CG Lecture 3

Polygon decomposition

1. Polygon triangulation
   - Triangulation theory
   - Monotone polygon triangulation
2. Polygon decomposition into monotone pieces
3. Trapezoidal decomposition
4. Convex decomposition
5. Other results

Motivation: Art gallery problem

**Definition:** two points $q$ and $r$ in a simple polygon $P$ can see each other if the open segment $qr$ lies entirely within $P$.

A point $p$ guards a region $R \subseteq P$ if $p$ sees all $q \in R$

**Problem:** Given a polygon $P$, what is the minimum number of guards required to guard $P$, and what are their locations?
Simple observations

• **Convex polygon**: all points are visible from all other points $\rightarrow$ only one guard in any location is necessary!

• **Star-shaped** polygon: all points are visible from any point in the kernel $\rightarrow$ only one guard located in its kernel is necessary.

Art gallery problem: upper bound

• **Theorem**: Every simple planar polygon with $n$ vertices has a triangulation of size $n-2$ *(proof later)*.

• $n-2$ guards suffice for an $n$-gon:
  - Subdivide the polygon into $n-2$ triangles (triangulation).
  - Place one guard in each triangle.
Art gallery problem: lower bound

- There exists a polygon with \( n \) vertices, for which \( \lfloor n/3 \rfloor \) guards are necessary.

- Therefore, \( \lfloor n/3 \rfloor \) guards are needed in the worst case.

Can we improve the upper bound?
Yes! In fact, at most \( \lfloor n/3 \rfloor \) guards are necessary.

Simple polygon triangulation

**Input:** a polygon \( P \) described by an ordered sequence of vertices \(<v_0, \ldots, v_{n-1}>\).

**Output:** a partition of \( P \) into \( n-2 \) non-overlapping triangles and the adjacencies between them.
Simple polygon triangulation: observations

• The triangulation is not unique. One of them suffices.
• The triangulation is always possible.
• No new vertices are required.
• The triangulation adds new edges, called *diagonals*, between existing vertices.

*Not all diagonals are valid!*
Triangulation theory

- A vertex is **convex** if its interior angle $< \pi$, otherwise it is **concave**.
- A **diagonal** is a new edge between two polygon vertices that is entirely inside the polygon.
- **Lemma 1**: every polygon has a convex vertex.
  
  **Proof**: the highest vertex (the one with the largest $y$ coordinate) is convex.
- **Lemma 2**: Every polygon with $n>3$ vertices has a diagonal.

Diagonals in polygons

**Proof**: let $v$ be a convex vertex and let $a$ and $b$ its adjacent vertices. Since $P$ is a simple polygon and $n>3$, there is no edge between $a$ and $b$.

Consider the following two cases:

1. the new edge $ab$ is a diagonal
2. Otherwise, there exists a vertex $x$ which is the closest to $v$ with respect to a line $L$ parallel to $ab$ which is a diagonal.
**Triangulation theory**

**Theorem:** Every simple polygon with \( n \) vertices has a triangulation with \( n-3 \) diagonals and \( n-2 \) triangles.

**Proof:** By induction on \( n \):
- **Basis:** A triangle \((n=3)\) has a triangulation (itself) with no diagonals and one triangle.
- **Induction on \( n \):**
  For an \( n \)-vertex polygon, construct a diagonal dividing the polygon into two polygons \( P_1 \) and \( P_2 \) with \( n_1 \) and \( n_2 \) vertices such that \( n_1 + n_2 - 2 = n \).
  Diagonals: \((n_1-3)+(n_2-3)+1 = (n_1+n_2-2)-3 = n-3\)
  Triangles: \((n_1-2)+(n_2-2) = (n_1+n_2-2)-2 = n-2\)

---

**Triangulation dual**

**Definition:** The triangulation dual \( T \) of a triangulation of a simple polygon \( P \) is a graph whose nodes are triangles and whose edges are adjacencies between triangles sharing an edge.

**Property:** the triangulation dual is a tree whose node degree is \( \leq 3 \).

**Proof:**
- Degree \( \leq 3 \) by construction: no triangle has more than three neighbors.
- No cycles: by contradiction. If it has a cycle, it is a polygon with a hole (not a simple polygon).
- If fact, \( T \) is a binary tree with root degree one or two!
Simple triangulation algorithm (1)

Idea: Reduce the polygon by clipping a triangle at each iteration. The clipped triangle will be formed by three consecutive vertices \((v_i, v_{i+1}, v_{i+2})\). The diagonal is \((v_i, v_{i+2})\).

