# Project 1 Report

Chandler Smith
Date: January 29th

*Note: 2 Time travel days used.

***A short description of the overall project in your own words. (200 words or less)***

The purpose of this assignment was to gain a deep knowledge of how to read, engage with, and alter images. Tasks 1 & 2 followed the initial cv tutorials, which allowed us to become familiar with OpenCV documentation. Task 3 made us of premade OpenCV functions, specifically cvtColor, and was an important step in solidifying how useful the OpenCV package can be.

Tasks 4-9 were a series of custom functions that built upon one another, resulting in a cartoon version of a live video stream. Task 4, a custom grayscale equation, was our first foray into the creative options for achieving the same goal, a simple grayscale. Task 5 enabled a deep understanding of 1x kernels as we were tasked with creating a 5x5 Gaussian filter. Task 6, creating 3x1 separable Sobel filters, provided the foundation for the remaining tasks. Without working separable sobel filters, the remaining tasks would execute correctly. SobelX identified all of the horizontal edges, and SobelY isolated the vertical edges.Task 7 took those sobal filters and combined them into an image that represented gradient magnitude, reducing the image to highlighting lines. Task 8 blurred the image, and quantized it, providing the initial elements for the canonization of the video. The final cartoon video is a combination of our sobel functions, magnitude functions, blur-quantize functions, and a blacking out of pixels that hit a certain threshold in a gradient magnitude calculation.

Finally, task 10 was to be creative and implement some additional filters onto our video. Overall, our goal was to build the building blocks for a cartoon image from scratch, and then put them together to produce a cartoon video filter.

***Any required images or text along with a short description of the meaning of each image or diagram.***

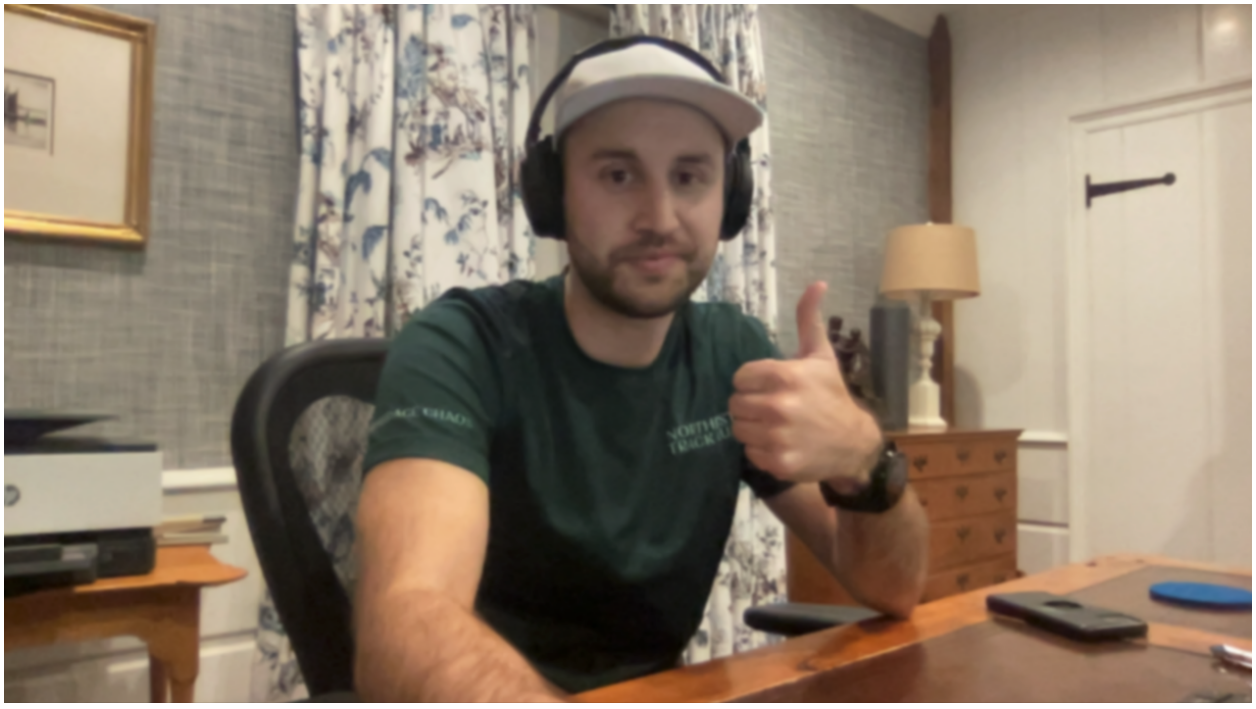
<u>Image 1: Grayscale via OpenCV function</u>

Grayscale image using the cvtColor openCV function. This function is designed to take an input image, run it through a color conversion, and produce a destination image based on that conversion. OpenCV outlines many color conversion in its documentation, and there are many different ways to get to grayscale. I chose the method cv::COLOR_RGB2GRAY which was one of these conversions. There are several ways to complete a conversion to gray; adding or removing and alpha channel, reversing the order of each channel, or converting to 16 bit RGB. My method converted RGB to gray by multiplying B by 0.114, G by0.587, and R by 0.299, resulting in grayscale picture.

Image 2: Custom Grayscale

While this is also a grayscale image, it is slightly different from the one above. Here, I did a custom calculation which divided each BGR value by three, iterating over each pixel.

Image 3 Gaussian as separable 1x5 filters

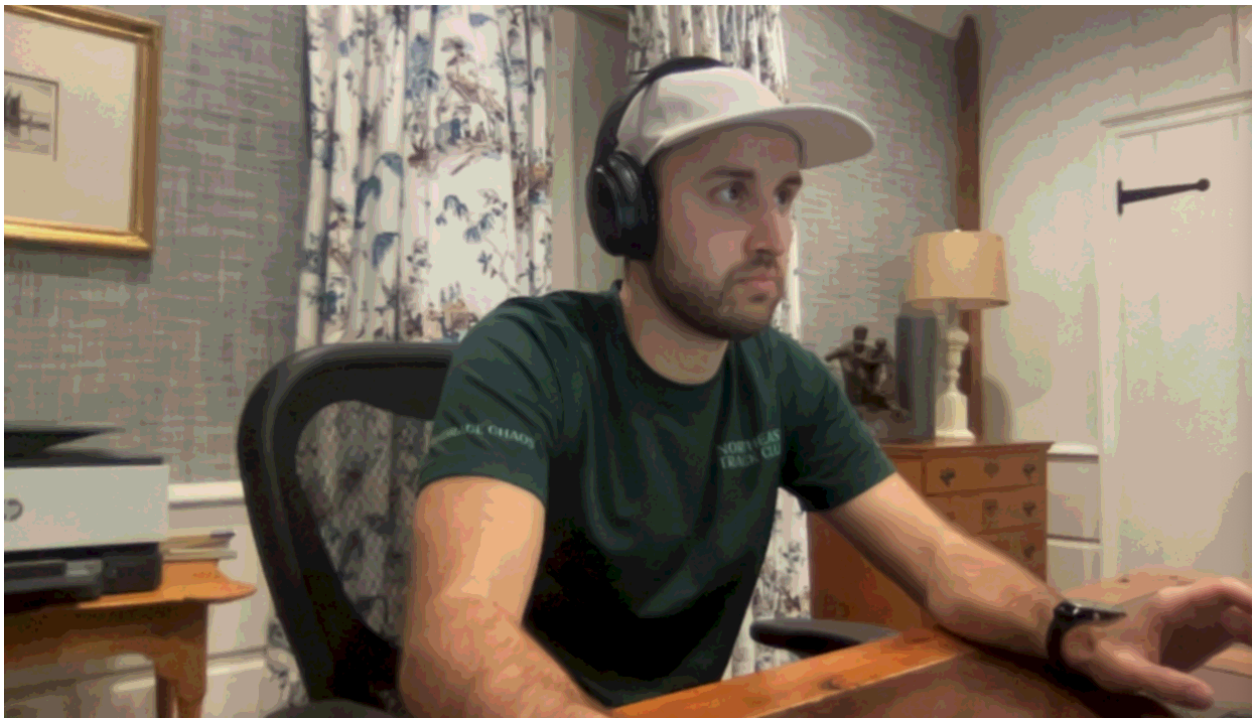


A blurred me! Here, I implemented a 5x5 Gaussian glitter which blurs an image. I did so by creating two, 5x1 kernels and used them to iterate over rows and columns. Sigma controls the high-frequency deduction, and is used to create the blur. Shoutout to Gopals help here!

