.