Department of Electronics and Communication

PES University

Digital Image Processing Project

Chroma Key Compositing using Python

Deepanjali Pandit	Nandana Harikumar	Nidhi Prakash
PES1UG19EC089	PES1UG19EC176	PES1UG19EC183
Section: B	Section: C	Section: C

(**Team ID: 13)**

Introduction

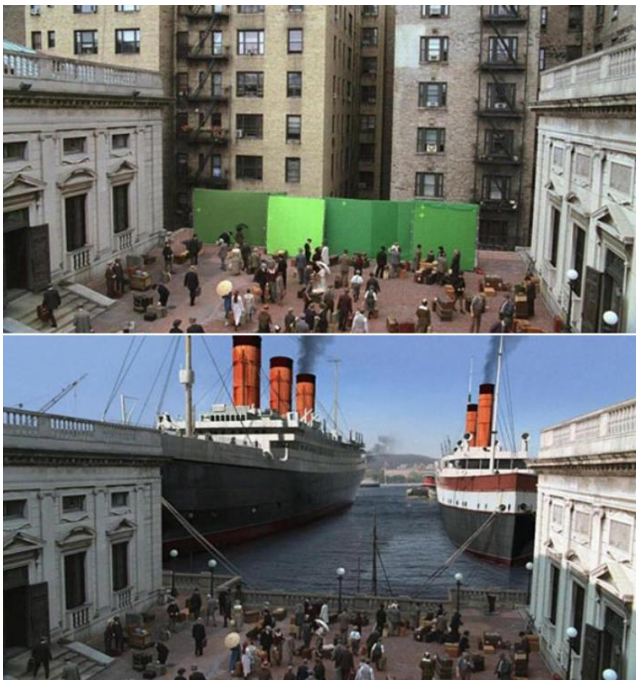
Chroma key compositing, also known simply as chroma keying, is a technique used in film, video and still photography to replace a portion of an image with a new image.

This technique is popular in many industries like newscasting, video games and motion picture as it enables using footage from other locations, artificial sets, etc. to be placed in the background to make it seem as though the subject of the video (reporter/actor) is in a different location. It involves filming actors and objects in front of a flat screen of a single colour. This screen is usually green or blue, and hence chroma keying is also often referred to as a 'green screen' or 'blue screen' effect.

An interesting application of this is using close-up footage of a miniature set placed in the background, making it seem as though the actors are in a full sized version or the model! In fact, it needn't even be a real model, a completely virtually created environment could be used as well, like we see in many sci-fi movies.

The chroma keying technique is commonly used in video production and post-production.

A few real life examples of chroma keying being used on film/TV sets are:



Source: https://digitalsynopsis.com/design/movies-before-after-green-screen-cgi/

Theoretical Details and Algorithm

Chroma key compositing is a visual-effects and post-production technique for that involves compositing, i.e, layering two images or video streams together based on colour hues (chroma range).

Such processes is made possible by making the foreground of an image transparent, where its foreground is composed of green and blue backgrounds that facilitates the process to isolate the interesting parts of an image from their irrelevant background, preserving only the pixels of

interests and then allowing separately filmed background footage or a static image to be inserted into the scene.

In this project, we primarily employ a python toolkit called *skimage*, to perform imaging operations. Operations involved are:

- Remove a green background from an image while isolating an object
- Blend two images into a single image after applying basic rotation, translation and scale operations
- Apply gaussian blur filter on background image

Code:

(Followed by output for each step)

Original images used as an example:

