Advanced Algorithms **Fall 2015**

Lecture 4: Geometric Algorithms (Convex Hull, Voronoi diagrams)

Faculty: K.R. Chowdhary *: Professor of CS*

4.1 Introduction

The excitement of computational geometry is due to a combination of factors: deep connections with classical mathematics and theoretical computer science on the one hand, and many ties with applications on the other. Indeed, the origins of the discipline clearly lie in geometric questions that arose in areas such as computer graphics and solid modeling, computer-aided design, robotics, computer vision, etc. Not only have these more applied areas been a source of problems and inspiration for computational geometry but, conversely, several techniques from computational geometry have been found useful in practice as well.

Nevertheless, many of the new algorithms are simple and practical to implement-it is only their analysis that requires advanced mathematical tools.

4.2 Convex Hull

Definition 4.1 Convex hull. *In mathematics, the convex hull or convex envelope of a set S of points $\Re^k$ in the Euclidean plane or Euclidean space is the smallest convex set that contains S.*

The convex hull of a set of points in $\Re^k$ is the most fundamental problem in computational geometry. Given a set of points, all we are interested in computing its *convex hull*, which is defined to be the smallest convex body containing these points. Of course, the first question one has to answer is how to represent the convex hull. An implicit representation is just to list all the extreme points, where as an explicit representation is to list all the extreme d-faces (also called $i-$faces) of dimensions $d = 0, 1, ..., k-1$.(see fig. 4.1).

Note that d-faces of dimension 0 are points, dimension 1 are lines, dimension 2 are panels of a cuboid, cube, cylinder, pyramid, etc. In figure 4.1, the $d-$faces of dimension 2 are the triangle faces: *abd*, *acd*, *bdc*, and *abc*.

Thus the complexity of any convex hull algorithm will have two parts, 1) *computation part* and *output part*.

For instance, when S is a bounded subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around S (see figure 4.2).

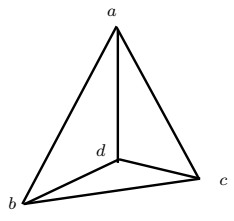

Figure 4.1: d-faces for $d = 2$ (4 d-faces for a pyramid).

4.2.1 Convex hull algorithms

Planar case. Consider the general case when the input to the algorithm is a finite unordered set of points on a Cartesian plane. An important special case is: points are given in the order of traversal of a simple polygon's boundary.

If all points are not on the same line, then their convex hull is a convex polygon whose vertices are some of the points in the input set. Its most common representation is the list of its vertices ordered along its boundary clockwise or counterclockwise. In some applications it is convenient to represent a convex polygon as an intersection of a set of *half-planes*, as discussed previously.

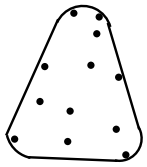

Figure 4.2: Convex hull (elastic band analogy).

For an arbitrary set of n points in two and three dimensions, we can compute the convex hull using the *Graham Scan, Gift-wrapping* and *divide-and-conquer* algorithms. Note that the convex hull of an arbitrary set of points in two dimensions is a *convex polygon*.

The **Graham scan** Algorithm:

1. Sorting the input set of points with respect to an interior point, say O, which is *centroid* of the first three nonlinear points. (The centroid for $(x_1, y_1), (x_2, y_3), (x_3, y_3)$ is $((x_1+x_2+x_3)/3, (y_1+y_2+y_3)/3)$, and sorting of every point (x_i, y_i) is with respect to the angle from centroid O.)
2. Connect these points into a star-shaped polygon P centered at O (see figure 4.3).
3. Performing linear scan of the sorted points to compute the convex hull of polygon.

The algorithm takes $O(n \log n)$ time as first step is dominating step, which requires the sorting of all the points. The figure 4.3 shows that points $(x_1, y_1), \ldots, (x_n, y_n)$, are sorted in the increasing order of the angle, with centroid centered at O. Note that only the farthest points are picked up to create a connecting boundary of the polygon, so that the convexity is maintained.

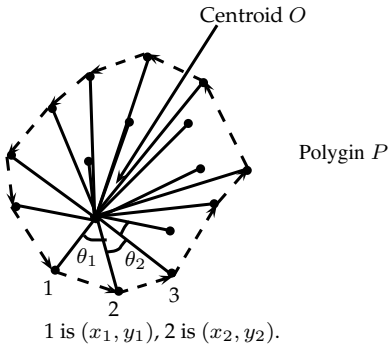

Figure 4.3: Graham scan algorithm geometry.

