Comp Photography
Final Project

Christopher DeJager
Spring 2017
cdejager3@gatech.edu

Palette Recolorization

The original focus of this project was to identify dominant hues in an image and then give the option to change them. I was successful at doing this so I also added the ability to take image masks to change the hue of localized regions.

The Goal of Your Project

Original project scope:

I would like to write a program which can identify palettes in an image and then you can optimally change the color of them. I intend to do this with two passes. The first pass would generate a histogram of the image and then would either automatically identify the pallets based on spikes in the image colors. In the second pass the user could identify what colors to change the pallets to.

What motivated you to do this project?

Messing with pallets in an image allows you to really change the charistic of an image. The “Palette-based Photo Recoloring” paper shows this. I hoped to recreate this by selecting dominate hues and then changing them. Doing this I thought I could take a picture from nature and make it look like it was from an alien world.

Scope Changes

These are common. Did you run into issues that required you to change project scope from your proposal?

I managed to write a program that could identify hues in an input image rather easily. Since I was not really interested in making a fancy gui for it I then went about making a program that would only change the colors locally. As shown in the “Colorization Using Optimization” paper.

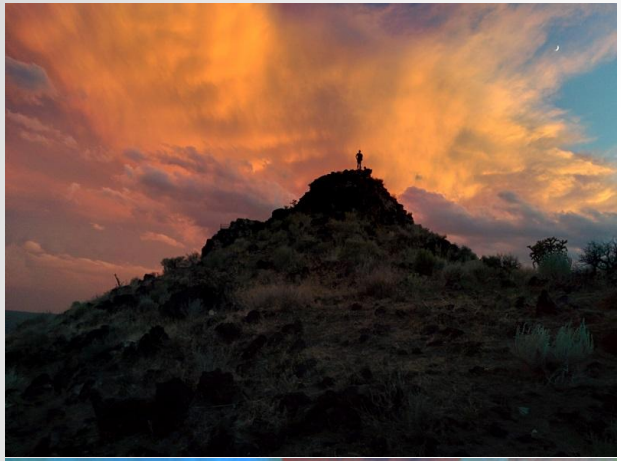
Fully changing the hue, saturation, and value of the output pixels proved to be challenging to make look good so my final product only changed the hue of the output area and left the saturation and value alone because this seemed to look much better. Unfortunately though you cannot really change the shade of the output colors because I am only changing hue.

