NUS-ISS
Vision Systems

Module 3 - Foundations of computer vision system (2) - Local feature and representation, part 1

Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

Learning objectives

- Read, display and write image
- Convert image’s colour format
- Expand image’s border
- Draw objects on image

Read image

yoshida1.jpg

- In opencv, use `imread()` to load an image

```
> import cv2

> imgfile = 'yoshida1.jpg'
> img     = cv2.imread(imgfile)
```

- Or simply just

```
> img     = cv2.imread('yoshida1.jpg')
```

- Supported formats: jpg, bmp, png, tif, tiff, pbm, ppm, hdr, pic
  - Check the below link for more detail: https://docs.opencv.org/3.4.2/d4/da8/group__imgcodecs.html

Check basic info

- Many times it is beneficial / required to check basic information about the image loaded
- In Spyder, that can be done in Variable explorer

```
                          i       j
img  |  uint8  |  (700,  477,  3)
                   height  width  channel
                          y       x
```

- To access through code, do

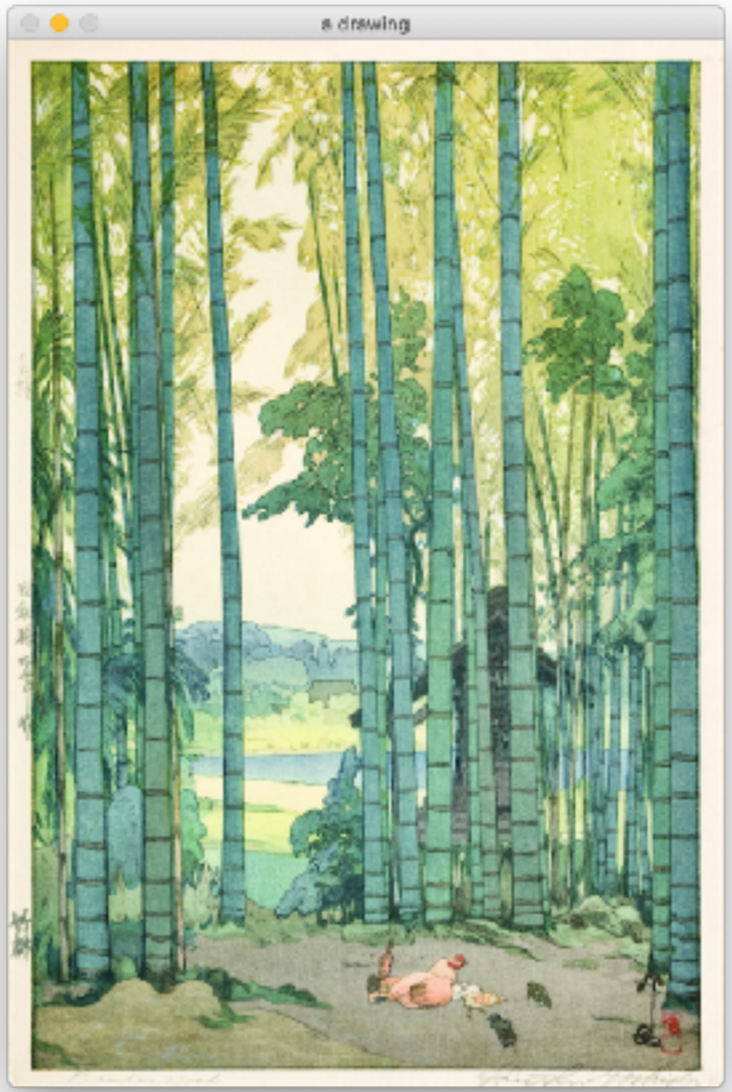
```
> img.shape
: (700, 477, 3)

> img.dtype
: dtype('uint8')

> print('img height: %d' % (img.shape[0]))
: img height: 700
```

yoshida1.jpg

Display image
Through opencv

- To display an image in opencv for `img`, we do

```
> cv2.imshow('a drawing',img)
> cv2.waitKey(0)
> cv2.destroyAllWindows()
```



- On some platforms, it needs the below 4 lines to prevent freezing of the window:

```
> cv2.waitKey(1)
> cv2.waitKey(1)
> cv2.waitKey(1)
> cv2.waitKey(1)
```

Display image
Through opencv

- `imshow()` does the display of image

```
cv2.imshow('a drawing',img)
```

name of the window — name of the variable

- `imshow()` should be followed by function `waitKey()`, which specifies how long the image should be specified in milliseconds

