Sublinear Geometric Algorithms

Successor search. Polygonal and polyhedral intersections

Mikhail Dubov

March 22, 2016

Outline

Outline

Context: Geometric algorithms

- **70's-90's: Classical computational geometry algorithms**
  - → Convex hulls
  - → Voronoi diagrams
  - → Delaunay trianguations
  - → Linear programming

Context: Geometric algorithms

- **70's-90's: Classical computational geometry algorithms**
  - → Convex hulls
  - → Voronoi diagrams
  - → Delaunay trianguations
  - → Linear programming

- **00's: Research on sublinear algorithms**
  - **Motivation:** Availability of massive geometric datasets
  - **Problem:** Impossible to examine more than a fraction of the input

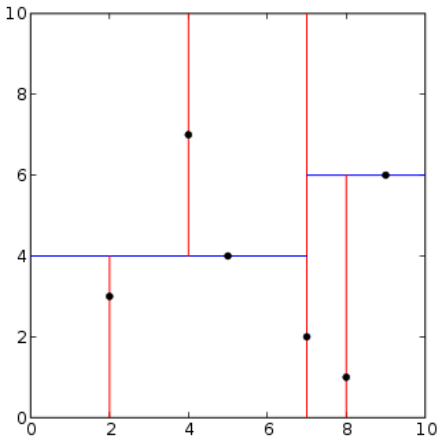
Context: Sublinear geometric algorithms

Two main approaches to achieve sublinearity:

Context: Sublinear geometric algorithms

Two main approaches to achieve sublinearity:

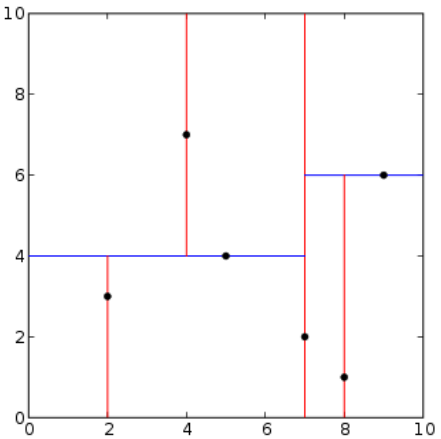
- **Data preprocessing**
  - Look at the whole data once, make the subsequent queries fast
  - **Example:** Point location in $\mathbb{R}^k$
    - One can build a kd-tree in $O(n)$
    - Nearest neighbor search: $O(\log n)$ per query on average



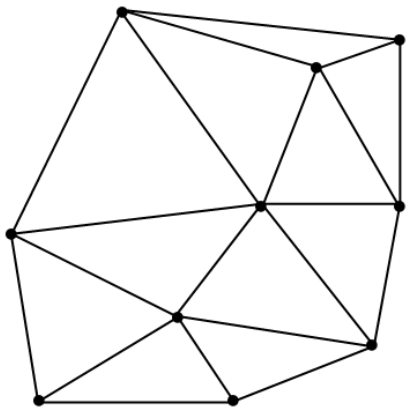
Context: Sublinear geometric algorithms

Two main approaches to achieve sublinearity:

- **Data preprocessing**
  - Look at the whole data once, make the subsequent queries fast
  - **Example:** Point location in $\mathbb{R}^k$
    - One can build a kd-tree in $O(n)$
    - Nearest neighbor search: $O(\log n)$ per query on average



- **Randomization**
  - Look only at a portion of the data
  - **Example:** Point location in Delaunay triangulations *(stay tuned)*
    - Expected $O(\sqrt{n})$ time per query **without preprocessing**



Context: Randomized algorithms

Two main types of randomized algorithms:

Context: Randomized algorithms

Two main types of randomized algorithms:

- **Monte Carlo algorithms:**
  - Running time is bounded
  - $\mathbb{P}(\text{Results are correct}) < 1$

Context: Randomized algorithms

Two main types of randomized algorithms:

- **Monte Carlo algorithms:**
  - Running time is bounded
  - $\mathbb{P}(\text{Results are correct}) < 1$

  Ex.: Miller-Rabin primality test

Context: Randomized algorithms

Two main types of randomized algorithms:

- **Monte Carlo algorithms:**
  - Running time is bounded
  - $\mathbb{P}$(Results are correct) < 1

  Ex.: Miller-Rabin primality test

- **Las Vegas algorithms:**
  - $\mathbb{E}$(Running time) is bounded
  - Results are always correct

Context: Randomized algorithms

Two main types of randomized algorithms:

- **Monte Carlo algorithms:**
  - Running time is bounded
  - $\mathbb{P}(\text{Results are correct}) < 1$

  Ex.: Miller-Rabin primality test

- **Las Vegas algorithms:**
  - $\mathbb{E}(\text{Running time})$ is bounded
  - Results are always correct

  Ex.: Randomized QuickSort

3	12	31	34	38	62	97

Random pivot selection $\Longrightarrow$ $O(n^2)$ time very unlikely

Context: Randomized algorithms

Two main types of randomized algorithms:

- **Monte Carlo algorithms:**
  - Running time is bounded
  - $\mathbb{P}(\text{Results are correct}) < 1$

  Ex.: Miller-Rabin primality test

- **Las Vegas algorithms:**
  - $\mathbb{E}(\text{Running time})$ is bounded
  - Results are always correct

  Ex.: Randomized QuickSort

3	12	31	34	38	62	97

Random pivot selection $\Longrightarrow$ $O(n^2)$ time very unlikely

$\rightarrow$ Monte Carlo algorithms **sometimes give wrong answers**

$\rightarrow$ Las Vegas algorithms **sometimes run for a very long time**

Our focus

In what follows, we will cover several

- **randomized Las Vegas** algorithms

Our focus

In what follows, we will cover several

- **randomized Las Vegas** algorithms
- with **no extra preprocessing**

Our focus

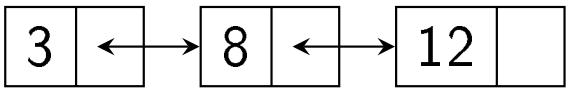
In what follows, we will cover several

- **randomized Las Vegas** algorithms
- with **no extra preprocessing**
- having $\mathbb{E}(\text{Running time}) = O(\sqrt{n})$.

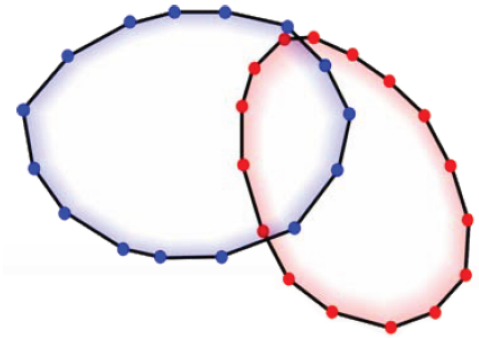
Our focus

In what follows, we will cover several

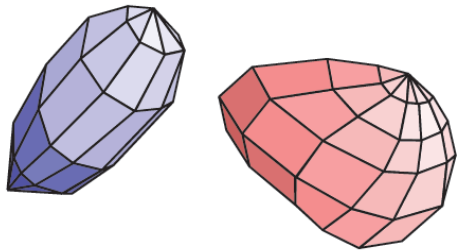
- **randomized Las Vegas** algorithms
- with **no extra preprocessing**
- having $\mathbb{E}(\text{Running time}) = O(\sqrt{n})$.



1D: Sucessor search



2D: Polygonal intersection



3D: Polyhedral intersection

Credit for images: Chazelle et al.

Outline

1D: Successor searching

Problem

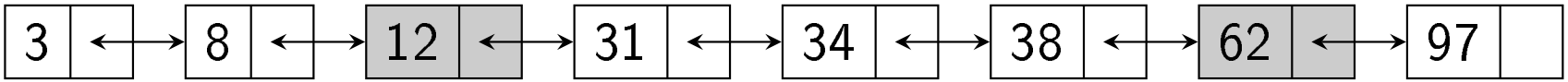
Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

Example:



$succ(50) = 62$
$succ(12) = 12$

1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

Complexity depends on how the list is stored:

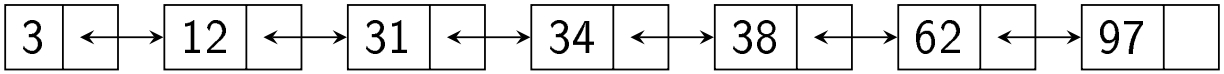
1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

Complexity depends on how the list is stored:

1. The location of elements is unknown



$\rightarrow O(n)$ time, $o(n)$ impossible

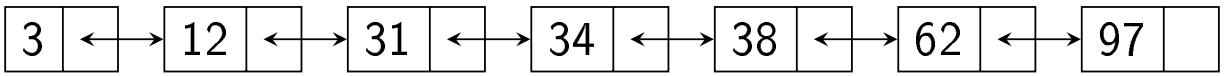
1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

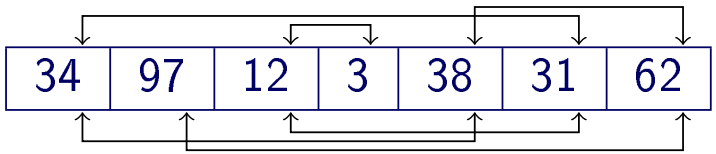
Complexity depends on how the list is stored:

1. The location of elements is unknown



$\rightarrow$ $O(n)$ time, $o(n)$ impossible

2. **List elements are stored in consecutive locations** ✓



$\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])

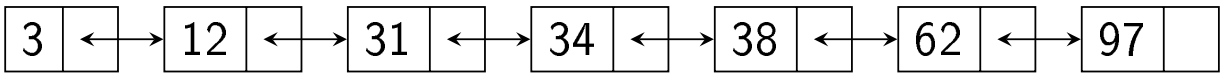
1D: Successor searching

Problem

Given a sorted doubly-linked list of n keys and a number x, find the smallest key $y \geq x$ (the successor of x).

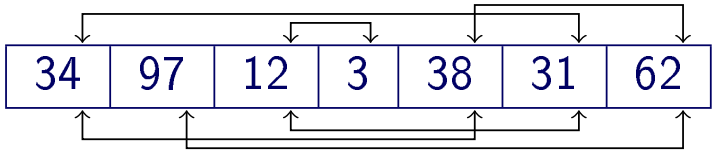
Complexity depends on how the list is stored:

1. The location of elements is unknown



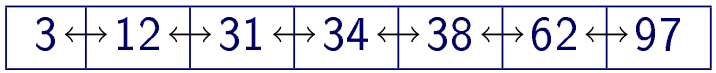
$\rightarrow$ $O(n)$ time, $o(n)$ impossible

2. **List elements are stored in consecutive locations** ✓



$\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])

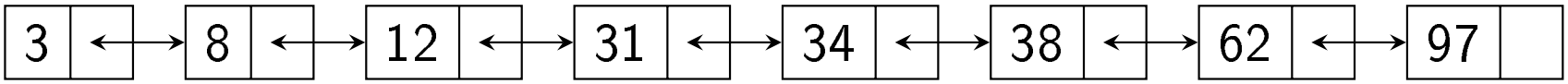
3. Elements are consecutive and ordered



$\rightarrow$ $O(\log n)$ time (binary search)

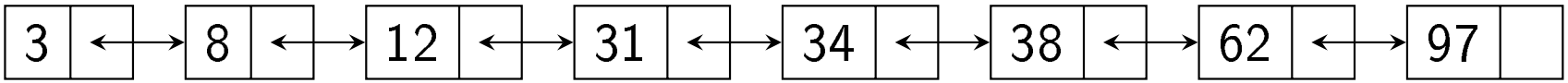
Successor searching: Idea of an algorithm

$succ(50) = ?$

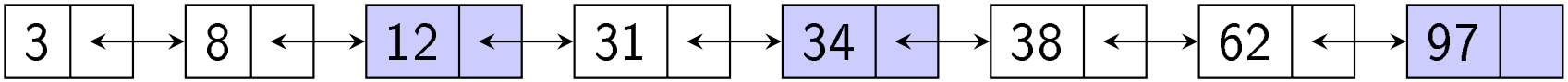


Successor searching: Idea of an algorithm

$succ(50) = ?$

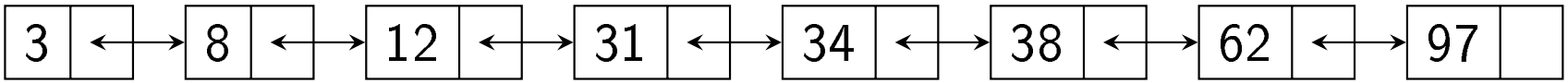


Sample some elements from the list:

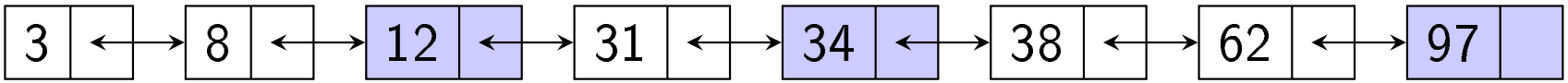


Successor searching: Idea of an algorithm

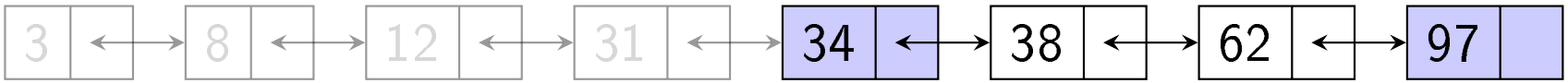
$succ(50) = ?$



Sample some elements from the list:

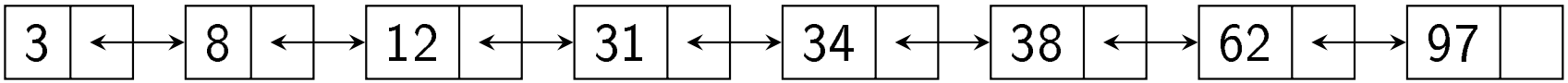


Find the elements in the sample that surround our target:

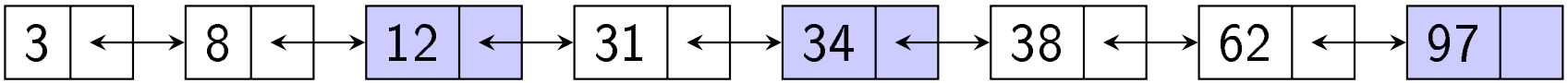


Successor searching: Idea of an algorithm

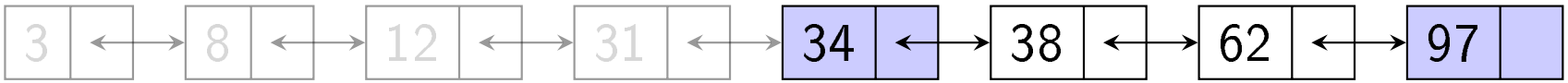
$succ(50) = ?$



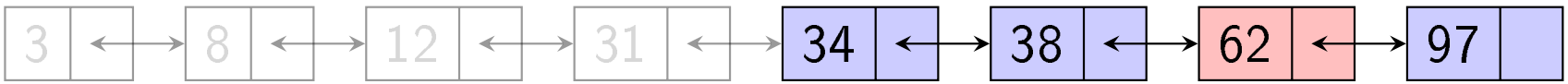
Sample some elements from the list:



Find the elements in the sample that surround our target:



Traverse the sublist and find the successor:



Successor searching: Randomized algorithm

Algorithm: Randomized successor searching in $O(\sqrt{n})$ time

Input : Doubly-linked list $A = A[1] \dots A[n]$ of n numbers stored in an array (table); number x

Output: The smallest number $y \in A$ s.t. $y \geq x$ (if exists)

1 Sample $S \subset A$, $|S| = \sqrt{n}$, from A uniformly at random

2 $p = \text{argmax}_{i=1\dots\sqrt{n}} S[i]$ s.t. $S[i] \leq x$ // predecessor in S

3 $q = \text{argmin}_{i=1\dots\sqrt{n}} S[i]$ s.t. $S[i] \geq x$ // successor in S

4 $s = \text{argmin}_{i=p\dots s} A[i]$ s.t. $A[i] \geq x$ // traverse A from p

5 **return** $A[s]$

Successor searching: Randomized algorithm

Algorithm: Randomized successor searching in $O(\sqrt{n})$ time

Input : Doubly-linked list $A = A[1] \dots A[n]$ of n numbers stored in an array (table); number x

Output: The smallest number $y \in A$ s.t. $y \geq x$ (if exists)

1 Sample $S \subset A$, $|S| = \sqrt{n}$, from A uniformly at random

2 $p = \text{argmax}_{i=1\dots\sqrt{n}} S[i]$ s.t. $S[i] \leq x$ // predecessor in S

3 $q = \text{argmin}_{i=1\dots\sqrt{n}} S[i]$ s.t. $S[i] \geq x$ // successor in S

4 $s = \text{argmin}_{i=p\dots s} A[i]$ s.t. $A[i] \geq x$ // traverse A from p

5 **return** $A[s]$

- **Note:** p and q may not exist
- Elements should be:
  - **consecutive** for efficient sampling in step 1
  - **doubly-connected** for list traversal in step 4

Successor searching: Analysis

Theorem

Successor searching can be done in $O(\sqrt{n})$ expected time per query, which is optimal.

Successor searching: Analysis

Theorem

Successor searching can be done in $O(\sqrt{n})$ expected time per query, which is optimal.

Proof

Some intuition:

- Our sample S is a subset of $\sqrt{n}$ elements from A, $|A| = n$
- We locate p and q in our sample in $O(|S|) = O(\sqrt{n})$ time
- The expected distance between these two elements is $|A|/|S| = n/\sqrt{n} = \sqrt{n}$
- So traversing $A[p] \dots A[q]$ takes $O(\sqrt{n})$ expected time

Successor searching: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Let $A[s]$ be the desired successor

Successor searching: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Let $A[s]$ be the desired successor
- Let $S[k]$ be the nearest element to A among those in S

Successor searching: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Let $A[s]$ be the desired successor
- Let $S[k]$ be the nearest element to A among those in S
- Event Q_d: we don't hit any of $A[s-d] \dots A[s+d]$ after taking $\sqrt{n}$ random samples

Successor searching: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Let $A[s]$ be the desired successor
- Let $S[k]$ be the nearest element to A among those in S
- Event Q_d: we don't hit any of $A[s-d] \dots A[s+d]$ after taking $\sqrt{n}$ random samples
- $\mathbb{P}(dist(A[s], S[k]) = d) = \mathbb{P}(Q_{d-1}) - \mathbb{P}(Q_d)$

Successor searching: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Let $A[s]$ be the desired successor
- Let $S[k]$ be the nearest element to A among those in S
- Event Q_d: we don't hit any of $A[s-d] \dots A[s+d]$ after taking $\sqrt{n}$ random samples
- $\mathbb{P}(dist(A[s], S[k]) = d) = \mathbb{P}(Q_{d-1}) - \mathbb{P}(Q_d)$
- $$\begin{aligned}\mathbb{E}(dist(A[s], S[k]) &= \sum_{i \geq 1} i \cdot (\mathbb{P}(Q_{i-1}) - \mathbb{P}(Q_i)) = \sum_{i \geq 0} \mathbb{P}(Q_i) \\ &\leq \sqrt{n} \sum_{c \geq 0} \mathbb{P}(Q_{c\sqrt{n}}) \leq \sqrt{n} \sum_{c \geq 0} (1 - c/\sqrt{n})^{\sqrt{n}} \\ &\leq \sqrt{n} \sum_{c \geq 0} e^{-c} = O(\sqrt{n})\end{aligned}$$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Yao's minimax principle:**

$$\mathbb{E}(\text{Running time of the optimal Las Vegas } \textbf{randomized algorithm})$$
$$\geq$$
$$\mathbb{E}(\text{Running time of an optimal } \textbf{deterministic algorithm} \text{ for any fixed input distribution})$$

- Study the distribution of difficult inputs
- Show that no deterministic algorithm can perform well on it

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- Our model:
  - **Input:** a permutation σ of $1 \dots n$, s.t. $A[\sigma(i)] = i$
  - **Goal:** find $succ(n)$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- Our model:
  - **Input:** a permutation σ of $1 \dots n$, s.t. $A[\sigma(i)] = i$
  - **Goal:** find $succ(n)$
- The optimal deterministic algorithm is a sequence of two types of operations:
  - **"Operation A":**
    - Pick a visited location $\sigma(i)$
    - Visit one of its neighbours: $T[\sigma(i-1)]$ or $T[\sigma(i+1)]$
  - **"Operation B":** Visit some unvisited $T[\sigma(i)]$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

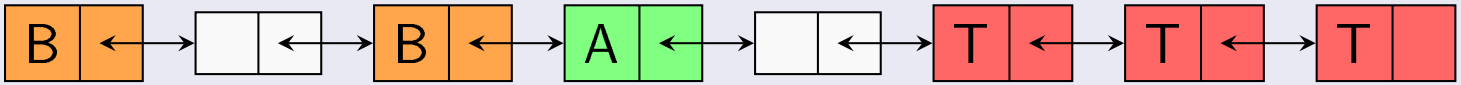
- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?

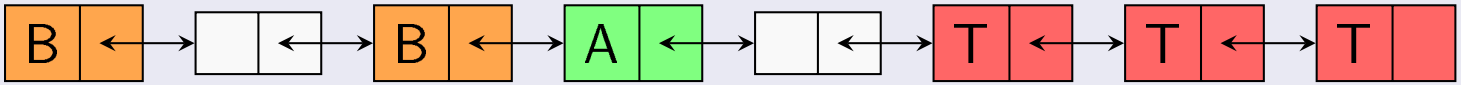


Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?



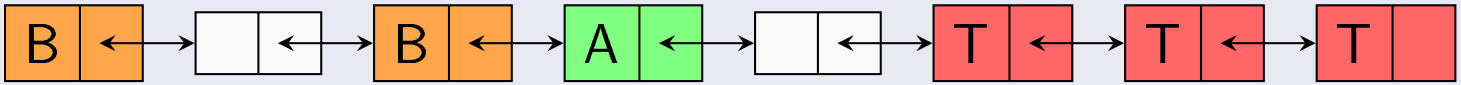
- A very crude estimate:
  $\mathbb{P}(\text{tail item not picked after } (a+b) \text{ ops.}) \geq (1 - \frac{\sqrt{n}+a+b}{n})^{a+b}$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?



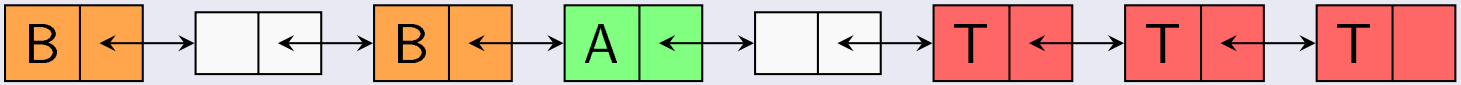
- A very crude estimate:
  $\mathbb{P}(\text{tail item not picked after (a+b) ops.}) \geq (1 - \frac{\sqrt{n}+a+b}{n})^{a+b}$
- Assume the total number of operations before visiting one of the last $\sqrt{n}$ elements in the tail of the list does not exceed $\sqrt{n}$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?



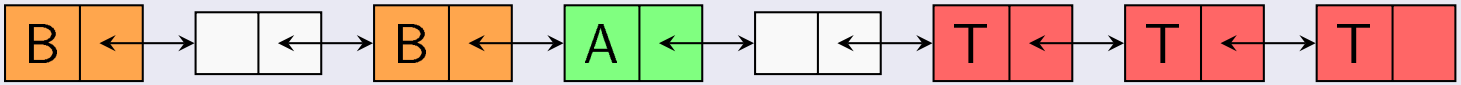
- A very crude estimate:
  $\mathbb{P}(\text{tail item not picked after (a+b) ops.}) \geq (1 - \frac{\sqrt{n}+a+b}{n})^{a+b}$
- Assume the total number of operations before visiting one of the last $\sqrt{n}$ elements in the tail of the list does not exceed $\sqrt{n}$
- However, we still have:
  $\mathbb{E}(a+b) \geq \sum_{a+b=1}^{\sqrt{n}} (a+b+1)\frac{\sqrt{n}}{n}(1 - \frac{\sqrt{n}+a+b}{n})^{a+b} = \Omega(\sqrt{n})$

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?