Showcase

Full Hue Change

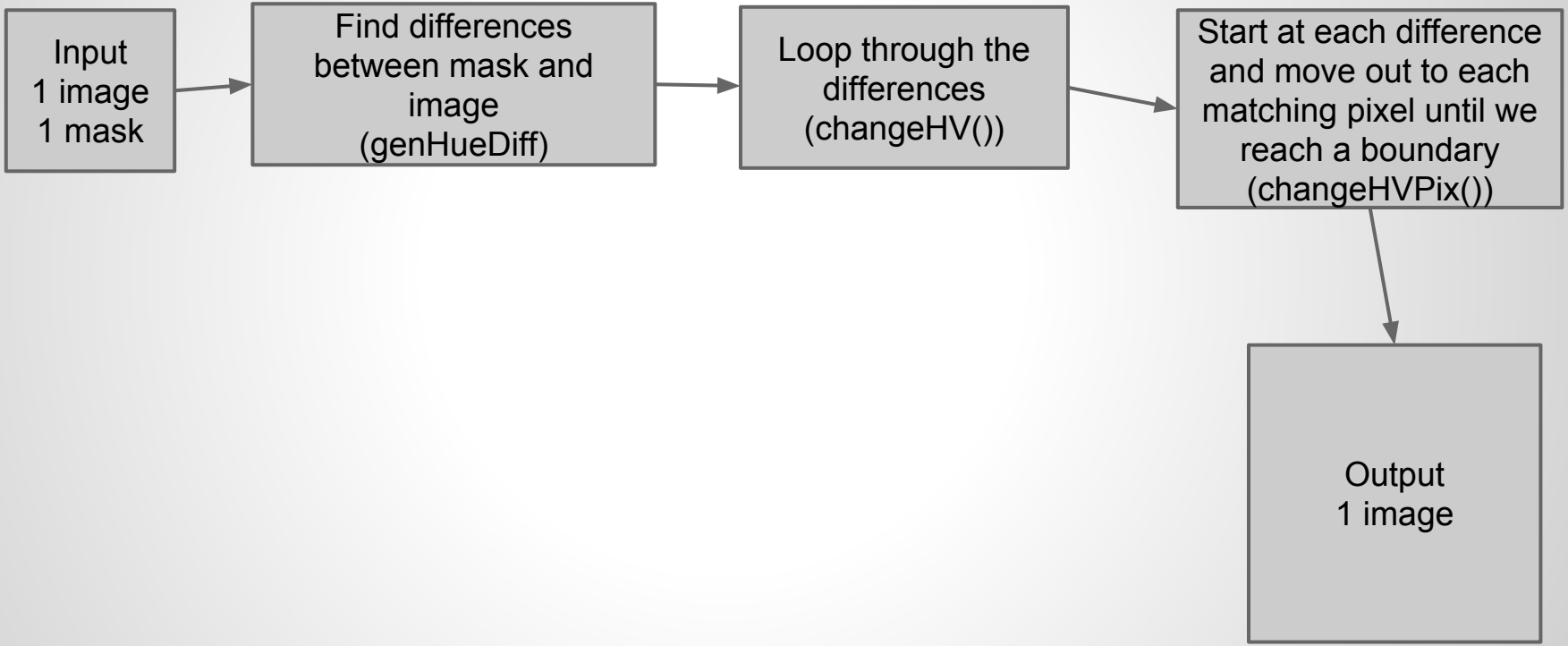
Positional Hue Change

Input

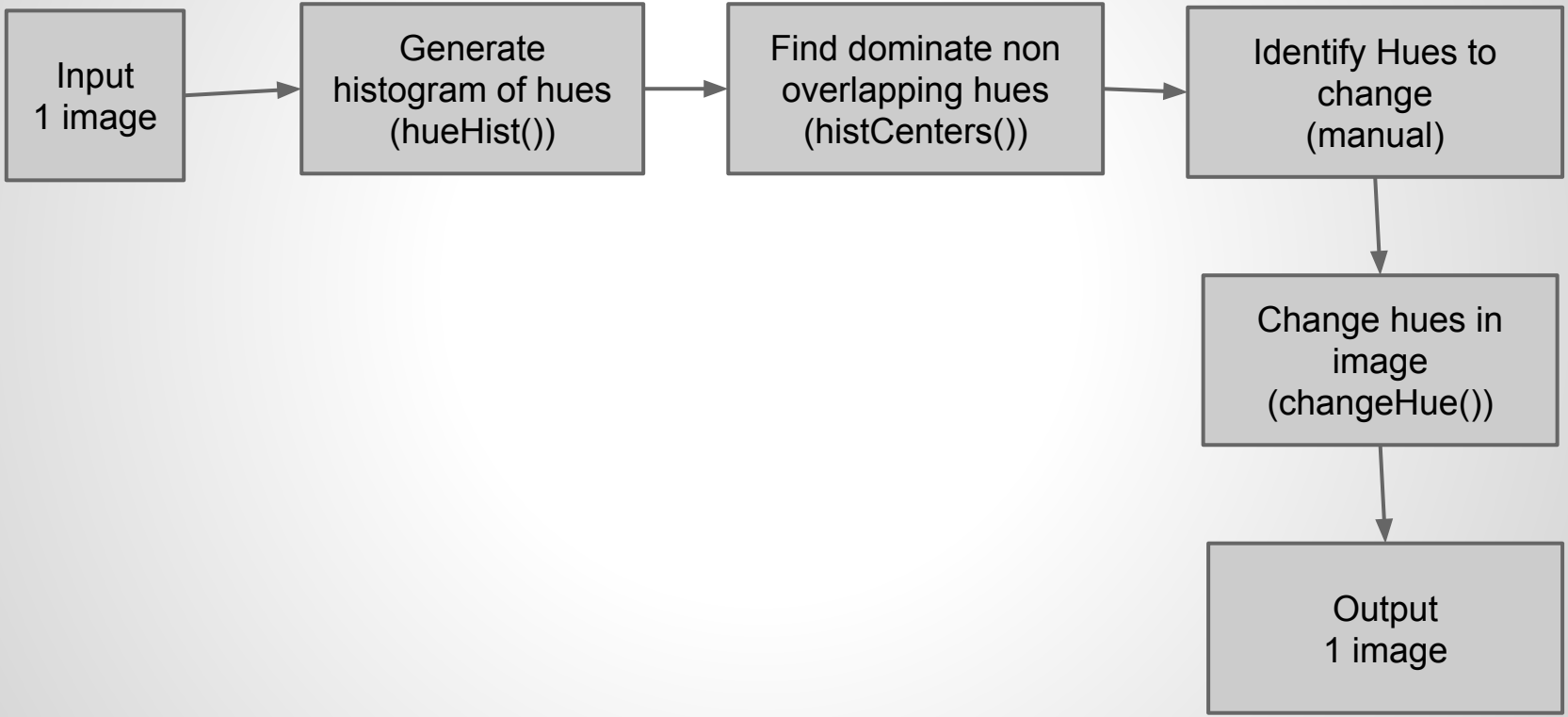


Output

Your Pipeline (Localized Hue Change)



