Conditional Statements and Advanced Image Manipulation

In previous examples, we have started to see that we can use `if`-statements to help us control how we make our color changes. We can continue with this idea by replacing one color with another color. Whenever the color of a pixel is "close enough" to the original color, we replace it. In a sense, we are looking at the "distance" between colors of pixels. (See "Aside: Distance Function" on the next page.)

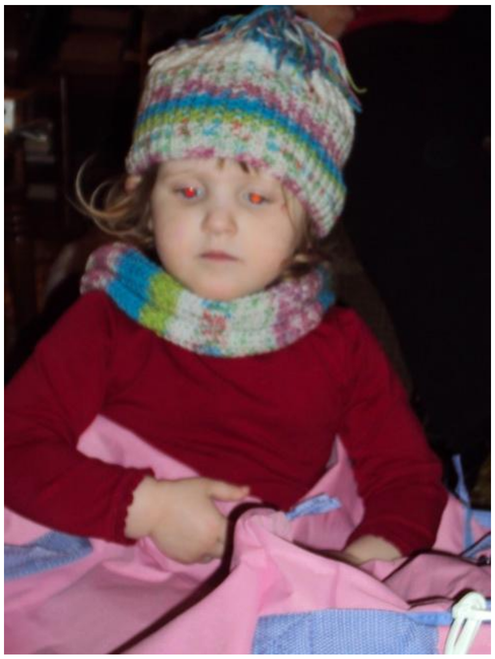
We will use this to remove red-eye in a picture as well as to replace backgrounds.

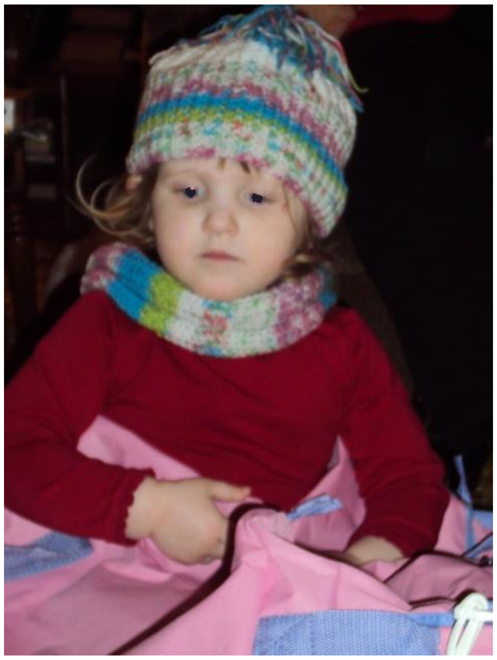
Example: Remove red-eye

```
def removeRedEye(pic, startX, endX, startY, endY, newColor):
  newPic = pic.copy()
  for y in range(startY, endY):
    for x in range(startX, endX):
      if distance(newPic.getpixel((x,y)),(255,0,0)) < 165:
        newPic.putpixel((x,y),newColor)
  return newPic
```

With a little experimenting to find the correct *x*- and *y*- ranges for the eyes and an appropriate new color to use, the `removeRedEye` function was called as follows:

```
pic = Image.open("/drive/MyDrive/ColabNotebooks/Picture1.jpg")
eye1 = removeRedEye(pic,127,147,134,148,(30,30,50))
eye2 = removeRedEye(eye1, 180,200, 140,152, (30,30,50))
```


Original Picture


Red-eye removed

Aside: Distance Function

The distance function used to compare colors is similar to that seen in mathematics. Recall that you can find the distance between two points in space by using the following formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

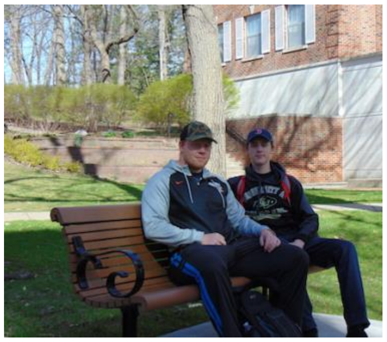
To compute the distance between the colors of two pixels, the following formula is used:

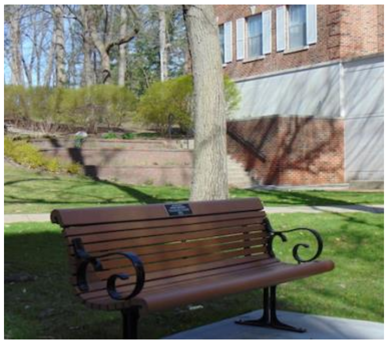
$$d = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2},$$

where (r_1, g_1, b_1) and (r_2, g_2, b_2) are the RGB values of the two pixels.

We can also use the distance function to aid in replacing backgrounds. Suppose we have a picture of someone and a picture of where they stood without them. Could we subtract the background of the person and then replace another background? We would have to figure out where the colors of the two pictures are exactly the same (i.e., the backgrounds should have the same colors in the regions where the person is not standing.)

