Project 1 Report

Name: Xujia Qin

Description:

This assignment involves using C++ and OpenCV to perform image processing tasks such as opening, capturing, manipulating, and saving images. It includes features like displaying images and live video, applying filters (grayscale, custom grayscale, sepia tone, blur, etc.), and measuring specific timing performance. A new component introduces the Depth Anything V2 Deep network with ONNX Runtime for depth estimation. Additionally, three filters will be implemented both from scratch and utilizing the existing OpenCV library as part of the extensions.

Required image 1

Implement a feature to switch the video to grayscale when the 'g' key is pressed, using OpenCV's built-in cvtColor function.

Required image 2

To generate the grayscale version, I used a color inversion method rather than the typical weighted average of the RGB channels.

For each pixel: Red, Green, and Blue channels are inverted: This function converts an image to grayscale using the standard weighted formula: (Y = 0.299R + 0.587G + 0.114B).

Differences from Default Grayscale:

The applyGrayscale method uses OpenCV's optimized cv::cvtColor function to convert the image in place to a single-channel grayscale (CV_8UC1). In contrast, the greyscale method manually applies a pixel-by-pixel conversion formula, preserving the 3-channel structure but with lower efficiency. The former is faster and more memory-efficient, while the latter provides more flexibility at the cost of higher memory usage and slower performance.

Required image 3

The applySepiaTone function applies a sepia effect to an image by iterating through each pixel and transforming its RGB values using a corresponding RGB coeffect that creates a warm, vintage tone. The new pixel values are then clamped to a max of 255 to ensure valid intensity levels.

Required Image 4: Implement blur filter

In timeBlur.cpp file, the blur5x5_1 function applies a Gaussian filter to the image using a 5x5 kernel, while the blur5x5_2 function applies a separable 5x5 blur using two 1D kernels for horizontal and vertical passes.

a. Time per image (1): 0.2296 seconds
b. Time per image (2): 0.1381 seconds

```
Time per image (1): 0.2296 seconds
Time per image (2): 0.1381 seconds
Terminating
```

Why the 2nd method it's faster:

I use separable filter 1*5 and 5*1 matrix to implement the blurring, so that it can reduce the operations compared with the 5 *5 Gaussian blurring.

From (25 + 24 + 1) = 50 ops per pixel reduce to (5 + 4 + 1) * 2 = 20 ops per pixel.

Task 7: implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters

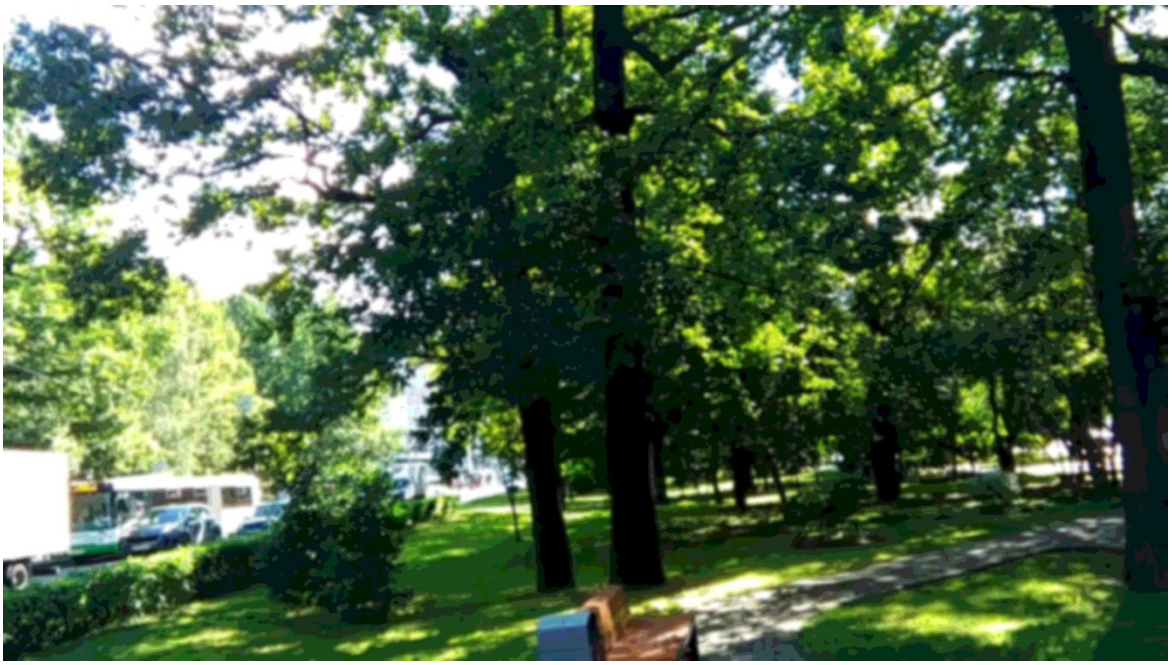
In filter.cpp file, the sobelX3x3 and sobelY3x3 functions perform Sobel edge detection in the X and Y directions respectively, using a 3x3 kernel. Both functions verify that the input image is in RGB format, create an empty destination image, and loop through each pixel to apply the respective Sobel kernel. These functions calculate the gradient along the horizontal (X) and vertical (Y) axes, which can then be combined to highlight edges in the image.

