CS 5330 - Computer Vision

Project 1 Report - Yanting Lai

1. Project Overview

This project focuses on implementing various image manipulation techniques and applying them to both static images and live video streams using OpenCV. The tasks range from simple grayscale transformations to more advanced filters like sepia tone, blur, and Sobel edge detection. Additionally, the project explores extensions such as cartoonization to create a more dynamic video-processing experience. The goal is to apply filters interactively in real-time video streams, enabling the user to switch between different effects by pressing corresponding keys. Each task demonstrates an understanding of how images can be manipulated using pixel-level operations and built-in OpenCV functions, while optimizing performance for video display.

2. Required Images and Descriptions

Image 1: Original and OpenCV Grayscale

- **Description**: The original video frame is captured and converted to grayscale using OpenCV's `cvtColor` function. This transformation converts the image using weighted averages of the color channels (BGR to grayscale conversion).
- **Keypress**: 'g' to toggle OpenCV grayscale.
- **Difference**: This grayscale conversion maintains the image's original brightness and contrast while eliminating color information.

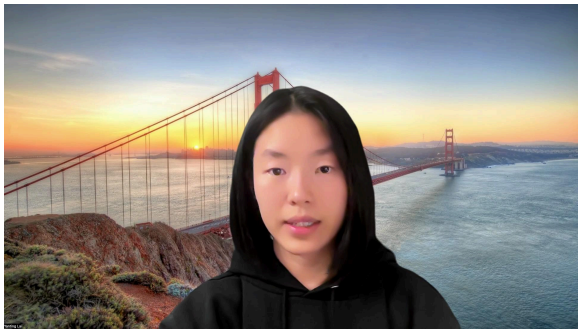


Image 2: Customized Grayscale

- **Description**: This custom grayscale filter subtracts the red channel from 255 and applies it uniformly across all three channels (BGR). This results in a unique grayscale effect where higher red intensities are darkened.
- **Keypress**: 'h' to toggle custom grayscale.
- **Difference**: Unlike the OpenCV grayscale, this method creates an inverted effect where areas with high red intensity appear darker in grayscale.

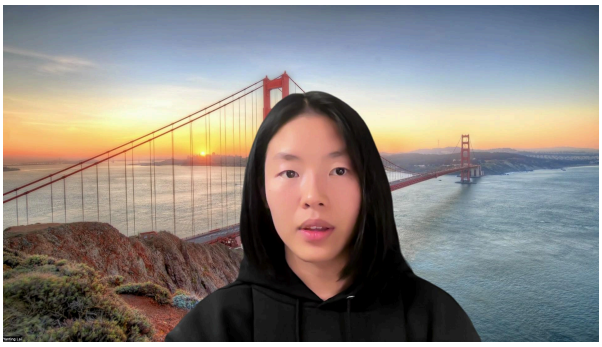
<table>
<tr>
<td>  </td>
<td>  </td>
</tr>
</table>

Image 3: Sepia Tone Filter

- **Description**: The sepia tone filter uses matrix multiplication to adjust the RGB values and produce a vintage, sepia effect. Each new pixel value is calculated from the original pixel values and checked to ensure no overflow occurs beyond 255.
- **Keypress**: 's' to toggle custom sepia filter on imgDisplay.cpp file.
- **Vignette Effect**: Additionally, a vignette is added to the edges of the image, simulating a fading effect typical of antique photos.
- **Process**: The R, G, B values are adjusted as follows:
    - Blue: 0.272*R* + *0.534*G + 0.131*B
    - Green: 0.349*R* + *0.686*G + 0.168*B
    - Red: 0.393*R* + *0.769*G + 0.189*B

<table>
<tr>
<td>  </td>
<td>  </td>
</tr>
</table>

Image 4: Original and 5x5 Blur (Naive)

- **Description**: The first blur filter (naive implementation) uses a 5x5 Gaussian kernel. The image is processed using nested loops to blur each color channel separately.
- **Keypress**: 'p' for 5x5 naive blur.
- **Timing Information**: This implementation takes longer due to direct pixel access and looping.
- **Result**: Smoother edges, but slower performance.

