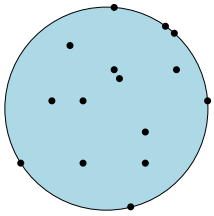
COP 4930/5930 – Computational Geometry[1]
Exam 2
Full grade: 20 points

1 Problem description



Given a set P of n points $p_1, p_2, \ldots, p_n$, the task is to find the *minimum enclosing disk* (MED) for P. The MED is the smallest disk (having the minimum possible radius) that contains all the points of P. See the figure for an example. It can be shown that MED for a pointset is unique. This problem finds its application in manufacturing industries and many other real-world scenarios. In this assignment, you are supposed to ***engineer*** two algorithms for this problem and experimentally compare their execution times.

The first one runs in $O(n^4)$ time. We are calling it **`MiniDiskNaive`**. This algorithm is based on a fairly basic observation. The MED contains either two points or three points on its boundary. So, we just brute-force over all point pairs and triplets and find the MED. Note that the MED must have the minimum possible diameter and encloses all the points in P. Can you think why it runs in $O(n^4)$ time? (no need to write an answer though).

The other one runs in $O(n)$ expected time. We are calling it **`MiniDiskIncremental`**. This algorithm is incremental, meaning the points are considered one at a time, and the MED is updated if required. Here is a pseudo-code for `MiniDiskIncremental`. D_i denotes the MED on the points $p_1, \ldots, p_i$. So in the very end, we return D_n. Note that this algorithm comprises three pseudo-codes, one calling the next. The recommended textbook and David Mount's lecture notes have this algorithm nicely explained.

Algorithm 1 : `MiniDiskIncremental(`P`)`

Compute a random permutation $p_1, \ldots, p_n$ of P;
Let D_2 be the MED for just the two points $\{p_1, p_2\}$;
for $i = 3$ to n **do**
  if $p_i \in D_{i-1}$ **then**
    $D_i = D_{i-1}$;
  else
    $D_i =$ `MiniDiskWithPoint`$(\{p_1, \ldots, p_{i-1}\}, p_i)$;
  end if
end for
return D_n;

[1] LaTeX *compiled on April 28, 2022 at 9:52am*

Algorithm 2 : `MiniDiskWithPoint`(P, q)

Compute a random permutation $p_1, \ldots, p_n$ of P;
Let D_1 be the MED for just the two points $\{q, p_1\}$;
for $j = 2$ to n **do**
  if $p_j \in D_{j-1}$ **then**
    $D_j = D_{j-1}$;
  else
    $D_j =$ `MiniDiskWith2Points`$(\{p_1, \ldots, p_{j-1}\},\ p_j,\ q\})$;
  end if
end for
return D_n;

Algorithm 3 : `MiniDiskWith2Points`(P, q_1, q_2)

Let D_0 be the MED for just the two points $\{q_1, q_2\}$;
for $k = 1$ to n **do**
  if $p_k \in D_{k-1}$ **then**
    $D_k = D_{k-1}$;
  else
    $D_k =$ the disk with q_1, q_2, p_k on its boundary;
  end if
end for
return D_n;

Implementation details. Please note that CGAL has a built-in `Circle` class. See here: `https://doc.cgal.org/latest/Kernel_23/classCGAL_1_1Circle__2.html`. Use it to work with circles. The constructors can be especially helpful for constructing circles with 2 and 3 points. For obtaining a random permutation use the built-in **`std::random_shuffle`** function from the `algorithm` header-file. Reference: `http://www.cplusplus.com/reference/algorithm/random_shuffle/`.

Graduate students only. You need to investigate what happens if, instead of randomly permuting the input in the second algorithm, you apply **`CGAL::spatial_sort`** on P inside the algorithms **`MiniDiskIncremental`** and **`MiniDiskWithPoint`**. Does running time increase, decrease, or stays roughly the same? You need to prepare graphs for this part showing the runtime plots for the two versions of **`MiniDiskIncremental`**: with and without spatial sort. Spatial sort usage example: `https://doc.cgal.org/latest/Spatial_sorting/Spatial_sorting_2small_example_delaunay_2_8cpp-example.html`. For graduate students, the implementation of the two algorithms is worth 15 points and 5 points for this comparative investigation.

Testing. You need to use pointsets generated randomly within a square for your experiments. Vary n from 100 to 1500. For every value of n, generate five samples and take the average of their runtimes. See the `tester()` function inside the `MED.h` header-file supplied to you. For visualization, you can use `QT`.

2 What to deliver?

1. **`MED.h`**: the following three incomplete functions must be carefully completed.
   (a) **`miniDiskNaive(...)`**
   (b) **`miniDiskIncremental(...)`**
2. Gnuplot runtime comparison graph for the two algorithms. Two plots in one graph.
3. ***Graduate students.*** Gnuplot runtime comparison graph for the two versions of `miniDiskIncremental` - with and without spatial sorting. Two plots in one graph.

Commenting is highly encouraged. Please test thoroughly before submission. You will be given zero if your program does not compile or gives unnecessary warnings. You are allowed to submit your solutions multiple times. We will grade your latest submission only. Your code will be graded using the CLion IDE on macOS/Linux.