Introduction

Lecture Topics

Questions ?
The Voronoi Diagram
- Problem definition:
  - Input: a set of points (sites) \( P \) in the plane.
  - Output: A planar subdivision \( S \) into cells. One cell per site. A point \( q \) lies in the cell corresponding to a site \( p \in P \) if \( p \) is the nearest site to \( q \).

Nearest Neighbor
- Problem definition:
  - Input: a set of points (sites) \( P \) in the plane and a query point \( q \).
  - Output: The point \( p \in P \) closest to \( q \) among all points in \( P \).

- Rules of the game:
  - One point set, multiple queries

- Applications:
  - Store Locator
  - Cellphones

Point in Polygon
- Problem definition:
  - Input: a polygon \( P \) in the plane and a query point \( p \).
  - Output: true if \( p \in P \), else false.

- Rules of the game:
  - One polygon, multiple queries

Point Location
- Problem definition:
  - Input: A partition \( S \) of the plane into cells and a query point \( p \).
  - Output: The cell \( C \in S \) containing \( p \).

- Rules of the game:
  - One partition, multiple queries

- Applications:
  - Nearest neighbor
  - State locator

Shortest Path
- Problem definition:
  - Input: Obstacle locations and query endpoints \( s \) and \( t \).
  - Output: the shortest path between \( s \) and \( t \) that avoids all obstacles.

- Application: Robotics

Convex Hull
- Problem definition:
  - Input: a set of points \( S \) in the plane.
  - Output: Minimal convex polygon containing \( S \).
Visibility

- Problem definition:
  - Input: a polygon $P$ in the plane and a query point $p$.
  - Output: Polygon $Q \subseteq P$, visible to $p$.

- Rules of the game:
  - One polygon, multiple queries
  - Applications: Security

Range Searching and Counting

- Problem definition:
  - Input: A set of points $P$ in the plane and a query rectangle $R$.
  - Output: (report) The subset $Q \subseteq P$ contained in $R$.
    (count) The size of $Q$.

- Rules of the game:
  - One point set, multiple queries.
  - Application: Urban planning, databases

Ray Tracing

- Applications:
  - Security
  - Ray Tracing

Basic Concepts

Questions?
### Representing Geometric Elements

- Representation of a line segment by four real numbers:
  - Two endpoints \((a, b)\) and \((c, d)\)
  - One endpoint \((a, b)\) with slope \((s)\) and length \((l)\)
  - One endpoint \((a, b)\), vector direction \((v)\), and parameter interval length \((l)\)

- Parametric form
  
  \[ p(t) = p_1 + t(\overrightarrow{p_1p_2}) = (1-t)p_1 + tp_2, \quad t \in [0,1] \]

- Different representations may affect the numeric accuracy of algorithms...

### Complexity (reminder)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Nickname</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f(n) = O(g(n)))</td>
<td>(n \in N \land \forall n \in N \exists C \in C )</td>
<td>(\leq)</td>
</tr>
<tr>
<td>(f(n) = \Omega(g(n)))</td>
<td>(g(n) \in O(f(n)))</td>
<td>(\geq)</td>
</tr>
</tbody>
</table>

### Convex Hull Algorithms

### Convex Hulls – Some Facts

- The convex hull of a set is unique.
- The boundary of the convex hull of a point set is a polygon on a subset of the points.

### Convexity and Convex Hull

- A set \(S\) is convex if any pair of points \(p, q \in S\) satisfy \(pq \subseteq S\).

- The convex hull of a set \(S\) is:
  - The minimal convex set that contains \(S\), i.e., any convex set \(C\) such that \(S \subseteq C\) satisfies \(\text{CH}(S) \subseteq C\).
  - The intersection of all convex sets that contain \(S\).
  - The set of all convex combinations of \(p_i \in S\), i.e., all points of the form:
    
    \[ \sum_{i=1}^{n} \alpha_i p_i, \quad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i = 1 \]

### Orientation

- Area

\[ \text{Area} = \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \end{vmatrix} = \frac{1}{2} \sum_{i=1}^{n} x_i(y_{i+1} - y_{i-1}) - x_{i-1}(y_i - y_{i+1}) \]

- The sign of the area indicates the orientation of the points.
- Positive area \(\equiv\) counterclockwise orientation \(\equiv\) left turn.
- Negative area \(\equiv\) clockwise orientation \(\equiv\) right turn.

