Assignment 2: Rotation and Projection
CS180 Winter 2021

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Jan 22, 2021 (Friday)

Due at 23:59 Anywhere on Earth (AOE) on Jan 28, 2020 (Thursday)

Notes:

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

1 Overview

In the lectures 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 this assignment, you are to simulate a simplified version of CPU-based rasterization renderer.

The task of is to make a rotation matrix and a perspective projection matrix. Given the 3D world coordinate of three vertices, $v_0(2.0, 0.0, -2.0)$, $v_1(0.0, 2.0, -2.0)$, $v_2(-2.0, 0.0, -2.0)$, you should transform them into screen space coordinate and draw the wireframe triangle on screen. (We provide you the *draw_triangle* function, you only need to construct the transformation matrices). In short, you need to do the model, view, projection, viewport transformation to see a triangle on the screen. In the code provided the model and projection transformation parts are left unfinished. If you are unclear about any concept above, review the lecture notes and post questions on Piazza.

You need to modify the following functions 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():** Provide other operation 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 it. 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 commands 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                      // Opens a window and renders the image
                                    // in it repeatedly until Esc is hit.
                                    // The triangle can be rotated with
                                    // A and D keys

$ ./Rasterizer -r 20 image.png      // Rotates the triangle by 20 degrees
                                    // and the image is saved in image.png
```

2 Getting started

You can work with the accompanying skeleton code in one of the following ways:

- Install the project dependencies listed below in your system. Download the code, build and run it locally.
- All dependencies are installed on the VM provided to you in the previous assignment. Download the skeleton into it and work there.

2.1 Own system

The following dependencies need to be installed in your system:

- CMake: This is the build tool needed to build the project.
- Eigen: This library is used in our project for linear algebra operations. Download the latest stable source code, extract it and rename the folder as `eigen3`
- OpenCV: This library is used for image manipulation and display. On Debian/Ubuntu, this can be easily installed by `apt install libopencv-dev`.

Now you should be able to tell the project how to find these libraries to build. IDEs a way to enter the path of the libraries and they generate `CMakeLists.txt` accordingly. To do this yourself, if the path of your downloaded `eigen3` folder is `<some-path>/eigen3`, add the following line at the end of `CmakeLists.txt` -

```
target_include_directories(Rasterizer PUBLIC <some-path>)
```

If OpenCV is installed correctly on your system, the

```
find_package(OpenCV REQUIRED)
target_link_libraries(Rasterizer \${OpenCV_LIBRARIES})
```

lines should take care of finding OpenCV. Now the project can be built and run.

```
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

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

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

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

Member functions:

- `set_model(const Eigen::Matrix4f m)`: Sets the class `model` matrix to the given parameter.
- `set_view(const Eigen::Matrix4f v)`: Sets the class `view` matrix to the given parameter.
- `set_projection(const Eigen::Matrix4f p)`: Sets the class `projection` transformation matrix to the parameter.

- `set_pixel(Vector2f point, Vector3f color)`: Sets the screen pixel at coordinates (x,y) to the color (r,g,b) by writing to the frame buffer.
- `draw(...)`: This method first does the necessary transformations by multiplying with model, view, projection matrices to get the vertices in normalized device coordinates (NDC). These coordinates range over $[-1, 1]$. Then the viewport transformation is done to map to our window size. This is implemented and is important for you to go through and understand.

These setter functions are to be used by you to set the model, view and projection transformation matrices of the rasterizer to the correct values.

In the `main.cpp` file the graphics pipeline is simulated. An instance of the rasterizer class is first defined and other necessary variables are set. Then a triangle is hard coded with three vertices (don't change them). Above the main function, there are 3 function definitions that each calculate model, view and projection matrices. Each of these functions must return the corresponding matrix, which are then passed to the rasterizer using the `set_model()`, `set_view()` and `set_projection()` functions respectively. Then the rasterizer's `draw` method is invoked to draw the image which is then displayed by OpenCV.

4 Grading and Submission

You are required to construct each matrix on your own. Do not call the Eigen library to set your matrix.

- `[5 points]` Implement the `get_model_matrix` function in `main.cpp`.
- `[5 points]` Implement the `get_projection_matrix` matrix in `main.cpp`.
- `[10 points]` Your code integrated correctly with the existing framework and the final triangle, after all transformations, can be viewed.
- `[10 points]` The triangle drawn on the screen rotates correctly when pressing the 'A' and 'D' key, or is draw appropriated when used from command line.
- `[Bonus 5 points]` Define and implement a new function in `main.cpp` to get the rotation matrix about any arbitrary axis.
  `get_rotation(Vector3f axis, float angle)`

After you finish the assignment, clean the project, include the CMakeLists.txt as well as other files of the project, whether you modified them or not in the folder.

Add a short PDF report of the images output by your code with descriptions for each of them. One image should show the triangle without any rotation and another must be showing it when it is rotated by 20 degrees. Write whether you did the bonus part or not and show an image with the triangle rotated 20 degrees about the vector $(1, 1, 0)$. Write your full name and perm number. Then compress the entire folder and rename it with your name, like "Kalyan.zip". Submit the .zip file on GauchoSpace before the deadline.