as background


as greenscreen image

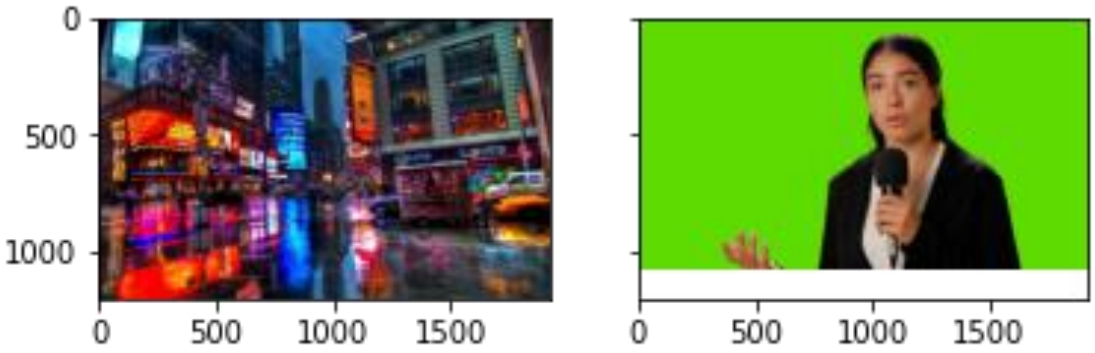
i) Import Python libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import skimage
from skimage import img_as_ubyte
from skimage import io
from skimage import filters
from skimage.morphology import disk
```

- skimage : scikit-image is a collection of algorithms for image processing and computer vision
- matplotlib : Matplotlib is a comprehensive library for creating static, animated and interactive visualizations in Python

ii) User Defined Functions:

1. **draw()** - To import and plot the background image and the green screen image

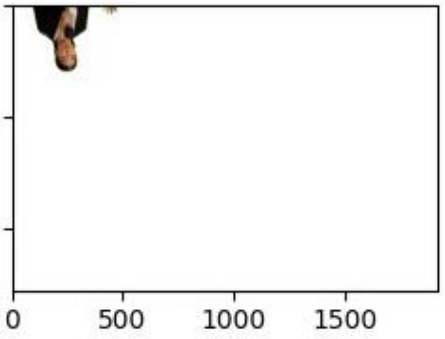



```
[35] def draw(bg, green):
         fig, ax = plt.subplots(1, 2, sharex=True, sharey=True)
         # fig.suptitle("close this window to continue")
         ax[1].imshow(green)
         ax[0].imshow(bg)
         plt.show()
```

2. **rotate()** - To rotate the green screen image by either 0°, 90°, 180°, -90° or -180°

```
def rotate(bg,green):
    r_list = [0,90,180,-90,-180]
    r = 1
    while(r not in r_list):
        r = int(input("enter the amount of rotation (0,90,180,-90,-180) ?...  "))
        if(r in r_list):
            green = skimage.transform.rotate(green, r, resize=True, mode='edge')
        else:
            print("\tinvalid value, try again")
    return green
```

3. **rescale()** - To alter the size of the green screen image but within a threshold so as to ensure that the green screen image doesn't exceed the size of the background image




```
[35] def rescale(bg,green):
        h,w,_ = green.shape
        h1,w1,_ = bg.shape
        maxs = min(h1/h, w1/w) - 0.01
        s = 0.0
        maxs = round(maxs,2)
        while(s<0.1 or s>maxs):
            s = float(input("enter the amount of scaling (between 0.1 and " + str(maxs) + ") ?...  "))
            if(s>=0.1 and s<=maxs):
                green = skimage.transform.rescale(green, s, multichannel=True)
            else:
                print("\tinvalid value, try again")
        return green
```

4. **blur()** - To blur the background image using a Gaussian blurring filter

Original image

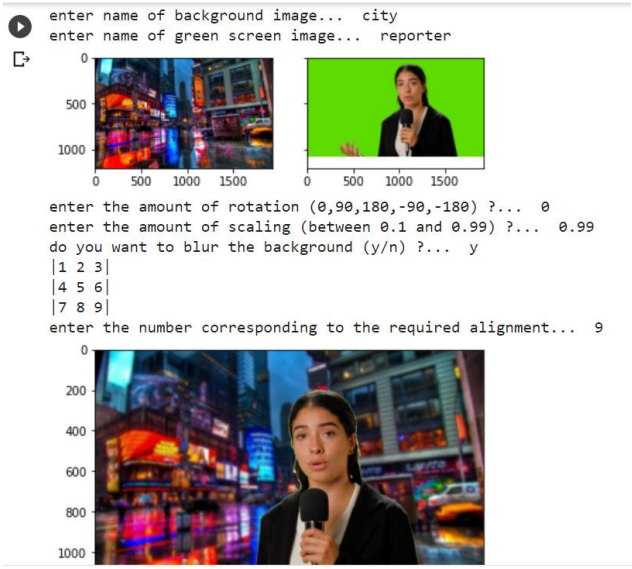
background blurred in final output

