CS 5330 Project 1

Sihe Chen (002085773) chen.sihe1@northeastern.edu

Description

In this project, we start from installing opencv and displaying a static image, and then integrate live camera video stream into our program, after which we implemented multiple filters and effects on the video frames. The filters include but are not limited to GrayScale filter, Sepia tone filter, Gaussian blur filter, Sobel X/Y/magnitude filter, as well as facial detection features. In the last three self-designed filters, frosted glass filter, time delay effect as well as facial improvement filter are implemented.

Environment Setup

- WSL2: `Linux LEGION-MARTIN 5.15.153.1-microsoft-standard-WSL2+ #1 SMP Sat Sep 14 13:55:47 PDT 2024 x86_64 x86_64 x86_64 GNU/Linux`
- Windows 11: Enterprise Version, 23H2, 22631.4168
- OpenCV version: 4.10.0-dev
- Compiler version (gcc/g++): 11.4.0
- one-hit command for compile & run: `cmake -B build && cd build && make && cd .. && ./build/vidDisplay`
- compilation products are stored in `build` so that they won't be mixed in code files.

Tasks

Task 1 Read an image from a file and display it

What I've done

- install opencv according to documentation
- run the provided test code

Task 2 Display live video

What I've done

- set up the camera
- write a main function
- compile and run

Problems I met with

User Input Handling

- We can only input when the video displayer is in the front (active). Otherwise the program cannot receive the input character.
- When there's no user input, `cv::WaitKey()` actually do read once and return `-1`. Therefore we need to handle the idle loop by not using user input when the return value is `-1`.

- ### How to set up camera in WSL
    - Step 1: compile a customized kernel that allows device to be shared between Windows and WSL.
    - Step 2: attach the camera to WSL and disable it in Windows
        - ```
          # on windows:
          usbipd list
          usbipd attach --wsl --busid <busid>

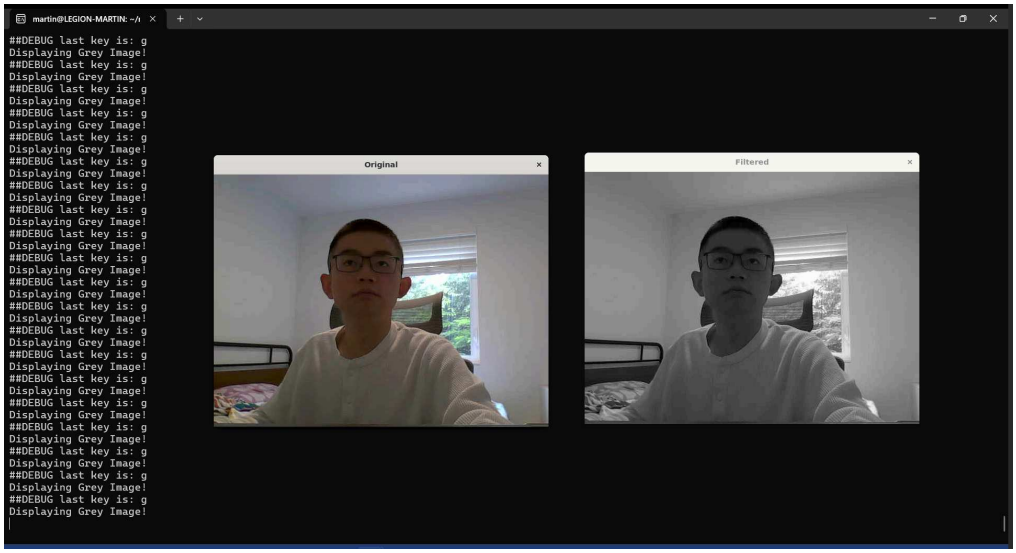
          # on linux:
          ls /dev/video* # shows /dev/video0
          ```
    - Add this line of code when reading from `VideoCapture`: `camera.set(cv::CAP_PROP_FOURCC, cv::VideoWriter::fourcc('M', 'J', 'P', 'G'));`
    - References:
        - https://github.com/PINTO0309/wsl2_linux_kernel_usbcam_enable_conf/
        - https://zenn.dev/pinto0309/articles/e1432253d29e30
        - https://forum.opencv.org/t/videoio-v4l2-dev-video0-select-timeout/8822/5

Further Discussion

- Probably the better idea is to directly run in Windows environment or to use a Linux virtual machine. WSL is convenient but not that convenient.

Task 3 Display greyscale live video

What I've done

- `cv::cvtColor(src, dst, cv::COLOR_BGR2GRAY);`
- 
          ```

- Explanation of how each channel is weighted in the conversion in official documentation:

  RGB ↔ GRAY

  Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

  $$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

  and

  $$\text{Gray to RGB[A]:} \quad R \leftarrow Y, G \leftarrow Y, B \leftarrow Y, A \leftarrow \max(ChannelRange)$$

  The conversion from a RGB image to gray is done with:

