Polygon Partitioning

Monotone Partitioning

Monotone polygons

Definition

- A polygonal chain $C$ is strictly monotone with respect to $L'$ if every line $L$ orthogonal to $L'$ meets $C$ in at most one point.
- A polygonal chain $C$ is monotone if $L$ has at most one connected component
  - Either empty, a single point, or a single line segment
- A polygon $P$ is said to be monotone with respect to a line $L$ if $\partial P$ can be split into two polygonal chains $A$ and $B$ such that each chain in monotone with respect to $L$.

Monotone Partitioning

- Define monotonicity
- Triangulate monotone polygons in linear time
- Partition a polygon into monotone pieces

Monotone polygons

- The two chains should share a vertex at either end.
- Figure 2.1
  - A polygon monotone with respect to the vertical line
  - Two monotone chains
    - $A = (v_0, ..., v_{15})$ and $B = (v_{15}, ..., v_{24}, v_0)$
    - Neither chain is strictly monotone (two horizontal edges $v_5v_6$ and $v_{21}v_{22}$)
Properties of Monotone Polygons

- The vertices in each chain of a monotone polygon are sorted with respect to the line of monotonicity.
- Let y axis be the fixed line of monotonicity.
- Then vertices can be sorted by y coordinate in linear time.
- Find a highest vertex, a lowest vertex and partition the boundary into two chains. (linear time)

Monotone Partitioning

- Define an interior cusp of a polygon as a reflex vertex $v$ whose adjacent vertices $v_-$ and $v_+$ are either both at or above, or both at or below, $v$. (See Figure 2.2)
- Interior cusp can't have both adjacent vertices with the same y coordinate as $v$.
- Thus, $d$ in Figure 2.2(b) is not an interior cusp.

Lemma 2.1.1

- If a polygon $P$ has no interior cups, then it is monotone.
- Lack of interior cusp implies strict monotonicity.
Triangulating a Monotone Polygon

- The shapes of monotone polygons are so special that they are easy to triangulate
- Linear time triangulation
- Hint of algorithm
  - Sort the vertices from top to bottom (in linear time)
  - Cut off triangles from the top in a “greedy” fashion

Algorithm
- For each vertex \( v \), connect \( v \) to all the vertices above it and visible via a diagonal
- Remove the top portion of the polygon thereby triangulated
- Continue with the next vertex below \( v \).
- One can show that at any iteration, \( v \in A \) is being connected to a chain of reflex vertices above it in the other chain \( B \).

For example, \( v_{16} \) is connected to \((v_{14}, v_{13}, v_{12})\) in the first iteration.
As a consequence, no visibility check is required
The algorithm can be implemented with a single stack holding the reflex chain above.
Between the linear sorting and this simple data structure, \( O(n) \) is achieved.

Polygon Partitioning

Trapezoidalization
A horizontal trapezoidalization of a polygon is obtained by drawing a horizontal line through every vertex of the polygon. Pass through each vertex \( v \) the maximal horizontal segment \( s \) such that \( s \subset P \) and \( s \cap \partial P = v \). See Figure 2.3. We only consider polygons whose vertices have unique \( y \) coordinates.

A trapezoid is a quadrilateral with two parallel edges. Call the vertices through which the horizontal lines are drawn supporting vertices. Every trapezoid has exactly two supporting vertices. The connection to monotone polygons:

- If a supporting vertex is on the interior of an upper or lower trapezoid edge, then it is an interior cusp.

