Convex Hull Algorithms

Claudio Mirolo

Dip. di Scienze Matematiche, Informatiche e Fisiche
Università di Udine, via delle Scienze 206 – Udine

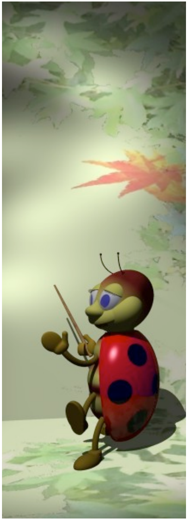
claudio.mirolo@uniud.it

Computational Geometry

www.dimi.uniud.it/claudio

Outline

1. Incremental algorithm
   - degeneracies
   - correctness
   - computational costs
2. Divide-et-impera algorithm
   - recursive approach
   - corrrectness
   - computational costs
3. Randomized algorithm
   - conflict graph
   - corrrectness
   - computational costs



Convex hull

- Given a set P of n points in the plane (space)
- “Smaller” convex region containing all points in P
- Region = convex polygon (polyhedron)

Convex hull

- Given a set P of n points in the plane (space)
- “Smaller” convex region containing all points in P
- Region = convex polygon (polyhedron)

Convex hull

- Given a set P of n points in the plane (space)
- “Smaller” convex region containing all points in P
- Region = convex polygon (polyhedron)

Convex hull: Motivations

- “Classical” problem in the field
- Easy to state…
- …and easy to solve, even efficiently
- Amenable to application of different general algorithmic approaches to CG problems

Convex hull: Motivations

- “Classical” problem in the field
- Easy to state…
- …and easy to solve, even efficiently
- Amenable to application of different general algorithmic approaches to CG problems

Convex hull: Motivations

- “Classical” problem in the field
- Easy to state…
- …and easy to solve, even efficiently
- Amenable to application of different general algorithmic approaches to CG problems

Convex hull: Motivations

- “Classical” problem in the field
- Easy to state…
- …and easy to solve, even efficiently
- Amenable to application of different general algorithmic approaches to CG problems

From definition to operational definition

- Convex hull = $\bigcap \{C : P \subset C\}$, C convex region
- Convex hull = $\bigcap \{H : P \subset H\}$, H halfplane
- Convex hull = $\bigcap \{H_{pq} : p, q \in P \wedge P \subset H_{pq}\}$
- Directed segment *pq*: halfplane interior on its left
- Minimal (and finite!) set of halfplanes

From definition to operational definition

- Convex hull = $\bigcap \{C : P \subset C\}$, C convex region
- Convex hull = $\bigcap \{H : P \subset H\}$, H halfplane
- Convex hull = $\bigcap \{H_{pq} : p, q \in P \wedge P \subset H_{pq}\}$
- Directed segment *pq*: halfplane interior on its left
- Minimal (and finite!) set of halfplanes

From definition to operational definition

- Convex hull = $\bigcap \{C : P \subset C\}$, $\quad C$ convex region
- Convex hull = $\bigcap \{H : P \subset H\}$, $\quad H$ halfplane
- Convex hull = $\bigcap \{H_{pq} : p, q \in P \wedge P \subset H_{pq}\}$
- Directed segment pq: halfplane interior on its left
- Minimal (and finite!) set of halfplanes

From definition to operational definition

- Convex hull = $\bigcap \{C : P \subset C\}$, C convex region
- Convex hull = $\bigcap \{H : P \subset H\}$, H halfplane
- Convex hull = $\bigcap \{H_{pq} : p, q \in P \wedge P \subset H_{pq}\}$
- Directed segment *pq*: halfplane interior on its left
- Minimal (and finite!) set of halfplanes

From definition to operational definition

- Convex hull = $\bigcap \{C : P \subset C\}$, $\quad C$ convex region
- Convex hull = $\bigcap \{H : P \subset H\}$, $\quad H$ halfplane
- Convex hull = $\bigcap \{H_{pq} : p, q \in P \wedge P \subset H_{pq}\}$
- Directed segment pq: halfplane interior on its left
- Minimal (and finite!) set of halfplanes

From operational definition to “brute force” algorithm

- pq bounds the convex hull $\Leftrightarrow P \setminus \{p, q\}$ on the left of pq
- `CGAL::left_turn( p1, p2, q )` $\rightarrow$ *code*
- When applied to a (small) set of *random* points it seems to work properly
- But what about the algorithm’s robustness?
- Indeed, a more accurate analysis is usually required

From operational definition to “brute force” algorithm

- pq bounds the convex hull $\Leftrightarrow P \setminus \{p, q\}$ on the left of pq
- `CGAL::left_turn( p1, p2, q )` $\rightarrow$ *code*
- When applied to a (small) set of *random* points it seems to work properly
- But what about the algorithm’s robustness?
- Indeed, a more accurate analysis is usually required

From operational definition to “brute force” algorithm

- pq bounds the convex hull $\Leftrightarrow P \setminus \{p, q\}$ on the left of pq
- `CGAL::left_turn( p1, p2, q )` $\rightarrow$ *code*
- When applied to a (small) set of *random* points
  it seems to work properly
- But what about the algorithm’s robustness?
- Indeed, a more accurate analysis is usually required

From operational definition to “brute force” algorithm

- pq bounds the convex hull $\Leftrightarrow P \setminus \{p, q\}$ on the left of pq
- `CGAL::left_turn( p1, p2, q )` $\rightarrow$ *code*
- When applied to a (small) set of *random* points it seems to work properly
- But what about the algorithm’s robustness?
- Indeed, a more accurate analysis is usually required

From operational definition to “brute force” algorithm

- *pq* bounds the convex hull $\Leftrightarrow P \setminus \{p, q\}$ on the left of *pq*
- `CGAL::left_turn( p1, p2, q )` $\rightarrow$ *code*
- When applied to a (small) set of *random* points
  it seems to work properly
- But what about the algorithm’s robustness?
- Indeed, a more accurate analysis is usually required

Degeneracies

- In a first stage it can be useful to ignore possible degenaracies
- Assuming some "general configuration" (e.g. no three colinear points)
- How to deal with colinear points on the boundary of the convex hull?
- `! CGAL::right_turn( p1, p2, q )` (?) → *code*
- Structural integrity for small perturbations (e.g. floating-point inaccuracies). . .

Degeneracies

- In a first stage it can be useful to ignore possible degenaracies
- Assuming some “general configuration” (e.g. no three colinear points)
- How to deal with colinear points on the boundary of the convex hull?
- `! CGAL::right_turn( p1, p2, q )` (?) $\rightarrow$ *code*
- Structural integrity for small perturbations (e.g. floating-point inaccuracies)…

Degeneracies

- In a first stage it can be useful to ignore possible degenaracies
- Assuming some "general configuration" (e.g. no three colinear points)
- How to deal with colinear points on the boundary of the convex hull?
- `! CGAL::right_turn( p1, p2, q )` (?) $\rightarrow$ *code*
- Structural integrity for small perturbations (e.g. floating-point inaccuracies). . .

Degeneracies

- In a first stage it can be useful to ignore possible degenaracies
- Assuming some “general configuration” (e.g. no three colinear points)
- How to deal with colinear points on the boundary of the convex hull?
- `! CGAL::right_turn( p1, p2, q )` (?) $\rightarrow$ *code*
- Structural integrity for small perturbations (e.g. floating-point inaccuracies)…

Degeneracies

- In a first stage it can be useful to ignore possible degenaracies
- Assuming some “general configuration” (e.g. no three colinear points)
- How to deal with colinear points on the boundary of the convex hull?
- `! CGAL::right_turn( p1, p2, q )` (?) $\rightarrow$ *code*
- Structural integrity for small perturbations (e.g. floating-point inaccuracies)…

Convex hull representation

- What is, exactly, the intended output representation?
- Topological/relational information
- Polygon sides in counterclockwise order
- Related structural integrity issues...
- However: the output adds only relational information to the input! (selection + coupling of input points)

Convex hull representation

- What is, exactly, the intended output representation?
- Topological/relational information
- Polygon sides in counterclockwise order
- Related structural integrity issues. . .
- However: the output adds only relational information to the input! (selection + coupling of input points)

Convex hull representation

- What is, exactly, the intended output representation?
- Topological/relational information
- Polygon sides in counterclockwise order
- Related structural integrity issues...
- However: the output adds only relational information to the input! (selection + coupling of input points)

Convex hull representation

- What is, exactly, the intended output representation?
- Topological/relational information
- Polygon sides in counterclockwise order
- Related structural integrity issues. . .
- However: the output adds only relational information to the input! (selection + coupling of input points)

Convex hull representation

- What is, exactly, the intended output representation?
- Topological/relational information
- Polygon sides in counterclockwise order
- Related structural integrity issues...
- However: the output adds only relational information to the input! (selection + coupling of input points)

Analysis of computational costs

- Identification of convex hull sides (*brute force*):
- Incidence between sides (*brute force*):
- Overall:

Analysis of computational costs

- Identification of convex hull sides (*brute force*): $O(\ n^3\ )$
- Incidence between sides (*brute force*):
- Overall:

Analysis of computational costs

- Identification of convex hull sides (*brute force*): $O(\ n^3\ )$
- Incidence between sides (*brute force*): $O(\ n^2\ )$
- Overall: $O(\ n^3\ )$

Looking for improvements. . .

- Can this approach be improved?
- *Walking* along the convex hull boundary
- Starting vertex. . .
- Jarvis' March (1973)

Looking for improvements. . .

- Can this approach be improved?
- *Walking* along the convex hull boundary
- Starting vertex. . .
- Jarvis' March (1973)

Looking for improvements. . .

- Can this approach be improved?
- *Walking* along the convex hull boundary
- Starting vertex. . .
- Jarvis' March (1973)

Looking for improvements. . .

- Can this approach be improved?
- *Walking* along the convex hull boundary
- Starting vertex. . .
- Jarvis' March (1973)

Jarvis' March

- h-sided convex hull:
- Worst case:
- However, it may be efficient if $h \ll n$
- $\rightarrow$ *code*

Jarvis' March

- h-sided convex hull: $O(\ hn\ )$
- Worst case:
- However, it may be efficient if $h \ll n$
- $\rightarrow$ *code*

Jarvis' March

- h-sided convex hull: $O(\ hn\ )$
- Worst case: $O(\ n^2\ )$
- However, it may be efficient if $h \ll n$
- $\rightarrow$ *code*

Jarvis' March

- h-sided convex hull: $O(\ hn\ )$
- Worst case: $O(\ n^2\ )$
- However, it may be efficient if $h \ll n$
- $\rightarrow$ *code*

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered…
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs:

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered...
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs:

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered. . .
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs:

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered...
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs:

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered…
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs:

Is a better solution conceivable?

