COSC 471: Computer Graphics

Fall 2021, Assignment 3

Triangles and Z-buffering

(Total points: 100)

Due date: Nov 4th, 2021 (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:
   FirstnameLastnamePA3.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

In the last assignment, although we drew a wireframe (lined) triangle on the screen, it did not seem very interesting. So this time we continue to take one step further: draw a solid triangle on the screen, in other words, rasterize a triangle. In the last assignment, after the viewport changed, we called the function **`rasterize_wireframe(const Triangle& t)`**. But this time, you need to fill in and call the function **`rasterize_triangle(const Triangle& t)`** yourself.

The internal workflow of this function is as follows:

- Create a 2-dimensional bounding box for this triangle.
- Traverse all the pixels in this bounding box (using their integer index). Then, use the screen space coordinates of the pixel center to check whether the center point is inside the triangle.
- If it is inside the triangle, compare the interpolated depth value at its location with the corresponding value in the depth buffer.
- If the current pixel point is closer to the camera, set the pixel color and update the depth buffer.

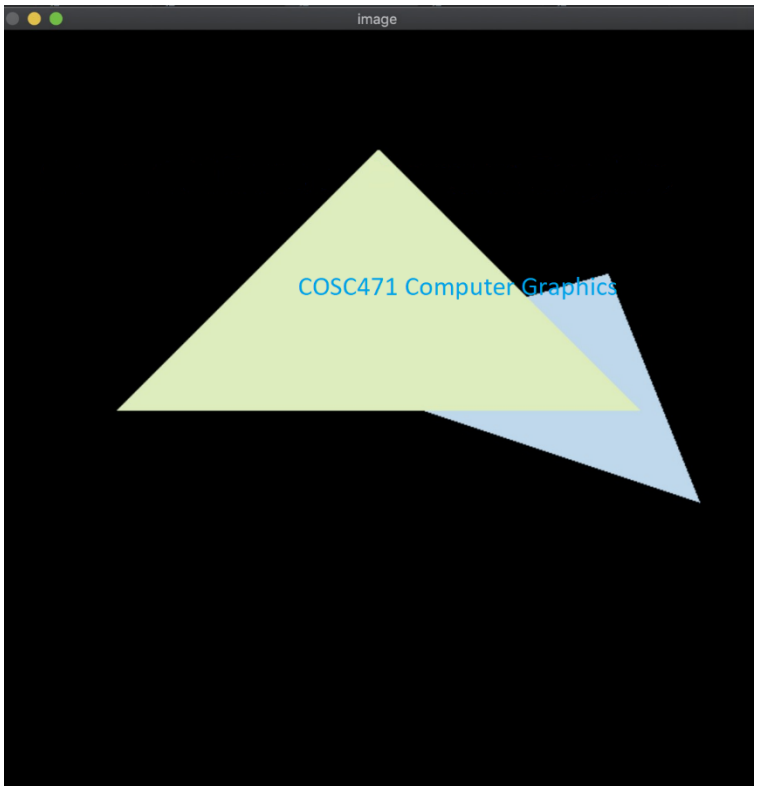
The functions you need to modify are as follows:

- **`rasterize_triangle()`**: execute the triangle rasterization algorithm
- **`static bool insideTriangle()`**: Check whether the point is inside the triangle. You can modify the definition of this function, which means that you can update the return type or function parameters in your own way.

Because we only know the depth values at the three vertices of the triangle, for the pixels inside the triangle, we need to use interpolation to get the depth value. We have already taken care of this part for you, because it uses some topics not been discussed yet. The interpolated depth value is stored in the variable **`z_interpolated`**.

Please note how we initialize the depth buffer and pay attention to the sign of z values. In order to make it easier for you to write code, we have reversed z to ensure that they are all positive numbers, and the larger the value, the further away from the viewpoint.

In this assignment, you do not need to deal with the rotation transformation, just return an identity matrix for the model transformation. Finally, we provide two hard-coded triangles to test your implementation. If the program is implemented correctly, you should be able to see the output image very similar to what is shown below:



2. Compiling

Download and use our updated skeleton code coming with this assignment on your own computer or virtual machine. Notice that the **`get_projection_matrix()`** in the **`main.cpp`** empty. Please copy and paste your implementation in the first assignment (assignment 1) to fill in this function.

Compile and run the code as previous assignments.

3. Assignment submission and evaluation

3.1 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 (if you complete bonus question, please include screenshot of it too), or explain this in the result analysis and screenshot file.

3.2 Evaluation

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

- Correctly implement triangle rasterization algorithm (40%)
- Correctly test whether the point is within the triangle (20%)
- Correctly implement the z-buffer algorithm and draw the triangles on the screen in order (20%)
- Compile and run codes correctly and submit all files correctly (10%)
- Screens shot and result analysis (10%)
- Bonus part: implement any Anti-aliasing algorithm to improve the display of triangle so that when we enlarge the display, there will be aliasing on triangle edges. (10%)