Monotone partition:

- Connect every interior vertex \( v \) to the opposing supporting vertex of the trapezoid \( v \) supports.
- Then, these diagonals partition \( P \) into monotonic pieces.
- Recall Lemma 2.1.1.
- For example, diagonal \( v_6v_4, v_{15}v_{12} \), and so on. (Figure 2.3)
**Plane Sweep**

Useful in many geometric algorithms
Main idea is to “sweep” a line over the plane maintaining some type of data structure along the line
Sweep a horizontal line \( L \) over the polygon, stopping at each vertex
Sorting the vertices by \( y \) coordinate \( O(n \log n) \) time

**Plane Sweep**

Fine the edge immediately to the left and immediately to the right of \( v \) along \( L \)
- A sorted list \( L \) of polygon edges pierced by \( L \) is maintained at all times
- For example, for the sweep line in the position shown in Figure 2.4 \( L = \{e_{19}, e_{18}, e_{17}, e_{6}, e_{8}, e_{10}\} \)
- How to determine that \( v \) lies between \( e_{17} \) and \( e_{6} \) in Figure 2.4?

**Plane Sweep**

Assume that \( e_i \) is a pointer to an edge and the vertical coordinate of \( v \) is \( y \).
Suppose we know the endpoints of \( e_i \)
Then, we can compute the \( x \) coordinate of the intersection between \( L \) and \( e_i \)
We can determine \( v \)'s position by computing \( x \) coordinates of each intersection
Time proportional to the length of \( L \) \( (O(n)) \) by a naive search from left to right
With an efficient data structure, a height-balanced tree, the search require \( O(\log n) \) time

**Plane Sweep**

![Figure 2.4 Plane sweep. Labels index edges.](image-url)
Plane Sweep

Updates at each event
There are three types of event (Figure 2.5)
Let \( v \) fall between edges \( a \) and \( b \) and \( v \) be shared by edges \( c \) and \( d \)

- \( c \) is above \( L \) and \( d \) is below. Then delete \( c \) from \( L \) and insert \( d \):
  - \( (..., a, c, b, ... ) \Rightarrow (..., a, d, b, ... ) \)
- Both \( c \) and \( d \) are above \( L \). Then delete both \( c \) and \( d \) from \( L \):
  - \( (..., a, c, d, b, ... ) \Rightarrow (..., a, b, ... ) \)
- Both \( c \) and \( d \) are below \( L \). Then insert both \( c \) and \( d \) into \( L \):
  - \( (..., a, b, ... ) \Rightarrow (..., a, c, d, b, ... ) \)

Plane Sweep

In Figure 2.4, the list \( L \) of edges pierced by \( L \) is initially empty, when \( L \) is above the polygon
Then follows this sequence as it passes each event vertex

\[
\begin{align*}
(e_{12}, e_{11}) \\
(e_{15}, e_{14}, e_{12}, e_{11}) \\
(e_{15}, e_{14}, e_{12}, e_{6}, e_{7}, e_{11}) \\
(e_{15}, e_{14}, e_{13}, e_{6}, e_{7}, e_{11}) \\
(e_{16}, e_{14}, e_{13}, e_{6}, e_{7}, e_{10}) \\
(e_{15}, e_{6}, e_{7}, e_{10}) \\
(e_{15}, e_{6}, e_{8}, e_{10}) \\
(e_{19}, e_{18}, e_{16}, e_{6}, e_{8}, e_{10}) \\
(e_{19}, e_{18}, e_{17}, e_{6}, e_{8}, e_{10})
\end{align*}
\]

Triangulation in \( O(n \log n) \)

Algorithm: Polygon Triangulation: Monotone Partition
Sort vertices by \( y \) coordinate.
Perform plane sweep to construct trapezoidization.
Partition into monotone polygons by connecting from interior cusps.
Triangulate each monotone polygon in linear time.

