Triangulation & Point Location

Xiaoqian Mu, Yuechuan Xue

In this project, we implement the sweepline triangulation and point location algorithm using C++ 11. DCEL structure is employed to describe the triangle, polygon and mesh result. To make use of the balanced binary tree, we implemented the red-black tree and made some changes to fit the persistent data structure in point location algorithm. In the end, all of the code are fully tested and those special cases are also well handled. OpenCV is used to for visualization and user interaction.

Requirement

- g++/clang/MSVC
- cmake (optional)
- OpenCV 2/3 (for visualization)

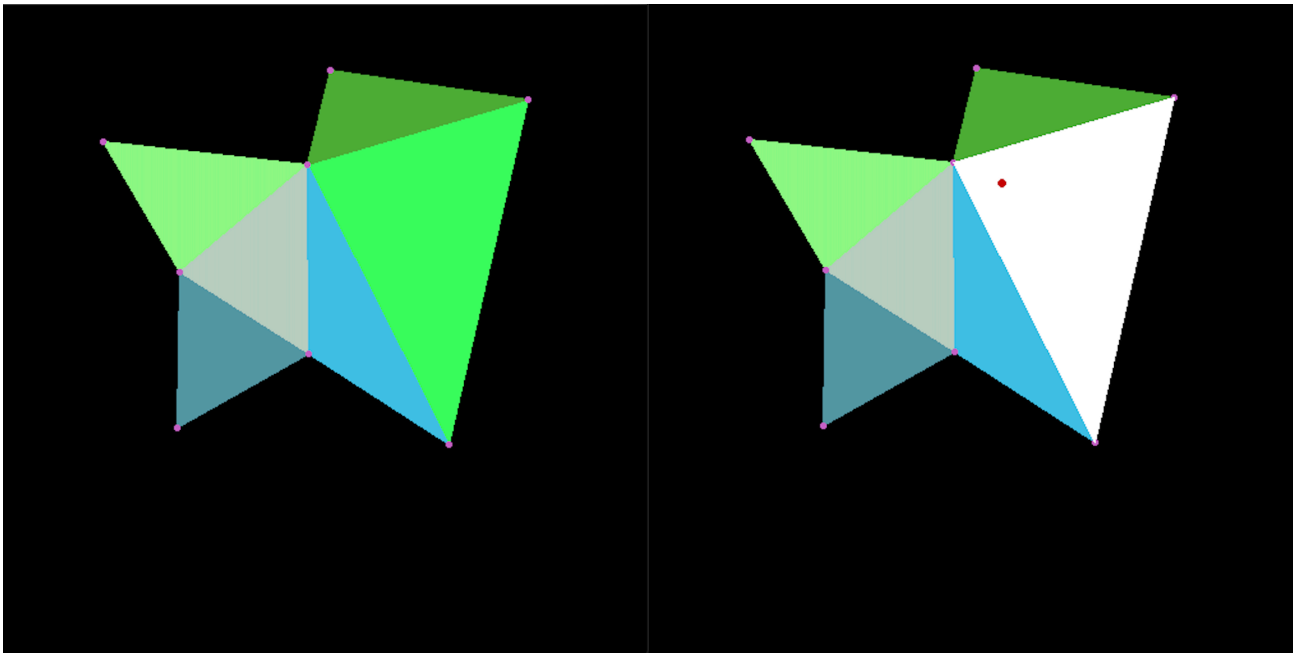
Build

```
cd PathToProject
mkdir build
cd build
cmake ..
make
```

Instructions

To test this project, you can either build the project by yourself or use our prebuild program under `build` folder, which contains both Windows x64 and Mac OS x64 version. Follow the steps listed below, the program will do triangulation and point location automatically.

1. Click 8 points **clockwisely** to determine the vertex of polygon and make sure not to make cross segments.
2. The program will show the result of triangulation automatically by dyeing these triangles with random color.
3. Click 1 point to determine the query point for point location procedure in or out of the polygon
4. The triangle containing the query point will become white or report query point not in the polygon in terminal.



Example

```
#include <iostream>
#include "geo_type.hpp"
#include "geo_dcel.hpp"
#include "geo_sweepline.hpp"
#include "geo_dcel_vertex.hpp"
#include "point_location.h"

using namespace geo;

void print_triangle(Triangle& triangle)
{
    std::cout << triangle.point1.x << " " << triangle.point1.y << std::endl;
    std::cout << triangle.point2.x << " " << triangle.point2.y << std::endl;
    std::cout << triangle.point3.x << " " << triangle.point3.y << std::endl;
}

int main(int argc, const char * argv[])
{
    // Initialization
    DoubleEdgeList dcel;
    std::vector<Vector2> points = {
        {400, 150}, {350, 350}, {300, 250}, {250, 300}, {300, 400},
        {200, 450}, {100, 350}, {200, 200}, {150, 150}, {250, 200},
        {300, 100}};
```

```
25
26      SweepLine sweepline;
27      std::vector<Triangle> triangles;
28      // Sweepline triangulation
29      // Input:  points
30      // Output: triangles and DCEL strucure
31      sweepline.triangulate(points, triangles, dcel);
32
33      // Print output triangles
34      for (auto triangle : triangles) {
35          print_triangle(triangle);
36          std::cout << std::endl;
37      }
38
39      // Point location initialization
40      struct sort_x {
41          bool operator()(const DoubleEdgeListVertex* v1, const
    DoubleEdgeListVertex* v2){
42              return v1->point.x<v2->point.x;
43          }
44      };
45      dcel.vertices.sort(sort_x());
46      Triangle triangle;
47
48      // Define query point
49      Vector2 query_point(230, 255);
50      PointLocation pl = PointLocation(&dcel);
51      if (pl.Find_point_location(query_point, triangle))
52      {
53          std::cout << "Find the query point: " << std::endl;
54          print_triangle(triangle);
55      }
56      else
57          std::cout << "Out of the polygon region." << std::endl;
58
59      return 0;
60  }
```

Credit

- https://en.wikipedia.org/wiki/Red_black_tree
- De Berg, Mark, et al. "Computational geometry." *Computational geometry*. Springer Berlin Heidelberg, 2000. 1-17.
- Sarnak, Neil, and Robert E. Tarjan. "Planar point location using persistent search trees." *Communications of the ACM* 29.7 (1986): 669-679.