Blending

Computational Photography: Project 3

1 - Objective

The goal of this project is to implement a basic photo blending framework in order to combine multiple images with alpha masks to create a new, single image of blended segments of the originals.

2 - Deadline

It should be submitted on T-Square by 11:55PM on Monday.

3 - Process

3.1 Download the base source

Download and unzip the folder with the base code for this project.

3.2 Project description

You will complete this project in stages. Each stage will have a result which you can compare with examples below and which you must submit with the code. Please start early, this project will be more involved than the last 2.

A common technique in computational photography is blending multiple images together by adjusting the translucency (alpha) value of one or more of the images and overlapping them. The color for each pixel in the final image is then calculated as a weighted average of the corresponding pixel in each overlapping images where the weights are the alpha values of each image.

You will implementing a version of this 'alpha blending' method. You will do so in 3 stages, described below. For each stage you will produce a result image which you will save and add to your write-up PDF. You will also save a working version of your code for each stage for submission.

Please note before you begin that you are NOT be allowed to use a PImage for the final blended image. You must interface with the canvas pixels directly, as shown in the source codes.

1. Blending Equivalent Images

Write all code for part 1 in p1.pde within the p1 folder.

Your first goal is to blend two overlapping images of the same size given an alpha mask. We will view this process as having one opaque image, the *canvas* C, and one overlapping and partially transparent image which we will call a *layer* L. The opacity of this *layer* is defined by the values given in the *mask* image M. With these terms defined we can write an equation for the color of any pixel in the resulting, blended image B as follows (where i,j are the x and y location of the pixel):

$$B_{Ri,j} = (M_{i,j} * L_{Ri,j}) + ((1 - M_{i,j}) * C_{Ri,j})$$

$$B_{Gi,j} = (M_{i,j} * L_{Gi,j}) + ((1 - M_{i,j}) * C_{Gi,j})$$

$$B_{Bi,j} = (M_{i,j} * L_{Bi,j}) + ((1 - M_{i,j}) * C_{Bi,j})$$

You may recognize this as the linear interpolation formula and you would be correct. The value of the *mask* M is our interpolation parameter and is always between 0 (translucent) and 1(opaque) and modifies determines how much of the *layer* image is shown over the *canvas*. Note that when you read the *mask* image in, you will most likely get the value in the range of 0-255. So, you will need to scale them before using the above formula. If you have done the blending properly with the *mask*, *canvas* and *layer* images defined in the source code, you should get an image identical to the example below. Save your resulting image and add it to your write-up PDF file.

---

2. Mobile Layer Blending

Write all code for part 2 in p2.pde within the p2 folder.

Once you can blend 2 images with an alpha mask, you may find that you would like to blend images of different sizes and/or offset your *layer* from your *canvas*. That will be your 2nd goal. The source code p2.pde will load in a *canvas* and *layer* of different dimensions such that part of the *layer* image will exist outside of the *canvas*. As such, your code from part 1 will probably break, part of your job in this part is to fix this. You can think of this as clipping the *layer* such that you only touch the pixels which remain inside of the canvas by finding the correct starting and ending x and y values for your iteration through the *layer* image.

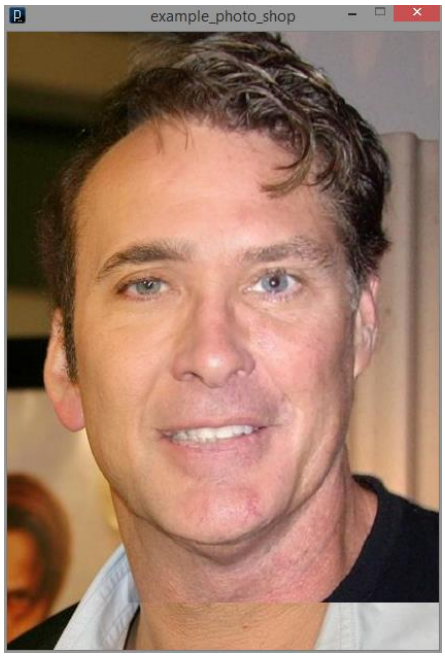
In order to have your *layer* offset from the *canvas*, you will need to keep track of its location (the location of its top left corner).

(At this point, the suggested strategy is to create a Layer class in a new tab which will store the *layer* image, the *mask* image for this *layer* and the x,y location of its top left corner. You will be needing multiple distinct *layers* for part 3).

