Chapter 2
Digital Imaging Fundamentals

2.1 Digital Image

Visually, a *digital image* is a rectangular array of (nominally) square elements called *pixels*, each showing a colour. Figure 2.1 shows a simple example, displayed on a screen (if you are reading this as an e-book) or printed on paper (if you are reading it as a print book).

2.2 sRGB Colour Space

sRGB is a standard [1] defining a colour space and viewing conditions for digital images [2]. It is available in virtually all current personal computers, digital cameras, scanners, displays and printers.

In brief, sRGB has:

Three *variables*: red, green and blue.
In each variable, a range of integer *intensities* R, G, B, where $0 \leq R \leq 255$, $0 \leq G \leq 255$, $0 \leq B \leq 255$.
In the whole space, $256^3 = 16.7$ million *colours*, where a colour is an additive mixture of three intensities: (R, G, B).
A subset of 256 *neutrals*, colours where $R = G = B$.

The sRGB variables are defined as three primary *light sources* (the same as for HDTV [4]), which have the CIExyY chromaticity coordinates [3]:

R: x = 0.64, y = 0.33.
G: x = 0.30, y = 0.60.
B: x = 0.15, y = 0.06.
White point: x = 0.3127, y = 0.3290.

Figure 2.2a shows the three primary colours, which combine additively as white, and (b) shows the sRGB gamut within the full CIE gamut. Thus, sRGB colour space is

A. Parkin, *Computing Colour Image Processing*,
https://doi.org/10.1007/978-3-319-74076-8_2

Fig. 2.1 A simple digital image. The array has 8 columns and 8 rows, hence 64 pixels, which are coloured white, black, red, green and blue

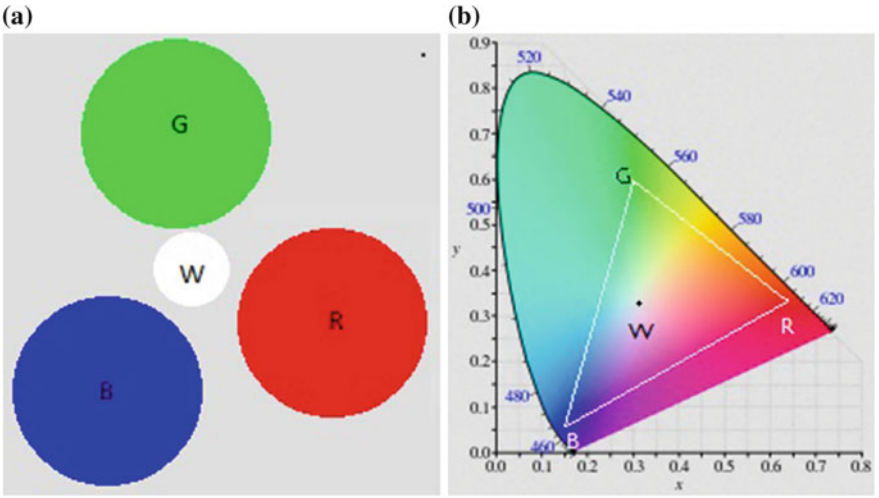

Fig. 2.2 **a** The three fundamental sRGB colours, which combine additively to show white. **b** The gamut of sRGB colours lies within the triangle of the three fundamental colours, which lies within the CIExyY chromaticity chart. (Notice that since this chromaticity chart is itself displayed here in sRGB, the colours are merely indicative, not CIE-accurate)

a subspace of CIE colour space. Any sRGB colour has a CIE equivalent, and some but not all CIE colours have an sRGB equivalent.

