Computational Geometry
(WS 2024/25)

Martin Held

FB Informatik
Universität Salzburg
A-5020 Salzburg, Austria
held@cs.sbg.ac.at

October 3, 2024

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Personalia

Instructor (VO+PS): M. Held.

Email: `held@cs.sbg.ac.at`.

Base-URL: `www.cosy.sbg.ac.at/~held`.

Office: PLUS, FB Informatik, Rm. 1.20, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.

Phone number (office): (0662) 8044-6304.

Phone number (secr.): (0662) 8044-6300.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Formalia

URL of course (VO+PS): Base-URL`/teaching/compgeo/comp_geo.html`.

Lecture times (VO): Friday 12^{15}–14^{15}.

Venue (VO): PLUS, FB Informatik, T03, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.

Lecture times (PS): Friday 11^{00}–12^{00}.

Venue (PS): PLUS, FB Informatik, T03, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.

Formalia

URL of course (VO+PS): Base-URL/teaching/compgeo/comp_geo.html.

Lecture times (VO): Friday 12^{15}–14^{15}.

Venue (VO): PLUS, FB Informatik, T03, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.

Lecture times (PS): Friday 11^{00}–12^{00}.

Venue (PS): PLUS, FB Informatik, T03, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.

Note — PS is graded according to continuous-assessment mode!

— regular attendance is compulsory!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Electronic Slides and Online Material

In addition to these slides, you are encouraged to consult the WWW home-page of this lecture:

www.cosy.sbg.ac.at/~held/teaching/compgeo/comp_geo.html.

In particular, this WWW page contains up-to-date information on the course, plus links to online notes, slides and (possibly) sample code.

A Few Words of Warning

- I hope that these slides will serve as a practice-minded introduction to various aspects of computational geometry. I would like to warn you explicitly not to regard these slides as the sole source of information on the topics of my course. It may and will happen that I'll use the lecture for talking about subtle details that need not be covered in these slides! In particular, the slides won't contain all sample calculations, proofs of theorems, demonstrations of algorithms, or solutions to problems posed during my lecture. That is, by making these slides available to you I do not intend to encourage you to attend the lecture on an irregular basis.

A Few Words of Warning

- I hope that these slides will serve as a practice-minded introduction to various aspects of computational geometry. I would like to warn you explicitly not to regard these slides as the sole source of information on the topics of my course. It may and will happen that I'll use the lecture for talking about subtle details that need not be covered in these slides! In particular, the slides won't contain all sample calculations, proofs of theorems, demonstrations of algorithms, or solutions to problems posed during my lecture. That is, by making these slides available to you I do not intend to encourage you to attend the lecture on an irregular basis.
- See also In Praise of Lectures by T.W. Körner.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

A Few Words of Warning

- I hope that these slides will serve as a practice-minded introduction to various aspects of computational geometry. I would like to warn you explicitly not to regard these slides as the sole source of information on the topics of my course. It may and will happen that I'll use the lecture for talking about subtle details that need not be covered in these slides! In particular, the slides won't contain all sample calculations, proofs of theorems, demonstrations of algorithms, or solutions to problems posed during my lecture. That is, by making these slides available to you I do not intend to encourage you to attend the lecture on an irregular basis.
- See also In Praise of Lectures by T.W. Körner.
- A *basic knowledge of algorithms, data structures and discrete mathematics*, as taught typically in undergraduate courses, should suffice to take this course. It is my sincere intention to start at such a hypothetical level of "typical prior undergrad knowledge". Still, it is obvious that different educational backgrounds will result in different levels of prior knowledge. Hence, you might realize that you do already know some items covered in this course, while you lack a decent understanding of some items which I seem to presuppose. In such a case I do expect you to refresh or fill in those missing items on your own!

Acknowledgments

These slides are partially based on notes and slides transcribed by various students — most notably Elias Pschernig, Christian Spielberger, Werner Weiser and Franz Wilhelmstötter — for previous courses on “Algorithmische Geometrie”. Some figures were derived from figures originally prepared by students of my lecture “Wissenschaftliche Arbeitstechniken und Präsentation”, while others were taken from papers and slides co-authored with members of my research group, such as Günther Eder, Stefan Huber, Stefan de Lorenzo, Willi Mann, Peter Palfrader, Christian Spielberger. I would like to express my thankfulness to all of them for their help. This revision and extension was carried out by myself, and I am responsible for any errors.

I am also happy to acknowledge that we benefited from material published by colleagues on diverse topics that are partially covered in this lecture. While some of the material used for this lecture was originally presented in traditional-style publications (such as textbooks), some other material has its roots in non-standard publication outlets (such as online documentations, electronic course notes, or user manuals).

Salzburg, July 2024 — Martin Held

Legal Fine Print and Disclaimer

To the best of our knowledge, these slides do not violate or infringe upon somebody else's copyrights. If copyrighted material appears in these slides then it was considered to be available in a non-profit manner and as an educational tool for teaching at an academic institution, within the limits of the "fair use" policy. For copyrighted material we strive to give references to the copyright holders (if known). Of course, any trademarks mentioned in these slides are properties of their respective owners.

Please note that these slides are copyrighted. The copyright holder(s) grant you the right to download and print it for your personal use. Any other use, including non-profit instructional use and re-distribution in electronic or printed form of significant portions of it, beyond the limits of "fair use", requires the explicit permission of the copyright holder(s). All rights reserved.

These slides are made available without warrant of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. In no event shall the copyright holder(s) and/or their respective employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, arising out of or in connection with the use of information provided in these slides.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Recommended Textbooks

- F.P. Preparata and M.I. Shamos.
  Computational Geometry – An Introduction.
  Springer-Verlag, 3rd edition, Aug 1993. ISBN 978-0387961316.
- J. O'Rourke.
  Computational Geometry in C.
  Cambridge University Press, 2nd edition, 2000. ISBN 978-0521649766.
  https://doi.org/10.1017/CBO9780511804120.
- M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars.
  Computational Geometry. Algorithms and Applications.
  Springer-Verlag, 3rd rev. edition, March 2008. ISBN 978-3540779735.
- F. Aurenhammer, R. Klein and D.-T. Lee.
  Voronoi Diagrams and Delaunay Triangulations.
  World Scientific Publ., Aug 2013. ISBN 978-981-4447-63-8.
- C.D. Toth, J. O'Rourke, J.E. Goodman.
  Handbook of Discrete and Computational Geometry.
  CRC Press, Nov 2017. ISBN 9781498711395.