- How much room to improve performances?
- Convex hull sides must be ordered. . .
- Let us consider the following point set:

$$P = \{(x_i, x_i^2) : 1 \leq i \leq n \wedge x_i \in \mathbb{R}\}$$

- P's convex hull sorts $\{x_i : 1 \leq i \leq n\} \subset \mathbb{R}$!
- Lower bound to computational costs: $\Omega(\ n \log n\ )$

Convex hull: Instrinsic complexity

- Jarvis' March is then interesting for $h = O(\log n)$
- But what about the (expected) number of sides?
- Clearly, it depends on point distribution
- See, e.g. (students' projects):
  - Har-Peled (1997, 2011)
  - Golin & Sedgewick (1988)

Convex hull: Instrinsic complexity

- Jarvis' March is then interesting for $h = O(\log n)$
- But what about the (expected) number of sides?
- Clearly, it depends on point distribution
- See, e.g. (students' projects):
  - Har-Peled (1997, 2011)
  - Golin & Sedgewick (1988)

Convex hull: Instrinsic complexity

- Jarvis' March is then interesting for $h = O(\log n)$
- But what about the (expected) number of sides?
- Clearly, it depends on point distribution
- See, e.g. (students' projects):
  - Har-Peled (1997, 2011)
  - Golin & Sedgewick (1988)

Convex hull: Instrinsic complexity

- Jarvis' March is then interesting for $h = O(\log n)$
- But what about the (expected) number of sides?
- Clearly, it depends on point distribution
- See, e.g. (students' projects):
  - Har-Peled (1997, 2011)
  - Golin & Sedgewick (1988)

Har-Peled (1997, 2011)

- Uniform point distribution in a disc: $O(\sqrt[3]{n})$
- Uniform distribution in a square/triangle: $O(\log n)$
- Uniform distribution in a k-sided polygon: $O(k \log n)$
- Jarvis' March Ok for uniform distributions in rectangles/triangles?
- Or we could do something better under similar assumptions?

Har-Peled (1997, 2011)

- Uniform point distribution in a disc: $O(\sqrt[3]{n})$
- Uniform distribution in a square/triangle: $O(\log n)$
- Uniform distribution in a k-sided polygon: $O(k \log n)$
- Jarvis' March Ok for uniform distributions in rectangles/triangles?
- Or we could do something better under similar assumptions?

Har-Peled (1997, 2011)

- Uniform point distribution in a disc: $O(\sqrt[3]{n})$
- Uniform distribution in a square/triangle: $O(\log n)$
- Uniform distribution in a *k*-sided polygon: $O(k \log n)$
- Jarvis' March Ok for uniform distributions in rectangles/triangles?
- Or we could do something better under similar assumptions?

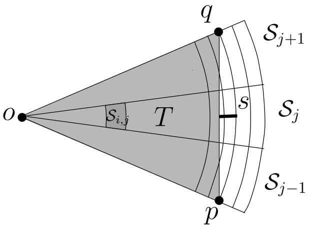
Har-Peled (1997, 2011)

- Uniform point distribution in a disc: $O(\sqrt[3]{n})$
- Uniform distribution in a square/triangle: $O(\log n)$
- Uniform distribution in a k-sided polygon: $O(k \log n)$
- Jarvis' March Ok for uniform distributions in rectangles/triangles?
- Or we could do something better under similar assumptions?

Har-Peled (1997, 2011)

- Uniform point distribution in a disc: $O(\sqrt[3]{n})$
- Uniform distribution in a square/triangle: $O(\log n)$
- Uniform distribution in a k-sided polygon: $O(k \log n)$
- Jarvis' March Ok for uniform distributions in rectangles/triangles?
- Or we could do something better under similar assumptions?

Har-Peled (2011)



Theorem 2.3 *The expected number of vertices of the convex hull of n points, chosen uniformly and independently from the unit disk, is $O(n^{1/3})$.*

Proof: We claim that the expected area of the convex hull of n points, chosen uniformly and independently from the unit disk, is at least $\pi - O\left(n^{-2/3}\right)$.

Indeed, let D denote the unit disk, and assume without loss of generality, that $n = m^3$, where m is a positive integer. Partition D into m sectors, $\mathcal{S}_1, \ldots, \mathcal{S}_m$, by placing m equally spaced points on the boundary of D and connecting them to the origin. Let $D_1, \ldots, D_{m^2}$ denote the m^2 disks centered at the origin, such that (i) $D_1 = D$, and (ii) $Area(D_{i-1}) - Area(D_i) = \pi/m^2$, for $i = 2, \ldots, m^2$. Let r_i denote the radius of D_i, for $i = 1, \ldots, m^2$.

Let $S_{i,j} = (D_i \setminus D_{i+1}) \cap \mathcal{S}_j$, and $S_{m^2,j} = D_{m^2} \cap \mathcal{S}_j$, for $i = 1, \ldots, m^2 - 1$, $j = 1, \ldots, m$. The set $S_{i,j}$ is called the i-th *tile* of the sector $\mathcal{S}_j$, and its area is π/n, for $i = 1, \ldots, m^2$, $j = 1, \ldots, m$.

Golin & Sedgewick (1988)

- Uniform distribution of n points in a square
- Inner construction (quadrilateral) to remove most points from further consideration in $O(\, n \,)$
- Expected number of remaining candidates: $O(\, \sqrt{n} \,)$
- $h = O(\, \sqrt{n} \,) \Rightarrow$ Jarvis' March in $O(\, \sqrt{n} \cdot \sqrt{n} \,) = O(\, n \,)$
- We certainly can't do better than this!

Golin & Sedgewick (1988)

- Uniform distribution of n points in a square
- Inner construction (quadrilateral) to remove most points from further consideration in $O(\ n\ )$
- Expected number of remaining candidates: $O(\ \sqrt{n}\ )$
- $h = O(\ \sqrt{n}\ ) \Rightarrow$ Jarvis' March in $O(\ \sqrt{n} \cdot \sqrt{n}\ ) = O(\ n\ )$
- We certainly can't do better than this!

Golin & Sedgewick (1988)

- Uniform distribution of n points in a square
- Inner construction (quadrilateral) to remove most points from further consideration in $O(\ n\ )$
- Expected number of remaining candidates: $O(\ \sqrt{n}\ )$
- $h = O(\ \sqrt{n}\ ) \Rightarrow$ Jarvis' March in $O(\ \sqrt{n} \cdot \sqrt{n}\ ) = O(\ n\ )$
- We certainly can't do better than this!

Golin & Sedgewick (1988)

- Uniform distribution of n points in a square
- Inner construction (quadrilateral) to remove most points from further consideration in $O(\ n\ )$
- Expected number of remaining candidates: $O(\ \sqrt{n}\ )$
- $h = O(\ \sqrt{n}\ ) \Rightarrow$ Jarvis' March in $O(\ \sqrt{n} \cdot \sqrt{n}\ ) = O(\ n\ )$
- We certainly can't do better than this!

Golin & Sedgewick (1988)

- Uniform distribution of n points in a square
- Inner construction (quadrilateral) to remove most points from further consideration in $O(\ n\ )$
- Expected number of remaining candidates: $O(\ \sqrt{n}\ )$
- $h = O(\ \sqrt{n}\ ) \Rightarrow$ Jarvis' March in $O(\ \sqrt{n} \cdot \sqrt{n}\ ) = O(\ n\ )$
- We certainly can't do better than this!

Questions still to be addressed. . .

- Can the optimal $O(\ n \log n\ )$ trend be achieved in general?
- In the *worst* case?
- In the *expected case*,
  but regardless of assumptions on the point distribution?

Questions still to be addressed. . .

- Can the optimal $O(\ n \log n\ )$ trend be achieved in general?
- In the *worst* case?
- In the *expected case*,
  but regardless of assumptions on the point distribution?

Questions still to be addressed. . .

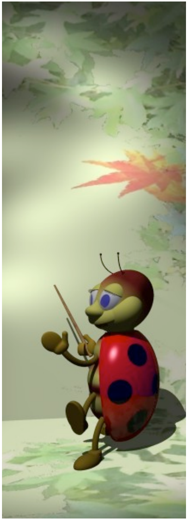
- Can the optimal $O(\, n \log n \,)$ trend be achieved in general?

- In the *worst* case?

- In the *expected case*,
  but regardless of assumptions on the point distribution?

Outline

1. Incremental algorithm
   - degeneracies
   - correctness
   - computational costs
2. Divide-et-impera algorithm
   - recursive approach
   - corrrectness
   - computational costs
3. Randomized algorithm
   - conflict graph
   - corrrectness
   - computational costs



Graham's scan

- Lower hull first
- Adding points one by one, *sorted left-to-right*
- Lower hull is updated after each addition

Graham's scan

- Lower hull first
- Adding points one by one, *sorted left-to-right*
- Lower hull is updated after each addition

Graham's scan

- Lower hull first
- Adding points one by one, *sorted left-to-right*
- Lower hull is updated after each addition

Scan step

- From *lower_hull*($\{\, p_1,\ p_2,\ p_3,\ \ldots,\ p_{i-1} \,\}$) ...
- ... to *lower_hull*($\{\, p_1,\ p_2,\ p_3,\ \ldots,\ p_{i-1},\ p_i \,\}$)
- Basic idea: Animation

Scan step

- From *lower_hull*($\{ p_1, p_2, p_3, \ldots, p_{i-1} \}$) . . .
- . . . to *lower_hull*($\{ p_1, p_2, p_3, \ldots, p_{i-1}, p_i \}$)
- Basic idea: Animation

Scan step

- From *lower_hull*($\{ p_1, p_2, p_3, \ldots, p_{i-1} \}$) ...
- ... to *lower_hull*($\{ p_1, p_2, p_3, \ldots, p_{i-1}, p_i \}$)
- Basic idea: Animation

Lower hull

- After the i-th step, p_1 and p_i belong to the lower hull
- p_1 and p_n belong to both the lower and upper hull
- Lower hull construction: Code

Lower hull

- After the i-th step, p_1 and p_i belong to the lower hull
- p_1 and p_n belong to both the lower and upper hull
- Lower hull construction: Code

