CS100 - Homework7

Overview

This is a project about CMake, project management, Eigen and some basic matrix operations. Basically, It contains 3 parts. But note that they are **not independent** with each other. You should first finish part1, then finish part2 then part3.

Submission

- **Deadline: 12.18**
- In this homework, you will also use git to submit the project to *Geekpie OJ*.
- To encourage debugging locally and not on the OJ output (which is not right for practical programming), the OJ will not be available from the date the homework is released.
    - You can start submitting this project after **12.10**
    - All 3 parts will be judged together, so you will submit your whole directory and all 3 parts will be judged
    - Your git repo should contain a directory, named `HW7`, just like the one we give you.

Percentage of this homework over the whole score: 11%

Part 1 - Image IO using Eigen

In this part, you will work in file `src/image_io.cpp`, you should finish following 2 functions.

- **Image loadImage(const std::string & pathToRaw)**
  - You should open the file corresponds to `pathToRaw`, and read the content of it. The input file contains `m+1` lines, the first line contains two integer `m` and `n`, `m` is the number of lines, `n` is the number of columns of the matrix For the next `m` lines, each line contains `n` integers, which are the pixels of image. When load a image, you should divide each pixel by 255 E.g. if the `matrix_file[i][j] = 128`, then `image[i][j] = 128.0f/255.0f` For a sample, you can refer to `soccer.txt` in directory `data`
  - For example, if the input is:

  ```
  2 3
  10 20 30
  40 50 60
  ```

  Then, the returned matrix should be:

  ```
  0.03922 0.07843 0.11765
  0.15686 0.19608 0.23529
  ```

  - @param pathToRaw: The image matrix file path, eg: `../data/soccer.txt`
  - @return: The image object load from TXT file

- **void saveImage(Image & img, const std::string & pathToRaw)**
  - You should open/create the file corresponds to `pathToRaw`, and write the given image to it. The output file contains `m` lines, For the each line in `m` lines, it contains `n` floats, which are the pixels of image. Each float ranges from 0 to 1.
  - For example, if the `img` is

    ```
    0.03922 0.07843 0.11765
    0.15686 0.19608 0.23529
    ```

    You should directly write them into the file. That is, the output file should be the same as `img`, it should also contain following lines:

    ```
    0.03922 0.07843 0.11765
    0.15686 0.19608 0.23529
    ```

- @param img: The image object need to write.
- @param pathToRaw: The image matrix file path, eg: `../data/soccer_derx.txt`

Part 2 - Single-Threaded image filtering

Overview

This part consists of implementing a single-threaded kernel-convolution. Kernel convolutions are important in several domains of computer science, such as machine learning (deep learning, CNNs, *https://en.wikipedia.org/wiki/Convolutional_neural_network*) or image processing (*https://en.wikipedia.org/wiki/Kernel_(image_processing)*).

In general terms, the operation consists of generating an output matrix from a given input matrix of similar size. Let o_{rc} be the value of the element in row r and column c in the output matrix. It is given as a result of convolving the values around the same element in the input matrix A with a small kernel matrix K. Let $(2h + 1)$ and $(2w + 1)$ be the height and width of the kernel. The value of an element in the output matrix is then given as follows(you can refer to *https://en.m.wikipedia.org/wiki/Kernel_(image_processing)*):

$$o_{rc} = \sum_{i=0:2h} \sum_{j=0:2w} K_{i,j} * A_{r-h+i,c-w+j}$$

where A_{rc} denotes the element of the input A in row r and column c. The operation may also be rewritten in terms of an element-wise matrix multiplication (also called the Hadamard product, refer to *https://en.m.wikipedia.org/wiki/Hadamard_product_(matrices)*):

$$o_{rc} = sum(K * A(r-h, c-w, 2h+1, 2w+1))$$

where $sum()$ is a function that returns the sum of all the elements of the matrix parameter, $*$ denotes the Hadamard product, and $A(x, y, h, w)$ denotes the sub-block of A with top-left corner located at (x, y) and size hxw.

Your Job

Your job for this part is to implement 3 functions in `FilteredImage.cpp`.