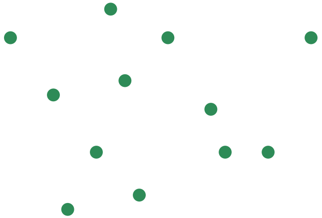
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Table of Content

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Motivation: Convex Hull

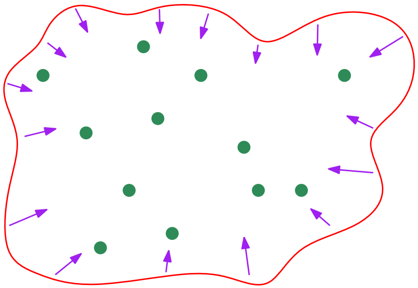
- Given is a set S of n points in $\mathbb{R}^2$.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

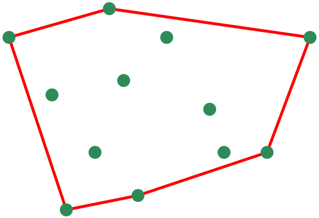
Motivation: Convex Hull

- Given is a set S of n points in $\mathbb{R}^2$.
- Question: How efficiently can we determine the convex hull of S?

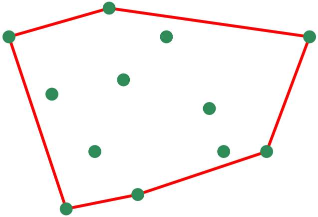


Motivation: Convex Hull

- Given is a set S of n points in $\mathbb{R}^2$.
- Question: How efficiently can we determine the convex hull of S?



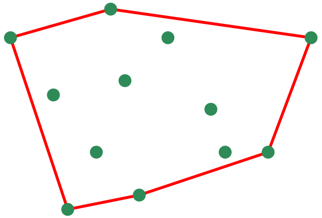
- Given is a set S of n points in $\mathbb{R}^2$.
- Question: How efficiently can we determine the convex hull of S?



- Answer: The convex hull of S can be computed in $O(n \log n)$ steps.

Motivation: Convex Hull

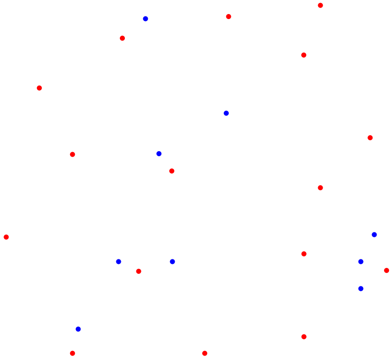
- Given is a set S of n points in $\mathbb{R}^2$.
- Question: How efficiently can we determine the convex hull of S?



- Answer: The convex hull of S can be computed in $O(n \log n)$ steps.
- Lower bound: In the worst case, $\Omega(n \log n)$ steps will be necessary.

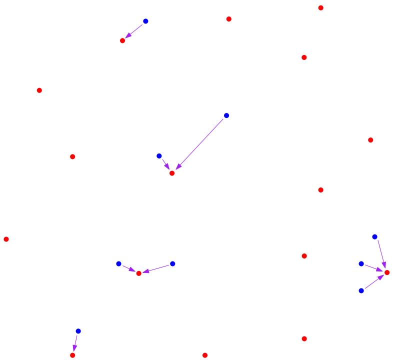
Motivation: Distance between Point Sets

- Le S_1 and S_2 be sets of blue and red points in $\mathbb{R}^2$.



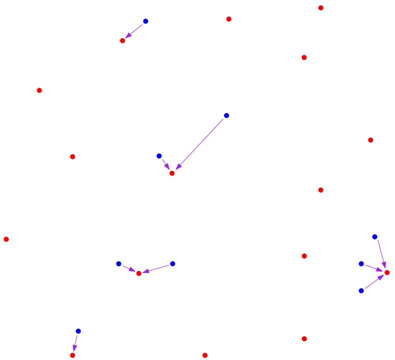
Motivation: Distance between Point Sets

- Le S_1 and S_2 be sets of blue and red points in $\mathbb{R}^2$.
- For each blue point, consider the distance to its closest red point.



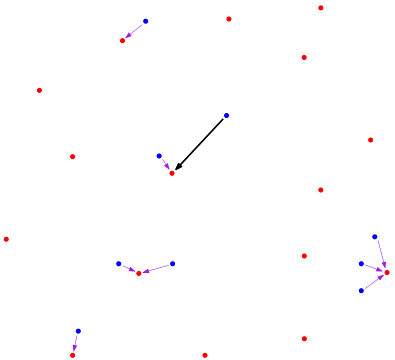
Motivation: Distance between Point Sets

- Le S_1 and S_2 be sets of blue and red points in $\mathbb{R}^2$.
- For each blue point, consider the distance to its closest red point.
- Question: What is the maximum of these distances?



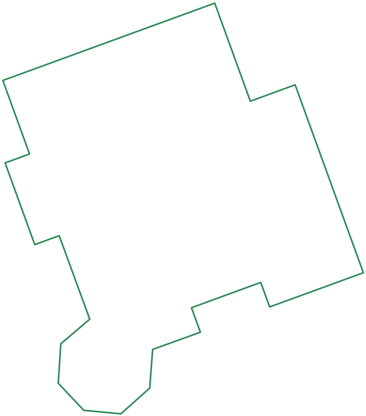
Motivation: Distance between Point Sets

- Le S_1 and S_2 be sets of blue and red points in $\mathbb{R}^2$.
- For each blue point, consider the distance to its closest red point.
- Question: What is the maximum of these distances?
- Answer: This is the so-called directed Hausdorff distance, and it can be obtained in $O(n \log n)$ time, where $n := \max\{|S_1|, |S_2|\}$.



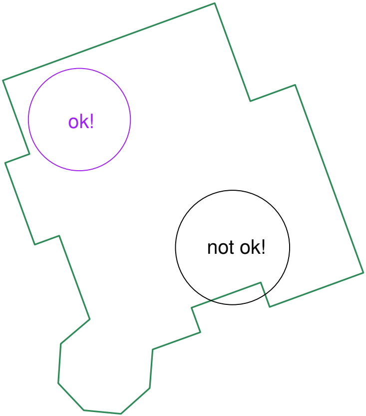
Motivation: Maximum Inscribed Circle

- Given is a simple polygon $\mathcal{P}$.



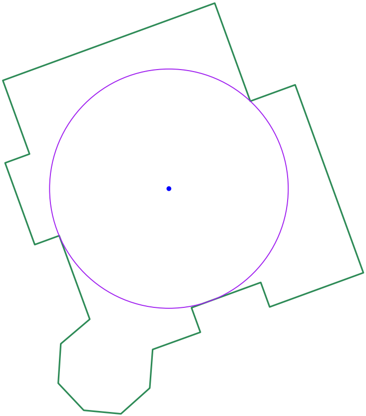
Motivation: Maximum Inscribed Circle

- Given is a simple polygon $\mathcal{P}$. A circle is called inscribed to $\mathcal{P}$ if it lies completely inside of $\mathcal{P}$.