Visually, sRGB colour space is best modelled as a Cartesian cube, shown in Fig. 2.3 in front and back views. One vertex of the cube is the origin (0 0 0) black. The three edges from the origin are coordinate axes calibrated in integer steps of intensity from 0 to 255. The R-axis goes from (0 0 0) black to (255 0 0) red, the G-axis to (0 255 0) green and the B-axis to (0 0 255) blue. The other four vertices

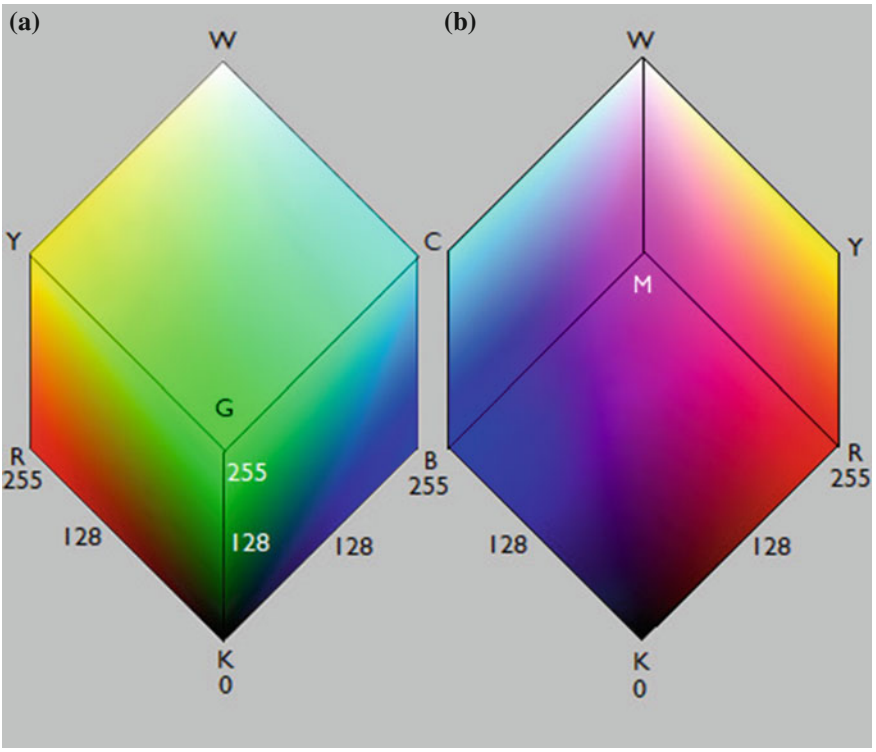

Fig. 2.3 SRGB cube model. **a** Front view. **b** Back view

are then (255 255 0) yellow, (255 255 255) white, (0 255 255) cyan and (255 0 255) magenta. The four body diagonals of the cube meet in the centre at (127 127 127) mid-grey. (Notice that in sRGB there is a systematic equivocation between intensity values 127 and 128. The middle value between 0 and 255 is 127.5, which can be arbitrarily rounded down or up without visual effect.)

2.3 Numerical Representation

Numerically, a digital image has:

Width W pixels.
Height H pixels.
Hence, *Extent* $E = W \times H$ pixels.

and a pixel has:

Location (X, Y), integers where $0 \leq X < W - 1$ and $0 \leq Y < H - 1$.
Colour (R, G, B), integers where $0 \leq R < 255$, $0 \leq G < 255$, $0 \leq B < 255$.

2.4 Scan Sequence

The conventional *scan sequence* of pixels in an image is shown in Fig. 2.4. Then, the complete numerical representation of the image is W, H followed by a list of (R, G, B) triples in scan sequence. For example, the simple image in Fig. 2.1 is represented numerically by the sequence:

```
8,8,(255,0,0),(255,255,255),(0,0,0),(255,255,255),(0,0,0),
(255,255,255),(0,0,0)),(255,255,255),(0,0,0),(255,255,255),
(0,0,0),(255,255,255),(0,0,0)),(255,255,255),(0,0,0),
(255,255,255),(255,0,0),(255,255,255),(0,0,0),(255,255,255),
(0,0,0),(255,255,255),(0,0,0)),(255,255,255),(0,0,0),
(255,255,255),(0,0,0),(0,255,0),(0,0,0)),(255,255,255),
(0,0,0),(255,255,255),(255,0,0),(255,255,255),(0,0,0),
(255,255,255),(0,0,0),(255,255,255),(0,0,0)),(255,255,255),
(0,0,0),(255,255,255),(0,0,0),(255,255,255),(0,0,0)),
(255,255,255),(0,0,0),(255,255,255),(255,0,0),(255,255,255),
(0,0,0),(255,255,255),(0,0,0),(255,255,255),(0,0,0)),
(255,255,255),(0,0,0),(255,255,255),(0,0,0),(255,255,255),
(0,0,0)),(255,255,255),(0,0,0),(0,0,255)
```