  ```
  cvtColor(src, bwsrc, cv::COLOR_RGB2GRAY);
  ```

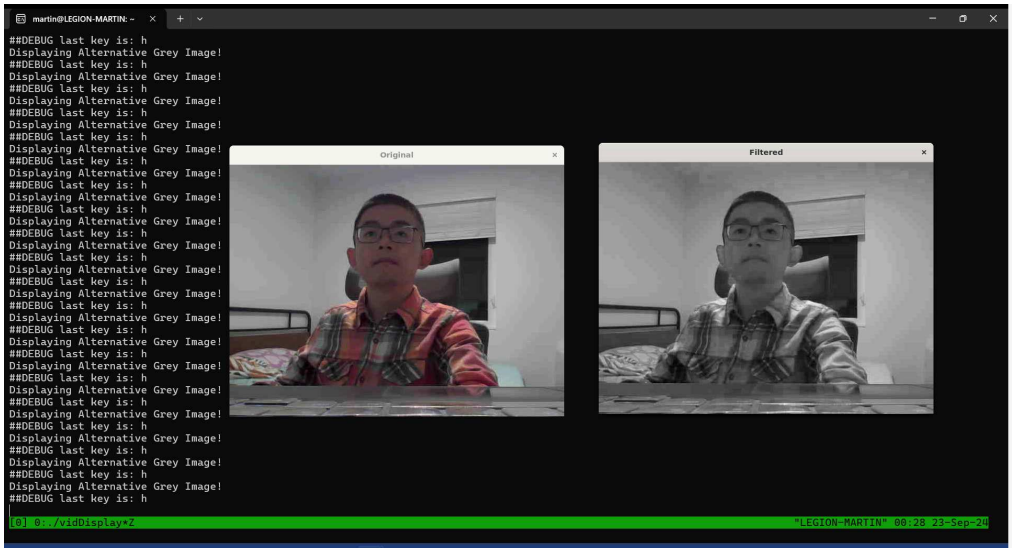
  More advanced channel reordering can also be done with **cv::mixChannels**.

  - Essentially this is because the sensitivity of human vision system to different colors, which indicates that we are mostly sensitive to green, less sensitive to red and least sensitive to blue. After abundant experiments, we decide on the weight of three colors in grayscale conversion.

Task 4 Display alternative greyscale live video

What I've done

- The grayscale filter that I use: GrayScale = max[R, G, B]

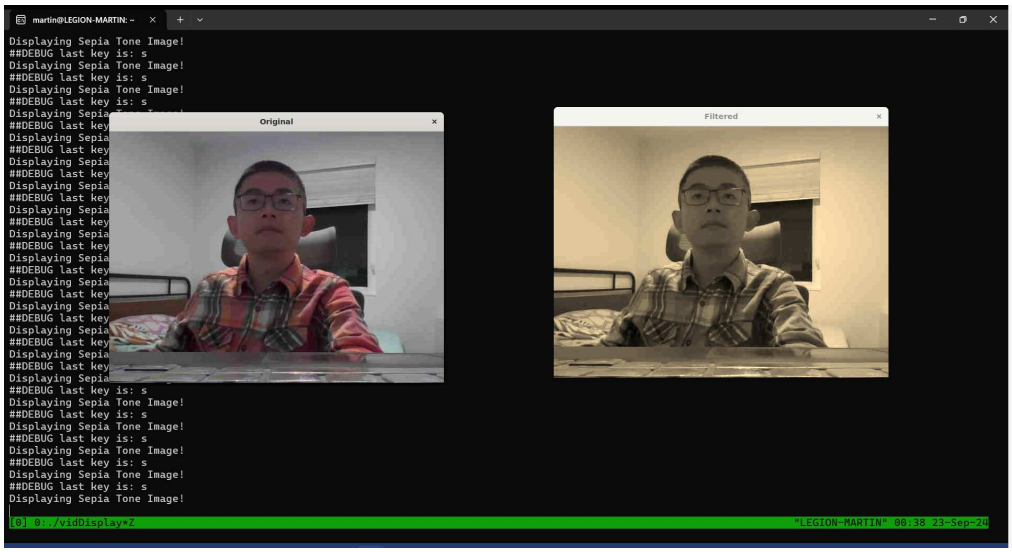


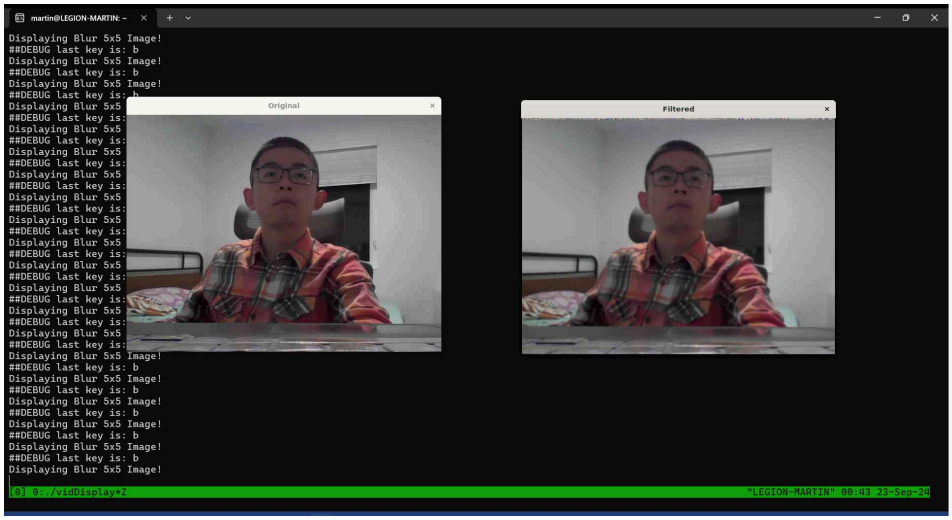
- Under the same lighting, the result of the alternative grayscale algorithm is brighter, since I used `max` to pick the grayscale value, and the larger the value is, the brighter the image will be.

Problems I met with (and also further discussion)

cvtColor create a new matrix!!!

- Every time we call `cvtColor`, it automatically creates a new matrix as `dst`, and the old memory is released. It usually makes more sense to let the caller of the interface manage the memory and inside the interface the memory is only used but not allocated, but it's not required as long as the assumption is aligned in the whole software system. In the alternative version, I took the same approach of creating a new matrix in the filter function.
- https://github.com/egonSchiele/OpenCV/blob/master/modules/imgproc/src/color.cpp

- # Task 5 Implement a Sepia tone filter
  - ## What I've done
    - 

    - We need to ensure the color value not greater than 255! My approach is to use `std::min` function to limit the computation result. `std::clamp` can also server the purpose.
- # Task 6 Implement a 5x5 blur filter
  - ## What I've done
    - A. naive implementation to get a baseline
      - 

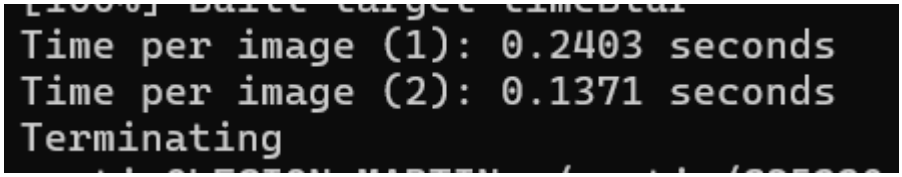
      - ```
        // ...
          for (int i = 2; i < src.rows - 2; ++i) {
            for (int j = 2; j < src.cols - 2; ++j) {
              for (int c = 0; c < 3; ++c) {
                uint32_t conv = 0;
                for (int k = -2; k <= 2; ++k) {
                  for (int l = -2; l <= 2; ++l) {
                    conv += src.at<cv::Vec3b>(i + k, j + l)[c] *
        coeff[k + 2][l + 2];
                  }
                }
                dst.at<cv::Vec3b>(i, j)[c] = static_cast<uint8_t>
        (conv / sum);
              }
        ```
        ```

