Planar Graphs, Polygons and Triangulations

Lecture 3, CS 4235
29 January 2004

Antoine Vigneron
antoine@comp.nus.edu.sg

National University of Singapore
Tutorials

• preparation
  • you are not expected to be able to solve all the exercises
  • the most important thing is that you *try* to solve them
  • exercises with star are difficult

• you can write down your answers and pass them to me
  • I will mark them
  • but these marks will not count towards your final grade
Outline

• reference: Dave Mount lecture notes, lectures 6 and 7
• planar graphs
  • straight line planar graphs
  • trapezoidal map
  • polygons
• triangulation
  • existence
  • algorithm
Planar Graphs
Definition

- graph embedded in $\mathbb{R}^2$
- edges do not intersect in their interior

vertices $V = \{v_1, v_2, \ldots, v_7\}$
edges $E = \{e_1, e_2, \ldots, e_8\}$
faces $F = \{f_1, f_2, f_3, f_\infty\}$
Properties of planar graphs

- 1 infinite face ($f_\infty$)
- Euler relation:
  - connected planar graph $|V| - |E| + |F| = 2$
  - $c$ connected components $|V| - |E| + |F| - c = 1$
  - proof?
- Theorem: $|E| \leq 3(|V| - 2)$ and $|F| \leq 2(|V| - 2)$
  - proof page 26 of D. Mount lecture notes
Properties (2)

• every planar graph has a straight line embedding

• not proven here
Planar Straight Line Graphs
Planar Straight Line Graphs

- planar graph with only straight line edges
- also called planar subdivision
- a data structure: doubly connected edge list
  - each edge is replaced by two directed half–edges

- the half–edges enclosing a face form a counterclockwise cycle
Doubly connected edge list

- vertex $v$
  - coordinates
  - an incident half–edge $\text{IncidentEdge}(v) = (v, w)$
- half edge $\overrightarrow{e}$
  - 3 edges $\text{Twin}(\overrightarrow{e}), \text{Next}(\overrightarrow{e}), \text{Prev}(\overrightarrow{e})$
  - vertex $\text{Origin}(\overrightarrow{e})$
  - a face $\text{IncidentFace}(\overrightarrow{e})$
- face $f$
  - a half–edge $\overrightarrow{e}(f)$ of its exterior boundary
  - a half–edge of each face contained in $f$; they are stored in a list $L(f)$
Faces in Doubly Connected Edge Lists

\[
f \left( \begin{align*}
\vec{e}^r(f) &= \vec{e}_3 \\
L(f) &= \{\vec{e}_1, \vec{e}_2\}
\end{align*} \right)
\]
Special Cases

- Polyline: the edges form a chain

- Convex subdivision: all faces are convex

- Polygons: a face of a PLSG (see below)
Trapezoidal map
Trapezoidal map

- start with a PSLG $\mathcal{G}$
- the trapezoidal map $\mathcal{T}(\mathcal{G})$ is the convex subdivision obtained by drawing vertical edges downward and upward from each vertex

- we draw a bounding box around $\mathcal{G}$ so that there is no infinite face, hence all faces of $\mathcal{T}(\mathcal{G})$ trapezoids
Computing $\mathcal{T}(\mathcal{G})$

• assume $\mathcal{G}$ has $n$ vertices
• input: a representation of $\mathcal{G}$ (for instance, a doubly connected edge list)
• output: a representation of $\mathcal{T}(\mathcal{G})$
• general position assumption: no two vertices have same $x$–coordinate
• idea: we will use plane sweep
• a modified version of the intersection detection algorithm
• first step: sort vertices by increasing $x$–coordinate
• an event: the sweep line reaches a vertex of $\mathcal{G}$
Computing $\mathcal{T}(\mathcal{G})$

- Known trapezoids
- Active trapezoids
- Not processed yet
Computing $\mathcal{T}(G)$

- invariants
  - we know the trapezoids that lie on the left of the sweep line
  - *active trapezoids*: trapezoids that intersect the sweep line
  - we know the order of the active trapezoids along the sweep line
  - we know the left, top and bottom edges of each active trapezoid
- an event: close some active trapezoids and create new ones
Computing $\mathcal{T}(G)$

