Python for INMGA

For the INMGA practical we use the programming language python. For visualization purposes OpenGL is used, callable from Python. It is assumed that you have basic knowledge of OpenGL. Hence, as a first step you have to get acquainted with python. A tutorial can be found at http://docs.python.org/tutorial/ Moreover, for these practicals a number of simple Python programs are provided: *example-list.py, example-fileio.py.* Understanding these programs means that your knowledge of Python is sufficient to start working on the assignments. In *example-OpenGL.py* a Python program is given that uses OpenGL. The Python program *example1.py* can be run by typing on the command line *python example1.py*

Assignment 1

A In the program *assignment1A.py* a number of random points are generated and sorted in the *x* direction. Between consecutive points line segments are drawn, i.e., a polygonal line is created. Calculate the length of the polygonal line and print the result.

B In the plane three points P1, P2, P3 are given. Give a method to determine whether, going from P1, via P2 to P3, you make a right turn, a left turn or no turn. Implement the method in a formal expression in Mathematica.

C In the program *assignment1C.py* a set P of random points is generated. Construct the convex hull of P and visualize the points and the convex hull. Use the pseudo code from *ConvexHull(P)* (so, not SlowConvexHull(P)) in chapter 1 of the book "Computational geometry, M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf". To be sure, the pseudo code is also given below. Also calculate the area of the convex hull.

D In assignment B and C you were supposed to work with floating point numbers. However, when P1, P2, P3 are almost collinear, the precision of floats (including doubles) may be insufficient, resulting in erroneous results. In that case it is better to work with the number type *rational*. To get some feeling for the robustness of calculations in floating point and rational number representations you will now do a small experiment. In this context, with floating point numbers the python float is meant, which is in fact what in C and C++ is called double. In python, a rational number consists of two integer values, the nominator and denominator. The number of digits in these numbers has no limit, i.e., they have as many digits as is required. The denominator is non-zero.

The actual experiment is as follows. Generate N lines, each given by y=a*x+b, with for example N=100. This means that N random numbers a and b have to be generated, and that they are stored.
Now calculate for every pair of lines, excluding identical pairs, the point p of intersection.

(Use Mathematica to derive formal expressions to determine the intersection point of two lines.) Check whether p coincides with both lines. Do this test for both number representations.
To get you started some code is given in program *assignment1D.py.* Report for both number types the number of points on both lines.

E Now do assignment 1C again, but when checking for a right or left turn use exact arithmetic, but only when you run the risk of getting wrong results when using floats. (This way of only doing exact computations when necessary is sometimes used in CGAL, a very sophisticated Computational Geometry Library.) Some code is given in *assigment1E.py.* This is a working program, just extend it with the appropriate code. To test your implementation on degenerate cases four methods are given to generate degenerate sets of points: generatePointsA(), generatePointsB(), generatePointsC() generatePointsD(). The last two give degenerate sets of points. Print for every point set the number of points on the convex hull.

```
Algorithm CONVEXHULL($P$)
Input. A set $P$ of points in the plane.
Output. A list containing the vertices of $\mathcal{CH}(P)$ in clockwise order.
1.  Sort the points by $x$-coordinate, resulting in a sequence $p_1, \dots, p_n$.
2.  Put the points $p_1$ and $p_2$ in a list $\mathcal{L}_{\text{upper}}$, with $p_1$ as the first point.
3.  for $i \leftarrow 3$ to $n$
4.      do Append $p_i$ to $\mathcal{L}_{\text{upper}}$.
5.          while $\mathcal{L}_{\text{upper}}$ contains more than two points and the last three points
                in $\mathcal{L}_{\text{upper}}$ do not make a right turn
6.              do Delete the middle of the last three points from $\mathcal{L}_{\text{upper}}$.
7.  Put the points $p_n$ and $p_{n-1}$ in a list $\mathcal{L}_{\text{lower}}$, with $p_n$ as the first point.
8.  for $i \leftarrow n-2$ downto 1
9.      do Append $p_i$ to $\mathcal{L}_{\text{lower}}$.
10.         while $\mathcal{L}_{\text{lower}}$ contains more than 2 points and the last three points
                in $\mathcal{L}_{\text{lower}}$ do not make a right turn
11.             do Delete the middle of the last three points from $\mathcal{L}_{\text{lower}}$.
12. Remove the first and the last point from $\mathcal{L}_{\text{lower}}$ to avoid duplication of the
    points where the upper and lower hull meet.
13. Append $\mathcal{L}_{\text{lower}}$ to $\mathcal{L}_{\text{upper}}$, and call the resulting list $\mathcal{L}$.
14. return $\mathcal{L}$
```