Chapter 2

Two- and Three-Dimensional Convex Hulls

In this chapter, we present two $O(n \log h)$-time convex hull algorithms in the plane E^2, the second of which is also extended to E^3 with the same optimal time complexity. Although algorithms with this time complexity were known, our methods are simpler than the previous and also illustrate the basic ideas to be used in the rest of this thesis for constructing convex hulls in higher dimensions. In particular, the first planar convex hull algorithm we present, which can be considered as a simplification of the original optimal method of Kirkpatrick and Seidel [KS86], serves as the basis of the four-dimensional output-sensitive algorithm in the next chapter.

2.1 A Simplified "Ultimate Planar Convex Hull Algorithm"

This section describes a simple $O(n \log h)$ convex hull algorithm in the plane. Since our algorithm can be viewed as a simplification of Kirkpatrick and Seidel's planar convex hull algorithm, we first sketch here their method for comparison.

Given an n-point set $P \subseteq E^2$, we want to construct the convex hull of P. It suffices to compute just the *upper hull* of P, consisting of the sequence of hull edges that have an upward normal vector. Then the lower hull can be computed in a similar manner by reflection and the convex hull can be obtained by joining these two hulls.

Kirkpatrick and Seidel's algorithm constructs the upper hull of P as follows: (i) find a point $p^* \in P$ with the median x-coordinate, (ii) compute the edge $\overline{p_1 p_2}$ of the upper hull that intersects the vertical line through p^* ($x[p_1] < x[p^*] < x[p_2]$), and (iii) recursively compute the upper hull of all points left of (and including) p_1 and the upper hull of all points right of (and including) p_2.

To find the edge $\overline{p_1p_2}$ (the *bridge*) that intersects a given vertical line in step (ii), Kirkpatrick and Seidel used a prune-and-search procedure, similar to the prune-and-search linear programming algorithm of Dyer [Dye84] and Megiddo [Meg83b, Meg84]. (In fact, bridge-finding can be formulated as a linear program in dual space.) Here is a high-level description what is involved in this prune-and-search procedure: First, points are paired, the slope of the line through each pair is calculated, and the median slope m is computed. Then the upper-hull vertex p_m with a supporting line of slope m is found. A comparison involving p_m and the given vertical line is then performed, which allows one point in half of the pairs be pruned. This step eliminates 1/4 of the points and the procedure is repeated.

This ends our brief sketch of Kirkpatrick and Seidel's algorithm. As a summary, we can say that their algorithm has two levels: the lower level is a prune-and-search procedure, and on top of that is a divide-and-conquer method. Our main observation is that we can get a simpler algorithm if we combine these two levels into one, i.e., if we use pruning directly for divide-and-conquer rather than for searching. As a result, we can skip the step that computes the point p^* with median x-coordinate and avoid actually searching for the bridge at each recursive step.

2.1.1 The prune-and-divide algorithm in the plane

We now give the details of our simplified planar convex hull algorithm. As in Kirkpatrick and Seidel's algorithm, only the upper hull of the given n-point set $P \subseteq E^2$ is computed. We first pair the points of P arbitrarily and calculate the slope of the line through each pair. We then find the median slope m and compute the upper-hull vertex p_m that has a

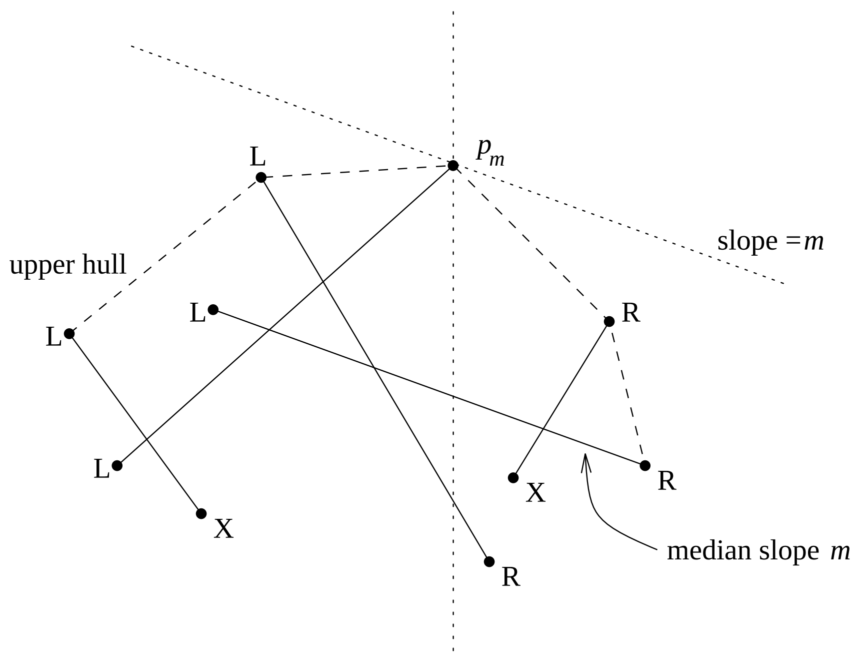

Figure 2.2: Pairing and pruning points in the plane. Points marked L belong to P_ℓ, points marked R belong to P_r, and points marked X belong to neither sets.

