COSC 471: Computer Graphics

Spring 2023, Assignment 2

Transformation and Projection

(Total points: 100)

Due date: March 21st, 2023 (end of the day)

Submission Instructions

1. Compile your solutions and generate screenshots of your output, then write your understanding of output in a file. Compress source codes output and your explanation into one compressed file.
2. Rename the compressed file following this notation:
   FirstnameLastnamePA2.zip (or any format you prefer)
   Do not use a space in the filename.
3. Upload and submit the compressed file through Blackboard.
4. **NO** late submission is accepted.

---

1. Overview

We have learned in the class how to using transformation to move an object in 2D and 3D. Now let us implement the concepts we learned into coding. In the next few assignments, you are required to implement a simplified version of CPU based Rasterization.

The main tasks of this programming assignment include implementing a transformation matrix and a view/projection matrix. Giving three 3D points **v0(2.0, 0.0, −2.0), v1(0.0, 2.0, −2.0), v2(−2.0, 0.0, −2.0),** you are required to transform these points to the camera/view/monitor coordinates system, and draw a lined triangle based on them (in the skeleton program provided, I have provided a **draw_triangle** function, and you are required to add in codes to transformation matrix). In general, you are required to build model, view, projection transformation matrix, so that we can display the lined triangle on the screen. If there is any question, please refer to the class slides and videos.

The functions you need to modify in file **main.cpp** include (Do NOT change function names and other predefined function to ensure the code is compliable):

- **`get_model_matrix(float rotation_angle)`**: build the model transformation matrix's each element, and then return the matrix. In this function, you only need to implement 3D rotation along z axis in this transformation matrix, and ignore translation and scaling.
- **`get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)`**: using the giving parameter, build a projection matrix and return it.
- **`[Optional] main()`**: any other functions if you think necessary.

When you build your transformation matrix correctly, rasterization will generate a window and display the lined triangle. Because rasterization works frame by frame, so you should be able to use **A** and **D** keys to rotate the triangle along z axis, and when you press **Esc** key, window will be closed and program is terminated.
Bonus part: implement triangle rotate along any axis which go through the original point.

Besides, you can also run the program using command line and pass the rotation angle as a parameter to the program (as shown below). In this way, there will be no window and results will be saved into a file, normally an image here (if not specified, by default it is saved in the file **output.png**). The image file is located in the same place as your executable file. For example, if your executable file is in the **build** folder, the image file will be the same location.

```
// loop running program, build a display window, you can control using A and D keys to rotate
./Rasterizer
// Run the program the rotate the triangle by 20 degree, then save the output in file output.png
./Rasterizer -r 20
// Run the program the rotate the triangle by 20 degree, then save the output in file image.png
./Rasterizer -r 20 image.png
```

2. Assignment skeleton

In this programming assignment, you don't need to use triangle files. But focus on understanding and modify files **rasterizer.hpp** and **main.cpp**. The file **rasterizer.hpp** is used for generating rasterization and completing rendering.

Rasterization in this program plays a very important role. Its variable and functions are explained as below:

Variables:

- **`Matrix4f model, view, projection`**: three transformation matrixes.
- **`vector<Vector3f> frame_buf`**: frame buffer, which is used to store the color data rendered on the screen.

Functions:

- **`set_model(const Eigen::Matrix4f& m)`**: set the internal model matrix as the parameters and pass to rasterization.
- **`set_view(const Eigen::Matrix4f& v)`**: set view transformation matrix as internal view matrix.
- **`set_projection(const Eigen::Matrix4f& p)`**: set internal projection matrix as given matrix **p,** and pass to rasterization.
- **`set_pixel(Vector2f point, Vector3f color)`**: set the pixel **(x, y)** as the color of **(r, g, b),** and write them into frame buffer.

