EDU - Tutorial on Computational Geometry (9201)

Oswin Aichholzer  20th June 2006, TU Graz
Thomas Hackl  21st June 2006, TU Graz
Tibor Steiner  22nd June 2006, TU Wien
Computational Geometry

- We have points and lines and something is happening …
Computational Geometry

- Computational Geometry typically deals with (many) (simple) objects of constant size (points and straight lines).
- Then we do something algorithmically or combinatorial …
- We are interested in the asymptotical time / space complexity of an algorithm or a structure. Depends on the input (and output) size.
- Main paradigm: Constant size operations need constant time.
  - Adding two numbers: one step, $O(1)$ time
  - Intersection of two circles: one step, $O(1)$ time
- Time needed (e.g. in seconds) depends on used primitives, precision, implementation, language, hardware, …
Computational Geometry

Two main development phases:

- Designing and analyzing the algorithm
  - design principles
  - geometrical, mathematical and combinatorial relations
  - (special) data structures
  - proofs of correctness in all cases (special cases)
  - analysis of asymptotical efficiency (average / worst case; time / space)
  - ...

- implementing the algorithm
  - constant size (time), but sometimes complex primitives
  - complicated data structures
  - special cases (ignored / not relevant for asymptotic efficiency)
  - numerical issues / robustness / exact computation
  - ...
Overview

- Part 1 (Oswin, Tuesday)
  - Introduction
  - Line segment intersection
  - Convex hulls, 2D
- Part 2 (Thomas, Wednesday)
  - Triangulations
  - Related structures (substructures, dual structures)
- Part 3 (Tibor, Thursday)
  - 3D and higher dimensional Algorithms
  - CGAL/Matlab demos
Algorithmic Paradigms

There are some basic principles when designing algorithms:

- Greedy algorithms
- Scan line principle
- Divide & conquer
- Dynamic programming
- Randomized algorithms
- …
Line Segment Intersection

- Given are $n$ segments in the plane. Do any pair of segments intersect?
Line Segment Intersection

- Application: Hidden line algorithms, layouts, collision detection, implicit in many geometric algorithms, …
Two segments intersect?

- When do two line segments intersect?
- Constant size problem 🚫 constant time operation!

- Implementation idea 1:
  - Solve a system of two linear equations for supporting lines: $y_i := k_i \cdot x_i + b_i$
    - Problems with vertical segments
    - Numerical problems for (near) parallel segments
- 🚫 bad idea!
Two segments intersect?

two line segments $ab$, $cd$ intersect

different orientations $abc$, $abd$ and
different orientations $cda$, $cdb$

compute the parity of the sign of the
determinant of four $3 \times 3$ matrices

orientation independent, numerical
stable, fast and reliable constant
time operation.
Naïve approach

- Testing two line segments takes constant $O(1)$ time
- First approach: Test any two pairs of line segments.
  - There are $n \cdot (n-1)/2 = \Theta(n^2)$ such pairs
  - Problem can be solved in quadratic time!
- There are examples with an quadratic number of line segment intersections:
- In the worst case we will need $\Theta(n^2)$ time to report them
- We can’t do better!
Improved approach

- Well, we can: We want to be faster if there are no or only few line segment intersections!
- Running time dependent on the input size \( (n=\text{number of segments}) \) AND output size \( (k=\text{number of intersections}) \)
  - output sensitive algorithm
- We will design an \( \mathcal{O}((n+k) \log n) \) time and \( \mathcal{O}(n+k) \) space algorithm.
Basic ideas

- Observation: If two segments intersect, their x-interval overlap.
Plane Sweep Technique

- Use a Scan Line L:
  - Traverse the scenario in x-direction
  - act locally
Plane Sweep Technique

- Scan Line L:
  - segments on L are sorted in y-direction
  - intersecting segments are neighbored in this sorting
Plane Sweep Technique

- Algorithm:
  - scan line is event driven (NOT continuous!)
  - maintain y-sorting for each event

segment start: insert in y-sorting
segment end: delete from y-sorting
intersection: switch in y-sorting
Pseudo-Code