Lower hull

- After the i-th step, p_1 and p_i belong to the lower hull
- p_1 and p_n belong to both the lower and upper hull
- Lower hull construction: Code

Upper hull

- Upper hull: similar processing, by symmetry
- E.g., adding points in backward order
- Upper hull construction: Code

Upper hull

- Upper hull: similar processing, by symmetry
- E.g., adding points in backward order
- Upper hull construction: Code

Upper hull

- Upper hull: similar processing, by symmetry
- E.g., adding points in backward order
- Upper hull construction: Code

Degeneracies: Vertically aligned points

- *Lexicographic order*
- Geometric interpretation of lexicographic order
- “Symbolic perturbation”

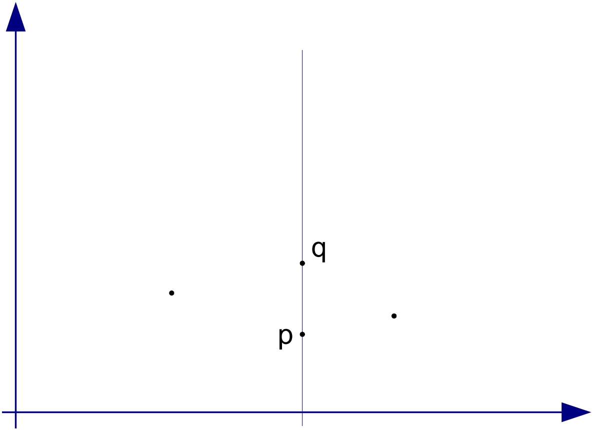
Degeneracies: Vertically aligned points

- *Lexicographic order*
- Geometric interpretation of lexicographic order
- "Symbolic perturbation"

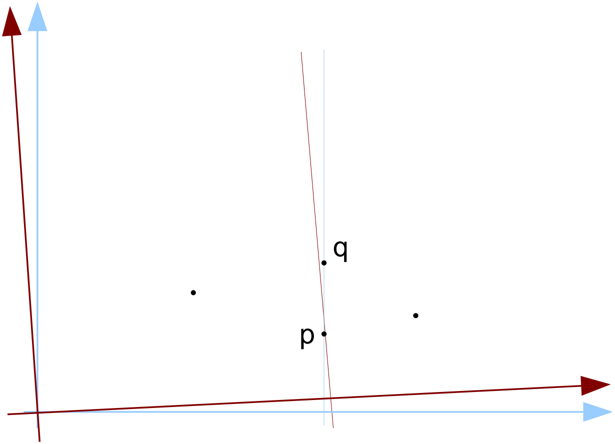
Degeneracies: Vertically aligned points

- *Lexicographic order*
- Geometric interpretation of lexicographic order
- “Symbolic perturbation”

Geometric interpretation



Geometric interpretation



Degeneracies: Collinear hull vertices

- *Strict* left turn
- Which is the result?
- Animation (vertical/horizontal point alignments)
- Code details. . .

Degeneracies: Collinear hull vertices

- *Strict* left turn
- Which is the result?
- Animation (vertical/horizontal point alignments)
- Code details…

Degeneracies: Collinear hull vertices

- *Strict* left turn
- Which is the result?
- Animation (vertical/horizontal point alignments)
- Code details. . .

Degeneracies: Collinear hull vertices

- *Strict* left turn
- Which is the result?
- Animation (vertical/horizontal point alignments)
- Code details. . .

Robustness?

- What about inaccurate floating-point calculations?
- *Structural integrity* is preserved!
- … As opposed to the “brute force” algorithm

Robustness?

- What about inaccurate floating-point calculations?
- *Structural integrity* is preserved!
- . . . As opposed to the “brute force” algorithm

Robustness?

- What about inaccurate floating-point calculations?
- *Structural integrity* is preserved!
- … As opposed to the “brute force” algorithm

Iteration invariants

Lower hull construction invariant:

- After i-th step: $P_i = \{ p_1, p_2, p_3, \ldots, p_{i-1}, p_i \}$
- No point of P_i lies below (to the right of) $lower\_hull(\ P_i\ )$
- $lower\_hull(\ P_i\ )$ is convex (left turn at each vertex)

Iteration invariants

Lower hull construction invariant:

- After i-th step: $P_i = \{ p_1, p_2, p_3, \ldots, p_{i-1}, p_i \}$
- No point of P_i lies below (to the right of) *lower_hull*(P_i)
- *lower_hull*(P_i) is convex (left turn at each vertex)

Iteration invariants

Lower hull construction invariant:

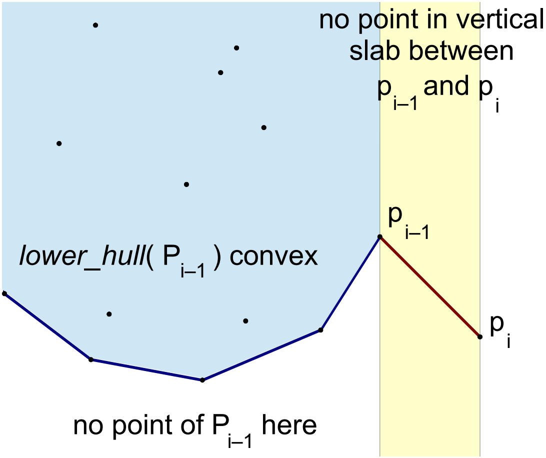
- After i-th step: $P_i \; = \; \{ \; p_1, \; p_2, \; p_3, \; \ldots, \; p_{i-1}, \; p_i \; \}$
- No point of P_i lies below (to the right of) *lower_hull*(P_i)
- *lower_hull*(P_i) is convex (left turn at each vertex)

Iteration invariants

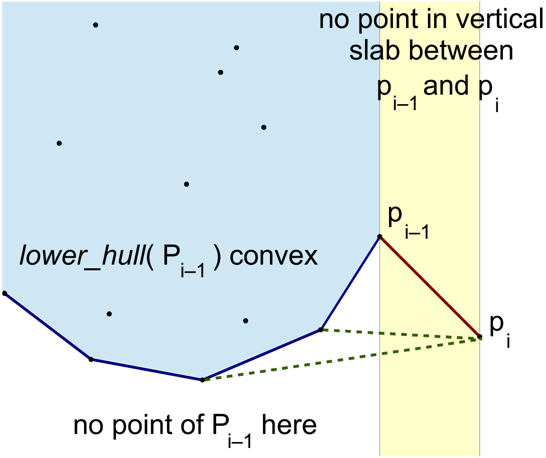
Lower hull construction invariant:

- After i-th step: $P_i \;=\; \{\; p_1,\; p_2,\; p_3,\; \ldots,\; p_{i-1},\; p_i \;\}$
- No point of P_i lies below (to the right of) *lower_hull*(P_i)
- *lower_hull*(P_i) is convex (left turn at each vertex)

Proof: Induction step



Proof: Induction step



Computational costs

- Sorting points in lexicographic order: $O(\ n \log n\ )$
- Adding points incrementally: $O(\ n\ )$ `for` iterations
- Updating lower/upper hull: $O(\ n\ )$ `while` iterations…
- … over *all* `for` iterations
  (at each further iteration a hull vertex is removed!)

Computational costs

- Sorting points in lexicographic order: $O(\, n \log n \,)$
- Adding points incrementally: $O(\, n \,)$ `for` iterations
- Updating lower/upper hull: $O(\, n \,)$ `while` iterations...
- ... over *all* `for` iterations
  (at each further iteration a hull vertex is removed!)

Computational costs

- Sorting points in lexicographic order: $O(\, n \log n \,)$
- Adding points incrementally: $O(\, n \,)$ `for` iterations
- Updating lower/upper hull: $O(\, n \,)$ `while` iterations…
- … over *all* `for` iterations
  (at each further iteration a hull vertex is removed!)

Computational costs

- Sorting points in lexicographic order: $O(\, n \log n \,)$
- Adding points incrementally: $O(\, n \,)$ `for` iterations
- Updating lower/upper hull: $O(\, n \,)$ `while` iterations…
- … over *all* `for` iterations
  (at each further iteration a hull vertex is removed!)

Optimality of $n \log n$

- May any better algorithm be conceived?
- *Sorting* $x_1, x_2, \ldots x_n$ can be reduced to *convex hull*...
- Consider point set $P = \{ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2) \}$
- Convex hull (sorted vertices): $n \log n$ lower bound

Optimality of $n \log n$

- May any better algorithm be conceived?
- *Sorting* $x_1, x_2, \ldots x_n$ can be reduced to *convex hull*...
- Consider point set $P = \{ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2) \}$
- Convex hull (sorted vertices): $n \log n$ lower bound

Optimality of $n \log n$

- May any better algorithm be conceived?
- *Sorting* $x_1, x_2, \ldots x_n$ can be reduced to *convex hull*...
- Consider point set $P = \{ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2) \}$
- Convex hull (sorted vertices): $n \log n$ lower bound

Optimality of $n \log n$

- May any better algorithm be conceived?
- *Sorting* $x_1, x_2, \ldots x_n$ can be reduced to *convex hull*. . .
- Consider point set $P = \{ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2) \}$
- Convex hull (sorted vertices): $n \log n$ lower bound

Variations on the theme. . .

- Graham's scan for "angular" order of points (around a convex hull vertex)
- Animation . . .
- . . . Try it yourself!

Variations on the theme...

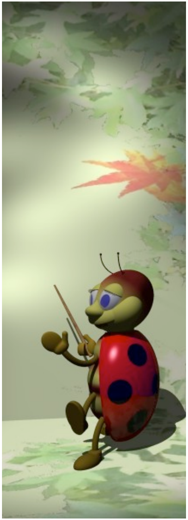
- Graham's scan for "angular" order of points (around a convex hull vertex)
- Animation ...
- ... Try it yourself!

Variations on the theme. . .

- Graham's scan for "angular" order of points (around a convex hull vertex)
- Animation . . .
- . . . Try it yourself!

