CSC320: Project 4

Triangulation Matting

Ramaneek Gill and Ryan D'Souza

March 30, 2015

Part 1

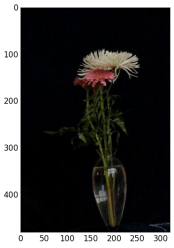
$$C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$C = F + (1 - \alpha)B$$
$$C = F + B - B\alpha$$
$$C - B = F - B\alpha$$
$$\Rightarrow$$

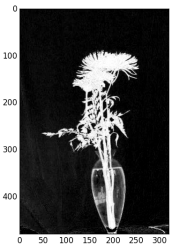
$$\begin{pmatrix} 1 & 0 & 0 & -B_1^r \\ 0 & 1 & 0 & -B_1^g \\ 0 & 0 & 1 & -B_1^b \\ 1 & 0 & 0 & -B_2^r \\ 0 & 1 & 0 & -B_2^g \\ 0 & 0 & 1 & -B_2^b \end{pmatrix} \begin{pmatrix} F_r \\ F_g \\ F_b \\ \alpha \end{pmatrix} = \begin{pmatrix} C_1^r - B_1^r \\ C_1^g - B_1^g \\ C_1^b - B_1^b \\ C_2^r - B_2^r \\ C_2^g - B_2^g \\ C_2^b - B_2^b \end{pmatrix}.$$

Part 2

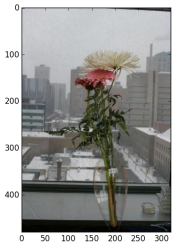
The input images were 4 input images. 2 were composite images of a flower with a white background and other one with a black background. The two different background images (white and black) were also given. These 4 input images were used with another image (a window background) to produce a new composite image a flower as the foreground with triangulation matting.



(a) The foreground computed from using two composite images with the same foreground but different backgrounds



(b) The alpha channel computed



(c) The custom composite image with the same foreground but new background

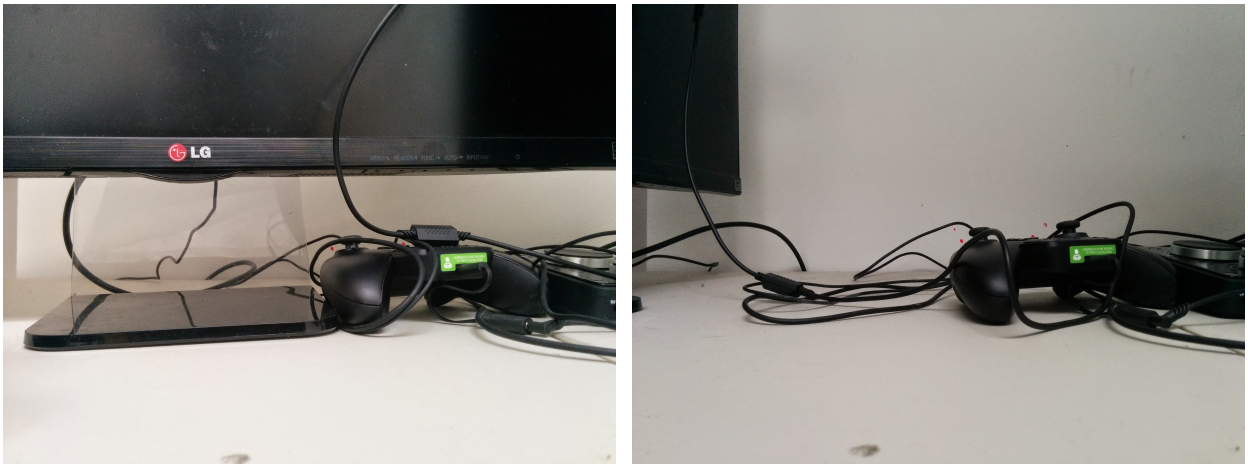
Figure 1: Images computed through the triangulation matting algorithm

Part 3

There is no way I can produce an image where the foreground is the exact same with two different backgrounds since I don't have a tripod, my only camera is the one included in my phone so the results will be very shaky. Below are my 5 input images and the results of the triangulation matting.

I got these images by doing the following:
1. Take a picture of a background.
2. Move foreground object in front of background (the mug in the figures below) and take a picture.
3. Change background, leave mug in same exact place and take a picture.
4. Remove foreground and take a picture.
5. Try to do all of these steps while balancing the camera (android phone) on a stable surface and try to move its angle of view as little as possible.

The fifth step (the lack of a tripod) was the reason the output from the triangulation matting algorithm was horrible. The input image backgrounds were not as similar we want them to be since the camera can view a very different image (because of projection) when comparing the data pixel-wise. From an object recognition standpoint the input images are perfect.