Image 4: Original and Gradient Magnitude

Arguably the most challenging series in this project was the SobelX, SobelY, Magnitude functions. These were imperative to get exactly correct as they carry implications for the cartoon image down the line. SobelX and Sobely functions iterate over the image pixels horizontally and vertically, identifying lines. Together, they can combine to produce the gradient magnitude with the appropriate calculations applied. Here, we use the Euclidean distance to calculate magnitude: I = sqrt( sx*sx + sy*sy ).

Image 5: BlurQuantize vs original



The blurQuantize image is really where we start to see the cartoon image come together. After running this image through the original blur function, we then work to quantize is based on a number of input levels. The levels provided are used to determine how many buckets we will divide 255 into. Once the bucket size is determined, we run through each pixel, dividing it by the bucket. To finish, we multiplied our quantized image back by the bucket size, resulting in a more unified color scheme.

Image 6: The cartoon

Welcome to the cartoon me! This filter was achieved by using two matrices and have them work together. First, we reimplement the magnitude filter via our sobelx and sobely functions. From there, we run blurQuantize on a separate version of the image. Now we essentially have two matrices, one of which holds a blurQuantized version and the other, a gradient number. As a

parameter, we take in magThreshold do the following: for every pixel that has a gradient larger than our threshold, color it black. The result is the cartoon image you see above.

Image 8: Rainbow sparkles

For another filter option, I colored the lined rainbow! Well, not actually rainbow but I generated a random color for each pixel, which makes for some cool outlines and sparkles. Additionally, I added a flashing/color-changing boarder to make it look like a party!

**A description of and example images for any extensions.**

Extension 1: Cinema

I created a widescreen movie-style frame for a spiderman, Myles Morales effect! This involved reimplementing the cartoon functionality and ensuring the size and boarders are proportional.

Extension 2: Old time cinema

My goal here was to replicate the style and feel of an old school movie. To achieve this, I made the frame widescreen, added a custom border, and distorted the quality to give it a bit more of a grainy, older vibe.

Extension 3: Oil Painting

I don’t know how to paint so I figured I would write a program to produce an oil painting of me! I utilized an opencv function from xphoto package to achieve the base effect while also adding small details like a boarder. I think this effect is really cool!

Extension 4: Laplacian

After grayscaling the image, I applied the laplacian filter, which then needed to be converted to scale. The result is a lovely, boardered laplacian filter video. I really like how the text pops here and how fast the video is.

Extension 5: Facial recognition

I couldn't quite get this to work but I can't find any bugs in the code so perhaps it will work for you!

**A short reflection of what you learned.**

This project was a first for a lot of reasons. First time using C++, Xcode, OpenCV, and makefiles. There was a huge learning curve when setting up my environment, and I am glad that that is established. Furthermore, I learned an immense amount about pixel manipulation. I started seeing images as matrices rather than images and further separated them into columns and rows to be iterated over independently. The iron-clad reasoning and way of seeing images was certainly the biggest takeaway from this assignment and building filters from scratch. I also understand filters in a much more robust way.

In terms of C++, the biggest learning curve was with data types. I continually failed to track the datatypes of the images I was passing in vs what I needed for manipulation. An example of this was the use of Vec3b vs Vec3s in the sobel functions. Huge shoutout to Gopal, Ravina and other TAs for helping me process and learn this.

**Acknowledgment of any materials or people you consulted for the assignment.**

I want to acknowledge the fantastic support of the course TAs: Gopal, Heet, Ravina, Santosh, and Mrudula. I also wanted to cite the following materials:

- OpenCV documentation and tutorials
- Wikipedia Sobol Operator
- Piazza questions and responses
- Real time video abstraction paper
- Lecture and lecture notes from Bruce Maxwell
- CV Mat datatypes page
- Adarsh Menon Youtube (face recognition)
- ChatGPT (troubleshooting only - not actually helpful)