Your Pipeline (Full Hue Change)

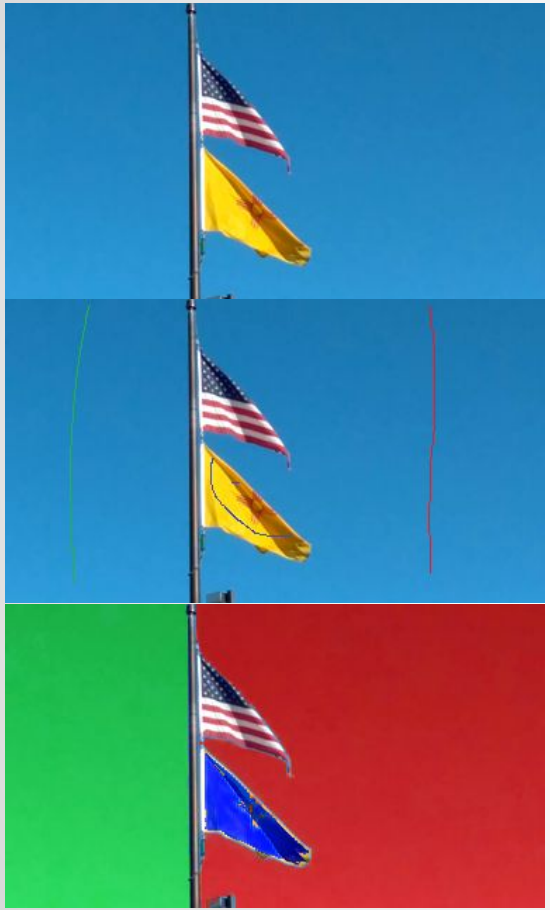


Demonstration: Result Sets (Localized Change)

Input

Mask

Output



Demonstration: Result Sets (Full Hue)

Input

Transform: [(9->99), (102-60)]

Output

Demonstration: Result Sets (Full Hue Mask)

Input

Mask

Output

There is a single purple pixel on the right half of the sky in the mask which allows the program to automatically use the transform (101->150)

Project Development

Use several slides for a detailed discussion of how you developed your project outputs. Tell your story. Difficulties with developing your code may also be discussed here, or in the following Computation section.

Include:

- Narrative of your progress and issues you faced.
- Include descriptions of problems, and how you handled them.
- Include both good and failed interim results (from various parts of your work and pipeline)
- Include a high level of details.
- What did you finish, what is not finished?
- What would you do differently?
- Use as many pages as necessary.

Project Development

The first function that I implemented was a function that found histograms of the hues in an input image. I called it hueHist(). I assumed hue was a value that ranged from 0-255 as all other values seem to in opencv. Therefore I thought there was a bug when I did not get hues above 180 in the histogram. Apparently (according to documentation) hues in opencv only go to 180.

I then started working on a function called histCenters(). This function takes in the full histogram and then tries to find non overlapping hues (of a specified width) in an output array with weights of how common they are. I mostly worked with a width of 18 because that gave me ~10 colors. If I used a wider value I could not identify as many colors and if I used a narrower value then it becomes difficult to replace a whole color later.

I wrote an auxiliary function called fil() which generates an array of size width that values the center higher than the edges. I can then apply this to each point in the hue histogram. From this avg histogram I found the strongest hues based on their neighbors of the specified width and sort the list before returning it in a list with its weight. For the sunset image this returns the following:

[(9, 263629.59259259258), (18, 144413.02160493823), (179, 123558.40740740742), (28, 35350.16358024691), (102, 23301.929012345681), (75, 20648.382716049386), (112, 18609.027777777774), (37, 18520.444444444442), (140, 15642.851851851849), (121, 14701.546296296296), (50, 10764.0)]

Project Development (cont'd)

From the output you can see the hues I used for the demonstration I wanted to target the red clouds 9 (the most common hue). I also wanted to change the blue sky. Since I knew pure blue is a hue of 120 I was able to guess that the sky was probably 102 based on the output.

The next step was to then write a function that could find the pixels of the specified hue and change them. I called this function changeHue(). The first version of this function took in the following (image, srcHue, dstHue, hueWidth). The hueWidth parameter specifies how far from the srcHue to match pixels. When the pixel is transformed its hue is the same offset of its original hue from srcHue. The goal of this was to keep the detail in the original image.