```
    }
  }
// ...
```

- B. speed up with matrix multiplication loop unrolling
  - ```
    // ...
    for (int i = 2; i < src.rows - 2; ++i) {
      for (int j = 2; j < src.cols - 2; ++j) {
        for (int c = 0; c < 3; ++c) {
          uint32_t conv = 0;
          conv += src.at<cv::Vec3b>(i - 2, j - 2)[c] * coeff[0][0] +
                  src.at<cv::Vec3b>(i - 2, j - 1)[c] * coeff[0][1] +
                  src.at<cv::Vec3b>(i - 2, j)[c] * coeff[0][2] +
                  src.at<cv::Vec3b>(i - 2, j + 1)[c] * coeff[0][3] +
                  src.at<cv::Vec3b>(i - 2, j + 2)[c] * coeff[0][4] +
                  src.at<cv::Vec3b>(i - 1, j - 2)[c] * coeff[1][0] +
                  src.at<cv::Vec3b>(i - 1, j - 1)[c] * coeff[1][1] +
                  src.at<cv::Vec3b>(i - 1, j)[c] * coeff[1][2] +
                  src.at<cv::Vec3b>(i - 1, j + 1)[c] * coeff[1][3] +
                  src.at<cv::Vec3b>(i - 1, j + 2)[c] * coeff[1][4] +
                  src.at<cv::Vec3b>(i, j - 2)[c] * coeff[2][0] +
                  src.at<cv::Vec3b>(i, j - 1)[c] * coeff[2][1] +
                  src.at<cv::Vec3b>(i, j)[c] * coeff[2][2] +
                  src.at<cv::Vec3b>(i, j + 1)[c] * coeff[2][3] +
                  src.at<cv::Vec3b>(i, j + 2)[c] * coeff[2][4] +
                  src.at<cv::Vec3b>(i + 1, j - 2)[c] * coeff[3][0] +
                  src.at<cv::Vec3b>(i + 1, j - 1)[c] * coeff[3][1] +
                  src.at<cv::Vec3b>(i + 1, j)[c] * coeff[3][2] +
                  src.at<cv::Vec3b>(i + 1, j + 1)[c] * coeff[3][3] +
                  src.at<cv::Vec3b>(i + 1, j + 2)[c] * coeff[3][4] +
                  src.at<cv::Vec3b>(i + 2, j - 2)[c] * coeff[4][0] +
                  src.at<cv::Vec3b>(i + 2, j - 1)[c] * coeff[4][1] +
                  src.at<cv::Vec3b>(i + 2, j)[c] * coeff[4][2] +
                  src.at<cv::Vec3b>(i + 2, j + 1)[c] * coeff[4][3] +
                  src.at<cv::Vec3b>(i + 2, j + 2)[c] * coeff[4][4];
    ```
    ```

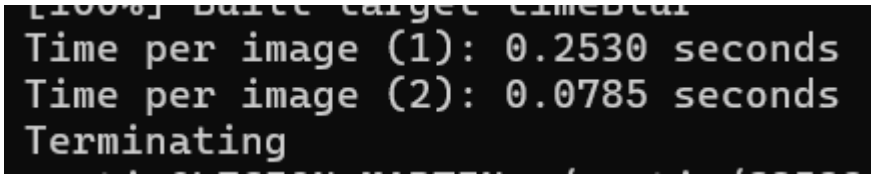
```
31        dst.at<cv::Vec3b>(i, j)[c] = static_cast<uint8_t>
    (conv / sum);
32      }
33    }
34  }
35  // ...
```

- comparison: naive vs innerloop unrolling



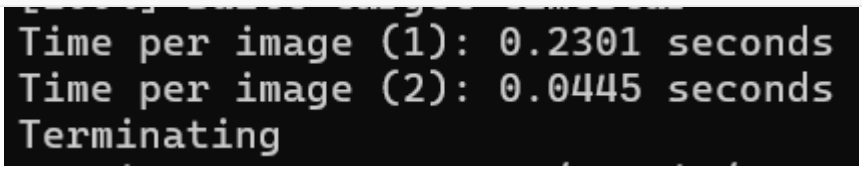
- C. use pointer when accessing elements in `cv::Mat`

```
1   // ...
2   for (int i = 2; i < src.rows - 2; ++i) {
3     const cv::Vec3b *row_m2 = src.ptr<cv::Vec3b>(i - 2);
4     const cv::Vec3b *row_m1 = src.ptr<cv::Vec3b>(i - 1);
5     const cv::Vec3b *row = src.ptr<cv::Vec3b>(i);
6     const cv::Vec3b *row_p1 = src.ptr<cv::Vec3b>(i + 1);
7     const cv::Vec3b *row_p2 = src.ptr<cv::Vec3b>(i + 2);
8     cv::Vec3b *dst_ptr = dst.ptr<cv::Vec3b>(i);
9     for (int j = 2; j < src.cols - 2; ++j) {
10      for (int c = 0; c < 3; ++c) {
11        uint32_t conv = 0;
12        conv +=
13            row_m2[j - 2][c] * coeff[0][0] + row_m2[j - 1][c]
    * coeff[0][1] +
14            row_m2[j][c] * coeff[0][2] + row_m2[j + 1][c] *
    coeff[0][3] +
15            row_m2[j + 2][c] * coeff[0][4] + row_m1[j - 2][c]
    * coeff[1][0] +
16            row_m1[j - 1][c] * coeff[1][1] + row_m1[j][c] *
    coeff[1][2] +
17            row_m1[j + 1][c] * coeff[1][3] + row_m1[j + 2][c]
    * coeff[1][4] +
18            row[j - 2][c] * coeff[2][0] + row[j - 1][c] *
    coeff[2][1] +
19            row[j][c] * coeff[2][2] + row[j + 1][c] *
    coeff[2][3] +
20            row[j + 2][c] * coeff[2][4] + row_p1[j - 2][c] *
    coeff[3][0] +
21            row_p1[j - 1][c] * coeff[3][1] + row_p1[j][c] *
    coeff[3][2] +
22            row_p1[j + 1][c] * coeff[3][3] + row_p1[j + 2][c]
    * coeff[3][4] +
23            row_p2[j - 2][c] * coeff[4][0] + row_p2[j - 1][c]
    * coeff[4][1] +
24            row_p2[j][c] * coeff[4][2] + row_p2[j + 1][c] *
    coeff[4][3] +
25            row_p2[j + 2][c] * coeff[4][4];
26        dst_ptr[j][c] = static_cast<uint8_t>(conv / sum);
27      }
28    }
29  }
30  // ...
```