Test for validity:
1. The diagonal does not intersect other polygon edges.
2. The diagonal must be inside the polygon (test that diagonal is inside normal cone).

```
proc triangulate(P)
    if |P| ≤ 3 output(P) and return;
    i ← 0;
    while diagonal(v_i, v_{i+2}) is not legal i++;
        output(v_i, v_{i+1}, v_{i+2});
        remove v_{i+1} from P;
        triangulate(P);
```

Complexity: \(n\) times \(n\) diagonal tests that take each \(O(n) \rightarrow O(n^3)\).

Sources of inefficiencies:
- repeated diagonal tests
- diagonals are not sorted or ordered

\(\rightarrow\) Precompute diagonals in \(O(n^2)\).
Triangulation algorithms

- What is the lower bound?
  - $O(n^2)$ by precomputing diagonals
  - Improve to $O(n \log n)$ by ordering them (see later).
  - Is less than $O(n \log n)$ possible?

<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 and Mehlhorn (1983)</td>
</tr>
<tr>
<td>1984</td>
<td>$O(n \log s)$, $s$ sinuosity</td>
<td>Chazelle and Incerpi (1984)</td>
</tr>
<tr>
<td>1988</td>
<td>$O(n + n \tau_{10}, \tau_0$ int. triangs.</td>
<td>Toussaint (1990)</td>
</tr>
<tr>
<td>1988</td>
<td>$O(n \log \log n)$</td>
<td>Tarjan and VanWyk (1988)</td>
</tr>
<tr>
<td>1989</td>
<td>$O(n \log^* n)$, randomized</td>
<td>Clarkson, Tarjan, and VanWyk (1989)</td>
</tr>
<tr>
<td>1990</td>
<td>$O(n \log^* n)$, bused. ints.</td>
<td>Kirkpatrick, Klawe, and Tarjan (1990)</td>
</tr>
</tbody>
</table>

O($n \log n$)-time triangulation algorithm

1) Partition the polygon into $y$-monotone pieces ("חתיכות מונוטוניות").

2) Triangulate each $y$-monotone piece separately.
Monotone polygons: definition

- A simple polygon is called monotone with respect to a line $L$ if for any line $L'$ perpendicular to $L$ the intersection of the polygon with $L'$ is connected.
- A polygon is called monotone if there exists any such line $\ell$.
- A polygon that is monotone with respect to the $x/y$-axis is called $x/y$-monotone.

**Question:** How can we check in $O(n)$ time whether a polygon is $y$-monotone?

---

Property of monotone polygons

**Definition:** a vertex $v$ is an interior cusp iff it is a concave vertex whose adjacent vertices are both at or above (at or below) line $L$.

**Theorem:** If a polygon $P$ has no interior cusps with respect to a line $L$, then it is monotone with respect to $L$.

**Proof sketch:** partition $P$ into two chains connecting the top and bottom vertices. Assume one of them is not monotone with respect to $L$. Then $P$ must contain an interior cusp above or below.
Triangulating a y-monotone polygon

• Sweep the polygon from top to bottom.
• Greedily triangulate anything possible above the sweep line, and then forget about this region.
  • When processing a vertex $v$, the unhandled region above it always has a simple structure:
    Two y-monotone (left and right) chains, each containing at least one edge. If a chain consists of two or more edges, it is reflex, and the other chain consists of a single edge whose bottom endpoint has not been handled yet.
  • Each diagonal is added in $O(1)$ time.

Triangulating a Y-monotone polygon

• Continue sweeping while one chain contains only one edge, while the other edge is concave.
• When a “convex edge” appears in the concave chain (or a second edge appears in the other one), triangulate as much as possible using a “fan”.
• Time complexity: $O(k)$, where $k$ is the complexity of the polygon.
Stack operations on chains

Monotone polygon triangulation

**Algorithm** TRIANGULATEMONOTONEPOLYGON($P$)

*Input.* A strictly $y$-monotone polygon $P$ stored in a doubly-connected edge list $D$.

*Output.* A triangulation of $P$ stored in the doubly-connected edge list $D$.

1. Merge the vertices on the left chain and the vertices on the right chain of $P$ into one sequence, sorted on decreasing $y$-coordinate. If two vertices have the same $y$-coordinate, then the leftmost one comes first. Let $u_1, \ldots, u_n$ denote the sorted sequence.
2. Push $u_1$ and $u_2$ onto the stack $S$.
3. for $j \leftarrow 3$ to $n - 1$
4. do if $u_j$ and the vertex on top of $S$ are on different chains
5. then Pop all vertices from $S$.
6. Insert into $D$ a diagonal from $u_j$ to each popped vertex, except the last one.
7. Push $u_{j-1}$ and $u_j$ onto $S$.
8. else Pop one vertex from $S$.
9. Pop the other vertices from $S$ as long as the diagonals from $u_j$ to them are inside $P$. Insert these diagonals into $D$. Push the last vertex that has been popped back onto $S$.
11. Add diagonals from $u_n$ to all stack vertices except the first and the last one.
Y-monotone polygons

Polygon vertices classification:

- A **start** (resp., **end**) vertex is a vertex whose interior angle is less than $\pi$ and its two neighboring vertices both lie below (resp., above) it.
- A **split** (resp., **merge**) vertex is a vertex whose interior angle is greater than $\pi$ and its two neighboring vertices both lie below (resp., above) it.
- All other vertices are **regular**.