- Question: How can this be used to determine whether a given point is “above” or “below” or “on” a given line segment? Is this numerically stable?
Possible Pitfalls

- Degenerate cases – e.g. 3 collinear points. Might harm the correctness of the algorithm. Segments AB, BC and AC will all be included in the convex hull.
- Numerical problems – We might conclude that none of the three segments belongs to the convex hull.

Convex Hull – Naive Algorithm

- Description:
  - For each pair of points construct its connecting segment and supporting line.
  - Find all the segments whose supporting lines divide the plane into two halves, such that one half plane contains all the other points.
  - Construct the convex hull out of these segments.
- Time complexity:
  - All pairs: \( \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2) \)
  - Check all points for each pair: \( O(n) \)
  - Total: \( O(n^2) \)

The Algorithm

- Sort the points in increasing order of x-coord:
  \( p_1, \ldots, p_n \)
  
  ```
  " Note – this is the only part not done in O(n) time "
  Push(S, p_1); Push(S, p_2);
  For i = 3 to n do
    While size(S) > 2 and Orient(p_i, top(S).second).second(S) < 0 /* left turn */
      do Pop(S);
    Push(S, p_i);
  Print(S);
  ``

Convex Hull – Graham’s Scan

- Ideas: Sort the points according to their x-coordinates. First we construct only the upper CH.
- Process the points from the leftmost to rightmost.
- Maintain the upper CH of all points from the leftmost one to the currently processed scanned point.
- Develop the left turn criteria for the last 3 processed points:
  - If we need to turn left when traveling along these points, the middle one is NOT on the upper CH, and we delete it.
  - Note: After deletion, we have new 3 points to consider.

Graham’s Scan – a Variant

- Assume the points are given in increasing x-coord order.
- Time Complexity: \( O(n \log n) \)
- Question: What are the pros and cons of this algorithm relative to the previous?

Graham’s Scan – Time Complexity

- Sorting = \( O(n \log n) \)
- If \( D_i \) is number of points popped on processing \( p_i \),
  \[
  \text{time} = \sum_{i=1}^{n} (D_i + 1) = n \sum_{i=1}^{n} D_i
  \]
- Each point is pushed on the stack only once.
- Once a point is popped – it cannot be popped again.
- Hence
  \[
  \sum_{i=1}^{n} D_i \leq n
  \]
- Question: What is actually \( \sum_{i=1}^{n} D_i \leq n \)?
Divide and Conquer

Convex Hull - Divide and Conquer

- Algorithm:
  - Find a point with a median x coordinate (time: $O(n)$)
  - Compute the convex hull of each half (recursive execution)
  - Combine the two convex hulls by finding common tangents. This can be done in $O(n)$.

- Complexity: $O(n \log n)$

Finding Common Tangents

A tangent line – a line cutting the CH at a single point

Consider a line passing through a vertex $v$ of $H_A$. How can we determine if $v$ is a tangent to $H_B$?

Finding Common Tangents

To find lower tangent:

- Find $a$ - the rightmost point of $H_A$.
- Find $b$ - the leftmost point of $H_B$.
- While $ab$ is not a lower tangent for $H_A$ and $H_B$, do:
  - If $ab$ is not a lower tangent to $H_B$, do $a = a + 1$ and move one point clockwise.
  - If $ab$ is not a lower tangent to $H_A$, do $b = b + 1$ and move one point counterclockwise.

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents
Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents
Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Finding Common Tangents

Output-Sensitive Convex Hull
Gift Wrapping

- Algorithm:
  - Find a point \( p_1 \) on the convex hull (e.g. the lowest point).
  - Rotate counterclockwise a line through \( p_1 \) until it touches one of the other points (start from a horizontal orientation).
  - **Question**: How is this done?

- Repeat the last step for the new point.
- Stop when \( p_1 \) is reached again.

- **Time Complexity**: \( O(nh) \), where \( n \) is the input size and \( h \) is the output (hull) size.

- Best alg in 2D: \( O(n \log h) \)
When designing a geometric algorithm, we first make some simplifying assumptions, e.g.:  
- General Position  
  - No 3 collinear points.  
  - No two points with the same x coordinate.  
  - etc.  
Later, we consider the general case:  
- How should the algorithm react to degenerate cases?  
- Will the correctness be preserved?  
- Will the runtime remain the same?

A reduction from sorting to convex hull is:  
- Given n real values $x_i$, generate n 2D points on the graph of a convex function, e.g. $(x_i, x_i^2)$.  
- Compute the (ordered) convex hull of the points.  
- The order of the convex hull points is the numerical order of the $x_i$.  
- So CH=$\Omega(n \log n)$