Guidelines for using the library

In this document you will find the guidelines for using the library on a Linux system.

1 Console execution

Access to the directory cgal and do the following:

```
$ make
$ ./boolop file_subject file_clipping
```

where:

- `file_subject` is the file containing the subject polygon
- `file_clipping` is the file containing the clipping polygon
- `file_result` will be used to save the result of the operation
- the last parameter is optional and indicates the kind of Boolean operation:
  - I stands for Intersection (the default operation)
  - U stands for Union
  - D stands for Difference between the subject and clipping polygons
  - X stands for eXclusive OR (Symmetric difference) between the subject and clipping polygons

The implementation is based on CGAL, so you need this library for a correct compilation. In the directory *cpp* you can find a non-robust implementation of the algorithm using only C++ code.

2 Format of polygon files

You can find polygons in the directory *polygons*. In the directory *polygons/random* you will find random non-self-intersecting polygons up to 10000 vertices. You can, for example, try:

```
$ ./boolop ../../polygons/random/p10000-0 ../../polygons/random/p1000-0
```

You can also create your own polygons. A text file containing a polygon has the following structure:

```
<number of contours>
<number of vertices of first contour>
<vertex-list of first contour>
<number of vertices of second contour>
<vertex-list of second contour>
...
<contourId>: <firstHole_contourId> <secondHole_contourId> ...
...
```

Next, you can see the content of the file *polygonwithtwocontours*, that belongs to the directory *polygons/samples*—see Figure 1:

```
2
3
   0.1 0.1
   0.3 0.1
   0.2 0.3
3
   0.6 0.1
   0.8 0.1
   0.7 0.3
```

Figure 1: Polygon with two contours.

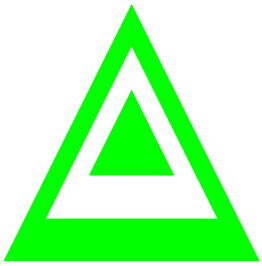

Figure 2: Polygon with holes.

As an example of a text file representing a polygon with holes you can see the file *polygonwithholes*, that belongs to the directory *polygons/samples*—see Figure 2:

```
3
3
-0.15 -0.15
0.45 -0.15
0.15 0.45
3
-0.05 -0.05
0.15 0.35
0.35 -0.05
3
0.05 0.05
0.25 0.05
0.15 0.25
0: 1
1: 2
```

The line `0: 1` means that the first specified contour has a hole: the sencond specified contour. The line `1: 2` means that the second specified contour has a hole: the third specified contour. Contours's IDs start from 0.

The algorithm works fine with polygons which holes are not specified. Anyway, the algorithm computes holes information, and writes text files representing polygons with holes information.

3 Admitted input polygons

The following features are allowed in input polygons:

- Contours can be described in clockwise or counterclockwise order.
- Polygons can have holes. However, you do not need to specify what contour are holes.
- A vertex of a polygon can touch (in a point) a vertex or edge of the same polygon.
- Self-intersecting polygons.

The following features are not allowed in input polygons:

- Overlapping edges (the intersection of two edges of the same polygon can be a point, but cannot be a segment).

The result polygon computed by the algorithm contains holes information. Furthermore, external contours are counterclockwise oriented. If a result polygon contains holes then "odd depth holes" are clockwise oriented and "even depth holes" are counterclockwise oriented.

4 Rendering the result of Boolean operations

If you want to render the result of a Boolean operation you should have installed Qt and OpenGL and type:

```
$ cd qt
$ qmake qt.pro
$ make
$ ./qt
```

This program includes an interesting option to watch step by step the computation of the Boolean operation.

5 License Issues

If you want to use the CGAL based implementation you must consult the CGAL's license policy. The C++ implementation is public domain software. For the code based on Qt you must consult the Qt's license policy.

6 Performance and robustness

The code based on CGAL uses an exact kernel, so it should be robust. The code based on C++ is not robust, however is quite faster than the code based on CGAL.

7 Acknowledgements

I would like to thank Antonio J. Rueda for his ideas to the early versions of the code. I would also like to thank Christian Woltering for his corrections in the function Polygon::**operator**>> and for formulating a correct definition of the field posSL of the SweeEvent structure.