TDT4195: Visual Computing Fundamentals

Computer Graphics - Assignment 2 Report

22nd September 2023

Oluwatobi OJEKANMI

1 Per-vertex Colors

1.2

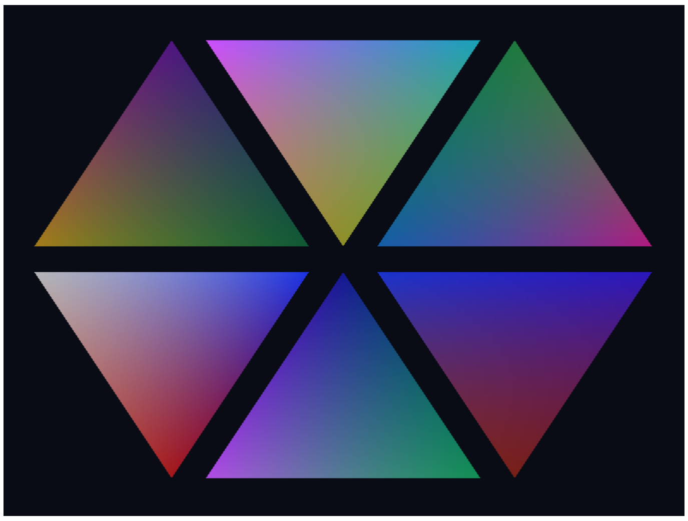

Figure 1: 6 triangles with distinct vertex colors

The vertex colors exhibit strong, distinct intensities at the vertices of the triangles, but gradually blends as we approach the center of each triangle. This gradual shift, known as smooth color transitions or interpolation, enhances visual aesthetics by avoiding abrupt color changes. The effect is achieved through fragment interpolation in the fragment shader.

When rendering a triangle, the rasterization process generates numerous fragments based on the triangle's specifications (size, shape, and orientation in space) and the screen's specifications (number of pixels and pixel area). The corresponding fragments derived from rasterization can be in the order of tens, hundreds, or thousands which is a lot more than the vertices originally specified. This implies that new attributes have to be generated for the fragments since we are only interested in further processing of the fragments, and not the vertices. Therefore, to determine the attributes for the fragments, the vertex attributes are interpolated for each fragment based on its position in the triangle. This process creates smooth color transitions and is applied to all fragment shader input attributes, as seen in Figure 1.

2 Alpha Blending and Depth

2.1

For this task, I drew 3 overlapping triangles: red, green, and blue with 0.5 alpha values and respective depth values of 0.9, 0.45, and 0. The corresponding triangles are shown in Figure 2.

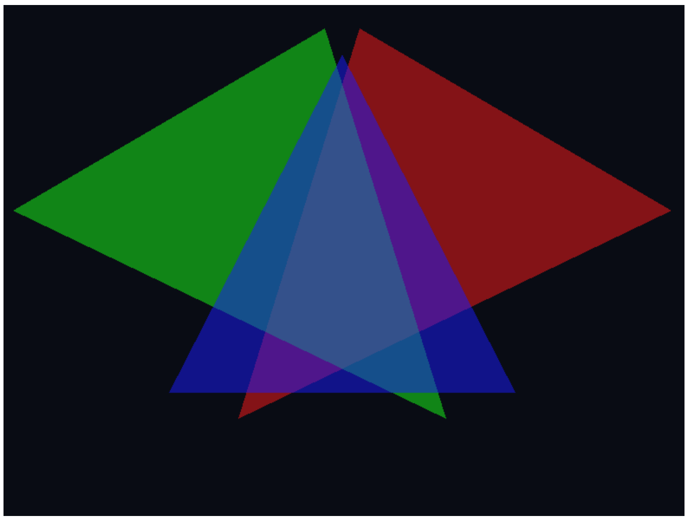

Figure 2: 3 Triangles plotted at different depths (drawn in a back-to-front order)

2.2 (i)

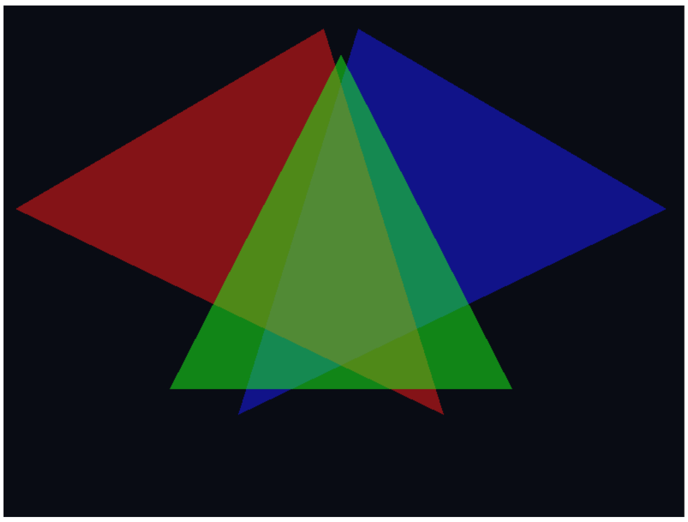

Figure 3: 3 Triangles plotted with shuffled color VBO

By swapping the colors of different triangles, I obtained a different blended color from the initial figure. The blended color obtained in Task 2.1 (Figure 2) is more bluish while the blended color obtained in this task (Figure 3) is more greenish. This is because the color of the nearest triangle has the most color contribution to the blended color and this contribution decreases with depth (i.e., the right triangle (farthest) is drawn first, then the left triangle, then the middle triangle (nearest). Thus, the color of the overlaying triangles thus have more dominance over the colors of the overlayed triangles).