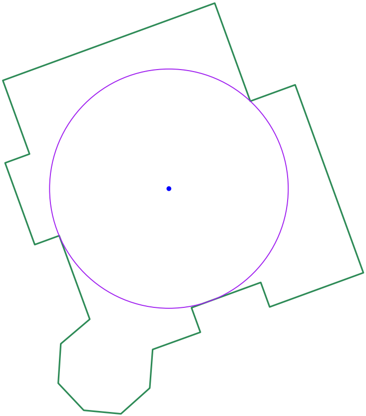
Motivation: Maximum Inscribed Circle

- Given is a simple polygon $\mathcal{P}$. A circle is called inscribed to $\mathcal{P}$ if it lies completely inside of $\mathcal{P}$.
- Question: How efficiently can we determine a maximum inscribed circle?



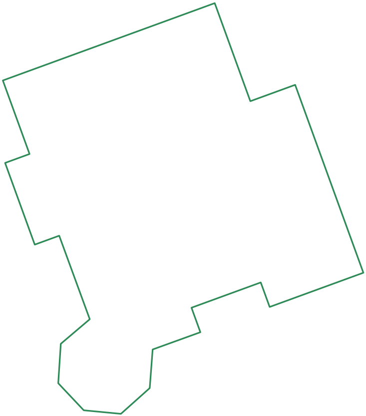
Motivation: Maximum Inscribed Circle

- Given is a simple polygon $\mathcal{P}$. A circle is called inscribed to $\mathcal{P}$ if it lies completely inside of $\mathcal{P}$.
- Question: How efficiently can we determine a maximum inscribed circle?
- Answer: In theory, a maximum inscribed circle can be computed in time linear in the number n of vertices of $\mathcal{P}$. (And $O(n \log n)$ time is achievable in practice.)



Motivation: Offset Pattern

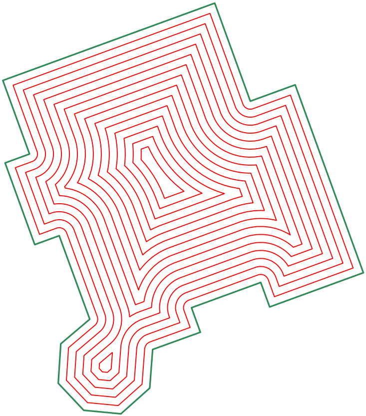
- Given is a simple polygon $\mathcal{P}$.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

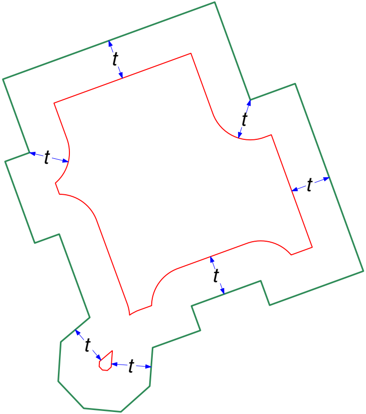
Motivation: Offset Pattern

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute offset patterns reliably and efficiently?



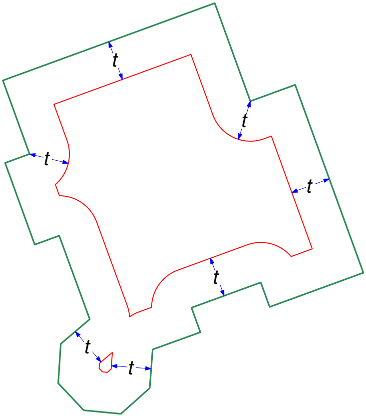
Motivation: Offset Pattern

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute offset patterns reliably and efficiently? How can we compute even just one offset?



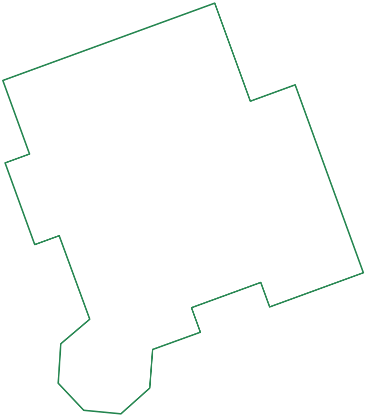
Motivation: Offset Pattern

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute offset patterns reliably and efficiently? How can we compute even just one offset?
- Answer: If the Voronoi diagram of the input is known, then all offset curves of one offset can be determined in $O(n)$ time.



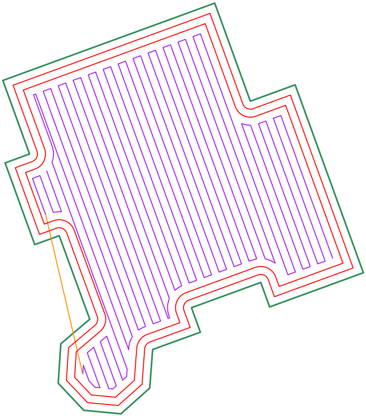
Motivation: Tool Path

- Given is a simple polygon $\mathcal{P}$.



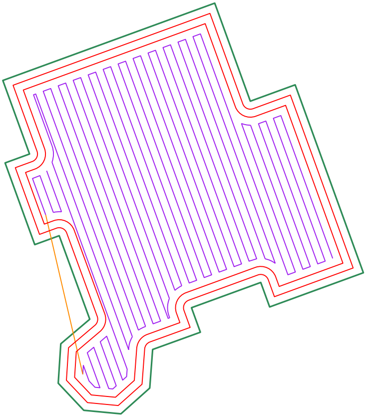
Motivation: Tool Path

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a tool path — e.g., for machining or 3D printing — inside of $\mathcal{P}$ reliably and efficiently?



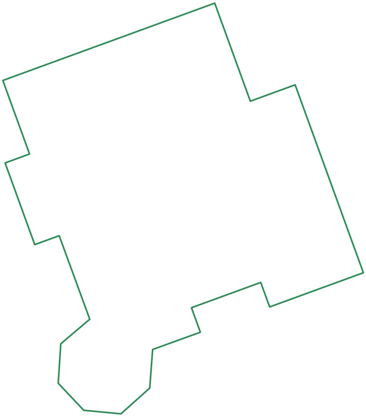
Motivation: Tool Path

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a tool path — e.g., for machining or 3D printing — inside of $\mathcal{P}$ reliably and efficiently?
- Answer: Again, this can be done with the help of Voronoi diagrams.