Outline

1. Incremental algorithm
   - degeneracies
   - correctness
   - computational costs
2. Divide-et-impera algorithm
   - recursive approach
   - corrrectness
   - computational costs
3. Randomized algorithm
   - conflict graph
   - corrrectness
   - computational costs



Preparata & Hong's recursive approach

- Preliminarily, points are sorted lexicographically
- Balanced bipartition through a vertical line
- Convex hull of the left half (recursively)
- Convex hull of the right half (recursively)

Preparata & Hong's recursive approach

- Preliminarily, points are sorted lexicographically
- Balanced bipartition through a vertical line
- Convex hull of the left half (recursively)
- Convex hull of the right half (recursively)

Preparata & Hong's recursive approach

- Preliminarily, points are sorted lexicographically
- Balanced bipartition through a vertical line
- Convex hull of the left half (recursively)
- Convex hull of the right half (recursively)

Preparata & Hong's recursive approach

- Preliminarily, points are sorted lexicographically
- Balanced bipartition through a vertical line
- Convex hull of the left half (recursively)
- Convex hull of the right half (recursively)

Preparata & Hong's recursive approach

- Boundary walks to draw the connecting edges (common tangent lines above and below)
- Cut & sew appropriate (half)chains of points and connecting edges
- Animation / Code

Preparata & Hong's recursive approach

- Boundary walks to draw the connecting edges (common tangent lines above and below)
- Cut & sew appropriate (half)chains of points and connecting edges
- Animation / Code

Preparata & Hong's recursive approach

- Boundary walks to draw the connecting edges (common tangent lines above and below)
- Cut & sew appropriate (half)chains of points and connecting edges
- Animation / Code

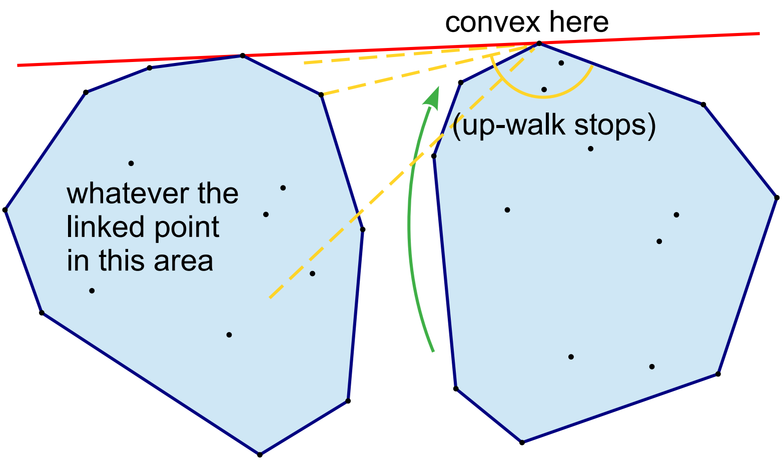
Correctness

- Recursive constructions are assumed to be correct (and base cases are)
- Walk(s) to determine connecting edges must come to an end and the resulting chain will be convex

Correctness

- Recursive constructions are assumed to be correct (and base cases are)
- Walk(s) to determine connecting edges must come to an end and the resulting chain will be convex

Correctness



Computational costs

- Sorting points in lexicographic order: $O(\ n \log n\ )$
- Walks + cut & sew: $O(\ n\ )$
- Well known equation: $T(\ n\ ) = 2\ T(\ n/2\ ) + O(\ n\ )$
- Solution: $T(\ n\ ) = O(\ n \log n\ )$

Computational costs

- Sorting points in lexicographic order: $O(\ n \log n\ )$
- Walks + cut & sew: $O(\ n\ )$
- Well known equation: $T(\ n\ ) = 2\,T(\ n/2\ ) + O(\ n\ )$
- Solution: $T(\ n\ ) = O(\ n \log n\ )$

Computational costs

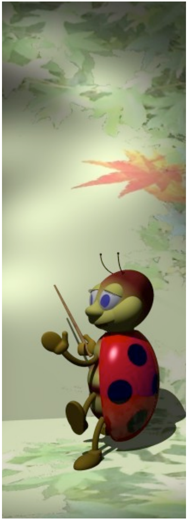
- Sorting points in lexicographic order: $O(\ n \log n\ )$
- Walks + cut & sew: $O(\ n\ )$
- Well known equation: $T(\ n\ ) \ = \ 2\ T(\ n/2\ ) \ + \ O(\ n\ )$
- Solution: $T(\ n\ ) \ = \ O(\ n \log n\ )$

Computational costs

- Sorting points in lexicographic order: $O(\ n \log n\ )$
- Walks + cut & sew: $O(\ n\ )$
- Well known equation: $T(\ n\ ) \ = \ 2\ T(\ n/2\ ) \ + \ O(\ n\ )$
- Solution: $T(\ n\ ) \ = \ O(\ n \log n\ )$

Outline

1. Incremental algorithm
   - degeneracies
   - correctness
   - computational costs
2. Divide-et-impera algorithm
   - recursive approach
   - corrrectness
   - computational costs
3. Randomized algorithm
   - conflict graph
   - corrrectness
   - computational costs



Randomization and conflict graph

Conflict-graph framework:

- *Objects*: problem input data — set S
- *Regions*: identified by $O(1)$ objects
- *Conflicts*: relationship between regions and objects

Result: Set of regions defined by the objects in S *without* conflicts with these objects

Randomization and conflict graph

Conflict-graph framework:

- *Objects*: problem input data — set S
- *Regions*: identified by $O(1)$ objects
- *Conflicts*: relationship between regions and objects

Result: Set of regions defined by the objects in S *without* conflicts with these objects

Randomization and conflict graph

Conflict-graph framework:

- *Objects*: problem input data — set S
- *Regions*: identified by $O(1)$ objects
- *Conflicts*: relationship between regions and objects

Result: Set of regions defined by the objects in S *without* conflicts with these objects

Randomization and conflict graph

Conflict-graph framework:

- *Objects*: problem input data — set S
- *Regions*: identified by $O(1)$ objects
- *Conflicts*: relationship between regions and objects

Result: Set of regions defined by the objects in S *without* conflicts with these objects

Randomization and conflict graph

Conflict-graph framework:

- *Objects*: problem input data — set S
- *Regions*: identified by $O(1)$ objects
- *Conflicts*: relationship between regions and objects

Result: Set of regions defined by the objects in S *without* conflicts with these objects

Conflict-graph

Assumptions (*static* approach):

- Links in both directions
  between regions and objects *in conflict*
- Direct link from object to conflict region
- Direct link from region to entry point
  (“iterator”) to list of conflicting objects

Conflict-graph

Assumptions (*static* approach):

- Links in both directions
  between regions and objects *in conflict*
- Direct link from object to conflict region
- Direct link from region to entry point
  (“iterator”) to list of conflicting objects

Conflict-graph

Assumptions (*static* approach):

- Links in both directions
  between regions and objects *in conflict*
- Direct link from object to conflict region
- Direct link from region to entry point
  (“iterator”) to list of conflicting objects

Conflict-graph as a general framework

Simple example: *Sorting* numbers

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph as a general framework

Simple example: *Sorting* numbers

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph as a general framework

Simple example: *Sorting* numbers

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph as a general framework

Simple example: *Sorting* numbers

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph as a general framework

Simple example: *Sorting* numbers

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph as a general framework

Simple example: A different point of view on *quick-sort*?

- *Objects*: real numbers x_i from finite set X
- *Regions*: intervals $[x_i, x_j]$ between (two) numbers from X
- *Conflicts*: $[x_i, x_j]$ and x are in conflict if $x \in]x_i, x_j[$

Result: pairs of numbers identifying intervals without conflicts are consecutive in the sorted sequence of X's elements

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer “sector” of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H
… Animation … incremental approach, indeed

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer “sector” of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H

… Animation … incremental approach, indeed

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer “sector” of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H
… Animation … incremental approach, indeed

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer “sector” of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H

… Animation … incremental approach, indeed

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer "sector" of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H
. . . Animation . . . incremental approach, indeed

Conflict-graph approach to the convex hull

Conflict-graph framework:

- *Objects*: points p_i from finite set P
- *Regions*: outer “sector” of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H
... Animation ... incremental approach, indeed

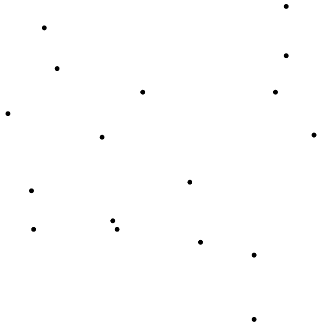
Conflict-graph approach to the convex hull

Conflict-graph framework:

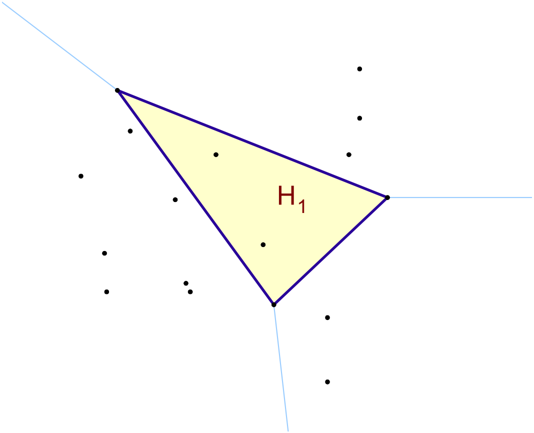
- *Objects*: points p_i from finite set P
- *Regions*: outer "sector" of (current) convex hull edge $p_i p_j$
- *Conflicts*: when $p \in P$ falls in one such outer sector

Result: edge chain H such that no point $p \in P$ lies outside H
... Animation ... incremental approach, indeed

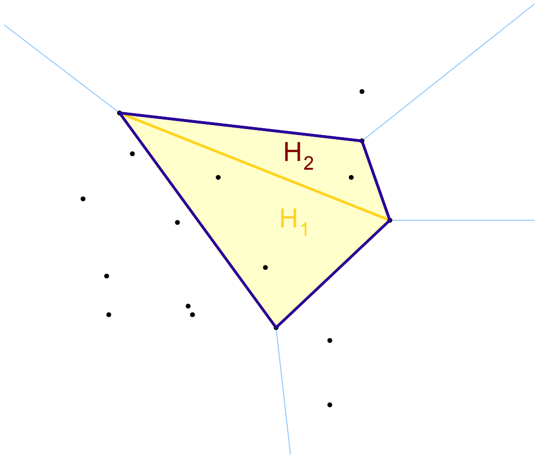
Randomized incremental convex hull algorithm