supporting line of slope m; this vertex can be computed by taking the maximum along a projection of P parallel to m. The x-coordinate of p_m is then used to divide P into two parts: P_ℓ, which contains p_m and all points to its left, and P_r, which contains p_m and all points to its right.

Now, if a pair has slope less than m, then the right point in the pair cannot participate in the upper hull of P_ℓ and thus can be pruned from P_ℓ. Similarly, if a pair has slope greater than m, then its left point in the pair cannot participate in the upper hull of P_r and can be pruned from P_r. Since half $(n/4)$ of the pairs have slope less than the median m and half have slope greater than m, pruning ensures that P_ℓ and P_r each contain at most $3n/4$ points. We then recursively compute the upper hull of P_ℓ and P_r. See Figure 2.2 for an example.

The pseudocode of the algorithm is given below. For convenience, we assume that the leftmost and rightmost points p_ℓ and p_r of P have been identified and we let n be the cardinality of the set $P^\bullet = P - \{p_\ell, p_r\}$ instead. In the interest of practical efficiency, line 1 has been added to the algorithm; it does not affect asymptotic worst-case performance.

```
Algorithm DivideHull2d($P^\bullet, p_\ell, p_r$)
[ Given $n$-point set $P^\bullet \subseteq E^2$ and points $p_\ell, p_r \in E^2$ such that $x[p_\ell] < x[p] < x[p_r]$ for
  all $p \in P^\bullet$, return the sequence of edges of the upper hull of $P = P^\bullet \cup \{p_\ell, p_r\}$. ]
1.  discard points from $P^\bullet$ that lie below $\overline{p_\ell p_r}$
2.  if $P^\bullet = \emptyset$ then return $\langle \overline{p_\ell p_r} \rangle$
    if $P^\bullet = \{p\}$ then return $\langle \overline{p_\ell p}, \overline{p p_r} \rangle$
3.  arbitrarily choose $\lfloor n/2 \rfloor$ disjoint pairs $\{\{s_1, t_1\}, \ldots, \{s_{\lfloor n/2 \rfloor}, t_{\lfloor n/2 \rfloor}\}\}$ from $P^\bullet$
        and order each pair so that $x[s_i] < x[t_i]$
4.  let $m_i = (y[t_i] - y[s_i]) \,/\, (x[t_i] - x[s_i]),\ \ i = 1, \ldots, \lfloor n/2 \rfloor$
        and $m$ = median of $\langle m_1, \ldots, m_{\lfloor n/2 \rfloor} \rangle$
5.  let $p_m$ = point in $P$ that maximizes $y[p_m] - m \cdot x[p_m]$
6.  let $P_\ell^\bullet = \{p \in P^\bullet : x[p] < x[p_m]\} - \{t_i : m_i \leq m\}$
        $P_r^\bullet = \{p \in P^\bullet : x[p] > x[p_m]\} - \{s_i : m_i \geq m\}$
7.  if $p_m = p_r$ then return DivideHull2d($P_\ell^\bullet, p_\ell, p_r$)
    if $p_m = p_\ell$ then return DivideHull2d($P_r^\bullet, p_\ell, p_r$)
    otherwise return the concatenation of
        DivideHull2d($P_\ell^\bullet, p_\ell, p_m$) and DivideHull2d($P_r^\bullet, p_m, p_r$)
```

Remark: It is not difficult to modify `DivideHull2d()` to work for point sets P not in general position. When there are more than one point in P that maximize $y[p_m] - m \cdot x[p_m]$ in line 5, we simply pick the leftmost one. When two points in a pair share the same x-coordinate, we can eliminate the bottom one.

2.1.2 Analysis of the prune-and-divide algorithm in the plane

Let $T(n, h)$ be the running time of algorithm `DivideHull2d()` on a point set with $n + 2$ points (i.e., n points excluding p_ℓ and p_r) and $h+1$ upper-hull vertices (i.e., h upper-hull edges). By noting that median-finding (line 4) can be done in linear time, we obtain the following recurrence for $T(n, h)$, where c denotes a constant:

$$T(n,h) \le \begin{cases} c & \text{if } n \le 1 \\ T(n_\ell, h) + cn & \text{if } n \ge 2 \text{ and } h_r = 0 \\ T(n_r, h) + cn & \text{if } n \ge 2 \text{ and } h_\ell = 0 \\ T(n_\ell, h_\ell) + T(n_r, h_r) + cn & \text{if } n \ge 2 \text{ and } h_\ell, h_r \ge 1 \end{cases}$$

for some $0 \le n_\ell,\, n_r \le \lceil 3n/4 \rceil$ and $h_\ell, h_r \ge 0$ with $n_\ell + n_r < n$ and $h_\ell + h_r = h$.

Using the concavity of the logarithm, one can then prove that $T(n, h) = O(n \log h)$ by induction. Here, we observe an alternative proof that is perhaps simpler as it avoids the use of induction. The proof is more general and provides better insight into recurrences of this kind by examining their *recursion trees.*

Let T be a rooted tree in which each node ν is assigned a cost $c(\nu) \in [0, \infty)$. We say that the cost function c is *α-fading* for a constant $\alpha \in (0, 1)$ if $c(\mu) \le \alpha\, c(\nu)$ for every node μ and its parent ν. As part of the analysis of their 3-d output-sensitive convex hull algorithm, Edelsbrunner and Shi [ES91, Lemma 3.1] proved that the total cost in such a tree is asymptotically bounded by the per-level cost times the logarithm of the number of nodes. Their proof uses a path compression operation that transforms T into a balanced tree. We give a simple, short proof of their result that avoids path compression altogether; we then improve the bound to depend on the number of leaves rather than the number of nodes.

Lemma 2.1.1 *In a recursion tree T with m nodes and ℓ leaves and an α-fading cost function c, if the sum of the costs at each level is bounded by C, then the sum of the costs of all nodes in T is (i) at most $C(\log_{1/\alpha} m + 2)$ and (ii) at most $C(\log_{1/\alpha} \ell + 1 + 1/(1-\alpha))$.*

Proof: Number the levels of the tree 0, 1, 2, ... with the root at level zero. Let $k = \lfloor \log_{1/\alpha} m \rfloor$. The sum of the costs at levels $0, 1, \ldots, k$ is bounded by $C(k+1) \leq C(\log_{1/\alpha} m + 1)$. Furthermore, by the α-fading property, each node on a level greater than k has cost bounded by $C\alpha^{k+1} \leq C/m$; hence, the sum of the costs at level $k+1, k+2, \ldots$ is bounded by C. Part (i) follows.

To prove part (ii), we choose $k = \lfloor \log_{1/\alpha} \ell \rfloor$ instead. As before, the sum of the costs at levels $0, 1, \ldots, k$ is bounded by $C(k+1) \leq C(\log_{1/\alpha} \ell + 1)$. Thus, we just have to account for the costs of nodes at levels greater than k. Note that each node belongs to some root-to-leaf path in T. By the α-fading property, the sum of the costs at levels $k+1, k+2, \ldots$ along such a path is bounded by

$$C\alpha^{k+1} + C\alpha^{k+2} + \ldots = \frac{C\alpha^{k+1}}{1-\alpha} \leq \frac{C}{(1-\alpha)\ell}.$$

Since there are ℓ root-to-leaf paths in total, the sum of the costs at levels $k+1, k+2, \ldots$ is bounded by $C/(1-\alpha)$. Part (ii) follows. □

With Lemma 2.1.1, it is now easy to show that the running time of algorithm `DivideHull2d`() is $O(n \log h)$. Consider the recursion tree generated by the calls to `DivideHull2d`(). It is clear that the sum of the costs at each level of the tree is bounded by cn and that the cost function satisfies the (3/4)-fading property. Since the number of leaves is at most h (as a new edge is discovered at every leaf), Lemma 2.1.1(ii) immediately implies that the total cost of the algorithm is bounded by $cn \log_{4/3} h + O(n)$.

The storage requirement of the algorithm is clearly linear. We have thus shown:

Theorem 2.1.2 *Algorithm* `DivideHull2d`() *computes the* $(h+1)$*-vertex upper hull of an* $(n+2)$*-point set* $P \subseteq E^2$ *in* $O(n \log h)$ *time and* $O(n)$ *space.*

Remarks:

1. Compared to the algorithm by Kirkpatrick and Seidel, `DivideHull2d`() is faster