Image Representation and Processing

cs4: Computer Science Bootcamp

Çetin Kaya Koç
http://koclab.cs.ucsb.edu/teaching/cs4
cetinkoc@ucsb.edu

Pixel

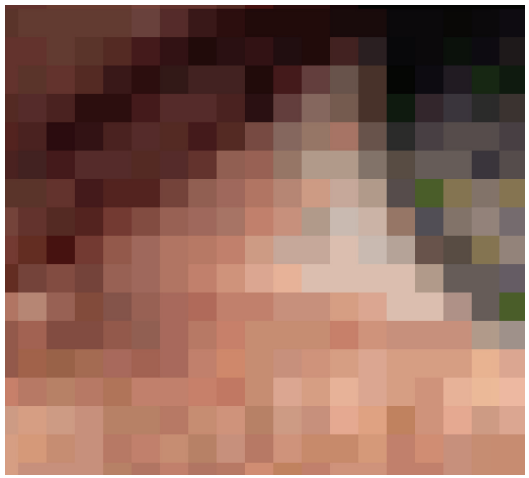
- A pixel, a picture element, is the smallest addressable element in an addressable display device
- It is smallest controllable piece of a picture represented on the screen
- The address of a pixel corresponds to its physical coordinates
- LCD pixels are manufactured in a two-dimensional grid, and are often represented using dots or squares
- Each pixel is a sample of an original image
- More samples provide more accurate representations of the original

Pixel

- In color image systems, a color is typically represented by three or four component intensities such as rgb (red, green, blue) or cmyb (cyan, magenta, yellow, black)
- Pixels can be used as a unit of measure such as: 2400 pixels per inch or 640 pixels per line
- The measures dots per inch (dpi) and pixels per inch (ppi) are sometimes used interchangeably
- dpi is a measure of a printer's density of dot (e.g., ink droplet) placement
- For example, a high-quality photographic image may be printed with 600 ppi on a 1200 dpi inkjet printer.

Image Size

- A typical representation of an image involves two parameter sets: Image size and the color intensities of each pixel
- Image size is the number of columns and rows in a rectangular image, such that each pixel is addressed by providing a column number and row number



RGB Colors

- On the other hand, each pixel will have 3 color intensities, according to the rgb (red, green, blue) model or 4 color intensities according to the cmyk (cyan, magenta, yellow, black) model
- The color intensity is an integer between 0 and a maximum number N
- 0 implies that color is nonexistent, while N implies it is the brightest
- The higher the value of N, the richer the color combinations
- For example N can be 255, so that each color intensity can be a number between 0 and 255, fitting into a single byte
- In the RGB model, we will have 3 bytes for each pixel, and therefore, $2^8 \times 2^8 \times 2^8 = 2^{24} = 16,777,216$ different colors

RGB Colors

- A pixel will have 3 color densities: red, green, blue, such that each value is an integer between 0 and 255, for example, $(r, g, b) = (175, 89, 67) = (AF, 59, 3E)$
- Pixels are also represented using 6-digit hex: AF593E
- In fact, this color has a name: Medium Brown (Crayola Brown)
- The basic colors, such as red, green, blue, brown, yellow, etc. are obtained by properly mixing the 3 colors: red, green, and blue

- An RGB color chart is found here: `https://www.rapidtables.com/web/color/RGB_Color.html`

RGB Matrix

- Consider an image of size 500x300 (500 columns, 300 rows)
- A particular pixel is addressed as (201,101) such that 201 is the column number and 101 is the row number

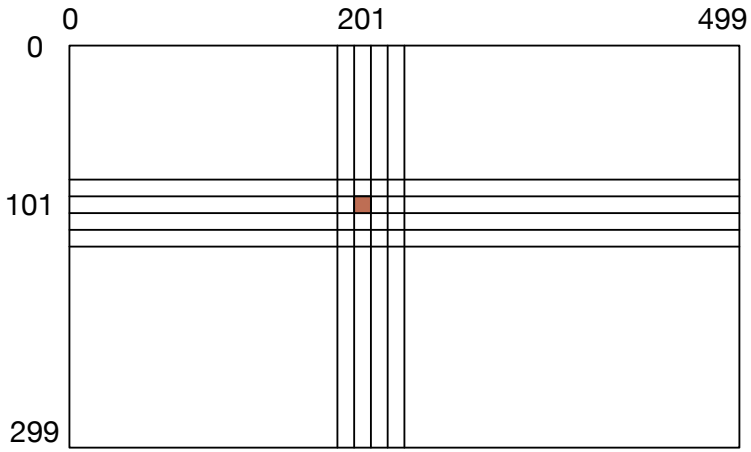


Image → Matrix

- Therefore, we can consider an image as an $n \times m$ matrix of integer entries such that each integer is 24 bits (between 0 and 16,777,215)
- Any operations we need to perform with or over this image will be using this matrix and its entries
- Some operations are as simple as displaying the image on a given display device (LCD, plasma, etc.)
- However, some other operations are significantly more challenging

Image Processing

- What operations can we do with an image?

- Color to grayscale conversion (to print on a printer)
- Changing its size (to fit into a window or a page)
- Edge detection (to detect movement over multiple images)
- Detect particular objects in image (recognition)
- Decide if two images are same or similar (similarity)
- ...

Image Processing using Python

- Python has a module called "cImage.py" which has basic image processing functions
- Here is a list of basic "cImage.py" functions:

```
import os
os.chdir("/Users/koc/Desktop/abc")
import cImage
mywin = cImage.ImageWin("leo",500,500)
im = cImage.FileImage("leo.gif")
im.draw(mywin)
p = im.getPixel(100,100)
r = p.getRed()
g = p.getGreen()
b = p.getBlue()
q = cImage.Pixel(175,89,67)
im.setPixel(100,100,q)
```

Converting a Color Image to Grayscale

- Gray RGB color code has equal red, green and blue values:

$$r = g = b$$

- We can read each pixel in the image, obtain the r, g, b values, take their average:

$$avg = (r + g + b)/3$$

and write back this average value in place of r, g, b values

Converting a Color Image to Grayscale

```
im = cImage.FileImage("image1.gif")
n = im.getWidth()
m = im.getHeight()
for i in range(n):
    for j in range(m):
        p = im.getPixel(i,j)
        r = p.getRed()
        g = p.getGreen()
        b = p.getBlue()
        avg = (r+g+b)//3
        q = cImage.Pixel(avg, avg, avg)
        im.setPixel(i,j,q)
im.save("image2.gif")
im.draw(mywin)
```

Halving an Image

- Consider an image of size $n \times m$
- Halving procedure will produce an image of size $(n/2) \times (m/2)$
- The division by 2 needs to be an integer division, e.g., $101/2 = 50$, since the column and row sizes can only be integers
- In Python we accomplish integer division by

```
newn = n//2
newm = m//2
```

Halving an Image

- Assume that we have a small image of size 8×6
- Thus we have 8 columns and 6 rows, all together 48 pixels
- The figure below depict the column and row indices (i, j) of pixels
- Furthermore, we also know that each pixel has 3 color intensities: red, green, blue values, each of which is between 0 and 255

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)

Halving an Image

- Halving an image may be accomplished by taking a neighborhood of $2 \times 2 = 4$ pixels, and throwing away 3 of them and keeping one

(i, j)	$(i+1, j)$
$(i, j+1)$	$(i+1, j+1)$

- For example, we can keep (i, j), "the top-left pixel", throwing away the other three
- We need to do this by starting from the pixel with index $(0, 0)$, and move to the right and to the bottom in the pixel matrix

$(0, 0)$	$(1, 0)$
$(0, 1)$	$(1, 1)$

- Keep (0,0) and throw away (1,0), (0,1), and (1,1)

Halving an Image

(0,0)	(1,0)	**(2,0)**	(3,0)	**(4,0)**	(5,0)	**(6,0)**	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	**(2,2)**	(3,2)	**(4,2)**	(5,2)	**(6,2)**	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	**(2,4)**	(3,4)	**(4,4)**	(5,4)	**(6,4)**	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)

(0,0)	**(2,0)**	**(4,0)**	**(6,0)**
(0,2)	**(2,2)**	**(4,2)**	**(6,2)**
(0,4)	**(2,4)**	**(4,4)**	**(6,4)**

→

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)

Halving an Image

- An inspection of the pixel indices show that we have the rule:

  If i and j are even integers, then keep pixel (i, j) and assign it to pixel $(i/2, j/2)$ in the new image, and throw away pixels $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$

- This gives us an algorithm for halving an image:

  1. Start with pixel $(i, j) = (0, 0)$
  2. Assign pixel (i, j) to pixel $(i/2, j/2)$ in the new image
  3. $i = i + 2$ and $j = j + 2$, and go to Step 2 if $i < n$ and $j < m$

Halving an Image

```
im1 = cImage.FileImage("image1.gif")
n = im1.getWidth()
m = im1.getHeight()
im2 = cImage.EmptyImage(n//2,m//2)
for i in range(0,n,2):
    for j in range(0,m,2):
        p = im1.getPixel(i,j)
        im2.setPixel(i//2,j//2,p)
im2.save("image2.gif")
im2.draw(mywin)
```

Doubling an Image

- Consider an image of size $n \times m$
- Doubling procedure will produce an image of size $(2n) \times (2m)$
- Assume that we have a small image of size 4×3
- Thus we have 4 columns and 3 rows, all together 12 pixels
- The figure below depict the column and row indices (i, j) of pixels

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)

Doubling an Image

- Doubling an image first requires an empty of size $(2n) \times (2m)$
- We then pick a pixel from the original image, and place it in the new image such that every neighborhood of 2×2 pixels get the same original pixel populated in 4 locations
- For example, if we pick pixel (0,0) from the original image, we make four copies of it, and place it in the new image in locations (0,0), (1,0), (0,1) and (1,1)

Doubling an Image

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)

(0,0)	(0,0)	(1,0)	(1,0)	(2,0)	(2,0)	(3,0)	(3,0)
(0,0)	(0,0)	(1,0)	(1,0)	(2,0)	(2,0)	(3,0)	(3,0)
(0,1)	(0,1)	(1,1)	(1,1)	(2,1)	(2,1)	(3,1)	(3,1)
(0,1)	(0,1)	(1,1)	(1,1)	(2,1)	(2,1)	(3,1)	(3,1)
(0,2)	(0,2)	(1,2)	(1,2)	(2,2)	(2,2)	(3,2)	(3,2)
(0,2)	(0,2)	(1,2)	(1,2)	(2,2)	(2,2)	(3,2)	(3,2)

Doubling an Image

```
im1 = FileImage("image1.gif")
n = im1.getWidth()
m = im1.getHeight()
im2 = EmptyImage(2*n,2*m)
for i in range(n):
    for j in range(m):
        p = im1.getPixel(i,j)
        im2.setPixel(2*i,2*j,p)
        im2.setPixel(2*i,2*j+1,p)
        im2.setPixel(2*i+1,2*j,p)
        im2.setPixel(2*i+1,2*j+1,p)
im2.save("image2.gif")
im2.draw(mywin)
```