CPSC 4040/6040
Computer Graphics Images

Assigned: Sept. 15, 2015
Due: 11:59pm, Tues., Sep. 29, 2015

Programming Assignment 3 — Green Screening *(Grading: 0–10 points)*

Problem Description

In this assignment you will develop two programs for green screening. Green screening is a process whereby a range of colors in an image (typically highly saturated greens) is used to determine an alpha channel for the image, for purposes of compositing. There are two steps.

First, you are to write code to generate an alpha channel mask for an image based upon image color information. Second, you will complete an image compositing program that uses the alpha channel information of a foreground image and computes the **over** operator. All final tests of your program and your homework submission may use any of the provided green screen images, but you are welcome to also use other green screen images you can find. For compositing, you can composite this picture onto any background image that you choose.

Both programs should make use of OIIO, and the second should make use of OpenGL *only* for displaying the compositing image, not computing it. These should build on the software base you developed for Programming Assignment 01.

Basic Requirements

Masking Your first program should be called `alphamask` and it should take an input image and use its color information to produce a four channel version of the image. The RGB channels of the image should remain unaltered, the only new data this program creates is the fourth α channel. I expect that you will call the program by using:

```
alphamask input.img output.img
```

Where `input.img` is any image file that can be read by OIIO and `output.img` is an image file that supports four channels. Of course, the expectation is that you will work with image formats that are supported by OIIO, thus you can expect to replace `.img` appropriately. In particular, the file type `.png` supports an alpha channel, but this is not the only one.

As we discussed in class, there are many techniques for green screening. For this assignment, I suggest you start with a simple approach and then try progressively more sophisticated techniques. Specifically, your first approach should allow for setting the alpha channel based on a range of hue values. To do so, first convert the color data to HSV and next select the pixels to mask based on those which have a hue near the value 120 (green on a 360 degree scale). I have included, in the appendix, some sample code to convert RGB to HSV values. Any pixel that falls within your hue threshold should be converted to have a transparent mask ($\alpha = 0$), while any pixel that falls outside of threshold should be opaque ($\alpha = 255$)

Next you should experiment in two directions:

1. Consider masking pixels based on not just hue, but also saturation and value. I suggest that your program read an input parameter file containing these values, so that you can simply edit this file and rerun your program while you tune and debug (this may be slightly less painful than doing all parameters as command line). No doubt, it will require some experimentation to get the range of hues and the thresholds set right to get results that you like. This arrangement will also allow you to easily modify the colors used to determine the alpha channel so that you can do "red screening", "blue screening" or even "black screening".

Be sure to include in the README exactly how to run your program and your preferred settings for test images.

2. Experiment with using a full greyscale mask for alpha, not just a binary mask as described above. Your function for masking pixels should be extended to alow for setting α to values other than 0 or 255 by examining color information.

Compositing The second program you are required to write is a compositing program called `composite`. It should read in two color images and their alpha channels, and display their composite (A **over** B). If the alpha channel is missing for the second image, your program should assume that the image is a background image (i.e. an image that is completely opaque). Your program should display the composite of A and B, and allow the user to write to an optional third file containing the newly composited image.

This means you must support:

```
composite A.img B.img
```

As well as

```
composite A.img B.img out.img
```

Where `A.img` and `B.img` are two image files, `.img` is an extension that OIIO supports, and in particular `A.img` uses a four channel data input format. `out.img` is an output file image.

Within your composite program, you must write your own code to compute A **over** B. Do not use `glBlend()` or an other OpenGL operation.

Code Documentation Please see Programming Assignment 01 for the expected requirements. Your code will be evaluated based on code structure, commenting, and the inclusion of a README and build script.

Advanced Extension for 6040 students (worth 4 points)

Extend your `alphamask` program to provide interactive masking using two different functions. Use two different command line flags to set the mode of interaction. The first flag will enable the Petro Vlahos algorithm mode, and assign α values based on his equations and a keyboard input. You should be able to execute your code with or without this flag, for example:

```
alphamask -pv
```

Initialize k to be 1, where $G = kB$ is the equation we discussed in class. When the user hits `UP` on the keyboard, it should increase the value of k. Likewise, pressing `DOWN` should decrease the value. Look up `glutSpecialFunc()` to handle these key presses. Only a binary mask is required, but just like the basic requirements you may want to consider how a greyscale alpha mask could be used.

