Handed out: 04/12/2010 Hand in: 16/12/2010

Computational Geometry - Exercise 4

This is the fourth exercise for the Computational Geometry class. After having gained some experience with CGAL and Qt, and in particular with convex hulls, you have the opportunity to dig into the `Arrangement_2` package of CGAL. You are provided with an application which demonstrates the concept of duality and arrangements. The task is to extend this application as described in detail in the following paragraphs. First, you should read chapter 8 in "Computational Geometry - Algorithms and Applications" from Mark de Berg and have a look at the `Arrangement_2` package in the official CGAL manual[1].

1 Compiling

At the beginning, you must unpack and extract the provided `exercise_4.rar` archive from the course web page. Compile it and scan read through the source code in order to get a better overview, you don't have to read it all in detail. You will have to write code at all the locations which have a `TODO` tag and there are some hints in the source code to provide help.

2 Description

The application provides the following four basic modes

```
Point and Lines
Point and Circles
Point and Segments
Discrepancy
```

and corresponding buttons and menu entries to switch between these modes. Furthermore, the first and the last mode provided the possibility to switch between the primal plane and the dual plane. The first mode, `Point and Lines`, is active when the program is started. It illustrates the concept of duality for a set of lines and a movable point, or a set of dual points and a movable dual line, respectively. You can generate a new set of lines (`Generate Lines`) and drag the orange point around the scene. Based upon where this point is located, the number of lines above the point are counted and colored differently. Similarly, in the dual plane, you are provided with a movable segment and the points below it are counted and colored differently compared to the points above the segment. Play with this mode and read the corresponding portions of the source code, this includes the CGAL definitions in `common_cgal_defs.h`, the complete `graphicsview2d.h` header file, and the methods `generateRandomPoints`, `addDualityPointsToScene`, and `computeLinesAbovePoint` contained in the `GraphicsView2D` class. Finally, you should also have a glance at the `GraphicsPointItem`, `GraphicsCircleItem`,

[1] http://www.cgal.org/Manual/

and the `GraphicsSegmentItem` class. You will have to extend the remaining modes as described in the following tasks.

3 Point and Circles - 2 points

All the code locations, which you have to implement, are commented with `TODO 1`, look at all of them in order to get a better understanding for this task. You must complete the `Point and Circles` mode. First you will have to create random circles, then you must add them to the scene, and finally color all the circles which contain the movable point. In order to create random circles, you must create a circle creator and a circle iterator with CGAL. These two have to be defined as typedefs at the beginning of the `graphicsview2d.h` header file. Look at the other type definitions next to them, this should give you a good idea as how to define them, you may also have to refer to the official CGAL manual. Then, implement the `generateRandomCircles` method, refer to existing methods such as `generateRandomPoints` and `generateRandomLines`. Finally, you have to complete the implementation of `computeCirclesAroundPoint`. For this, you should make use of the circle equation

$$r^2 = (x - c_x)^2 + (y - c_y)^2$$

and refer to the existing method `computeLinesAbovePoint` as well as to the official CGAL documentation of `Point_2` and `Circle_2`. The circles containing the movable point should be colored differently. When you are done, recompile the program and check your result by switching to the `Point and Circles` mode and hitting the `Generate Circles` button.

4 Point and Segments - 3 points

All the code locations, which you have to implement, are commented with `TODO 2`, look at all of them in order to get a better understanding for this task. You must complete the `Point and Segments` mode. Similar to the previous task, you must first implement the `generateRandomSegments` method. Then, you have to complete the `addSegmentArrangementToScene` method. The segment arrangement consists of vertices and segments. For this you should refer to the chapter about the `Arrangement_2` package in the official CGAL manual. This will show you how to work with the extended DCEL data structures in CGAL. Furthermore, there are very good code snippets which may be partially used for this exercise. When you have implemented these two methods, the arrangement of segments should be visible in the application after you switch to the `Point and Segments` mode and press the `Generate Arr Segments` button. Color the `GraphicsPointItem` and `GraphicsSegmentItem` objects as you prefer. (Hint: White might not be a wise choice.) Finally, finish the implementation of the `computeCellInArrangement` method. You have to use the point location concept defined at the beginning of `graphicsview2d.h`. You may also define another point location concept, but this is not necessary. The DCEL face containing the point should have all the bounding edges colored differently. When you are done, recompile the program and check your result by dragging the movable point around the segment arrangement.

5 Discrepancy - 4 points

All the code locations, which you have to implement, are commented with `TODO 3`, look at all of them in order to get a better understanding for this task. You must complete the `Discrepancy` mode. There is no movable point available and all the calculations are done once you hit the `Generate Arr Lines` button. The primal plane will show the line arrangement while the dual plane will draw the point set, for which we want to find the minimal discrepancy value. The resulting line arrangement vertex, which corresponds to the minimal discrepancy, and its dual line in the dual plane will be colored orange. The line arrangement consists of vertices, segments, and rays. You should color these primitives differently. First, you have to create and add a line arrangement to the scene by completing the `addLineArrangementToScene` method. This is more complicated than in the previous tasks, so be patient. When you are done with this part, recompile and start the application. Whenever you hit the `Generate Arr Lines` button, a new line arrangement should be visible in the scene. The last method you have to complete is called `computeMinimalDiscrepancy`.

6 What to hand in

You have to hand in your source code and the created executable. Please do so in the same structure as in the given `exercise_4.rar` and upload it to your folder on the course web page. Finally, you may also add a brief description (.txt, .pdf) of any peculiar problems you encountered during the implementation or special features you might have added to the program. You will be given nine points in total if all the task are fulfilled.

Have fun and always remember: At the very beginning, there was just a point set... ;-)