CSE 2050 - Programming in a Second Language (C++) Mini Project 1

October 1, 2014

1 Due Dates

- **Monday, October 9, by 11.59pm**

2 General information

- Submit only the source files and the associated makefile. A single makefile should build all programs (i.e., multiple targets).
- Submit a single ".tar.gz" file containing the directory with all programs (and the makefile). To compress the directory, type the following command on the linux terminal:

```
tar -zcvf archive_name.tar.gz directory_to_compress
```

3 Project Summary

In this project, you will implement a simple video-processing program. Your program will load a video into memory and perform a set of processing tasks on it. The program will display a menu to the user with the following options:

1. Load video
2. Save video
3. Save an image frame
4. Delete an image frame
5. Calculate the average image frame

Instead of using a standard video format such as `.mpg` or `.avi`, videos in this assignment will be represented by a sequence of images. Image sequences will be provided in a directory. The name of the directory is the name of the video, i.e., a video called `surprise` is stored as an image sequence in a directory named `surprise/`.

The entire video (i.e., all images in the sequence) must be kept in memory. The data structure for storing the video will be a singly linked list. You can also use a doubly linked list if you want. Each node of the list is an image (i.e., video frame). You are expected to implement all components of the linked list, i.e., from allocation to inclusion of nodes, traversing the list, and deletion of nodes.

Memory for the nodes must be dynamically allocated (i.e., not an array). The basic image structure looks like:

```
struct Image {
    string fileName;
    unsigned char pixelmap[418][312];
    unsigned int width;
    unsigned int height;
};
```

The basic node for the singly linked looks like:

```
struct Frame{
    Image data;
    Frame *next;
};
```

Because storing and processing videos are memory-demanding tasks, your implementation should avoid duplicating memory as much as possible (e.g., copying). You should also choose data types that fit the the nature (and range) of the data (e.g., use `unsigned int` for matrix dimensions). While you will not use use classes or object-orientation design in this assignment, your implementation should use C++ (e.g., you should use `cout` not `printf`, `new` not `malloc`).

Coding issues pointed out in the feedback provided in previous assignments as well as during lectures will be considered for grading purposes. For example, all functions should be commented and main steps clearly indicated. Error checks for memory allocation, file operations, and menu options should be implemented using newer standards (and style). User input should also be checked and handled accordingly. Other problems that will lead to deductions in grade include memory leaks, dangling pointers, and poor coding style.

4 Image Format

Images will be saved in plain ascii format. This is a very simple image format. The format consists of a three-line header field followed by the image data (i.e., pixel values). An example of a small image is given below:

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

where:

- First line: Magic number P2 that identifies the type of image.
- Second line: The width and height of the image.
- Third line: The maximum pixel value in the image.
- Each pixel in the raster has white space before and after it. There must be at least one character of white space between any two pixels, but there is no maximum.
- No line should be longer than 70 characters.
- There is a newline character at the end of each of these lines.

5 Background on image and videos

5.1 Images and their representation

Computationally, an image is usually represented by a two-dimensional matrix (of numbers). Here, each element of the matrix is assigned a numerical value representing the brightness of the point. These points are called *pixels* (i.e., pictorial elements). In general, images can be of two types: *graylevel* or *color*. Graylevel images look like “black-and-white” pictures, which use shades of gray to represent the brightness of the scene points reflected into the photographic camera. Computationally, a graylevel image is represented by a single numerical matrix while a color image is represented by three numerical matrices accounting for three basic color componentes red, green, and blue (RGB). To display a color image, the computer combines the values of the red, green, and blue components to produce the final color to be visualized on the monitor.

In this project, we will use graylevel images for their simplicity. For an example of a gray-level image, consider a white “square” on top of a gray square, both on a black background as shown in Figure 1.

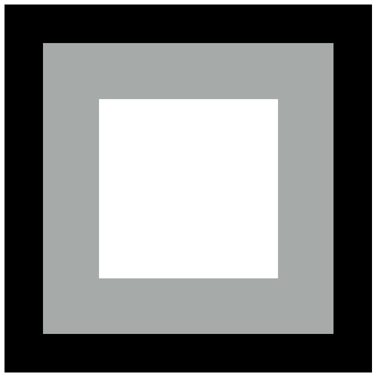

Figure 1: An example of a gray-level image.

If we assume the image in Figure 1 has a size of 8*x*8 pixels, then it can be represented by the following matrix:

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 100 & 100 & 100 & 100 & 100 & 0 \\ 0 & 100 & 255 & 255 & 255 & 255 & 100 & 0 \\ 0 & 100 & 255 & 255 & 255 & 255 & 100 & 0 \\ 0 & 100 & 255 & 255 & 255 & 255 & 100 & 0 \\ 0 & 100 & 255 & 255 & 255 & 255 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{1}$$

Each number (i.e., pixel value) in the matrix I, is a brightness value. Mathematically, we represent the value of individual pixels by $I(i, j)$, where index i is the row and index j is the column in the matrix (i.e., pixel coordinates). It is common for pixels to be assigned brightness values in the range from 0 to 255 (i.e., 256 shades of gray).

5.2 Videos

We can think of videos as a sequence of images of same dimension. We can also think of videos as a volume containing many planes, each plane is an image (i.e., matrix) captured at a specific instant in time. Figure 2 shows a sequence of images of a facial expression as 3-D volume (with some transparency).

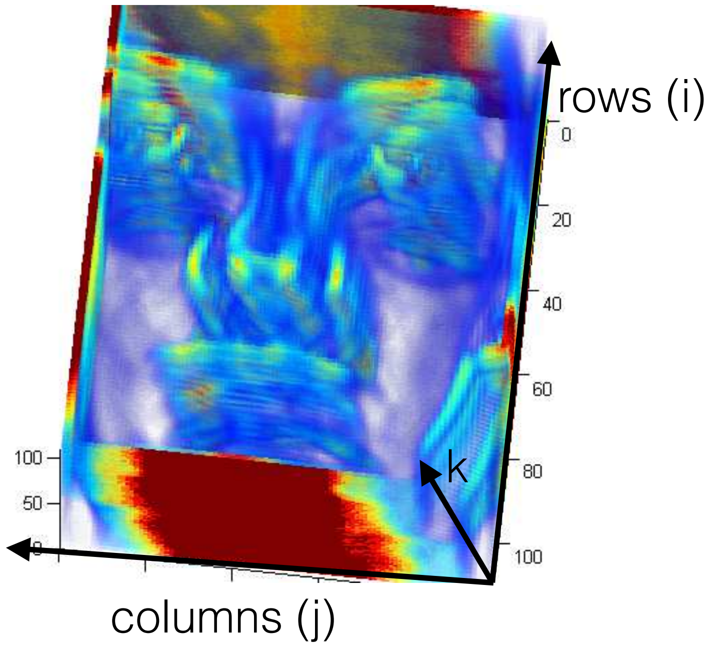

Figure 2: A video of the expression "smiling" shown as a 3-D volume stored in a 3-D matrix. Transparency is applied for better visualization.

5.3 Calculating the average image

Consider a video $V = \{I_1, I_2, \ldots, I_K\}$ containing K images (or frames). The average image is the average of all images in the sequence of images. The average is calculated pixel by pixel. For example, if the image sequence is represented by a 3-D matrix $V_{M\times N\times K}$ then the average image is given by:

$$avg(i,j) = \frac{1}{K}\sum_{k=1}^{K} V(i,j,k). \tag{2}$$

Figure 3 illustrates the calculation.

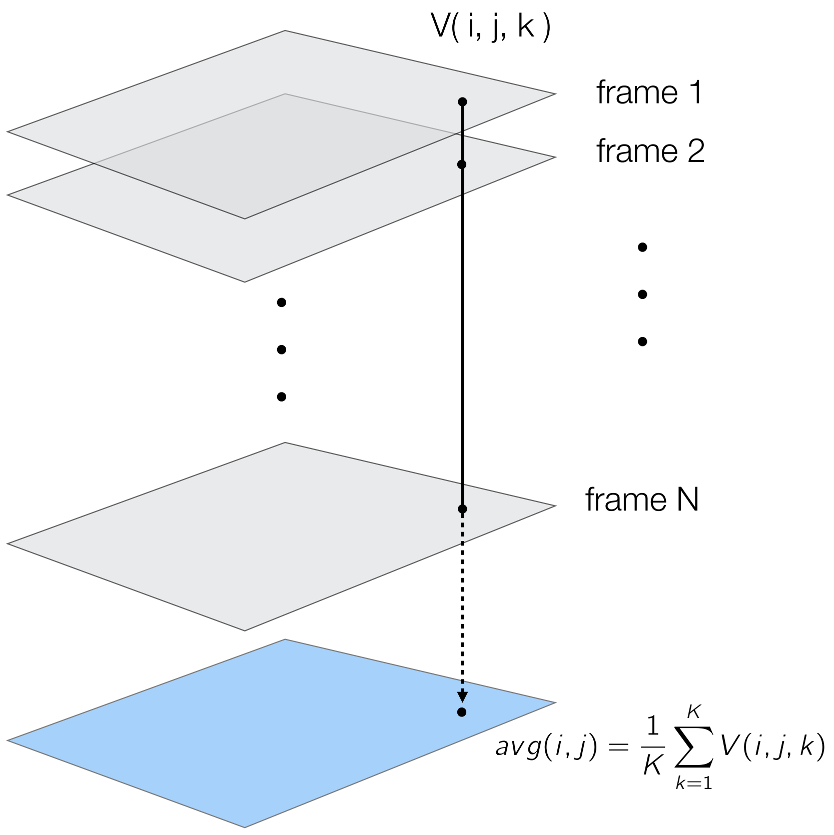

Figure 3: Generating the average frame. First, consider the video as a stack of images (i.e., a 3-D volume). Then, calculate the average value over the time dimension, k, for each pixel location (i,j). The resulting matrix is the average image.

A toy example of the average calculation for three matrices is shown below. The matrices represent small images.

$$\begin{aligned} I_1 + I_2 + I_3 &= \begin{bmatrix} 1 & 3 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 4 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 9 \\ 8 & 3 \end{bmatrix} \\ &= \begin{bmatrix} \frac{(1+1+1)}{3} & \frac{(3+1+9)}{3} \\ \frac{(4+4+8)}{3} & \frac{(3+0+3)}{3} \end{bmatrix} \\ &= \begin{bmatrix} 1.0 & 4.3 \\ 5.3 & 2.0 \end{bmatrix}. \end{aligned} \tag{3}$$