The **Gift-wrapping algorithm.**

One can also use the Gift-wrapping technique to compute the convex polygon. Starting with a vector of points that is known to be on the convex hull, say, the point O, with the smallest y-coordinate, we sweep a half-polygon. We then march to v_1, repeat the same process, and find next vertex v_2. This process terminates when we reach O again. This is similar to *wrapping an object with rope*. Finding the next vertex takes time proportional to the number of points remaining. Thus, total time is $O(n\mathcal{H})$, where $\mathcal{H}$ denotes the number of points in the convex polygon. This algorithm is more efficient (only $O(\log n)$), than Graham-scan algorithm, if the number of points on the convex polygon is small.

Theorem 4.2 *The convex hull of a set S of n points in $\Re^k$ can be computed in $O(n \log \mathcal{H})$ time for $k = 2$ or for $k = 3$. Here, $\mathcal{H}$ is number of* i-faces, *$i = 0, 1, k-1$.*

Assuming all your polygons are in counter-clockwise order, the moment your non-initial polar angle makes left turn, you know it's not convex. You could see if the line segments intersect with each other to figure out if the polygon is complex. Note that, in the figure 4.4 the angles $\alpha_i \geq 180°$.

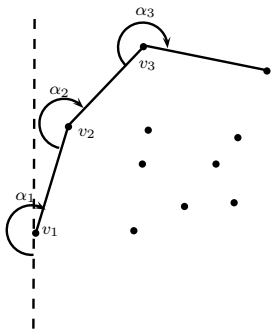

Figure 4.4: Gift wrapping Algorithm geometry.

4.3 Proximity

Geometric problems surrounds pertaining to the questions of how close two geometric entities are among a collection of objects or how similar two geometric patterns match each other. For example, in pattern classification and clustering, features that are similar according to some metric are to be clustered in a group. The two aircrafts that are the closest at any time instant in the air space will have the largest likelihood of collision with each other. In some cases one may be interested in how far apart or how dissimilar the objects are. Some of these proximity-related problems will be addressed in this section.

4.3.1 Closest Pair

Consider a set S of n points in $\Re^k$ (set of real numbers in k dimensional space). The closest pair problem is to find in S a pair whose distance is the minimum, i.e., find p_i and p_j, such that $d(p_i, p_j) = min_{k \neq l}\{d(p_k, p_l)$, for all point $p_k, p_l \in S\}$, where $d(a, b)$ denotes the Euclidean distance between a and b. The brute-force method takes $O(k.n^2)$ time by computing all $O(n^2)$ inter-point distances and taking the minimum; the pair that gives the minimum distance is the closest pair. Here, the k multiplier is due to dimension $d \geq 2$.

As is well-known, in one dimension, one can solve the problem much more efficiently: since the closest pair of points must occur consecutively on the real line, one can sort these points and then scan them in order to solve the closest pair problem in $O(n \log n)$ time. The time complexity turns out to be the best possible, since the problem has a lower bound of $\Omega(n \log n)$, following from a linear time transformation from the element uniqueness problem.

But unfortunately there is no total ordering for points in $\Re^k$ for $k \geq 2$, hence, sorting is not applicable. We will show that by using divide-and-conquer approach, one can solve this problem in $O(n \log n)$ optimal time. Let us consider the case when $k = 2$. In the following, we only compute the minimum distance between the closest pair; the actual identity of the closest pair that realizes the minimum distance can be found easily by some straightforward book-keeping operations. Consider a vertical separating line $\mathcal{V}$ that divides S into S_1 and S_2 such that $|S_1| = |S_2| = n/2$. Let δ_i denote the minimum distance defined by the closest pair of points in S_i, $i = 1, 2$. Observe that the minimum distance defined by the closest pair of points in S is either δ_1, δ_2, or $d(p, q)$ for some $p \in S_1$ and $q \in S_2$. In the former cases, we are done. In the latter, points p and q must lie in the vertical strip of width $\delta = min\{\delta_1, \delta_2\}$ on each side of the separating line $\mathcal{V}$ (see figure 4.5).

The problem now reduces to that of finding the closest pair between points in S_1 and S_2 that lie inside the strip $\mathcal{L}$ of width 2δ. This subset of points $\mathcal{L}$ possesses a special property, known as *sparsity*, i.e., for each square box (a hypercube of higher dimensions) of length 2δ the number of points in $\mathcal{L}$ is bounded by a constant $c = 4 \times 3^{k_1}$, since in each set S_i, there exists no point that lies in the interior of the δ-ball centered at each point in S_i, $i = 1, 2$.

4.4 Voronoi diagrams

In mathematics, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called *seeds, sites, or generators*) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called *Voronoi cells*. Put simply, it's a diagram created by taking pairs of points that are close together and drawing a line that is equidistant between them and perpendicular to the line connecting them. That is, all points on the lines in the diagram are equidistant to the nearest two (or more) source points.

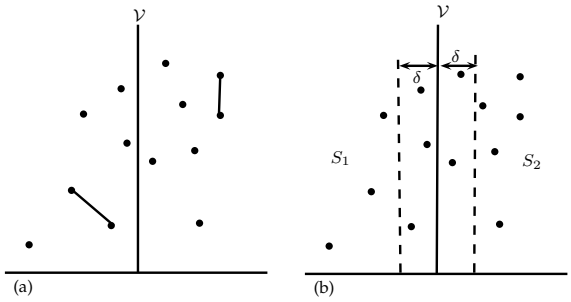

Figure 4.5: Divide-and-conquer scheme for closest pair problem. Solutions to subproblems S_1 and S_2 (a) and candidates must lie in the vertical strip of width δ on each side of $\mathcal{V}$ (b).

It is named after *Georgy Voronoi*, and is also called a Voronoi tessellation, a Voronoi decomposition, a Voronoi partition, or a Dirichlet tessellation (after Peter Gustav Lejeune Dirichlet). Voronoi diagrams have practical and theoretical applications to a large number of fields, mainly in science and technology but also including visual art.

Definition 4.3 *The Voronoi digram $\mathcal{V}(S)$ of set S of points, called $S = \{s_1, s_2, \ldots, s_n\}$ in $\Re^k$ is a partition of $\Re^k$ into Voronoi cells $V(S_i)$, $i = 1, 2, \ldots, n$, such that each cell contains points that are closer to site s_i than to any other site s_j, $j \neq i$, i.e.,*

$$V(s_i) = \{x \in \Re^k | d(x, s_i) \leq d(x, s_j) \forall s_j \in \Re^k, j \neq i\} \tag{4.1}$$

Figure 4.7(a) shows the Vornoi diagram of 16 point sites in two dimensions. Figure 4.7(b) shows the straight line dual graph of the Voronoi diagram, which is also called Delaunary triangulation.

In two dimensions, $\mathcal{V}(S)$ is a planar graph and is of size linear in $|S|$. In dimension $k \geq 2$, the total number of d-faces of dimensions $d = 0, 1, \ldots, k-1$, in $\mathcal{V}(S)$ is $O(n^{d/2})$.

Construction of Voronoi diagram in two Dimensions. The Voronoi diagram possesses many properties that are proximity related. For instance, the closest pair problem for S can be solved in linear time after the Voronoi diagram has been computed. Because this pair of points must be adjacent in the Denaunay triangulation, all one has to do is examine all adjacent pairs of points and report the pair with the smallest distance.

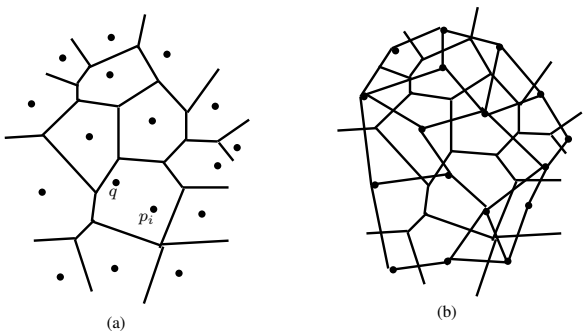

Figure 4.6: A Vornoi diagram of a set of 16 points in the plane.

Illustration. As a simple illustration, consider a group of shops in a city. Suppose we want to estimate the number of customers of a given shop. With all else being equal (price, products, quality of service, etc.), it is reasonable to assume that customers choose their preferred shop simply by distance

considerations: they will go to the shop located nearest to them. In this case the Voronoi cell R_k of a given shop P_k can be used for giving a rough estimate on the number of potential customers going to this shop (which is modeled by a point in our city).

For most cities, the distance between points can be measured using the familiar Euclidean distance:

$$\begin{aligned} l_2 &= d[(a_1, a_2), (b_1, b_2)] \\ &= \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \end{aligned}$$

4.4.1 Applications of Vornoi diagrams

1. In astrophysics, Voronoi diagrams are used to generate adaptive smoothing zones on images, adding signal fluxes on each one.

2. In epidemiology, Voronoi diagrams can be used to correlate sources of infections in epidemics. One of the early applications of Voronoi diagrams implemented, showed the correlation between residential areas on the map of Central London whose residents had been using a specific water pump, and the areas with most deaths due to the outbreak of epidemics.

3. A point location data structure can be built on top of the Voronoi diagram in order to answer nearest neighbor queries, where one wants to find the object that is closest to a given query point. Nearest neighbor queries have numerous applications. For example, one might want to find the nearest hospital, or the most similar object in a database. A large application is vector quantization, commonly used in data compression.

4. In hydrology, Voronoi diagrams are used to calculate the rainfall of an area, based on a series of point measurements.

5. In ecology, Voronoi diagrams are used to study the growth patterns of forests and forest canopies, and may also be helpful in developing predictive models for forest fires.

4.5 Arrangements

We start with : line arrangements on the plane. Intuitively, the problem is, given n straight lines on the plane, determine how they partition the plane and how the pieces fit together. (See Figure 3.) Formally, an arrangement consists of cells: 0-dimensional cells called "vertices," 1-dimensional cells called "edges," and 2-dimensional cells called "regions" or "faces."

Suppose we have n lines. How many vertices are there in the arrangement? Well, every two lines will intersect to produce a vertex, so the number is just ${}^nC_2$. At this point, the reader may object, "What if some lines are parallel or concurrent?" In our discussion, we will consider things to be in "general position." In this case, we assume that no 2 lines are parallel and no 3 lines are concurrent.

How many edges are there? Each line gets chopped into n edges by the other $n - 1$ lines, so there are ${}^nC_2$ total edges. How many faces? Each vertex is the bottom vertex of a face. This gives $({}^nC_2)$ faces. In addition, there are $n + 1$ faces at the bottom of the arrangement that do not have bottom vertices. Therefore, the total number of faces is just $({}^nC_2) + n + 1$.

In geometry an arrangement of lines is the partition of the plane formed by a collection of lines. Bounds on the complexity of arrangements have been studied in discrete geometry, and computational geometers have found algorithms for the efficient construction of arrangements.

For any set A of lines in the Euclidean plane, one can define an equivalence relation on the points of the plane according to which two points p and q are equivalent if, for every line l of A, either p and q are both on l or both belong to the same open half-plane bounded by l. When A is finite or locally finite, the equivalence classes of this relation are of three types:

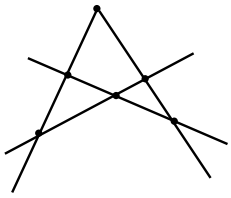

Figure 4.7: A Simple line arrangements.

1. the interiors of bounded or unbounded convex polygons (the cells of the arrangement), the connected components of the subset of the plane not contained in any of the lines of A,
2. open line segments and open infinite rays (the edges of the arrangement), the connected components of the points of a single line that do not belong to any other lines of A, and
3. single points (the vertices of the arrangement), the intersections of two or more lines of A.

These three types of objects link together to form a cell complex covering the plane. Two arrangements are said to be isomorphic or combinatorially equivalent if there is a one-to-one adjacency-preserving correspondence between the objects in their associated cell complexes.

4.6 Exercises

1. Define convex hull. Is it possible to have convex-hull of two dimensional space? Can it exists in $k-$dimensional space $\Re^k$, where $k > 2$? Justify your answer.
2. What are the different algorithms to find out convex-hull in Euclidean space $\Re^k$ for $k = 2$ for set of points $|S| = n$? Write the steps for each algorithm and compute their time complexity.
3. What is meant by *proximity* in a set of points $|S| = n$ in Euclidean space $\Re^k$ for $k = 2$? What may be the applications of computing the proximity?
4. Explain *Voronoi* diagrams and discuss their applications.

References

[1] Thomas H. Corman, et al., "Introduction to Algorithms, 3rd ed." *PHI*, 2009.

[2] Allen B. Tucker, Jr., "The computer Science and Engineering Handbook," *CRC Press*, 1997.