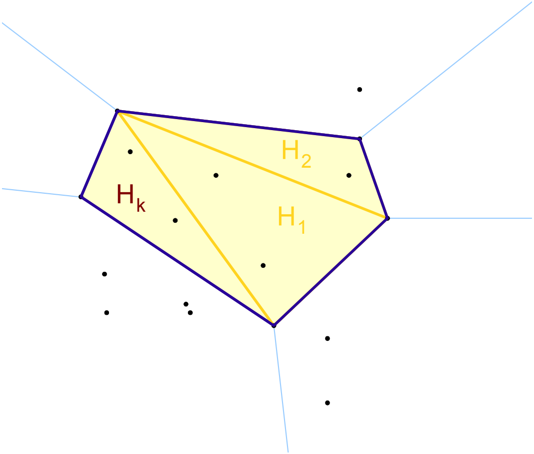
Randomized incremental convex hull algorithm



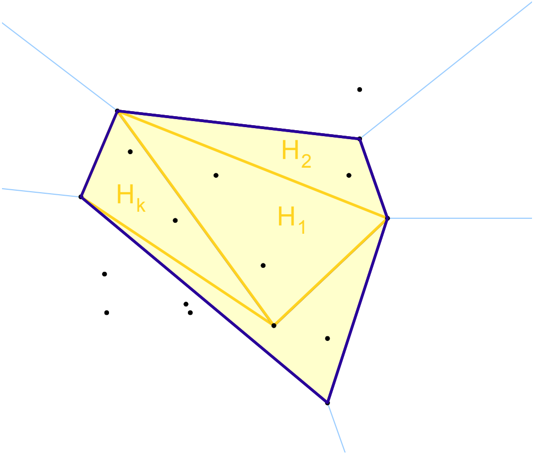
Randomized incremental convex hull algorithm



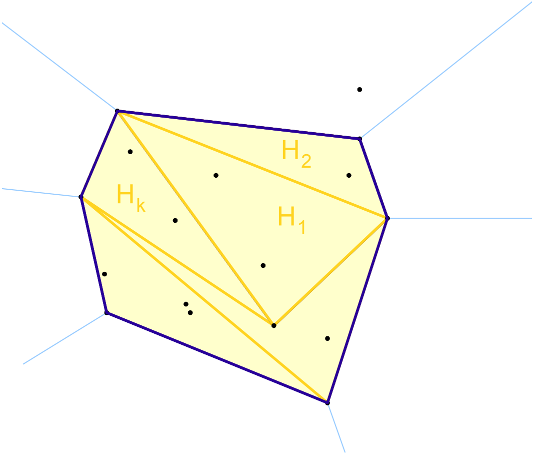
Randomized incremental convex hull algorithm



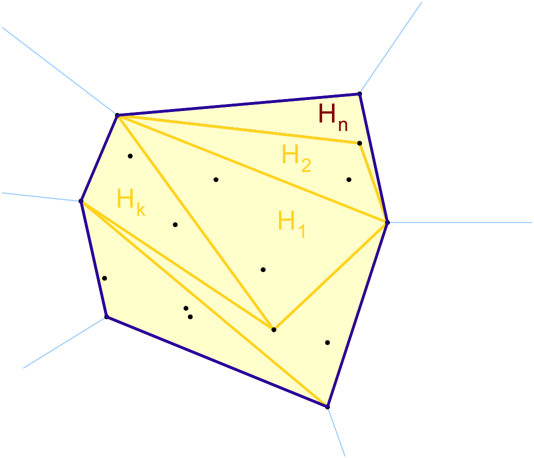
Randomized incremental convex hull algorithm



Randomized incremental convex hull algorithm



Randomized incremental convex hull algorithm



Conflict-graph approach to the convex hull

Conflict-graph framework:

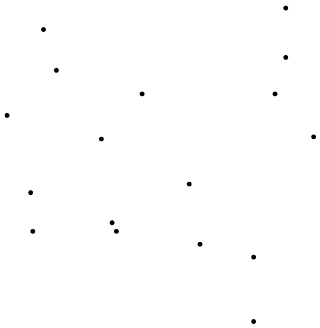
- *Objects*
- *Regions*
- *Conflicts*

Result: no point $p \in P$
lies outside H_n

Points p_i from finite set P

Conflict-graph framework:

- *Objects*
- *Regions*
- *Conflicts*



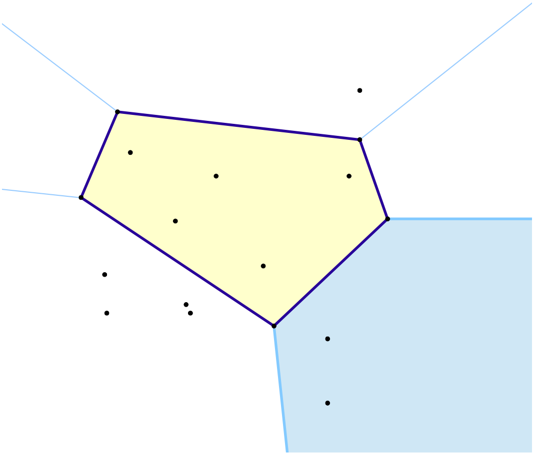
Result: no point $p \in P$ lies outside H_n

Outer sector of hull edge $p_i p_j$

Conflict-graph framework:

- *Objects*
- *Regions*
- *Conflicts*

Result: no point $p \in P$ lies outside H_n

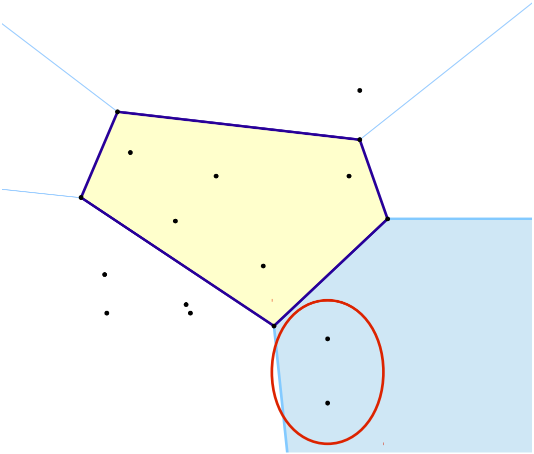


$p \in P$ falls in one such outer sector

Conflict-graph framework:

- *Objects*
- *Regions*
- *Conflicts*

Result: no point $p \in P$ lies outside H_n

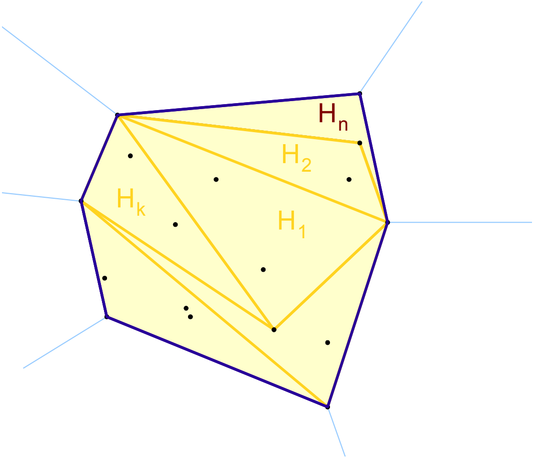


$p \in P$ falls in one such outer sector

Conflict-graph framework:

- *Objects*
- *Regions*
- *Conflicts*

Result: no point $p \in P$ lies outside H_n



Correctness

- Straightforward
- Walk(s) to get rid of non-convex vertices must come to an end and the updated hull will be convex

Correctness

- Straightforward
- Walk(s) to get rid of non-convex vertices must come to an end and the updated hull will be convex

Computational costs

At each stage, to update the convex hull from H_k to H_{k+1}, the algorithm accomplishes three main tasks:

- It removes a few "old" edges from H_k
- It adds two "new" edges to build H_{k+1}
- It re-arranges the links of the conflict graph between the outer regions of the new edges and all the points in the conflict lists of the removed edges

Computational costs

At each stage, to update the convex hull from H_k to H_{k+1}, the algorithm accomplishes three main tasks:

- It removes a few “old” edges from H_k
- It adds two “new” edges to build H_{k+1}
- It re-arranges the links of the conflict graph between the outer regions of the new edges and all the points in the conflict lists of the removed edges

Computational costs

At each stage, to update the convex hull from H_k to H_{k+1}, the algorithm accomplishes three main tasks:

- It removes a few “old” edges from H_k
- It adds two “new” edges to build H_{k+1}
- It re-arranges the links of the conflict graph between the outer regions of the new edges and all the points in the conflict lists of the removed edges

Removing and creating edges

- Only edges that have been created can be removed
- Two edges are created at each of the $O(n)$ stages
- Hence, the overall costs of both
  removing and creating edges are bound by $O(n)$

Removing and creating edges

- Only edges that have been created can be removed
- Two edges are created at each of the $O(n)$ stages
- Hence, the overall costs of both removing and creating edges are bound by $O(n)$

Removing and creating edges

- Only edges that have been created can be removed
- Two edges are created at each of the $O(n)$ stages
- Hence, the overall costs of both removing and creating edges are bound by $O(n)$

Updating the conflict graph

- Expected cost
- Approach: backward analysis
- Points which do not fall outside H_k at some stage will no longer be taken into account
- Focus on a set $P_k \subset P$ of k points (k-th stage) . . .

Updating the conflict graph

- Expected cost
- Approach: backward analysis
- Points which do not fall outside H_k at some stage will no longer be taken into account
- Focus on a set $P_k \subset P$ of k points (k-th stage) . . .

Updating the conflict graph

- Expected cost
- Approach: backward analysis
- Points which do not fall outside H_k at some stage will no longer be taken into account
- Focus on a set $P_k \subset P$ of k points (k-th stage) ...

Updating the conflict graph

- Expected cost
- Approach: backward analysis
- Points which do not fall outside H_k at some stage will no longer be taken into account
- Focus on a set $P_k \subset P$ of k points (k-th stage) ...

Backward analysis

- Because of the randomization, each of the k points in P_k may be the last added with equal probability $1/k$
- Let p be this last added point
- Then, either p is interior to H_k (no related processing) or p is a vertex of H_k with incident edges e' and e''
- Graph links re-arranged at the k-th stage (only) for points in the conflict lists of e' and e'' ...

Backward analysis

- Because of the randomization, each of the k points in P_k may be the last added with equal probability $1/k$
- Let p be this last added point
- Then, either p is interior to H_k (no related processing) or p is a vertex of H_k with incident edges e' and e''
- Graph links re-arranged at the k-th stage (only) for points in the conflict lists of e' and e'' ...