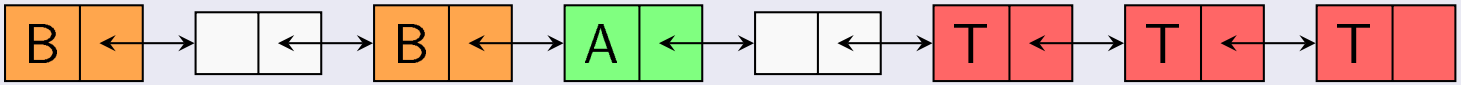
- A very crude estimate:
  $\mathbb{P}(\text{tail item not picked after (a+b) ops.}) \geq (1 - \frac{\sqrt{n}+a+b}{n})^{a+b}$
- Assume the total number of operations before visiting one of the last $\sqrt{n}$ elements in the tail of the list does not exceed $\sqrt{n}$
- However, we still have:
  $\mathbb{E}(a+b) \geq \sum_{a+b=1}^{\sqrt{n}} (a+b+1)\frac{\sqrt{n}}{n}(1 - \frac{\sqrt{n}+a+b}{n})^{a+b} = \Omega(\sqrt{n})$
- Once in the tail, the deterministic algorithm can get to the last element in $O(\sqrt{n})$ operations "A"

Successor searching: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- **Question:** On a random input, how likely is it to discover one of the last $\sqrt{n}$ items after **a** operations "A" and **b** operations "B"?



- A very crude estimate:
  $\mathbb{P}(\text{tail item not picked after (a+b) ops.}) \geq (1 - \frac{\sqrt{n}+a+b}{n})^{a+b}$
- Assume the total number of operations before visiting one of the last $\sqrt{n}$ elements in the tail of the list does not exceed $\sqrt{n}$
- However, we still have:
  $\mathbb{E}(a+b) \geq \sum_{a+b=1}^{\sqrt{n}} (a+b+1)\frac{\sqrt{n}}{n}(1 - \frac{\sqrt{n}+a+b}{n})^{a+b} = \Omega(\sqrt{n})$
- Once in the tail, the deterministic algorithm can get to the last element in $O(\sqrt{n})$ operations "A"
- So the average running time of our optimal deterministic algorithm is $\Omega(\sqrt{n})$

Outline

2D: Convex polygonal intersection

Problem

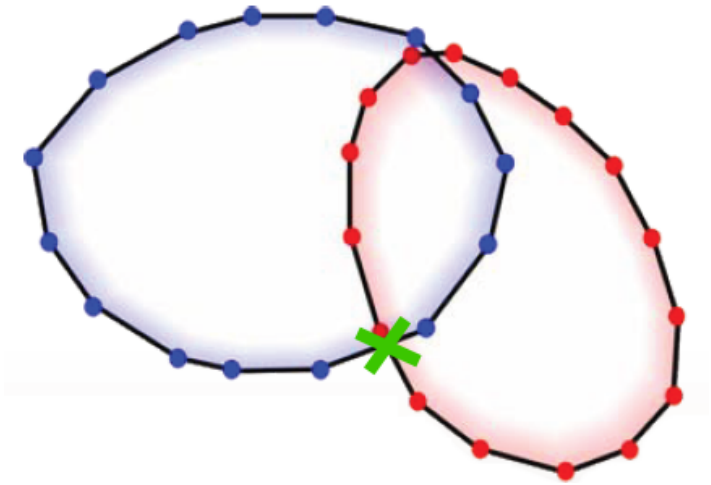
Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

Example:



Intersection: **YES**

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

Again, complexity depends on how polygons are stored:

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

Again, complexity depends on how polygons are stored:

1. General case
   $\rightarrow O(n)$ time (e.g. via Linear Programming, Seidel [1990])

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

Again, complexity depends on how polygons are stored:

1. General case
   $\rightarrow O(n)$ time (e.g. via Linear Programming, Seidel [1990])

2. **Vertices are stored in a clockwise order in a doubly-linked list (allowing for random sampling)** ✓
   $\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])

2D: Convex polygonal intersection

Problem

Given two convex polygons P and Q, with n vertices each, determine whether they intersect or not and, if they do, report one point in the intersection.

Again, complexity depends on how polygons are stored:

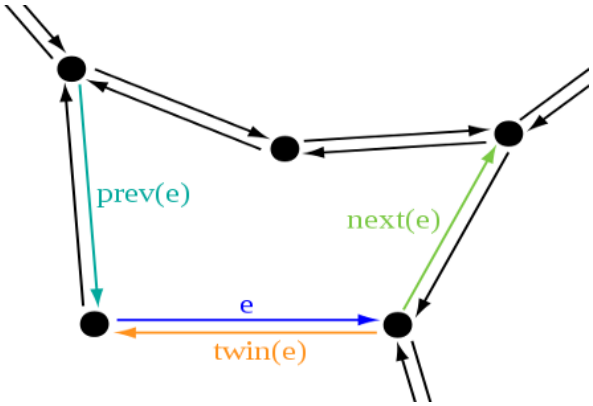
1. General case
   $\rightarrow O(n)$ time (e.g. via Linear Programming, Seidel [1990])
2. **Vertices are stored in a clockwise order in a doubly-linked list (allowing for random sampling)** ✓
   $\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])
3. Vertices are stored in an array in cyclic order
   $\rightarrow O(\log n)$ time (Chazelle and Dobkin [1987])

Detour: Classical geometric data structures

Idea: allow for quick traversal between faces, edges, and vertices due to the explicitly linked structure of the objects.

Detour: Classical geometric data structures

Idea: allow for quick traversal between faces, edges, and vertices due to the explicitly linked structure of the objects.

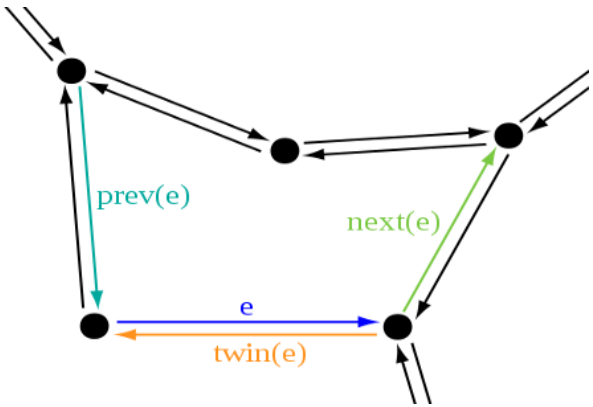


Doubly connected edge list (DCEL)

- Doubly-linked list of half-edges
- Each half-edge bounds a single face
- Standard data structure in **CGAL** (`www.cgal.org`)

Detour: Classical geometric data structures

Idea: allow for quick traversal between faces, edges, and vertices due to the explicitly linked structure of the objects.

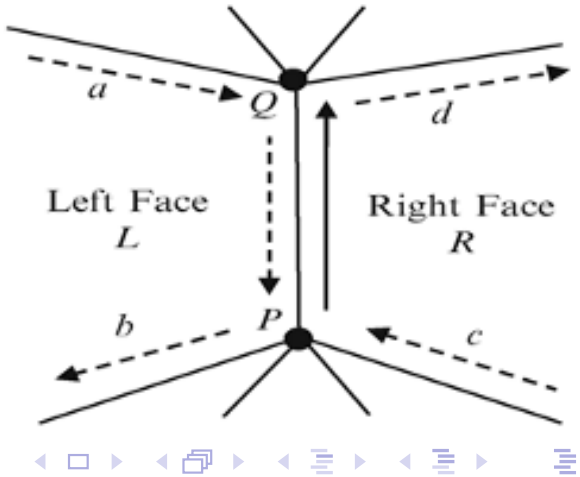


Doubly connected edge list (DCEL)

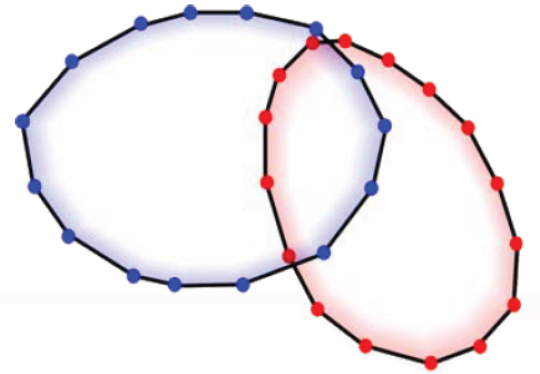
- Doubly-linked list of half-edges
- Each half-edge bounds a single face
- Standard data structure in **CGAL** (`www.cgal.org`)

Winged edge

- Also edge-based
- For each edge, stores its vertices, left and right faces etc.

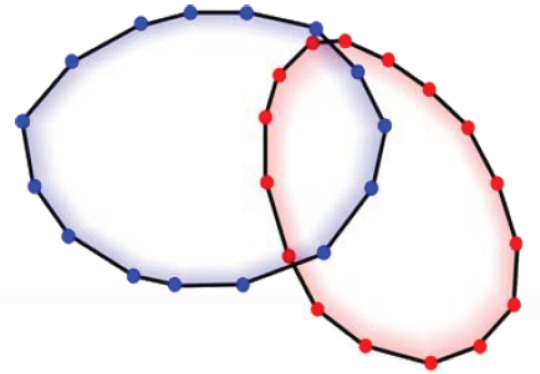


Convex polygonal intersection: Idea of an algorithm

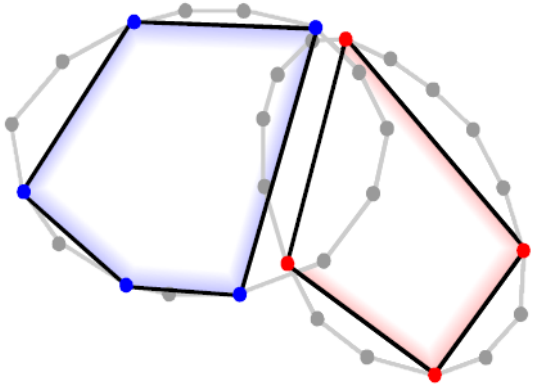


Start with the input polygons

Convex polygonal intersection: Idea of an algorithm

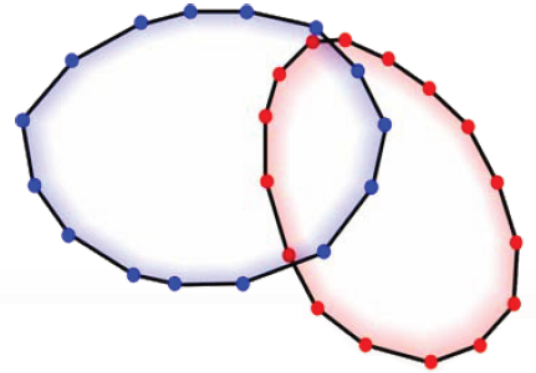


Start with the input polygons

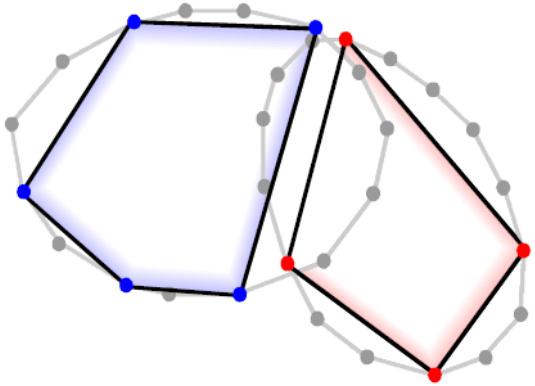


Simplify polygons by sampling. Check if they intersect

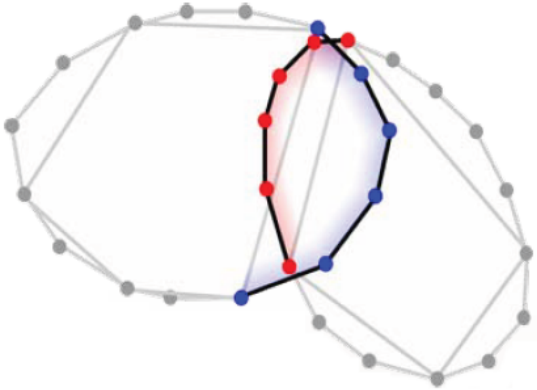
Convex polygonal intersection: Idea of an algorithm



