Assignment 1: Transformations
CS180/CS280 Fall 2022

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on September 30th, 2022 (Friday)

Due at 11:59PM Oct 6th, 2022 (Thursday)

Notes:

- Be sure to read the university's Academic Integrity policy.
- Any updates or correction will be posted on EdStem, so check there occasionally.
- You must do your own work independently.

---

1 C++ Review and Eigen Introduction

We have provided code skeleton in `main.cpp`. Before getting into the assignment, here is a brief review of C++ and a short introduction to Eigen library.

1.1 Developing Tools

We recommend you to use Visual Studio Code (VS Code) as the editor and compile and run your codes in the terminal.

1.2 C++ 101

This subsection provides some basic knowledge about C++ that is relevant to the course assignments. If you want to learn more, please go to `https://devdocs.io/cpp/` or Stack Overflow.

1.2.1 Headers

C++ adopts the convention of using header files to contain declarations. You make the declarations in a header file, then use the `#include` directive in every `cpp` file or other header files that require that declaration. The `#include` directive inserts a copy of the header file directly into the `cpp` file before compilation.

In practice, you can include additional libraries by using `#include`:

```cpp
#include <cmath>
#include <iostream>
```

The above codes include the necessary libraries for C++ input/output and mathematical calculations.

1.2.2 Functions

A function is a block of code that only runs when it is called. The function named with **main** is the entry point of a program.

```cpp
int main() {
    float a = 1.0, b = 2.0;
    std::cout << a << std::endl;
    std::cout << a/b << std::endl;
    std::cout << std::sqrt(a) << std::endl;
    std::cout << std::acos(-1) << std::endl;
    std::cout << std::sin(30.0/180.0*acos(-1)) << std::endl;
    return 0;
}
```

The above program outputs the following calculation results: a, $\frac{a}{b}$, $\sqrt{a}$, $\arccos(-1)$, $\sin(30°)$, where $a = 1$ and $b = 2$, and exits safely.

1.2.3 Common Errors

- Compile Error: try to solve it based on the error message. If you cannot solve it by yourself, you can search the error message on Stack Overflow to find similar cases.

- undefined reference to xxx: usually linking errors. Check if the function is declared in the header file, but has not been implemented in the `cpp` file. Or check the linking configurations in `CMakeLists.txt`.
- Segmentation Fault: usually caused by index out of bounds, too much stack usage.
- Bus Error: the causes are usually similar to the causes of the segmentation fault.
- Math Error: usually caused by dividing it by 0.

1.3 Eigen

This course uses Eigen as the C++ library for linear algebra. Its official documentation can be found at `http://eigen.tuxfamily.org`. We have installed Eigen for you in the `vdi` file, so you do not need to worry about the installation.

1.3.1 Headers

In order to use Eigen in your project, it needs to be included:

```
#include <Eigen/Core>
```

1.3.2 Vectors and Matrices

This part only provides an overview of vectors and matrices operations in Eigen. For a more thorough explanation, please refer to `https://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html`.

```
    // Example of vectors
    std::cout << "Example of vectors \n";
    // vectors definition
    Eigen::Vector3f v(1.0f,2.0f,3.0f);
    Eigen::Vector3f w(1.0f,0.0f,0.0f);
    // vectors output
    std::cout << "Example of output \n";
    std::cout << v << std::endl;
    // vectors addition
    std::cout << "Example of addition \n";
    std::cout << v + w << std::endl;
    // vectors scalar multiplication
    std::cout << "Example of scalar multiplication \n";
    std::cout << v * 3.0f << std::endl;
    std::cout << 2.0f * v << std::endl;
```

The above code shows the definition, output, addition, and scalar multiplication of 3D floating-point vectors. Please try computing the dot product of two vectors on your own.

```
    // Example of matrices
    std::cout << "Example of matrices \n";
    // matrices definition
    Eigen::Matrix3f i,j;
    i << 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0;
    j << 2.0, 3.0, 1.0, 4.0, 6.0, 5.0, 9.0, 7.0, 8.0;
    // matrices output
```

```
    std::cout << "Example of output \n";
    std::cout << i << std::endl;
    // matrices addition i + j
    // matrices scalar multiplication i * 2.0
    // matrices multiplication i * j
    // multiply a matrix and a vector i * v
```

This sample provides the definition and output of matrices. Please explore the usage described in the comments above.

2 Skeleton Code Compilation and Execution

2.1 Installation of Eigen

If you have bash, run `install.sh` found in the assignment1 folder.

If not, do the following:

1. Download Eigen Library `https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.zip`.
2. Extract the Eigen files to `/path/to/assigmnment1/external/eigen-3.4.0`. You will use this path with `g++` or CMake to compile your code.

2.2 Compilation with g++

We have provided skeleton code in `main.cpp` for you to work on. Open a terminal in the folder that contains this `main.cpp` file and run the commands to test `Eigen` installation. Please make sure that the `g++` is referring to the correct path to `Eigen`.

```
g++ -I ./external/eigen-3.4.0 main.cpp -o assignment1
./assignment1
```

2.3 Compiling with CMake

We have provided skeleton code in `main.cpp` for you to work on. Open a terminal window under the folder that contains `main.cpp`, and run the following commands to compile and run your program:

```
# Make a folder named build, and enter it. This is the folder that stores all the compiling and linking results.
mkdir build; cd build

# .. means the parent directory. It's the directory containing CMakeLists.txt. This step generates a Makefile using cmake.
cmake ..

# Compile your program. 'Make' will read the Makefile by default. Errors will show up in the terminal if there is something wrong.
make

# Run your program. 'assignment1' is the name of the executable file. You can change it in the CMakeLists.txt.
```

```
./assignment1

# Delete all the compiled results before submission.
cd ..
rm -r build
```

3 Problem Set

For problems 2, 3 and 4 go to `main.cpp` provided with the assignment. Add codes in appropriate places.

1. (5 points) Describe what this 2D homogeneous transform matrix does for a point: $\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$

2. (5 points) Write the 3×3 transformation matrix for a $45°$ **clockwise** rotation in 2D (assuming homogeneous coordinates). Populate the `rot_45` matrix variable using `<<` operator in `main.cpp`.

3. (5 points) Write the 4×4 transform matrix to move a point by $(-2, 8, 3)$. Populate the `translation` matrix variable using `<<` operator in `main.cpp`.

4. (20 points) In computer graphics, we often need to map points in one rectangle to a new rectangle area. Suppose the bottom left corner and top right corner of original rectangle are $(1, 5)$ and $(4, 8)$. The bottom left corner and top right corner of new rectangle are $(2, 0)$ and $(8, 1)$. This can be achieved by a sequence of three steps:

   (a) (5 points) Move point $(1, 5)$ to the origin.

   (b) (5 points) Scale the rectangle to be the same size as the target rectangle.

   (c) (5 points) Move back points to new position.

   Populate the matrix for each step (a,b,c), and the multiplication result of these matrices in the correct order (d) in `main.cpp`.

4 Submission

Please submit **ONLY ONE** zip file on gauchospace containing your project (specifically, `CMakeLists.txt` and `main.cpp`) and a report no more than one page.

The report should contain the answer to problem 1, and the matrices populated in problem 2, 3, and 4. You can either write them down, or show the screenshot of your program's output.

Make sure the zip file contains **NO** compiled results, for example, you should **NOT** include the `build` folder into zip file.