- **FilteredImage::FilteredImage(Image & img)**
  - This is a constructor of `class FilteredImage`, which is defined in `FilteredImage.hpp` You should set the member variable `_img` as given parameter `img`
  - @param img: The image that will be filtered
- **void FilteredImage::applyKernel(Image & input, Image & output, Kernel & K)**
  - This is a member function of class FilteredImage, which is defined in `FilteredImage.hpp` You should do convolution operation for the input image That is, OUTPUT = INPUT * KERNEL, where `*` is the convolution operator
  - You can refer to the formula and image shown in the previous section.
  - @param input: The input image
  - @param output: The output image
  - @param kernel: The convolution kernel

- **Image & FilteredImage::get(int type)**
  - This is a member function of `class FilteredImage`, which is defined in `FilteredImage.hpp`
  - Here, you need to implement 4 types of kernel convolution job: `BLUR`, `DER_X`, `DER_Y`, `DER_MAG`. For each type, it corresponds to an int (you can see it in `types.hpppp`)
  - For `BLUR`, `DER_X` and `DER_Y`, we provide 3 different kernels for you to do that. That is, if you want the image filtered by `der_x`, just use `applyKernel(inImg, outImg, kernel_der_x)`, The kernels are shown below.
    - kernel_blur

```
Kernel K_blur(5,5);
K_blur << (2.0f/159.0f), ( 4.0f/159.0f), ( 5.0f/159.0f), ( 4.0f/159.0f), (2.0f/159.0f),
          (4.0f/159.0f), ( 9.0f/159.0f), (12.0f/159.0f), ( 9.0f/159.0f), (4.0f/159.0f),
          (5.0f/159.0f), (12.0f/159.0f), (15.0f/159.0f), (12.0f/159.0f), (5.0f/159.0f),
          (4.0f/159.0f), ( 9.0f/159.0f), (12.0f/159.0f), ( 9.0f/159.0f), (4.0f/159.0f),
          (2.0f/159.0f), ( 4.0f/159.0f), ( 5.0f/159.0f), ( 4.0f/159.0f), (2.0f/159.0f);
```

    - kernel_der_x

```
Kernel K_der_x(3,3);
K_der_x << -1.0f, 0.0f, 1.0f,
           -2.0f, 0.0f, 2.0f,
           -1.0f, 0.0f, 1.0f;
```

    - kernel_der_y

```
Kernel K_der_y(3,3);
K_der_y << -1.0f, -2.0f, -1.0f,
            0.0f,  0.0f,  0.0f,
            1.0f,  2.0f,  1.0f;
```

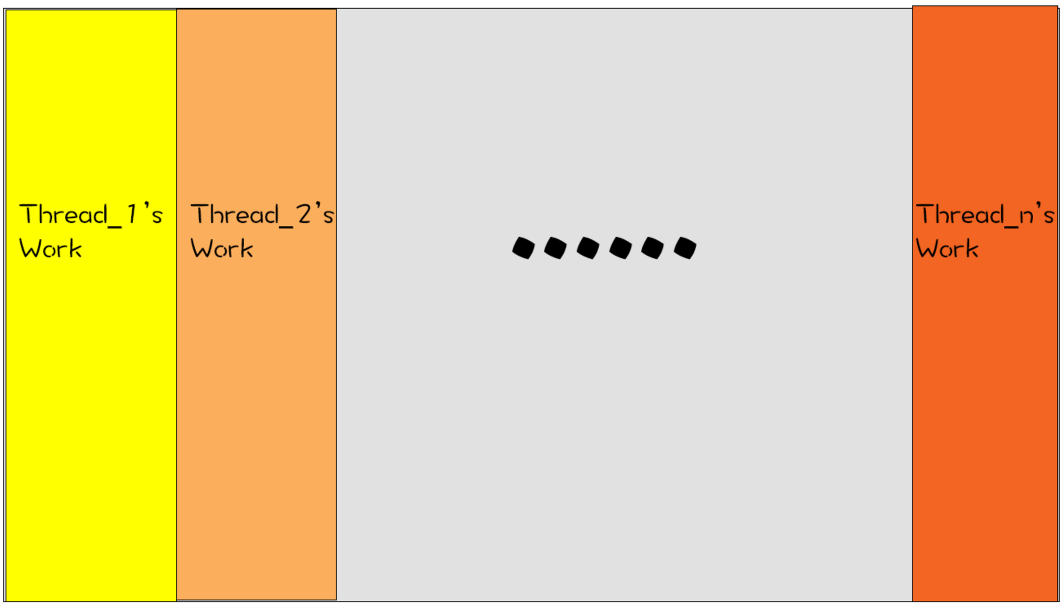
  - `DER_MAG` is a bit more difficult. You need follow thse steps (let `inImg` be the input image):
    - First, calculate the result filtered by `der_x` kernel (we name the result as `dxRes`)
    - Second, calculate the result filtered by `der_y` kernel (we name the result as `dyRes`)
    - Then, we can calculate the outImg by that formula: $outImg(i,j) = \sqrt{dxRes(i,j)^2 + dyRes(i,j)^2}$

- Here, we have a map called `_filteredImages`. The key of this map is an enumType(int), and the value of this map is a `shared_ptr<Image>`. when you call function `FilteredImage::get( int type )`, we should first check whether `_filteredImages[type]` exists.
  - If so (result doesn't exist), calculate the outImg by `applyKernel`, and let `_filteredImages[type]` points to your result, finally return your result.
  - Otherwise(if result exists), return the reference to the result saved in the `_filteredImages`.
- Hint: The 4 cases should be very similar.

Part 3 - Multi-Threaded image filtering

In this part, you will need to implement 3 functions in `ThreadedFilteredImage.cpp`, you need to divide the image into some fragments. For each thread, it should compute just one fragment.

The basic idea is shown in the following image:



The input Image

- **ThreadedFilteredImage::ThreadedFilteredImage(Image & img, int numberThreads)**
  - This is a constructor of class ThreadedFilteredImage, which is defined in `ThreadedFilteredImage.hpp`. You should set the member variable `_img` as given parameter `img`, You should set the member variable `_numberThreads` as given parameter `numberThreads`
  - @param img: The image that will be filtered
  - @param numberThreads: The number of threads to use
- **void ThreadedFilteredImage::applyKernel(Image & input, Image & output, Kernel & K)**
  - It's the similar as `applyKernel` in `class FilteredImage`, But you need to do that job using parallelism (using `# of threads = _numberThreads`)
  - You will found these functions / classes are useful:

- `std::vector<T>`, `std::thread`, `std::bind()`, `std::thread::join()`, `std::thread::joinable()`

- **void ThreadedFilteredImage::applyKernelThread(int startingCol, Image & input, Image & output, Kernel & K)**
  - For parallelism, you will need to divide the image into `k` different fragments, which `k` is the number of threads.
  - This function is each thread's job. The basic idea is, divide the image by column, It should start from `startingCol`, ends at `startingCol + (totalCol / numThreads)`. It's similar as `applyKernelThread` in `class FilteredImage`, but now, you should not convolute the whole image, just a part of that.

Debugging & Testing

- Thread Synchronization
  - Ask yourself whether or not there are any critical sections in the code?
  - You may find your program runs fine without adding any thread synchronization mechanisms, and you may try to think about why this is the case.
  - To think more about thread synchronization, that's very important for you (Not only for this HW), but also some further courses and projects, like Operating Systems /*PintOS*/.
- For local testing, you can follow these instructions:
  - Finish your code.
  - Go to your homework directory (you can use `cd` command to do this)
  - Create a directory `build`, and enter to it. (use `mkdir build && cd build`)
  - Use CMake to create a Makefile (use `cmake ..`)
  - Compile your project (use `make`)
  - If success, you can see a directory called `test` under `build`, enter it and you will see these files:
    - This is a sample contents under `build/test`

```
drwxr-xr-x  7 wuty  staff   224B 11 29 16:34 CMakeFiles
-rw-r--r--  1 wuty  staff   8.3K 11 29 16:34 Makefile
-rw-r--r--  1 wuty  staff   993B 11 29 16:34 cmake_install.cmake
-rwxr-xr-x  1 wuty  staff   176K 11 29 16:37 runfilters
-rwxr-xr-x  1 wuty  staff   221K 11 29 16:37 runfiltersTh
-rwxr-xr-x  1 wuty  staff    18K 11 29 16:39 testIO
```

  - Then, you can do debugging & testing
    - Use `./testIO <DIR_PATH> <FILE_NAME>` to test part1 (e.g. `./testIO ../../data soccer`)
    - Use `./runfilters <DIR_PATH> <FILE_NAME>` to test part2 (e.g. `./runfilters ../../data soccer`)
    - Use `./runfiltersTh <DIR_PATH> <FILE_NAME>` to test part3 (e.g. `./runfiltersTh ../../data soccer`)
- To transform image to TXT file, you can use a MATLAB script (`data/original_images/transform_image.m`).