- \( X = \emptyset, \ Y = \emptyset \)
- insert all start- and endpoints of segments into \( X \)
- WHILE \( X \neq \emptyset \) DO
  - \( m = \min\{X\} \), delete \( m \) from \( X \)
  - IF \( m \) is left endpoint THEN insert segment in \( Y \)
    ELSE IF \( m \) is right endpoint THEN delete segment from \( Y \)
    ELSE switch the two segments intersecting at \( m \)
  - FOR all new neighbors \( s_1s_2 \) in \( Y \) DO
    - IF \( s_1s_2 \) intersect in \( p \) to the right of \( L \) THEN
      - insert \( p \) in \( X \)
      - report \( p \)
Details on the Implementation I

- X-data structure handles ‘events’:
  - insert start- and endpoints and detected intersection
  - find x-minimum
  - delete x-minimum

- Priority queue problem, use e.g. heap data structure
  - per segment 2 start/end points inserted \( (n \text{ times}) \)
  - per intersection one insertion \( (k \text{ times}) \)
  - Delete each element when scanned over \( (n+k \text{ times}) \)
  - each step takes time logarithmic in the structure size
  - \( (n+k) \log (n+k) \) = \( (n+k) \log n \) time and \( (n+k) \) space
Details on the Implementation II

- Y-data structure handles ‘sorted segments on L’:
  - insert segments into sorting
  - Delete segments from sorting
  - Switch two segments in sorting

- Dictionary problem, use e.g. (2-4)-trees
  - per segment 1 points inserted, 1 deleted \((n \text{ times})\)
  - per intersection one switch \((k \text{ times})\)
  - Insertion, deletion takes time logarithmic in the structure size, switch takes constant time
  - \(\Theta(n \log n + k)\) time and \(\Theta(n)\) space
Details on the Implementation III

- Scan line reduces a 2-dimensional static problem to a 1-dimensional dynamic problem
- Insertion of segments in Y-sorting: segments, NOT values!
- Intersections found in arbitrary order (NOT X-sorted)
- Intersections might be found multiple times
- Presented algorithm has $(n+k) \log n$ time and $(n+k)$ space
- Can be improved to $(n \log n + k)$ time and $(n)$ space
- To only detect whether intersections exist (k={0,1}): optimal $(n \log n)$ time and $(n)$ space suffices
- Algorithm can be used for other geometric primitives, like circles, disks, …
Convex Hulls

- Let $S=\{p_1, p_2, \ldots, p_n\}$ be a set of $n$ points.
- The convex hull of $S$ is the smallest convex polygon containing $S$.

Applications:
- Diameter of point set
- Linear separability of set
- Width of a polygon
- Implicit in many CG-algorithms
Graham Scan

There exist many time and space optimal solutions to compute the convex hull.

Graham Scan [1972]:
- easy to implement 2D-approach:
- a modified (rotational) plane sweep (scan line) technique
  - 1) sort points cyclically
  - 2a) scan points in this order
  - 2b) remove non-extreme points
Graham Scan

1) sort points cyclically
2a) scan points in this order
2b) remove non-extreme points

- scan line: act only locally
- maintain convex hull of subset considered so far
Graham Scan

data-structure: STACK

- PUSH, POP as usual
- TOP topmost element
- TOP\(^+\) second element from top

Algorithm:

- \(p_1\) = point with minimum y-coordinate
- sort all other points cyclically around \(p_1\)
- \(\text{PUSH}(p_1, p_2, p_3)\)
- FOR \(i = 4\) TO \(n\) DO
  - WHILE \(\angle(TOP^+, TOP, p_i) > \pi\) DO
    - POP
    - \(\text{PUSH}(p_i)\)
Analysis Graham Scan

Algorithm:
- $p_1 =$ point with minimum y-coordinate
- sort all other points cyclically around $p_1$
- PUSH($p_1$, $p_2$, $p_3$)
- FOR i = 4 TO n DO
  - WHILE $\angle(TOP^+, TOP, p_i) > \pi$ DO
    - POP
    - PUSH($p_i$)

Analysis:
- $\mathcal{O}(n)$ time
- Sorting takes $\mathcal{O}(n \log n)$ time
- Each stack operation needs constant time
- Every point is popped at most once
- Every point is pushed at most once

A single WHILE loop may take linear time, BUT: all WHILE loops together take $\mathcal{O}(n)$ time

Theorem: Graham Scan computes the convex hull of $n$ points in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.
Remarks on Graham Scan