In file **main.cpp**, we have simulated the graphic pipeline. First, we define the example of rasterization, and set the variables for it. Then we get a hard-coded triangle with three points (do NOT change it). In the main function, we have defined three functions to calculate model, view, and projection matrixes, and each function has a corresponding return matrix. These return values will be pass to rasterization through three functions namely **set_model(), set_view()**, and **set_projection()**. Finally, the rasterization will show the transformation results on screen.

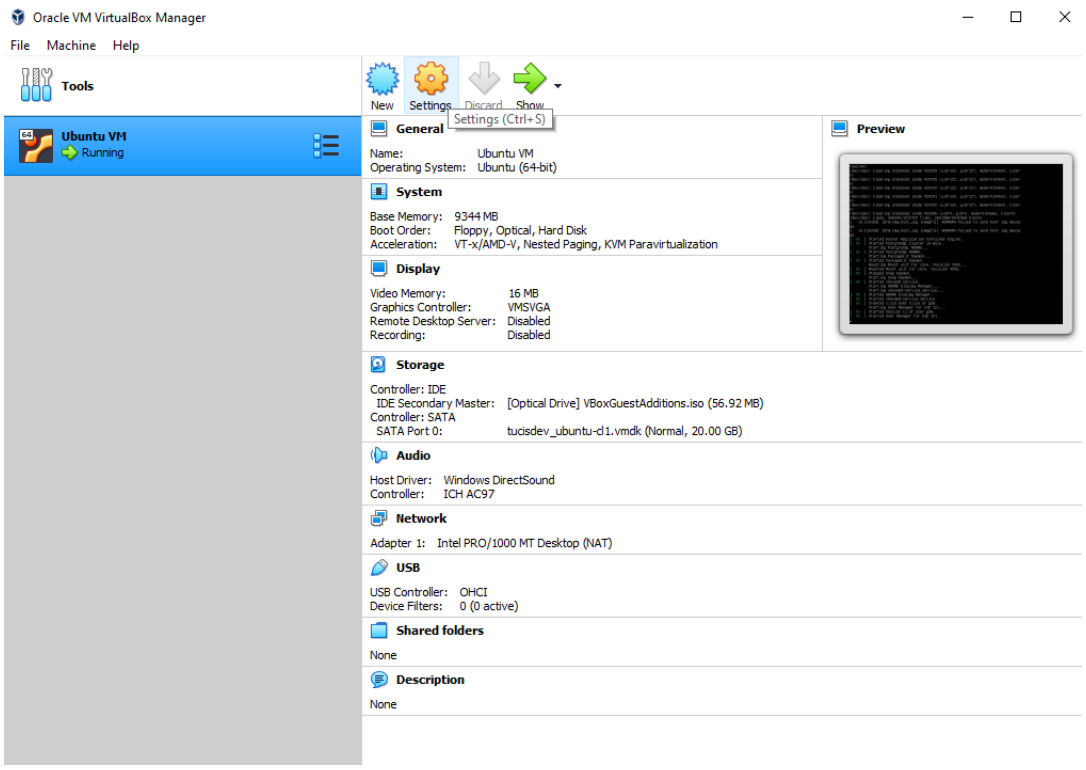
After model, view, and projection transformation, we should have these three points (of the triangle) in canonical space coordinate. Canonical space coordinate consists of three **x, y, z** in the range of **[-1,1].** Next, we need to change viewpoint and display on the screen, which has been completed in rasterization and we do not need to worry about it. But you need to understand this important step in graphic pipeline.