Start with the input polygons



Simplify polygons by sampling. Check if they intersect



If not, test the potentially overlapping region

Convex polygonal intersection: Randomized algorithm

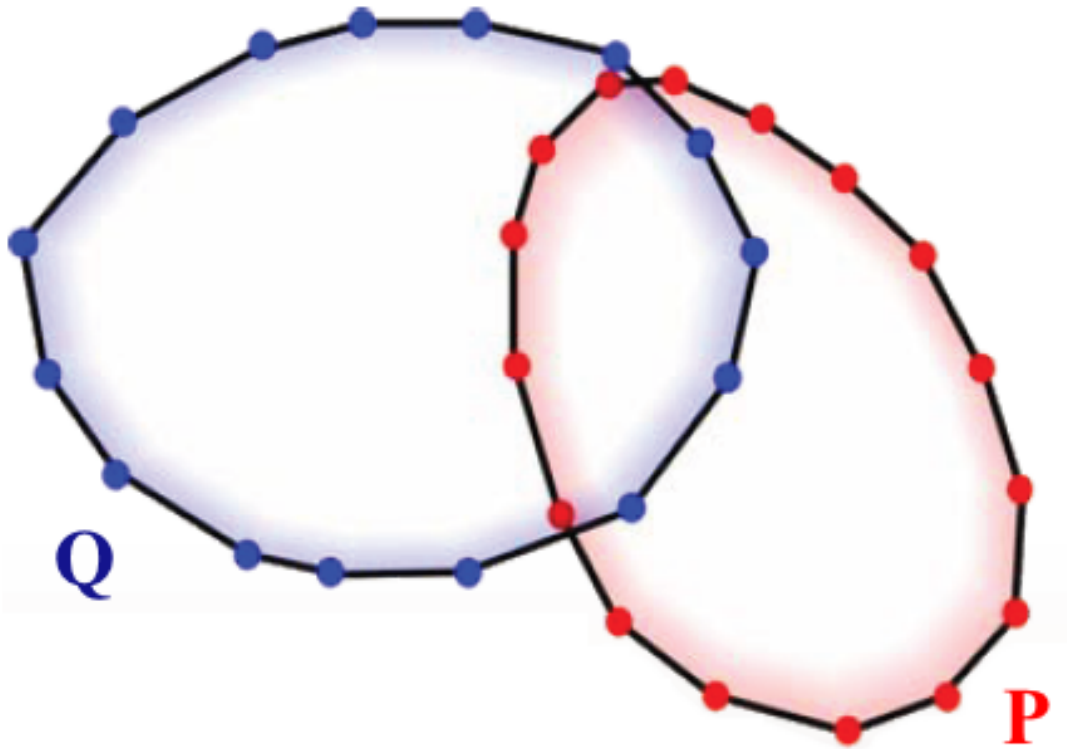
Algorithm: Convex polygonal intersection in $O(\sqrt{n})$ time

Input : Two convex polygons P and Q, $|P_{\text{vertices}}| = |Q_{\text{vertices}}| = n$
Output: Report one point in $P \cap Q$ (if they intersect)

1 Sample $S_P \subset P_{\text{vertices}}$, $S_Q \subset Q_{\text{vertices}}$, $|S_P| = |S_Q| = \sqrt{n}$ from P and Q randomly
2 Let $R_P = conv(S_P) \subset P$, $R_Q = conv(S_Q) \subset Q$ `// convex hulls (conceptual)`
`// Linear Programming finds` $R_P \cap R_Q$ `without computing convex hulls`
3 **if** $R_P \cap R_Q \neq \emptyset$ **then**
4 | **return** *Intersection point of* R_P *and* R_Q
5 **end**
6 **else**
7 | $\mathcal{L}$ = bi-tangent separating line for R_P and R_Q
8 | C_P = part of P to the R_Q side of $\mathcal{L}$
9 | C_Q = part of Q to the R_P side of $\mathcal{L}$
10 | Check R_P and C_Q for intersection. If they don't intersect, find a line $\mathcal{L}'$ that separates R_P and C_Q, and compute C'_P, the part of P on the other side of $\mathcal{L}'$. Test C'_P and C_Q for intersection.
11 | If no intersection is found, repeat step 10 for P and Q swapped.
12 **end**