Having the location of the top left corner of the image, as defined above, we can make a slight adjustment in the equation from part 1:

$$B_{Ri,j} = (M_{i+x,j+y} * L_{Ri+x,j+y}) + ((1 - M_{i+x,j+y}) * C_{Ri,j})$$
$$B_{Gi,j} = (M_{i+x,j+y} * L_{Gi+x,j+y}) + ((1 - M_{i+x,j+y}) * C_{Gi,j})$$
$$B_{Bi,j} = (M_{i+x,j+y} * L_{Bi+x,j+y}) + ((1 - M_{i+x,j+y}) * C_{Bi,j})$$

Once you have made the aforementioned adjustments, you should be able to blend the "cage-hoff" image and move the "hoff" *layer* into a more reasonable location such as that shown below:

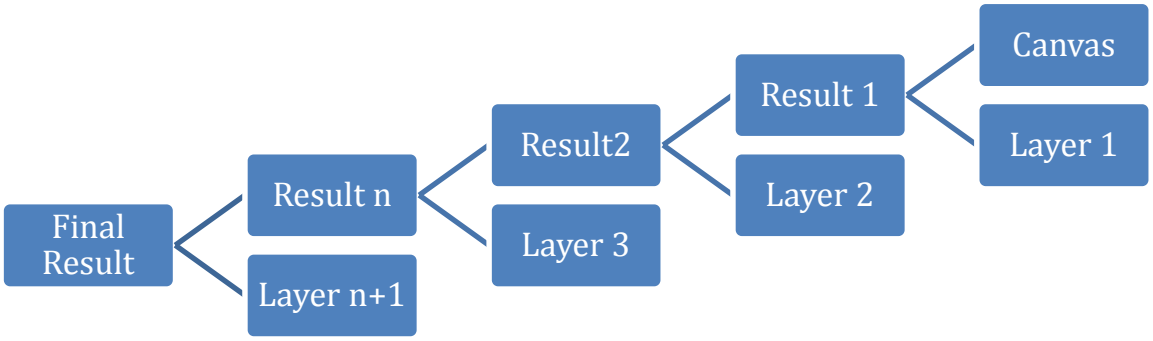


---

3. Multiple Layers

Write all code for part 3 in p3.pde within the p3 folder

Your final goal will be to allow a user to load multiple *layers* all with their own pre-defined *mask*, move the layers around and blend them together into a single image to create a mosaic like effect. Until now, we have only combined one image with another. Now, we must consider how to blend multiple images together. Since the sum of all masks located at a single pixel could be greater than 1, we cannot simply combine them all as a weighted average. Instead, we will chain the blend operations, considering only one *layer* at a time. Beginning with *layer* 1, we will calculate the blend between the *canvas* C and that *layer* L_i. We will then use the result of this blend as the new C and calculate the blend for *layer* L_{i+1}. We will continue to do this until all the layers are blended into a final resulting image. The graphic below shows this:



You should be able to support up to 10 *layers* of any dimension, each with a distinct *mask*, and each able to be selected with the number keys 0-9 and moved by dragging the mouse. Once you have this framework you should create a blended mosaic image of no less than 4 *layers* over a *canvas*. This mosaic should be unique and creative. It may be ridiculous (such as the example shown below) or tastefully artistic. Your mosaic will be shown to the class in the lecture on Tuesday following the Monday submission deadline.

Project Write-up:

You should save 1 image from each of the above parts and place them all, with captions into a PDF document. On the last page of the document, after the images, write a brief paragraph about the mosaic you chose, which parts of the project you found most challenging and how long you worked on the assignment. Provide any feedback you may have about project structure or content as well (if you prefer to do so anonymously, piazza is also an option, but we will not grade you based on your feedback here).

3.3 Source code

Each Processing sketch loads in two images and an alpha mask and displays the first *canvas* image. It is up to you to do more.

You should modify the source code as explained in the description and comment your code (include your name in the header). The source code is written in Processing. Visit "Processing.org/reference/" for more information on built in functions and structure. Please note that **you are not allowed to use built in processing functions** to accomplish the tasks listed in the project description. We want you to access and directly manipulate the pixels in question to perform each task. **Please note before you begin that you are NOT be allowed to use a PImage for the final blended image. You must interface with the canvas pixels directly, as shown in the source code.**

When in doubt, ask.

3.4 Authorship Rules

The code that you turn in entirely your own. You are allowed to talk to other members of the class and to the Professor and the TA about general implementation of the assignment. It is, for example, perfectly fine to discuss how one might organize the data for a matrix stack. It is also fine to seek the help of others for general Processing/Java programming questions. You may not, however, use code that anyone other than yourself has written. Code that is explicitly not allowed includes code taken from the Web, from books, from previous

assignments or from any source other than yourself. You should not show your code to other students. Feel free to seek the help of the Professor and the TA's for suggestions about debugging your code.

3.5 Submission

For this assignment, you will submit 3 Processing sketches and a PDF as stated in the project description.

Your unedited pictures and masks should be located in the data folder where the default photos also reside for each Processing sketch you submit. Your write-up in PDF format should be in the same folder as your collection of code folders. In order to run the source code, it must be in a folder named after the main file. When submitting any assignment, leave code in this folder such that all code will run without any changes to folder structure or code. For this assignment, compress the parent folder containing your code folders and PDF and submit that via T-square.