3 trapezoids are closed

4 new active trapezoids are created
Computing $T(G)$

- at event $i$ suppose $k_i$ trapezoids are closed or created
- event $i$ can be handled in $O(k_i \log n)$ time
- amortized analysis
  - $T(G)$ has at most $3n$ vertices
  - so there are $O(n)$ trapezoids
  - each trapezoid is created and closed one time only
  - so $\sum k_i = O(n)$
- overall, the algorithm runs in $O(n \log n)$ time
Polygons and Triangulations
Polygons

- A polygon is a face of a Planar Straight Line Graph
- A *simple polygon* is the region enclosed by a simple (=non-intersecting) polyline

![Diagram of simple polygon and polygon with holes]
A **Triangulation** of a polygon $P$ is a partition of $P$ into triangles whose vertices are the vertices of $P$.

A polygon may have several triangulations.

A triangulation is a planar straight line graph.
Applications

- meshing $\Rightarrow$ scientific computing
- visibility problems
  - graphics
  - art gallery problem (see Notes page 27)
- preprocessing step of many geometric algorithms
Existence of a triangulation

- We prove that every polygon $P$ admits a triangulation.
- Definition: a diagonal of $P$ is a line segment $pq$ such that $p$ and $q$ are vertices of $P$ and the interior of $pq$ is in the interior of $P$.

- Lemma 1: every polygon $P$ with more than three vertices admits a diagonal.
Proof of Lemma 1

- let $v$ be the leftmost vertex of $P$
- let $u$ and $w$ be its neighbors
- if $uw$ is a diagonal we are done
Proof of Lemma 1

• if \( \overline{uw} \) is not a diagonal

• let \( v' \) be the vertex in triangle \( (u, v, w) \) that is farthest from \( \overline{uw} \)

• then \( \overline{vv'} \) is a diagonal: if an edge was crossing it, one of its endpoints would be farther from \( \overline{uw} \) and inside \( (u, v, w) \)
Proof of existence

- Theorem: any polygon $P$ admits a triangulation
- Proof:
  - if $P$ has 3 vertices, then $P$ is its own triangulation
  - otherwise insert a diagonal of $f$
    - if $P$ becomes disconnected, we know by induction that the two faces can be triangulated, so we are done
    - if $P$ is still connected, repeat the process of inserting a diagonal
  - this algorithm halts since $|E| < |V|^2$ and $|V|$ is constant
More results

- any triangulation of a simple polygon with $n$ vertices has $n - 2$ faces and $n - 3$ diagonals
- we can find a diagonal in $O(n)$ time
- we can find a triangulation in $O(n^2)$ time
- is there a faster algorithm?
  - yes, there is an optimal $O(n \log n)$ time and $O(n)$ space algorithm
  - this is what we will see next
- there is an $O(n)$ time algorithm for simple polygons
  - very difficult, we do not study it
Triangulating a monotone polygon
Definition

• an $x$–monotone polygon is a polygon such that for all vertical line $l$, the intersection $P \cap l$ is a line segment.

• equivalently, it is a simple polygon whose boundary consists of two $x$–monotone polylines.
Algorithm

- plane sweep approach
- the sweep line $l$ moves from left to right and stops at each vertex of $P$
  - we can sort these vertices in $O(n \log n)$ time
  - we can also do it in $O(n)$ time. How?
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example

![Graph Example](image-url)
Example
Proof of correctness

- Invariant:
  - the non triangulated region to the left of the sweep line is delimited by an edge on one side and a reflex chain on the other side
  - we can maintain this invariant (see D. Mount notes)
Analysis

- vertices can be sorted along the \(x\)-axis in \(O(n)\) time
- we maintain the reflex chain in a stack
  - push and pop in \(O(1)\) time
- each vertex is pushed and popped at most once
- this algorithm runs in optimal \(\Theta(n)\) time
- we can use Doubly Connected Edge Lists
Partitioning a polygon into monotone pieces
Problem

- we want to partition a polygon $P$ into a collection of $x$–monotone polygons with same vertex set
Algorithm

- we will find an $O(n \log n)$ time algorithm
- combined with previous section, it yields an $O(n \log n)$ time algorithm to triangulate an arbitrary polygon
- idea: first compute the trapezoidal map
  - it takes $O(n \log n)$ time
  - exercise for next week: how to obtain a monotone partition once we have the trapezoidal map?