Assignment 3: Triangles and Z-buffering
CS180/CS280 Fall 2022

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Oct 14th, 2022 (Friday)

Due at 11:59 pm on Oct 20nd, 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 last assignment, we drew a wire-frame triangle on the screen, which doesn't look so interesting. This time we will move one step forward – draw solid triangles on the screen, in other words, rasterize a triangle. Last time, after the view-port transformation, we called the `rasterize_wireframe(const Triangle& t)`, this time you need to implement and call the function
`rasterize_triangle(const Triangle& t)`.
The general workflow inside this function is:

1) Create the 2D bounding box of the triangle.
2) Iterate through all the pixels (using their **integer** indices) that are inside this bounding box. Then, use the screen-space coordinate of the center of the pixel to check if the center point is inside the triangle.
3) If it is inside, then compare the **interpolated** depth value at its position to the corresponding value from the depth buffer.
4) If the current point is closer to the camera, set the pixel color and update the depth buffer.

Functions that you need to edit:

- **rasterize_triangle(const Triangle& t):** Perform the triangle rasterization algorithm. It is located in the file `rasterizer.cpp`.
- **static bool insideTriangle(int x, int y, const Vector3f* v):** Test if a point is inside a triangle. You can change the definition of this function. That means, you can update the return type, or the function parameters the way you want. This function needs to be called by `rasterize_triangle()` in `rasterizer.cpp`.
- **get_model_matrix(float rotation_angle)**: For this assignment you don't need to handle any rotation, just return an identity matrix. It is located in the fille `main.cpp`.
- **get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar):** This is also left empty in the file `main.cpp`. Copy-paste your implementation of this function from second assignment. Please make sure that the copied implementation is correct.

Since we only know the depth value at the three vertices of the triangle. For the pixels inside the triangle, we need to do the **interpolation** to get its depth value. We have handled this part for you, since it is the topic that has not been covered in the lecture yet. The interpolated depth value is saved in the variable `z_interpolated`. You can find this part of the code in the comments in the `rasterize_triangle(const Triangle& t)`.

Pay attention to how we initialize the depth buffer and the sign of your z values. In this assignment, you don't need to handle the rotation transformation, just return an identity matrix for the model transformation. Finally, we provide two hard-coded triangles to test your implementation, if you do it correctly, you will see the output image like this:

The text **WILL NOT** be there and also note how one triangle is on top of another. You will get an incorrect result if the depth testing part is not implemented correctly.

2 Getting started

Download and use our updated skeleton code package either on your own machine or in the virtual machine. You will notice that the `get_projection_matrix()` function in `main.cpp` is left empty. Copy-paste your implementation from the **second assignment** to fill in this function.

3 Grading

1. (5 points) Submission is in the correct format, includes all necessary files. Code can compile and run.
2. (20 points) Implement the triangle rasterization algorithm correctly.
3. (10 points) Implement the inside triangle test correctly.
4. (10 points) Implement the z-buffer algorithm correctly. The triangles are drawn on screen in correct order.
5. (Bonus 5 points) Anti-aliasing by super-sampling: You may notice that the result image is quite jaggy when we zoom in. We can solve this by super-sampling. Sample each pixel by 2×2 samples and compare the result before and after. You need to create a larger frame buffer to do this. One way to think about is that instead of considering only the center of the pixel, each pixel will now have 4 samples (if 2×2 is used). Color of the pixel will be the average of all the samples' colors.

   **Note: The bonus will not give you bonus points in your final grade, it only counts towards a bonus to your learning. However, if you receive an A at the end of this quarter and you did the bonus questions, then your grade will become an A+.

4 Submission

After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, whether you modified it or not. Add a **SHORT** report file in the directory, write down your full name and perm number inside

it. Tell us whether you did the bonus part or not. You can also include some screenshot or result images in your report. You **DO NOT** need to include a lot of information in the report. Then compress the entire folder and rename it with your name, like "Goksu.zip". Submit the **ONLY ONE** .zip file on GauchoSpace. Please **DO NOT** contain build folder inside the .zip file.