Required Image 4: gradient magnitude image from the X and Y Sobel images

In filter.cpp, the magnitude function computes the gradient magnitude for each pixel by combining the X and Y Sobel gradients. It first verifies that the input images are of type CV_16SC3 and have matching sizes. Then, for each pixel the gradient magnitude is calculated for all three-color channels and stored in the output image, which has the type CV_8UC3. The output generates the edge detected of the input.

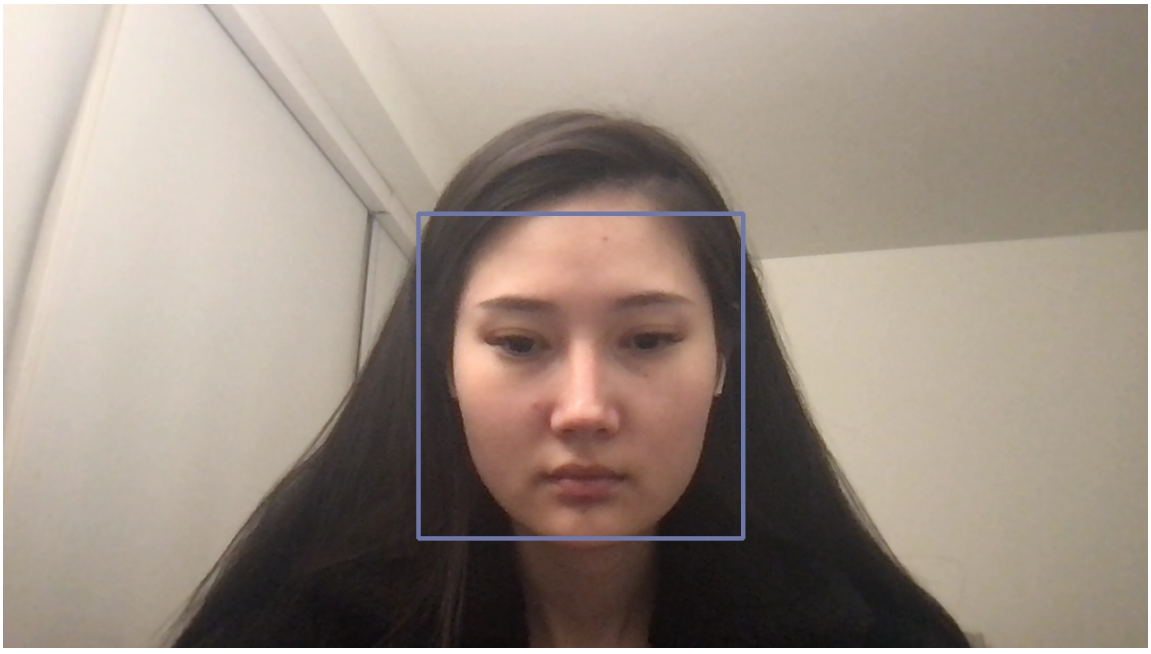
Required Image 5: blurs and quantizes a color image

The blurQuantize function in filter.cpp first applies a 15 * 15 Gaussian blur to the input image and then quantizes the color channels of each pixel based on the the default leve value as 10. It ensures that the input image has 3 RGB channels performs the blurring using the Gaussian kernel, and quantizes each color channel by rounding the pixel value to the closest quantization level.