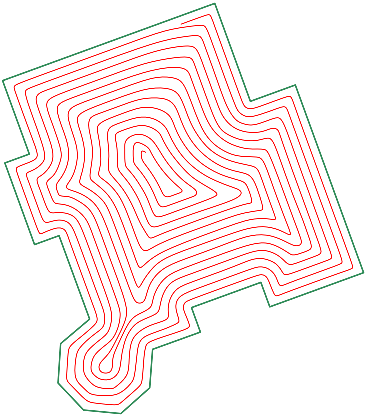
Motivation: Smooth Tool Path

- Given is a simple polygon $\mathcal{P}$.



Motivation: Smooth Tool Path

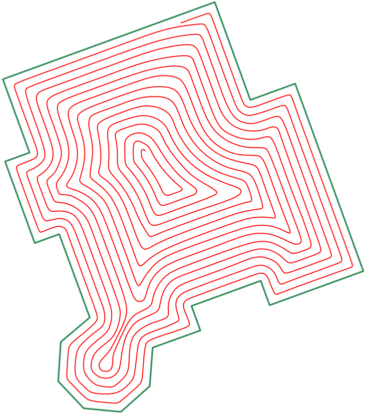
- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a smooth tool path — e.g., for high-speed machining — inside of $\mathcal{P}$ reliably and efficiently?



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

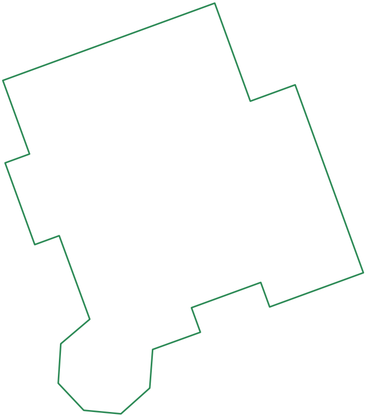
Motivation: Smooth Tool Path

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a smooth tool path — e.g., for high-speed machining — inside of $\mathcal{P}$ reliably and efficiently?
- Answer: Again, this can be done with the help of Voronoi diagrams.



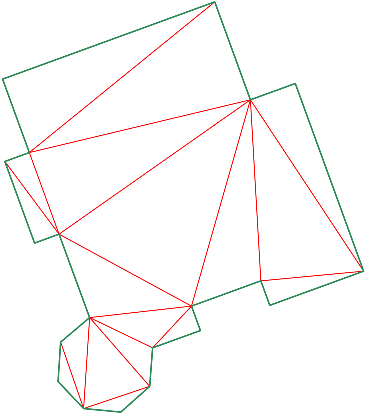
Motivation: Triangulation

- Given is a simple polygon $\mathcal{P}$.



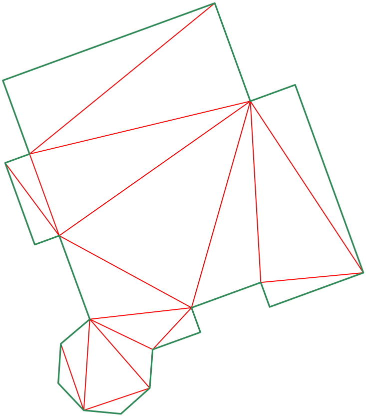
Motivation: Triangulation

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a triangulation of $\mathcal{P}$ reliably and efficiently?



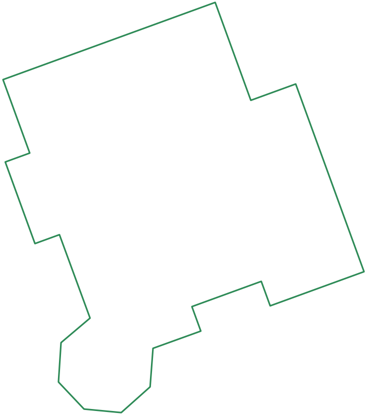
Motivation: Triangulation

- Given is a simple polygon $\mathcal{P}$.
- Question: How can we compute a triangulation of $\mathcal{P}$ reliably and efficiently?
- Answer: In theory, a triangulation can be computed in time linear in the number n of vertices of $\mathcal{P}$. (And slightly super-linear time is achievable in practice.)



Motivation: Automatic Roof Construction

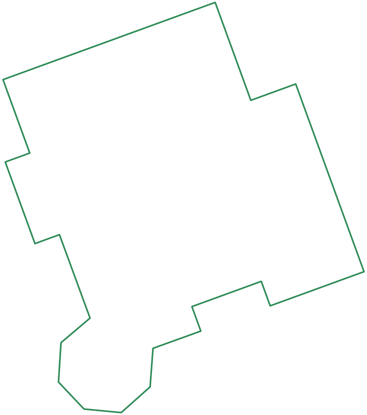
- Given is a simple polygon $\mathcal{P}$, which we consider as the cross-section of a house.

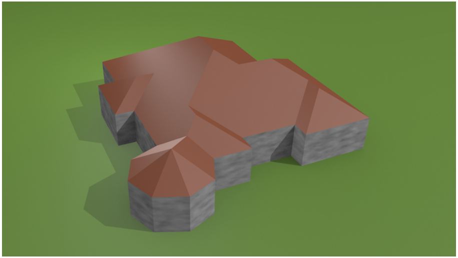


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Motivation: Automatic Roof Construction

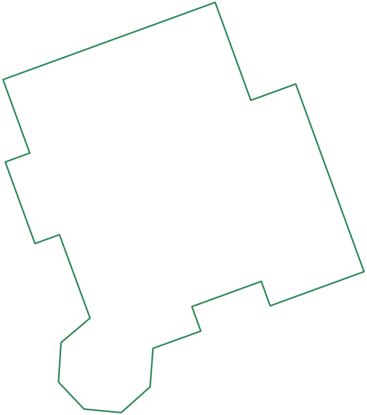
- Given is a simple polygon $\mathcal{P}$, which we consider as the cross-section of a house.
- Question: How can we compute a roof for $\mathcal{P}$?

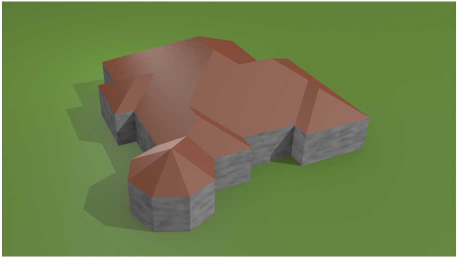




Motivation: Automatic Roof Construction

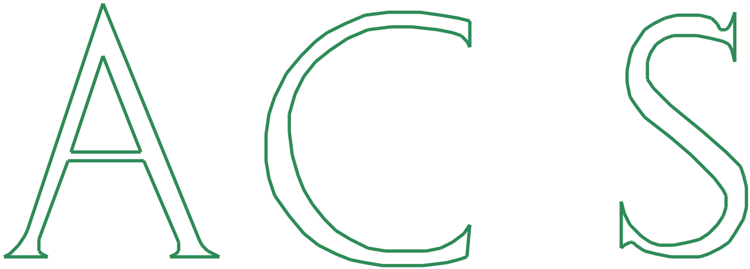
- Given is a simple polygon $\mathcal{P}$, which we consider as the cross-section of a house.
- Question: How can we compute a roof for $\mathcal{P}$?
- Answer: This can be done with the help of straight skeletons.