Backward analysis

- Because of the randomization, each of the k points in P_k may be the last added with equal probability $1/k$
- Let p be this last added point
- Then, either p is interior to H_k (no related processing) or p is a vertex of H_k with incident edges e' and e''
- Graph links re-arranged at the k-th stage (only) for points in the conflict lists of e' and e'' ...

Backward analysis

- Because of the randomization, each of the k points in P_k may be the last added with equal probability $1/k$
- Let p be this last added point
- Then, either p is interior to H_k (no related processing) or p is a vertex of H_k with incident edges e' and e''
- Graph links re-arranged at the k-th stage (only) for points in the conflict lists of e' and e'' ...

Backward analysis

- … the cost of re-arranging links at stage k is $O(l' + l'')$, where l', l'' are the sizes of the conflict lists of e', e''
- Then, the expected cost of the k-th stage is

$$\sum_{p \in P_k} \frac{1}{k}\, O(l' + l'') \quad = \quad \frac{1}{k} \sum_{e \in H_k} 2\, O(l) \quad \leq \quad \frac{2}{k}\, O(n)$$

since the conflict lists contain $n - k \leq n$ points overall[1]

[1] actually $l' + l''$ may underestimate the costs at stage k, but $O(n)$ recovers anything lost, or… only re-assigned conflicts need to be accounted for, since a point can be removed from the conflict lists at most once

Backward analysis

- … the cost of re-arranging links at stage k is $O(l' + l'')$, where l', l'' are the sizes of the conflict lists of e', e''
- Then, the expected cost of the k-th stage is

$$\sum_{p \in P_k} \frac{1}{k} \, O(l' + l'') \quad = \quad \frac{1}{k} \sum_{e \in H_k} 2 \, O(l) \quad \leq \quad \frac{2}{k} \, O(n)$$

since the conflict lists contain $n - k \leq n$ points overall[1]

[1] actually $l' + l''$ may underestimate the costs at stage k, but $O(n)$ recovers anything lost, or… only re-assigned conflicts need to be accounted for, since a point can be removed from the conflict lists at most once … $O(n)$ overall

Backward analysis

- … the cost of re-arranging links at stage k is $O(l' + l'')$, where l', l'' are the sizes of the conflict lists of e', e''
- Then, the expected cost of the k-th stage is

$$\sum_{p \in P_k} \frac{1}{k} O(l' + l'') \quad = \quad \frac{1}{k} \sum_{e \in H_k} 2\, O(l) \quad \leq \quad \frac{2}{k} O(n)$$

since the conflict lists contain $n - k \leq n$ points overall[1]

[1] actually $l' + l''$ may underestimate the costs at stage k, but $O(n)$ recovers anything lost, or…
only re-assigned conflicts need to be accounted for, since a point can be removed from the conflict lists at most once $\rightarrow$ $O(n)$ overall.

Backward analysis

- … the cost of re-arranging links at stage k is $O(l' + l'')$, where l', l'' are the sizes of the conflict lists of e', e''
- Then, the expected cost of the k-th stage is

$$\sum_{p \in P_k} \frac{1}{k} \, O(l' + l'') \quad = \quad \frac{1}{k} \sum_{e \in H_k} 2 \, O(l) \quad \leq \quad \frac{2}{k} \, O(n)$$

since the conflict lists contain $n - k \leq n$ points overall[1]

[1] actually $l' + l''$ may underestimate the costs at stage k, but $O(n)$ recovers anything lost, or…
only re-assigned conflicts need to be accounted for, since a point can be removed from the conflict lists at most once $\rightarrow$ $O(n)$ overall.

Backward analysis

- … the cost of re-arranging links at stage k is $O(l' + l'')$, where l', l'' are the sizes of the conflict lists of e', e''
- Then, the expected cost of the k-th stage is

$$\sum_{p \in P_k} \frac{1}{k} \, O(l' + l'') \quad = \quad \frac{1}{k} \sum_{e \in H_k} 2 \, O(l) \quad \leq \quad \frac{2}{k} \, O(n)$$

since the conflict lists contain $n - k \leq n$ points overall[1]

[1] actually $l' + l''$ may underestimate the costs at stage k, but $O(n)$ recovers anything lost, or…
only re-assigned conflicts need to be accounted for, since a point can be removed from the conflict lists at most once $\rightarrow$ $O(n)$ overall.

Backward analysis

- Expected cost of the k-th stage: $O(n)/k$
- Notice that $O(n)/k$ does not depend on the specific P_k, but the result would be the same for any $P_k \subset P$ of size k
- $O(n)/k$ is the expected cost of the k-th stage…

Backward analysis

- Expected cost of the k-th stage: $\qquad O(n)/k$
- Notice that $O(n)/k$ does not depend on the specific P_k, but the result would be the same for any $P_k \subset P$ of size k
- $O(n)/k$ is the expected cost of the k-th stage...

Backward analysis

- Expected cost of the k-th stage: $O(n)/k$
- Notice that $O(n)/k$ does not depend on the specific P_k, but the result would be the same for any $P_k \subset P$ of size k
- $O(n)/k$ is the expected cost of the k-th stage...

Backward analysis

- $O(n)/k$ is the expected cost of the k-th stage
- Then, the expected overall cost of re-arranging links is

$$\sum_{k=4}^{n} \frac{O(n)}{k} = O(n) \sum_{k=4}^{n} \frac{1}{k}$$

i.e. $O(\, n \log n \,)$ [$1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k}, \ldots$ harmonic series]

- ...which dominates the running time of the algorithm

Backward analysis

- $O(n)/k$ is the expected cost of the k-th stage
- Then, the expected overall cost of re-arranging links is

$$\sum_{k=4}^{n} \frac{O(n)}{k} \quad = \quad O(n) \sum_{k=4}^{n} \frac{1}{k}$$

i.e. $O(\, n \log n \,)$ $\quad [\ 1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k}, \ldots$ harmonic series $]$

- …which dominates the running time of the algorithm

Backward analysis

- $O(n)/k$ is the expected cost of the k-th stage
- Then, the expected overall cost of re-arranging links is

$$\sum_{k=4}^{n} \frac{O(n)}{k} \quad = \quad O(n) \sum_{k=4}^{n} \frac{1}{k}$$

i.e. $O(\ n \log n\ )$ $\quad$ [$1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k}, \ldots$ harmonic series]

- …which dominates the running time of the algorithm

Backward analysis

- $O(n)/k$ is the expected cost of the k-th stage
- Then, the expected overall cost of re-arranging links is

$$\sum_{k=4}^{n} \frac{O(n)}{k} \quad = \quad O(n) \sum_{k=4}^{n} \frac{1}{k}$$

i.e. $O(\, n \log n \,)$ [$1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k}, \ldots$ harmonic series]

- …which dominates the running time of the algorithm

Log trend of the harmonic series: Why?

$$1, \quad \frac{1}{2}, \quad \frac{1}{3}, \quad \frac{1}{4}, \quad \frac{1}{5}, \quad \frac{1}{6}, \quad \frac{1}{7}, \quad \frac{1}{8}, \quad \frac{1}{9}, \quad \ldots, \quad \frac{1}{15}, \quad \frac{1}{16}, \quad \ldots$$

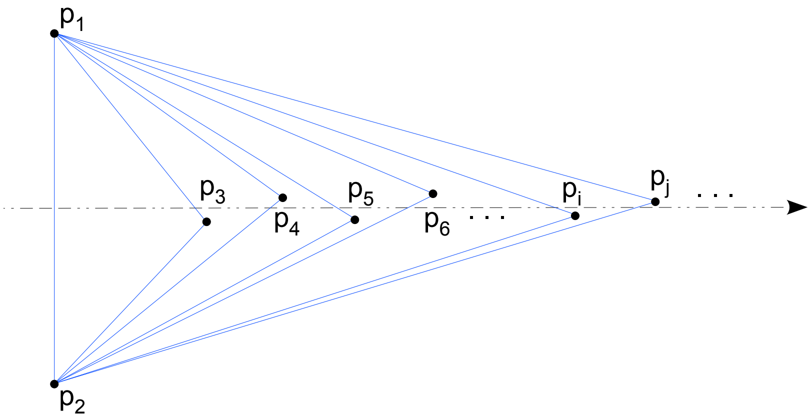
Log trend of the harmonic series: Why?

$$1, \quad \underbrace{\frac{1}{2}, \quad \frac{1}{3},}_{<\ 1} \quad \frac{1}{4}, \quad \frac{1}{5}, \quad \frac{1}{6}, \quad \frac{1}{7}, \quad \frac{1}{8}, \quad \frac{1}{9}, \quad \ldots, \quad \frac{1}{15}, \quad \frac{1}{16}, \quad \ldots$$

Log trend of the harmonic series: Why?

$$1,\quad \underbrace{\frac{1}{2},\quad \frac{1}{3},}_{<\ 1}\quad \underbrace{\frac{1}{4},\quad \frac{1}{5},\quad \frac{1}{6},\quad \frac{1}{7},}_{<\ 1}\quad \underbrace{\frac{1}{8},\quad \frac{1}{9},\quad \ldots,\quad \frac{1}{15},}_{<\ 1}\quad \frac{1}{16},\quad \ldots$$

Incremental algorithm: Worst-case point sequence

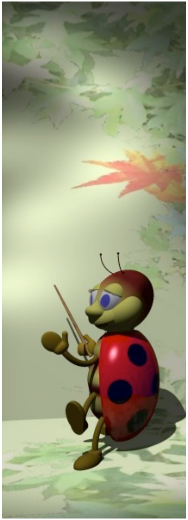


Outline

4 Semi-dynamic algorithms

5 Related results
- Convex hull in 3D
- Miscellaneous results

6 References



Static vs. semi-dynamic algorithms

- Conflict graph $\quad\rightarrow\quad$ influence graph
- *Influence graph*: tree-like, incrementally updated structure
- *static* algorithm $\quad\rightarrow\quad$ *semi-dynamic* algorithm
- Same computational trend, for *random* input data, provided the cost of each graph update is $O(\log n)$

Static vs. semi-dynamic algorithms

- Conflict graph $\rightarrow$ influence graph
- *Influence graph*: tree-like, incrementally updated structure
- *static* algorithm $\rightarrow$ *semi-dynamic* algorithm
- Same computational trend, for *random* input data, provided the cost of each graph update is $O(\log n)$

