MCS 481 / David Dumas / Spring 2011

Project 3: Point Location

Due Friday, April 1

0. Overview

In this project you will study CGAL's support for point location in planar subdivisions. While the project guidelines will allow you to focus more on programming or on timing experiments, according to your preference, this project does not have separate "coding" and "experimental" options.

1. Point Location Experiments

CGAL supports polygonal planar subdivisions as special cases of 2-dimensional *arrangements*. Several methods are provided for answering point location queries, including the randomized incremental construction of a trapezoidal map and search structure discussed in lecture.

Study one or more of the point location strategies offered by CGAL for a number of planar subdivisions. Experiment with subdivisions of different sizes, collecting data on both construction and query time. Compare these to the expected construction and query cost.

The CGAL examples do not include a program that can perform point location queries on an arbitrary subdivision, nor are there any programs for generating subdivisions with large complexity. Therefore, the key tasks for this project are:

(1) Create a program that takes a subdivision and list of query points as input and uses CGAL to answer the point location queries
(2) Generate subdivisions and query points for testing
(3) Collect and analyze timing data

For task (1), I have written a simple point location demo program (`point-location-query`) that you can use to test CGAL's implementation of the trapezoidal method. This program gives only basic location information, reporting whether a point lies in a bounded face of the subdivision or not. You are not required to use `point-location-query`; feel free to write your own program, or to modify this one to suit your needs.

For task (2), a simple solution would be to adapt a random polygon generator from project 2 (e.g. `rand-ngon-cgal`) to generate single-polygon planar subdivisions with a given number of edges. You would only need to change the output format of the polygon generator to match the requirements of the point location program. Random query points could be generated by one of the programs from project 1.

If you have a programming background, a more interesting approach to task (2) would be to write your own program for generating planar subdivisions and query points. You could try to generate subdivisions with many faces, with topologically complex faces (many holes), or with properties that lead to worst-case behavior for certain point location algorithms.

The actual experiments (task 3) follow the same general pattern as the experimental options in projects 1 and 2. The new feature here is that there are both construction and query times to measure and compare to theory. For meaningful query time data,

Project description v1.0 (March 13, 2011)

you should perform many queries on a given subdivision and report the average time per query; use sufficiently many points so that the total time to answer all location queries is at least a few seconds.

Balance. This project description is deliberately open-ended, giving you quite a bit of flexibility to do more or less coding, or to study aspects of the point location capabilities of CGAL that interest you. Some constraint is necessary, however, in order to ensure balance between different ways of completing the project.

Therefore, in order to be eligible for full credit, your submission must include *at least one* of the following components:

- A custom subdivision and query generator program (i.e. writing your own program instead of re-using `rand-ngon-cgal` with minor changes)
- Comparison of the trapezoidal point location method and CGAL's "landmark" method. (Compare them in terms of both theoretical and observed characteristics. The landmark method can be enabled with minor modifications to `point-location-query`.)

What to submit. Write a report describing how you completed the tasks (1)-(3) listed above, presenting your results, and analyzing the data. If you wrote any programs as part of the project, describe your design and any important implementation decisions you had to make. If you modified any CGAL example programs or code from this or other project descriptions, describe your modifications and why they were necessary.

Submit the report in class or by email. Send me an archive by email containing the source code for all of the programs you used or modified. Do not include any binaries in your email attachment.

2. Point Location Demo Program

The source code for `point-location-query` is available from the course web page. This program uses CGAL to answer simple point location queries on an arbitrary polygonal planar subdivision. As mentioned above, you may choose to use or modify this program for timing experiments. This section documents the program's function and its input and output formats.

The program takes two command-line arguments, a *subdivision file* and a *query file*. For example, in a Linux terminal the command

```
./point-location-query subdiv.txt queries.txt > output.txt
```

would read a subdivision from `subdiv.txt`, create a DCEL representation and a search structure, read query points from `queries.txt`, and write point location information for each of these query points to `output.txt`. Timing information and a few messages about the input will appear on `stderr`. The file formats are described below.