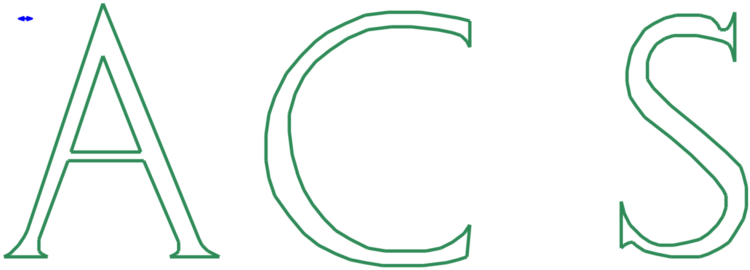
Motivation: Approximation of Polygonal Profiles

- How can we solve the following approximation problem?
  - For a set of planar (polygonal) profiles,



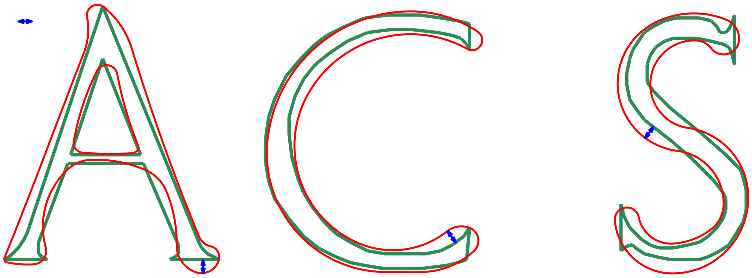
Motivation: Approximation of Polygonal Profiles

- How can we solve the following approximation problem?
  - For a set of planar (polygonal) profiles,
  - and an approximation threshold given,



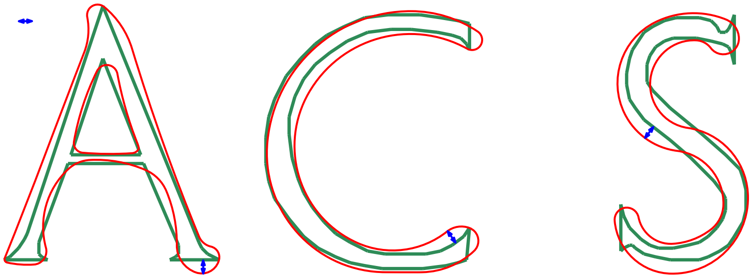
Motivation: Approximation of Polygonal Profiles

- How can we solve the following approximation problem?
  - For a set of planar (polygonal) profiles,
  - and an approximation threshold given,
  - compute an approximation such that the approximation threshold is not exceeded.



Motivation: Approximation of Polygonal Profiles

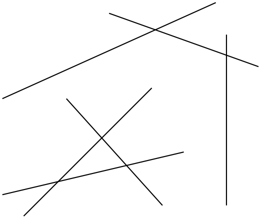
- How can we solve the following approximation problem?
  - For a set of planar (polygonal) profiles,
  - and an approximation threshold given,
  - compute an approximation such that the approximation threshold is not exceeded.



- Approximations can be obtained by biarc or B-spline curves, based on tolerance zones generated by means of Voronoi diagrams.

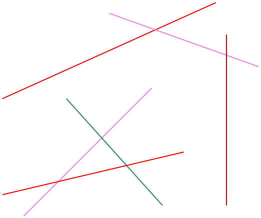
Motivation: Minimum Plane Partition

- CG:SHOP Geometric Optimization Challenge 2022: Given is a set S of n line segments in the plane.



Motivation: Minimum Plane Partition

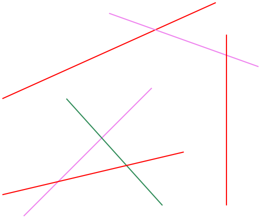
- CG:SHOP Geometric Optimization Challenge 2022: Given is a set S of n line segments in the plane.
- We seek a partitioning of S into a minimum number of k subsets $S_1, \ldots, S_k$ such that, for all $1 \leq i \leq k$, the line segments of S_i do not intersect pairwise.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Motivation: Minimum Plane Partition

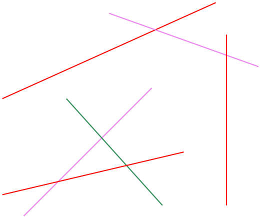
- CG:SHOP Geometric Optimization Challenge 2022: Given is a set S of n line segments in the plane.
- We seek a partitioning of S into a minimum number of k subsets $S_1, \dots, S_k$ such that, for all $1 \leq i \leq k$, the line segments of S_i do not intersect pairwise.



- Question: How can we check in $o(n^2)$ time whether any pair of line segments of S intersect? How can we determine all intersections efficiently?

Motivation: Minimum Plane Partition

- CG:SHOP Geometric Optimization Challenge 2022: Given is a set S of n line segments in the plane.
- We seek a partitioning of S into a minimum number of k subsets $S_1, \ldots, S_k$ such that, for all $1 \leq i \leq k$, the line segments of S_i do not intersect pairwise.

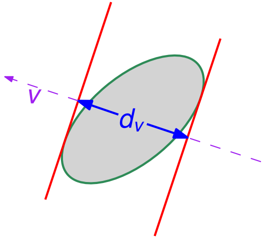


- Question: How can we check in $o(n^2)$ time whether any pair of line segments of S intersect? How can we determine all intersections efficiently?
- Answer: This can be done with the help of a plane sweep.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

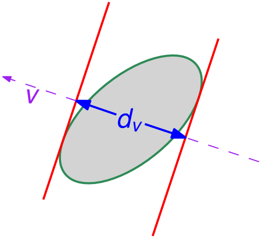
Geometric Intuition: Roundness of a Convex Shape

- We define the width of a planar shape relative to a direction vector v as the minimum distance d_v of its two parallel lines of support normal to the direction vector such that the shape is enclosed ("caliper probe").