- 

- D. split matrix multiplication into row multiplication and column multiplication
  - ```
    // ...
    cv::Mat mul_row(src.rows, src.cols, CV_32SC3);
    cv::Mat mul_col(src.rows, src.cols, CV_32SC3);
    // 1. compute rows
    for (int i = 2; i < src.rows - 2; ++i) {
      const cv::Vec3b *src_m2_ptr = src.ptr<cv::Vec3b>(i - 2);
      const cv::Vec3b *src_m1_ptr = src.ptr<cv::Vec3b>(i - 1);
      const cv::Vec3b *src_0_ptr = src.ptr<cv::Vec3b>(i);
      const cv::Vec3b *src_p1_ptr = src.ptr<cv::Vec3b>(i + 1);
      const cv::Vec3b *src_p2_ptr = src.ptr<cv::Vec3b>(i + 2);
      cv::Vec3i *mul_row_ptr = mul_row.ptr<cv::Vec3i>(i);
      for (int j = 0; j < src.cols; ++j) {
        mul_row_ptr[j][0] = static_cast<int>(src_m2_ptr[j][0])
    * split[0] +
                            static_cast<int>(src_m1_ptr[j][0])
    * split[1] +
                            static_cast<int>(src_0_ptr[j][0]) *
    split[2] +
                            static_cast<int>(src_p1_ptr[j][0])
    * split[3] +
                            static_cast<int>(src_p2_ptr[j][0])
    * split[4];
        mul_row_ptr[j][1] = static_cast<int>(src_m2_ptr[j][1])
    * split[0] +
                            static_cast<int>(src_m1_ptr[j][1])
    * split[1] +
                            static_cast<int>(src_0_ptr[j][1]) *
    split[2] +
                            static_cast<int>(src_p1_ptr[j][1])
    * split[3] +
                            static_cast<int>(src_p2_ptr[j][1])
    * split[4];
        mul_row_ptr[j][2] = static_cast<int>(src_m2_ptr[j][2])
    * split[0] +
                            static_cast<int>(src_m1_ptr[j][2])
    * split[1] +
                            static_cast<int>(src_0_ptr[j][2]) *
    split[2] +
                            static_cast<int>(src_p1_ptr[j][2])
    * split[3] +
                            static_cast<int>(src_p2_ptr[j][2])
    * split[4];
      }
    }
    // 2. compute cols
    for (int i = 0; i < src.rows; ++i) {
      const cv::Vec3i *mul_row_ptr = mul_row.ptr<cv::Vec3i>(i);
      cv::Vec3i *mul_col_ptr = mul_col.ptr<cv::Vec3i>(i);
      for (int j = 2; j < src.cols - 2; ++j) {
        mul_col_ptr[j][0] =
            mul_row_ptr[j - 2][0] * split[0] + mul_row_ptr[j -
    1][0] * split[1] +
    ```
    ```

```
        mul_row_ptr[j][0] * split[2] + mul_row_ptr[j + 1][0] * split[3] +
        mul_row_ptr[j + 2][0] * split[4];
    mul_col_ptr[j][1] =
        mul_row_ptr[j - 2][1] * split[0] + mul_row_ptr[j - 1][1] * split[1] +
        mul_row_ptr[j][1] * split[2] + mul_row_ptr[j + 1][1] * split[3] +
        mul_row_ptr[j + 2][1] * split[4];
    mul_col_ptr[j][2] =
        mul_row_ptr[j - 2][2] * split[0] + mul_row_ptr[j - 1][2] * split[1] +
        mul_row_ptr[j][2] * split[2] + mul_row_ptr[j + 1][2] * split[3] +
        mul_row_ptr[j + 2][2] * split[4];
  }
}
// 3. fill result
for (int i = 0; i < src.rows; ++i) {
  const cv::Vec3i *mul_col_ptr = mul_col.ptr<cv::Vec3i>(i);
  cv::Vec3b *dst_ptr = dst.ptr<cv::Vec3b>(i);
  for (int j = 0; j < src.cols; ++j) {
    dst_ptr[j][0] = static_cast<uint8_t>(mul_col_ptr[j][0] / sum);
    dst_ptr[j][1] = static_cast<uint8_t>(mul_col_ptr[j][1] / sum);
    dst_ptr[j][2] = static_cast<uint8_t>(mul_col_ptr[j][2] / sum);
  }
}
// ...
```

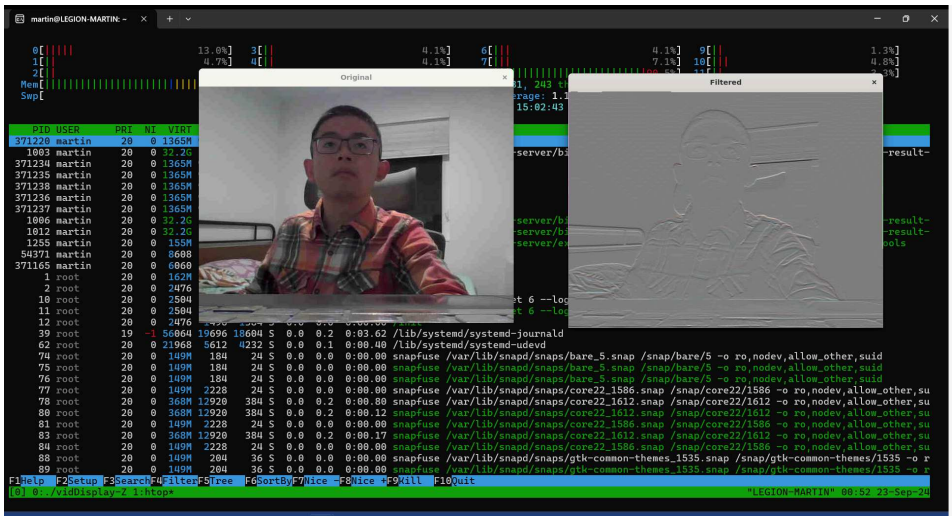
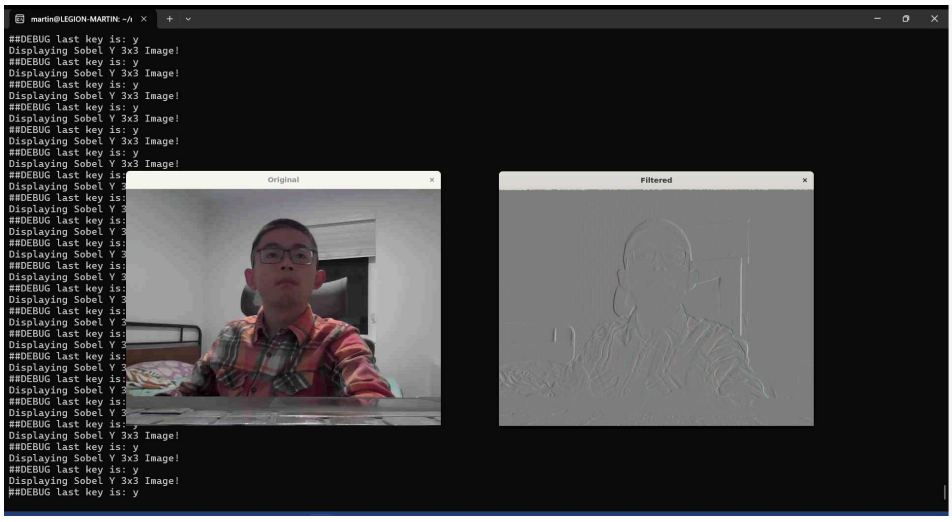
- 

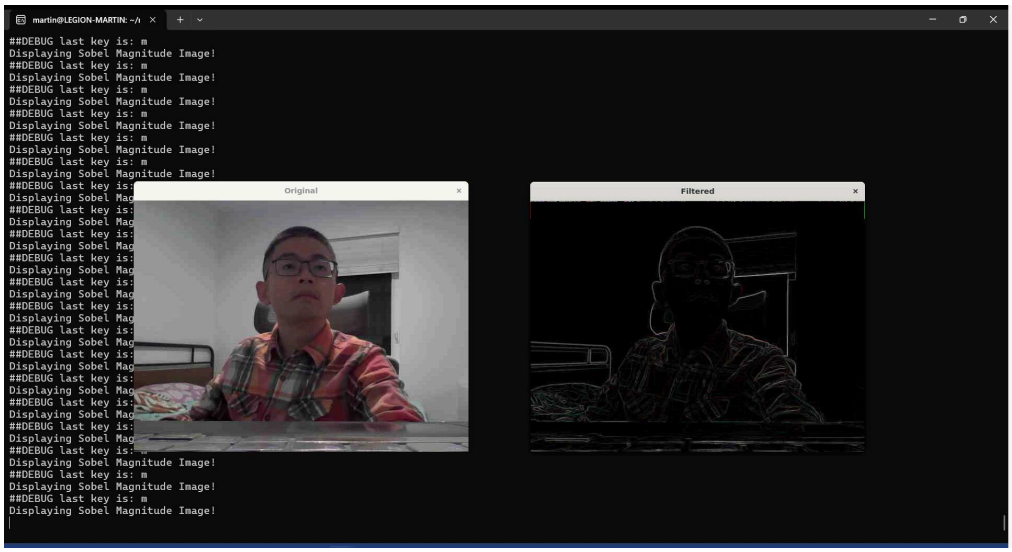
- By calling this function in vidDisplay, I found that there were actually bugs in the code because the image looks weird, but they were extremely hard to find with human eyes. I then sent the code to LLM and it immediately found the bugs, where I typoed the index in "split" matrix.
- The time reduction of splitting multiplication is within expectation, since the operations were reduced from 25 muls + 24 adds to 10 muls + 8 adds (approximately 2 times of acceleration because muls are dominant).

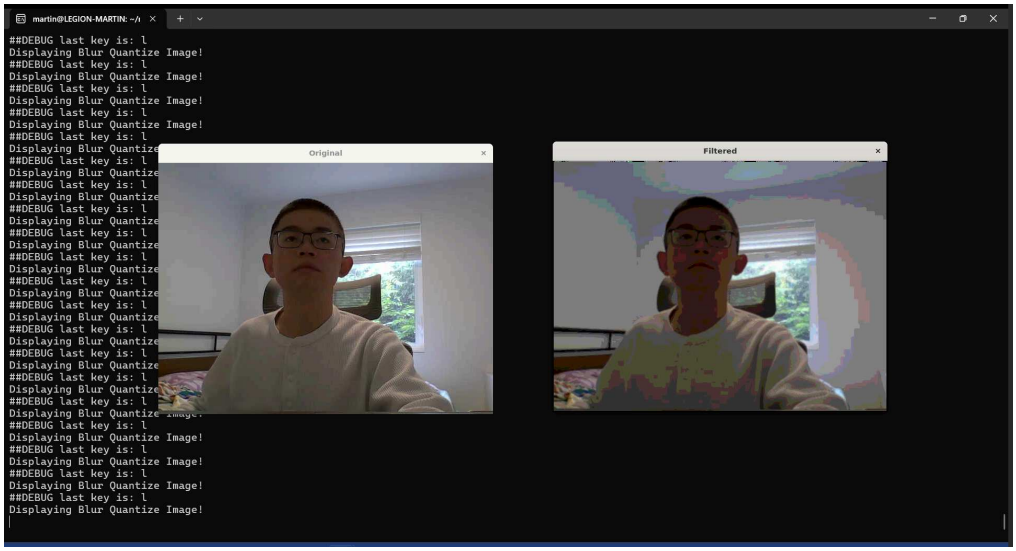
Further Discussion

Can the compiler help to do loop unrolling?

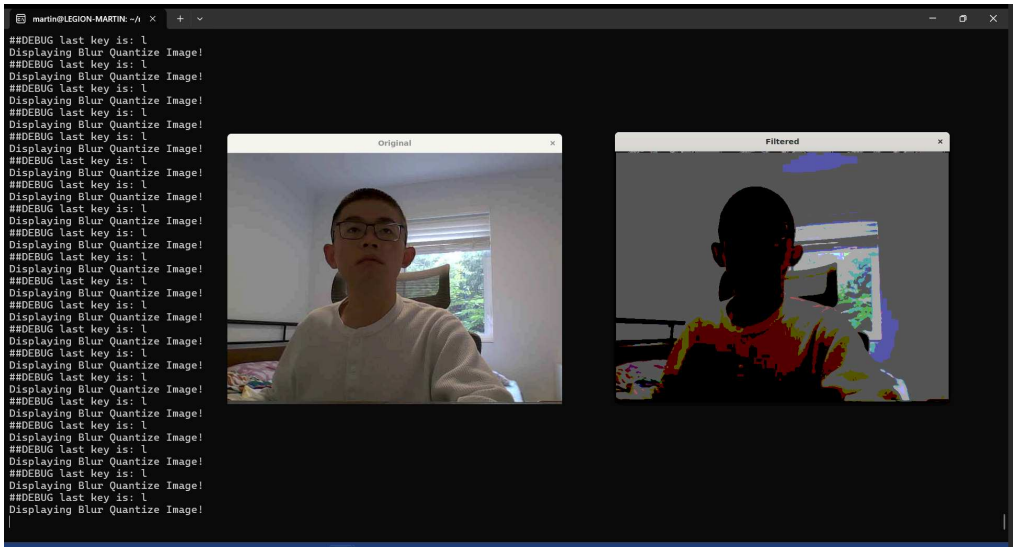
- Loop unrolling is extremely tedious and easy to make a hard-to-debug mistake. If compilers can help with fixed length loop unrolling things will be much easier. If not, AI-assisted compiler can definitely do that. (will research on this topic later!)