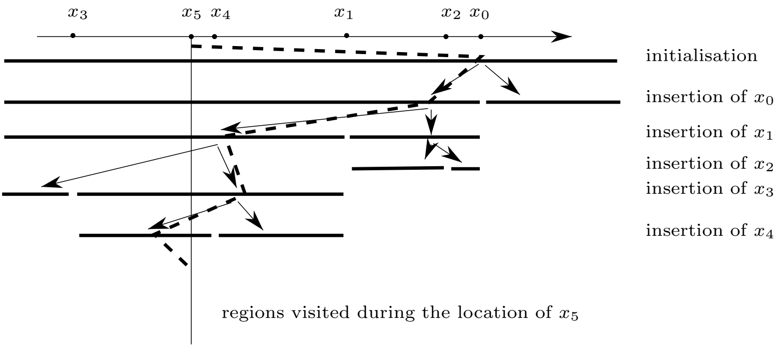
Static vs. semi-dynamic algorithms

- Conflict graph $\rightarrow$ influence graph
- *Influence graph*: tree-like, incrementally updated structure
- *static* algorithm $\rightarrow$ *semi-dynamic* algorithm
- Same computational trend, for *random* input data, provided the cost of each graph update is $O(\log n)$

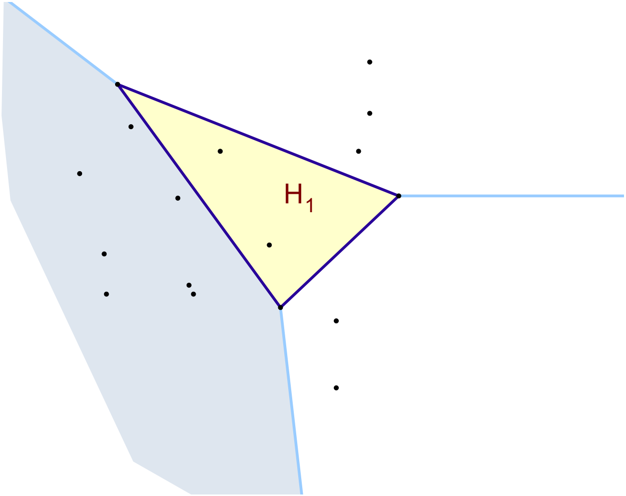
Static vs. semi-dynamic algorithms

- Conflict graph $\rightarrow$ influence graph
- *Influence graph*: tree-like, incrementally updated structure
- *static* algorithm $\rightarrow$ *semi-dynamic* algorithm
- Same computational trend, for *random* input data, provided the cost of each graph update is $O(\log n)$

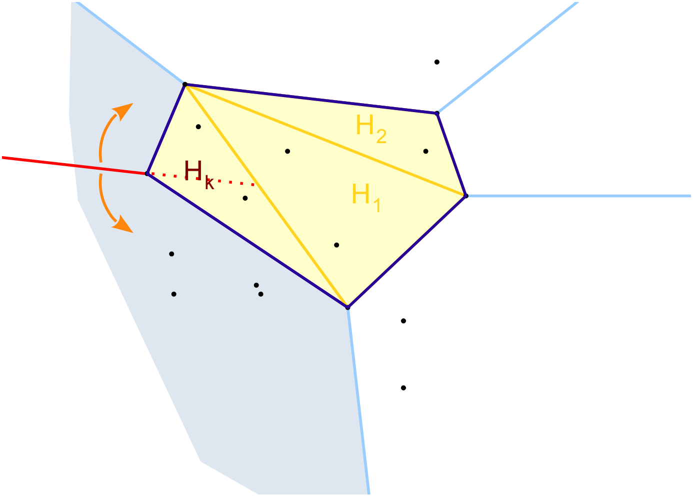
Devillers (1996)



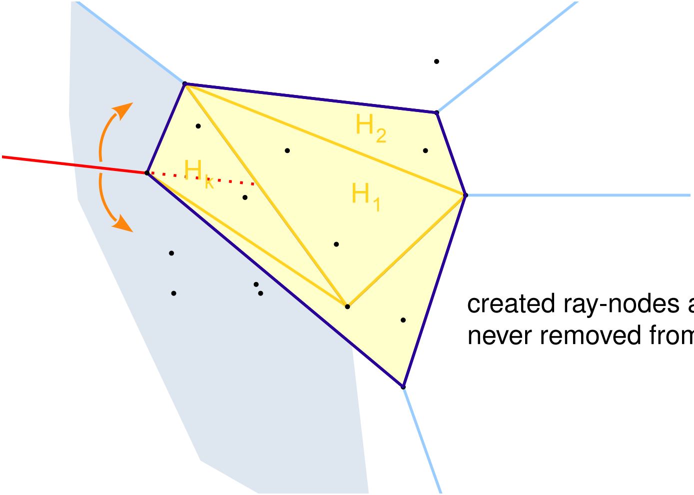
Semi-dynamic convex hull algorithm: Embedded tree



Semi-dynamic convex hull algorithm: Embedded tree

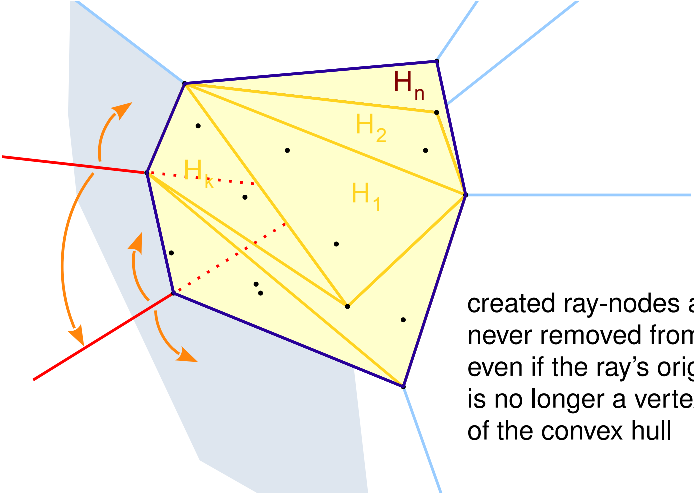


Semi-dynamic convex hull algorithm: Embedded tree



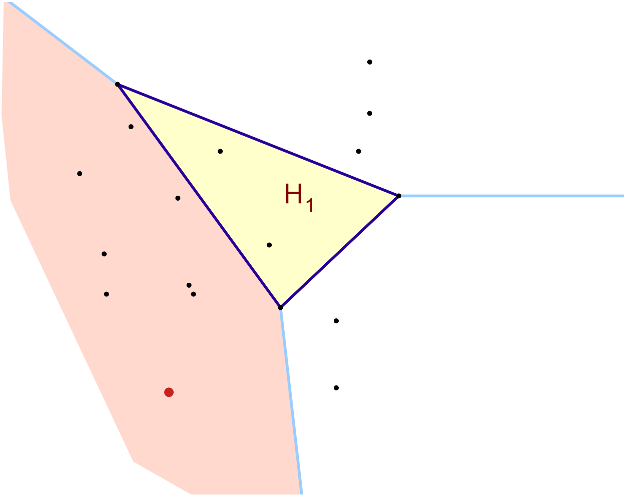
created ray-nodes are never removed from trees

Semi-dynamic convex hull algorithm: Embedded tree

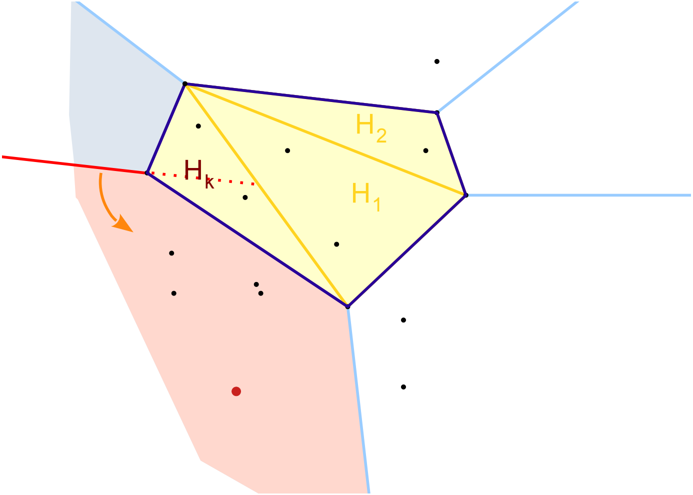


created ray-nodes are never removed from trees even if the ray's origin is no longer a vertex of the convex hull

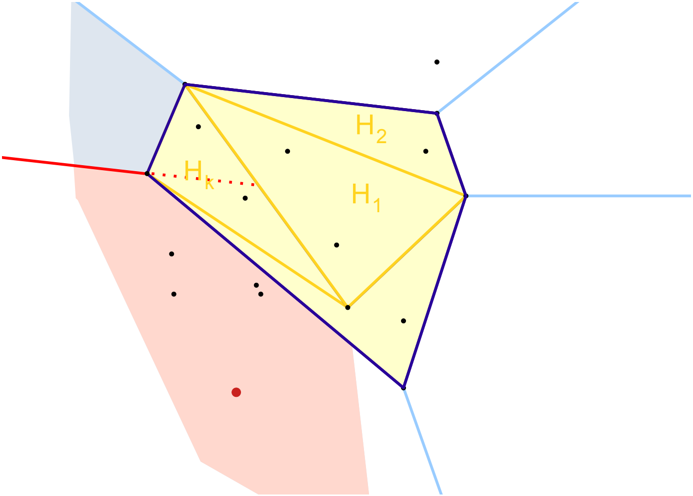
Semi-dynamic algorithm: Point location



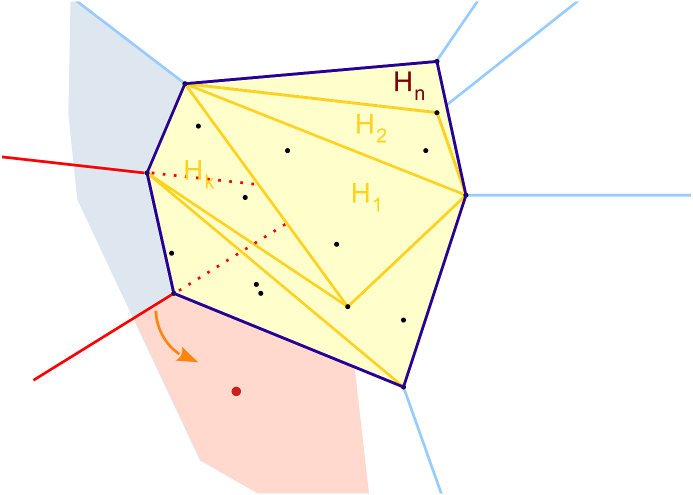
Semi-dynamic algorithm: Point location



Semi-dynamic algorithm: Point location



Semi-dynamic algorithm: Point location

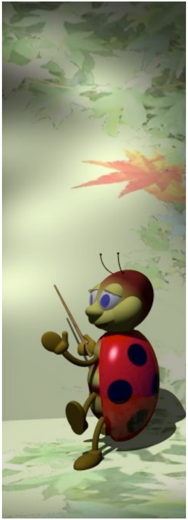


Outline

4 Semi-dynamic algorithms

5 Related results
- Convex hull in 3D
- Miscellaneous results

6 References



Convex hull in 3D

- No straightforward 3D generalization of Graham's scan
- Divide-et-impera approach:
  Preparata & Hong (1977), $O(n \log n)$
- Randomized incremental approach:
  E.g. see survey in Devillers (1996), $O(n \log n)$