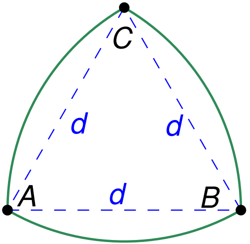
Geometric Intuition: Roundness of a Convex Shape

- We define the width of a planar shape relative to a direction vector v as the minimum distance d_v of its two parallel lines of support normal to the direction vector such that the shape is enclosed (“caliper probe”).



- Question: Can we conclude that the shape resembles a circle of diameter d if an arbitrarily large number of caliper probes all yield a uniform width d (irrespective of the direction vectors chosen)?

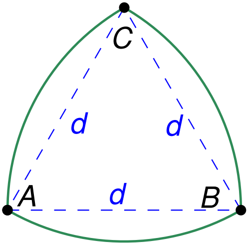
- Answer: No!! Even an infinite number of caliper probes all would yield a uniform width d for a Reuleaux triangle!

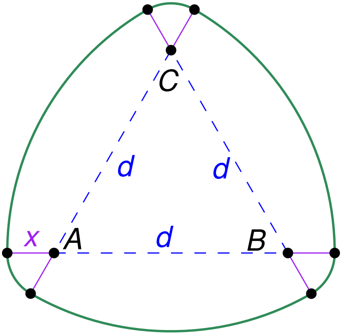


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Intuition: Roundness of a Convex Shape

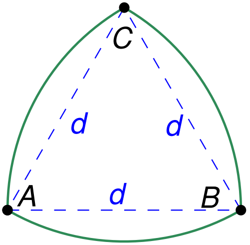
- Answer: No!! Even an infinite number of caliper probes all would yield a uniform width d for a Reuleaux triangle!

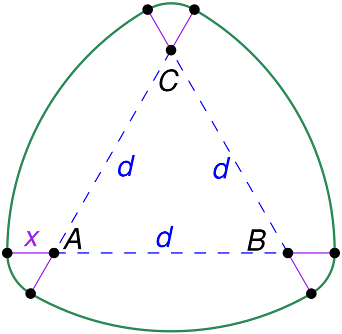




Geometric Intuition: Roundness of a Convex Shape

- Answer: No!! Even an infinite number of caliper probes all would yield a uniform width *d* for a Reuleaux triangle!

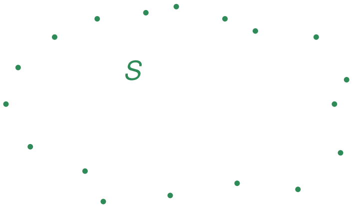




- Apparently, three caliper probes where applied when checking parts of the Challenger's solid-fuel booster rockets for roundness. (R. Feynman (1988): "What do you care what other people think?")

Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

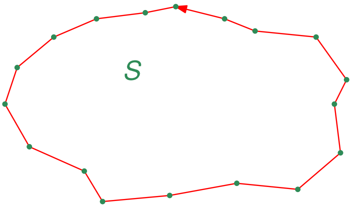
Input: A set S of n points in the Euclidean plane.



Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

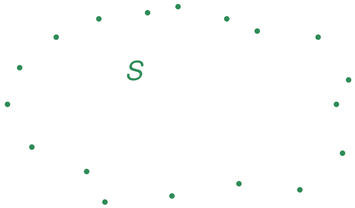


Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:

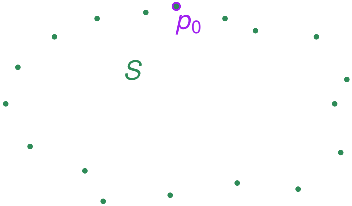


Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:
  1. Pick a point $p_0 \in S$.

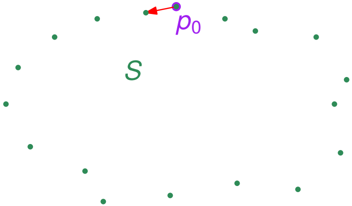


Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:
  1. Pick a point $p_0 \in S$.
  2. Find its nearest neighbor $p' \in S$, move to p', and let $p := p'$.



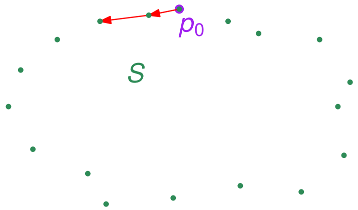
Geometric Intuition or “It’s Obvious!?”

Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:
  1. Pick a point $p_0 \in S$.
  2. Find its nearest neighbor $p' \in S$, move to p', and let $p := p'$.
  3. Continue from p to the nearest unvisited neighbor $p' \in S$ of p, and let $p := p'$.

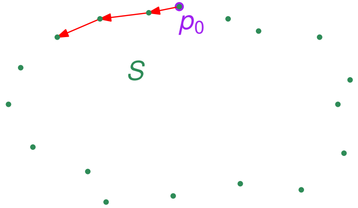


Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:
  1. Pick a point $p_0 \in S$.
  2. Find its nearest neighbor $p' \in S$, move to p', and let $p := p'$.
  3. Continue from p to the nearest unvisited neighbor $p' \in S$ of p, and let $p := p'$.
  4. Repeat the last step until all points have been visited, and return back to p_0.

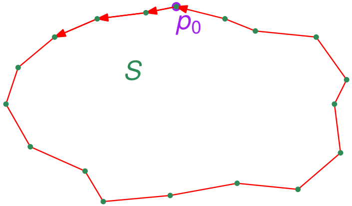


Problem: EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP)

Input: A set S of n points in the Euclidean plane.

Output: A cycle of minimal length that starts and ends in one point of S and visits all points of S.

- Natural strategy to solve an instance of ETSP:
  1. Pick a point $p_0 \in S$.
  2. Find its nearest neighbor $p' \in S$, move to p', and let $p := p'$.
  3. Continue from p to the nearest unvisited neighbor $p' \in S$ of p, and let $p := p'$.
  4. Repeat the last step until all points have been visited, and return back to p_0.



Geometric Intuition or "It's Obvious!?"

- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Intuition or “It’s Obvious!?”

- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn’t it?

Geometric Intuition or "It's Obvious!?"

- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn't it?
- Well . . .

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Intuition or "It's Obvious!?"

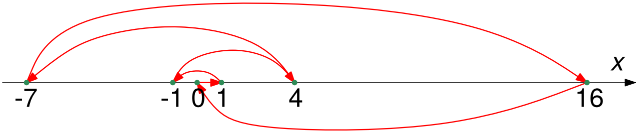
- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn't it?
- Well . . . The tour computed need not even be close in length to the optimum tour!

Geometric Intuition or “It’s Obvious!?”

- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn’t it?
- Well . . . The tour computed need not even be close in length to the optimum tour!

Geometric Intuition or “It’s Obvious!?”

