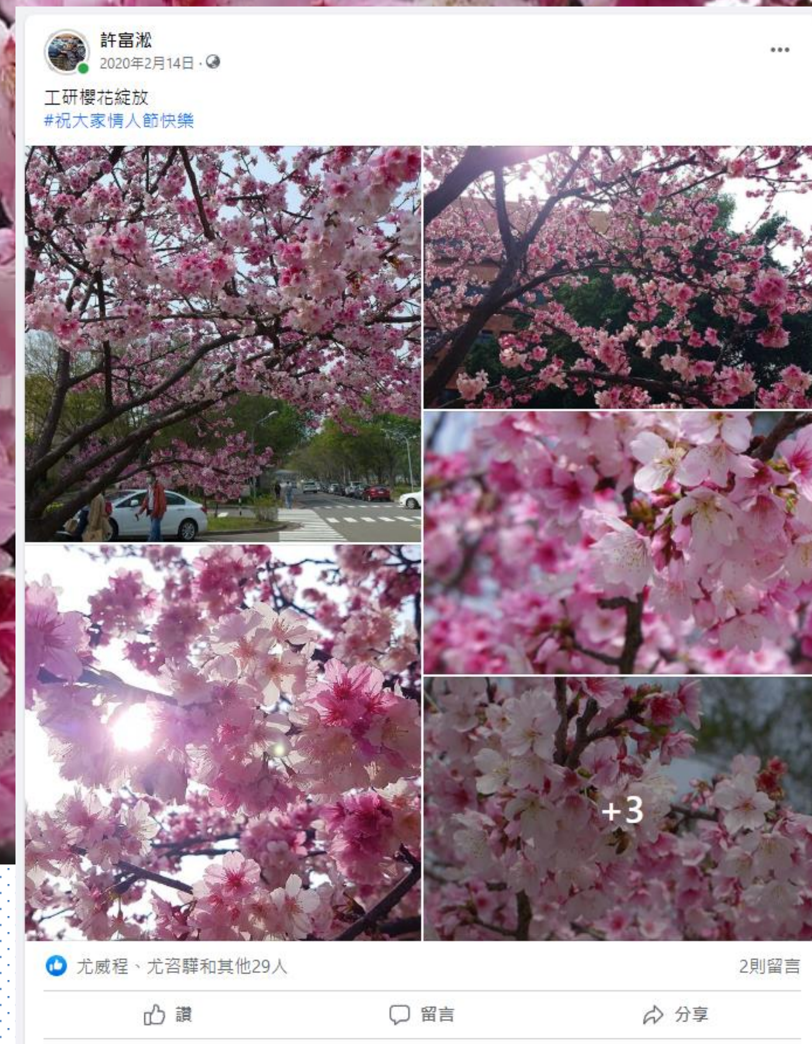
Motion Capture and Future Interaction Technology Research

Introduction to OPENCV

Speaker: Fu-Song Hsu

Outline

- Reading an image
- Extracting the RGB values of a pixel
- Extracting the Region of Interest (ROI)
- Resizing the Image
- Rotating the Image
- Drawing a Rectangle
- Displaying text


許富淞
2020年2月14日 ·
工研櫻花綻放
#祝大家情人節快樂
+3
尤威程、尤咨驊和其他29人
2則留言
讚
留言
分享

Reading an image

In [8]:

```python
# Importing the OpenCV library
import cv2
from IPython.display import display
from PIL import Image

# Reading the image using imread() function
image = cv2.imread('pics\\sakura.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Converting BGR to RGB
display(Image.fromarray(image))

# Extracting the height and width of an image
h, w = image.shape[:2]
# Displaying the height and width
print("Height = {},  Width = {}".format(h, w))
```

Import the OpenCV library (cv2)

Import the IPython to show images in Jupyter Notebook

Extracting the RGB values of a pixel

2. Extracting the RGB values of a pixel

In [9]:

```
# Extracting RGB values.
# Here we have randomly chosen a pixel
# by passing in 100, 100 for height and width.
(B, G, R) = image[100, 100]

# Displaying the pixel values
print("R = {}, G = {}, B = {}".format(R, G, B))

# We can also pass the channel to extract
# the value for a specific channel
B = image[100, 100, 0]
print("B = {}".format(B))
```

```
R = 126, G = 70, B = 175
B = 175
```

Resizing the Image

In [5]:
```
# resize() function takes 2 parameters,
# the image and the resolution
resize = cv2.resize(image, (800, 800))

display(Image.fromarray(resize))
```

In [6]:
```
# Calculating the ratio
ratio = 800 / w

# Creating a tuple containing width and height
dim = (800, int(h * ratio))

# Resizing the image
resize_aspect = cv2.resize(image, dim)

display(Image.fromarray(resize_aspect))
```

Drawing a circle

cv2.circle(img, center, radius, color, thickness)

- **img –** It is the image on which the circle has to be drawn.
- **center –** It is the coordinates of the center of the circle
- **radius –** It is the radius of the circle.
- **color –** It is the color of the circle in RGB.
- **thickness –** It is the thickness of the circle line.

Exercises: Sakura Snow

```
void cv::putText ( InputOutputArray  img,
                   const String &    text,
                   Point             org,
                   int               fontFace,
                   double            fontScale,
                   Scalar            color,
                   int               thickness = 1,
                   int               lineType = LINE_8,
                   bool              bottomLeftOrigin = false
                 )
```

Draws a text string.

The function putText renders the specified text string in the image. Symbols that cannot be rendered using the specified font are replaced by question marks. See getTextSize for a text rendering code example.

Parameters

img	Image.
text	Text string to be drawn.
org	Bottom-left corner of the text string in the image.
fontFace	Font type, see **cv::HersheyFonts**.
fontScale	Font scale factor that is multiplied by the font-specific base size.
color	Text color.
thickness	Thickness of the lines used to draw a text.
lineType	Line type. See the line for details.
bottomLeftOrigin	When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Displaying text

In [22]:
```
# Copying the original image
output = image.copy()

# Adding the text using putText() function
text = cv2.putText(output, 'OpenCV Demo', (100, 400),
cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)

display(Image.fromarray(output))
```

Changing the contrast and brightness

Brightness and contrast adjustments

- Two commonly used point processes are *multiplication* and *addition* with a constant:

$$g(x) = \alpha f(x) + \beta$$

- The parameters $\alpha > 0$ and β are often called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness* respectively.
- You can think of $f(x)$ as the source image pixels and $g(x)$ as the output image pixels. Then, more conveniently we can write the expression as:

$$g(i,j) = \alpha \cdot f(i,j) + \beta$$

  where i and j indicates that the pixel is located in the *i-th* row and *j-th* column.

```python
image = cv2.imread('pics\\sakura.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Converting BGR to RGB

if image is None:
    print('Could not open or find the image: ', args.input)
    exit(0)
new_image = np.zeros(image.shape, image.dtype)
alpha = 1.0 # Simple contrast control
beta = 0    # Simple brightness control
# Initialize values
print(' Basic Linear Transforms ')
print('-------------------------')
try:
    alpha = float(input('* Enter the alpha value [1.0-3.0]: '))
    beta = int(input('* Enter the beta value [0-100]: '))
    print()
except ValueError:
    print('Error, not a number')
# Do the operation new_image(i,j) = alpha*image(i,j) + beta
# Instead of these 'for' loops we could have used simply:
# new_image = cv.convertScaleAbs(image, alpha=alpha, beta=beta)
# but we wanted to show you how to access the pixels :)
for y in tqdm(range(image.shape[0])):
    for x in range(image.shape[1]):
        for c in range(image.shape[2]):
            new_image[y,x,c] = np.clip(alpha*image[y,x,c] + beta, 0, 255)
    time.sleep(0.01)
#cv2.imshow('Original Image', image)
#cv2.imshow('New Image', new_image)
display(Image.fromarray(image))
display(Image.fromarray(new_image))
```

we load an image using cv::imread and save it in a Matrix object

we need a new Mat object to store output data

- Initial pixel values equal to zero
- Same size and type as the original image

perform the operation $g(i,j)=\alpha\cdot f(i,j)+\beta$
we will access to each pixel in image.

Blending two images

From our previous tutorial, we know already a bit of *Pixel operators*. An interesting dyadic (two-input) operator is the *linear blend operator*:

$$g(x) = (1 - \alpha) f_0(x) + \alpha f_1(x)$$

By varying α from $0 \rightarrow 1$ this operator can be used to perform a temporal *cross-dissolve* between two images or videos, as seen in slide shows and film productions (cool, eh?)

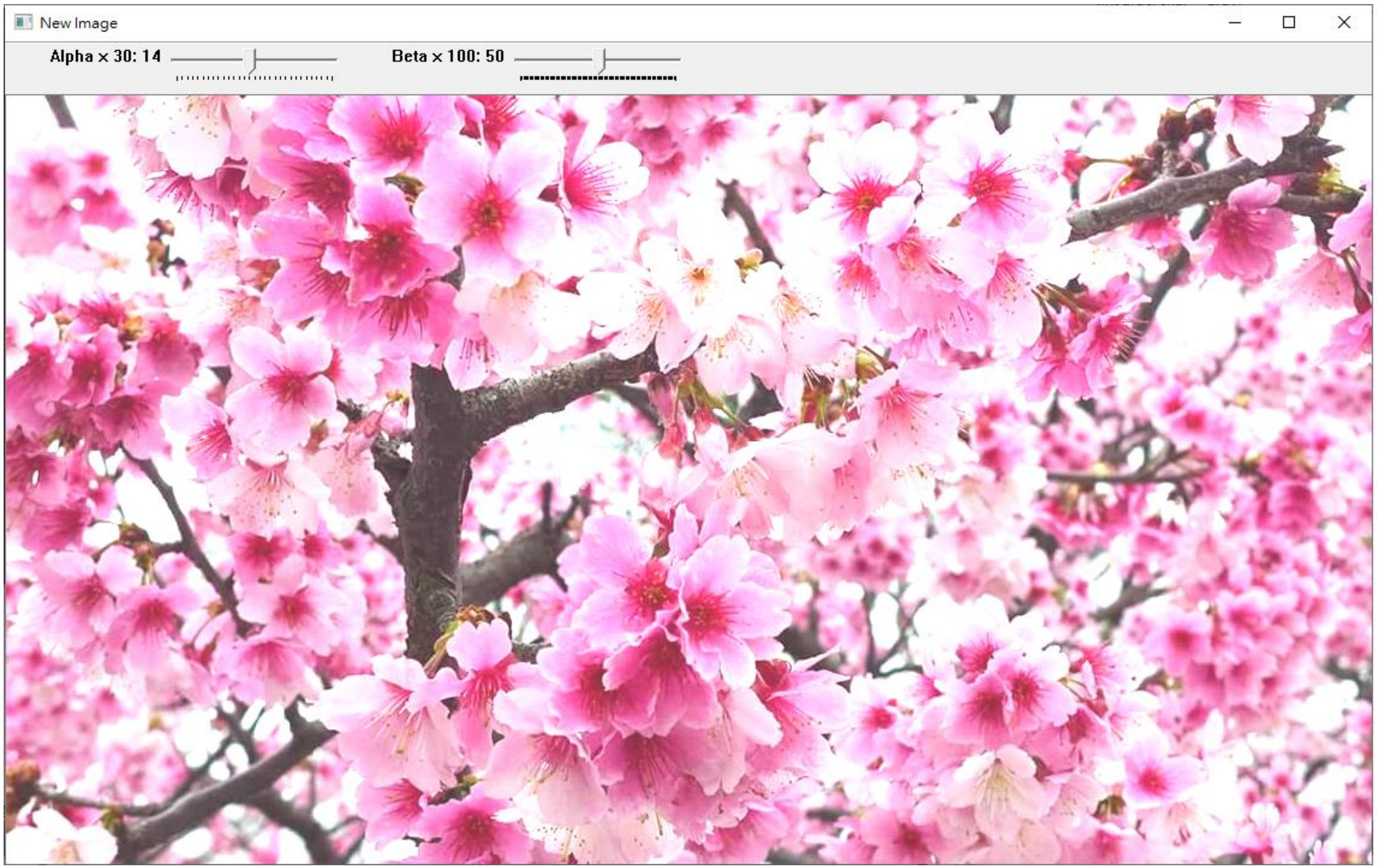
```python
import cv2
import argparse
alpha_slider_max = 100
title_window = 'Linear Blend'
def on_trackbar(val):
    alpha = val / alpha_slider_max
    beta = ( 1.0 - alpha )
    dst = cv2.addWeighted(src1, alpha, src2, beta, 0.0)
    cv2.imshow(title_window, dst)
#parser = argparse.ArgumentParser(description='Code for Adding a Trackbar to our applications tutorial.')
#parser.add_argument('--input1', help='Path to the first input image.', default='LinuxLogo.jpg')
#parser.add_argument('--input2', help='Path to the second input image.', default='WindowsLogo.jpg')
#args = parser.parse_args()
src1 = cv2.imread('pics\\LinuxLogo.jpg')
src2 = cv2.imread('pics\\WindowsLogo.jpg')
if src1 is None:
    print('Could not open or find the image: ', args.input1)
    exit(0)
if src2 is None:
    print('Could not open or find the image: ', args.input2)
    exit(0)
cv2.namedWindow(title_window)
trackbar_name = 'Alpha x %d' % alpha_slider_max
cv2.createTrackbar(trackbar_name, title_window , 0, alpha_slider_max, on_trackbar)
# Show some stuff
on_trackbar(0)
# Wait until user press some key
cv2.waitKey()
```

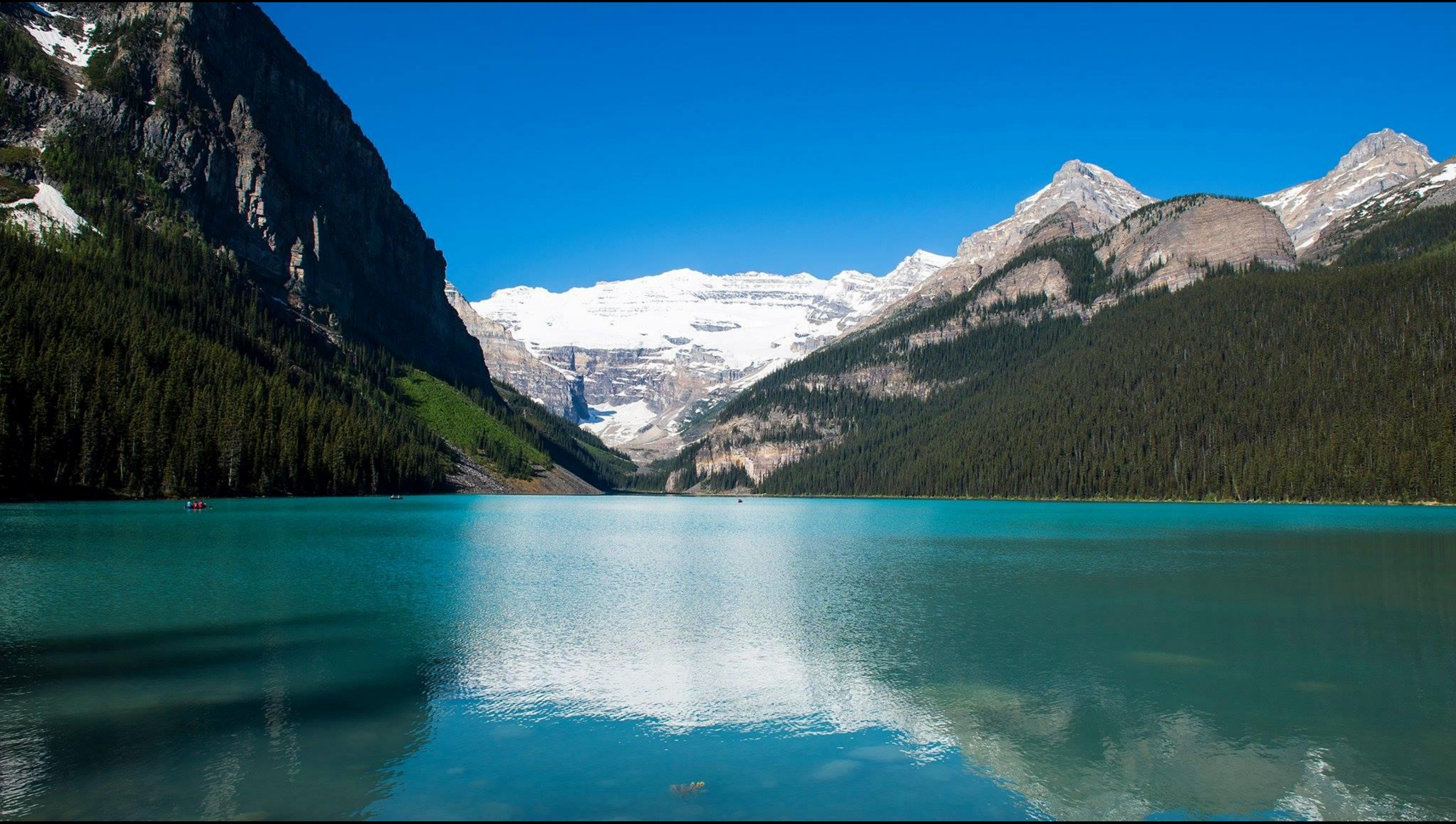
Now we need to generate the g(x) image.
For this, the function **addWeighted()** comes quite handy:

We used the following images: LinuxLogo.jpg and WindowsLogo.jpg

Now we can create the Trackbar

CW: Adding a Trackbar to our applications





Q&A