Optimized 5x5 Blur

- **Description**: The optimized blur filter uses separable 1x5 filters to improve performance. This method first applies a horizontal pass and then a vertical pass, significantly reducing the computational load.
- **Keypress**: 'b' for optimized blur.
- **Timing Information**: This version is faster than the naive implementation due to separable filters.
- **Result**: Similar blurring effect with improved performance.

Time Improvement analysis:

The separable filter method in **blur5x5_2** shows a clear improvement in performance compared to **blur5x5_1**. By splitting the 5x5 kernel into two 1x5 kernels, **blur5x5_2** significantly reduces processing time while maintaining the blur effect.

Filter	Average Time (seconds)
blur5x5_1	0.20 - 0.21
blur5x5_2	0.10 - 0.11

```
(base) sundri@YANTINGs-MacBook-Air build % ./imgDisplay imgDisplay
blur5x5_1 Time: 0.203642 seconds
blur5x5_1 Time: 0.208438 seconds
blur5x5_1 Time: 0.210444 seconds
blur5x5_2 Time: 0.117471 seconds
blur5x5_2 Time: 0.105063 seconds
blur5x5_1 Time: 0.214997 seconds
blur5x5_2 Time: 0.116159 seconds
blur5x5_2 Time: 0.105289 seconds
blur5x5_2 Time: 0.113677 seconds
```

Image 5.1: Sobel X and Y Edge Detection

- **Description**: The Sobel X and Y filters detect vertical and horizontal edges, respectively. Sobel X highlights vertical edges, while Sobel Y focuses on horizontal edges. The result is shown using `convertScaleAbs` to normalize the signed short output.
- **Keypress**: 'x' for Sobel X and 'y' for Sobel Y.
- **Result**: Clear edge detection that highlights boundaries in the image.

Image 5.1.1: Sobel X Filter applied

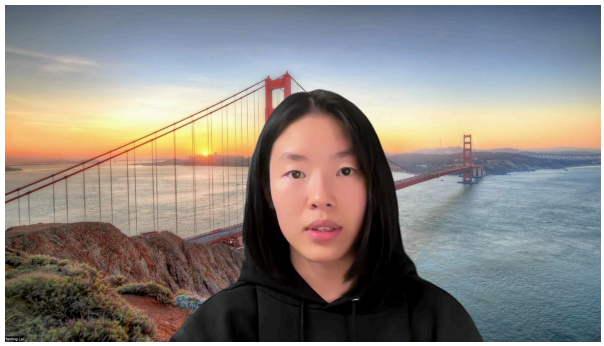
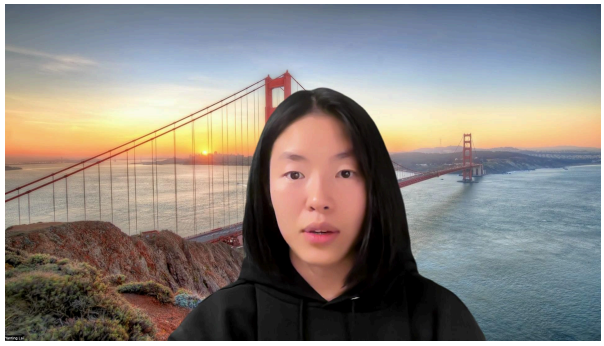