Algorithm 2.1 \( O(n \log n) \) polygon triangulation.
Polygon Partitioning

Partition into Monotone Mountains

Monotone Mountains

- Lemma 2.3.1
  - Every strictly convex vertex of a monotone mountain $M$, with the possible exception of the base endpoints, is an ear tip
  - Proof
    - Let $a$, $b$, $c$ be three consecutive vertices of $M$, with $b$ a strictly convex vertex not an endpoint of the base $B$.
    - The direction of monotonicity is horizontal
    - Aim to prove that $ac$ is a diagonal by contradiction

Monotone Mountains

- Lemma 2.3.1
  - Proof (con't)
    - Assume that $ac$ is not a diagonal
    - By Lemma 1.5.1, either it is exterior in the neighborhood of an endpoint or it intersects $\partial M$.
    - Case 1. Suppose that $ac$ is locally exterior
      - If $a$ or $c$ is not an endpoint of $B$, contradiction by the assumption that $b$ is convex.
      - If $a$ or $c$ is the right endpoint of $B$, the same contradiction.
      - Or $ac$ is below $B$, it is impossible because $c$ lies above $B$
    - Thus, $ac$ is locally interior

Monotone Mountains

A monotone mountain is a monotone polygon with one of its two monotone chains a single segment, the base.

See Figure 2.6

Note that both end points of the base must be convex.
Monotone Mountains

- **Lemma 2.3.1**
  - Proof (con’t)
    - **Case 2.** Assume that $ac$ intersects $\partial M$
    - This requires a reflex vertex $x$ to be interior to $\triangle abc$ (cf. Figure 1.12)
    - Because $x$ is interior, it cannot lie on either chain $C=(a,b,c)$ or $B$.
    - Thus, a vertical line $L$ through $x$ meets $\partial M$ in at least three points (contradiction)

Triangulating a Monotone Mountain

Linear time algorithm
The base is identified in linear time
- The base endpoints are extreme along the direction of monotonicity
- Leftmost and rightmost vertices
The “next” convex vertex is found without a search in constant time
- Similar with updating the ear tip status in Section 1.4
- Instead, update the convexity status
- Use stored internal angles of vertices
- By subtracting from $a$ and $c$’s angles appropriately upon removal of $\triangle abc$
- Require linking the convex vertices into a list and updating the list with each ear clip

Triangulating a Monotone Mountain

**Algorithm:** TRIANGULATION OF MONOTONE MOUNTAIN
Identify the base edge.
Initialize internal angles at each nonbase vertex.
Link nonbase strictly convex vertices into a list.
while list nonempty do
  For convex vertex $b$, remove $\triangle abc$.
  Output diagonal $ac$.
  Update angles and list.

Algorithm 2.2 Linear-time triangulation of a monotone mountain.

Trapezoidalization to monotone mountains

Build a monotone mountain from trapezoids abutting on a particular base edge, for example $v_{11}v_{12}$ (Figure 2.7)
$T(i, j)$ represents the trapezoid with support vertices $v_i$ and $v_j$, below and above
Lemma 2.3.2

- In a trapezoidalization of a polygon $P$, connecting every pair of trapezoid–supporting vertices that do not lie on the same (left/right) side of their trapezoid partitions $P$ into monotone mountains.

- Proof
  - Lemma 2.1.1 guarantees that the pieces of the partition are monotone.
  - Thus only need to prove that each piece has one chain that is a single segment.

Proof (con’t)

- Suppose to the contrary that both monotone chains $A$ and $B$ of one piece $Q$ of the partition each contain at least two edges.
- Let $z$ be the topmost vertex of $Q$, adjacent on $Q = A \cup B$ to vertices $a$ and $b$, with $b$ below $a$ (Figure 2.8)
- In order for $B$ to contain more than just the edge $zb$, $b$ cannot be the endpoint
- But the upper supporting vertex $c$ of $T(b, c)$ cannot lie on $zb$, for $c$ must lie at or below $a$
- Thus $c$ is not on the same side of $T(b, c)$ as $b$, and the diagonal $cb$ is included in the partition
- This contradicts the assumption that $B$ extends below $b$. 

Figure 2.7 A partition into monotone mountains.
Trapezoidalization to monotone mountains

Polygon Partitioning

Linear-Time Triangulation

Table 2.1, History of triangulation algorithms.

