Assignment 3: Implementation of ear clipping method for simple polygon triangulation [16 points]

Computational Geometry course at DCGI FEE CTU, winter 2020
Petr Felkel, Vojtěch Bubník

This is the third exercise for the Computational Geometry class. Its first goal is to implement an ear clipping method to triangulate a simple planar polygon (Jordan polygon). Its second goal is to convert the triangulation to Delaunay triangulation.

You can choose from two variants of the implementation:

1. Implementation using a simple data structure – list of triangles as triples of point indices without any other topology information stored, or
2. Implementation using the DCEL representation from an open-source library, OpenMesh.

In the first variant, the ear cutting is more straightforward to implement as we do not need to update information about the topology in the DCEL data structure. However, the Delaunay triangulation is a more complex task as the topology information is missing.

In the second variant with the OpenMesh library, it is harder to implement the ear cutting, but the Delaunay triangulation is more straightforward.

Ear clipping triangulation [11 points]

One way to triangulate a simple polygon is based on the fact that any simple polygon with at least four vertices without holes has at least two "ears." Ears are triangles with two sides being the edges of the polygon and the third one entirely inside it (i.e., a diagonal not crossing any other polygon edge. In other words, the ear does not contain any other simple polygon vertex) [1]. The algorithm consists of finding such an ear, removing it from the polygon (which results in a new simple polygon having at least two ears), and repeating these steps until there is only one triangle left [2].

This algorithm is easy to implement but slower than some other algorithms, and it only works on polygons without holes. This method is known as *ear clipping* and sometimes *ear trimming*. An efficient algorithm for cutting off ears in linear time each discovered Hossam ElGindy, Hazel Everett, and Godfried Toussaint [3].

Delaunay retriangulation [5 points]

Take the triangulation made by ear cutting as an input and convert it to Delaunay triangulation by iterative testing of inner edges and edge flipping of non-Delaunay edges. Use the `inCircle` predicate to check the Delaunay conditions.

Open Mesh library

OpenMesh [4], [5] is a generic and efficient data structure for representing and manipulating polygonal meshes. OpenMesh is developed at the Computer Graphics Group, RWTH Aachen. The development was funded by the German Ministry of Research and Education (BMBF). OpenMesh makes use of the GNU Lesser General Public License v3.

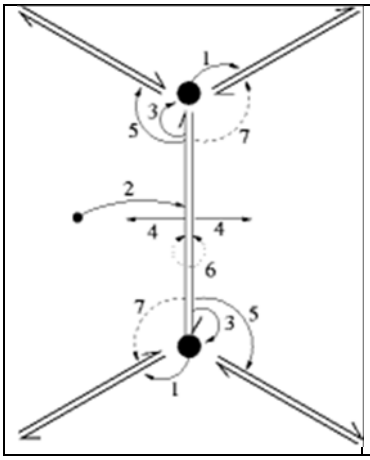
OpenMesh was designed with the following goals in mind:

- Flexibility: provide a basis for many different algorithms without the need for adaptation.
- Efficiency: maximize time efficiency while keeping memory usage as low as possible.
- Ease of use: wrap a complex internal structure in an easy-to-use interface.

For a detailed introduction, read the subsections of "Using and understanding OpenMesh" [6]. You can find some essential pieces of information in the following text.

The Halfedge Data Structure

OpenMesh uses a halfedge-based mesh representation based on Linked lists or arrays – a variant of the DCEL data structure described in the lecture. It provides an explicit representation of vertices, halfedges, edges, and faces. It uses the following convention:



- Each **vertex** references one outgoing halfedge, i.e., one of the halfedges that start at this vertex (1).
- Each **face** references one of the halfedges bounding it (2).
- Each **halfedge** provides a handle to
  - the vertex the halfedge points to (3),
  - the face the halfedge belongs to (4)
  - the next halfedge inside the face (ordered counter-clockwise) (5),
  - the opposite halfedge (6),
  - (Optionally: the previous halfedge in the face (7)).

Figure 1. DCEL data structure in OpenMesh library

Note to (1):
For efficiency reasons, a boundary vertex references a boundary edge (such edge belongs (4) to a surrounding face, i.e., it has the surrounding face on its left side).