Subdivision file format. The planar subdivision is specified in a text file containing a list of vertices followed by a list of edges. The vertices are given as pairs of coordinates separated by whitespace, one pair per line. The first blank line in the file signals the end of the vertex list, and subsequent non-blank lines are considered as edges. An edge is specified by a pair of zero-based indices of its endpoints in the vertex list. Any line in the file that begins with the character `#` is considered a comment, and is ignored.

The edges in the subdivision must be non-crossing, i.e. any intersection between a pair of edges must be a vertex of both of them.

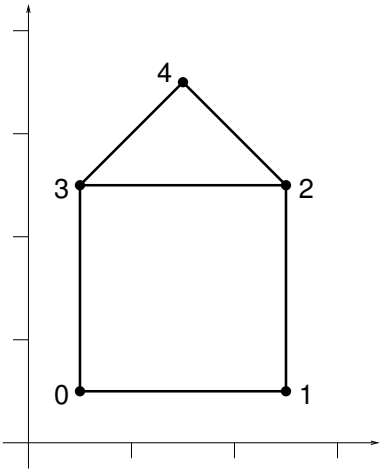
Here is a sample subdivision file:

```
0.5 0.5
2.5 0.5
2.5 2.5
0.5 2.5
1.5 3.5

0 1
1 2
2 3
3 0
2 4
3 4
```

The following file represents the same subdivision of the last one, but it contains comments:

```
# Simple example subdivision
#
# Vertices begin on the next line
0.5 0.5
2.5 0.5
2.5 2.5
0.5 2.5
1.5 3.5

# Edges start on next line (since previous line is blank)
0 1
1 2
2 3
3 0
2 4
3 4
```

The subdivision represented by these files is shown below; there are five vertices, six edges, and three faces (the unbounded face, a triangle, and a square). In the diagram, vertices are labeled by their indices.

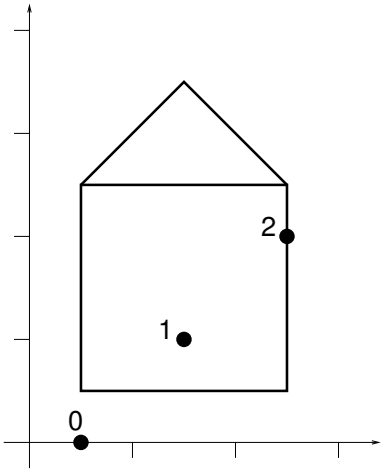


Query file format. The query file is a simple list of query points. The points are given as pairs of coordinates separated by white space, one pair per line. Blank lines are ignored, and unlike the subdivision file, comments are not allowed.

Here is a sample query file:

```
0.5 0.0
1.5 1.0
2.5 2.0
```

The following diagram shows these query points (and their zero-based indices in the query point list) overlaid on the subdivision example considered above:



Output format. For each query point, a single line is written to `stdout` containing the zero-based index of the query point, its coordinates, and some information about its location in the subdivision. If the point lies in the interior of a face of the subdivision, it is reported whether this face is bounded or unbounded, along with the total the number of holes in the face. If the point lies on an edge or vertex of the subdivision, then this fact is reported (without distinguishing between the vertex and edge cases).

For example, when given the sample subdivision and query files above, the program `point-location-query` writes the following to `stdout`:

```
Query 0: (0.5,0) is located in the unbounded face (which has 1 holes)
Query 1: (1.5,1) is located in a bounded face (which has 0 holes)
Query 2: (2.5,2) lies on an edge or vertex of the subdivision.
```

The demo program also writes a few messages to `stderr`. These messages indicate how many vertices and edges were read from the subdivision file and the total processing time for each of three key computational steps in the program:

(1) The construction of a DCEL from the subdivision file
(2) The construction of a search structure (and trapezoidal map) for the subdivision
(3) Answering the point location queries

For example, when given the sample subdivision and query files above, the program `point-location-query` might write the following to `stderr`:

```
Read 5 vertices.
Read 6 edges.
TIMING: 0 seconds - build DCEL
TIMING: 0 seconds - build search structure
TIMING: 0 seconds - queries
```

Note that timing results will depend on the computer used for the tests and on the random shuffle of the input segments. This example has only a few vertices and query points, so on most systems the construction and query times will be smaller than the resolution of the `C` library function `clock()` used for timing (and will be reported as zero).