Image 5.1.2: Sobel Y Filter applied



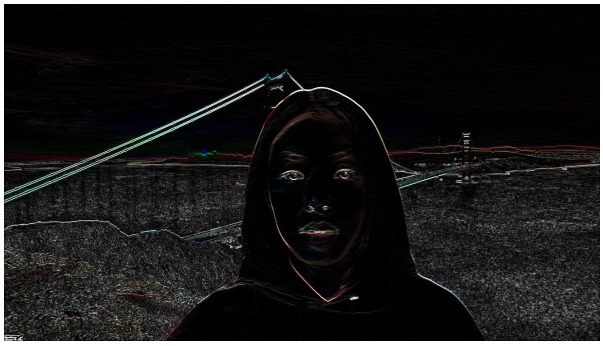
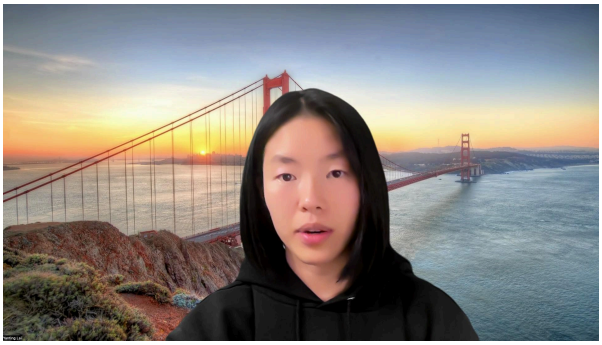


Image 5.2: Gradient Magnitude

- **Description**: The gradient magnitude is computed from the Sobel X and Y outputs using the Euclidean distance formula. This filter combines both horizontal and vertical edges into a single image.
- **Keypress**: 'm' for magnitude.
- **Result**: Enhanced edges, capturing both horizontal and vertical directions.



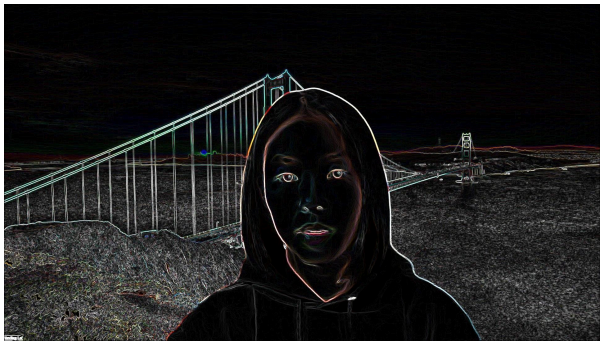
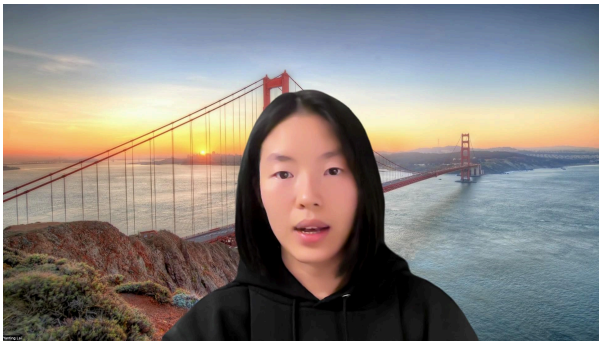


Image 6: Face Detection

- **Description**: The face detection feature uses a pre-trained Haar Cascade classifier to detect faces within the video stream. Detected faces are highlighted with boxes around them.
- **Keypress**: 'f' for face detection.
- **Result**: Accurate face detection with real-time performance.



