PA5 Part A and B Documentation

Program Description

Creates parallel and perspective projections for each of 3 points per line in input files.

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:
  - Both:
    - Normal is $\vec{0}$: When the plane's normal is 0, no plane can be generated.
  - Part A:
    - Projection vector is parallel to plane($\vec{v} \bullet \vec{n} = 0$): The line of projection never touches the plane therefore no projection exists or infinitely many exist.
    - Projection vector is $\vec{0}$: if the projection vector is zero nothing can be projected. Conveniently, this only happens when $\vec{v} \bullet \vec{n} = 0$ like above since any dot product of $\vec{0}$ is 0.
  - Part B:
    - Vector from origin to $\mathbf{x}$ is parallel to plane($\mathbf{x} \bullet \vec{n} = 0$): The line of projection never touches the plane therefore no projection exists or infinitely many exist.
- 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 in non-homogeneous matrix form with the following equations:
  - Parallel Projection: $\mathbf{x}' = \mathbf{x} + \frac{[\mathbf{q}-\mathbf{x}]\bullet\vec{n}}{\vec{v}\bullet\vec{n}}\vec{v}$
  - Perspective Projection: $\mathbf{x}' = \frac{\mathbf{q}\bullet\vec{n}}{\mathbf{x}\bullet\vec{n}}\mathbf{x}$

- In parallel projection, if one point can't be drawn, none of them can as $\vec{v} \bullet \vec{n} = 0$ is independent of the points. Thus, I will check for the projection existence once, and if it does not exist, I will fill the file with errors without looking at the points.

Pseudocode

```
STRUCT Input:
      MEMBER plane_
      MEMBER proj_dir_
      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 main():
      CALL SolveFile() for each file

      RETURN 0

FUNCTION ParallelProjExists(plane, proj_dir):
      RETURN plane.GetNormalVec().dot(proj_dir) != 0 and
                  plane_.normal != {0,0,0}

FUNCTION PerspectiveProjExists(plane, point):
      RETURN plane.GetNormalVec().dot(point) != 0 and
                              plane_.normal != {0,0,0}
```

```
FUNCTION ParallelProj(plane, point, proj_dir):
      DECLARE x_prime = proj_dir

      DECLARE numerator = plane.GetNormalVec().dot(plane.GetNormalTail() -
                              point)
      DECLARE denominator = proj_dir.dot(plane.GetNormalVec())
      SET x_prime = x_prime * (numerator / denominator)

      SET x_prime = x_prime + point

      RETURN x_prime

FUNCTION PerspectiveProj(plane, point):
      DECLARE x_prime = point

      DECLARE numerator = plane.GetNormalVec().dot(plane.GetNormalTail())
      DECLARE denominator = point.dot(plane.GetNormalVec())
      SET x_prime = x_prime * (numerator / denominator)

      RETURN x_prime

FUCNTION SolveFile(input_path, output_path_a, output_path_b):
      DECLARE input raw_input = CALL of GetInputAsMatrix with input_path
      OPEN output_file_a at output_path_a
      OPEN output_file_b at output_path_b

      RUN PartA(input, output_file_a)
      RUN PartB(input, output_file_b)

      CLOSE output_file_a
      CLOSE output_file_b

FUNCTION PartA(input, output_file):
      IF ParallelProjExists(input.plane_, input.proj_dir_):
            FOR row in input.points_:
                  FOR point in row:
                        PRINT ParallelProj(input.plane_, point,
                                    input.proj_dir_) to output_file

                  PRINT newline to ouput_file
      ELSE:
            FOR row in input.points_:
```

```
                FOR point in row:
                      PRINT “Invalid Computation” to output_file

                PRINT newline to ouput_file

FUNCTION PartB(input, output_file):
      FOR row in input.points_:
            FOR point in row:
                IF PerspectiveProjExists(input.plane_, point):
                      PRINT PerspectiveProj(input.plane_, point) to
                                  output_file
                ELSE:
                      PRINT “Invalid Computation” to output_file

            PRINT newline to ouput_file
```