PA4 Part B Documentation

Program Description

Transforms input files with lists of 3D points into a lighting environment with a light source and eye location. The files are interpreted where each line except for the first and last represent triangles. These triangles then are determined whether or not to be front-facing, their normalized surface lighting intensity computed with lambertian shading. A third number is computed which combines the front facing and surface lighting algorithms to output 0 if a surface is back-facing.

Important Library Details

- Eigen
    - Library path: the headers for the Eigen library are located in /usr/include/eigen3 on my Linux machine.
    - Library version: I have installed Eigen version 3.4.0.

Marginal Cases

- Invalid inputs:
    - The coordinates of the centroid of a triangle are the same as the source of light($\mathbf{l} = \mathbf{c}$): this results in the vector from the centroid to the light source being a zero vector, which cannot be normalized to later create a light intensity level between 1 and 0.
    - The coordinates of the centroid of a triangle are the same as the eye($\mathbf{e} = \mathbf{c}$): this results in the vector from the centroid to the eye being a zero vector, which cannot be normalized as required to decide if a triangle should be culled.
    - Any two points of a triangle are the same($\mathbf{p}_n = \mathbf{p}_m$): If two points of a triangle are the same, a normal vector for use in culling cannot be generated as the “triangle” forms either a line or a point.
- Invalid computations:
    - All important computations in the Eigen implementation methods were handled by Eigen, and the outputs have been checked.
    - All custom implementation methods simply added two doubles and stored them in the corresponding double element of a result matrix. The outputs have also been checked.

Design Choices

- Triangle objects will not contain the vertices which actually define them, only the centroid and normal vector as they are all that are necessary for the calculations.

- The equation $i = max(0, \vec{n} \bullet \vec{v}_L)$ - where $\vec{n}$ is the normal vector to the triangle and $\vec{v}_L$ is the vector from the centroid in the direction of the light - will be used to determine the light intensity as seen by the viewer of each triangle. If both $\vec{n}$ and $\vec{v}_L$ are normalized before the calculation, then the resulting intensity will be a magnitude between 0 and 1. The max in the equation helps set the intensities of surfaces pointing away from the light source to 0, as in those cases $\vec{n} \bullet \vec{v}_L$ is a negative number.
- For checking for the edge cases, a tolerance will be used to counteract floating point errors as the coordinate of the centroids must be calculated.

Pseudocode

```
STRUCT Input:
    MEMBER dimension_num_
    MEMBER num_mat_

FUNCTION GetInput(input_path)

CLASS Triangle:
    // MEMBER vertices_
    MEMBER centroid_
    MEMBER normal_vec_

    CONSTRUCTOR Triangle(vertex_1, vertex_2, vertex_3):
        // SET vertices_ = [vertex_1, vertex_2, vertex_3]

        SET centroid_ = (vertex_1 + vertex_2 + vertex_3) / 3

        SET edge_1 = vertex_2 - vertex_1
        SET edge_2 = vertex_3 - vertex_1

        SET normal_vec_ = edge_1.cross(edge_2)
        NORMALIZE normal_vec_

    FUNCTION FindIntensity(light):
        SET light_dir = light - centroid_
        NORMALIZE light_dir

        SET intensity = normal_vec_.dot(light_dir_)

        IF intensity < 0:
            SET intensity = 0
```

```
        RETURN intensity

    FUNCTION ShouldCull(light, eye):
        SET eye_dir = eye - centroid_
        NORMALIZE eye_dir

        RETURN normal_vec_.dot(eye_dir) < 0

    FUNCTION FindCulledIntensity(light, eye):
        IF ShouldCull(light, eye):
            RETURN 0
        ELSE:
            RETURN FindIntensity(light)


Int main():
    CALL SolveFile() for each file

    RETURN 0

FUNCTION SolveFile(input_path, output_path):
    SET input = GetInput(input_path).num_mat_
    OPEN output_file at output_path

    SET eye = [input[0][0], input[0][1], input[0][2]]
    SET light = [input[0][3], input[0][4], input[0][5]]

    FOR each row_num in input except for 0:
        SET vertex_1 = [input[row_num][0], input[row_num][1], input[row_num][2]]
        SET vertex_2 = [input[row_num][3], input[row_num][4], input[row_num][5]]
        SET vertex_3 = [input[row_num][6], input[row_num][7], input[row_num][8]]

        IF vertex_1 == vertex_2 or vertex_1 == vertex_3 or vertex_2 == vertex_3:
            PRINT “Invalid Computation” to output_file
        ELSE:
            INITIALIZE triangle(vertex_1, vertex_2, vertex_3)

            IF EqualsWithinTolerance(triangle.GetCentroid, light) or
                    EqualsWithinTolerance(triangle.GetCentroid, eye):
                PRINT “Invalid Computation” to output_file
            ELSE:
                PRINT triangle.ShouldCull, triangle.FindIntensity and
                        triangle.FindCulledIntensity to output_file
```

```
CLOSE output_file
```