Vertex classification

- □ = start vertex
- ■ = end vertex
- ● = regular vertex
- ▲ = split vertex
- ▼ = merge vertex
Y-monotone polygons: properties

**Theorem:** A polygon without *split* and *merge* vertices is $y$-monotone.

**Proof:** Since there are only start/end/regular vertices, the polygon must consist of two $y$-monotone chains. Alternatively, do a case analysis.

- To partition a polygon to monotone pieces, eliminate split (merge) vertices by adding diagonals upward (downward) from the vertex. Naturally, the diagonals must not intersect!

---

Split and merge vertices

![Split and merge vertices](image)
Monotone partitioning

- Classify all vertices.
- Sweep the polygon from top to bottom.
- Maintain the edges intersected by the sweep line \( L \) sorted by \( x \) coordinate.
- Maintain vertex events in an event queue \( Q \) sorted by \( y \) coordinate.
- Eliminate split/merge vertices by connecting them to other vertices.
- For each edge \( e \), define \( \text{helper}(e) \) as the lowest vertex (seen so far) above the sweep line visible to the right of the edge.
- \( \text{helper}(e) \) is initialized by the upper endpoint of \( e \).

Monotone partitioning (cont.)

- A split vertex may be connected to the helper vertex of the edge immediately to its left.
- However, a merge vertex should be connected to a vertex which has not been processed yet!
- Clever idea: Every merge vertex is the helper of some edge, and will be handled when this edge "terminates".
Monotone partitioning algorithm

**Input:** A counterclockwise ordered list of vertices. The edge $e_i$ immediately follows the vertex $v_i$.

Construct $Q$ on the vertices of $P$ using $y$-coordinates. (when two or more vertices have the same $y$-coordinates, the vertex with the smaller $x$-coordinate has priority.)

- Initialize $L$ to be empty.
- While $Q$ is not empty:
  - Pop vertex $v$;
  - Handle $v$.

Note: No new events are generated during execution.
No split/merge vertex remains unhandled.

---

Monotone partitioning

Handling a *start* vertex $v_i$:

- Add $e_i$ to $L$
- $helper(e_i) \leftarrow v_i$
Monotone partitioning

Handling an end vertex $v_i$:
• If $\text{helper}(e_{i-1})$ is a merge vertex, then connect $v_i$ to $\text{helper}(e_{i-1})$
• Remove $e_{i-1}$ from $L$

Monotone partitioning

Handling a split vertex $v_i$:
• Find in $L$ the edge $e_j$ directly to the left of $v_i$
• Connect $v_i$ to $\text{helper}(e_j)$
• $\text{helper}(e_j) \leftarrow v_i$
• Insert $e_i$ into $L$
• $\text{helper}(e_i) \leftarrow v_i$
Monotone partitioning

Handling a merge vertex $v_i$:
- If $\text{helper}(e_{i-1})$ is a merge vertex, then connect $v_i$ to $\text{helper}(e_{i-1})$
- Remove $e_{i-1}$ from $L$
- Find in $L$ the edge $e_j$ directly to the left of $v_i$
- If $\text{helper}(e_j)$ is a merge vertex, then connect $v_i$ to $\text{helper}(e_j)$
- $\text{helper}(e_j) \leftarrow v_i$

Monotone partitioning

Handling a regular vertex $v_i$:
- If the polygon’s interior lies to the left of $v_i$ then:
  - Find in $L$ the edge $e_j$ directly to the left of $v_i$
  - If $\text{helper}(e_j)$ is a merge vertex, then connect $v_i$ to $\text{helper}(e_j)$
  - $\text{helper}(e_j) \leftarrow v_i$
  - Else:
    - If $\text{helper}(e_{i-1})$ is a merge vertex, then connect $v_i$ to $\text{helper}(e_{i-1})$
    - Remove $e_{i-1}$ from $L$
    - Insert $e_i$ into $L$
    - $\text{helper}(e_i) \leftarrow v_i$
Monotone polygon algorithm

**Algorithm** MAKEMONOTONE(\(P\))

*Input.* A simple polygon \(P\) stored in a doubly-connected edge list \(D\).