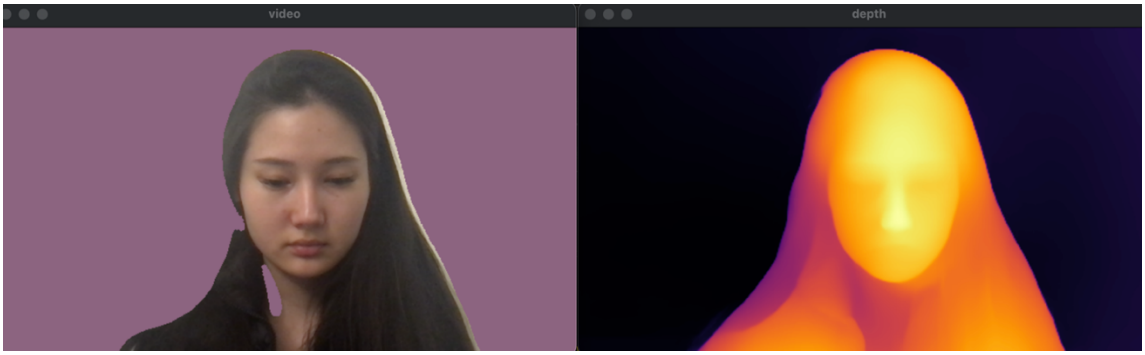
Required Image 6: face detected

With integrating the detectFaces function, we uses a Haar cascade classifier to detect faces in a grayscale image and stores the results in a vector of rectangles (size adjusted back to the full size). Then, the drawBoxes function draws rectangles around the detected faces, with frame, face, wcolor(RGB), and # of pixels defined.



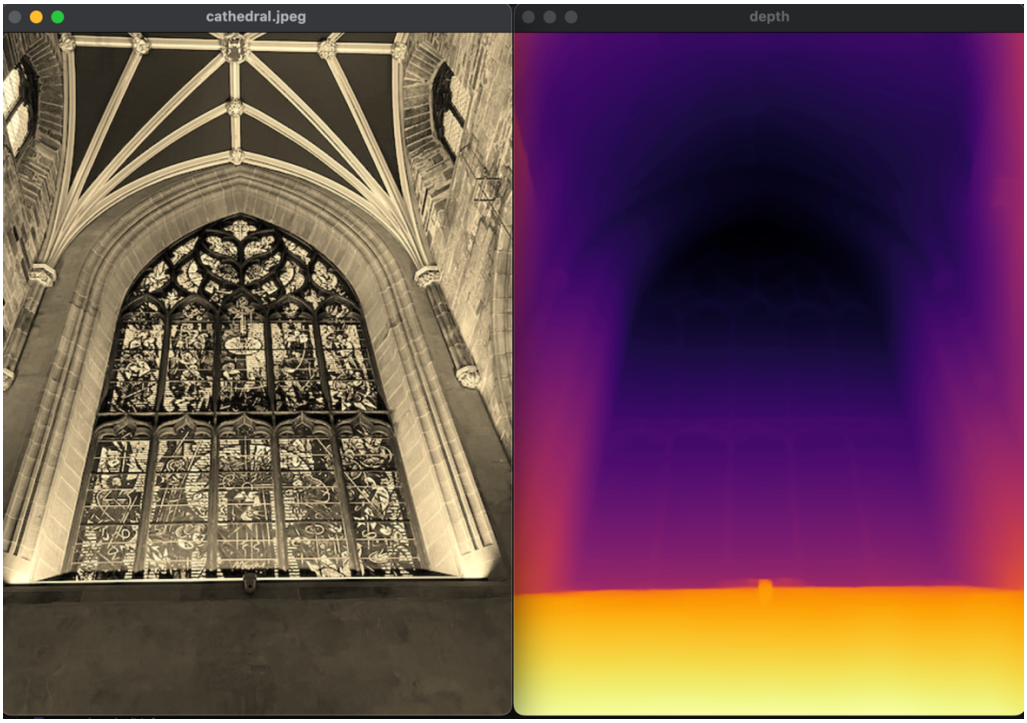
Required Images 7: a depth image from my video stream

In da2-example.cpp file, we utilized DA2Network from model_fp16.onnx model to generate the output (video feed) image as a depth map, where the depth value is stored at dst matrix, and the value shows the distance of each point in the image from the camera. A color map then is applied to the scaled depth values using cv::applyColorMap(). This function converts the grayscale depth map into a visualization color image.