```python
def blur(bg,green):
    b = 'x'
    while(b!='y' and b!='Y' and b!='n' and b!='N'):
        b = input("do you want to blur the background (y/n) ?...  ")
        if(b=='y' or b=='Y'):
            bg = filters.gaussian(bg,sigma=5,multichannel=True)
        elif(b=='n' or b=='N'):
            break
        else:
            print("\tinvalid value, try again")
    return bg
```

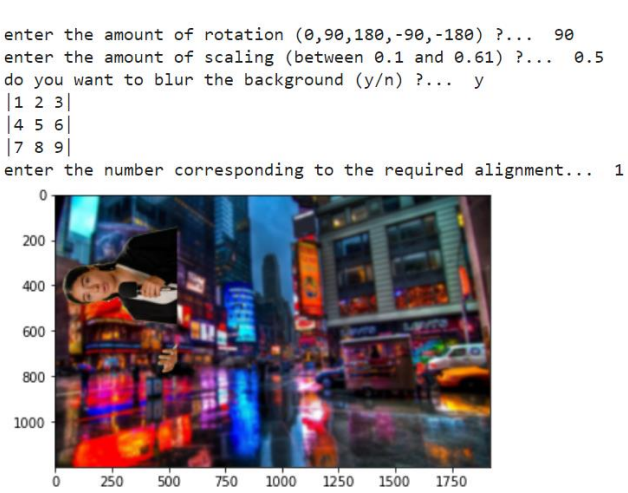
5. **align()** - For the spatial orientation of the green screen image on the background image.

   Examples with different alignment, rotation and scaling:

   a) alignment= 9(lower right)
   rotation= 0
   scale= 0.99

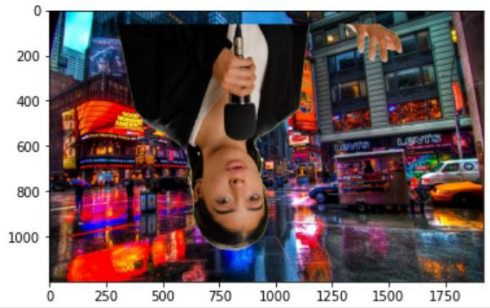


   b) alignment= 1(upper left)
   rotation= 90
   scale= 0.5



c) alignment= 5(center)
rotation= -180
scale= 0.99

```
enter the amount of rotation (0,90,180,-90,-180) ?...  -180
enter the amount of scaling (between 0.1 and 0.99) ?...  0.99
do you want to blur the background (y/n) ?...  n
|1 2 3|
|4 5 6|
|7 8 9|
enter the number corresponding to the required alignment...  5
```




```
[35] def align(bg,green):
         h,w,_ = green.shape
         h1,w1,_ = bg.shape
         a = 0
         print("|1 2 3|")
         print("|4 5 6|")
         print("|7 8 9|")
         while(a<1 or a>9):
             a = int(input("enter the number corresponding to the required alignment...  "))
             if(a>=1 and a<=9):
                 if(a==1):
                     coord=(0,0)
                 elif(a==2):
                     coord=(0,int(w1/2-w/2))
                 elif(a==3):
                     coord=(0,w1-w)
                 elif(a==4):
                     coord=(int(h1/2-h/2),0)
                 elif(a==5):
                     coord=(int(h1/2-h/2),int(w1/2-w/2))
                 elif(a==6):
                     coord=(int(h1/2-h/2),w1-w)
                 elif(a==7):
                     coord=(h1-h,0)
                 elif(a==8):
                     coord=(h1-h,int(w1/2-w/2))
                 elif(a==9):
                     coord=(h1-h,w1-w)
             else:
                 print("\tinvalid value, try again")
         return coord
```

6. **remove_green()** - Following the notion of ratio, the chroma-key strategy implemented in this project is based on the ratio of the green, red, and blue channels compared to the max-brightness level found on the pixel.

Therefore, if the green-ratio is found systematically higher than the red-ratio and the blue-ratio, it arguably implies the prevalence a green-ish color in the pixel.

Based on such distinction of green-ish pixel, a threshold constraint is imposed in order to determine whether to preserve the pixel by setting a close to zero alpha value, which eventually will lead to completely ignoring such pixels.