In the following example, we have an image of 2 people on a bench, an image of just a bench, and then the new background, some ivy. We'd like to get the 2 people sitting in the ivy.


Original


BG only

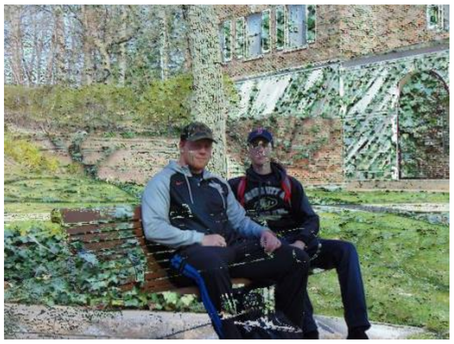
New BG

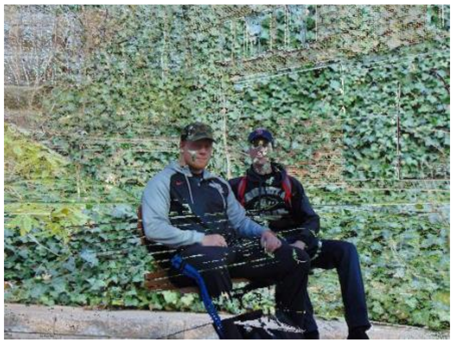
We can use the following function to attempt to do this:

Example: Swap backgrounds by subtraction

```
# replace background around subject with new background
def swapBack(bgWithPerson, onlyBG, newBG, threshold):
  newPic = bgWithPerson.copy()
  for y in range(newPic.height):
    for x in range(newPic.width):
      pxcolor = newPic.getpixel((x,y))
      bgcolor = onlyBG.getpixel((x,y))
      if distance(pxcolor, bgcolor) < threshold:
        newPic.putpixel((x,y), newBG.getpixel((x,y)))
  return newPic
```

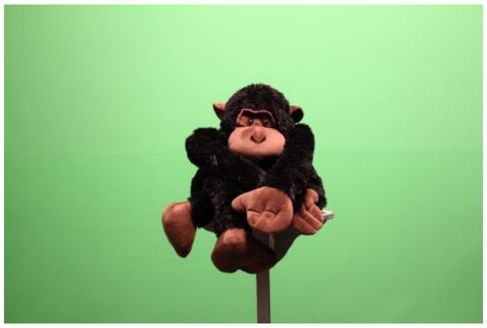
In this function, the parameter `bgWithPerson` is a picture with people in front of some background, the parameter `onlyBG` is a picture of just that background, and the parameter `newBG` is some new background, like the Eiffel Tower. When we compare the colors of corresponding pixels in the `bgWithPerson` and the `onlyBG` pictures, the difference between these colors should be very small, except where the people are. So, we would check where that distance is small, and then replace the color of those pixels with the color from the pixels in the new background.


First try, threshold 25

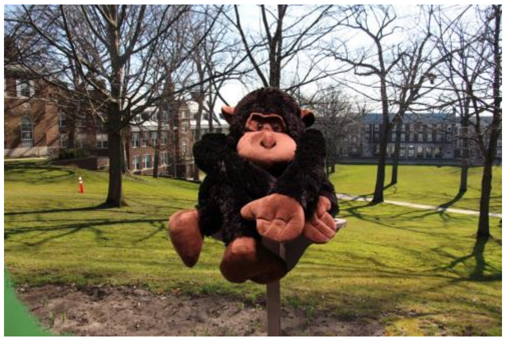

Second try, threshold 40

In these results, we see that the first threshold of 25 seems to be too small to get much of the ivy to show through. When increasing the threshold to 40, we get more of the background to be replaced, but parts of the people are getting replaced as well. There are various issues that cause this, including how the pictures were taken, as well as slight changes in lighting and shadows between the pictures.

Television producers will sometimes use an effect called *chromakey*. This is very common with weather forecasters. The idea is that the person will stand in front of a solid color background (typically green or blue) and then the background will be replaced with some other image, like a map or important building. In the following example, we have two girls in front of a green screen, and then a picture of them on campus. The green background then gets replaced with the campus picture.


green-screen picture

background picture


Result of chromakey

The following function was used to produce this new image. Depending on the actual shade of green in the green screen picture, the definition of green (*i.e.*, the condition in the `if`-statement) may need to be altered.

```
# replace the green background with a more interesting
# background
def chromakey(subjectOnGreen, bg):
  newpic = subjectOnGreen.copy()
  for y in range(newpic.height):
    for x in range(newpic.width):
      rvalue, gvalue, bvalue = newpic.getpixel((x,y))
      if rvalue < 200 and bvalue < 190 and gvalue > 150:
        newpic.putpixel((x,y), bg.getpixel((x,y)))
  return newpic
```

The condition in the `if`-statement could be further refined to remove the little bit of green in the bottom left corner, if desired.

Activity: Color Replacements