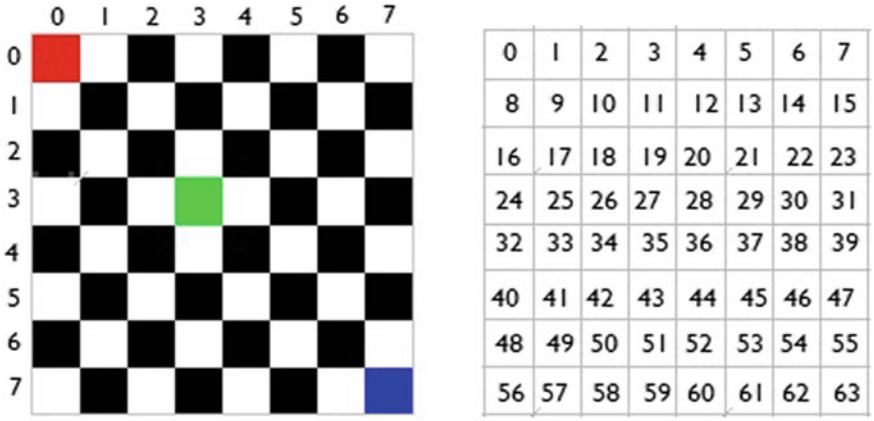

Fig. 2.4 Scan sequence

2.5 Computer Processing of Images

Computationally, the numerical representation of an image can be created, stored, transformed, displayed, printed and transmitted via a computer, using any suitable programming language. Python [5] is particularly suitable, as a scripting language with the associated Python Imaging Library (PIL) [6], Tkinter, numpy and scipy languages. For example, the following Python script will select, open and display any .bmp image in the current user directory:

```
# pyopsh: Python script to open and show a .bmp image.
# Written by Alan Parkin 2017.

from PIL import Image
import os, sys

# Enter any .bmp filemame in working directory.
imagefilename = raw_input(''Image filename?'')

#Open it, print filename, W, H, E, and show it.
im = Image.open(imagefilename)
width = im.size[0]
height = im.size[1]
extent = width * height
print ''filename'', imagefilename
print ''width W'', width
print ''height H'', height
print ''extent E'', extent
im.show()
```

The Python output is:

```
Python 2.7.11 |Anaconda 2.4.1 (64-bit)| (default,
 Jan 19 2016, 12:08:31) [MSC v.1500 64 bit (AMD64)]
 on win32
Type ''copyright'', ''credits'' or ''license()''
 for more information.
>>>
================ RESTART: C:\Users\Alan
 Parkinalan\pyopsh.py ================
Image filename? sim8.bmp
filename sim8.bmp
width W 8
height H 8
extent E 64
>>>
```