2.2 (ii)

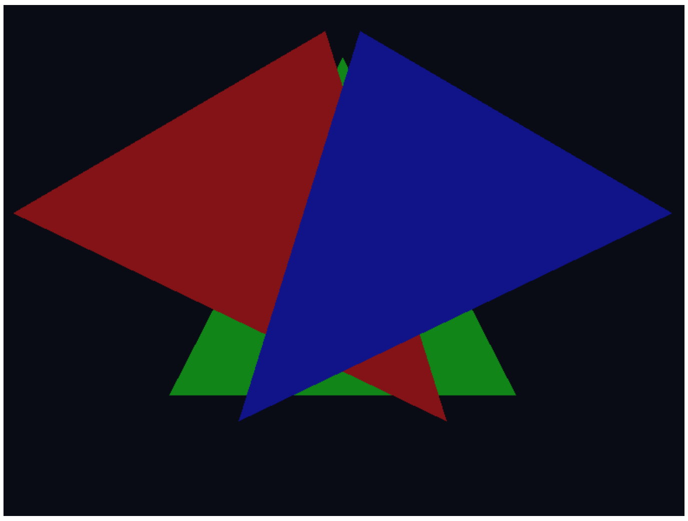

Figure 4: 3 Triangles drawn in a front-to-back order

To swap the depth (z-coordinate) at which different triangles are drawn, I multiplied the depth values of each triangle's vertices by -1 such that the nearest triangle becomes the farthest, while the farthest becomes the nearest. This causes the transparency effect to disappear. This occurred because the triangles were drawn in a front-to-back order and as such overlapping fragments of the triangles farther from the camera are considered occluded by the fragments of the nearest triangle after depth testing.

The z- or depth buffer stores depth values for pixels on the screen, and when triangles are drawn, OpenGL checks these values to determine which fragments should be displayed. Since the first triangle, drawn closest to the camera, populates the depth buffer with its values, fragments from the other triangles have greater depth values in the overlapping areas. Consequently, these fragments are considered occluded and not drawn, leading to the observed behavior. To see all three triangles in overlapping sections, drawing them in a back-to-front order would ensure fragments from each triangle are visible when they overlap (as done in the previous tasks).

3 The Affine Transformation Matrix

b.

For this exercise, I used a row-major transformation matrix and post-multiplied the vertex coordinates with the transformation matrix as shown below.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

where $[x', y', z']$ are the transformed coordinates of $[x, y, z]$.

As a result, modifying individual values in the transformation matrix yields the following effects:

1. **a**: Scales or flips the geometry along the x-axis.
    - Values of **a** between 0 and 1 shrink the geometry.

- Values of **a** greater than 1 enlarge/magnify the geometry.
- Values of **a** less than 0 scale and flip/mirror the geometry about the x-axis.

2. **b**: Shears the geometry along the x-axis.
   - Positive values of **b** shear the geometry in the positive x-axis direction.
   - Negative values of **b** shear the geometry in the negative x-axis direction.
3. **c**: Translates the geometry along the x-axis.
   - Positive values of **c** translate the geometry in the positive x-axis direction.
   - Negative values of **c** translate the geometry in the negative x-axis direction.
4. **d**: Shears the geometry along the y-axis.
   - Positive values of **d** shear the geometry in the positive y-axis direction.
   - Negative values of **d** shear the geometry in the negative y-axis direction.
5. **e**: Scales or flips the geometry along the y-axis.
   - Values of **e** between 0 and 1 shrink the geometry.
   - Values of **e** greater than 1 enlarge/magnify the geometry.
   - Values of **e** less than 0 scale and flip/mirror the geometry about the y-axis.
6. **f**: Translates the geometry along the y-axis.
   - Positive values of **f** translate the geometry in the positive y-axis direction.
   - Negative values of **f** translate the geometry in the negative y-axis direction.

c.

The rotation matrix about the z-axis is given by

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Comparing this rotation matrix ($\boxed{2}$) to the given transformation matrix ($\boxed{1}$), we will need to provide values for elements a, b, d, and e at the same time to rotate the geometry about the z-axis. However, I only provided values for one element at a time while implementing the task. Thus, I can conclude that none of the transformations observed were rotations.

4 Combinations of Transformations

Please note that I disabled back-face culling for this task so as to visualize 2D shapes in 360°. I also used the following keys to move the camera in the scene

1. **Translation in the x and y axes**: WASD keys
   - **W**: translate the camera towards the positive y-axis (i.e., move the camera up)
   - **A**: translate the camera towards the negative x-axis (i.e., move the camera left)
   - **S**: translate the camera towards the negative y-axis (i.e., move the camera down)

- **D**: translate the camera towards the positive x-axis (i.e., move the camera right)

2. **Translation in the z-axis**: left shift and left control
   - **Left shift**: translate the camera towards the positive z-axis (i.e., move the camera away from the scene)
   - **Left control**: translate the camera towards the negative z-axis (i.e., move the camera towards the scene)
3. **Pitch and Yaw**: Arrow keys
   - **Up**: rotate the camera clockwise around the x-axis
   - **Down**: rotate the camera counter-clockwise around the x-axis
   - **Left**: rotate the camera clockwise around the y-axis
   - **Right**: rotate the camera counter-clockwise around the y-axis

5 Optional Tasks

I attempted optional task 1. Kindly comment lines 532 to 534 and uncomment lines 537 to 547 of the main.rs to run the code for the task.