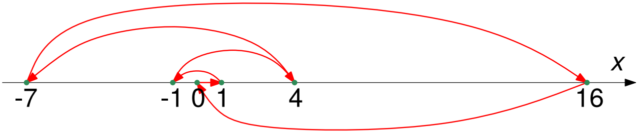
- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn’t it?
- Well . . . The tour computed need not even be close in length to the optimum tour!
- In the example, the tour computed has length 58, while the optimum tour has length 46!



- The strategy to always pick the shortest missing link can be seen as a *greedy* strategy.
- It is obvious that this strategy will always solve ETSP, isn’t it?
- Well . . . The tour computed need not even be close in length to the optimum tour!
- In the example, the tour computed has length 58, while the optimum tour has length 46!

Geometric intuition . . .

. . . is important, but may not replace formal reasoning. Intuition might misguide, and computational geometry without formal reasoning does not make sense.



1 **Introduction**

- Motivation
- History
- Notation
- Math Basics

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

History of Computational Geometry

- 1000 BCE: Length, area and volume are known for simple objects (cube, box, cylinder).
- Antiquity: Move from empirical mathematics to deductive mathematics.
- Thales of Milet ($\approx$ 600 BCE): He proved(!) that the two base angles of an isosceles triangle are identical.
- Euclid of Alexandria ($\approx$ 300 BCE): “The Elements”.
  - definitions,
  - five postulates,
  - five axioms,
  - 115 propositions.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

History of Computational Geometry

- Da Vinci (1452–1519) and others: Introduced perspective and projective geometry.
- Descartes (1596–1650) and P. de Fermat (1607–1665): Coordinates and the foundation of analytical geometry.
- Riemann (1826–1866): Differential geometry.
- Poincaré (1854–1912) and D. Hilbert (1862–1943): Axiom-based mathematics, proved consistency of axioms.

History of Computational Geometry

- Da Vinci (1452–1519) and others: Introduced perspective and projective geometry.
- Descartes (1596–1650) and P. de Fermat (1607–1665): Coordinates and the foundation of analytical geometry.
- Riemann (1826–1866): Differential geometry.
- Poincaré (1854–1912) and D. Hilbert (1862–1943): Axiom-based mathematics, proved consistency of axioms.

- Knuth: “The Art of Computer Programming” published 1968–1973.

History of Computational Geometry

- Da Vinci (1452–1519) and others: Introduced perspective and projective geometry.
- Descartes (1596–1650) and P. de Fermat (1607–1665): Coordinates and the foundation of analytical geometry.
- Riemann (1826–1866): Differential geometry.
- Poincaré (1854–1912) and D. Hilbert (1862–1943): Axiom-based mathematics, proved consistency of axioms.

- Knuth: “The Art of Computer Programming” published 1968–1973.

- Bézier, Forrest, Riesenfeld: Modeling of spline curves and surfaces called “computational geometry”.
- Minsky and Papert: Book entitled “Perceptrons” with chapter on “computational geometry”: Which geometric properties of a figure can be recognized with neural networks?

History of Computational Geometry

- Da Vinci (1452–1519) and others: Introduced perspective and projective geometry.
- Descartes (1596–1650) and P. de Fermat (1607–1665): Coordinates and the foundation of analytical geometry.
- Riemann (1826–1866): Differential geometry.
- Poincaré (1854–1912) and D. Hilbert (1862–1943): Axiom-based mathematics, proved consistency of axioms.

- Knuth: “The Art of Computer Programming” published 1968–1973.

- Bézier, Forrest, Riesenfeld: Modeling of spline curves and surfaces called “computational geometry”.
- Minsky and Papert: Book entitled “Perceptrons” with chapter on “computational geometry”: Which geometric properties of a figure can be recognized with neural networks?

Birth of today’s computational geometry

Shamos (1978): PhD thesis “Computational Geometry” at Yale University, USA.

Notation

- Numbers:
  - The set $\{1, 2, 3, \ldots\}$ of natural numbers is denoted by $\mathbb{N}$, with $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.
  - The set $\{2, 3, 5, 7, 11, 13, \ldots\} \subset \mathbb{N}$ of prime numbers is denoted by $\mathbb{P}$.
  - The (positive and negative) integers are denoted by $\mathbb{Z}$.
  - $\mathbb{Z}_n := \{0, 1, 2, \ldots, n-1\}$ and $\mathbb{Z}_n^+ := \{1, 2, \ldots, n-1\}$ for $n \in \mathbb{N}$.
  - The reals are denoted by $\mathbb{R}$; the non-negative reals are denoted by $\mathbb{R}_0^+$, and the positive reals by $\mathbb{R}^+$.

Notation

- Numbers:
  - The set $\{1, 2, 3, \ldots\}$ of natural numbers is denoted by $\mathbb{N}$, with $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.
  - The set $\{2, 3, 5, 7, 11, 13, \ldots\} \subset \mathbb{N}$ of prime numbers is denoted by $\mathbb{P}$.
  - The (positive and negative) integers are denoted by $\mathbb{Z}$.
  - $\mathbb{Z}_n := \{0, 1, 2, \ldots, n-1\}$ and $\mathbb{Z}_n^+ := \{1, 2, \ldots, n-1\}$ for $n \in \mathbb{N}$.
  - The reals are denoted by $\mathbb{R}$; the non-negative reals are denoted by $\mathbb{R}_0^+$, and the positive reals by $\mathbb{R}^+$.
- Open or closed intervals $I \subset \mathbb{R}$ are denoted using square brackets: e.g., $I_1 = [a_1, b_1]$ or $I_2 = [a_2, b_2[$, with $a_1, a_2, b_1, b_2 \in \mathbb{R}$, where the right-hand "[" indicates that the value b_2 is not included in I_2.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Notation

- Numbers:
  - The set $\{1, 2, 3, \ldots\}$ of natural numbers is denoted by $\mathbb{N}$, with $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.
  - The set $\{2, 3, 5, 7, 11, 13, \ldots\} \subset \mathbb{N}$ of prime numbers is denoted by $\mathbb{P}$.
  - The (positive and negative) integers are denoted by $\mathbb{Z}$.
  - $\mathbb{Z}_n := \{0, 1, 2, \ldots, n-1\}$ and $\mathbb{Z}_n^+ := \{1, 2, \ldots, n-1\}$ for $n \in \mathbb{N}$.
  - The reals are denoted by $\mathbb{R}$; the non-negative reals are denoted by $\mathbb{R}_0^+$, and the positive reals by $\mathbb{R}^+$.
- Open or closed intervals $I \subset \mathbb{R}$ are denoted using square brackets: e.g., $I_1 = [a_1, b_1]$ or $I_2 = [a_2, b_2[$, with $a_1, a_2, b_1, b_2 \in \mathbb{R}$, where the right-hand "[" indicates that the value b_2 is not included in I_2.
- The set of all elements $a \in A$ with property $P(a)$, for some set A and some predicate P, is denoted by

$$\{x \in A : \ P(x)\} \quad \text{or} \quad \{x : x \in A \wedge \ P(x)\}$$

or

$$\{x \in A \mid P(x)\} \quad \text{or} \quad \{x \mid x \in A \wedge \ P(x)\}.$$