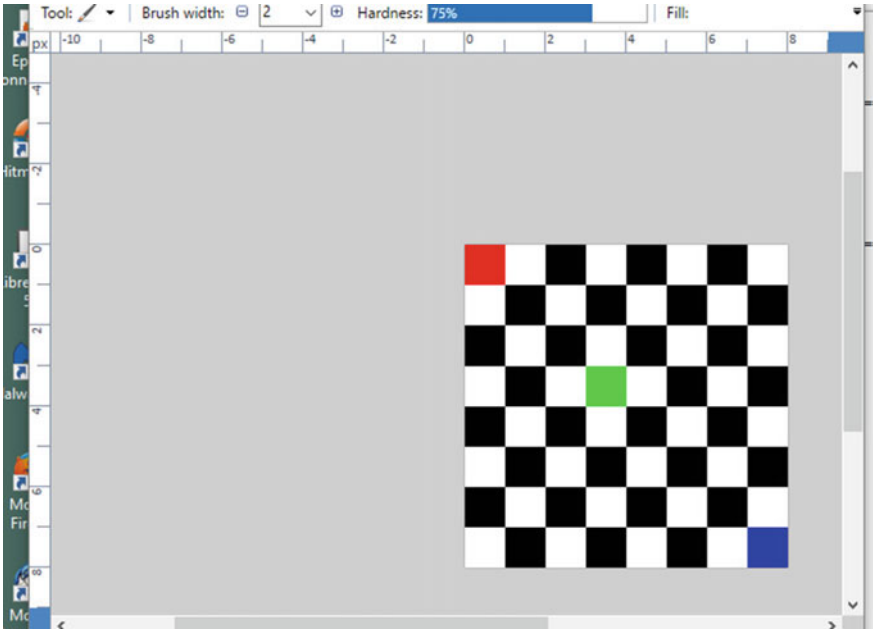

Fig. 2.5 Simple image sim8.bmp opened and displayed in Paint (where it is magnified by a factor of eight, as shown by the marginal rulers)

and the image displayed in the Microsoft Paint image editor, as shown in Fig. 2.5, whence it can be saved to storage or otherwise disposed of.

2.6 Location Resolution

Numerically, the *location resolution* of an image is the smallest detail which it can show. In a digital image, pixels are indivisible, so the smallest detail is one pixel distinguished (by colour) from its neighbours. Numerically, the *location resolution limit* $LOCRES = 1/E$, one pixel in E, where E is the extent, that is, width $\times$ height, of the image.

Notice that pixels, extent and location resolution are of indefinite size: they take on size only when displayed on a device which has a fixed *pixel pitch* of so many pixels per inch (ppi) or pixels per millimetre (ppm). For example, the simple image in Fig. 2.1 has width $W = 8$, height $H = 8$, extent $E = 64$ and location resolution limit $LOCRES = 1/64$. The smallest detail it can show is 1/64 of the extent, at whatever size it is displayed or printed.

A digital image with large extent E, such as a typical camera or scanner image, has extremely fine location resolution: perhaps 1 in 10 million or more. Such fine resolution is often unnecessary and inconvenient. It is common practice to reduce location resolution to suit the purpose in hand, by resizing down or by severe cropping. For methods, see Chap. 5.

2.7 Colour Resolution

Numerically, the *colour resolution* of an image is the least difference of colours which it can show. In a digital image, the least difference is one *step* in the colour space of the image. Numerically, the sRGB colour space has three axes, each with $S = 256$ steps of intensity; hence, the cube contains $S^3 = 256 \times 256 \times 256 = 16.7$ million colours, each different from its neighbours. Call this number the *diversity* D of the sRGB colour space. We define the *colour resolution limit COLRES* as $1/D$, one colour in D.

A digital image with large colour diversity D, such as a typical sRGB camera or scanner image, has an extremely fine colour resolution: 1 in 16.7 million. Such fine resolution is often unnecessary and inconvenient: as when, for example, we want to analyse colour distribution in an image, or discern essentials from inessentials, or make systematic changes of colour. We can simplify an image by reducing colour diversity to a subspace of sRGB, thus coarsening colour resolution.

We can define a series of subspaces, or restricted *palettes*, within sRGB by taking fewer than $S = 256$ steps of intensity per axis of the sRGB cube. For example, a minimal palette P2 has $S = 2$, hence diversity $D = S^3 = 8$ colours, just those at the vertices of the cube. Figure 2.6 shows palette P2.

