Assignment 1 - Convex Hull and Plane Sweep Algorithm

Duy Pham - 0980384
Mazen Aly - 0978251
Pattarawat Chormai - 0978675

November 16, 2015

1

Algorithm 1 SmallConvexHull

```
Require: set of points $P$
  if $|P| < 5$ then
    return true
  end if
  Find $P_1$, $P_2$, the left-most and the right-most points from $P$
  Find $P_3 \in P$, which is the farthest point from the line $P_1P_2$
  Find $P_4 \in P$, which is the farthest point from the triangle $P_1P_2P_3$ and outside
  the triangle region.
  if $P4$ does not exist then
    return true
  end if
  if One point in $P$ is outside the polygon $P_1P_2P_3P_4$ then
    return false
  end if
  return true
```

Proof. We will prove that the algorithm returns the correct result.

The convex hull covers all of the points in the set (P), by definition. Therefore, it covers the left-most and the right-most points; so P_1 and P_2 belong to the resulting convex hull of the set P.

P_3 is the farthest point from the line P_1P_2. If P_3 does not belong to the convex hull, then the convex hull does not cover P_3. It contradicts the definition of the convex hull. Thus, P_3 must belong to the convex hull.

Similarly, $P4$ is the farthest point from the triangle $P_1P_2P_3$, which means $P4$ must belong to the convex hull.

If in the set P, there is a point outside of the polygon $P_1P_2P_3P_4$, then we need more points to construct the convex hull because these 4 points are proven to be in the resulting convex hull. Thus, the convex hull contains more than 4 vertices. Otherwise, the convex hull obviously contains less than 5 vertices.

Therefore, the algorithm is correct.

Now, we will prove that the algorithm runs in $O(n)$ time.

For finding $P_i, 1 \leq i \leq 4$, it takes $O(n)$ time.

For checking that if any point is outside of the polygon, it takes $O(n)$ time.

Therefore, the overall time complexity is $O(n)$. $\square$

3

(a)

Our rectangle is a region of size $2\delta \times \delta$.

Because δ is the smallest diameter among all three-disks, then the maximum distance among any three points must be at least some constant factor of δ. (Indeed, if the three points create an equilateral triangle, then the distance is $\frac{\sqrt(3)\cdot\delta}{2}$. If the third point is very close to the line created from the first 2 points, then the maximum distance is δ).

Now we try to put as many points as possible into the rectangle region, following the rule: there are no three-disks with diameters less than δ. It is obvious that we have to put the points at a smallest possible distance to each other.

To maximize the number of the points, we will put the first point P_1 in the center of the left border of the rectangle. Then we can insert the next point P_2 at a very close position to P_1 (much smaller than δ). Then the next point P_3 must be at least at a distance of δ from either P_1 or P_2. There is also the other case when we put P_2, P_3 in which $P_1P_2P_3$ forms an equilateral triangle, and the distance of the edges is $\frac{\sqrt(3)\cdot\delta}{2}$. If we put more points, there will be some 3 points that create a smaller disk, which violates the rule. So this is the best that we can do for the left half of the rectangle.

Similarly, we do the same thing for the other half of the rectangle, however we have to take care of the situation when there are 3 points close to the center of the rectangle which can form a smaller three-disk, which is also another constraint.

Therefore, the number of points in the rectangle is bounded by a constant.

(b)

Algorithm 2 PlaneSweepThree-Disk

```
Require: set of points $P$
  Initialize an event queue $Q$ and a status data structure $S$ storing $p_{i.x}$
  $\delta \leftarrow 0$
  Sort $P$ in $y$ value in descending order and put it into $Q$
  Construct the first three-disk from the first 3 elements $p_1, p_2$ and $p_3$ from $Q$
  and update $\delta$
  for $p_i \in Q$ where $3 < i < n$ do
      Remove $p_j$ from $S$ where $p_{j.y} > \delta + p_{i.y}$
      Add $p_{i.x}$ to $S$
      Find $R$, a set of points whose $x$ value is in the range $[p_{i.x} - \delta, p_{i.x} + \delta]$ from
  $S$
      if $|R| > 1$ then
          Find a smaller three-disk by trying all combinations of $R \cup \{p_i\}$ and
  update $\delta$
      end if
  end for
  report $\delta$
```

The detailed description is in algorithm 2. In this algorithm, we use an array for Q and a Binary Search Tree for S. Instead of comparing every combination, it scans from the top point to the bottom point and update the smallest three-disk at each step.

The algorithm returns the correct solution. If there is a smaller three-disk in the set of points, it should be covered when the algorithm handles the lowest point among the three, and δ will be updated.

The cost of sorting the events is $O(n \log n)$.

For each of the event point, each removal from S takes constant time, and the total removal time depends on the number of points to be deleted, the insertion also takes $O(\log n)$ time, the extraction of the values inside the rectangle depends on the number of items inside that range - which is bounded by a constant, the number of combinations needed to find a new diameter is also a constant (because the numbers of points inside the rectangle of size $2\delta \times \delta$ is bounded by a constant, as being proven in part (a)).

Let k be the number of points to be deleted at a time. In the normal case, if $k = O(\log n)$, the algorithm runs in $O(n \log n)$ time.

The degenerate case is when most of the points ($k = O(n)$) are inside the range δ (and the x-coordinates spread widely), and the next point is very far below, then the removal is costly because we need to remove k points after leaving that range. In such a case, it takes $O(n)$ to remove every item above the range of δ. Then the algorithm runs in $O(n^2)$ time.