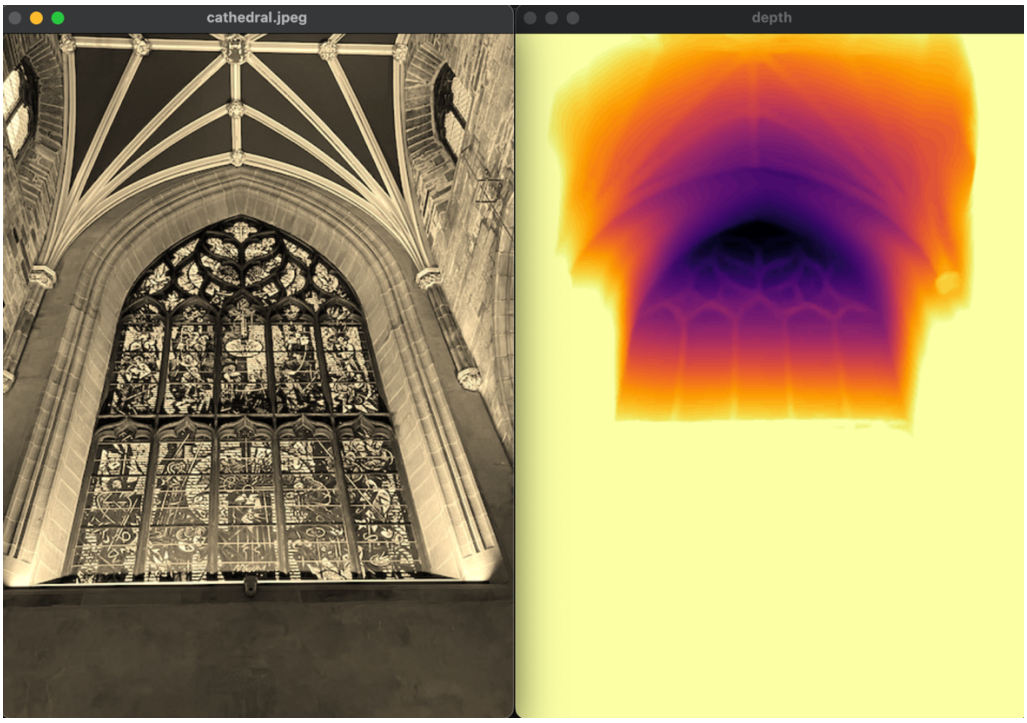


Required Images 8: an image of sepia tone filter using depth values

On top of the previous framework, I've applied sepia tone filer to the image (video feed) first and then adjust the depth values by 5 times of the previous value.



After scaling the depth values by 5 times:



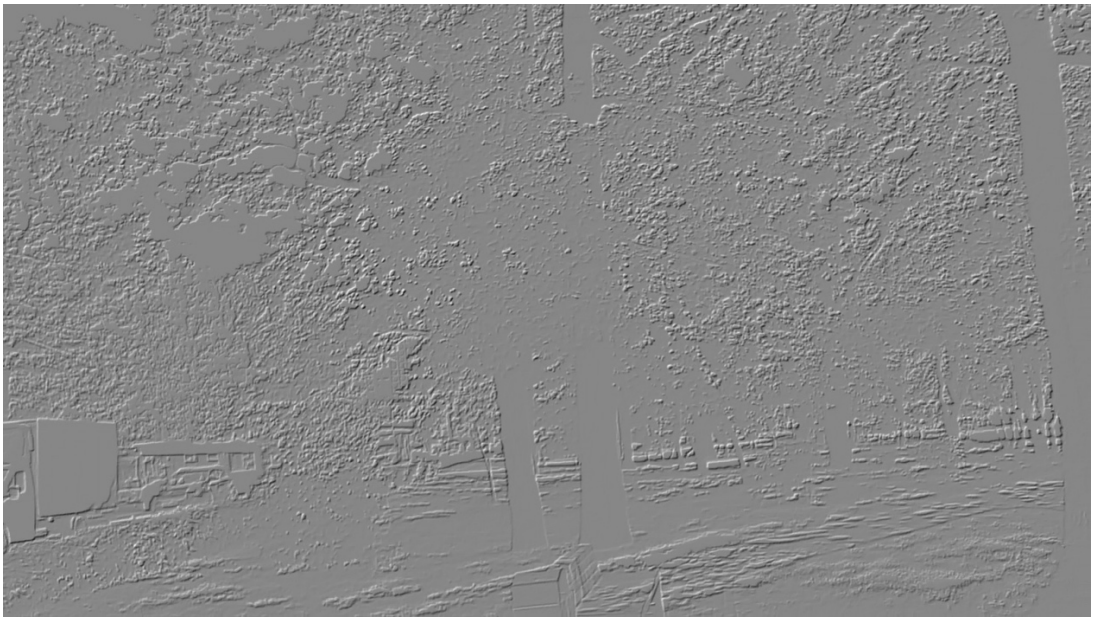
Required Images 9-11:

Cartoon effect:

The function first converts the image to grayscale and applies a median blur to reduce noise. It then detects edges, applies a bilateral filter to smooth colors while preserving edges, and combines the edges with the quantized colors for a cartoonish effect.

Emboss effect:

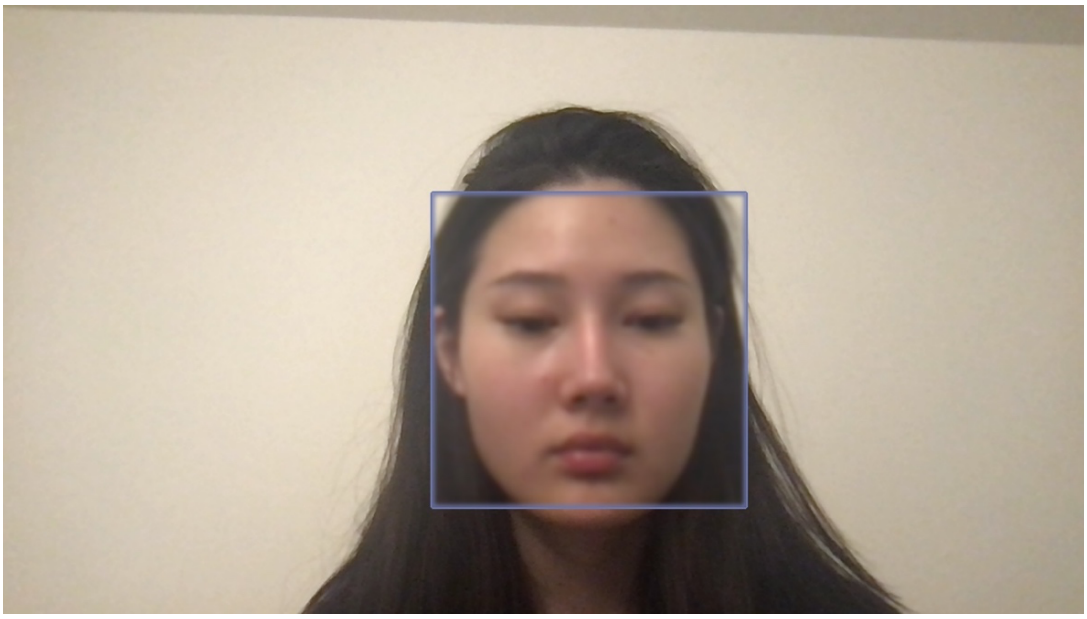
This effect computes Sobel gradients to detect edges in both X and Y directions, then applies a directional vector to create the embossing effect. The result is normalized to 8-bit values for display, enhancing the edge and depth perception.



Face blurred effect:

The applyFaceBlur function loops through each detected face in the faces vector and extracts the region of interest (ROI) corresponding to the face. We then applies a Gaussian

blur to the face ROI with a kernel size of 15x15, which can be adjusted to control the blur strength.