Note to the storage of halfedges:

The halfedges are stored in pairs in the edge table where each Edge contains an array of two opposite halfedges:

```
class Edge
{
  friend class ArrayKernel;
  Halfedge  halfedges_[2];
};
```

Storing in pairs allows to address the individual halfedges by the lowest significant bit of HalfedgeHandle and simultaneously address a complete Edge (a pair of halfedges) by right-shifted HalfedgeHandle.
The EdgeHandle contains an index to the table of Edges.
The HalfedgeHandle index is created from the EdgeHandle index by left bit shift and the addition of the last bit 0 or 1

> The `mesh.halfedge_handle(*edgeIt, 0) //` returns the first halfedge of the edge `*edgeIt`.
> The `mesh.halfedge_handle(*edgeIt, 1) //` returns the second halfedge of the edge `*edgeIt`.

Useful OpenMesh methods:

Having different object handles (hh=halfedge handle, fh = face handle, vh = vertex handle, etc.), the useful get and set methods of the mesh object are (see Figure 1 as a reference for member numbers). We omit the word handle in the description of parameters, so "vertex" means "vertex handle," etc. See the OpenMesh documentation [6], [7] for details:

(1) Each **vertex** references one outgoing halfedge, i.e., one of the halfedges that start at this vertex (1).

```
mesh.halfedge_handle(vh)         // get one halfedge starting in vertex vh
mesh.set_halfedge_handle(vh, hh) // set this halfedge
```

(2) Each **face** references one of the halfedges bounding it

```
mesh.halfedge_handle(fh)      // get one of the face halfedges
mesh.set_face_handle(hh, fh)  // set one of the face halfedges
```

Each **halfedge** provides a handle to:

(3) the vertex the halfedge points to

```
mesh.to_vertex_handle(hh)            // get vertex to which hh points
mesh.set_vertex_handle(hh, to_vh);   // set vertex to which hh points
```

and additionally

```
mesh.from_vertex_handle(hh)   // get starting vertex of hh (to of opposite)
```

(4) the face the halfedge belongs to

```
mesh.face_handle(hh)          // get face left from the edge
set_face_handle(hh, fh)       // set face left from the edge
```

(5) the next halfedge inside the face (ordered counter-clockwise)

```
mesh.next_halfedge_handle(hh) // get next CCW edge
mesh.set_next_halfedge_handle(hh, hhNext)   // set pointer to next CCW edge
```

(6) the opposite halfedge

```
mesh.opposite_halfedge_handle(hh)           // the twin halfedge
```

(7) the previous halfedge in the face

```
mesh.prev_halfedge_handle(hh) // handle of the previous CCW edge
mesh.set_prev_halfedge_handle(hh, hhPrev)  // set pointer to prev CCW edge
```

See also:

```
mesh.point(vh);  // get position for vertex handle vh
mesh.insert_edge(hhFrom, hhTo)  // insert an edge between to_vh(hhFrom) and from_vh(hhTo)
```

The tasks:

1) Implement the ear clipping method and test it on provided sets of points [11 points]

Implement the ear clipping method using the chosen data structure. Submit the resulting modified main.cpp file.

3-ears_width_OpenMesh.zip
: The version using OpenMesh implementation of DCEL data structure – please, unpack the included OpenMesh.zip

3-ears_no_OpenMesh.zip
: The version using the list of triangles as triples of point indices without any other topology information stored

2) Try to find two simple polygons where naïve and precise predicates give different results and explain them [bonus 2 points]

3) Implement the Constraint Delaunay re-triangulation [5 points]

Perform the edge-flip operation for inner edges until no edge has a positive inCircle test.

Links

[1] Meisters, G. H., "Polygons have ears," American Mathematical Monthly, June/July 1975, pp. 648-651.
http://www.cgeo.ulg.ac.be/CG/polygons_have_ears.pdf

[2] Polygon Triangulation. http://en.wikipedia.org/wiki/Polygon_triangulation

[3] ElGindy, H., Everett, H., and Toussaint, G. T., (1993) "Slicing an ear using prune-and-search," *Pattern Recognition Letters*, **14**, (9):719–722.
http://www-cgrl.cs.mcgill.ca/~godfried/publications/ear.pdf

[4] Botsch, Mario; S. Steinberg; S. Bischoff; L. Kobbelt: OpenMesh - a generic and efficient polygon mesh data structure, 1st OpenSG Symposium, 2002.
http://www.graphics.rwth-aachen.de/media/papers/openmesh1.pdf

[5] What is OpenMesh?
http://www.openmesh.org/intro/

[6] Using and understanding OpenMesh
http://www.openmesh.org/media/Documentations/OpenMesh-Doc-Latest/a00006.html

[7] OpenMesh::Concepts::KernelT< FinalMeshItems > Class Template Reference
http://www.openmesh.org/media/Documentations/OpenMesh-Doc-Latest/a00181.html