Convex polygonal intersection: Randomized algorithm

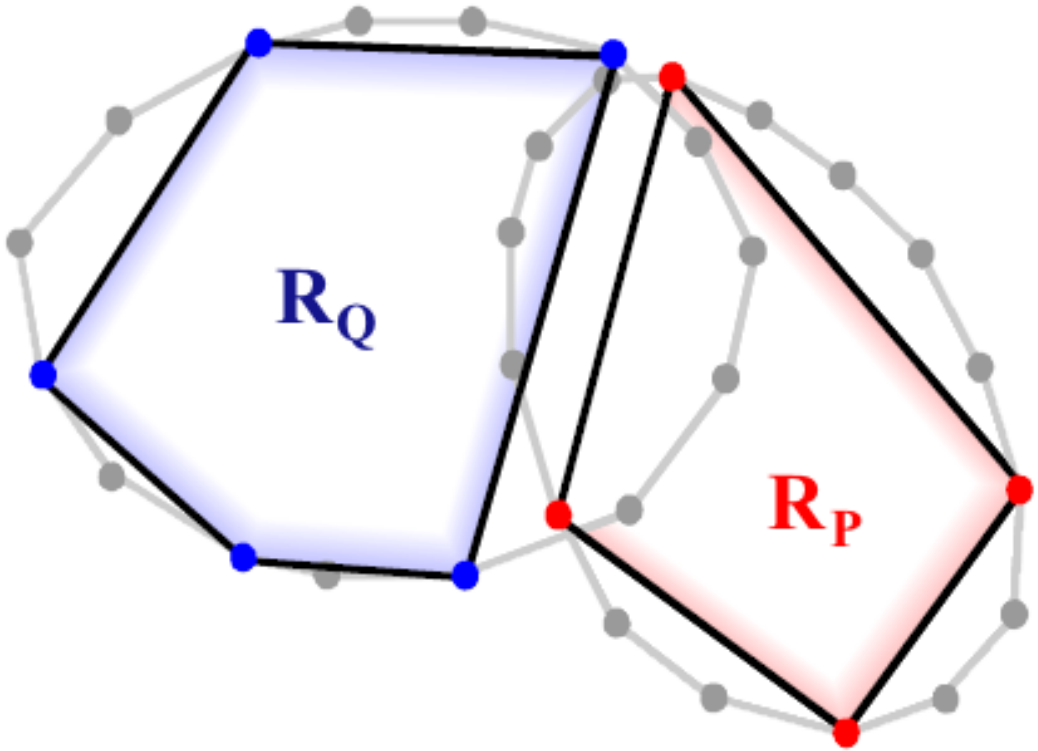
Example



Input: Polygons P and Q

Convex polygonal intersection: Randomized algorithm

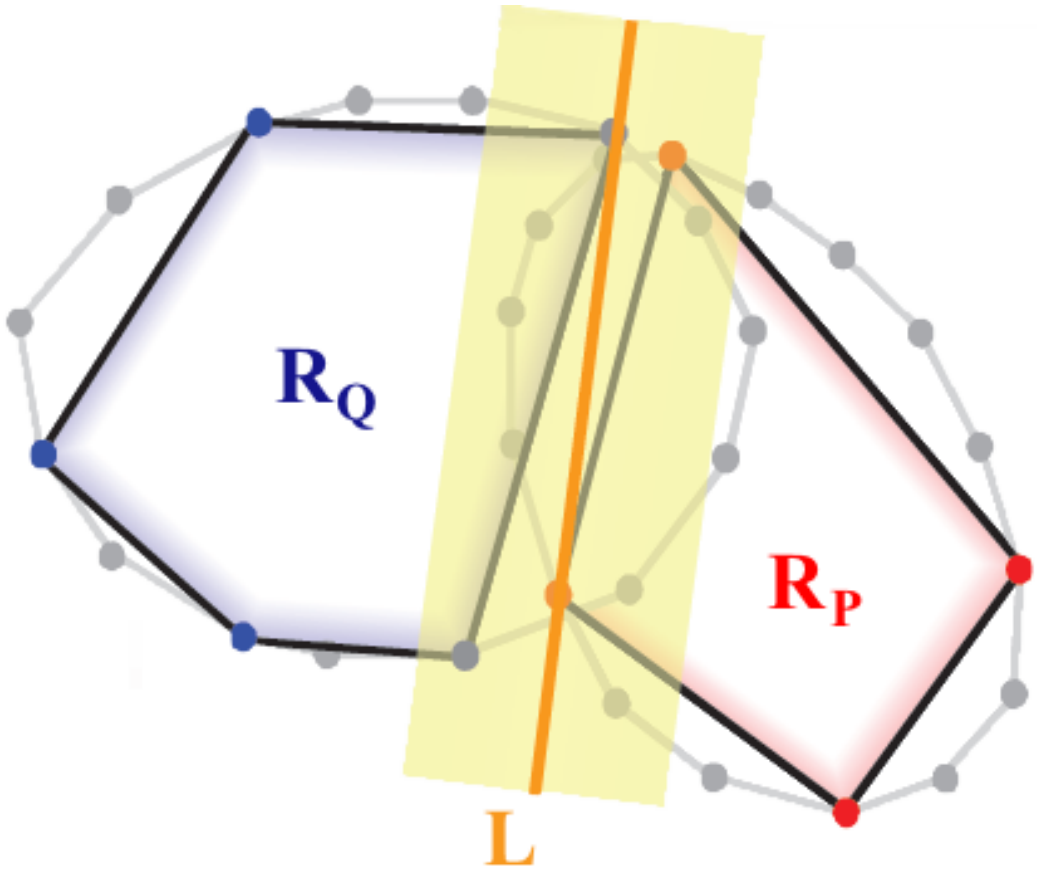
Example



Simplified versions of P and Q:
convex hulls R_P and R_Q of size $O(\sqrt{n})$

Convex polygonal intersection: Randomized algorithm

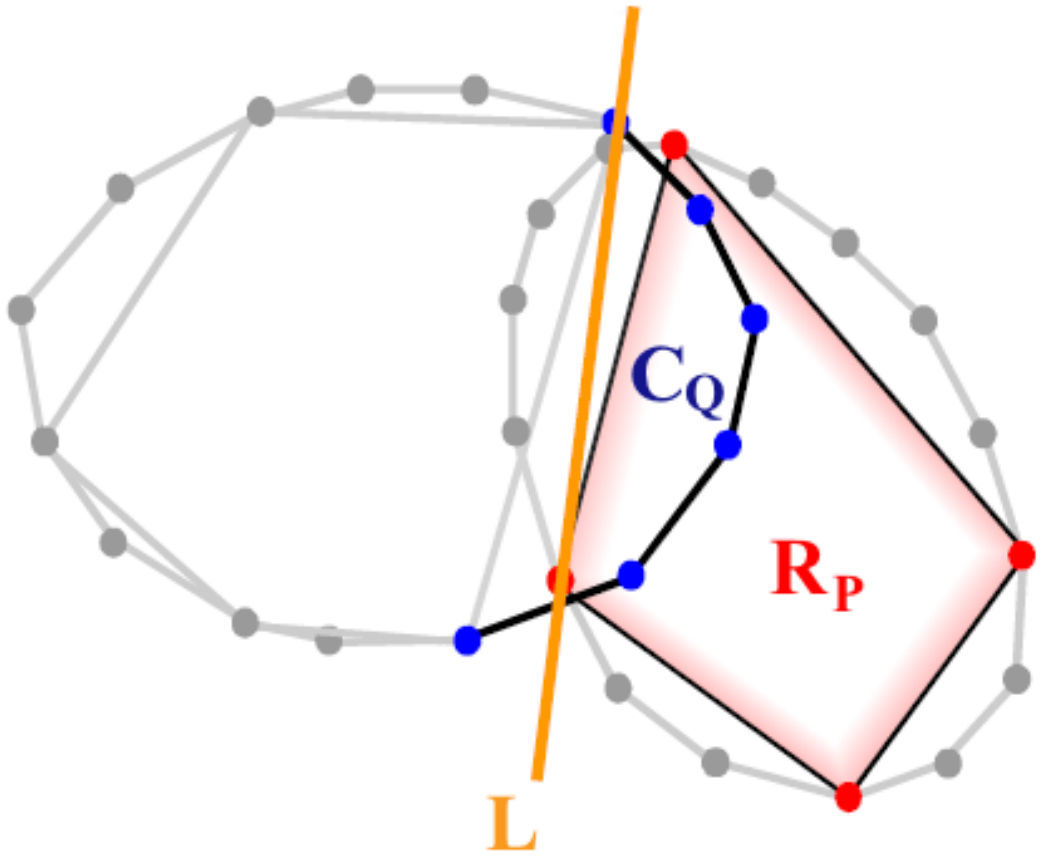
Example



$R_P \cap R_Q = \emptyset \implies$ compute bi-tangent separating line L

Convex polygonal intersection: Randomized algorithm

Example

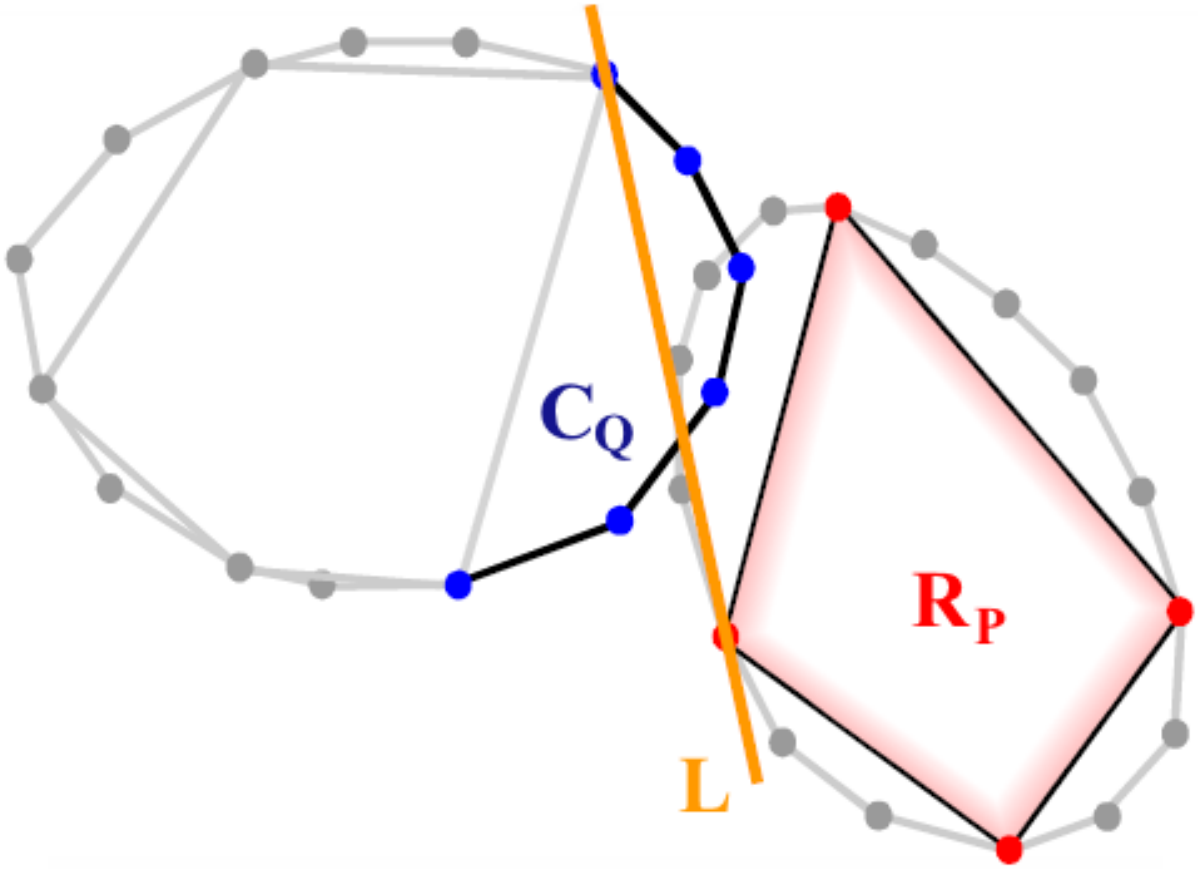


Compute C_Q, the part of Q to the other side of L.

