Assignment 2: Rotation and Projection
CS180/CS280 Fall 2022

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Oct 7th, 2022 (Friday)

Due at 11:59pm on Oct 13th, 2022 (Thursday)

Notes:

- Be sure to read the Programming and Collaboration Policy on course website.
- Any updates or correction will be posted on EdStem, so check there occasionally.
- You must do your own work independently.
- Read through this document carefully before posting questions on EdStem.
- Submit your assignment on GauchoSpace before the due date.

---

1 Overview

In the lecture so far, we have learned how to use matrix transformations as a tool to arrange objects in 2D or 3D space. Now it's time to get some hands-on experience by implementing a few simple transformation matrices. In the next three assignments, we will ask you to simulate a simplified version of CPU-based rasterization renderer.

The task of this assignment is to fill in a rotation matrix and a perspective projection matrix. Given the 3D world coordinate of three vertices, v0(1.0,0.0,-2.0), v1(0.0,2.0,-3.0), v2(-2.0,0.0,-2.0), you should be able to transform them into screen space coordinate and draw the wireframe triangle on the screen. (We provide you the *draw_triangle* function, you only need to construct the transformation matrices). In short, we need to do the model, view, projection, viewport transformation to see a triangle on our screen. The skeleton code left the model and projection transformation parts to you. If you are unclear about any concept above, review the lecture notes and post questions on EdStem.

Functions that you need to modify are located in the main.cpp file:

- **get_model_matrix(float rotation_angle):** Set the model transformation matrix element by element, and return the matrix. Here, you should only implement 3D rotation that's the result of rotating the triangle around Z axis. You don't need to handle translation and scale.
- **get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar):** Set the perspective projection matrix element by element with the given parameters, and return it.
- **[Optional] main() or other location:** Provide other operations you need.

Don't change the signature of any of the functions, and other implemented functions. It is your responsibility that your code compiles and runs without problems.

Inside these functions, create the corresponding model and projection matrices, and return your matrix. When you get the transformations correctly, you will see a triangle that's drawn in wireframe.

The frames are rendered and drawn one by one. When you get the transformations, the rasterizer will create a window showing you the triangle. Then, you can use **A** and **D** keys to rotate the triangle around the Z axis. If you press the **Esc** key, the window will be closed and the program will terminate.

It's also possible to use the program from the command line. You can run it using the following command to run and pass the rotation angle to the program. In this case, there won't be any windows, but the program will save the output image to the given filename (or as **output.png** if no name is specified from the command line). The location of the new created image will be right next to the executable. So if your executable is in a **build** folder, the image will be inside this folder.

The command line can be used like these:

```
./Rasterizer  // Runs the program in a loop, creates the
              // window and you can rotate the triangle
              // with A and D keys.

./Rasterizer -r 20 image.png // Runs the program, rotates the
                             // triangle 20 degrees, and saves
                             // the output to a file named
                             // image.png
```

2 Getting started

You can skip this section if you have already installed the virtual machine we provided from the second attachment.

After downloading the skeleton code, you have two options to run it:

- Install all the dependencies, build and run the skeleton code on your local system.
- Install VirtualBox or UTM, download and import our pre-configured Ubuntu virtual machine. See the second attachment for more details.

2.1 Own system

First install CMake either from the source (https://cmake.org/install/), or from **apt** using "**apt install cmake**" if you haven't already. CMake is an open-source, cross-platform tool to build software packages.

Then install the two libraries we use:

- Eigen: For linear algebra operations.
- OpenCV: For image manipulation and display.

After that, it's time to build the project with CMake, a lot of modern IDEs can take care of this process automatically.

```
mkdir build    // Create a folder to keep the build artifacts
cd build       // Go into the build folder
cmake ..       // Run CMake, by giving the path to the
               // CMakeLists.txt file as argument
make -j4       // Compile your code by make, -j4 parallelizes
               // the compilation through 4 cores
./Rasterizer   // Run your code
```

3 Skeleton code structure

In this assignment, you don't need to use the triangle class. So the files you need to understand and modify are: rasterizer.hpp, main.cpp. The **rasterizer.hpp** file provides the interface for the renderer, and handles the drawing.

The rasterizer class provides the important functionality in our system.
Member variables:

- Matrix4f mode, view, projection: Three matrices for our transformation.
- `vector<Vector3f> frame_buf`: Frame buffer object, which stores the color data to be dumped onto the screen.

Member functions:

- set_model(const Eigen::Matrix4f& m): Sets the internal **model** matrix to the parameter. This function is going to be used to pass your matrices into the rasterizer.
- set_view(const Eigen::Matrix4f& v): Sets the view transformation matrix to the internal **view** matrix.
- set_projection(const Eigen::Matrix4f& p): Sets the internal **projection** transformation matrix to the given matrix p. This function is used to set your projection matrix to the rasterizer.
- set_pixel(Vector2f point, Vector3f color): Set the screen pixel point(x,y) to color (r,g,b). Write into the corresponding frame buffer position.

In the main.cpp we simulate the graphics pipeline. We define an instance of the rasterizer class first, and set up the necessary variables of it. Then we are given a hard coded triangle with three vertices (Don't change it.). Above the main function, there are 3 function definitions that each calculate model, view and projection matrices. Each of these functions return the corresponding matrix. Then, the return value for these functions are passed into the rasterizer using the **set_model()**, **set_view()** and **set_projection()** functions. Then, in each frame the rasterizer handles the transformations.

After transforming the given geometry with model, view, projection matrices, we end up with three vertices in normalized device coordinates (NDC). NDC has the x, y, and z coordinates range over [-1,1]. The next step is to do the viewport transformation, mapping the coordinates with our window size (window_width * window_height). These are all done inside the rasterizer, so you don't need to worry about those, however it's important to see and understand how these work.

REMEMBER: C++ uses radians as input to trigonometric functions.

4 Grading and Submission

4.1 Grading

You should try to use `<<` to set the matrices.

1. (5 points) Implement the model matrix correctly.
2. (5 points) Implement the perspective projection matrix correctly.
3. (10 points) Your code integrated correctly with the existing framework, you can view the final triangle after the transformations.
4. (10 points) The triangle drawn on the screen rotates correctly when pressing the 'a' and 'd' key, or use it from command line. Note that you should turn caps off to make the rotation work correctly.
5. (Bonus 5 points) Define and implement the function in main.cpp to get rotation matrix of arbitrary axis. You can use Rodrigues' Rotation Formula for the implementation, and add another function named: `Eigen::Matrix4f get_rotation(Vector3f axis, float angle)`.

4.2 Submission

After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, as well as the files of the project, whether you modified them or not. Please do **NOT** include anything from the `build` folder. Add a short README.md file in the directory, write down your full name and perm number inside it. Tell us whether you did the bonus part or not, and also include all necessary information for running the extra credits. Then compress the entire folder and rename it with your name, like "Goksu.zip". Submit only **ONE** .zip file on GauchoSpace before the deadline.