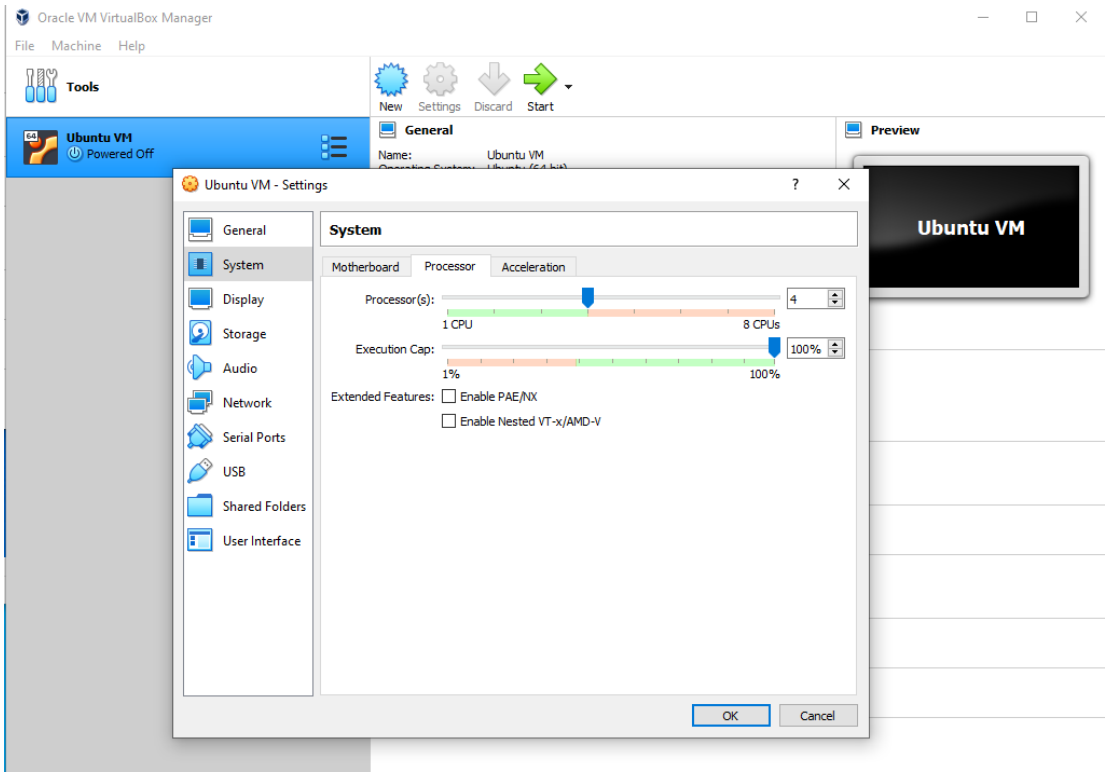
3. Compiling

This programming assignment requires of using two library: Eigen and OpenCV. Please make sure download and set them correctly. If you work on virtual machine and compile using **CMake,** please use the commands line below in the terminal:

```
// build a folder called “build”
Mkdir build
// move to the build folder
Cd build
// based on the directory provided in CMakeLists.txt to run CMake
cmake ..
// compile using make, -j4 uses 4 cores to optimize compile
make -j4
// run
./Rasterizatier
```

If you have not set the core numbers for virtual machine, click “setting” and choose “” to set the core numbers.





4. Assignment submission and evaluation

4.1 Submission

Read the assignment description carefully to make sure you understand the programing assignment correctly. Modify the skeleton codes as required. Compile your solutions and generate screenshots of your output, then write your understanding of output in a file. Compress source codes output and your explanation into one compressed file named **FirstnameLastnamePA2.zip (or any format you prefer).** Submit the compressed file through Blackboard before deadline.

When you complete the assignment, please check your project clearly and make sure your submission folder includes **CMakeLists.txt file** and all the source files no matter you changed them or not. In addition, please include a **README.md** file to explain if you complete the bonus question, or explain this in the result analysis and screenshot file.

4.2 Evaluation

This programming assignment has two parts: basic parts and bonus part. Here is the grading details:

- Build model matrix correctly (25%)
- Build view and projection matrix correctly (25%)
- Compile and run codes correctly and can see the transformed triangle (20%)
- When press A and D keys, triangle can rotate; or using command line to get the rotation results (20%)

- Screens shot and result analysis (10%)
- Bonus part: in the main.cpp file, build a function to let allow rotation along any give axis which go through original point. This function can look like this:
  `Eigen::Matrix4f get_rotation(Vector3f axis, float angle)`