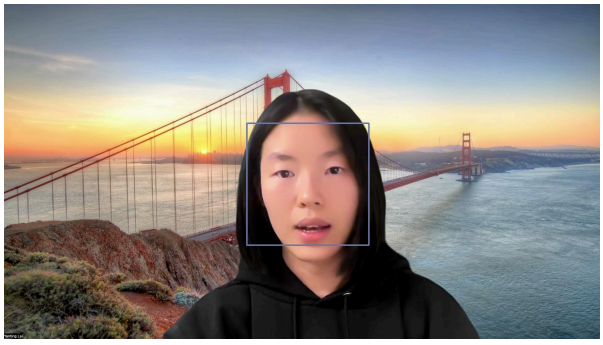
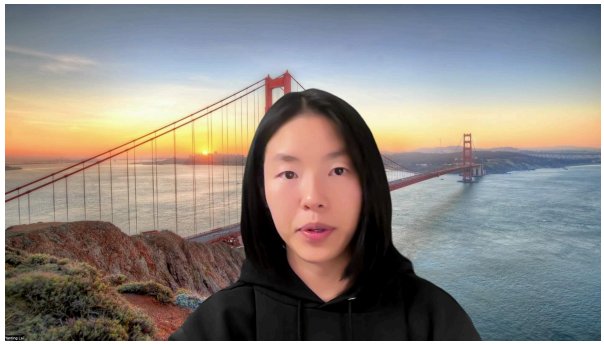


Image 7: Blur and Quantization

- **Description**: The image is first blurred, and then each color channel is quantized into a fixed number of levels. This reduces the color palette while maintaining a smooth appearance.
- **Keypress**: 'l' for blur and quantization.
- **Result**: A cartoon-like effect with reduced color precision.



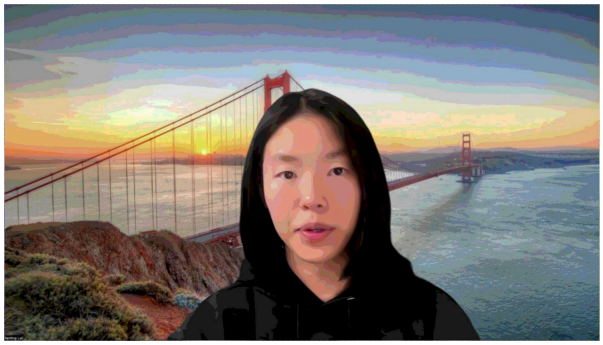


Image 8: Embossing Filter

- **Description:** The embossing effect uses the Sobel X and Sobel Y signed outputs and takes a dot product with a selected direction (e.g., (0.7071, 0.7071)) to generate an embossed look. The image appears raised, with lighting coming from a particular direction.
- **Keypress:** 'e' for embossing filter.
- **Result:** The image has a 3D embossed effect with emphasized edges.

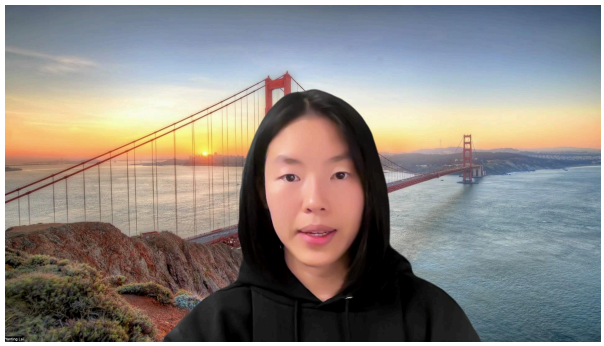


Image 9: Negative Filter

- **Description:** The negative filter inverts the colors of the image. Each pixel's RGB values are subtracted from 255 to create the negative effect.
- **Keypress:** 'n' for negative filter.
- **Result:** A striking negative image where bright areas become dark and vice versa.

