PA5 Part C Documentation

Program Description

Calculates the closest point and the distance between a bisector plane of two points and a third point.

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 first two points of an input line are the same($\mathbf{p}_1 = \mathbf{p}_2$): If two points of a line are the same, a normal vector cannot be generated for 3D planes.
- Invalid computations:
  - All important computations in the Eigen implementation methods were handled by Eigen, and the outputs have been checked.

Design Choices

- $\mathbf{x}$ will be solved using the line-plane intersection algorithm by creating a line $\mathbf{x} = \mathbf{p} + t\vec{n}$ where p is the point to project and $\vec{n}$ is the normal vector of the plane. So: $\mathbf{x} = \mathbf{p} + \frac{(\mathbf{q}-\mathbf{p})\bullet\vec{n}}{\vec{n}\bullet\vec{n}}\vec{n}$.
- Distance calculations will reuse code from PA4.

Pseudocode

```
STRUCT Input:
      MEMBER points_

FUNCTION GetInput(input_path)

CLASS PointNormalPlane:
      MEMBER normal_vec_
      MEMBER normal_vec_tail_
```

```
      CONSTRUCTOR PointNormalPlane(normal_vec, normal_vec_tail):
            SET normal_vec_ = normal_vec.normalized
            SET normal_vec_tail_ = normal_vec_tail

      FUNCTION FindDistanceToPoint(point):
            SET A = normal_vec_[0]
            SET B = normal_vec_[1]
            SET C = normal_vec_[2]
            SET D = -normal_vec_.dot(normal_vec_tail_)

            RETURN abs(A * x1 + B * x2 + C * x3 + D)

      FUNCTION FindClosestPoint(point):
            DECLARE closest_point = normal_vec_

            DECLARE numerator = normal_vec_.dot(normal_vec_tail_ - point)
            DECLARE denominator = normal_vec_.dot(normal_vec_)

            SET closest_point = closest_point * (numerator / denominator)

            SET closest_point = closest_point + point

            RETURN closest_point

FUNCTION GenerateBisectorPlane(point_1, point_2):
      SET midpoint = 0.5 * point_1 + 0.5 * point_2
      SET normal_vector = point_2 - point_1

      INITIALIZE ret_plane with midpoint and normal_vector
      RETURN ret_plane

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

      RETURN 0

FUCNTION SolveFile(input_path, output_path):
      DECLARE input raw_input = CALL of GetInput with input_path
      OPEN output_file at output_path

      FOR row from 0 to input.points_.length - 1:
```

```
            CALL SolveLine(input.points_[row], output_file)

      CLOSE output_file

FUNCTION SolveLine(points_row, output_file):
      IF points_row[0] == points_row[1]:
            PRINT “Invalid Computation” to output_file
      ELSE:
            DECLARE plane = CALL generateBisector(points_row[0],
                                                  points_row[1])

            DECLARE distance = plane.FindDistanceToPoint(points_row[2])
            DECLARE closest_point = plane.FindClosestPoint(points_row[2])

            PRINT closest_point and distance to output_file
```