- Easy to implement, reliable, stable
- Time and space optimal: No deterministic sorting algorithm using element-comparison can be faster than $\Omega(n \log n)$.
- Efficient:
  - $\log n$ - terms come from sorting (small constants)
  - geometric part (left/right decisions …) runs in $\mathcal{O}(n)$ time
- Can be used to triangulate a point set in $\mathcal{O}(n \log n)$ time (see EDU-Talk tomorrow)
- Implicit 2-dimensional: sorting not possible in 3D or higher
  - No way to mimic Graham Scan for higher dimensions
Iterative Insertion

New approach: Iterative Insertion

- To complicated for 2D, but can be generalized to arbitrary dimension

- Randomized algorithm:
  - good expected running time (with high probability)
  - expectation of running time does not depend on input, but only on random-generator (coin, dice, …)
  - result is deterministic
  - Running time might be bad in the worst case, but probability smaller than collapse of operating system …
Iterative Insertion

Algorithm:

- start with random CH = triangle $p_1p_2p_3$
  (CH = tetrahedron $p_1p_2p_3p_4$ in 3D etc.)
- consider remaining points in random order
- IF $p_i$ lies inside CH THEN ignore $p_i$
  ELSE update CH with $p_i$
  • add tangents from $p_i$ to CH
  • remove inner elements of CH
Analysis I

- Analysis:
  - per iteration add $\leq 2$ new edges
  - delete $k_i \geq 0$ inner edges
  - $k_i$ might be large!
  - observe: $\sum_{i=4}^{n} k_i \leq 2(n-3)$
  - $\bigcup (n)$ steps

- But: we assumed **oracle** to
  - decide whether $p_i \in CH$
  - provide visible (inner/to delete) edge of CH
Oracle:
- take center point \( c = (p_1 + p_2 + p_3) / 3 \)
- connect all not-yet-inserted points to \( c \)
- store intersected edge of \( CH \) as witness (if it exists!)
- when a point is inserted, use witness
- witnesses have to be updated!
New witnesses can be found in $O(1)$ time per witness (choose one of two tangents)

Running time is $O(\text{number of witnesses}) + O(n)$ (see slide Analysis I)

$E[\text{number of witnesses}] = n \cdot E[\text{number of witnesses per point}]$

$\ldots$ (next slide) $\ldots = n \cdot \Theta(\log n)$

Overall (expected) running time: $O(n \log n)$

memory: $O(n)$ to store witness information
Analysis III

- E [ number of witnesses per point ]:
  - let $q$ be a fixed point, inserted (in random order) as $i^{th}$ point
  - let $p$ be a fixed point, inserted as $j^{th}$ point, before $q$ ($j < i$)
  - probability that the witness of $q$ changes, when $p$ is inserted $= \frac{2}{j}$
  - $E [ \text{number of witnesses for } q ] = \sum_{j=4}^{i} \frac{2}{j} < 2 \cdot \sum_{j=1}^{i} \frac{1}{j} = 2 \cdot H_{i-1} = \Theta (\log i) = \Theta (\log n)$
  - $H_i \ldots$ harmonic numbers
Notes on Iterative Insertion

- Can be generalized to 3D and beyond
  - start with tetrahedron (not triangle)
  - uses faces as witnesses (not edges)
- Needs randomization: not an on-line algorithm
- Insertion of one point might take linear time, but very unlikely
- No special distribution for point required
- Can be derandomized to run in deterministic $\mathcal{O}(n \log n)$ time
Further Reading

- Animation of convex hull algorithms (2D/3D) see e.g.:
  [Link](http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html)

- More about the design and analysis of algorithms:
  [Link](http://www.igi.tugraz.at/oaich/vorlesungen/e&a.html)

- Books:
  - *Introduction to Algorithms*
    Cormen, Leiserson, Rivest, Stein, MIT Press
  - *Computational Geometry – Algorithms and Applications*
    de Berg, van Kreveld, Overmars, Schwarzkopf, Springer-Verlag
  - *Algorithms in Combinatorial Geometry*
    Edelsbrunner, Springer-Verlag
  - *Computational Geometry*
    Preparata, Shamos, Springer-Verlag
Thanks ...

Thank you!

to be continued ... tomorrow

Danke!
23rd European Workshop on Computational Geometry

EWCG 07 March 19 – 21 2007, Graz, Austria

http://ewcg07.tugraz.at