Point-In-Shape Tool

This C++ tool determines whether a given point lies **inside** or **outside**, of a complex polygon (supports non-orthogonal shapes too) that may include **cutouts (holes)**.

This project implements a 2D point-in-polygon algorithm in C++, supporting complex polygons with holes (cutouts). It uses the **ray casting algorithm** and **edge detection** to determine whether a point lies inside or outside.

File Structure

point_in_shape.h: Header for core algorithm

point_in_shape.cpp: Function implementations

point_in_shape_main.cpp: Main driver (parses input and invokes logic)

test.h: Header for test algorithm

test.cpp: Test function implementations

point_in_shape_main_test.cpp: Main driver for tests

geometry_file: Folder containing data to test. File "shape.txt" consist of the shape given in the assigment.

□ Build Instructions

Requirements:

- C++17 compiler (`g++`, `clang++`, etc.)

Build using g++:

```
cd cadenceSD_test
g++ -std=c++17 point_in_shape.cpp point_in_shape_main.cpp -o point_in_shape_check
```

Usage

This tool checks if a given point (x, y) lies inside a polygon (with optional cutouts) defined in **shape_file**.

```
./point_in_shape_check <shape_file> <x> <y>
```

Example:

```
./point_in_shape_check geometry_files/shape.txt 25000 5000
```

Building and Running Tests

The test cases included are comprehensive. They check if the programme is behaving in the expected manner. Some of the test data is included in the folder **geometry_files**.

```
g++ -std=c++17 point_in_shape_test.cpp test.cpp point_in_shape.cpp -o point_in_shape_test
```

```
./point_in_shape_test
```

Algorithm Design

Problem Statement

Determine whether a test point P lies inside or outside of a polygonal region defined by an outer boundary (outline) and optional cutouts (holes).

Assumptions

- INSIDE
  - A point is considered inside if it lies within the polygon region or on its edge.
- OUTSIDE
  - A point is considered outside if it lies outside the outer polygon, or
  - Inside any cutout polygon (holes).
- Cutouts
  - Are assumed to be fully enclosed within the outer polygon.
  - Do not overlap with each other.

Solution Overview

1. Step 1: Check the outer boundary

   - We first check if the P is located on the outer polygon's edge using **edge detection**.
   - If not, then we use **ray casting** to find the point is inside or outside the outer polygon.

2. Step 2: Check the cutout regions

   - If the point is inside the outline, we check if it lies within any of the cutouts one by one using the same **edge detection** and **ray casting**.
   - If the point lies inside any cutout, it is considered outside the final shape.

Ray Casting Algorithm

The core logic uses the **ray casting** method (source: https://en.wikipedia.org/wiki/Point_in_polygon). It determines point inclusion by counting edge crossings:

- Cast a horizontal ray from the test point to the right (chosen direction).
- Count how the number of times it intersects the polygon's edges.
- If the count is **odd** -> point is **inside**.
- If the count is **even** -> point is **outside**.
- If it lies **exactly on an edge or vertex**, it's considered **on the boundary** (special handling is required which is done using **edge detection**).

Special cases which were handled and tested to give the correct output:

- When the ray intersects with a vertex. This could lead to multiple counts. Measures added to count only once.
- When the ray lies along the edge. The algorithm ignores that edge.
- When the ray intersects at the shared endpoint of two edges (e.g., the end of one and start of another), depending on the edge ordering, it may be miscounted. Only one crossing is registered per true upward or downward crossing. The algorithm ensures consistent direction-aware handling so each vertex contributes to at most one count.

Edge Detection Algorithm

To determine if a point lies exactly on an edge, the following checks are applied:

- Collinearity: The point must lie on the infinite line defined by the segment.
- Bounding box test: The point must lie between the segment endpoints.

This also handles floating-point errors and avoids misclassification near edge boundaries.

Limitations & Exceptions

The current implementation assumes well-formed, non-overlapping polygons and cutouts. The following edge cases are not fully supported:

- Overlapping Cutouts
  - If the cutouts overlap. Then the point located on the overlaping edges will be classified as inside. This in some cases may not be true.
- Edge Overlap between Outline and Cutout
  - If a cutout shares an edge with the outer polygon, points on that shared edge fall into an ambiguous zone.
  - This implementation treats such points as inside, which may not always reflect the intended geometry.

Author

Aditi Roy (23/05/2025)