# FINAL TEAM PROJECT REPORT

Computer Vision - CS231.J21.KHTN

# CAMERA360

Lecturer: Nguyen Vinh Tiep
Vo Thi Huyen Trang, 16521283
Duong Chi Vinh, 16521438
Le Nguyen Ky Duyen, 16520311

***Abstract*—**

**One of the most common tasks in computer vision is image processing. Often this means some operations on an image, in order to get an enhanced image or to extract some useful information from it. This article describes some methods used in a process that we call Camera360. We illustrate three major operations, namely Face Filter, Filter Camera and Color Transfer.**

Fig. 2. We use Alpha Blending to put image overlays

## I. Face Filter

You are probably familiar with Face Filter. It provides filter features which you can put some cool and funny images on your portrait.

**Method:**

To put some image overlays on your face. We detect your face in which you put image overlay on. We use Dlib to detect it. This function returns coordinate of each positions in the face.

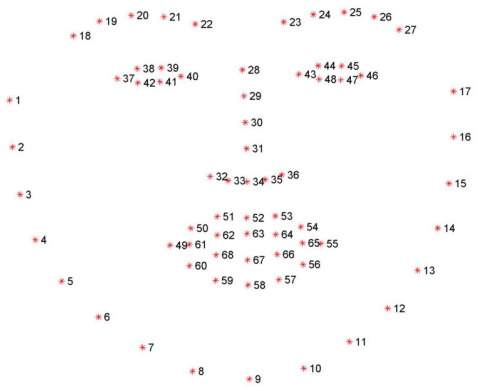

Fig. 1. If you want to get position of eyes on your face. You can get it by approaching array from 37 - 42 or 43 - 48

After getting the positions you need. Then, We put image overlays on the face. *Notice that the image putting on it is a PNG, with a transparent background. The transparency is important, as we only want to place the eyes, without a white box around it*. About Alpha blending method, it is the process of overlaying a foreground image with transparency over a background image. The transparency is often the fourth channel of an image ( e.g. in a transparent PNG), but it can also be a separate image. This transparent mask is often called the alpha mask or the alpha matte. At every pixels of the image, we need to combine the foreground image color (F) and the background image color (B) by using the alpha mask ($\alpha$).

*Note*: The value of $\alpha$ used in the equation is actually the pixel value in the alpha mask divided by 255. So, in the equation below, $0 \leq \alpha \leq 1$

$$I = \alpha\ F + (1 - \alpha)\ B$$

From the equation above, you can have the following observations.

1. When $\alpha = 0$, the output pixel color is the background.
2. When $\alpha = 1$, the output pixel color is simply the foreground.
3. When $0 < \alpha < 1$ the output pixel color is a mix of the background and the foreground. For realistic blending, the boundary of the alpha mask usually has pixels that are between 0 and 1.

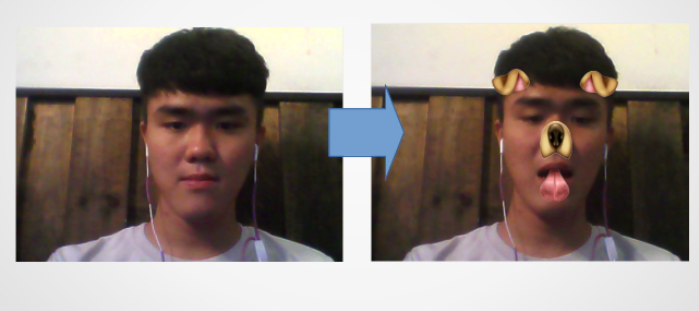

Fig. 3. This is an image putting a filter

**Demerits:**

The size of image depends on that of camera, which usually make a big image. Moreover this method computes on each image pixel leading to slow processing speech.

## II. Color Transfer between Images

Our implementation of color transfer is (loosely) based on Color Transfer between Images by Reinhard et al, 2001. In this paper, Reinhard and colleagues demonstrate that by utilizing the L*a*b* color space. Unlike histogram based color transfer methods which require computing the CDF of each channel and then constructing a Lookup Table (LUT), this method relies strictly on the mean and standard deviation of the pixel intensities in the L*a*b* color space, making it extremely efficient and capable of processing very large images quickly.

Fig. 4. Applying the colors of a painting to a portrait.

We use a simple statistical analysis to impose one images color characteristics on another. We can achieve color correction by choosing an appropriate source image and apply its characteristic to another image.

**Method:**

Because our goal is to manipulate RGB images, which are often of unknown phosphor chromaticity, we *first* show a reasonable method of converting RGB signals to color space L*a*b* (both the source and the target image) by using function cv2.COLOR-BGR2LAB in Opencv.

Then, we compute the mean and standard deviation of the pixel intensities for each of the L*, a*, and b* channels, respectively (for both the source and target images).