<table>
<thead>
<tr>
<th>Year</th>
<th>Complexity</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1911</td>
<td>$O(n^2)$</td>
<td>Lennes (1911)</td>
</tr>
<tr>
<td>1978</td>
<td>$O(n \log n)$</td>
<td>Garey et al. (1978)</td>
</tr>
<tr>
<td>1983</td>
<td>$O(n \log r)$, $r$ reflex</td>
<td>Hertel &amp; Mehlhorn (1983)</td>
</tr>
<tr>
<td>1984</td>
<td>$O(n \log s)$, $s$ sinosity</td>
<td>Chazelle &amp; Incerpi (1984)</td>
</tr>
<tr>
<td>1988</td>
<td>$O(n + n c)$, $c_0$ int. triangs.</td>
<td>Toussaint (1990)</td>
</tr>
<tr>
<td>1986</td>
<td>$O(n \log \log n)$</td>
<td>Tarjan &amp; Van Wyk (1988)</td>
</tr>
<tr>
<td>1989</td>
<td>$O(n \log^* n)$, randomized</td>
<td>Clarkson, Tarjan &amp; Van Wyk (1989)</td>
</tr>
<tr>
<td>1990</td>
<td>$O(n \log^* n)$, bnded. ints.</td>
<td>Kirkpatrick, Klawe &amp; Tarjan (1990)</td>
</tr>
<tr>
<td>1990</td>
<td>$O(n)$</td>
<td>Chazelle (1991)</td>
</tr>
<tr>
<td>1991</td>
<td>$O(n \log^* n)$, randomized</td>
<td>Seidel (1991)</td>
</tr>
</tbody>
</table>

Chazelle’s algorithm ($O(n)$)
- Main structure: visibility map
- Generalization of a trapezoidalization to drawing horizontal chords toward both sides of each vertex in a polygonal chain
- Mimics merge sort, a common technique for sorting by divide-and-conquer
- The polygon of $n$ vertices is partitioned into chains with $n/2$ vertices, and into chains of $n/4$ vertices, and so on
- This leads to an $O(n \log n)$ time complexity
### Linear-Time Triangulation

Chazelle’s algorithm ($O(n)$) (con’t)
- Improves on this by dividing the process into two phases
- First, only coarse approximations of the visibility maps
- Coarse enough so that the merging can be accomplished in linear time
- Second, refines the coarse map into a complete visibility map, again in linear
- A triangulation is then produced from the trapezoidalization defined by the visibility map as before

### Randomized Triangulation

Seidel’s algorithm ($O(n \log^* n)$)
- Follows the trapezoidalization $\rightarrow$ monotone mountains $\rightarrow$ triangulation path
- His improvement is in building the trapezoidalization quickly
- Uses a “query structure” $Q$, a data structure that permits location of a point in its containing trapezoid in time proportional to the depth of the structure
- The depth of the structure could be $\Omega(n)$ for $n$ segments, but
- By adding the segments in random

### Randomized Triangulation

Randomized algorithms
- Can be expected to work well on all inputs
- Developed into a key technique for creating algorithms that are both efficient and simple

Seidel’s algorithm
- The visibility map by inserting the segments in random order in $O(n \log n)$ time and $O(n)$ space
- Results in another $O(n \log n)$ algorithm

### Randomized Triangulation

Seidel’s algorithm (con’t)
- The segments form the edges of a simple polygon
- This can be exploited by running the algorithm in $\log^* n$ phases
- In phase $i$, a subset of the segments is added in random order, producing $Q_i$
- The entire polygon is traced through $Q_i$, locating each vertex in a trapezoid of the current visibility map
Randomized Triangulation

Seidel's algorithm (con't)
- In phase $i + 1$, more segments are added
- The knowledge of where they were in $Q_i$ helps locate their endpoints more quickly
- The process is repeated until the entire visibility map is constructed

Analysis of the expected time
- Over all possible $n!$ segment insertion orders
- It shows the expected time is $O(n \log^* n)$