If I wanted to change multiple hues in the input image with this function I called it twice. However this made it impossible to swap two colors in the output. For example if I want to transform red clouds to blue and the blue sky to red, the first call would transform the red clouds to blue and the second transform would change the blue sky and blue clouds to red.

Project Development (cont’d)

To allow for many more transforms I change the function changeHue() to take a list of transforms to do. This way it can be sure to only transform each pixel once.

At this point I decided to add an ability to automatically find the transforms based on a mask image where every pixel that changes compared to the input image is converted to a hue transformation. It is important to note that this means you need to use a lossless format for the input and mask. From this you can simply mark the colors in the input image with one pixel of the color you would like them to be and run the program. This function is called genHueDiff().

I did find that although this method was more convenient than the histogram method the histogram method is still a bit better for things like the sunset image where you need to find the exact center of the hue you want to change rather than guessing.

Project Development (cont'd)

The next ability that I wanted to add was a way to limit the changes to a particular region. I striped the code from changeHue() and wrote a function called changeHVPix() that only works on single pixel and then if that pixel was changed recursively calls all the pixels around it.

This function did not actually work because the stack in Python is too small to deal with this much recursion so I had to rewrite the function to run using a work queue.

I had trouble limiting the changes to a particular region (colors like white, gray, and black do not have a reliable hue). To fix this I started filtering the edge of the region by saturation and value as well.

The last thing I did was try to transform the value of the dst pixel to match the transform. I was never able to make this look realistic so I eventually gave up and commented out the code.

Project Development (cont'd)

If I were to do this again the first thing I would do was do the functions on HSL rather than HSV. From what I have read this is closer to how the human eye sees and I think this might allow me to transform the image by more than just hue and also match color.

I also need to find a way to deal with what I think is the noise from jpeg images. In the image below I tried to switch the coat colors of the two people. The coat on the right transforms easily but the coat on the left has weird pixelation.

Computation: Code Functional Description

fil(width, alpha=0.5)

This function generates a numpy array of the width specified. The output array is highest in the middle and lowest on the edges. The sum is always 1. Alpha is a parameter ranging (0-1) where if you increase it it will more heavily weight the center. If it is zero then the output array all has the same value.

The purpose of this function is to provide a distribution to find the best average hue to use as the src for a hue transform.

hueHist(image)

This function produces a histogram of the hues in the image with a bucket of 1 for each hue (0-180). The function does this by just going through the whole picture and adding one to the value of the hue bucket for that pixel.

Computation: Code Functional Description

genHueDiff(image, mask)

This function finds each difference between the image and mask and then returns those differences in a list that has both hues, saturations, and values. It also includes the location of the difference. To do this the function scans both images pixel by pixel and adds the difference to a list that it eventually returns.

Computation: Code Functional Description

changeHVPos(hsvImg, imgMark, tp, pos, hwidth, swidth, vwidth, stack)

This function first checks if the pixel should be changed. Tp is a tuple of the transform to be performed. The widths refer to the maximum difference this pixel can be from the target pixel before the transform will not be performed. imgMark is a matrix that keeps track of the pixels that have already been changed. Stack is a work stack which neighbor pixels will be added to if the transform is performed (this function was originally recursive).

To do all this the function first check if the transform should be performed. If not the function returns. Then the function calculates the transform and performs it. Finally it adds its neighbors to the stack so that they will be checked.

Computation: Code Functional Description

changeHV(image, tPairs, hwidth=18, swidth=128, vwidth=128)

This function takes in a list of transforms in tPairs setups the boolean matrix to keep track of changed pixels and then calls changeHVPix() on each transform.

changeHue(image, tPairs, hwidth=18, vwidth=512)

This function takes in a list of transforms in tPairs. It ignores the positional data and instead just performs them globally on any pixel that meets the transform criteria. To do this it scans each pixel in the image and then sees if the hue of that pixel is within the hwidth of the src hue. If so it transforms it.

Computation: Code Functional Description

histCenters(hueHst, width=18)

This function takes in a histogram of hues and then tries to find the center of dominant hues in it. To do this it calculates a weighted average of each hue with the some of the hues around it of the specified width. Then it takes this average list and finds the best ones in that width. Finally is sorts the output by the dominance of that hue.

Resources

http://docs.opencv.org/trunk/df/d9d/tutorial_py_colorspaces.html

https://docs.python.org/3/tutorial/datastructures.html

http://docs.opencv.org/3.0-beta/modules/imgcodecs/doc/reading_and_writing_images.html

Appendix: Your Code

https://github.com/cadejager/cs6475_finalproject/blob/master/finalproject.py

Credits or Thanks

http://www.cs.princeton.edu/~huiwenc/pub/sig2015-palette.pdf

http://www.cs.huji.ac.il/~yweiss/Colorization/