```
[43] def remove_green(img):

         norm_factor = 255

         """obtain the ratio of the green/red/blue channels based on
         the max-brightness of the pixel"""
         red_ratio = img[:, :, 0] / norm_factor
         green_ratio = img[:, :, 1] / norm_factor
         blue_ratio = img[:, :, 2] / norm_factor

         """darker pixels would be around 0. In order to ommit removing
         dark pixels we add .3 to make small negative numbers positive"""

         red_vs_green = (red_ratio - green_ratio) + .3
         blue_vs_green = (blue_ratio - green_ratio) + .3

         """now pixels with a negative value would have a high
         probability to be background green pixels."""
         red_vs_green[red_vs_green < 0] = 0
         blue_vs_green[blue_vs_green < 0] = 0
```

```
[43]     """combine the red_vs_green and blue_vs_green ratios
         to create an aplha mask"""
         alpha = (red_vs_green + blue_vs_green) * 255
         alpha[alpha > 50] = 255

         img[:, :, 3] = alpha
```

7. **blend()** – To combine all the implemented functions and obtain the desired result.


```
[35] def blend(bg, green):

         draw(bg,green)

         green = remove_green(green)
         green = rotate(bg,green)
         green = rescale(bg,green)
         bg = blur(bg,green)
         coord = align(bg,green)

         green = img_as_ubyte(green)
         bg = img_as_ubyte(bg)
         (x_size, y_size, _) = green.shape

         (x_ini, y_ini) = coord
         x_end = x_ini + x_size
         y_end = y_ini + y_size

         bg_crop = bg[x_ini:x_end,y_ini:y_end,:]

         """only the part of "green" with alpha values higher than 10 are taken
         "bg" pixels are replaced with corresponding "green" pixels for this part"""
         pixel_preserve = (green[:, :, -1] > 10)
```

```
[35]
        bg_crop = bg[x_ini:x_end,y_ini:y_end,:]

        """only the part of "green" with alpha values higher than 10 are taken
        "bg" pixels are replaced with corresponding "green" pixels for this part"""
        pixel_preserve = (green[:, :, -1] > 10)

        bg_crop[pixel_preserve] = green[pixel_preserve]

        bg[x_ini:x_end, y_ini:y_end, :] = bg_crop

        return bg
```

iii) Implementation:

```
"""
"bg" is the background image and "green" is the image with the green background
"""

bg_path = input('enter name of background image...  ')
bg = skimage.io.imread('/content/background/' + bg_path + '.jpg', pilmode="RGBA")

green_path = input('enter name of green screen image...  ')
green = skimage.io.imread('/content/greenscreen images/' + green_path + '.jpg', pilmode="RGBA")

img = blend(bg,green)

plt.imshow(img)
plt.show()

plt.axis("off")
plt.imshow(img)
```

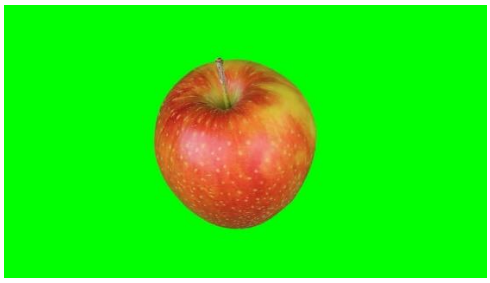
Final output

Results

Therefore, we were able to perform chroma key compositing on an image with a greenscreen as background and replace that with the desired background by removing a green background from an image while isolating an object, blending two images into a single image after applying basic rotation, translation and scale operations and applying gaussian blur filter on background image(if desired).

Few more examples using different background and greenscreen image combinations are:

1. A dessert with an apple composited on top:



2. Remote composited over a street/building:

3. An office with a clapperboard composited on top: