ECE454, Fall 2017
Homework2: Memory Performance
Assigned: Sept 21st, Due: Oct 13th, 11:59PM

The TAs for this assignment are: Jack Luo (jack.luo@mail.utoronto.ca)
and Michelle Wong (michelley.wong@mail.utoronto.ca)

1 Introduction

After great success with your previous client, OptsRus has a second client: a virtual reality headset startup. The startup is co-founded by a group of hardware geeks: those who like to design electrical circuits and integrate sensors. The VR headset prototype hardware is almost ready but lacks a high performance software image rendering engine. The hardware engineers have already written functionally correct code in C, but need your help to improve the software performance and efficiency.

The rendering engine's input is a preprocessed time-series data set representing a list of object manipulation actions. Each action is consecutively applied over a 2D object in a bitmap image such that the object appears moving in reference to the viewer. In order to generate smooth and realistic visual animations, sensor data points are sampled at 1500Hz or 25x normal screen refresh rate (60 frames/s).

Figure 1 shows all of the possible object manipulation actions. Your goal is to process all of the basic object manipulation actions and output rendered images for the display at 60 frames/s.

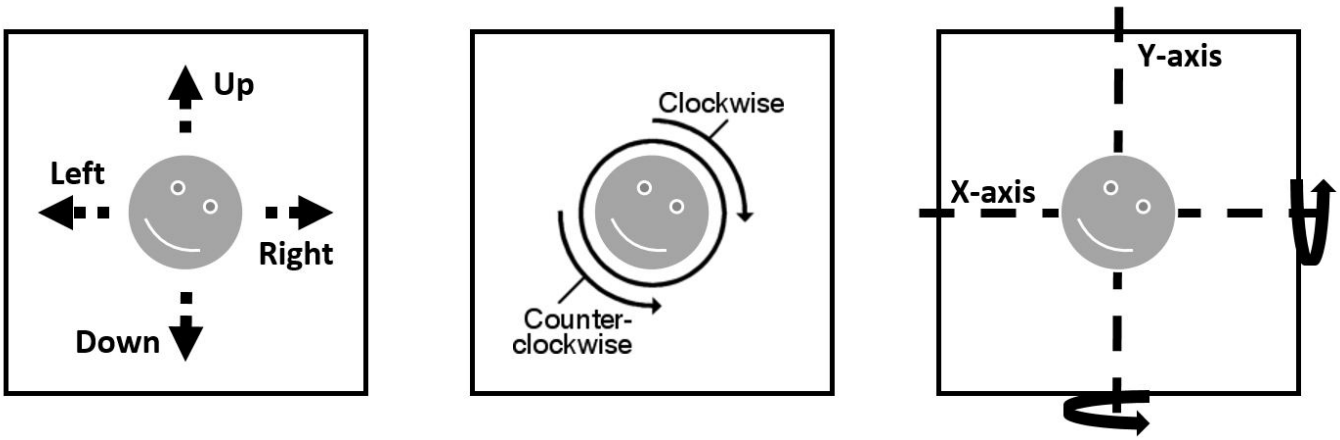

Figure 1: Basic Actions

2 Implementation Overview

Fundamentally, the rendering engine you are asked to optimize is very simple in design. This section will briefly describe all important parts you need to know for this lab. A well-documented trivial reference implementation is

provided to help you get started.

2.1 Input Files

There are two files used as input. The first one is a standard 24bit bitmap image file with a square dimension. The second one is a list of processed sensor values stored inside a csv (comma separated value) file.

For the bitmap image file, white pixels `(RGB=255,255,255)` is considered as the background and non-white pixels are considered as part of an object. You can generate your own image files using Microsoft Paint in Windows or GIMP on linux. After drawing your own square image, export the bitmap to 24bit bitmap format. If you are using GIMP, make sure under the compatibility options, check "Do not write color space information" option. Then under advanced options, select "R8 G8 B8" under 24 bits. We packaged a tiny 10 x 10 pixel bitmap image named `object_2D.bmp` in the lab assignment package which helps you get started.

The list of processed sensor values can be viewed as a list of key-value pairs. The key represents a basic object manipulation action and the value is the magnitude of the specified action. An example sensor value input file is shown below:

```
W,6     // shift object up by 6 pixels
A,5     // shift object left by 5 pixels
S,4     // shift object down by 4 pixels
D,3     // shift object right by 3 pixels
CW,2    // rotate entire image clockwise by 180 degrees
CCW,1   // rotate entire image counter clockwise by 90 degrees
MX,1    // mirror entire image along the X-axis in the middle
MY,0    // mirror entire image along the Y-axis in the middle
```

2.2 Data Structures

Frame Buffer and Dimension

The input bitmap image has already been parsed for you. The image pixel data is stored into the following data structure below:

```
unsigned char *frame_buffer;     // [R1][G1][B1][R2][G2][B2]...
unsigned int width, height;      // dimension in number of pixels
```

Sensor Values

The processed sensor values input file has also been parsed for you and stored inside a key-value pair array. The array has enough storage for more than 10,000 key-value pairs. As mentioned earlier, the key represents the basic object manipulation action and the value represents the magnitude. It is stored in the following data structure below:

```
// KV Data Structure Definition
struct kv {
char *key;
int value;
};

// KV Data Structure Array
struct kv sensor_values[10240];
```

2.3 Key Function and Definitions

In this lab, you are only allowed to modify a single file (`implementation.c`). Inside this file, there are three important function which you should take extra precaution.

print_team_info()

This method is used to print out the team information to the stdout. It is called upon startup of the program. These information is used by the auto-marking system. We suggest you to modify them before starting the lab.

The content of the method is the following:

```
// Please modify this field with something interesting
char team_name[] = "default-name";

// Please fill in your information
char student1_first_name[] = "john";
char student1_last_name[] = "doe";
char student1_student_number[] = "0000000000";

// Please fill in your partner's information
// If yon't have partner, do not modify this
char student2_first_name[] = "joe";
char student2_last_name[] = "doe";
char student2_student_number[] = "0000000001";
```

implementation_driver()

This method is the main entry point to your code. All of the available data is passed to your implementation via this function. You should not modify the prototype of this function. Currently, a naive but working solution of the lab inside `implementation_driver()` is provided to help you get started. However, you are free to modify the implementation of this function as well as modify and delete anything else in this file except for the `print_team_info()` function mentioned above. Please make sure the `implementation_driver()` function is always reachable from main.

The prototype of the function is the following:

```
void implementation_driver(
   struct kv *sensor_values, int sensor_values_count,
   unsigned char *frame_buffer,
   unsigned int width, unsigned int height,
   bool grading_mode
);
```

verifyFrame()

You must call this function for each frame you are required to output. Before you call this function, please make sure that you pass in valid data of the correct type. **Failing to do this step will generate errors in the program, thus failing the implementation verification check.** The prototype of the function is the following:

```
void verifyFrame(unsigned char *frame_buffer,
                 unsigned int width, unsigned int height,
                 bool grading_mode);
```

3 Performance Measurement of the Tool

Perf Tool

To gain insight into the cache behavior of your implementation, you can use the `perf` infrastructure to access the hardware performance counters of the processor. For example, to output the first-level cache misses generated by your program `foo` you would execute:

```
perf stat -e L1-dcache-load-misses foo
```

You can view a listing of all performance counters that you can monitor by running:

```
perf list
```

Note that you can monitor multiple counters at once by using multiple `-e` options in one command line. `perf` has many other features that you can learn more about by browsing:

```
perf --help
```

For example, you can consider monitoring TLB misses or other more advanced events. A small write-up on `perf` is available here:

`http://www.pixelbeat.org/programming/profiling/`

Unfortunately there is not a lot of documentation on `perf` yet as it is so new, but the "help" information is clear.

Gprof & GCov

If you have successfully completed lab1, you should be familiar with these two tools. They can be very useful in pin-pointing the bottleneck of your program.

Note: To configure these tools to use within your project, you will need to provide additional cmake commandline options while generating the make file. A quick example can be found on Stackoverflow for Gprof integration.

4 Team Information

Important: Before you start, you should fill in the requested information in `print_team_info()` with information about your team (group name, first names, last names, student numbers). Each team should consist of **2** students unless an exception is allowed by the instructor.

5 Setup

5.1 Initial Setup

Start by copying the `hw2.tar.gz` file from UG shared directory
`/cad2/ece454f/hw2/hw2.tar.gz`
into a protected directory within your **UG** home directory.

Then run the command:
`tar xzvf hw2.tar.gz`
This will cause a number of files to be unpacked into the directory.
The **ONLY** file you will be modifying and handing in is `implementation.c`.
You are **prohibited** to modify other files.

Looking at the file `implementations.c`, you need to insert the requested identifying information about the individuals comprising your programming team.
Do this right away so you don't forget.

5.2 Compilation

The lab assignment utilizes the open-source cross-platform CMake packaging system to manage the source code. Unlike the simple projects you have seen before, the Makefile is automatically generated based on your computer configuration.

Below are the instructions to compile the project:

```
> cd <project directory>     // Navigate to the lab assignment directory you extracted
> mkdir bin && cd bin        // Make a new directory called bin, then navigate inside
> cmake ../                  // Use cmake to generate the Makefile automatically
```

After the simple configuration steps, the make file is automatically generated. Simply run the Makefile and an executable named `ECE454_Lab2` should appear within the `bin` folder.

5.3 Coding Rules

The coding rules are very simple. You may write any code you want, as long as it satisfies the following:

- Your submission contains only the modified implementation.c file and no other files.
- The code you write must conform to the C99 standard. This is an artificial limitation to prevent you from using advanced constructs and optimizations available in C++11 or C++17. You are not allowed to modify the `cmakelist.txt` file, and as a result, you will not be able to bypass this limitation.
- You must not interfere or attempt to alter the time measurement mechanism.
- Your submitted code does not print additional information to stdout or stderr.
- You cannot use multiple threads. It would be hard anyways since you are not allowed to modify the Makefile.
- You are not allowed to embed assembly into your program. If you have that much time, try optimizing C code instead.
- You are not allowed to use `#pragma` compiler directives

6 Evaluation

The lab is evaluated when the grading mode is turned on. The grading mode previously mentioned in many sections is controlled via an additional flag `-g` option flag via command line (see example terminal output below). The evaluation turns on instrumentation code which measures the total clock cycle used by the `implementation_driver()` function. When you evaluate your implementation using the command below, you should receive similar output.

```
/home/user/ECE454/hw2/bin/ECE454_Lab2 -g -f sensor_values.csv -i object_2D.bmp
Loading input sensor input from file: sensor_values.csv
Loading initial 2D object bmp image from file: object_2D.bmp
********************************************************************************
Team Information:
   team_name: default-name
   student1_first_name: john
   student1_last_name: doe
   student1_student_number: 0000000000
   student2_first_name: joe
   student2_last_name: doe
   student2_student_number: 0000000001
********************************************************************************
Performance Results:
   Number of cpu cycles consumed by the reference implementation: 124374
   Number of cpu cycles consumed by your implementation: 125073
   Optimization Speedup Ratio (nearest integer): 1
********************************************************************************
```

6.1 Marking Scheme

The total available marks are divided into 2 portions (60% + 40%). The first portion is non-competitive and designed to allow people who gave minimal effort to pass the course. The second portion is competitive and we would like to award marks proportionally to the team effort.

Non-Competitive Portion - 60%

The non-competitive portion should be fairly easy for everyone who puts in the minimal acceptable amount of effort. The TA will be taking notes of all attendee's performance speedup during 1st week of tutorial session for this lab. After the end of the first week, we will establish a speedup threshold based on either 1. the **average speed up of the class** after the first week calculated and posted on Piazza or 2. **a cut-off threshold posted by the instructors** on Piazza if the class average speedup cannot be calculated or is considered by us to be either unexpectedly high or too low. If you can achieve this minimum level of performance improvement, the TA will assign full marks to you for this portion.

Competitive Portion - 40%

This lab is designed to have very high potential for performance optimization. One can easily achieve at least 50x performance speedup compared to the reference solution. For this portion, we will be using an automated scoring system. Once you submit your work using the usual `submitece` command, your submission will be placed onto a queue for auto-grading.

The marks will be assigned with this formula:

mark = (your speedup - worst speedup) / (top speedup - worst speedup)

The competitive portion of the mark should be available via a web portal within maximum of 24hrs. In most cases, your updated score should be reflected on the web portal automatically within a very short period of time. TAs will announce the scores daily labeled with team-names on piazza.

Note: Web portal access information will be released on Piazza one week after the lab is released. Before portal access is enabled, the TA may instruct the class to submit their code in order to be able to evaluate performance e.g., because the TA cannot access your directory to run your code directly. This will be posted on piazza. In case you are asked to submit for early evaluation, you will always be able to resubmit your code later.

7 Assignment Assumptions

In the auto-grader, you can assume the following points when writing your algorithm:

1. Object will always be visible and never shifted off the image frame.
2. Don't need to output incomplete frames (image frame composing of less than 25 object manipulating actions)
3. Solution will only need to handle square image with size up to 10,000 pixels in width and height.
4. Solution will only need to perform maximum of 10,000 sensor value inputs.
5. All solutions will be tested on the same dedicated machine. The dedicated machine is either one of the ug lab machines with Intel Core i7 4790 CPU or a machine with very similar hardware. Therefore, we recommend that you run and evaluate your code on one of the ug lab machines of this type. The grades will be assigned via performance measurement generated on the dedicated machine.

8 Logistics

You should work in a group of up to **two** people in solving the problems for this assignment. Any clarifications and revisions to the assignment will be posted on the course piazza page.

9 Submission

When you have completed the lab (including if you are asked to submit for early evaluation), you will submit only the file `implementation.c` that contains your solution. **Submit only from one partner's ug account**. If there are two submissions, a **random partner's submission will be selected**.

The standard procedure to submit your assignment is by typing
`submitece454f 2 implementation.c`
on one of the UG machines.

- Make sure you have included your identifying information in the `print_team_info()` function.
- Remove any extraneous print statements.

10 Bonus Marks!

For the top 3 teams which have ranked top 3 for more than 5 cumulative days on the web portal, each team will receive 1% after the 5th day. For the top team only, at the very end, they will be awarded an additional 1% marks.

The goal of this lab is to fully utilize your knowledge to optimize for memory performance. However, if you are really willing to learn and explore additional concepts to achieve maximum performance, you may look into AVX or SSE instructions. You can **submit an additional version** of assignment utilizing SIMD instructions (`#pragma` allowed for this version only) **to the TA directly**, by email including your source code for additional up to 3% bonus marks for your effort. You will still need to submit a version following the standard coding rules via the standard `submitece` command for normal evaluation.