Case 1: $R_P \cap C_Q \neq \emptyset \implies$ we are done

Convex polygonal intersection: Randomized algorithm

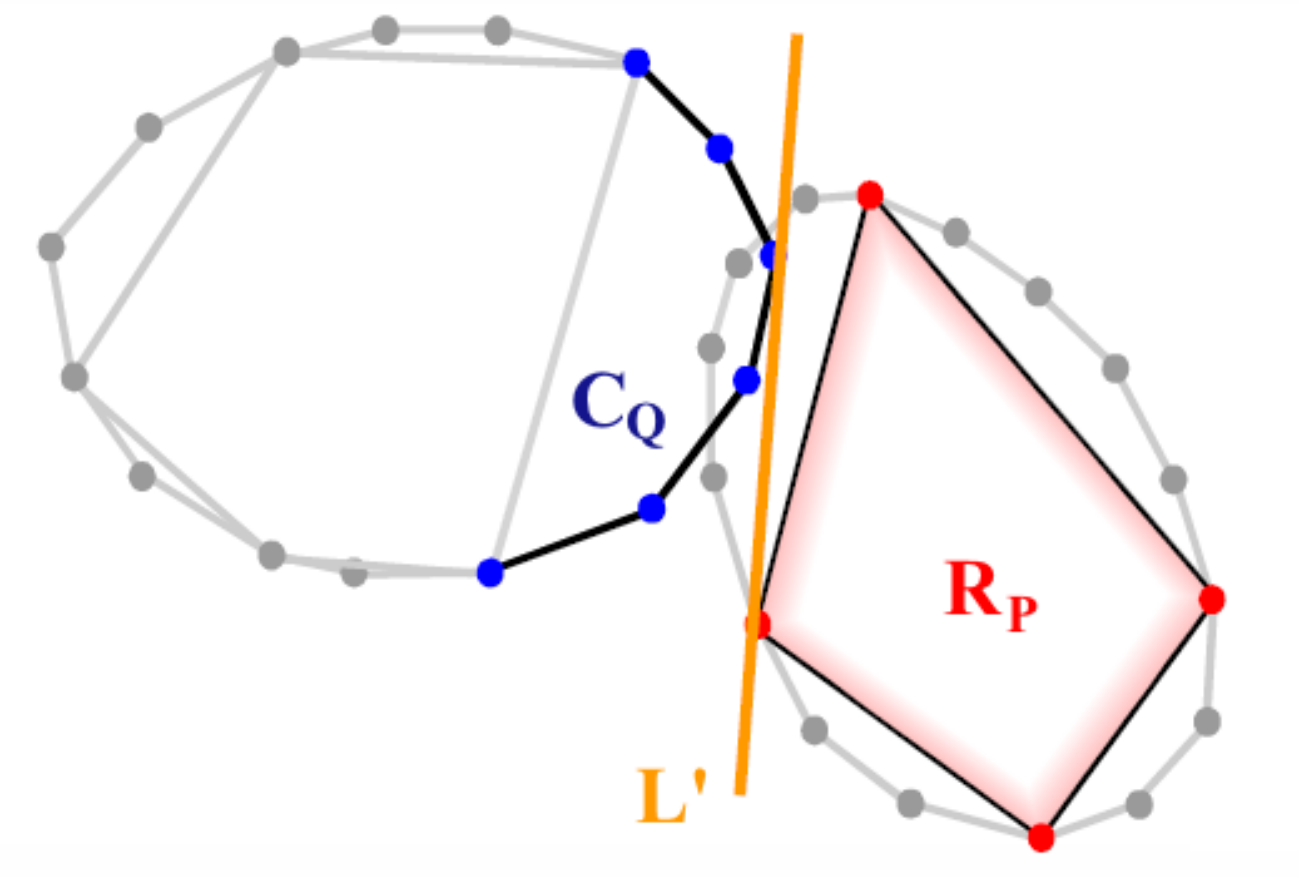
Example



Case 2: $R_P \cap C_Q = \emptyset$

Convex polygonal intersection: Randomized algorithm

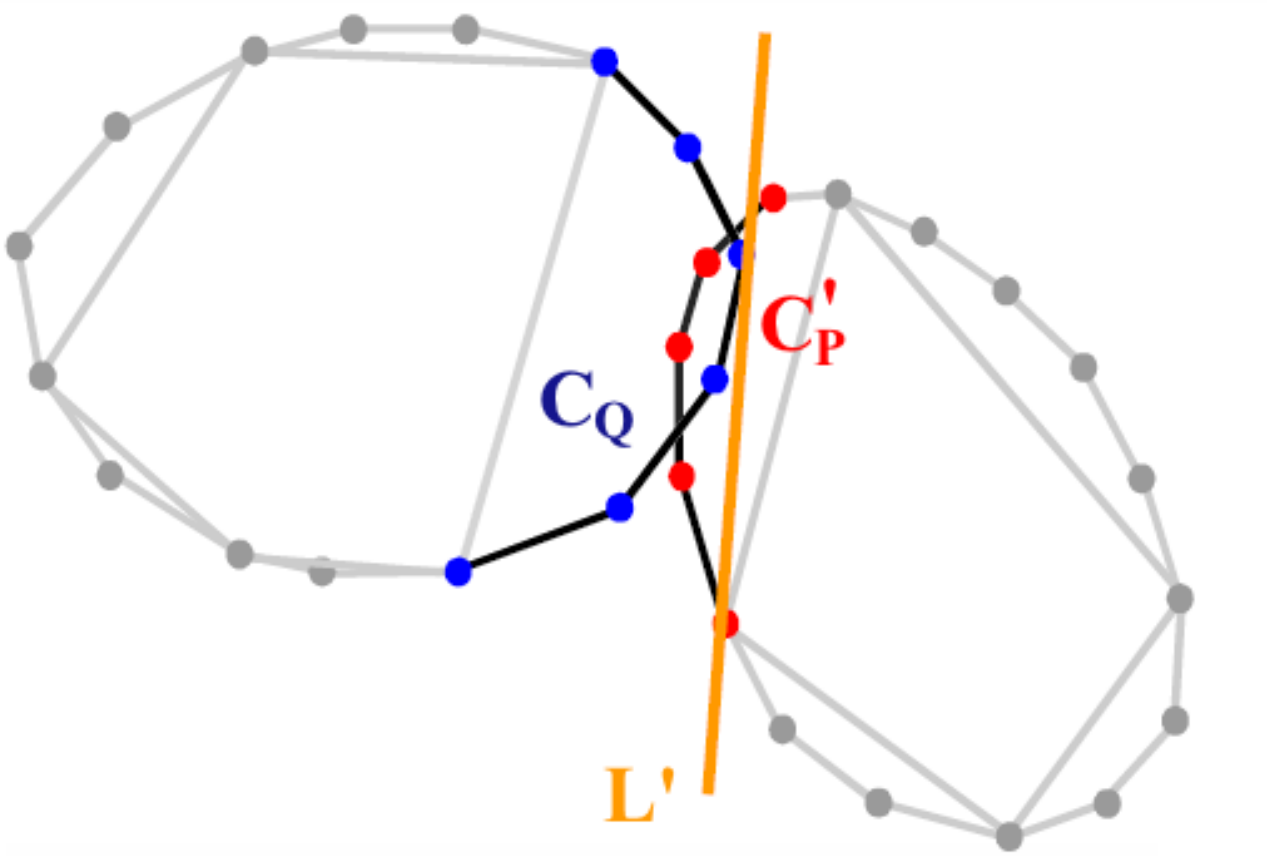
Example



$\Longrightarrow$ compute L', the separating line for R_P and C_Q

Convex polygonal intersection: Randomized algorithm

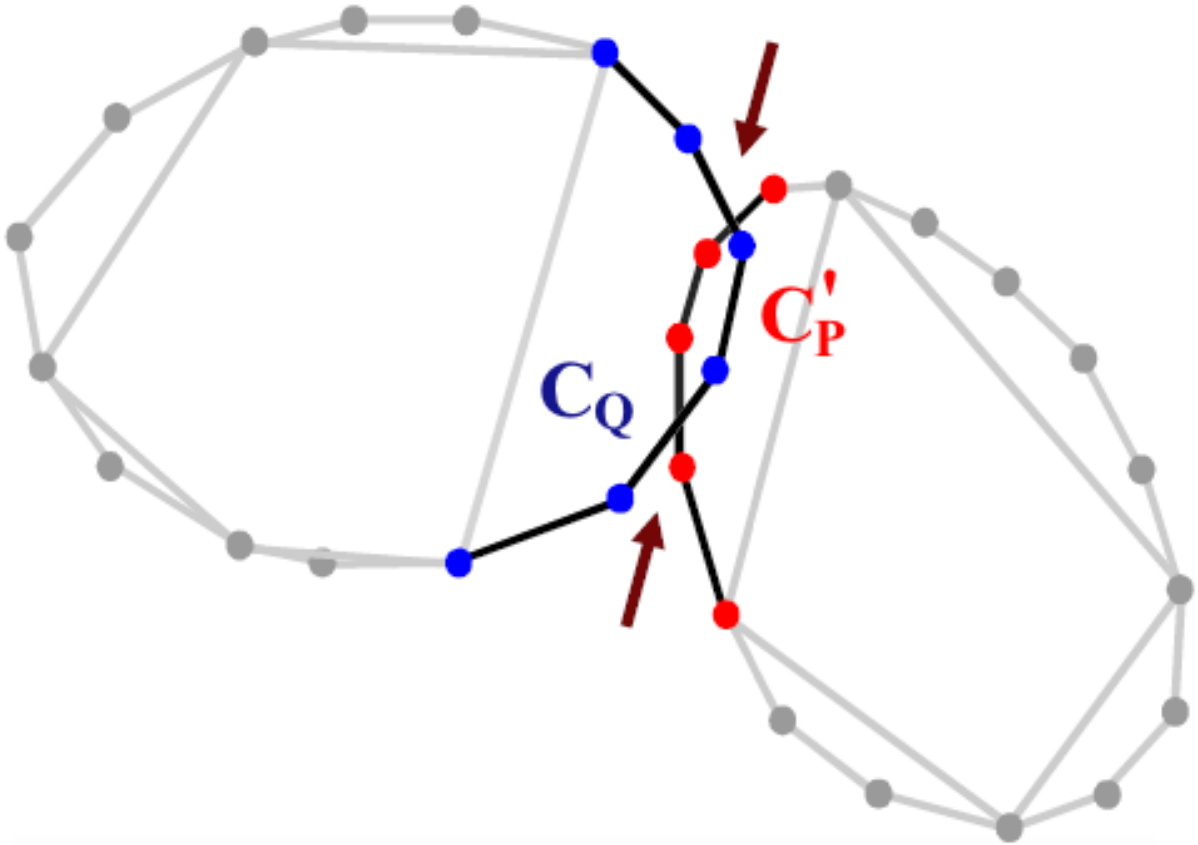
Example



Compute C'_P, the part of P to the other side of L'.
Test C'_P and C_Q for intersection

Convex polygonal intersection: Randomized algorithm

Example



Case 1: $C'_P \cap C_Q \neq \emptyset \implies$ we are done

Case 2: $C'_P \cap C_Q = \emptyset \implies$ re-run with P and Q exchanged

Convex polygonal intersection: Randomized algorithm

Q. Why don't we design a **recursive** algorithm instead?

Convex polygonal intersection: Randomized algorithm

Q. Why don't we design a **recursive** algorithm instead?
A. Because our model restricts us to **global** sampling only.

Convex polygonal intersection: Randomized algorithm

Q. Why don't we design a **recursive** algorithm instead?
A. Because our model restricts us to **global** sampling only.

- We can only sample efficiently in the main problem, but not in subproblems
- So we have to treat our intersection subproblems in a "classical" (linear) way

Convex polygonal intersection: Analysis

Theorem

To check whether two convex n-gons intersect can be done in $O(\sqrt{n})$ expected time, which is optimal.

Convex polygonal intersection: Analysis

Theorem

To check whether two convex n-gons intersect can be done in $O(\sqrt{n})$ expected time, which is optimal.

Proof

Some intuition:

- For intersection tests at each step we use Linear Programming, which works in $O(r)$ time for inputs of size r
- After sampling $\sqrt{n}$ vertices from each n-gon, we have $r = \sqrt{n}$
- If the intersection test for our simplified polygons R_P and R_Q fails, then the subsequent tests involving C_P, C_Q etc. will still process a sublinear number of vertices

Convex polygonal intersection: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Assume we sample r vertices in the first step

Convex polygonal intersection: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Assume we sample r vertices in the first step
- Running time is $O(r + |C_P| + |C'_P| + |C_Q| + |C'_Q|)$ (follows directly from the algorithm description, assuming we use LP for intersection tests to execute them in linear time)

Convex polygonal intersection: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Assume we sample r vertices in the first step
- Running time is $O(r + |C_P| + |C'_P| + |C_Q| + |C'_Q|)$ (follows directly from the algorithm description, assuming we use LP for intersection tests to execute them in linear time)
- **Key observation:** one can show that $\mathbb{E}|C_P| = O(n/r)$; the same is true for $\mathbb{E}|C'_P|$, $\mathbb{E}|C_Q|$ and $\mathbb{E}|C'_Q|$ (see the paper)

Convex polygonal intersection: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

- Assume we sample r vertices in the first step
- Running time is $O(r + |C_P| + |C'_P| + |C_Q| + |C'_Q|)$ (follows directly from the algorithm description, assuming we use LP for intersection tests to execute them in linear time)
- **Key observation:** one can show that $\mathbb{E}|C_P| = O(n/r)$; the same is true for $\mathbb{E}|C'_P|$, $\mathbb{E}|C_Q|$ and $\mathbb{E}|C'_Q|$ (see the paper)
- $\mathbb{E}$(Running time) becomes $O(r + n/r)$

Convex polygonal intersection: Analysis

Proof (Cont.)

Part I: Expected time is $O(\sqrt{n})$.

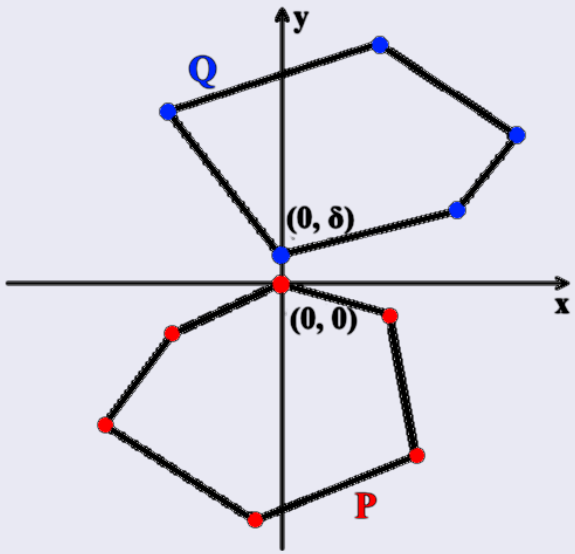
- Assume we sample r vertices in the first step
- Running time is $O(r + |C_P| + |C'_P| + |C_Q| + |C'_Q|)$ (follows directly from the algorithm description, assuming we use LP for intersection tests to execute them in linear time)
- **Key observation:** one can show that $\mathbb{E}|C_P| = O(n/r)$; the same is true for $\mathbb{E}|C'_P|$, $\mathbb{E}|C_Q|$ and $\mathbb{E}|C'_Q|$ (see the paper)
- $\mathbb{E}$(Running time) becomes $O(r + n/r)$
- Setting $r = \sqrt{n}$ makes it $O(\sqrt{n})$

Convex polygonal intersection: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- Yao's minimax principle $\Longrightarrow$ find a difficult distribution:



- Polygons P and Q lie to the opposite sides of the x-axis
- P's highest vertex is $p = (0, 0)$; Q's lowest vertex is $q = (0, \delta)$
- $P \cap Q \neq \emptyset \iff \delta = 0$

Convex polygonal intersection: Analysis

Proof (Cont.)

Part II: $O(\sqrt{n})$ is optimal.

- Any algorithm that detects intersection must have access to q to check that it lies in the origin
- The only operations allowed are:
  - random sampling of edges
  - edge-traversing via links
- The same argument as for the **successor search** problem yields the $\Omega(\sqrt{n})$ bound

Outline

3D: Convex polyhedral intersection

Problem

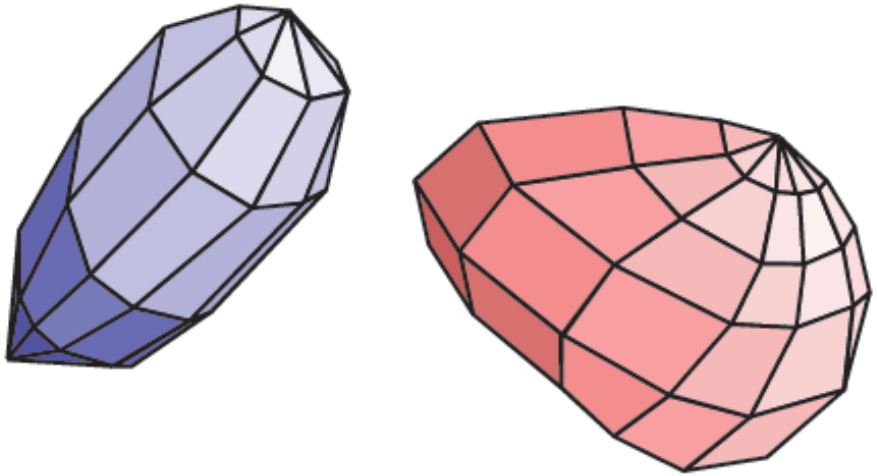
Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

Example:



Intersection: **NO**

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

Problem complexity depends on the underlying data structures:

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

Problem complexity depends on the underlying data structures:

1. General case
   $\rightarrow O(n)$ time (via Linear Programming)

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

Problem complexity depends on the underlying data structures:

1. General case
   $\rightarrow$ $O(n)$ time (via Linear Programming)

2. **Linked lists for edges/vertices/faces, no preprocessing** ✓
   $\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])

3D: Convex polyhedral intersection

Problem

Given two n-vertex convex polyhedra P and Q in $\mathbb{R}^3$, determine whether they intersect or not and, if they do, report one point in the intersection.

Problem complexity depends on the underlying data structures:

1. General case
   $\rightarrow$ $O(n)$ time (via Linear Programming)
2. **Linked lists for edges/vertices/faces, no preprocessing** ✓
   $\rightarrow$ expected $O(\sqrt{n})$ time (Chazelle et al. [2005])
3. P and Q have been preprocessed in $O(n)$
   $\rightarrow$ $O(\log n)$ time per query (Dobkin and Kirkpatrick [1990])

Convex polyhedral intersection: Idea of an algorithm

We can design an algorithm of the same structure as in 2D!

Convex polyhedral intersection: Idea of an algorithm

We can design an algorithm of the same structure as in 2D!

However, as we are now in 3D:

- We will compute a separating plane $\mathcal{L}$ instead of a separating line in the case there is no intersection
- Pieces cut off by $\mathcal{L}$ will be convex polyhedra, not polygons

Convex polyhedral intersection: Idea of an algorithm

We can design an algorithm of the same structure as in 2D!