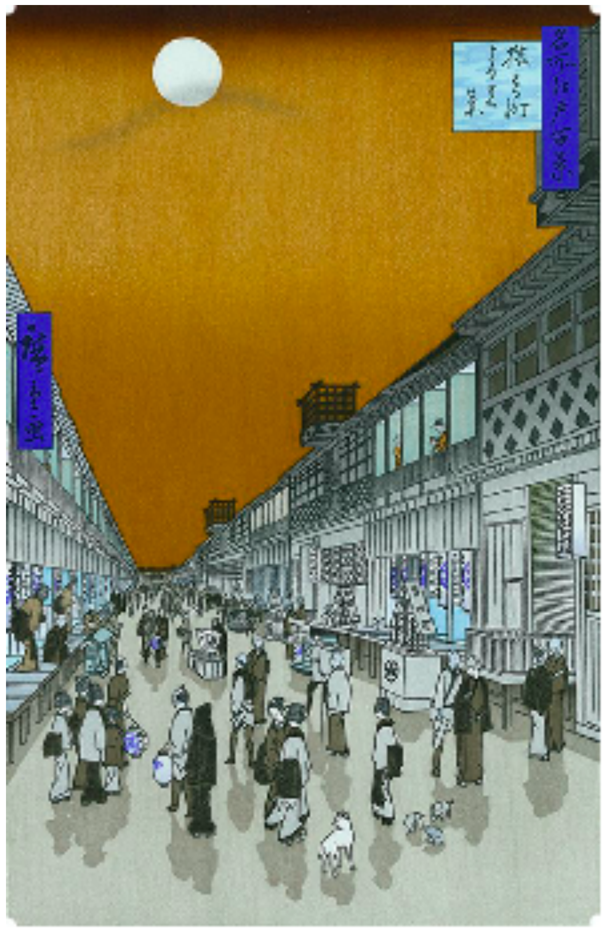
```
cv2.waitKey(0)
```

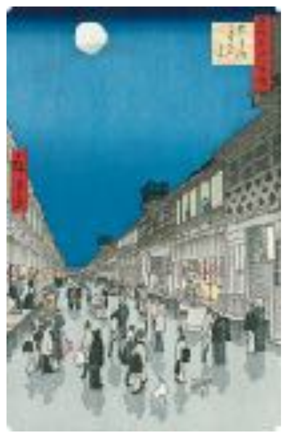
value zero stands for waiting infinitely

- `destroyAllWindows()` shuts down all windows opened through opencv

```
> cv2.imshow('a drawing',img)
> cv2.waitKey(0)
> cv2.destroyAllWindows()
```

Colour format
BGR


BGR format displayed by function that expects RGB format



- The output of `imread()` is a numpy array

- This implies that we can manipulate the output variable using numpy's function/method

- For a colour image `img`, `imread()` gives a 3D numpy array, in BGR format

`img[:,:,0]`	→	**B**lue channel
`img[:,:,1]`	→	**G**reen channel
`img[:,:,2]`	→	**R**ed channel

- However, many other libraries process image array only in RGB format

Colour format conversion

- Use `cvtColor()` to convert colour format

name of the variable

```
> imgc = cv2.cvtColor(img,
                      cv2.COLOR_BGR2GRAY)
```

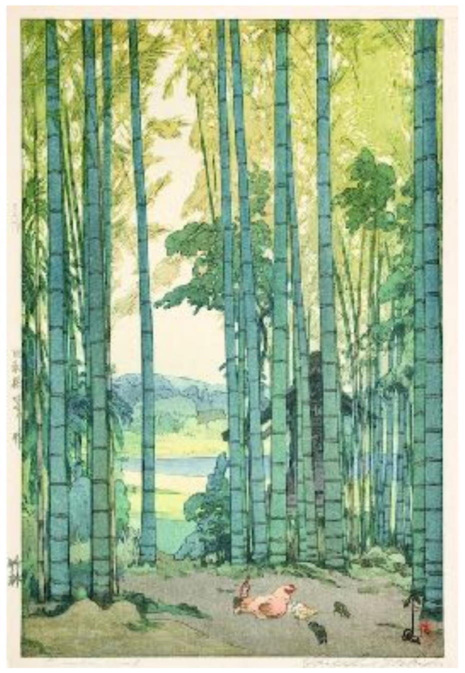
colour conversion code

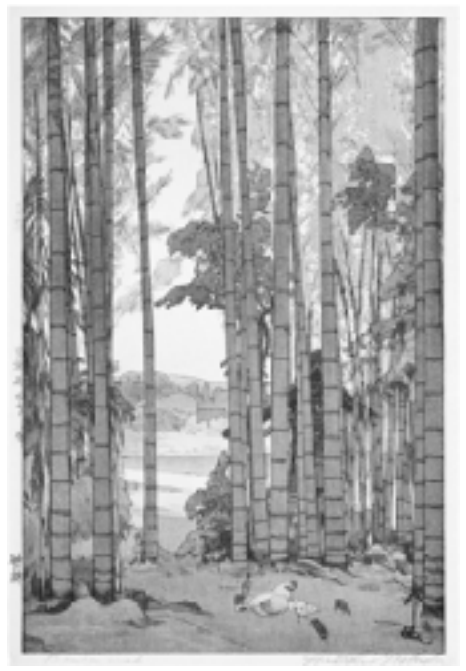
- Other conversion codes:

```
cv2.COLOR_BGR2RGB
cv2.COLOR_RGB2BGR
cv2.COLOR_BGR2GRAY
cv2.COLOR_BGR2YUV
cv2.COLOR_YUV2BGR
cv2.COLOR_BGR2Luv
cv2.COLOR_Luv2BGR
…
```

- Check the below link for more detail

  https://docs.opencv.org/3.4.2/d7/d1b/group__imgproc__misc.html





Colour format conversion

- When we convert a colour image to gray, the output is no longer a 3D numpy array

```
> imgc.shape
: (700, 477)
```

- To get back a 3D BGR array, we can do

```
> imgd = cv2.cvtColor(imgc,
                               cv2.COLOR_GRAY2BGR)
```

