Yuyang Tian CS5330-Proj1

Project description

This project explores the mechanics of opening, capturing, manipulating, and writing images.

For handling still images, you can use the `imgDisplay.cpp` file.

The core of the project is a live-stream video application, `vidDisplay.cpp`, which allows users to apply various filters in real time using keyboard inputs. Filters include grayscale, sepia tone, blur, Sobel X, Sobel Y, gradient magnitude, and face detection. Many of these filters are implemented in `filter.cpp`.

For Task 11, I chose to generate a 3D Point Cloud from a depth map using `da-video.cpp`. To achieve this, I added two new files, `pcl3d.h` and `pcl3d.cpp`, which facilitate the generation of point cloud data and visualization of the 3D cloud.

For extension, it's a meme generator designed for creating memes by adding customizable top and bottom text to an image with outlined styling for better readability.

Task 1 - display image

The main function, a `Mat` is initialized to store a image, use `imshow` function to show a picture in a window. `waitkey` function waits users for hitting a key in the keyboard, if the key is 's', the image is saved to the host, or any other keys, the window will be closed and program exits.

Task2 - display live video

To display a video, the `VideoCapture` class is used to open the camera and capture frames.

A video consists of a sequence of images, where each image is read into a `Mat` object. If the image is not empty, it is displayed in a window. This process runs in a loop, creating a live video stream.

The `waitKey` function listens for user input. If the key pressed is 's', the current frame is saved. If 'q' is pressed, the program exits.

Task3 - grayscale

If the last user input is 'g', the video switches to a grayscale version. The `isGrey` variable determines the frame's color mode. When the `isGrey` is true, the `frame` Mat is converted to grayscale and displayed in the window.

However, since the mode depends on the most recent selection, pressing any other key will set `isGrey` to `false`, reverting the video to its original color mode, which is BGR.

Transformation within RGB space like adding and removing the alpha channel, conversion to grayscale using

$$RGB(A)\ to\ Gray: Y = 0.299 * R + 0.587 * G + 0.114 * B$$

Task4 - alternative greyscale

I created custom weights for the RGB channels as follows:

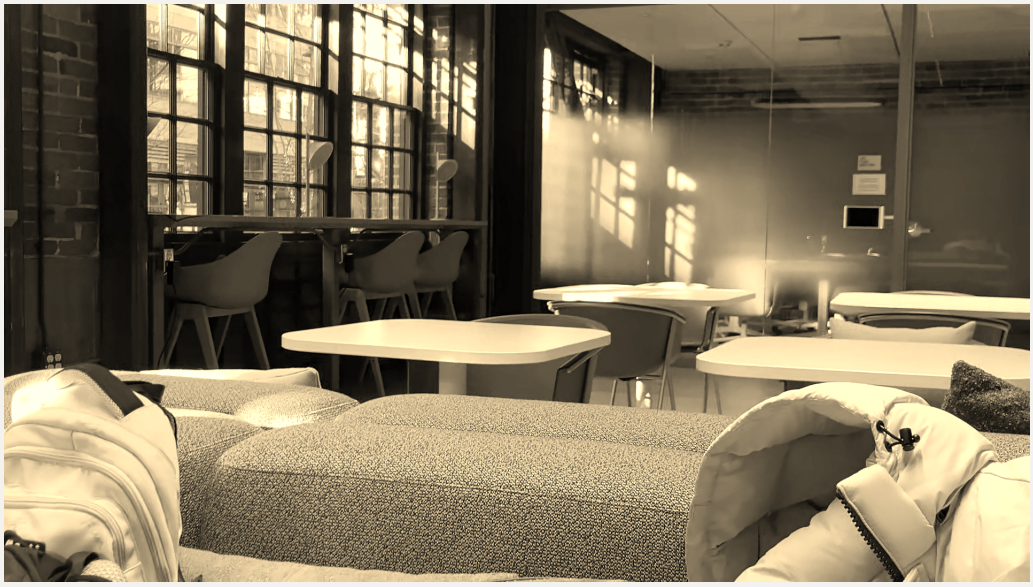
$$RGB(A)\ to\ Gray: Y = 0.005 * R + 0.8 * G + 0.195 * B$$

This alternative grayscale image appears darker in shaded areas (like chairs) and brighter in well-lit regions(windows), providing a different visual effect.

- Alternative:

Task5 - sepiatone filter with extension

I applied matrix transformation manually for each pixel to ensure transforming the original RGB values. After calculating the new values for each channel, we ensure they are within the range `[0,255]` by using `std::min()` to clip the values to 255 if they exceeds that range.



For the vignetting extension, since we want the image become darker towards edges, for each pixels, the distance to the center of the image is calculated using the Euclidean distance. And the vignetting effect is achieved by scaling the color values of each pixel. The further a pixel is from the center, the smaller the scaling factor.
$vignetteFactor = 1 - \frac{distance}{maxDistance}$

Task6 - Blur filter

- Version1:
    - Time that blurred 20 times using version1: 1.2279 seconds
- Version2:
    - Time that blurred 20 times using version1: 0.4816 seconds

- Before: 
- After: 

Ane the reason why the version2 is faster:

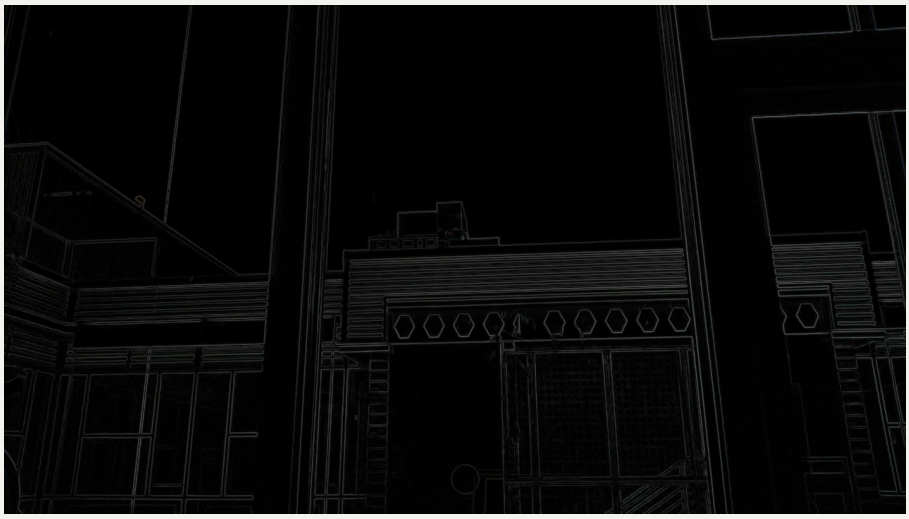
1. Separable Filtering:
    - Split the 5x5
    - Reduces operations from 25 to 10 per pixel
2. Uses `ptr()` instead of `at()` for direct memory access

Task7 - Sobel X and Soble Y

- Sobel X use two separable filter $[1, 0, -1]$ (Horizontal) and $[1, 2, 1]$ (Vertical)
- Sobel Y use two separable filter $[1, 0, -1]$ (Vertical) and $[1, 2, 1]$ (Horizontal)
- Both of them first perform the Gaussian smoothing and store the intermidiate result into a temporary `Mat` and then do the derivative step. The final result is stored in a `CV_16S3C` Mat.
- In order to show the image using `imshow`, using the function `convertScaleAbs` function to change the result to 8 bit

Task 8 - Magnitude

A function that generates a gradient magnitude image using Euclidean distance for magnitude. The magnitude image shows the strength of the pixels' intensity change using both x and y direction.



Task 9 - color quantization

I first figured out the size of a bucket using b = 255/levels, storing each fixed color value for each level in an array `colorBucket`, so the value can be retrieved instead of real-time calculation.

More levels we pass, more similar the result is to the original image.

- Original:

- Quantized(level = 10):

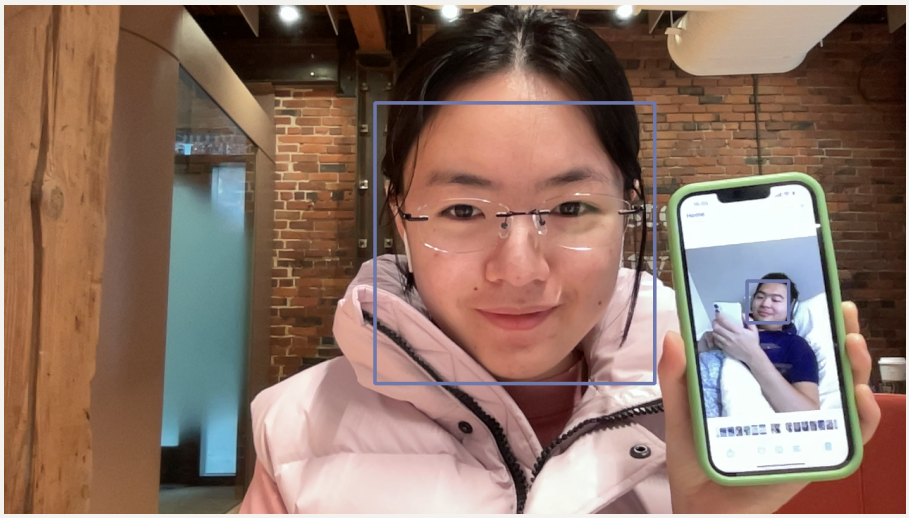
- Quantized(level = 5):

- Quantized(level = 2):

Task 10 - Detect face

This detection is achieved by calling `faceDetection` and drawing a vector of rectangular indicting the position of faces.

As displayed in this picture, my boyfriend's face and I have been detected.



Task 11 - DA2

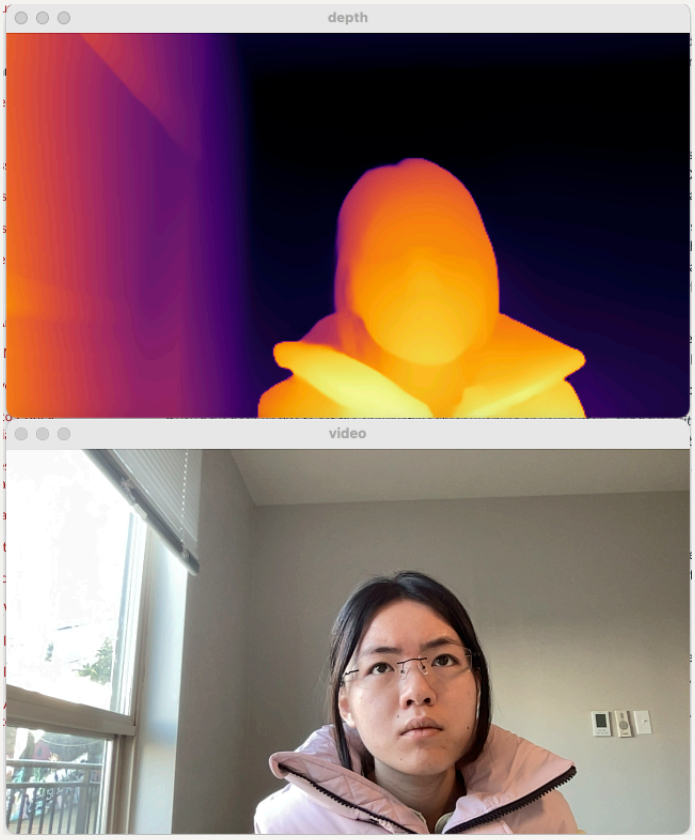
I am generally interested in Spatial Intelligence and learned about that before the Generative Model, 3D reconstruction is commonly used approach to create a 3d scene of an image. So I tried to create a 3D point cloud out of the depth information.

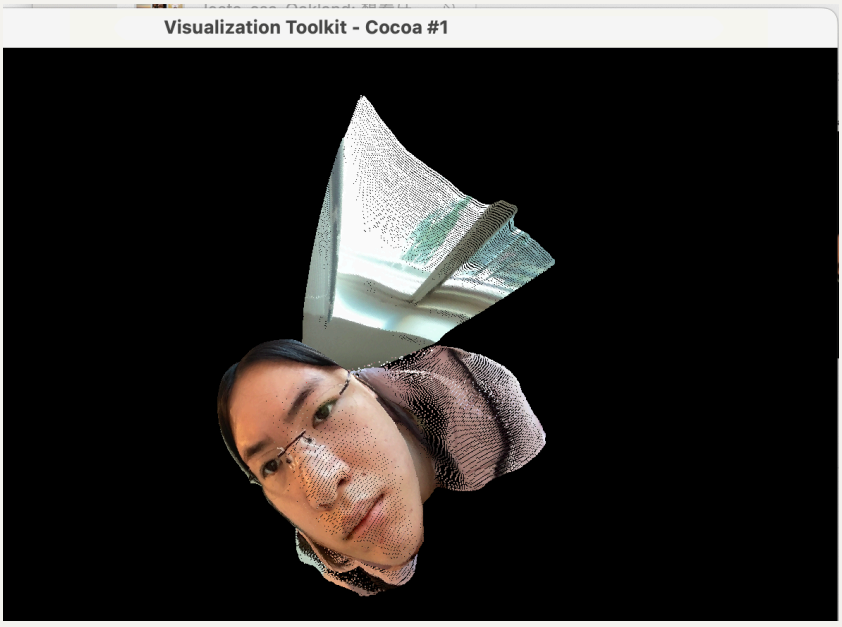
Although I spent a few hours figuring it out, I found that in the end it follows a relatively straightforward logic:

1. Generate the 3D point cloud data by projecting RGB image pixels and depth information using a camera intrinsic matrix, which gives us points with [x,y,z,color] information.
2. Save this data in PCD format (other formats work too) and use a viewer to display it. I used the PCL library for this, though Open3D would work as well.

It's a 3D model so you can acutally drag it in every direction and zoom out. There's some pictures of this 3D point cloud. It's weird but funny!

You can find the code in `da2-video.cpp`, `pcl3d.h` and `pcl3d.cpp`.





Task 12 - blur background filter, fog filter,

1. Blur the image outside of found faces. (Built upon face detector)

   We already have a vector of face positions, so we want to exclude these areas and blur the other areas. So we gonna use `Mask` in openCV to mask face as 0 and background as 1.

2. Add fog to the image using the depth values

   So the fogFactor is calculated by
   $1.0f - exp(-invertedDepth * invertedDepth * fogDensity)$

   each pixel is converted following
   $(1.0 - fogFactor) * pixel + Vec3b(255, 255, 255) * fogFactor$

3. Make the face colorful, while the rest of the image is greyscale.

Similar with the filter that blurs the image outside of found faces, the filter utilize the face vector to create corresponding circle masks. And then we need to use `copyTo()` with the inverted mask (`~mask`). As a result, faces stay original but background become blurred.

Extension Video showcase

The `MemeEditor` is a simple tool designed for creating memes by adding customizable top and bottom text to an image with outlined styling for better readability. It provides an interactive interface using OpenCV, allowing users to preview edits in real-time.

How to Use:

1. Call `editMeme(Mat& img)` with the image you want to edit.
2. During editing:
    - `t`: Add or edit top text.
    - `b`: Add or edit bottom text.
    - `c`: Clear all text.
    - `s`: Save the meme and exit.
    - `q`: Quit without saving.
3. Changes are displayed live, and the meme is saved as a file when finalized.

Reflection

My biggest feeling towards this course and this project is that I fell in love with Computer Vision! There's so much we can do with image pixels. It makes me even more curious about the biggest unanswered questions in this field and the scenarios where Computer Vision can benefit humanity.

Another lesson I learned is that I can be tricked into thinking I understand something just by watching the lecture. But when it comes to applying it, I realize I don’t have all the scaffolding to remember and implement it on my own. The project definitely helped me solidify my understanding. For example, I thought I already knew how to perform convolution using loops, but it turned out I spent a while figuring out those tricky details.

Also, the precious part of this project was that it gave me the opportunity to explore. I learned that we can use depth maps to convert 2D images into 3D models, which is pretty cool. For an extension, I originally wanted to create an effect where, if it recognized me making a heart-hand gesture, the frame would burst into many colorful images. But in the end, I realized it could be extremely difficult without a state-of-the-art hand gesture library—even Google hasn't open-sourced one like that.

In summary, it was definitely a no-pain-no-gain journey.

Acknowledgement

https://www.baeldung.com/cs/focal-length-intrinsic-camera-parameters

https://www.youtube.com/watch?v=fi7gQI4PWkY

https://docs.opencv.org/4.5.1/dc/d84/group__core__basic.html#ga7d080aa40de011e4410bca63385ffe2a

https://paperswithcode.com/task/depth-estimation

https://onnxruntime.ai/docs/

https://stackoverflow.com/questions/64094463/how-can-i-generate-a-point-cloud-from-a-depth-image-and-rgb-image

https://pcl.readthedocs.io/projects/tutorials/en/master/reading_pcd.html

https://pcl.readthedocs.io/projects/tutorials/en/master/cloud_viewer.html#cloud-viewer

https://www.baeldung.com/cs/focal-length-intrinsic-camera-parameters

https://opengl-notes.readthedocs.io/en/latest/topics/texturing/aliasing.html

https://stackoverflow.com/questions/11416032/how-to-zero-everything-within-a-masked-part-of-an-image-in-opencv

https://pyimagesearch.com/2021/01/04/opencv-augmented-reality-ar/