(a) Background 1 (b) Background 2

Figure 2: Background input images

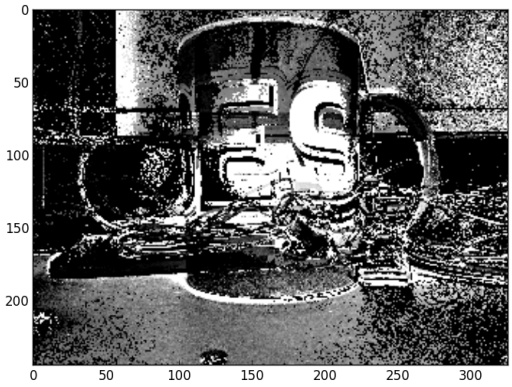
Now for the results of the triangulation matting algorithm:

(a) Composite 1, (includes b1) (b) Composite 2, (includes b2)

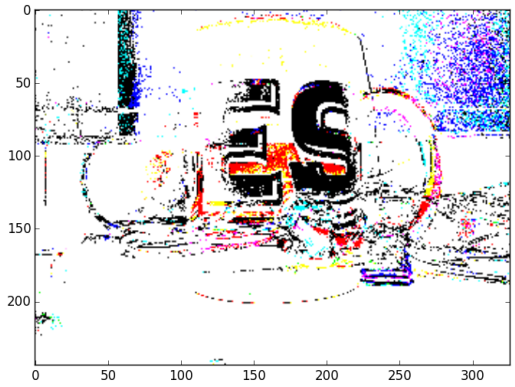
Figure 3: Composite input images

(a) A background we want to combine with the mug (foreground object) in the two images directly above.

Figure 4: New background input image

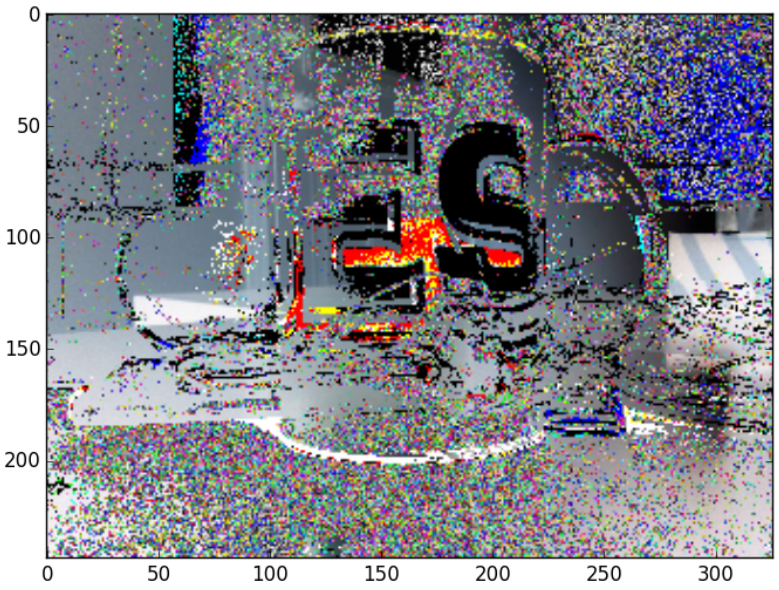


(a) Alpha image computed with the images in Figure 2 and 3.



(b) Foreground image computed with the images in Figure 2 and 3.

Figure 5: Composite input images



(a) The new composite image produced by doing triangulation matting with figure 4 and 5. Pretty lousy results eh? Blame the output of these input images on my lack of skills as a photographer, not a computer scientist!

Figure 6: New background input image

p4.py:

```
from pylab import *
import numpy as np
import random
import matplotlib.cbook as cbook
import random
import time
from scipy.misc import imread
from scipy.misc import imsave
from scipy.misc import imresize
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os

matplotlib
gray()

os.chdir('C:/Users/Ramaneek/SkyDrive/Documents/Github/CSC320-Winter-2014/project 4/')

# c = f + (1-alpha)b

## MAIN
b1 = imread('images/flowers-backA.jpg')/255.0
c1 = imread('images/flowers-compA.jpg')/255.0
b2 = imread('images/flowers-backB.jpg')/255.0
c2 = imread('images/flowers-compB.jpg')/255.0
new_b = imread('images/window.jpg')/255.0

print "Read the input images"

new_c = np.zeros(new_b.shape)
alpha = np.zeros((b1.shape[0], b1.shape[1]))
f = np.zeros(b1.shape)

for i in range(b1.shape[0]):
    for j in range(b2.shape[1]):
        #set A
        A = np.matrix([ [1,0,0,-b1[i,j,0]],
                        [0,1,0,-b1[i,j,1]],
                        [0,0,1,-b1[i,j,2]],
                        [1,0,0,-b2[i,j,0]],
                        [0,1,0,-b2[i,j,1]],
                        [0,0,1,-b2[i,j,2]] ])

        #get the pseudo inverse of A
        A_pinv = linalg.pinv(A)

        #set b
        b = np.matrix([ [c1[i,j,0]-b1[i,j,0]],
                        [c1[i,j,1]-b1[i,j,1]],
                        [c1[i,j,2]-b1[i,j,2]],
                        [c2[i,j,0]-b2[i,j,0]],
```

```
                    [c2[i,j,1]-b2[i,j,1]],
                    [c2[i,j,2]-b2[i,j,2]] ])

        #solve for x by A^-1*b
        x = np.clip(dot(A_pinv,b), 0, 1)

        #set foreground
        for k in range(3):
            f[i,j,k] = x[k]

        #set alpha channel
        alpha[i,j] = x[3]

        #set new composite image
        for k in range(3):
            new_c[i,j,k] = f[i,j,k] + np.dot((1-alpha[i,j]), new_b[i,j,k])

print "Computed alpha channel and isolated the foreground"
figure(1)
imshow(alpha)
figure(2)
imshow(f)
figure(3)
imshow(new_c)

show()
```