Adithya Palle
CS 5330 Project 1

Special Video Effects

Overview

This project delves into various image manipulation algorithms, showcasing their application in a real-time video display system. The work covers pixel transformations, area filters, separable filters, and quantization, as well as advanced machine learning techniques such as Haar Cascade facial detection and depth estimation using DepthAnything. Additionally, effects like brightness adjustment, image denoising, and fog simulation were implemented to enhance visual processing. As an extension, I explored more intricate image manipulation by applying swirl distortion to the detected facial regions within the images.

These experiments highlight a range of image processing methods that form the foundation of computer vision. They demonstrate the effectiveness of simple filtering and pixel-based techniques in a technology landscape increasingly dominated by neural networks.

Greyscale Weighting

When converting a standard RGB image into grayscale, OpenCV performs a weighted sum of the values from each color channel to a single value between 0 and 255. Per the docs, the weights are 0.299 for red, 0.587 for green, and 0.114 for blue. It's important to note that grayscaling is a lossy conversion as multiple different combinations of RGB values can result in the same gray value due to the nature of addition.

Adithya Palle
CS 5330 Project 1

Custom Greyscale

For the customized grayscale, I decided to generate the gray value by summing up the values for all RGB channels and then modding the sum by 256. This method is worse than the original weighted sum method for grayscaling because the brightness in the output image does not have a linear relationship with bright colors in the original image. This is because the modulus operation causes the values to “loop” back to 0 if the sum is too large. For example, a full red image with (255,1,0) would have become completely black with a 0 value at every pixel where as a lighter red would actually be brighter in the grayscale image. While this is not a feasible grayscaling method, it’s interesting to see how the modulus operation creates “randomness” in the brightness of the grayscale image.

Sepia Filter

To ensure only the original values were used to compute the new values from the sepia filter, I stored the original RGB values in integer variables and computed the new values using this original integer values only. I computed all three new values into integers before setting the pixel values in the destination matrix. This assures that all 3 new values are only calculated with the original RGB values and not the modified one.

Adithya Palle
CS 5330 Project 1

Blurring

The first implementation of the blur filter naively applied NxN multiplications at each valid pixel and computed their sum to place in the destination image. Furthermore, cv::Mat.at was used for all accesses leading to redundant accesses of the row pointer for the same row. The second implementation was optimized to retrieve a pointer to each row of pixels only once per row and then access the values at each row using that row pointer. Furthermore, I split up the gaussian kernel into a separable [1,2,4,2,1] filter and then applied this filter horizontally in one pass to mutate the copy of the source image and then applied it again vertical on the updated copy to create the same convolution output but with only 2N operations per pixel. As a result, the first version of this filtering algorithm ran at an average of 0.101 seconds over 100 blurs on the same 720x1080 image, and the optimized version ran at an average of 0.023 seconds with the same configuration. This means that these optimizations improved the average speed by **4 times.**

Adithya Palle
CS 5330 Project 1

Gradient Magnitude of Sobel X and Sobel Y

Below is an example of the gradient magnitude image resulting from taking the combined magnitudes of the Sobel X and Sobel Y filter outputs.

Adithya Palle
CS 5330 Project 1

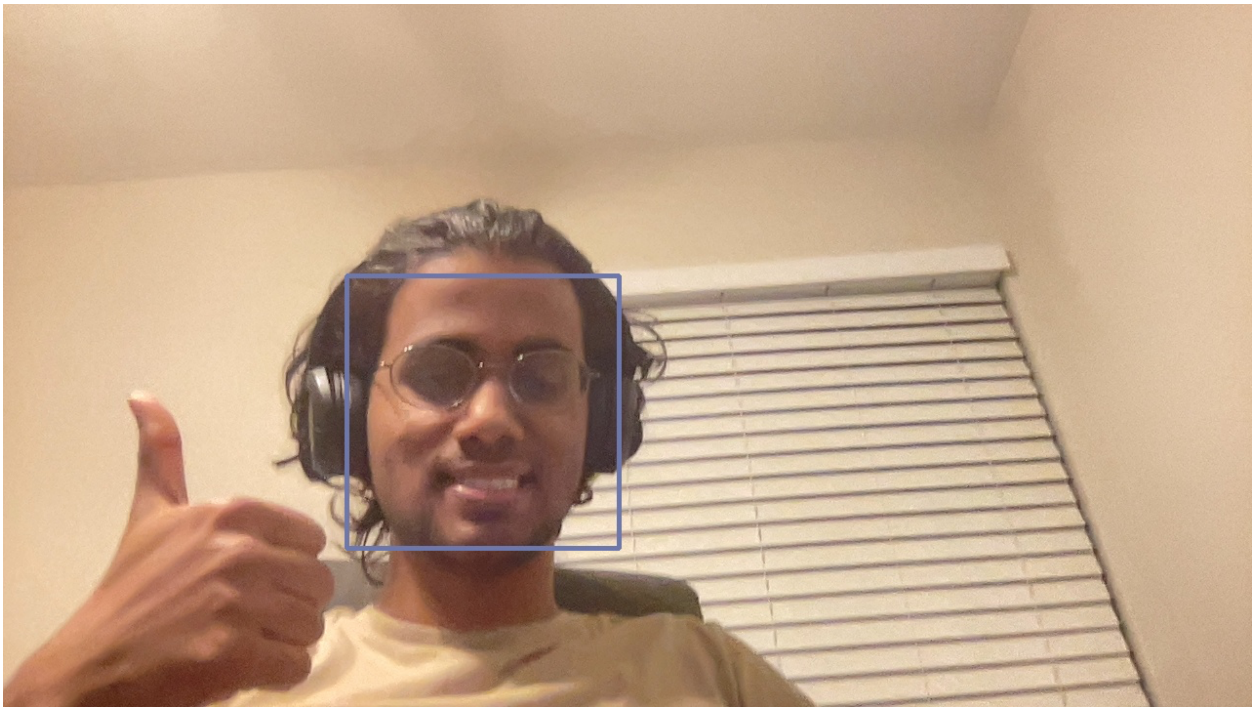
Blurring and Quantization

Below is the output of the blurQuantize function which applies a Gaussian blur and them quantizes the image into 10 buckets.

Adithya Palle
CS 5330 Project 1

Face Detection

Below is an image of a frontal face detector using a pretrained haar cascade model.



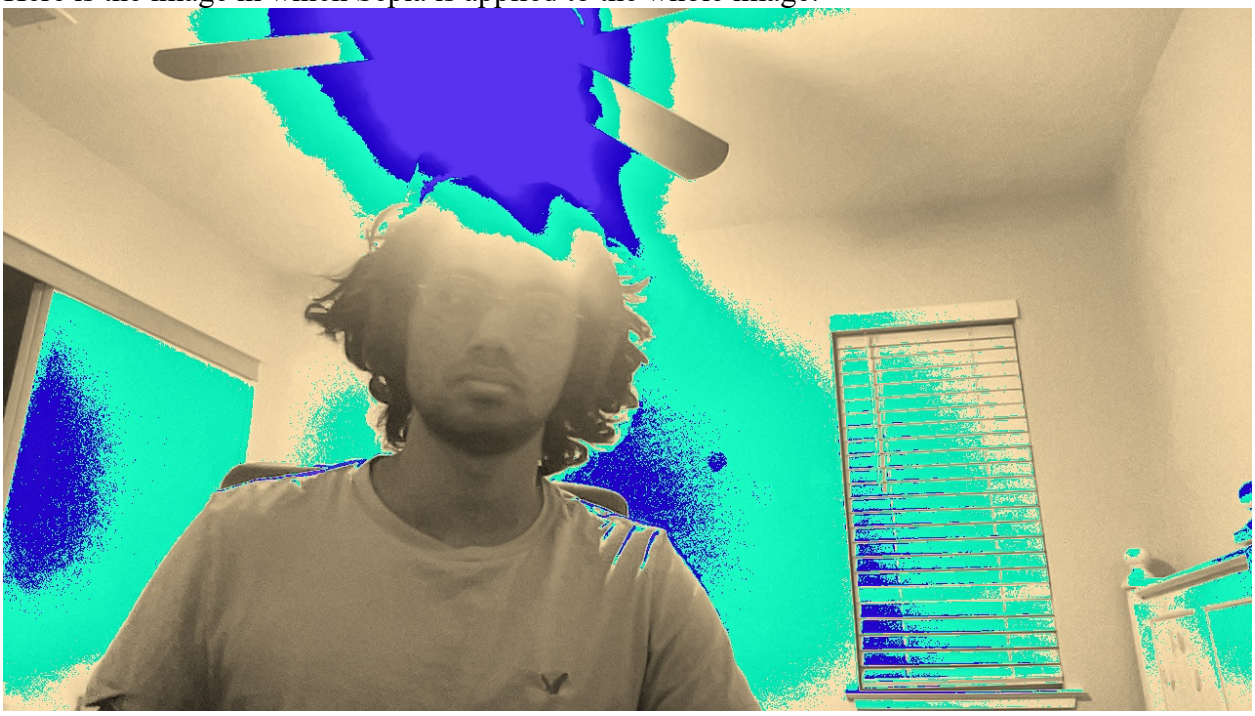
Depth Anything (Required Images 7-8)

Below is a depth image , in which brighter orange colors represent closer objects.

Adithya Palle
CS 5330 Project 1

With this depth information, I decided to implement a function that applies the sepia affect to the foreground of the image. It operates by taking the grayscale depth image produced by DepthAnything and restoring pixels in the sepia image that have a depth value less than a specific threshold. The threshold I found effective was 128 . Here is a resultant image:

Here is the image in which Sepia is applied to the whole image:



Adithya Palle
CS 5330 Project 1

Clearly the background is preserved in the first image as our depth detector deems its depth as being too far to be a foreground element.

Additional Effects

Adjust Brightness of Image

I decided to add a feature where users could hit the “+” or “-“ keys to increase or reduce the brightness of the image respectively. This works by adding or subtracting a value of 10 to each channel of the original image, clamping it in the range of [0,255]. Below are the resulting darkened (left), regular (middle), and brightened (right) image.

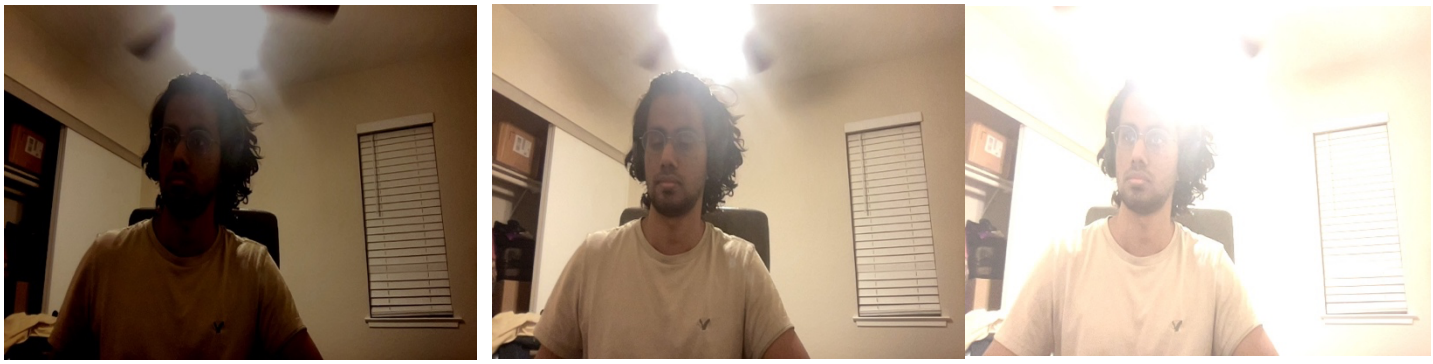
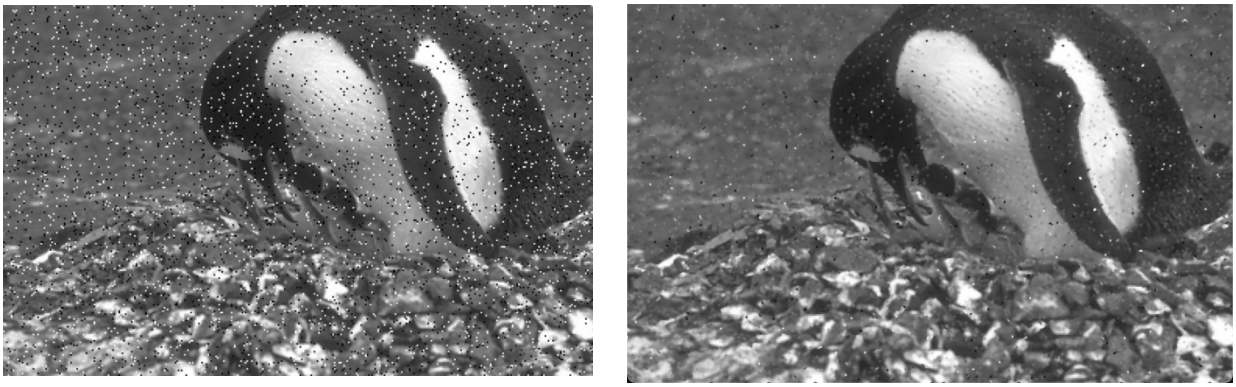


Image Denoising

In order to eliminate noise such as salt and pepper noise in the image, I implemented a median filter using a 3x3 kernel. I used an O(N lg N) sorting algorithm to compute the median of the neighborhood. It can be enabled in the video application by pressing the ‘r’ key. Below is the result of applying the filter to a noisy image.



Not all the noise is removed due to the small kernel size, but I decided to keep it small to prevent and unnecessary smoothing of the image.

Adithya Palle
CS 5330 Project 1

Image Fogging

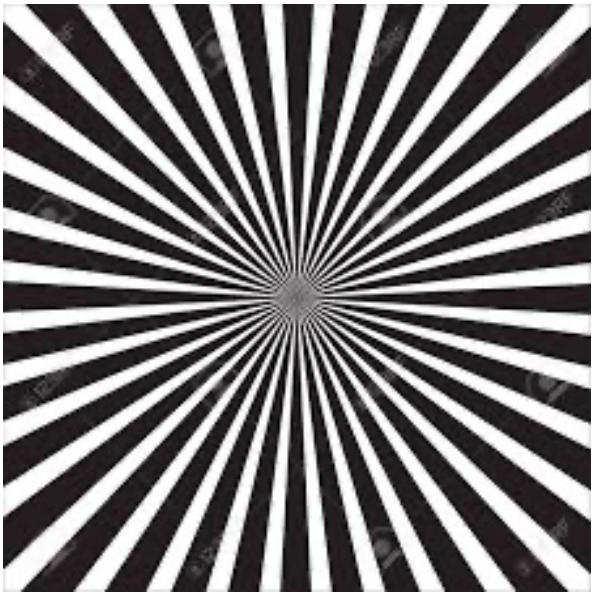
I created a depth-based fogging effect by modifying each pixel value to be a weighted sum of a fog coefficient (F) and the original image color. The Fog coefficient was calculated using and exponential relationship with the depth value of the image, so further objects were foggier than closer ones. The exact formulas are as follows:

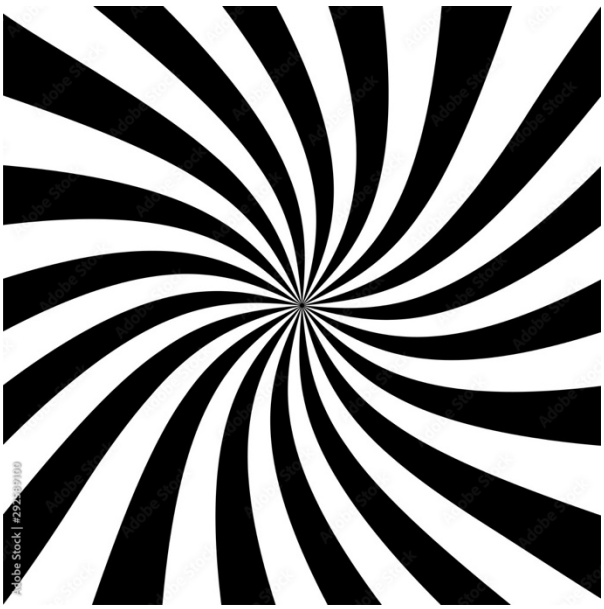
$$F = 1 - e^{-k*d}$$

$$\text{Output Color} = (1 - F) * \text{Image Color} + F * \text{Fog Color}$$

Here, d represents the normalized fog value, where values closer to 1 represent deeper objects. K is the fog density coefficient which can be increased to provide more dense fog. Below is the result of an image with the fogging effect applied with K = 4 and using (128,128,128) (Gray) as the fog color.

Extension – Facial Swirl Distortion





Adithya Palle
CS 5330 Project 1

For my extension I decided to make a swirling distortion effect in the detected facial region of the video stream, similar to those found on popular facial filter apps such as Macbook's Photo Booth or Snapchat. This essentially applies a rotation to the pixels in the facial region using the spherical coordinate system. The rotation is dependent on the distance from the center of the facial region, with points closer to the center being rotated more than points further from the center based on an exponential decay. The exact equations for determining the rotated pixel position from the origin pixel are as follows:

Original Coordinates:

$$i, j = \text{row, column of pixel}$$
$$F_x = \text{center of width face rectangle}$$
$$F_y = \text{center of height face rectangle}$$
$$R_{face} = \text{radius of min enclosing circle of face rectangle}$$
$$x = j - F_x$$
$$y = i - F_y$$
$$r = \sqrt{x^2 + y^2}$$
$$\theta = \tan^{-1}\left(\frac{y}{x}\right)$$

Rotated Coordinates:

$$\theta' = \theta + \pi * e^{-\frac{r}{R_{face}} * S}$$
$$x' = r\cos(\theta') + F_x$$
$$y' = r\sin\theta' + F_y$$

When implementing this algorithm, the pixel values were calculated by applying the reverse of the above rotation. Each pixel in the output image was then updated based on the corresponding pixel value obtained from the inverted rotation. This approach, known as inverse mapping, ensures that there are no holes in the distortion where certain pixels might otherwise be left without updates. Below is the result of the swirl distortion on the face.

Adithya Palle
CS 5330 Project 1

The swirl affect can be applied in the video display application by pressing the ‘n’ key.

Reflection

This project significantly enhanced my understanding of images and how they can be manipulated to create the effects we see in many modern applications. Starting from manipulating individual pixels to applying convolution and more advanced area filtering methods to create visually appealing effects was a great experience that strengthened my intuition about filtering. I learned that I can better understand the impact of a filter by mentally applying it to a small region of the image and visualizing how the output would look (such as with edge detectors). This approach has made me feel more confident and equipped to design custom filters on my own.

I now understand that images are essentially 3-dimensional matrices, and simple mathematical operations can be used to transform them. Additionally, my C++ coding and debugging skills improved significantly throughout this project. Previously, I hadn’t focused much on performance in my code, but seeing how slow algorithms can drastically impact the performance of real-time video processing made me much more conscious of writing efficient and effective code.

My favorite part of the project was the extension. Growing up, I was always fascinated by the weird facial distortions I could create on my iPad. Being able to implement those effects myself using simple trigonometry was incredibly exciting. I couldn’t find many resources online, apart

from a StackOverflow post, so I had to derive most of the math and design the algorithm on my own. Successfully executing this was deeply satisfying.

Overall, this project solidified my interest in computer vision. I thoroughly enjoyed visualizing the outputs of my algorithms and exploring the intersection of mathematics and creativity.

Acknowledgements

I'd like to thank Professor Bruce Maxwell for instruction on these topics prior to me conducting this project, it was very insightful in implementing many of the algorithms described. All other consulted sources are listed in the section below.

Sources

General Knowledge - Computer Vision: Algorithms and Applications, 2nd ed., Richard Szeliski

Sobel filters - https://en.wikipedia.org/wiki/Sobel_operator

Swirl Effect - https://stackoverflow.com/questions/225548/resources-for-image-distortion-algorithms

OpenCV Documentation - https://docs.opencv.org/