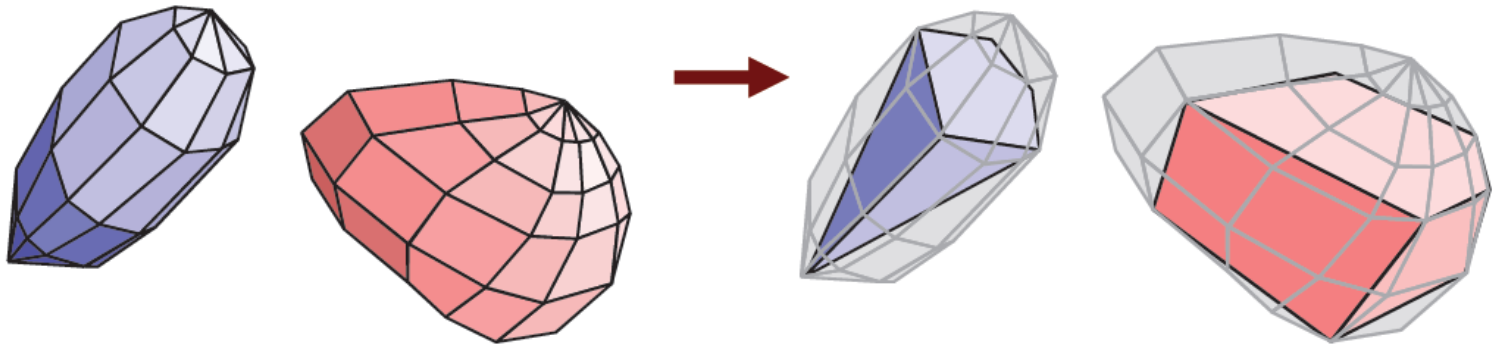
However, as we are now in 3D:

- We will compute a separating plane $\mathcal{L}$ instead of a separating line in the case there is no intersection
- Pieces cut off by $\mathcal{L}$ will be convex polyhedra, not polygons

Implementation requires solving some additional technical issues... (see the paper)

Convex polyhedral intersection: Randomized algorithm

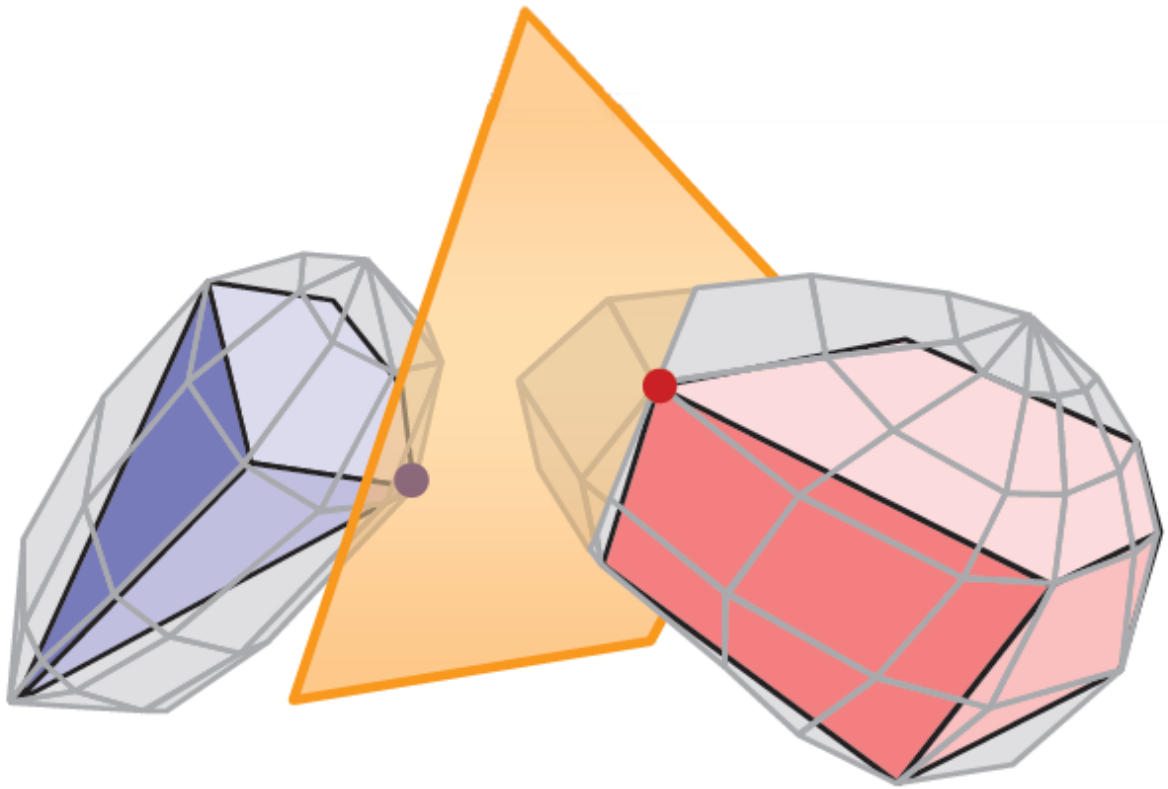
Example



Simplify polyhedra P and Q
using $O(\sqrt{n})$ randomly sampled vertices

Convex polyhedral intersection: Randomized algorithm

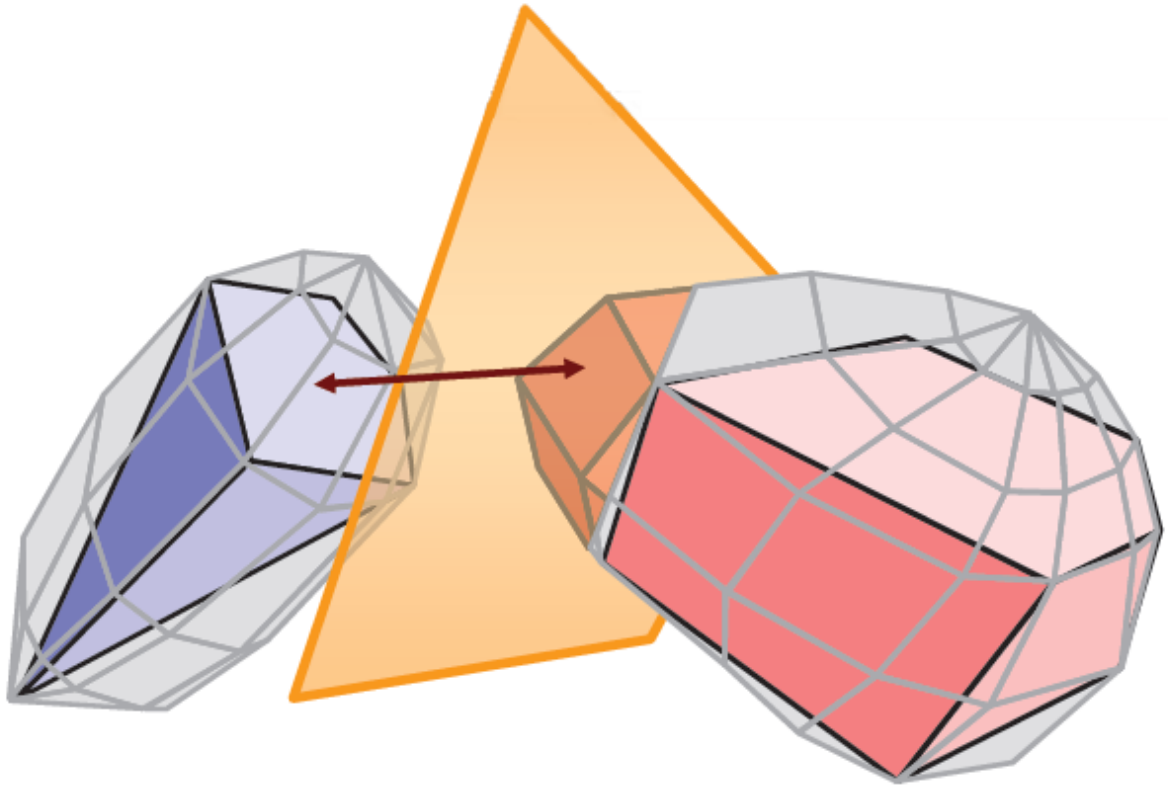
Example



Test intersection of samples (R_P and R_Q) via LP.
If they don't intersect, compute a separating plane L

Convex polyhedral intersection: Randomized algorithm

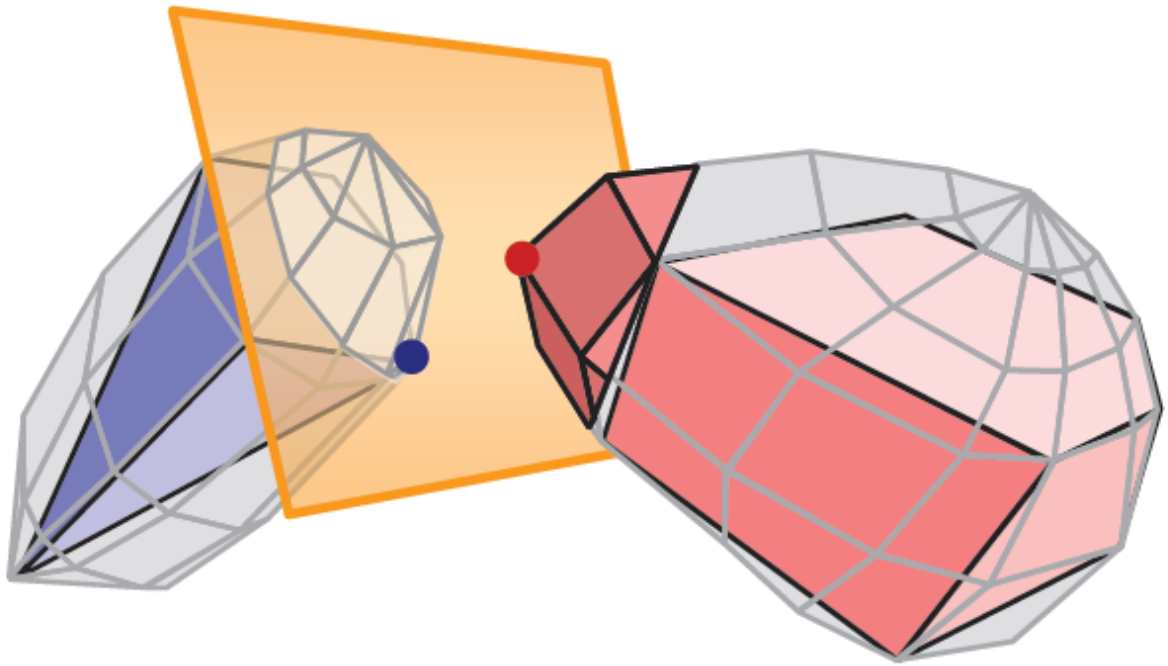
Example



Compute the part of P lying to the other side of L.
Test for intersection with R_Q

Convex polyhedral intersection: Randomized algorithm

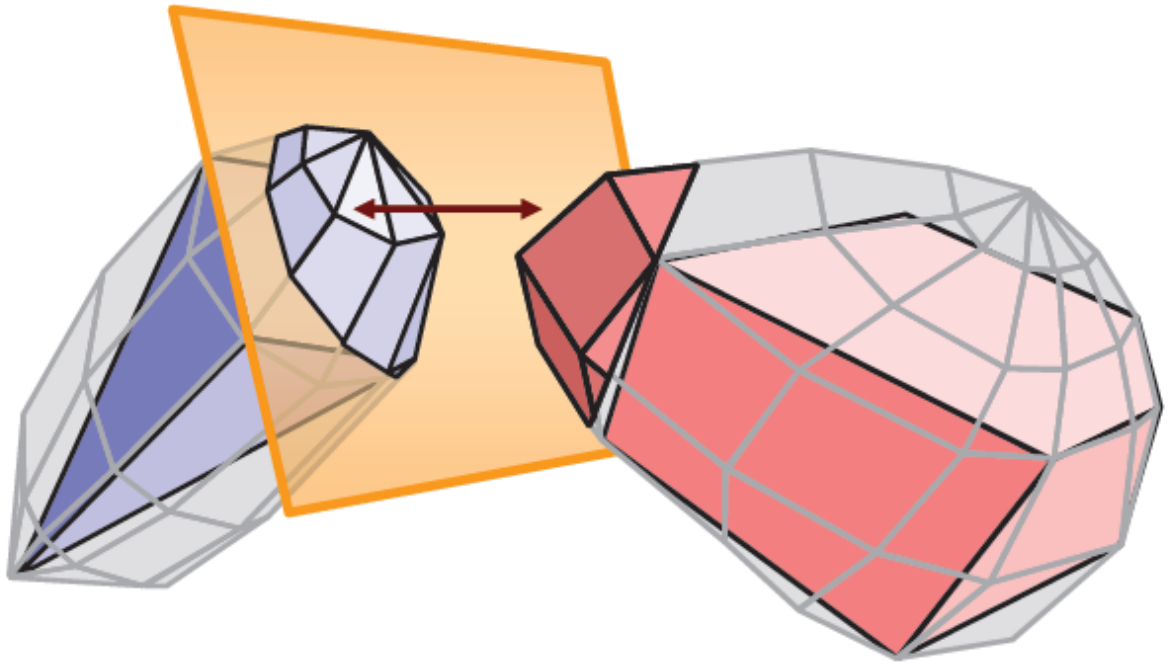
Example



If again no intersection is found, update the separating plane to L'

Convex polyhedral intersection: Randomized algorithm

Example



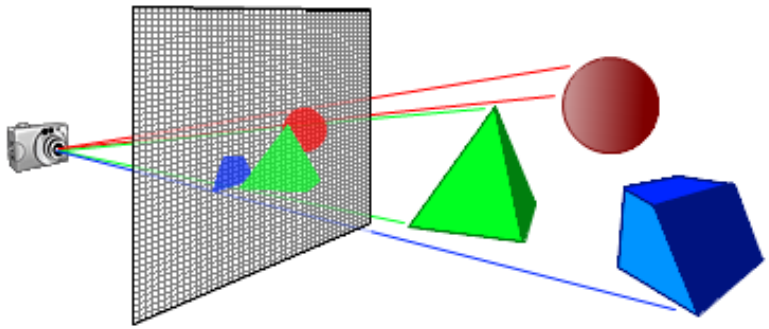
Compute the part of Q lying to the other side of L'.
If still no intersection, swap P and Q and repeat.