First, we subtract the mean from the data points:

$$l^* = l - \langle l \rangle$$
$$\alpha^* = \alpha - \langle \alpha \rangle$$
$$\beta^* = \beta - \langle \beta \rangle$$

Then, we scale the data points comprising the synthetic image by factors determined by the respective standard deviations:

$$l' = \frac{\sigma_t^l}{\sigma_s^l} l^*$$
$$\alpha' = \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha^*$$
$$\beta' = \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta^*$$

Next, instead of adding the averages that we previously subtracted, we add the averages computed for the photograph. Finally, we convert the result back to RGB.

*Also, we make a function in order to clip values that fall outside the range [0, 255] (in the OpenCV implementation of the L*a*b* color space, the values are scaled to the range [0, 255], although that is not part of the original L*a*b* specification). To be more precise, please take a look at the attached code!*

**Demerits:**

While the Reinhard et al. algorithm is extremely fast, there is one particular downside it relies on global color statistics, and thus large regions with similar pixel intensities values can dramatically influence the mean (and thus the overall color transfer).

*The results quality depends on the images similarity in composition!*

## III. Camera Filter

In the age of smart phone, its worth explaining what the term filter used in photography. A filter is used to alter or adjust the light coming into the camera in some way. Lets take a look at the five best camera filters that you can use to enhance your photos.

There are in fact many different filters which you can use, and not all of them are going to be mentioned here.

**Inverse filter:**

Inverts every bit of an image/video.

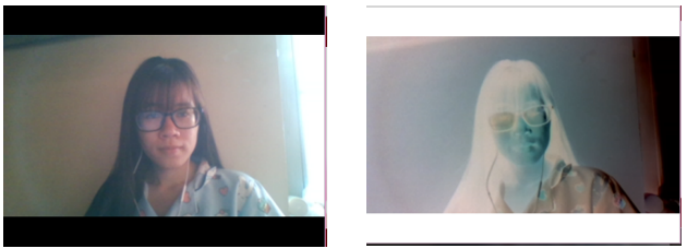


Fig. 5.

**Sepia filter:**

Converting a color image into sepia image is very simple. All we have to do is repeating 3 simple steps for each pixels of the image.

1. Get the RGB value of the pixel.
2. Calculate tr, tg and tb using the formula
tr = 0.393R + 0.769G + 0.189B
tg = 0.349R + 0.686G + 0.168B
tb = 0.272R + 0.534G + 0.131B
*Take the integer value.*
3. Set the new RGB value following below condition:

If tr > 255 then r = 255 else r = tr
If tg > 255 then g = 255 else g = tg
If tb > 255 then b = 255 else b = tb

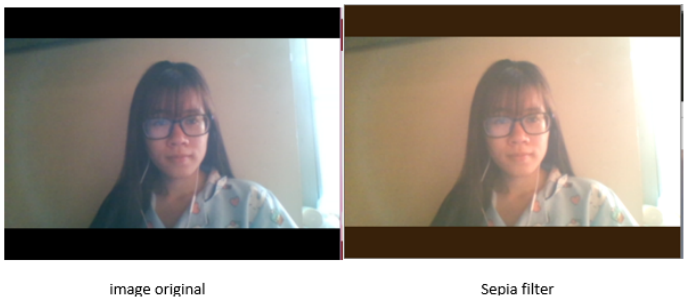


Fig. 6.

**Circle blur filter:**

Rule: Draw a circle at the center position. Then use the Gaussian Blur filter to blur the surrounding areas.

Gaussian equation:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

1D

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2D (Image)

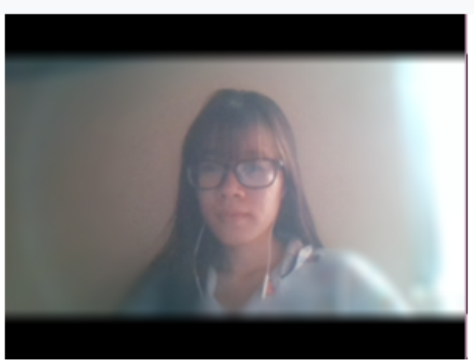

Fig. 7. Circle blur filter example

## IV. Division of labor

1. Coding(program):
Face Filter: Duong Chi Vinh
Color Transfer: Vo Thi Huyen Trang
Camera Filter: Le Nguyen Ky Duyen
2. Work on Reprt:
Face Filter: Duong Chi Vinh
Color Transfer: Vo Thi Huyen Trang
Camera Filter: Le Nguyen Ky Duyen
3. Presentation in class:
Face Filter: Duong Chi Vinh
Color Transfer: Vo Thi Huyen Trang
Camera Filter: Le Nguyen Ky Duyen

## References

[1] Computer Vision: Algorithms and Applications(Richard Szeliski)

[2] Color Transfer between Images(Reinhard et al, 2001, University of Utah)

[3] wedsite: digital photography school.com. Post: 5 camera filters enhance your photography

[4] wedsite: opencvexamples.blogspot.com. Post: bitwise and or xor and not

[5] wedsite: doc.opencv.org