- Or we can do

for blue channel

for red channel

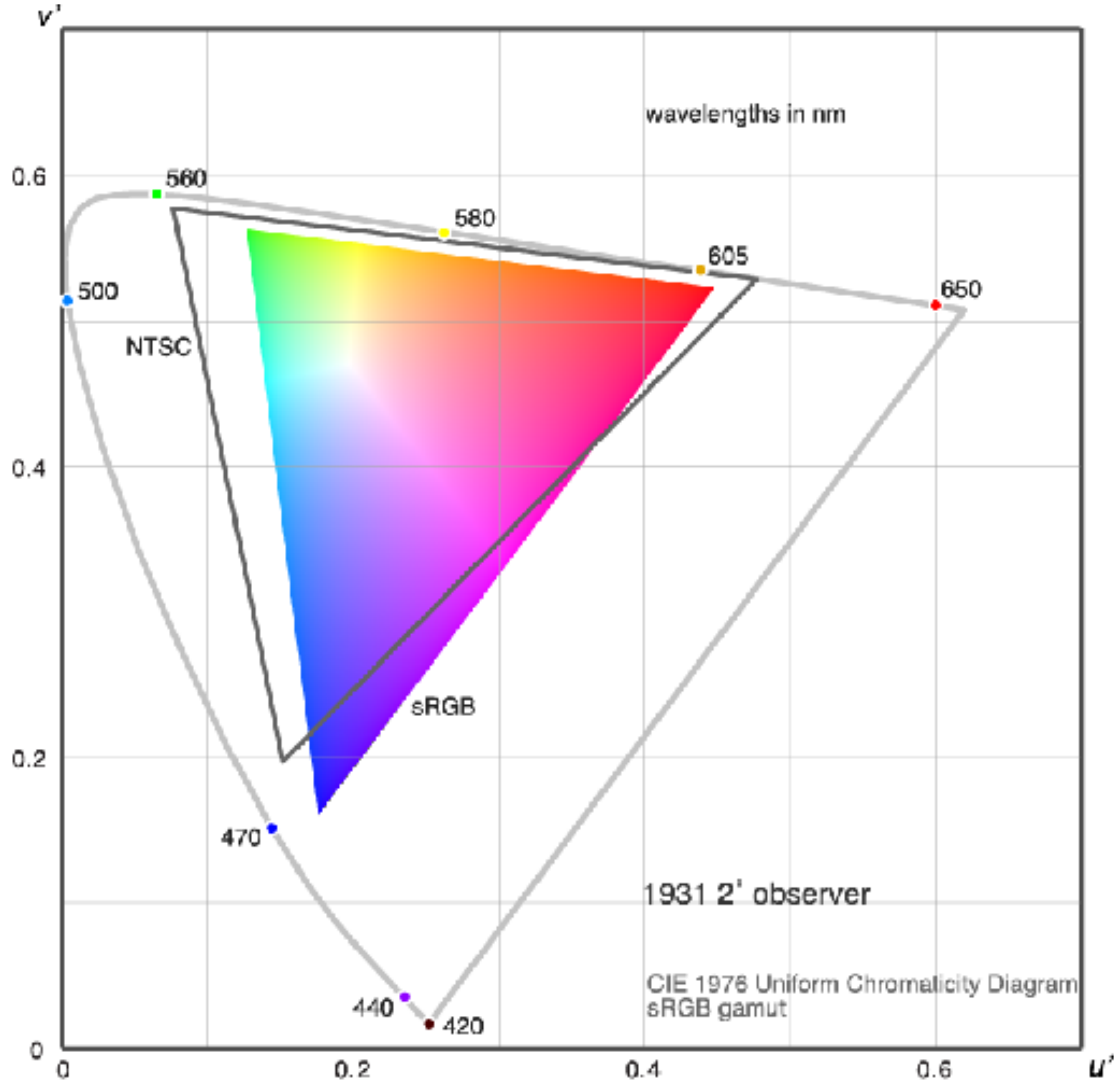
```
> imge = cv2.merge((imgc,imgc,imgc))
```

for green channel

A gray is simply a colour that has the same value in R,G and B

Why often do we need to convert colour image into gray scale image?

The theory of colour



Display image, again

Through matplotlib

- We can use matplotlib to display an image `img`, to do that, we write

```
> import matplotlib.pyplot as plt
> plt.axis('off')
> plt.imshow(cv2.cvtColor(img,
                          cv2.COLOR_BGR2RGB))
> plt.show()
```

- In the above codes, we first turn off axis in the plot

- Then use `plt.imshow()` to display image, but that function expects image in RGB format

- Thus need to use `cv2.cvtColor()` to convert `img` into RGB format

- Finally, use `plt.show()` to get the plot displayed

axis on
axis off
0
100
200
300
400
500
600
0
100
200
300
400

Display image, again

Through matplotlib

- To display a grayscale image `imgc`, we need extra settings on `plt.imshow()`

```
> plt.axis('off')
> plt.imshow(imgc,
             cmap='gray',
             vmin=0,
             vmax=255)
> plt.show()
```

- `cmap='gray'` : inform the function to use grayscale colour map
- `vmin=0` : map value 0 to black
- `vmax=255` : map value 255 to white

Expand image border

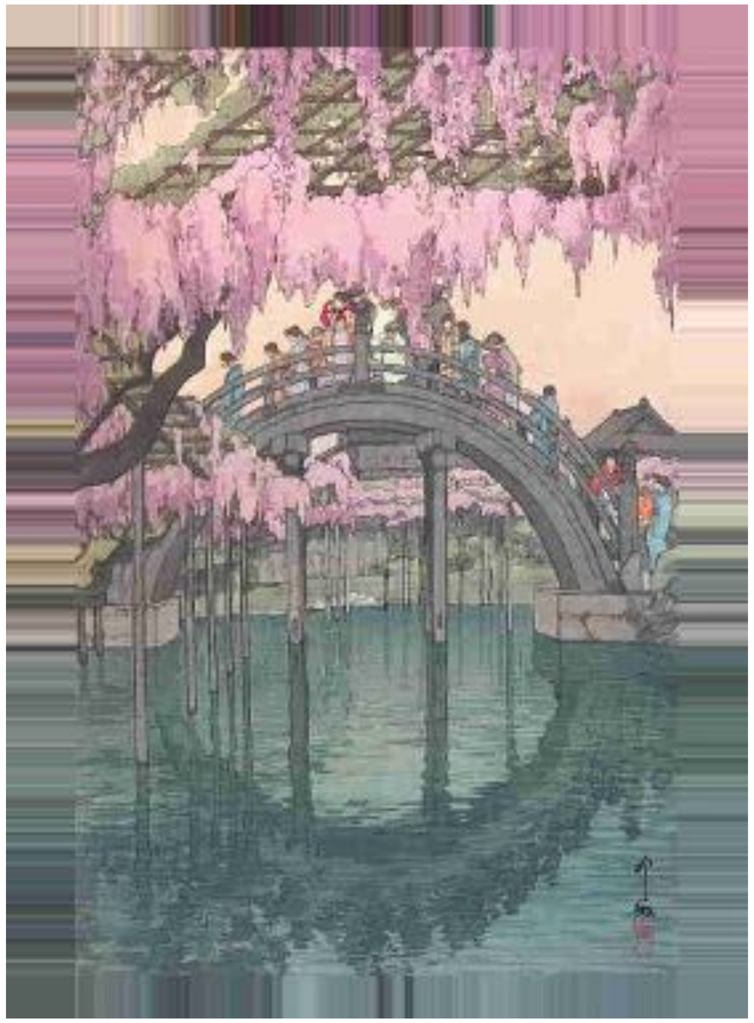
- Often in image processing it is necessary to expand image border to avoid undesired effect

- Use `cv2.copyMakeBorder()` to expand border

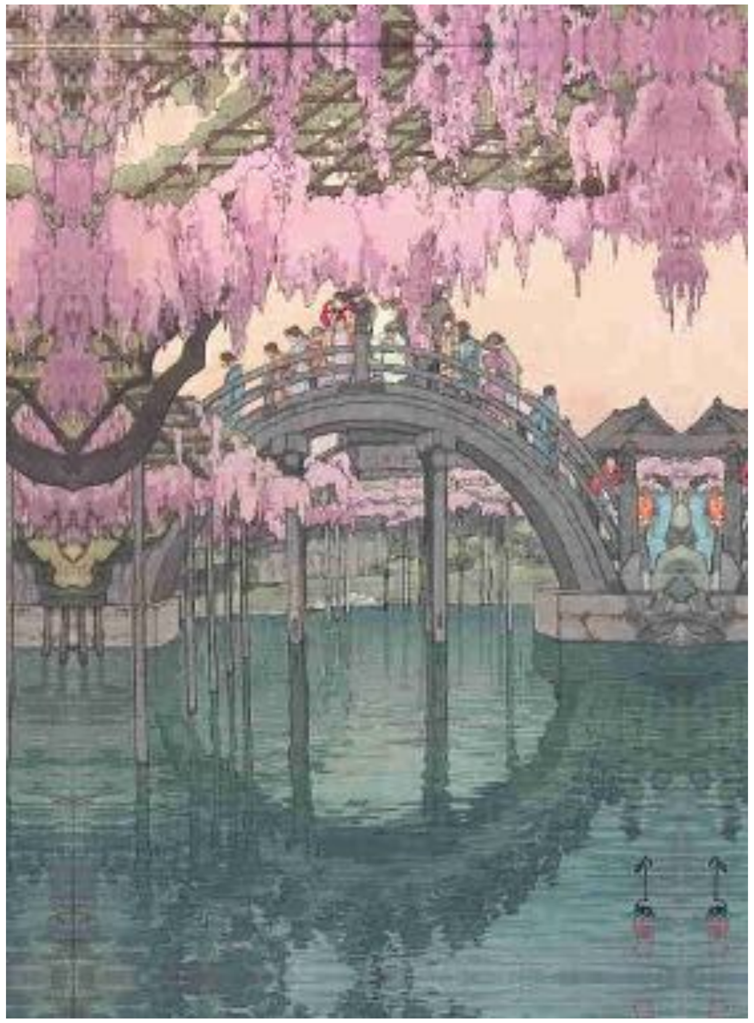
```
> imgf = cv2.imread('yoshida2.jpg')
> bdrx = cv2.copyMakeBorder(imgf,
                                30,    top
                                30,    bottom
                                50,    left
                     right      50,
                borderType      cv2.BORDER_REPLICATE)
```

Expand image border

`cv2.BORDER_REPLICATE`

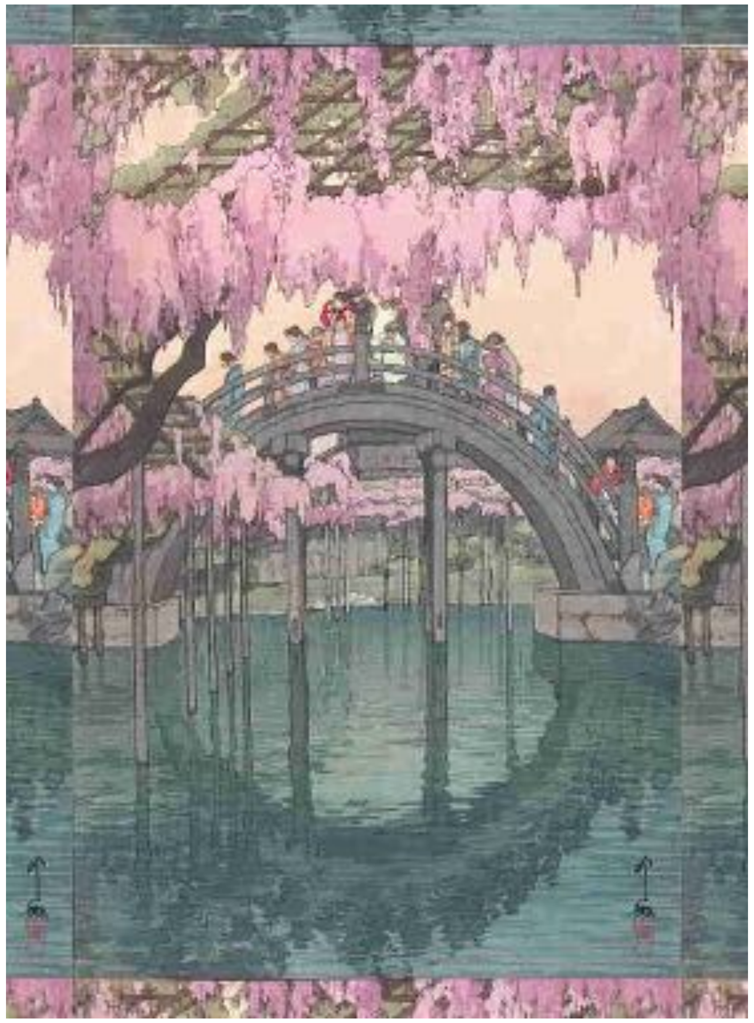


`cv2.BORDER_REFLECT`

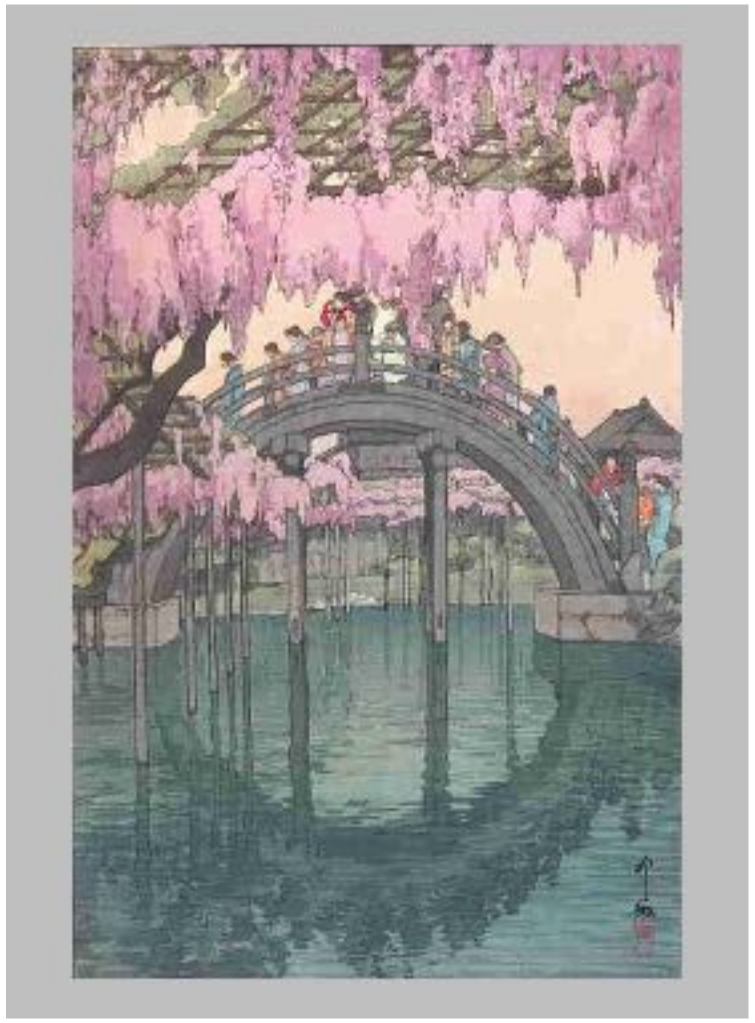


Expand image border

`cv2.BORDER_WRAP`



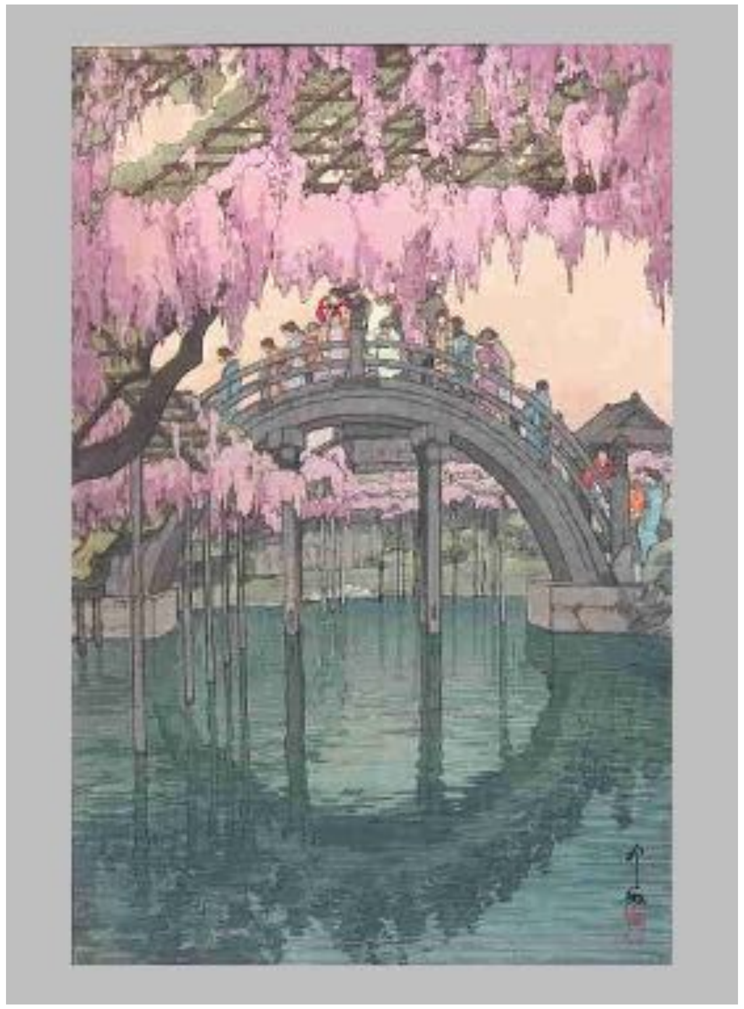
`cv2.BORDER_CONSTANT`



Expand image border

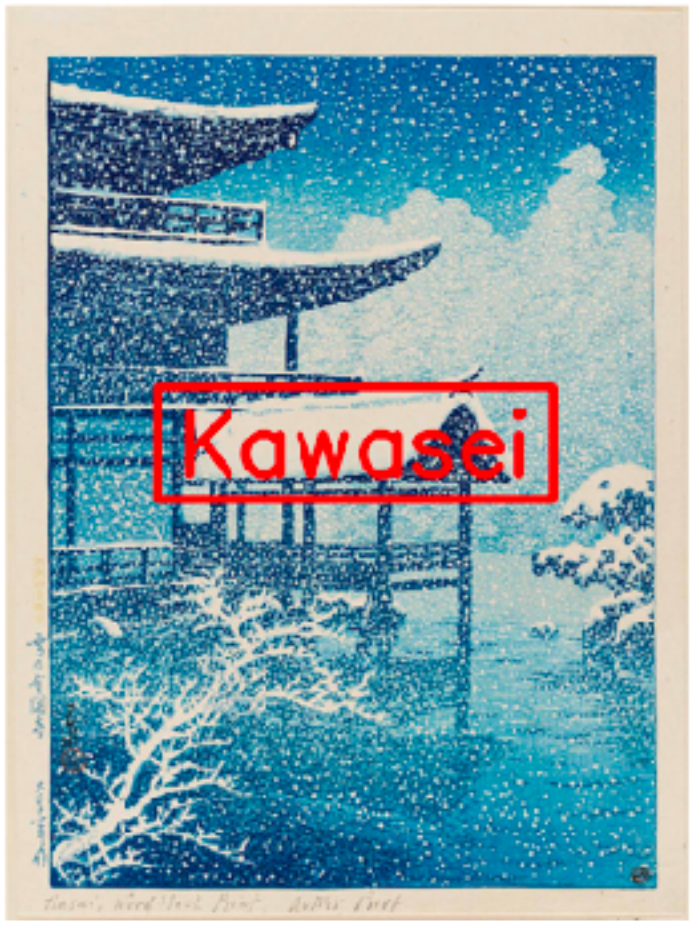
- When we select borderType `cv2.BORDER_CONSTANT`, we need to input one extra parameter:

- The colour we want on the border



```
> bdrx = cv2.copyMakeBorder(imgf,
                                         30,     top
                                         30,     bottom
                                         50,     left
                            right        50,
                            borderType   cv2.BORDER_CONSTANT,
                            colour       value=(191,191,191))
                                                B   G   R
```

Draw objects



- We can draw objects on image using API provided by opencv
- Generally in computer vision we use these API to make annotations on image
- Some of the functions available:

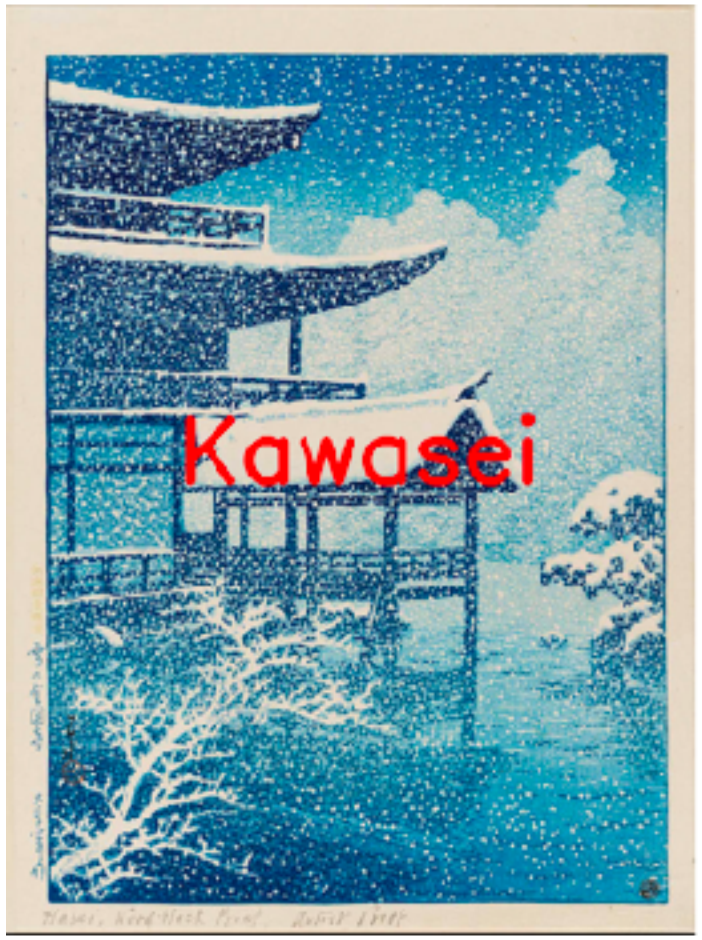
```
cv2.line()
cv2.circle()
cv2.rectangle()
cv2.ellipse()
cv2.putText()
…
```

- After the drawing, we display the product either using `cv2.imshow()` or `plt.imshow()`

Draw text

- To write words on image, we use `cv2.putText()`

```
> imgg = cv2.imread('kawasei1.jpg')

> cv2.putText(imgg,
                                        'Kawasei',                text to display
       (x, y) position at bottom-left   (150,410),
                            font type   cv2.FONT_HERSHEY_SIMPLEX,
                           font scale   2.5,
                               colour   (0,0,255),
                       font thickness   5,
                            line type   cv2.LINE_AA)
```

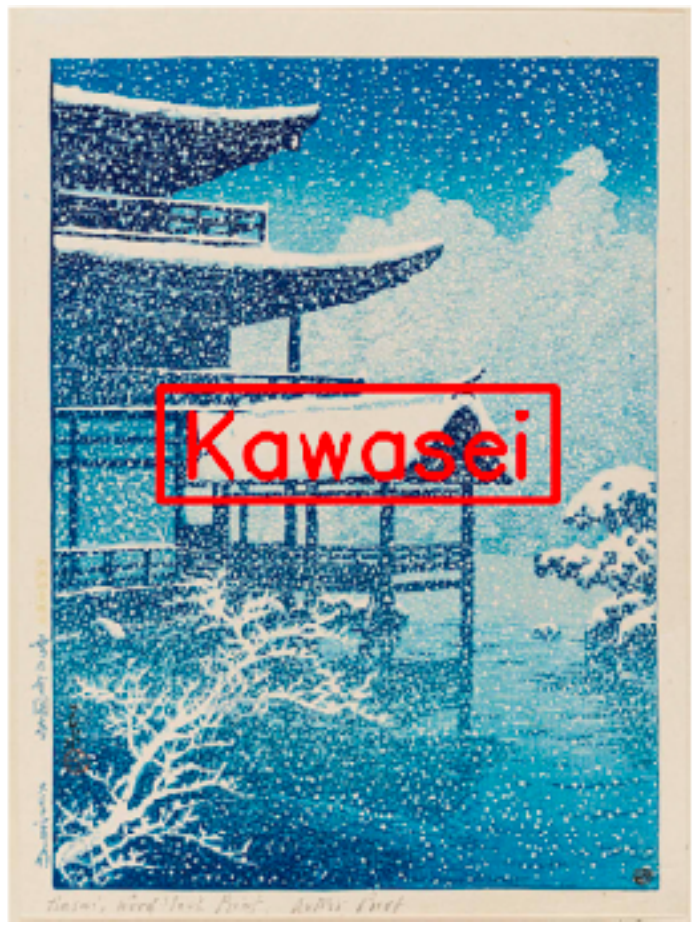


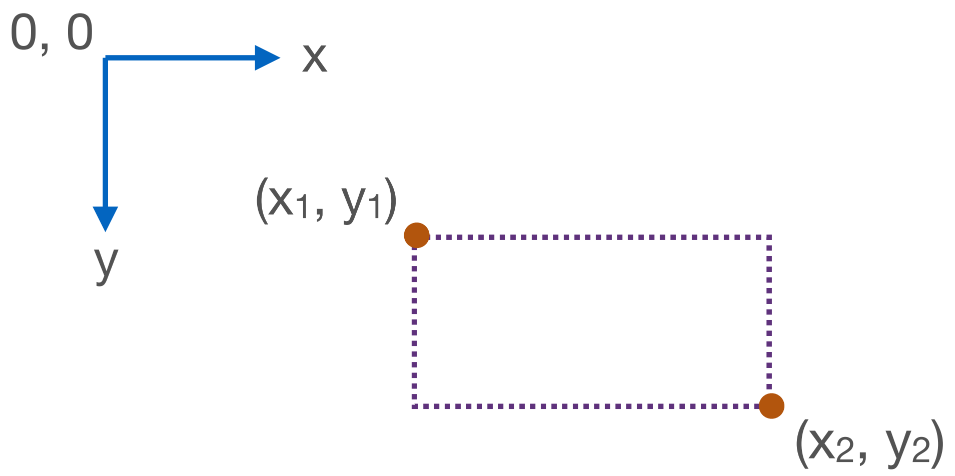
- `cv2.FONT_HERSHEY_SIMPLEX` stands for normal size sans-serif font
- `cv2.LINE_AA` gives anti-aliased line

Draw rectangle

- To draw rectangle on image, we use `cv2.rectangle()`

```
> cv2.rectangle(imgg,
                        (132,334),  (x1, y1)
             (x2, y2)   (480,432),
               colour   (0,0,255),
            thickness   5)
```





Saving image

- After all the hard work, it would be useless if we can't save the final output

- To save an image, we use `cv2.imwrite()`

```
> cv2.imwrite('kawas.jpg',  file name
    variable  imgg)
```

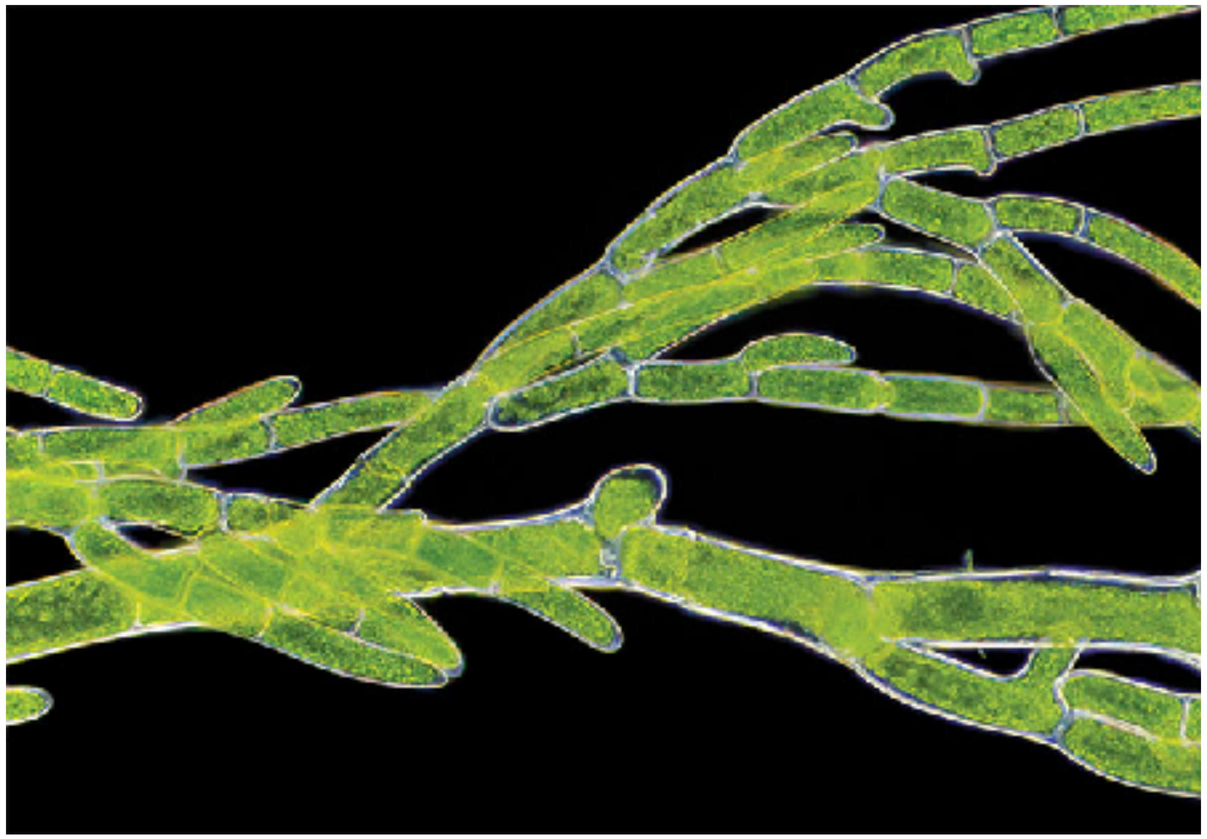
- Supported formats: jpg, png, tif, tiff, hdr, exr
  - Check the below link for more detail: https://docs.opencv.org/3.4.2/d4/da8/group__imgcodecs.html

Image understanding

Area and perimeter

- Exercise: Propose steps to determine the area and the perimeter of the algae in the image without using numpy and any opencv or image library (except for image reading and display) ?

- Write your answer in Word or PowerPoint and submit to luminus



Source: BUCCANEERSHIP/ISTOCK/GETTY IMAGES PLUS