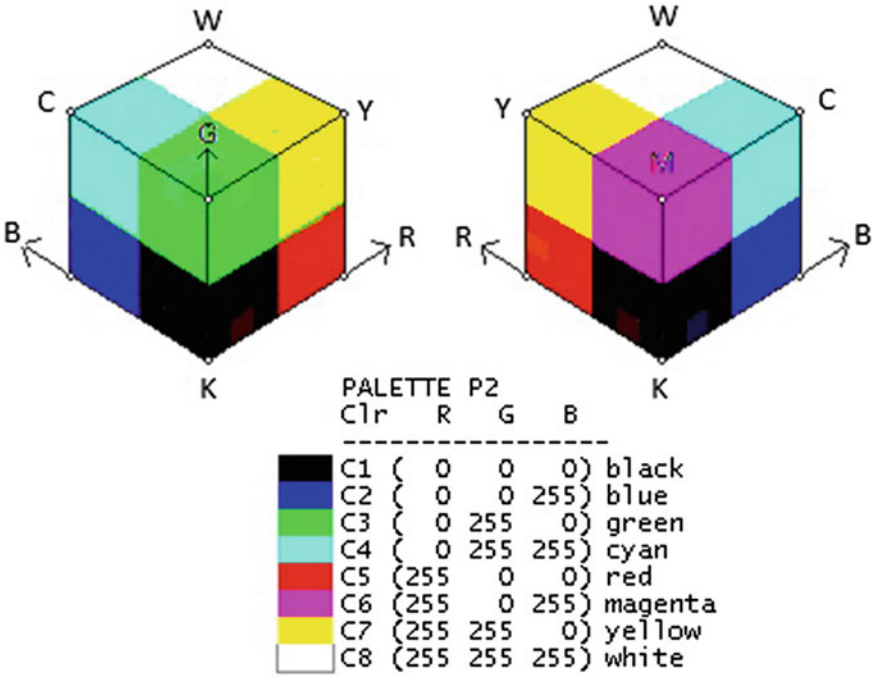

Fig. 2.6 Palette P2, a subset of sRGB, with $S = 2$ steps per cube axis

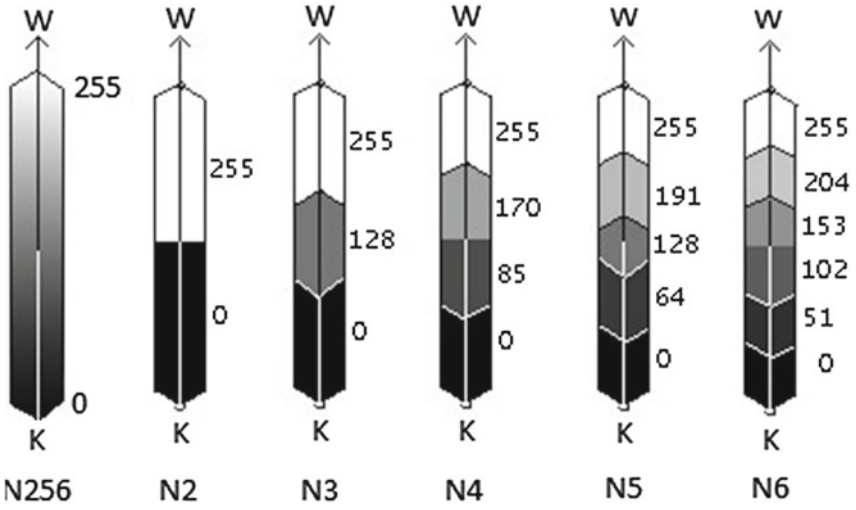

Fig. 2.7 Neutral palette N256, and subsets N2, N3, N4, N5, N6 with 2, 3, 4, 5 and 6 steps along the axis

An important subspace of sRGB is the *neutral palette* or greyscale N256, where ($R = G = B$). This has just one axis, the body diagonal of the cube from black (0,0,0) to white (255,255,255). It has $S = 256$ steps of intensity I, and hence contains just 256 neutrals, with diversity $D = 256$, and $COLRES = 1/256$, one grey in D. A series of neutral greyscales can be made by taking fewer than $S = 256$ steps of intensity on the diagonal axis. For example, a minimal neutral greyscale N2 has $S = 2$, hence diversity $D = 2$ colours, just black and white. Figure 2.7 shows neutral palettes N256, N2, N3, N4, N5 and N6.

For methods of reducing colour resolution, see Chap. 6.

References

1. sRGB Color Space. https://webstore.iec.ch/publication/6168
2. sRGB Color Space. https://en.wikipedia.org/wiki/SRGB
3. CIE 1931 color space. https://en.wikipedia.org/wiki/CIE_1931_color_space
4. ITU Rec.709. https://en.wikipedia.org/wiki/Rec._709
5. Python Software Foundation. https://www.python.org
6. Fredrik Lundh effbot. http://effbot.org