Polygon Partitioning

Convex Partitioning

Two goals
- Partition a polygon into as few convex pieces as possible
- Do so as fast as possible

Compromise on the number of pieces
Find a quick algorithm whose output size is bounded with respect to the optimum

Two types of partition may be distinguished
- By diagonals
- By segments

Optimum Partition

- Lemma 2.5.1 (Chazelle)
  - Let $\Phi$ be the fewest number of convex pieces into which a polygon may be partitioned. For a polygon of $r$ reflex vertices, $\lceil r/2 \rceil + 1 \leq \Phi \leq r + 1$
  - Proof
    - Drawing a segment that bisects each reflex angle results in a convex partition
    - The number of pieces is $r + 1$ (Figure 2.10)
    - At most two reflex angles can be resolved by a single partition segment (Figure 2.11)
    - This results in $\lceil r/2 \rceil + 1$ convex pieces
Hertel and Mehlhorn Algorithm

A very clean algorithm that partitions with diagonals quickly
- has bounded “badness” in terms of the number of convex pieces

A diagonal \( d \) is essential for vertex \( v \) if removal of \( d \) makes \( v \) non-convex

The algorithm
- start with a triangulation of \( P \)
- remove an inessential diagonal
- repeat

**Lemma 2.5.2**
- There can be at most two diagonals essential for any reflex vertex
- Proof
  - Let \( v \) be a reflex vertex and \( v_+ \) and \( v_- \) its adjacent vertices.
  - At most one essential diagonal in the halfplane \( H_+ \) to the left of \( vv_+ \)
  - If there were two, the one closest to \( vv_+ \) could be removed (Figure 2.12)
  - Similarly, there can be at most one essential diagonal in the halfplane \( H_- \) to the left of \( vv_- \)
  - Together these cover the interior angle at \( v \), and so there are at most two diagonals essential for \( v \)
Hertel and Mehlhorn Algorithm

Lemma 2.5.3
- The Hertel-Mehlhorn algorithm is never worse than four-times optimal in the number of convex pieces.
- Proof
  - By Lemma 2.5.2, each reflex vertex can be "responsible for" at most two essential diagonals.
  - The number of essential diagonals can be no more than $2r$.
  - Thus, the number of convex pieces $M$ produced by the algorithm satisfies $2r + 1 \leq M$.
  - Since $\Phi \geq r^2/2^3 + 1$ by Lemma 2.5.1,
    - $4\Phi \geq 2r + 4 > 2r + 1 \geq M$.

Optimal Convex Partitions

Finding a convex partition optimal in the number of pieces is much more time consuming than finding a suboptimal one.
- Greene’s algorithm runs in $O(r^2n^2) = O(n^4)$ time.
- Keil’s algorithm improved it to $O(r^2n\log n) = O(n^3\log n)$ time.
- Both employ dynamic programming.

The problem is even more difficult if the partition may be formed with arbitrary segments.
- Figure 2.13.
- Chazelle solve this problem in his thesis with an intricate $O(n + r^2) = O(n^3)$ algorithm.

Optimal Convex Partitions

FIGURE 2.13 An optimal convex partition. Segment $s$ does not touch $\partial P$. 
Approximate Convex Decomposition (ACD)

- **ACD**
  - All sub-models will have tolerable concavity
  - Convex decomposition is useful but can be costly to construct
  - may result in unmanageable number of components

- **Benefits of ACD**
  1. Number of sub-models is significantly less
  2. Reveal structural features

- More results

#### Benefits of ACD

- **Input model**
- **Measure concavity**
- **Shaded using its concavity**
- **Decompose at areas with high concavity**

- More results

Conclusion

- Polygon decomposition
  - decompose to monotonic polygon/mountain
  - trapezoidal decomposition
  - convex decomposition

- Homework assignment
  - Ex: 2.2.3-2, 2.2.3-4, 2.5.4-7