        - ### Can the compiler help split the matrix multiplication?
            - One of the most trivial approaches is to set up a table that stores all the matrices that can be split when doing multiplication. The solution can be very customized to different applications.
- # Task 7 Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters
    - ## What I've done
        - sobel x:
            - 
        - sobel y:
            - 
    - ## Problems I met with
        - ### How to express vector of int16_t in OpenCV?
            - `cv::Vec3s`, where s stands for `short int`.

- # Task 8 Implement a function that generates a gradient magnitude image from the X and Y Sobel images
  - ## What I've done
    - Implement the gradient with `sqrt(sobel_x, sobel_y)`. `sqrt` function naturally maps pixel value from [-255, 255] to [0, 255], so we don't need to worry about data type and directly convert the value from `int16_t` to `uint8_t`!
    - 

- # Task 9 Implement a function that blurs and quantizes a color image
  - ## What I've done
    - levels = 10
    

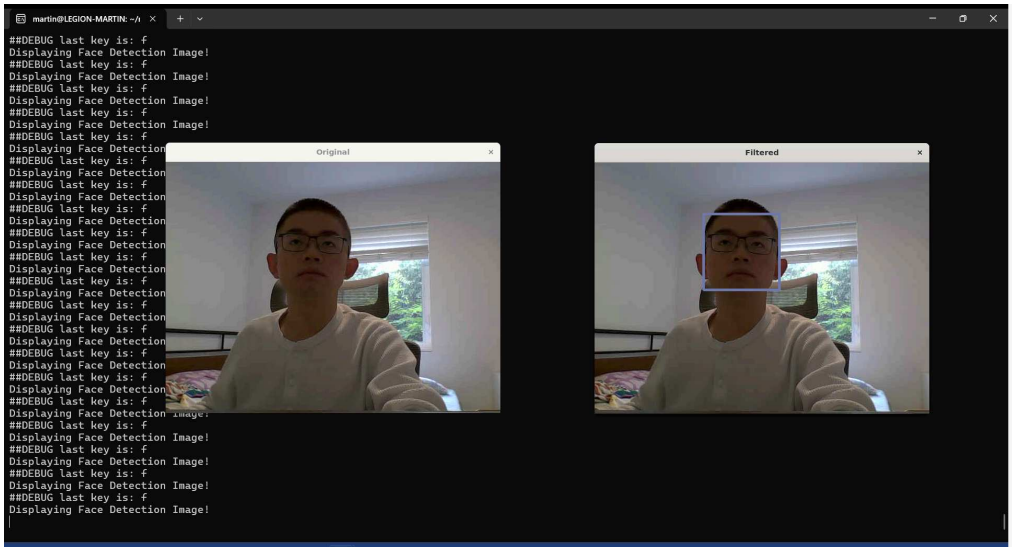
- levels = 3



- Quantization essentially uses less bits to express at the cost of accuracy. It's a commonly-used technique in acceleration where the precision of floating point can be traded off with faster computation of integer.

Task 10 Detect faces in an image

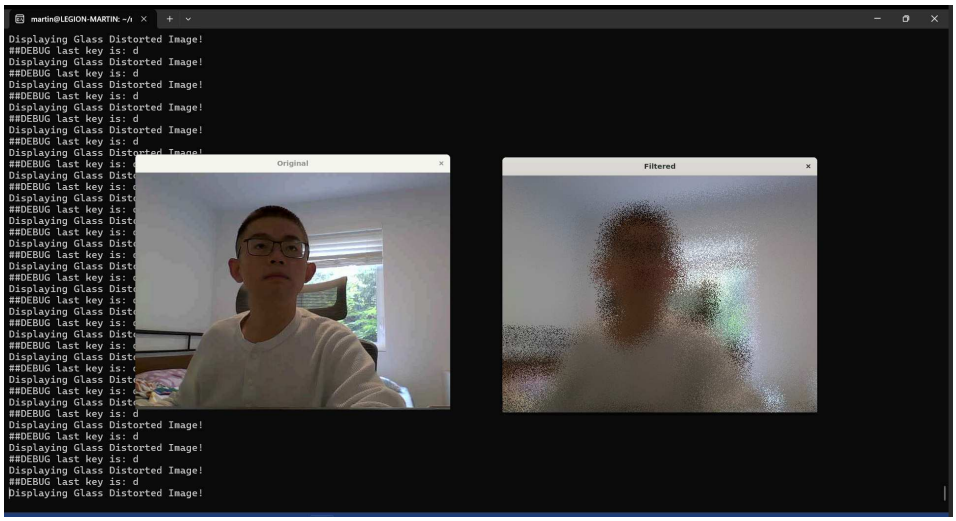
What I've done

- Move `faceDetect.cpp` and `faceDetect.h` into the directory.
- Include opencv in `faceDetect.h`.
- I forgot to break the switch clause, so it kept displaying the original image. Spent about 15mins debugging... uhh!
- 

Further Discussion

How does it work?

- Cascade Classifier: cascading multiple stages (weak classifiers) to achieve robust and efficient object detection.