Your second flag will enable spill suppression (by default, none will be applied). You should be able to execute your code with or without this flag, for example:

```
alphamask -ss
```

would execute the command and apply spill suppression. Note that the basic requirement for the assignment only creates alpha values—you should use this flag as a way to enable the modification of color values in the output image. You may choose any scheme you like to adjust colors, either

using some of the basic techniques we discussed in class that adjust the green channel using simple schemes or a scheme of your own design. You need not have mouse/keyboard interactions for this flag, but you can consider it.

Both of these flags can either be used separately, or together. In theory, with both, you should be able to achieve some reasonably high quality green screen effects!

Submission

(Please read all of these instructions carefully.)

Please write the program in C or C++, and I recommend using OpenGL and GLUT graphics routines for the display. Please take extra care to make sure that your homework compiles on the School of Computing's Ubuntu Linux cluster—testing on your own home machine, even if it runs Ubuntu, may not be sufficient. Remember:

> both a working build script and README must be provided

Consequently, to receive *any credit whatsoever* this code must compile on the cluster, even if it does not accomplish all of the tasks of the assignment. The grader will give a zero to any code that does not compile.

Submit using the handin procedure outline at `https://handin.cs.clemson.edu/`. You are welcome to use the commandline interface, but the web interface is sufficient. The assignment number is `pa03`.

Finally, since we are using multiple files, please only submit a single file which has aggregated everything. This file should be named `[username]_pa03.tgz` where `[username]` is your Clemson id (please remove the brackets []). For example, mine would be `levinej_pa03.tgz`. Please make sure your build script is at the top level directory.

Rubric for Grading (Programming Assignment 3 — Green Screening)

4040 students will be graded based on the following requirements:

I. +4 Masking

This can include: correctly reading and writing of images, correctly computing HSV values, and building (at least a) binary alpha mask based on hue range, using parameters for specifying a range of values in hue, saturation, and value to set α, and supporting a greyscale alpha mask.

II. +3 Compositing

This can include: computing the composite correctly, displaying the composited image A **over** B, and writing out the composited image (when a third filename is specified).

III. +2 Clarity

Does the code have good structure? Is the code commented and is a README provided? Is a working build script provided?

IV. +1 Quality

Finally, we will evaluate your code based on the quality of the code's ability to produce an alpha mask and composite, subject to the grader's discretion.

6040 students will receive 6 points for completing the above requirements, scaling the above by 60%. To achieve 10 points they must also complete:

I. +2 Petro Vlahos Method

Code supports the Petro Vlahos algorithm with the switch `-pv`. An interactive alpha mask program is shown, where the user can vary the parameter k in the Petro Vlahos algorithm.

II. +2 Spill Suppression

Code supports spill suppression with the switch `-ss`, using an algorithm of their choice to adjust the colors of the output image. Quality of the code's ability to spill suppress and reduce the excess green, subject to the grader's discretion.

Going above and beyond these requirements may result in extra credit at the discretion of the instructor and grader. Please note any extra features you implement in the README.

Appendix: Code to convert RGB to HSV

Adapted from Foley, Van Dam, Feiner, and Hughes, pg. 592.

```
/*
  Input RGB color primary values: r, g, and b on scale 0 − 255
  Output HSV colors: h on scale 0–360, s and v on scale 0–1
*/

#define maximum(x, y, z) ((x) > (y)? ((x) > (z)? (x) : (z)) : ((y) > (z)? (y) : (z)))
#define minimum(x, y, z) ((x) < (y)? ((x) < (z)? (x) : (z)) : ((y) < (z)? (y) : (z)))

void RGBtoHSV(int r, int g, int b, double &h, double &s, double &v){

  double red, green, blue;
  double max, min, delta;

  red = r / 255.0; green = g / 255.0; blue = b / 255.0;  /* r, g, b to 0 − 1 scale */

  max = maximum(red, green, blue);
  min = minimum(red, green, blue);

  v = max;        /* value is maximum of r, g, b */

  if (max == 0) {    /* saturation and hue 0 if value is 0 */
    s = 0;
    h = 0;
  } else {
    s = (max - min) / max;          /* saturation is color purity on scale 0 − 1 */

    delta = max - min;
    if (delta == 0) {                  /* hue doesn't matter if saturation is 0 */
      h = 0;
    } else {
      if (red == max) {                /* otherwise, determine hue on scale 0 − 360 */
        h = (green - blue) / delta;
      } else if (green == max) {
        h = 2.0 + (blue - red) / delta;
      } else {/* (blue == max) */
        h = 4.0 + (red - green) / delta;
      }
      h = h * 60.0;
      if(h < 0) {
        h = h + 360.0;
      }
    }
  }
}
```