PA4 Part A Documentation

Program Description

Finds normals/bisector planes between two points and calculates distances from a third point to those normals/bisectors planes.

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 or 2D lines.
  - The number of dimensions is not 2 or 3: this assignment only calculates answers in 2D and 3D, thus 4D input would not mean anything.
- 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

- I do not need to convert planes to point normal form as they will already be in point normal form after initialization.
- I will be using the parametric form of 2D lines as they are easier to work with. Since this is not a high speed application, I am okay with the more intensive calculation for distances from points to planes.
- I will be using the implicit form of planes by itself($\vec{n} \bullet (\mathbf{x} - \mathbf{p}) = 0$) as opposed to the expansion of the form($Ax_1 + Bx_2 + Cx_3 + D = 0$) arbitrarily, as it doesn't matter much.
- The constructor for Plane will only create planes of Point-Normal form to speed up calculations later.

Pseudocode

```
STRUCT Input:
    MEMBER dimension_num_
    MEMBER num_mat_

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)

CLASS ParametricLine2D:
    MEMBER vec_v_
    MEMBER point_on_line_

    CONSTRUCTOR ParametricLine2D(point_1, point_2):
        SET vec_v_ = point_2 - point_1
        SET point_on_line_ = point_1

    FUNCTION FindDistanceToPoint(point):
        SET vec_w = point - point_on_line_

        SET cosine_alpha = vec_v_.dot(vec_w) / (vec_v_.norm * vec_w.norm)
        SET sine_alpha = sqrt(1 - cosine_alpha * cosine_alpha)
        SET distance = vec_w.norm * sine_alpha

        RETURN distance

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

```
        RETURN 0

FUNCTION FindOrthonormal(vec):
        SET ret_vec = [-vec[1], vec[0]]
        NORMALIZE ret_vec

        RETURN ret_vec

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

        IF input.dimension_num_ == 2:
                CALL Solve2D(input.num_mat_, output_file)
        ELSE IF input.dimension_num_ == 3:
                CALL Solve3D(input.num_mat_, output_file)
        ELSE:
                PRINT “Invalid Computation” to output_file

        CLOSE output_file

FUNCTION Solve2D(num_mat, output_file):
        FOR EACH row in num_mat:
                SET point_1 = [row[0], row[1]]
                SET point_2 = [row[2], row[3]]

                IF point_1 == point_2:
                        PRINT “Invalid Computation”
                ELSE:
                        INITIALIZE line(point_1, point_2)

                        SET orthonormal = FindOrthonormal(line.GetVecV)
                        PRINT orthonormal.transpose() to output_file

                        SET point_3 = [row[4], row[5]]
                        SET distance = line.FindDistanceToPoint(point_3)
                        PRINT distance to output_file
```

```
FUNCTION Solve3D(num_mat, output_file):
        FOR EACH row in num_mat:
                SET point_1 = [row[0], row[1], row[2]]
                SET point_2 = [row[3], row[4], row[5]]

                IF point_1 == point_2:
                        PRINT “Invalid Computation”
                ELSE:
                        SET bisector = GenerateBisectorPlane(point_1, point_2)

                        SET normal = bisector.GetNormalVec
                        PRINT normal.transpose() to output_file

                        SET point_3 = [row[6], row[7], row[8]]
                        SET distance = bisector.FindDistanceToPoint(point_3)
                        PRINT distance to output_file
```