Convex hull in 3D

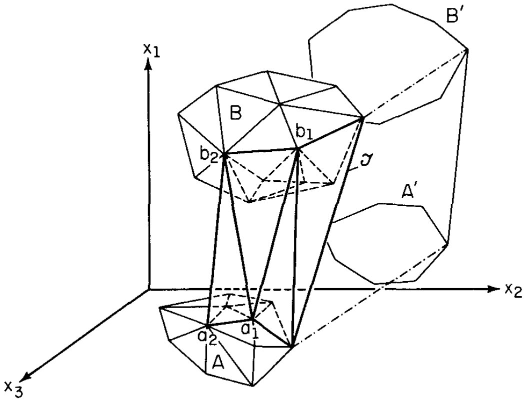
- No straightforward 3D generalization of Graham's scan
- Divide-et-impera approach:
  Preparata & Hong (1977), $O(\ n \log n\ )$
- Randomized incremental approach:
  E.g. see survey in Devillers (1996), $O(\ n \log n\ )$

Convex hull in 3D

- No straightforward 3D generalization of Graham's scan
- Divide-et-impera approach:
  Preparata & Hong (1977), $O(\, n \log n \,)$
- Randomized incremental approach:
  E.g. see survey in Devillers (1996), $O(\, n \log n \,)$

Preparata & Hong (1977)

Fig. 4. Merging two convex hulls. Construction of $\mathfrak{J}$.



Remark: Euler's formula for planar graphs

- Relation between # of vertices (V), edges (E), faces (F), and connected components (C) of a planar graph
- $$V + F = E + C + 1$$
- *Proof*: adding vertex or edge connecting two vertices
- Base case (1 vertex): $V = 1,\ F = 1,\ E = 0,\ C = 1$

Remark: Euler's formula for planar graphs

- Relation between # of vertices (V), edges (E), faces (F), and connected components (C) of a planar graph
- $$V + F = E + C + 1$$
- *Proof*: adding vertex or edge connecting two vertices
- Base case (1 vertex): $V = 1, \; F = 1, \; E = 0, \; C = 1$

Remark: Euler's formula for planar graphs

- Relation between # of vertices (V), edges (E), faces (F), and connected components (C) of a planar graph
- $$V + F = E + C + 1$$
- *Proof*: adding vertex or edge connecting two vertices
- Base case (1 vertex): $V = 1, \; F = 1, \; E = 0, \; C = 1$

Remark: Euler's formula for planar graphs

- Relation between # of vertices (V), edges (E), faces (F), and connected components (C) of a planar graph
- $$V + F = E + C + 1$$
- *Proof*: adding vertex or edge connecting two vertices
- Base case (1 vertex): $V = 1,\ F = 1,\ E = 0,\ C = 1$

Remark: Euler's formula for planar graphs

- Invariant: $V + F = E + C + 1$
- Inductive step...
- Adding a disconnected vertex:

  $$V' = V + 1,\ F' = F,\ E' = E,\ C' = C + 1$$

- Adding an edge between disconnected components:

  $$V' = V,\ F' = F,\ E' = E + 1,\ C' = C - 1$$

- Adding an edge within a connected component:

  $$V' = V,\ F' = F + 1,\ E' = E + 1,\ C' = C$$

Remark: Euler's formula for planar graphs

- Invariant: $V + F = E + C + 1$
- Inductive step...
- Adding a disconnected vertex:

$$V' = V + 1,\ F' = F,\ E' = E,\ C' = C + 1$$

- Adding an edge between disconnected components:

$$V' = V,\ F' = F,\ E' = E + 1,\ C' = C - 1$$

- Adding an edge within a connected component:

$$V' = V,\ F' = F + 1,\ E' = E + 1,\ C' = C$$

Remark: Euler's formula for planar graphs

- Invariant: $V + F = E + C + 1$
- Inductive step...
- Adding a disconnected vertex:

$$V' = V + 1,\ F' = F,\ E' = E,\ C' = C + 1$$

- Adding an edge between disconnected components:

$$V' = V,\ F' = F,\ E' = E + 1,\ C' = C - 1$$

- Adding an edge within a connected component:

$$V' = V,\ F' = F + 1,\ E' = E + 1,\ C' = C$$

Remark: Euler's formula for planar graphs

- Invariant: $V + F = E + C + 1$
- Inductive step…
- Adding a disconnected vertex:

$$V' = V + 1,\ F' = F,\ E' = E,\ C' = C + 1$$

- Adding an edge between disconnected components:

$$V' = V,\ F' = F,\ E' = E + 1,\ C' = C - 1$$

- Adding an edge within a connected component:

$$V' = V,\ F' = F + 1,\ E' = E + 1,\ C' = C$$

Remark: Euler's formula for planar graphs

- Invariant: $V + F = E + C + 1$
- Inductive step...
- Adding a disconnected vertex:

$$V' = V + 1,\ F' = F,\ E' = E,\ C' = C + 1$$

- Adding an edge between disconnected components:

$$V' = V,\ F' = F,\ E' = E + 1,\ C' = C - 1$$

- Adding an edge within a connected component:

$$V' = V,\ F' = F + 1,\ E' = E + 1,\ C' = C$$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow$ $E < 3V$ and $F < 2V$
- To sum up:

  $V = O(\ E\ );\ E = O(\ V\ );\ F = O(\ V\ ) = O(\ E\ )$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $\quad V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow \; E < 3V$ and $F < 2V$
- To sum up:

  $V = O(\,E\,);\; E = O(\,V\,);\; F = O(\,V\,) = O(\,E\,)$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow$ $E < 3V$ and $F < 2V$
- To sum up:

  $V = O(E)$; $E = O(V)$; $F = O(V) = O(E)$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow$ $E < 3V$ and $F < 2V$
- To sum up:

  $V = O(\ E\ );\ E = O(\ V\ );\ F = O(\ V\ ) = O(\ E\ )$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $\quad V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow \; E < 3V$ and $F < 2V$
- To sum up:

  $V = O(\,E\,);\; E = O(\,V\,);\; F = O(\,V\,) = O(\,E\,)$

Remark: Euler's formula for polyhedra

- Invariant ($C = 1$): $\quad V + F = E + 2$
- No disconnected vertices: $V \leq 2E$
- No two edges between the same pair of vertices:

  $F \leq 2E/3$

  i.e., $2E$ halfedges and ≥ 3 halfedges per face
- Hence: $E = V + F - 2 < V + 2E/3$
- $\Rightarrow \; E < 3V$ and $F < 2V$
- To sum up:

  $V = O(\ E\ );\ E = O(\ V\ );\ F = O(\ V\ ) = O(\ E\ )$

More about convex hull algorithms

- Jarvis' march (2D) / gift wrapping (3D)
  - simple to code
  - running time: $O(\ nh\ )$
  - h = hull vertices: may it be convenient?
- Optimal output sensitive algorithms
  - E.g., Chan (1996)
  - running time: $O(\ n \log h\ )$
  - more complex structure

More about convex hull algorithms

- Jarvis' march (2D) / gift wrapping (3D)
  - simple to code
  - running time: $O(\ nh\ )$
  - h = hull vertices: may it be convenient?
- Optimal output sensitive algorithms
  - E.g., Chan (1996)
  - running time: $O(\ n \log h\ )$
  - more complex structure

More about convex hull algorithms

- Jarvis' march (2D) / gift wrapping (3D)
  - simple to code
  - running time: $O(\ nh\ )$
  - h = hull vertices: may it be convenient?
- Optimal output sensitive algorithms
  - E.g., Chan (1996)
  - running time: $O(\ n \log h\ )$
  - more complex structure

More about convex hull algorithms

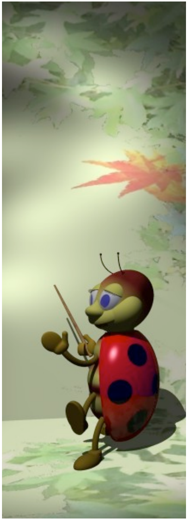
- Jarvis' march (2D) / gift wrapping (3D)
  - simple to code
  - running time: $O(\ nh\ )$
  - h = hull vertices: may it be convenient?
- Optimal output sensitive algorithms
  - E.g., Chan (1996)
  - running time: $O(\ n \log h\ )$
  - more complex structure

Outline

4 Semi-dynamic algorithms

5 Related results
- Convex hull in 3D
- Miscellaneous results

6 References



Epilogue. . .

"Convex hull is the favorite paradigm of computational geometers.

Although the description of the problem is fairly simple, its solution takes into account all aspects of computational geometry."

Olivier Devillers (1996)

References

R.L. Graham (1972)
An efficient algorithm for determining the convex hull of a finite planar set
Information Processing Letters, 1

R.A. Jarvis (1973)
On the identification of the convex hull of a finite set of points in the plane
Information Processing Letters, 2

References

F.P. Preparata & S.J. Hong (1977)
Convex hulls of finite sets of points
in two and three dimensions
Communications of the ACM, 157

O. Devillers (1996)
An introduction to randomization
in computational geometry
Theoretical Computer Science, 20(2)

References

T.M. Chan (1996)
Optimal output-sensitive convex hull algorithms
in two and three dimensions
Discrete & Computational Geometry, 16

S. Har-Peled (2011)
On the expected complexity of random convex hulls
ArXiv e-prints

M. Golin & R. Sedgewick (1988)
Analysis of a simple yet efficient convex hull algorithm
Proc. of the 4th Symp. on Computational Geometry