Outline

Ray shooting (casting)

Ray shooting (ray casting): the problem of ray-surface intersection detection with. Some computer graphics applications:

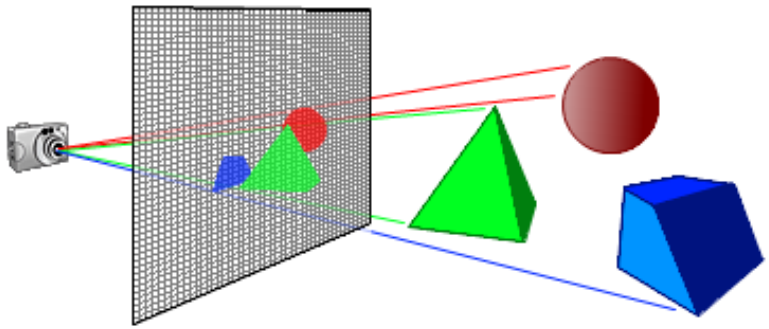
- Determining the first object intersected by a ray
- Hidden surface removal
- Ray tracing rendering



Ray shooting (casting)

Ray shooting (ray casting): the problem of ray-surface intersection detection with. Some computer graphics applications:

- Determining the first object intersected by a ray
- Hidden surface removal
- Ray tracing rendering



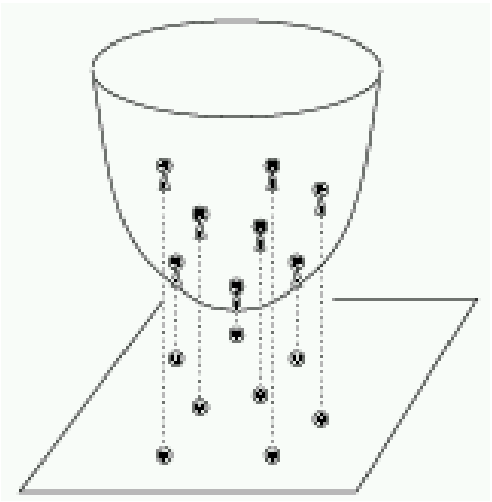
Can reuse the same algorithm!

- Rays are "very skinny" polyhedra
- We need only one intersection point

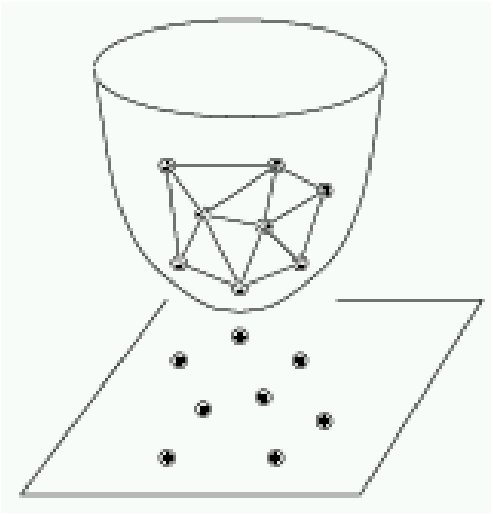
$\rightarrow$ Can compute ray-surface intersections in expected $O(\sqrt{n})$ time

Point location in Delaunay triangulations

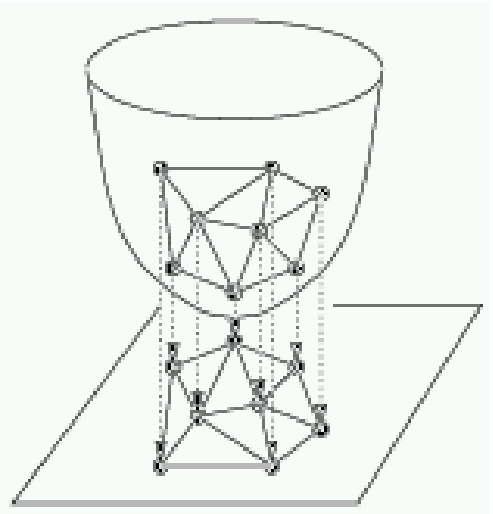
Relationship between **Delaunay triangulations** and **convex hulls**:



Project onto paraboloid
$(x, y, x^2 + y^2)$



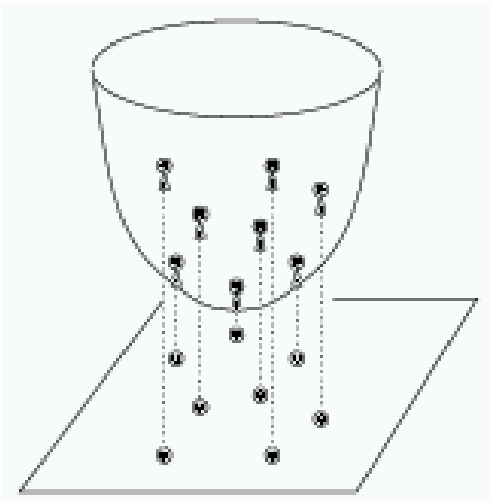
Compute convex hull



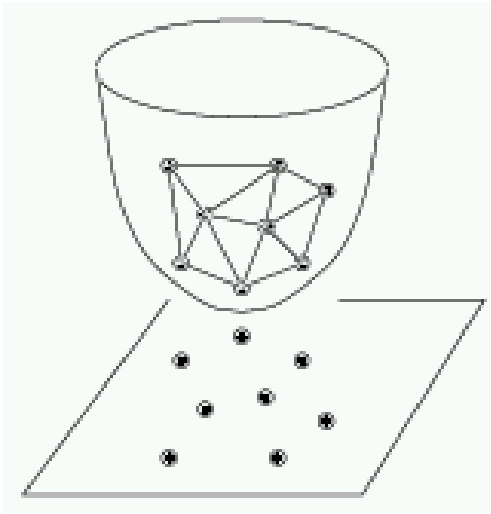
Project lower hull faces
back to the plane

Point location in Delaunay triangulations

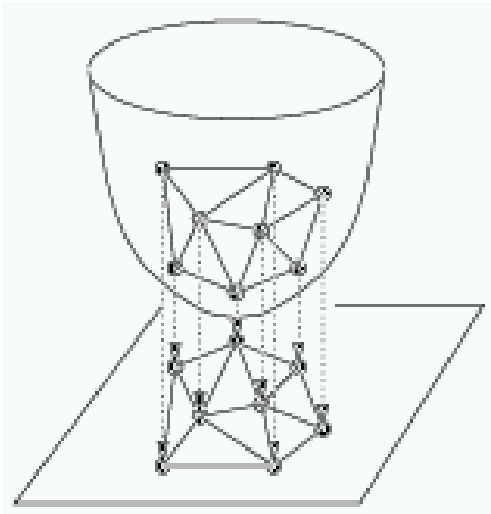
Relationship between **Delaunay triangulations** and **convex hulls**:



Project onto paraboloid $(x, y, x^2 + y^2)$



Compute convex hull



Project lower hull faces back to the plane

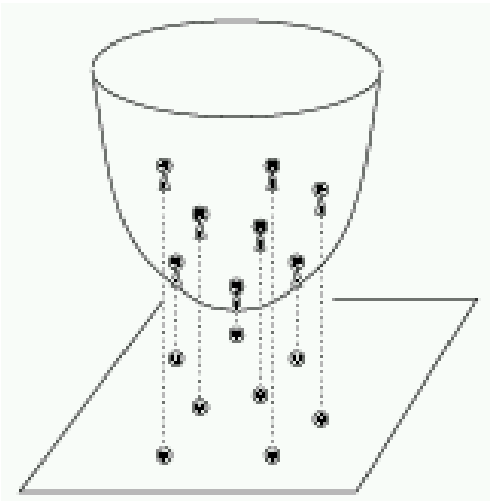
Point location in a Delaunay triangulation is equivalent to:

- Lifting the points to the paraboloid
- Shooting a ray towards their lower convex hull

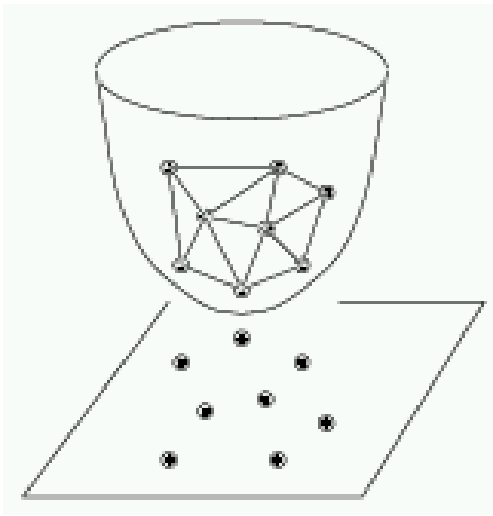
$\rightarrow$ can again be done in expected $O(\sqrt{n})$ time

Point location in Delaunay triangulations

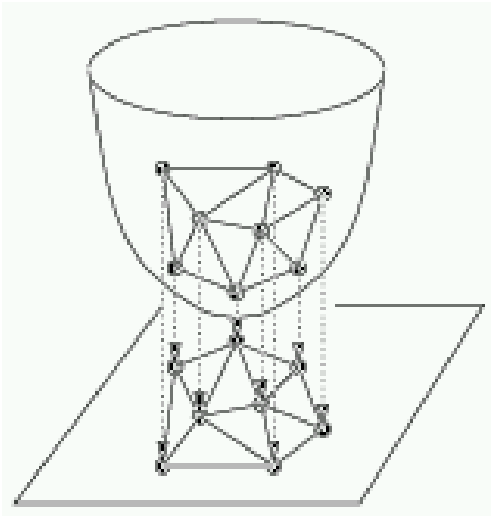
Relationship between **Delaunay triangulations** and **convex hulls**:



Project onto paraboloid
$(x, y, x^2 + y^2)$



Compute convex hull



Project lower hull faces back to the plane

Point location in a Delaunay triangulation is equivalent to:

- Lifting the points to the paraboloid
- Shooting a ray towards their lower convex hull

$\rightarrow$ can again be done in expected $O(\sqrt{n})$ time

Similar construction works for **Voronoi diagrams**

References I

B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, January 1987. ISSN 0004-5411. doi: 10.1145/7531.24036. URL http://doi.acm.org/10.1145/7531.24036.

Bernard Chazelle, Ding Liu, and Avner Magen. Sublinear geometric algorithms. In Artur Czumaj, S. Muthu Muthukrishnan, Ronitt Rubinfeld, and Christian Sohler, editors, *Sublinear Algorithms*, volume 05291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005. URL http://dblp.uni-trier.de/db/conf/dagstuhl/P5291.html#ChazelleLM05.

References II

David P. Dobkin and David G. Kirkpatrick. Determining the separation of preprocessed polyhedra: A unified approach. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, pages 400–413, New York, NY, USA, 1990. Springer-Verlag New York, Inc. ISBN 0-387-52826-1. URL http://dl.acm.org/citation.cfm?id=90397.91344.

Raimund Seidel. Linear programming and convex hulls made easy. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 211–215, New York, NY, USA, 1990. ACM. ISBN 0-89791-362-0. doi: 10.1145/98524.98570. URL http://doi.acm.org/10.1145/98524.98570.