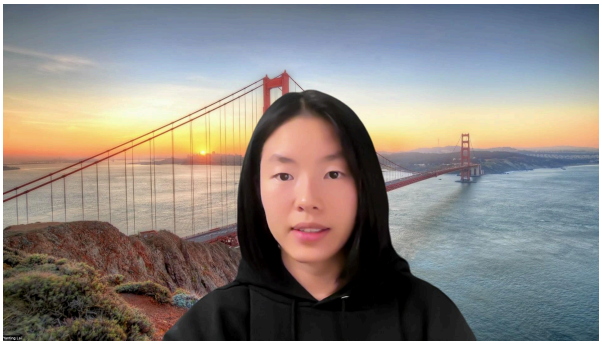
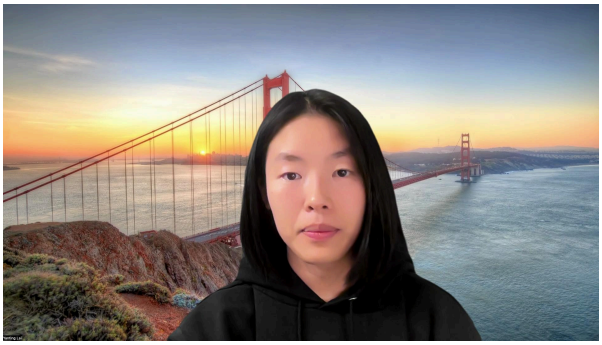
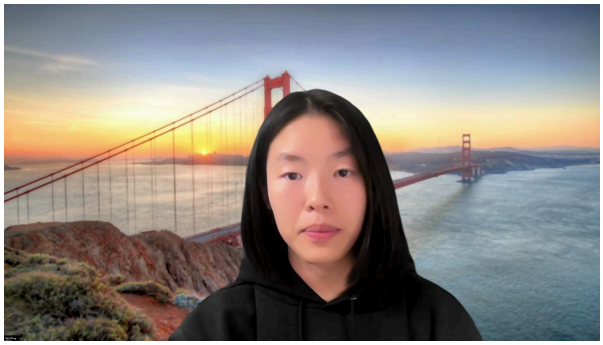


Image 10: Median Filter

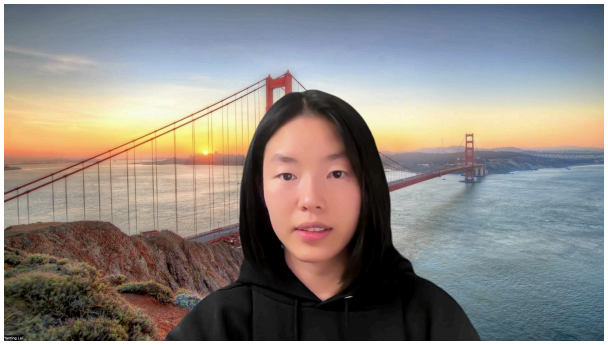
- **Description**: The **median filter** is applied to reduce noise, particularly **salt-and-pepper noise**, while preserving the edges of the image. Unlike other smoothing filters, the median filter replaces each pixel with the median value of the surrounding pixels, which helps in removing noise while maintaining the structure of the image.
- **Keypress**: Press 'k' to toggle the **median filter**.
- **Result**: Noticeably cleaner and smoother image with noise removed, and the edges are well-preserved, without the blurring effect common to Gaussian or mean filters.





3. Extension: Cartoonization

- **Description**: The cartoonization effect combines bilateral filtering, edge detection, and color quantization. Bilateral filtering smooths the image while preserving edges, and the quantization reduces the color palette. Edges are emphasized by overlaying Sobel edge detection results.
- **Keypress**: 'c' for cartoonization.
- **Result**: The image appears cartoonish with simplified colors and pronounced edges.




4. Reflection

In this project, I learned how to manipulate images and video streams using pixel-level operations, filter design, and optimization techniques. Implementing custom filters, such as sepia tone and cartoonization, gave me insight into the intricacies of color manipulation and the importance of performance considerations when processing live video. I also deepened my understanding of how separable filters can drastically improve performance. Overall, I've gained a deeper understanding about filters and how they are being implemented.

Things to improve: 1) Cartoon function should reduce to one object in a frame 2) figure out ways to make the blur filter faster in a video 3) Start the project early, coz it's a lot!

5. Acknowledgment

I would like to acknowledge the following resources I've used for the project

- ChatGPT for constructing the report skeleton and code structure as well as debugging
- OpenCV tutorials for basic functions
- Youtube videos for filtering concepts
- Haar Cascade XML for face detection provided by OpenCV