COMP10001 Foundations of Computing
Images

Semester 2, 2018

Chris Leckie & Nic Geard

THE UNIVERSITY OF MELBOURNE

Lecture Agenda

- Last lecture:
  - HTML
- This lecture:
  - Image representation/digitisation
  - Image manipulation

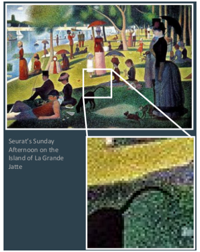
The Power of Images

- Many examples of images that changed history ...
- Facebook: 350 million photos uploaded every day on average
- Instagram: 70 million photos uploaded every day on average
- Humans are visual creatures

Raster vs. Vector Images

Raster images:

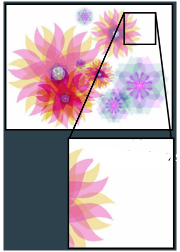
represented as individual colour "pixels"; navigate from top-left row-by-row to bottom-right



Popular formats: PNG, JPEG, GIF, PDF

Vector images:

represented as points, lines, curves; order of vectors unimportant

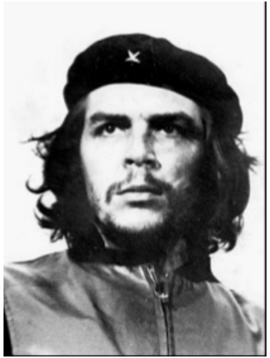


Popular format: SVG, PDF

Types of Digital Image

Binary:

Grayscale:



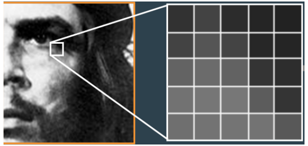
Colour:

Digital Image Representation I

- For grayscale (raster) images, each pixel is a single integer specifying its grayscale value (e.g. 0 = black; 255 = white)

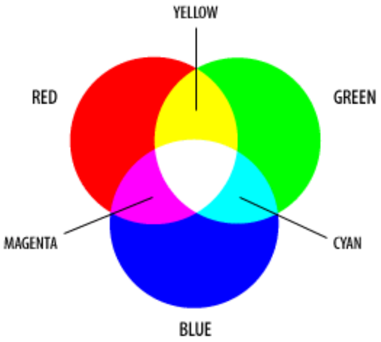
50	68	44	37	34
67	85	68	39	37
99	107	97	52	45
114	118	120	92	52
112	115	115	114	88

$\Downarrow$ $\Uparrow$



Digital Image Representation II

- For colour (raster) images, each pixel is instead a *triple* of integers, specifying the RGB values of the pixel (e.g. (0,0,0) = black; (255,0,0) = red)



Digital Image Representation III

- The RGB values are often expressed in the form of a "hexadecimal" value representing the 24-bit RGB value (e.g. (255,0,0) = (11111111,00000000,00000000) = (FF,00,00) = FF0000 = red):

E9A88C	EAA38D	E9A68C
E9A88C	ECA98C	E7A390
EBAA8E	ECA98F	EBA794

- For all raster images, the width and height of the image are also required in order to render the image from the list of pixel values

Computational Counting I

- Conventionally, we are used to representing numbers in decimal (base 10) format:

$$\begin{aligned} 2014_{10} &= 2 \times 1000 + 0 \times 100 + 1 \times 10 + 4 \times 1 \\ &= 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 4 \times 10^0 \end{aligned}$$

- Computers internally represent numbers in binary (base 2) format:

$$\begin{aligned} 1001_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 9 \end{aligned}$$

Computational Counting II

- A single-digit binary number (and the basic unit of storage/computational) is known as a "bit" (short for binary digit)
- Bits are generally processed as vectors of 8 bits (= a "byte" or "octet") or larger
- A convenient representation for bit sequences is "hexadecimal" (base 16); one byte = 8 bits = two "hex" digits (why?)
- The 16 hexadecimal digits:

Hex:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Dec:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bin:	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Computational Counting III

- Converting from bin(ary) to hex(adecimal):

$$
\begin{array}{ccc}
 & 110110_2 & \\
 & \Downarrow & \\
11_2 & & 0110_2 \\
\Downarrow & & \Downarrow \\
0011_2 & & 0110_2 \\
\Downarrow & & \Downarrow \\
3_{16} & & 6_{16} \\
 & \Downarrow & \\
 & 36_{16} &
\end{array}
$$

Computational Counting IV

- To indicate what “base” a (non-decimal) number is in, Python uses the following prefixes:
  - hexadecimal (base 16): `0x`
  - octal (base 8): `0o`
  - binary (base 2): `0b`

```
>>> 0b11001 == 0o31 == 25 == 0x19
True
```

Computational Counting V

- It is also possible to "cast" a (decimal) `int` to a string representation in one of the other bases using `hex`, `oct` and `bin`, resp.:

```
>>> hex(25)
'0x19'
>>> oct(25)
'0o31'
>>> bin(25)
'0b11001'
```