Extensions:

1. **Vignette filter:**
   Based on the sepia filter, I've calculated the vignette factor (1.0 - (dist / maxDist)) for each pixel and then adjust the sepia-toned values based on the distance of the pixel from the image center.

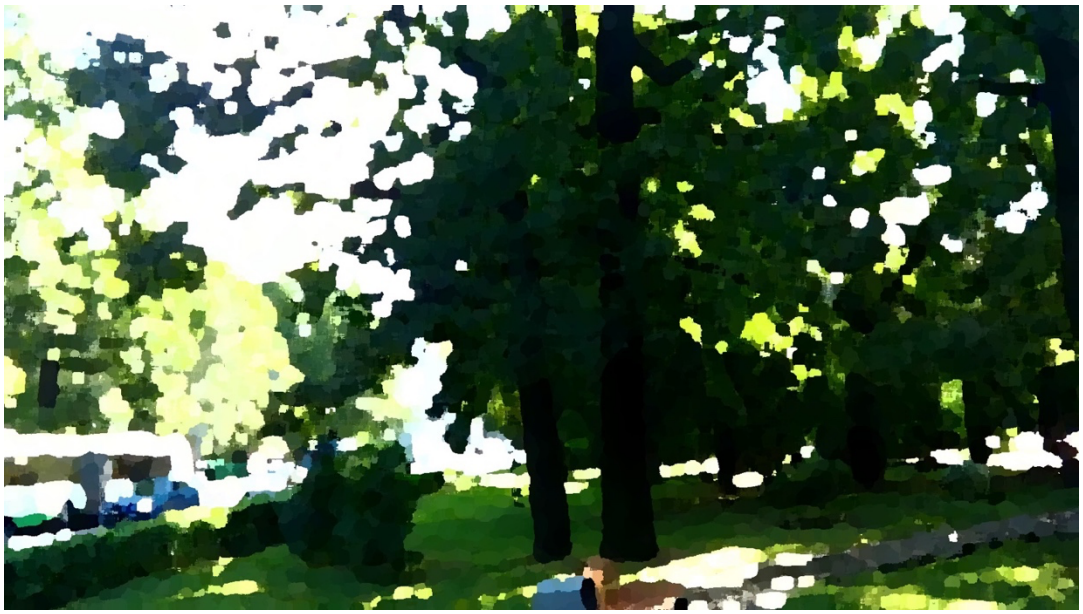
2. Pencil sketch

The pencilSketch function converts an input image to grayscale, inverts the grayscale image, and applies a Gaussian blur to the inverted image. It then creates a pencil sketch effect by dividing the grayscale image by the blurred inverted image.

3. Oil painting

Use opencv xphoto library with oil painting intrinsic effect

The image is converted to the default color space, COLOR_BGR2GRAY. For each pixel, the program calculates a histogram of its neighbors (size 2*size+1) in the first color plane and assigns the most frequent value. This results in an oil painting-like effect. Parameter 4 of oilPainting method reduces the image's dynamic range, enhancing the effect.



Reflection

I first learned the basics of RGB channel manipulation to create filters and adjust colors. Building on that, I discovered that separable filters are more efficient than single Gaussian filters because they require fewer operations. Other filters, such as bilateral filters used for cartoon effects, are important for smoothing colors while preserving edges. I also worked with Sobel x and Sobel filters to detect vertical and horizontal edges. Additionally, I explored the face detection framework and integrated the pre-trained DA2 Network into my program to generate color maps for images or video feeds. Finally, I utilized other OpenCV libraries, like xphoto, to apply additional filters.

Acknowledgement

Makefile usage:

https://cs.colby.edu/maxwell/courses/tutorials/maketutor/

Onnx runtime setup:

https://github.com/microsoft/onnxruntime/issues/8556

bilateral filtering:

https://www.geeksforgeeks.org/python-bilateral-filtering/

vignette filtering:

https://www.geeksforgeeks.org/create-a-vignette-filter-using-python-opencv/

pencil sketch filtering:

https://dev.to/azure/opencv-how-to-add-a-pencil-sketch-effect-to-an-image-using-python-3j3b

xphoto oil painting filtering:

https://docs.opencv.org/4.x/dd/d8c/tutorial_xphoto_oil_painting_effect.html