- The weight values are stored in the xml file and loaded into the program.
- This classifier seems to have been trained on front-face images only, as when I turn my face to the side, the box disappeared.

- # Task 11 Implement three more effects on your video
  - ## Glass Distortion
    - ### Goal
      - Apply a frosted glass effect over the image, essentially blurring it. The algorithm is not modifying the pixel value, but moving the pixel to a random neighbor position.
    - ### Design
      - `int glassDistort(const cv::Mat &src, const int distortionStrength, cv::Mat &dst)`
        - `distortionStrength` stands for the farthest distance that a pixel can be moved horizontally and vertically.
    - ### Result
      - 

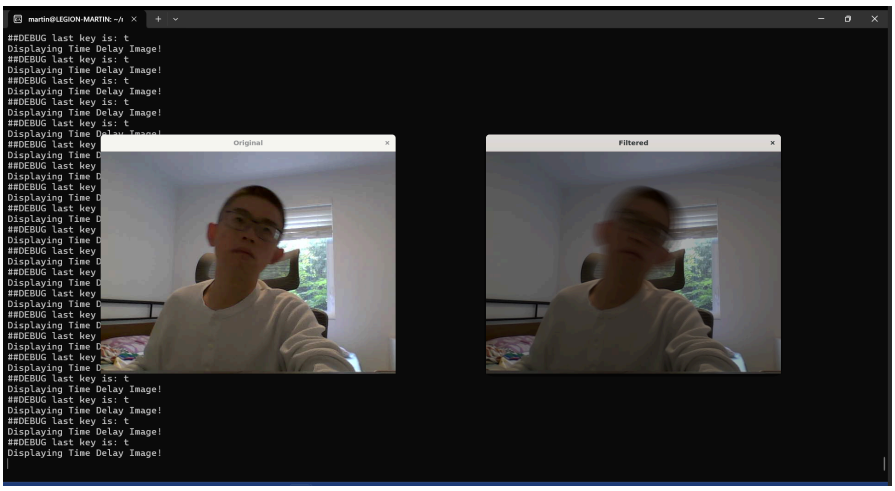
  - ## Time Delay Effect
    - ### Goal
      - Merge multiple layers of frames in a fix time range, creating a slow motion effect.
    - ### Design
      - ```
        class TimeDelay {
        public:
          TimeDelay(int count, float alpha) : count_(count), alpha_(alpha) {}
          ~TimeDelay() = default;
          int timeDelay(const cv::Mat &src, cv::Mat &dst);

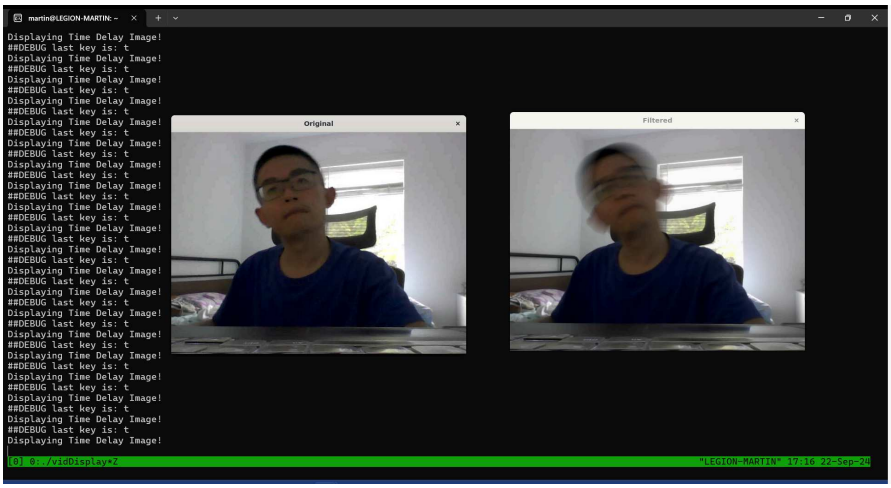
        private:
          int count_;
          float alpha_;
          std::deque<cv::Mat> frames_;
        };

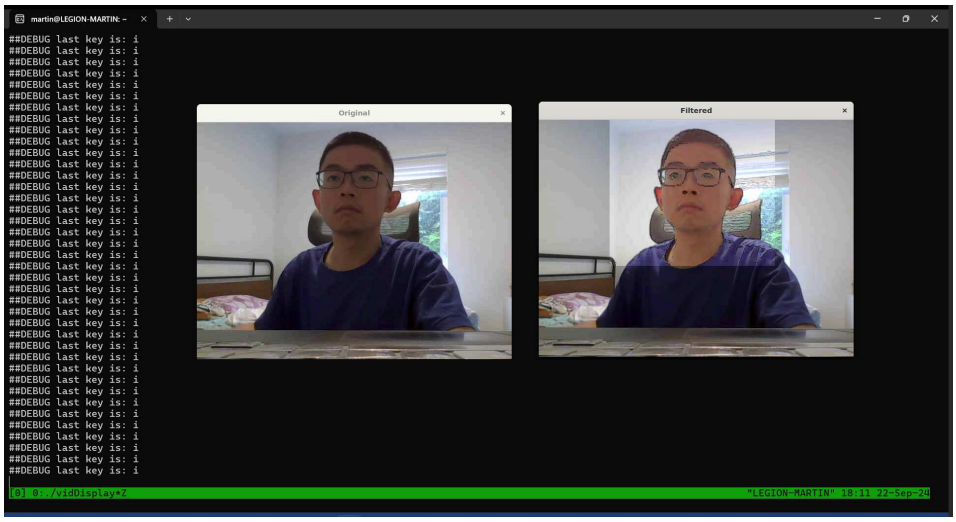
        ```
        ```

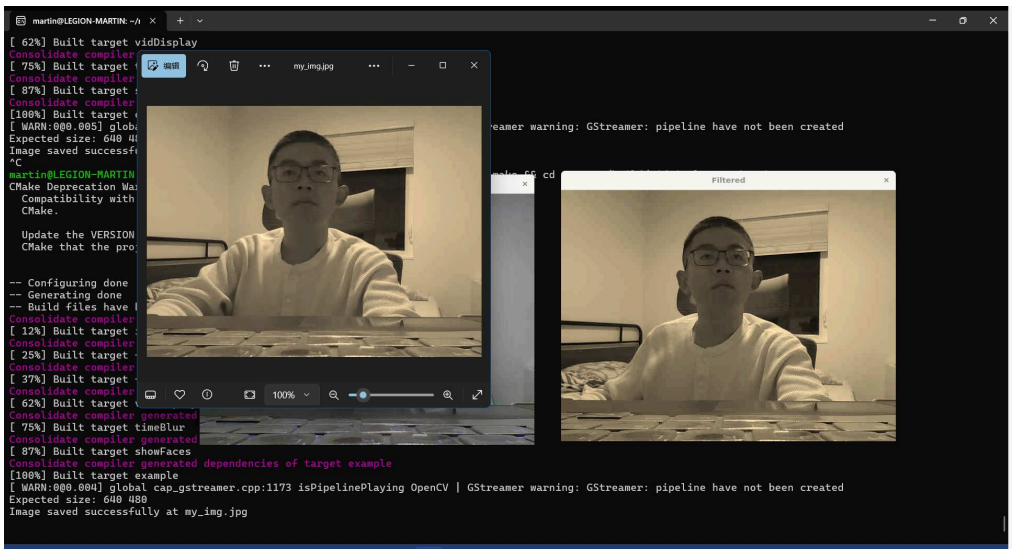
- `count_`: number of frames in the buffer to merge together.
- `alpha_`: weight of each image in the result image.
- merging algorithm:
  - `dst = dst * (1-alpha_) + frame[i] * alpha_`
  - If all frames are the brightest ([R, G, B] = [255, 255, 255]), and we take `count_` as 5, `alpha_` as 0.2, the result would be $255 - (1 - alpha\_)^{count\_-1} \times 255 \approx 153$, significantly darken the image.
  - Therefore, we take large enough `alpha_` and large enough `count_` will brighten the image up. However, if the `alpha_` is quite large, the result image will show resemblance to the original image, with weaker slow-motion effect. Therefore, we increase the `count_` parameter.
- A new `cv::Mat` is needed for every frame. In the original code, I declared the frame variable outside the `while` loop, resulting in all the frames in the buffer essentially pointing to the same memory area and not giving expected effects.

Result

- `count_`: 5, `alpha_`: 0.2: too dark.
  - 

- `count_`: 20, `alpha_`: 0.2: approximately the same brightness.
  - 

- ## Facial Improvement
  - ### Goal
    - Improve skin quality as well as brighten up the facial region.
  - ### Design
    - To improve skin quality, we blur the skin and keep the edges. The edges are detected using Sobel filter, while we use Gaussian filter implementation `blur_5x5` to blur.
    - To brighten the facial region, we convert the image from RGB color space to HSV, increase V value and convert back to RGB.
  - ### Result
    - 

    - The facial region is brightened to make features and details more visible. The edges are kept and the skin is polished. However, the style is not unified very well and creating a funny artistic effect.
- # Extension Task 1
  - ## Save Favorite Image
    - When users find good moments to capture, they can press `p` to save it.
    - The default path is `./img.jpg`, but user can send in customized path via command line argument as well.
    - 

Reflection

- In this project, I learned about setting up compilation environment involving OpenCV and some makefile basics. I also implemented several filters with knowledge obtained from lectures, which is a great way to review those details. And it feels so great to actually see the magnificent effect produced by code written with my own hands!
- LLM is helpful with debugging code and generating docs! Nice job!