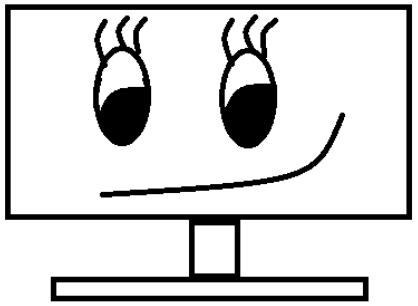
Seneca



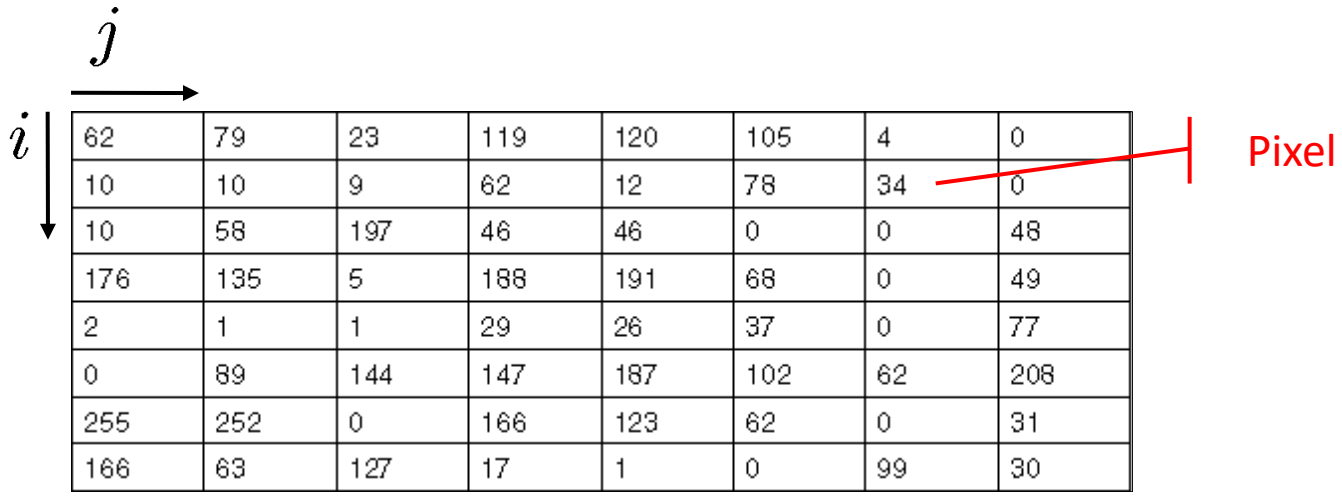
CVI620/ DPS920
Introduction to Computer Vision

Digital Images

Seneca College
Vida Movahedi

Digital images: Grayscale image

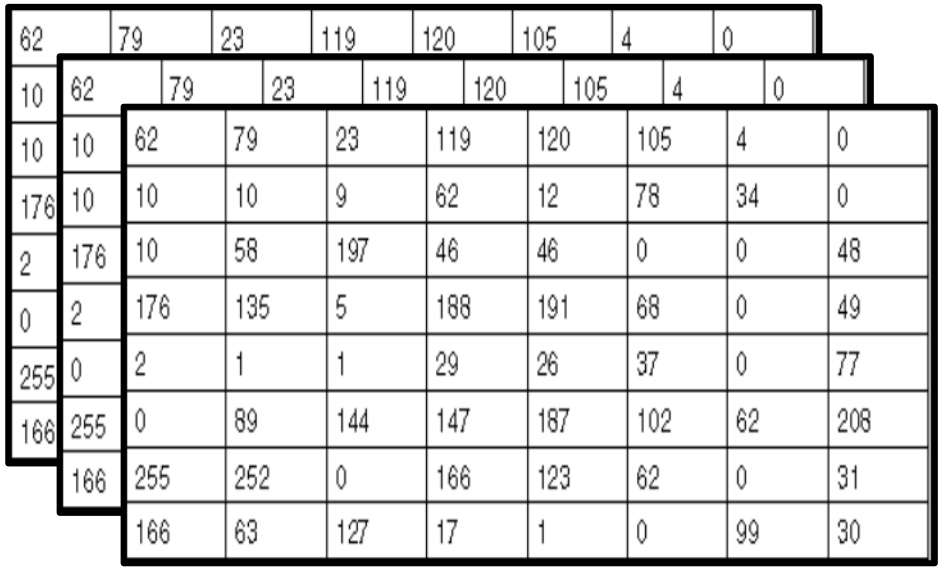
- A **grayscale** image is an array of brightness intensity values
- Often expressed in 8 bits (0 to 255)



62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Digital images: Color and Beyond

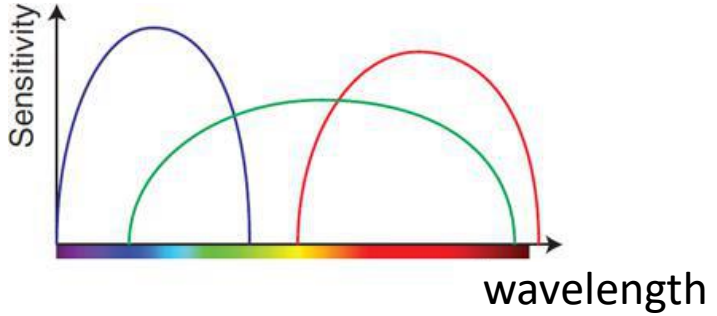
- A **color** image consists of three color maps
- Each color map is a 2D array, therefore a color image is a 3D array
- Multispectral images: may include frequencies beyond visible light spectrum
- Depth maps
  - Depth sensors, Kinect
  - Lidar (light and radar?)
  - Calculated using stereo, motion, shadows, structured light, etc.



62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Color

- Human vision system
  - Three different kinds of cones responding to different ranges in the color spectrum
- Digital cameras
  - Red, Green, Blue sensors (sensitive to different portions of the color spectrum)
- Standards
  - CIE RGB
  - CIE XYZ:
    - Y is the luminance or perceived brightness
  - CIELAB (L*a*b*)
    - Based on how human subjects perceive different colors
    - L* is the lightness
  - L*u*v*
  - YCbCr (used in compression algorithms)
  - Many more …



$$Y = 0.299R + 0.587G + 0.114B$$

OpenCV: Basics

Example- images

Source: OpenCV: Getting Started with Images

```
# Import libraries
import cv2 as cv
import sys

# Read an image
img = cv.imread("starry_night.jpg")

if img is None:
    sys.exit("Could not read the image.")

# Show image
cv.imshow("Display window", img)
k = cv.waitKey(0)

# Save image if 's' is pressed
if k == ord("s"):
    cv.imwrite("starry_night.png", img)
```

Loading Images [2]

- cv::imread()
  - Reads image from file
  - Decompresses to an image array

```
cv::Mat cv::imread(
            const string& filename,                 // Input filename
            int flags = cv::IMREAD_COLOR            // Flags set how to interpret file
);
```

  - If it fails, returns an empty cv::Mat
  - See Table 8.1 for flags
  - Note **BGR** standard (not RGB)

Saving Images [2]

- cv::imwrite()
  - Compresses image to specific format
  - Writes compressed image to file

```
bool cv::imwrite(
            const string& filename,                     // Input filename
            cv::InputArray image,                       // Image to write to file
            const vector<int>& params = vector<int>()   // (Optional) formats
);
```

  - The extension in filename is used to determine format
  - Supported formats: .jpg or .jpeg; .jp2, .tif or .tiff, .png, .bmp, .ppm, .pgm
  - Returns true if successful.

Saving images- cont.

- Second argument is the image to be stored.
- Only 8-bit, single- or three-channel images written by imwrite()
- Codecs (compression and decompression libraries):
  - OpenCV codecs
  - External libraries

Creating & destroying a window

```
int cv::namedWindow(
  const string&  name,                    // Handle used to identify window
  int            flags = 0                // Used to tell window to autosize
);
```

- name
  - A string shown on top of window
  - Also used as the handle to the window
- flags
  - 0 (default): the user resizes the window
  - *cv::WINDOW_AUTOSIZE*: the window is automatically resized to content; user cannot resize window

```
int cv::destroyWindow(
  const string&  name,                    // Handle used to identify window
);
```

Drawing an image

```
void cv::imshow(
  const string&  name,                    // Handle used to identify window
  cv::InputArray image                    // Image to display in window
);
```

- The window has its own copy of the image
- If image is changed, the windows contents will NOT be automatically updated
- Use *cv::waitKey()* for a pause to allow OpenCV windows to be updated
  - Events are fetched and handled when this function is called

Listen to keyboard input

```
int cv::waitKey(
  int delay = 0                    // Milliseconds until giving up (0='never')
);
```

- Waits for *delay* milliseconds
- If *delay=0*, waits indefinitely
- Returns (the ASCII code of) the key pressed
- Returns -1, otherwise

Other window-related functions

```
void cv::moveWindow( const char* name, int x, int y );
void cv::destroyAllWindows( void );
int  cv::startWindowThread( void );
```

- *moveWindow():* the upper-left corner of the window is moved to (x,y)
- *destroyAllWindows():* close all windows
- *startWindowThread():*
  - Starts a thread to take care of updating windows automatically
  - Otherwise, use *waitKey()* to allow time for this

Example- Videos

Source: OpenCV: Getting Started with Videos

```
import numpy as np
import cv2 as cv

cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    # Our operations on the frame come here
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv.imshow('frame', gray)
    if cv.waitKey(1) == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv.destroyAllWindows()
```

Reading Video

- cv::VideoCapture
  1. Reading frames from a video file

```
cv::VideoCapture::VideoCapture (
        const string& filename                    // Input filename
);
```

- If opened successfully, cv::VideoCapture::isOpened() will return true

  2. Reading frames from a camera

```
cv::VideoCapture::VideoCapture (
        int device                                // Video capture device id
);
```

- Identification number – zero when only one camera
- Can specify domain (see [2] Chap 8, Table 8-3, pp. 191)

Image as an array in NumPy

- In OpenCV C++, the data type for working with images is cv::Mat.
- In OpenCV/ Python, we use NumPy library for working with arrays efficiently.

```
import numpy as np
import cv2 as cv

# Create a black image
img = np.zeros((256,512,3), np.uint8)

# Create a random image
img = np.random.randint(0, 255,
      size=(256,512,3),dtype=np.uint8)
```

```
# (nrows, ncolumns, nchannels if color)
>>> print( img.shape )
(256, 512, 3)

# 256 * 512 * 3
>>> print( img.size )
393216

# image datatype
>>> print( img.dtype )
uint8
```

Source: OpenCV: Basic Operations on Images

Accessing and Modifying Pixels

```
>>> px = img[100,100]
>>> print( px )
[157 166 200]

# accessing only blue pixel
>>> blue = img[100,100,0]
>>> print( blue )
157

# modify a pixel value
>>> img[100,100] = [255,255,255]
>>> print( img[100,100] )
[255 255 255]
```

Better way:

```
# accessing RED value
>>> img.item(10,10,2)
59

# modifying RED value
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

Color channels

- Using OpenCV or NumPy

```
# Split into 3 channels
>>> b,g,r = cv.split(img)

# Merge 3 channels into one image
>>> img = cv.merge((b,g,r))

# Use NumPy indexing
# get blue channel
>>> b = img[:,:,0]

# Set all red values to zero
>>> img[:,:,2] = 0
```

Example: max red value

```
img = cv.imread("sample.png")
rows, cols, ncolor = img.shape

red = 2 # index of red values in (b,g,r)
max = 0
for i in range(rows):
    for j in range(cols):
        k = img.item(i, j, red)
        if k > max:
            max = k

print("Maximum red value in image is ", max)
```

Example: copy and paste Region of Interest (ROI)

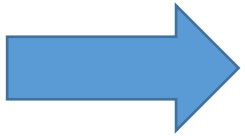
Source: OpenCV: Basic Operations on Images

>>> ball = img[280:340, 330:390]

>>> img[273:333, 100:160] = ball

Example- Cropping

```
cropped = img[411:1560, 1700:3000]
```



Example- Padding

Zero Padding

Padding in OpenCV

```
void cv::copyMakeBorder(
  cv::InputArray    src,                    // Input image
  cv::OutputArray   dst,                    // Result image
  int               top,                    // Top side padding (pixels)
  int               bottom,                 // Bottom side padding (pixels)
  int               left,                   // Left side padding (pixels)
  int               right,                  // Right side padding (pixels)
  int               borderType,             // Pixel extrapolation method
  const cv::Scalar& value = cv::Scalar()    // Used for constant borders
);
```

Table 10-1. borderType options available to cv::copyMakeBorder(), as well as many other functions that need to implicitly create boundary conditions

Border type	Effect
`cv::BORDER_CONSTANT`	Extend pixels by using a supplied (constant) value
`cv::BORDER_WRAP`	Extend pixels by replicating from opposite side
`cv::BORDER_REPLICATE`	Extend pixels by copying edge pixel
`cv::BORDER_REFLECT`	Extend pixels by reflection
`cv::BORDER_REFLECT_101`	Extend pixels by reflection, edge pixel is not "doubled"
`cv::BORDER_DEFAULT`	Alias for `cv::BORDER_REFLECT_101`

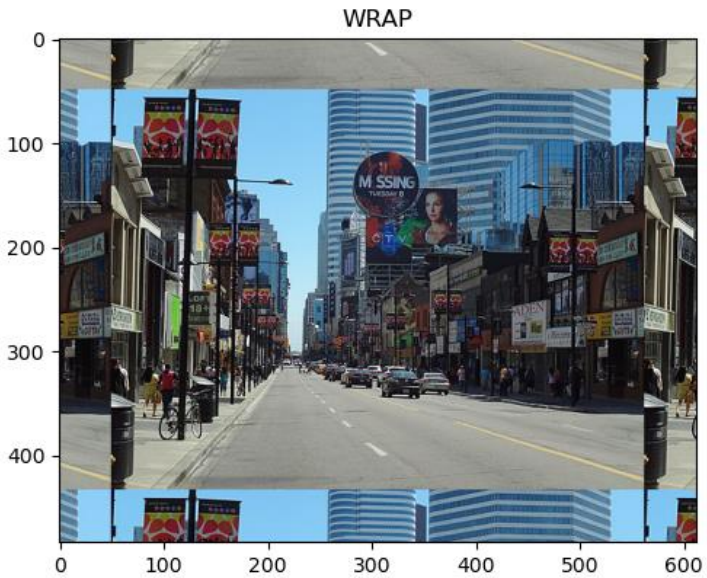
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
BLUE = [255,0,0]
bsz  = 50

img1 = cv.imread('Yonge_Street.jpg')
replicate =    cv.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv.BORDER_REPLICATE)
reflect =      cv.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv.BORDER_REFLECT)
reflect101 =   cv.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv.BORDER_REFLECT_101)
wrap =         cv.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv.BORDER_WRAP)
constant=      cv.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv.BORDER_CONSTANT,value=BLUE)

plt.subplot(231), plt.imshow(cv.cvtColor(img1,cv.COLOR_BGR2RGB)), plt.title('ORIGINAL')
plt.subplot(232), plt.imshow(cv.cvtColor(replicate,cv.COLOR_BGR2RGB)), plt.title('REPLICATE')
plt.subplot(233), plt.imshow(cv.cvtColor(reflect,cv.COLOR_BGR2RGB)), plt.title('REFLECT')
plt.subplot(234), plt.imshow(cv.cvtColor(reflect101,cv.COLOR_BGR2RGB)), plt.title('REFLECT_101')
plt.subplot(235), plt.imshow(cv.cvtColor(wrap,cv.COLOR_BGR2RGB)), plt.title('WRAP')
plt.subplot(236), plt.imshow(cv.cvtColor(constant,cv.COLOR_BGR2RGB)), plt.title('CONSTANT')
plt.show()
```

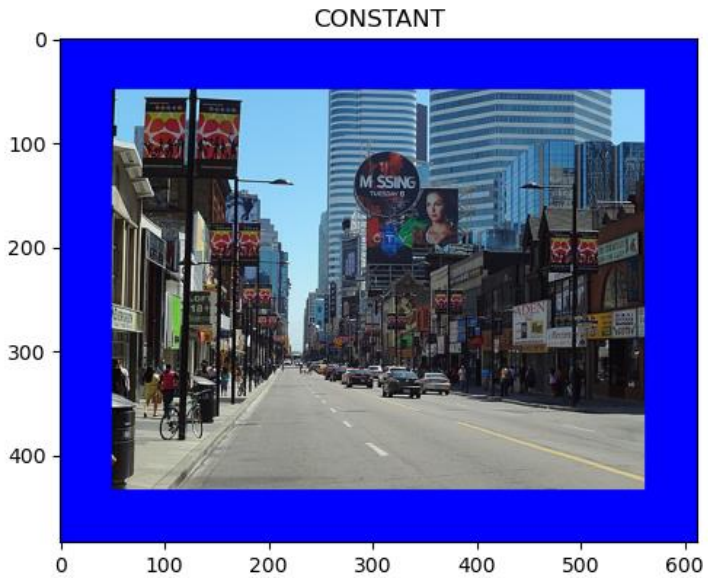

ORIGINAL


REPLICATE


REFLECT


REFLECT_101


WRAP


CONSTANT

Summary

- 3D properties of the object are transformed into 2D image features by Perspective Projection on a photosensitive image plane.
- Using formulas from optics, we can calculate the distance of objects in focus, and the size of the circle of confusion for objects out of focus.
- Sampling, quantization, and compression are applied to images in a digital camera, resulting in smaller file sizes, but also some loss of information.
- OpenCV provides functions for reading, writing, and working with images and videos.

References

[1] Computer Vision: Algorithms and Applications, R. Szeliski (http://szeliski.org/Book)

[2] Learning OpenCV 3, A. Kaehler & G. Bradski

- Available online through Safari Books, Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51532449200003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US

[3] Practical introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe

- Available through Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51428109500003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US