- NB, for memory and storage, sizes are generally reported in bytes ("B") vs. network speeds which are reported in bits ("b")

Computational Counting VI

- Because of the size/speed of modern-day computers/networks, numbers are usually reported in kilo-, mega-, giga-, tera- units:
  - kilo ("k" or "ki") $= 2^{10} = 1,024$
  - mega ("M" or "Mi") $= 2^{20} = 1,048,576$
  - giga ("G" or "Gi") $= 2^{30} = 1,073,741,824$
  - tera ("T" or "Ti") $= 2^{40} = 1,099,511,627,776$

  although, to confuse things, storage is often reported with a base of 10 rather than 2 (1kB $=$ 1000 bytes, etc.)

Creating an Image

We will use the Python Imaging Library (PIL/Pillow)
(PyX module for vector graphics)

```
from PIL import Image

width,height = 10,10

img = Image.new("RGB", (width,height))
pix = img.load()

for x in range(width):
    for y in range(height):
        pix[x,y] = (255,0,0)

img.save('red_block.png')
```

Image Processing in Python I

- Open an image file:

```
from PIL import Image

def open_img(fname):
    img = Image.open(fname)
    pix = img.load()
    mode = img.mode
    dimensions = img.size # width,height
    return dimensions,pix,mode
```

Image Processing in Python II

- Save an image file:

```
def save_img(dim,parray,fname,mode="RGB"):
    img_out = Image.new(mode, dim)
    pix_out = img_out.load()
    for x in range(dim[0]):
        for y in range(dim[1]):
            pix_out[x,y] = parray[x][y]
    img_out.save(fname)
    show_img(fname)
```

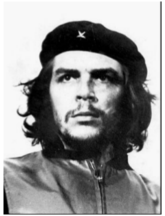
Image Processing in Python III

- Display an image file in HTML:

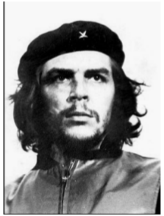
```
def show_img(fname):
    print('<html>')
    print('<body>')
    print('<img src="{fname}" alt="Yikes!"/>'\
              .format(fname=fname))
    print('</body>')
    print('</html>')
```

Reflecting an Image I

- Task: reflect an image in the vertical axis



$\Rightarrow$



Reflecting an Image I

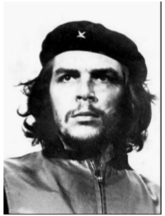
- Task: reflect an image in the vertical axis

```
from PIL import Image

def vertical(infile,outfile):
    dim,pix_in,mode = open_img(infile)
    img_out = Image.new(mode, dim)
    pix_out = img_out.load()
    for x in range(dim[0]):
        for y in range(dim[1]):
            pix_out[x,y] = pix_in[dim[0]-x-1,y]
    img_out.save(outfile)
```

Reflecting an Image II

- Task: reflect an image in the horizontal axis



$\Rightarrow$

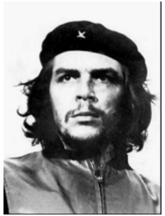
Reflecting an Image II

- Task: reflect an image in the horizontal axis

```
def horizontal(infile,outfile):
    dim,pix_in,mode = open_img(infile)
    img_out = Image.new(mode, dim)
    pix_out = img_out.load()
    for x in range(dim[0]):
        for y in range(dim[1]):
            pix_out[x,y] = pix_in[x,dim[1]-y-1]
    img_out.save(outfile)
```

Grayscale to Binary Conversion

- Task: convert a grayscale image to a binary image



$\Rightarrow$

Grayscale to Binary Conversion

- Task: convert a grayscale image to a binary image

```
from PIL import Image

def gray2binary(infile,outfile):
    dim,pxl,mode = open_img(infile)
    assert mode == "L"
    img_out = Image.new("1", dim)
    pix_out = img_out.load()
    threshold = 128
    for x in range(dim[0]):
        for y in range(dim[1]):
            if pxl[x,y] < threshold:
                pix_out[x,y] = 0
            else:
                pix_out[x,y] = 1
    img_out.save(outfile)
```

Colouring a Binary Image

- Task: recolour the white in a binary image

$\Rightarrow$

Colouring a Binary Image

- Task: recolour the white in a binary image

```
from PIL import image

def binary2colour(infile,outfile,newcol=(255,0,0)):
    dim,pxl,mode = open_img(infile)
    assert mode == "1"
    img_out = Image.new("RGB", dim)
    pix_out = img_out.load()
    for x in range(dim[0]):
        for y in range(dim[1]):
            if pxl[x,y] == 0:
                pix_out[x,y] = (0,0,0)
            else:
                pix_out[x,y] = newcol
    img_out.save(outfile)
```

Lecture Summary

- What are raster vs. vector images?
- How are raster images stored internally?
- How are colours represented?
- How do we process images in Python (variously)?