CS315 – Introduction to Computer Graphics
Winter, 2022

Assignment 4

Assigned Date: Wednesday, March 23, 2022
Due Date: Wednesday, April 6, 2022

Programming Question (Submission required)

Add shading calculation to your Assignment 3's program.

You may refer to the two example programs posted in Chapter 6 for different approaches to shading implementation:

"*shadedSphere1*"

- Normal vector is accurate at each vertex on the sphere with unit length radius and centered at the origin;
- Shading value is calculated at each vertex in the vertex shader;
- Rasterization process will linearly interpolate the shading values at corners across the polygon (here is triangle)
- This known as Gouraud Shading (I prefer call it Gouraud interpolation)

"*shadedSphere2*"

- Normal vector is accurate at each vertex on the sphere with unit length radius and centered at the origin (same as shadedSphere1);
- Rasterization process will linearly interpolate the normal vector at corners across the polygon (here is triangle)
- Shading value is calculated at each pixel in the fragment shader using the interpolated normal;
- This known as Phong Shading (I prefer call it Phong interpolation)

You may choose either approach.

Practice Questions (Submission Not Required)

1. Cohen-Sutherland Clipping Algorithm

   a. Suppose the top, bottom, left and right clipping boundaries are defined as:

      ```
      const int Xmin = 0;
      const int Xmax = 1200;
      const int Ymin = 0;
      const int Ymax = 800;
      ```

      Write a function with the following function heading:

      ```
      unsigned char outcode(vec2 p);
      ```

      where p is a 2D vertex, and the return value is the computed "outcode" (stored in the lower 4 bits) based on the definition given in page 405 of the textbook.

   b. Write another function that takes two vertices, p1 and p2, of a line segment as the input parameters, and return the clipped results in p1 and p2 as well. In addition, this function returns an integer value with 1 indicating a non-empty result, or 0 otherwise:

      ```
      int Cohen-Sutherland-Clipping(vec2 p1, vec2 p2);
      ```

   Hint: You should follow the cases described in the textbook, pages 404 to 406. You may also find that your program structure could be a little bit better if you switch the order of cases 2 and 3.

   c. Simulate your function developed in (b) with the following testing parameters:
      - p1 = (50, 400), p2 = (600, 700)
      - p1 = (1230, 400), p2 = (1650, 800)
      - p1 = (850, 400), p2 = (1650, 900)
      - p1 = (-50, 600), p2 = (600, 900)

   Hint: Your answer should be step-by-step and include: outcode for each vertex, which case is executed, the intersection point calculated and its new outcode if any, etc.

2. Refer to the incremental algorithm for drawing lines in my supplemental note on Bresenham algorithm.

   a. If -1.0 < m < 0.0, does this function draw the line properly? Why?

   b. If |m| > 1.0, does this function the line properly? Why?

   c. Modify this function such that it will draw lines properly in any cases.

3. Question 8.18, page 447 of the textbook.

   Consider the edge of a polygon between vertices at (x_1, y_1) and (x_2, y_2). Derive an efficient algorithm for computing the intersection of all scan lines with this edge. Assume that you are working in screen coordinates.

   Hint: Assume $y_1 < y_2$. You have a loop structure like the following:

   ```
   x = x1;
   for (y=y1; y<= y2; y++)
   {
     x = ???;
   }
   ```

   You may define additional local variable(s) as you need. You may add statements anywhere. The goal is to compute x inside the loop correctly and in the most efficient way that you can think about.

General Note:

1. Only the question 1 is a programming problem. You do not have to demonstrate this program to the marker. Instead, you submit your program with sample screen shots demonstrating your shading results.

2. Your submission should include the following documents in a single zip file:

   - Well-documented source program, and sample screen shots