*Output.* A partitioning of \(P\) into monotone subpolygons, stored in \(D\).

1. Construct a priority queue \(Q\) on the vertices of \(P\), using their \(y\)-coordinates as priority. If two points have the same \(y\)-coordinate, the one with smaller \(x\)-coordinate has higher priority.
2. Initialize an empty binary search tree \(T\).
3. **while** \(Q\) is not empty
4. **do** Remove the vertex \(v_i\) with the highest priority from \(Q\).
5. **end do**
6. Call the appropriate procedure to handle the vertex, depending on its type.

Event handling (1)

**HANDLESTARTVERTEX**\( (v_i) \)

1. Insert \(e_i\) in \(T\) and set helper\( (e_i) \) to \(v_i\).

**HANDLEENDVERTEX**\( (v_i) \)

1. **if** helper\( (e_{i-1}) \) is a merge vertex
2. **then** Insert the diagonal connecting \(v_i\) to helper\( (e_{i-1}) \) in \(D\).
3. **end if**
4. Delete \(e_{i-1}\) from \(T\).

**HANDLESPLITVERTEX**\( (v_i) \)

1. Search in \(T\) to find the edge \(e_j\) directly left of \(v_i\).
2. Insert the diagonal connecting \(v_i\) to helper\( (e_j) \) in \(D\).
3. helper\( (e_j) \) ← \(v_i\)
4. Insert \(e_i\) in \(T\) and set helper\( (e_i) \) to \(v_i\).
Event handling (2)

**HANDLEMERGEPATTERN**(*v*)
1. if helper(*e*_i−1) is a merge vertex
2. then insert the diagonal connecting *v* to helper(*e*_i−1) in *D*.
3. Delete *e*_i−1 from *T*.
4. Search in *T* to find the edge *e* directly left of *v*.
5. if helper(*e*_j) is a merge vertex
6. then insert the diagonal connecting *v* to helper(*e*_j) in *D*.
7. helper(*e*_j) ← *v*.

**HANDLEREGULARVERTEX**(*v*)
1. if the interior of *P* lies to the right of *v*
2. then if helper(*e*_i−1) is a merge vertex
3. then insert the diagonal connecting *v* to helper(*e*_i−1) in *D*.
4. Delete *e*_i−1 from *T*.
5. Insert *e* in *T* and set helper(*e*_i) to *v*.
6. else Search in *T* to find the edge *e* directly left of *v*.
7. if helper(*e*_j) is a merge vertex
8. then insert the diagonal connecting *v* to helper(*e*_j) in *D*.
9. helper(*e*_j) ← *v*.

Example

- □ = start vertex
- ■ = end vertex
- ● = regular vertex
- ▲ = split vertex
- ▼ = merge vertex
Time complexity of polygon triangulation

- Partitioning the polygon into monotone pieces: \( O(n \log n) \)
- Triangulating all the monotone pieces: \( O(n) \)
- Total: \( O(n \log n) \)

Trapezoidal decomposition

- Decompose a polygon into trapezoids with two edges perpendicular to a given line.
- Trapezoids are trivially triangulated
- BUT – we need new intermediate vertices.
- Useful for point search and other tasks.
- Obtained directly from a sweep line algorithm: at each vertex, compute the new supporting vertices left and right. Close trapezoids according to neighborhood relations.
Convex partitioning

**Problem:** partition a polygon $P$ into a small (fewest) number of convex pieces.

**Possibilities:**
- Use only vertices of $P$
- Use only vertices on edges of $P$
- Use new internal vertices (Steiner points)

**Theorem:** The smallest number of convex pieces $\lambda$ that is needed to partition $P$ is:
$$\left\lfloor \frac{r}{2} \right\rfloor + 1 \leq \lambda \leq r + 1$$

where $r$ is the number of concave vertices.

---

Hertl and Melhorn algorithm

**Idea:** triangulate polygon, then remove “non-essential” diagonals. A diagonal is non-essential if its removal creates non-convexities.

**Note:** only concave vertices have essential diagonals!

**Complexity:** $O(n \log n)$ for triangulation.

Is this strategy optimal? NO!

**Theorem:** the number of pieces is never worse than four times the optimal.
Other results

• Optimal $O(n)$ triangulation algorithm [Chazelle 1991]
• Finding the minimum number of guards for a simple polygon is NP-hard [Aggarwal, 1984].
• 3D version: find a tetrahedrization of a simple polyhedra.
  • Not always possible without introducing new vertices!
  • The decision problem (are new vertices necessary?) Is NP-complete.
• Algorithm runs in $O(nr + r^2 \log r)$ time [Chazelle and Palios, 1990].