Notation

- Numbers:
  - The set $\{1, 2, 3, \ldots\}$ of natural numbers is denoted by $\mathbb{N}$, with $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.
  - The set $\{2, 3, 5, 7, 11, 13, \ldots\} \subset \mathbb{N}$ of prime numbers is denoted by $\mathbb{P}$.
  - The (positive and negative) integers are denoted by $\mathbb{Z}$.
  - $\mathbb{Z}_n := \{0, 1, 2, \ldots, n-1\}$ and $\mathbb{Z}_n^+ := \{1, 2, \ldots, n-1\}$ for $n \in \mathbb{N}$.
  - The reals are denoted by $\mathbb{R}$; the non-negative reals are denoted by $\mathbb{R}_0^+$, and the positive reals by $\mathbb{R}^+$.
- Open or closed intervals $I \subset \mathbb{R}$ are denoted using square brackets: e.g., $I_1 = [a_1, b_1]$ or $I_2 = [a_2, b_2[$, with $a_1, a_2, b_1, b_2 \in \mathbb{R}$, where the right-hand "[" indicates that the value b_2 is not included in I_2.
- The set of all elements $a \in A$ with property $P(a)$, for some set A and some predicate P, is denoted by

  $$\{x \in A : \ P(x)\} \quad \text{or} \quad \{x : x \in A \wedge \ P(x)\}$$

  or

  $$\{x \in A \mid P(x)\} \quad \text{or} \quad \{x \mid x \in A \wedge \ P(x)\}.$$

- Bold capital letters, such as **M**, are used for matrices.
- The set of all (real) $m \times n$ matrices is denoted by $M_{m \times n}$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Notation

- Points are denoted by letters written in italics: p, q or, occasionally, P, Q. We do not distinguish between a point and its position vector.
- The coordinates of a vector are denoted by using indices (or numbers): e.g., $v = (v_x, v_y)$ for $v \in \mathbb{R}^2$, or $v = (v_1, v_2, \ldots, v_n)$ for $v \in \mathbb{R}^n$.
- In order to state $v \in \mathbb{R}^n$ in vector form we will mix column and row vectors freely unless a specific form is required, such as for matrix multiplication.

Notation

- Points are denoted by letters written in italics: p, q or, occasionally, P, Q. We do not distinguish between a point and its position vector.
- The coordinates of a vector are denoted by using indices (or numbers): e.g., $v = (v_x, v_y)$ for $v \in \mathbb{R}^2$, or $v = (v_1, v_2, \ldots, v_n)$ for $v \in \mathbb{R}^n$.
- In order to state $v \in \mathbb{R}^n$ in vector form we will mix column and row vectors freely unless a specific form is required, such as for matrix multiplication.
- The vector dot product of two vectors $v, w \in \mathbb{R}^n$ is denoted by $\langle v, w \rangle$. That is, $\langle v, w \rangle = \sum_{i=1}^{n} v_i \cdot w_i$ for $v, w \in \mathbb{R}^n$.
- The vector cross-product (in $\mathbb{R}^3$) is denoted by a cross: $v \times w$.
- The length of a vector v is denoted by $\|v\|$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Notation

- Points are denoted by letters written in italics: p, q or, occasionally, P, Q. We do not distinguish between a point and its position vector.
- The coordinates of a vector are denoted by using indices (or numbers): e.g., $v = (v_x, v_y)$ for $v \in \mathbb{R}^2$, or $v = (v_1, v_2, \ldots, v_n)$ for $v \in \mathbb{R}^n$.
- In order to state $v \in \mathbb{R}^n$ in vector form we will mix column and row vectors freely unless a specific form is required, such as for matrix multiplication.
- The vector dot product of two vectors $v, w \in \mathbb{R}^n$ is denoted by $\langle v, w \rangle$. That is, $\langle v, w \rangle = \sum_{i=1}^{n} v_i \cdot w_i$ for $v, w \in \mathbb{R}^n$.
- The vector cross-product (in $\mathbb{R}^3$) is denoted by a cross: $v \times w$.
- The length of a vector v is denoted by $\|v\|$.
- The straight-line segment between the points p and q is denoted by $\overline{pq}$.
- The supporting line of the points p and q is denoted by $\ell(p, q)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Polygonal Curve

Definition 1 (Polygonal curve, Dt.: Polygonzug)

Consider the sequence of points $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, for some $d, n \in \mathbb{N}_0$. The *polygonal curve* (or *polygonal chain*, *polygonal profile*) specified by these points ("vertices") is given by $\gamma\colon [0, n] \to \mathbb{R}^d$ with

$$\gamma(t) := p_i + (t - i) \cdot (p_{i+1} - p_i) \quad \text{if } t \in [i, i+1] \text{ for } i \in \{1, 2, \ldots, n-1\}.$$

Polygonal Curve

Definition 1 (Polygonal curve, Dt.: Polygonzug)

Consider the sequence of points $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, for some $d, n \in \mathbb{N}_0$. The *polygonal curve* (or *polygonal chain*, *polygonal profile*) specified by these points ("vertices") is given by $\gamma\colon [0, n] \to \mathbb{R}^d$ with

$$\gamma(t) := p_i + (t - i) \cdot (p_{i+1} - p_i) \quad \text{if } t \in [i, i+1] \text{ for } i \in \{1, 2, \ldots, n-1\}.$$

- Hence, a polygonal curve is a sequence of finitely many vertices connected by straight-line segments such that each segment (except for the first) starts at the end of the previous segment.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Polygonal Curve

Definition 1 (Polygonal curve, Dt.: Polygonzug)

Consider the sequence of points $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, for some $d, n \in \mathbb{N}_0$. The *polygonal curve* (or *polygonal chain*, *polygonal profile*) specified by these points ("vertices") is given by $\gamma\colon [0, n] \to \mathbb{R}^d$ with

$$\gamma(t) := p_i + (t - i) \cdot (p_{i+1} - p_i) \quad \text{if } t \in [i, i+1] \text{ for } i \in \{1, 2, \ldots, n-1\}.$$

- Hence, a polygonal curve is a sequence of finitely many vertices connected by straight-line segments such that each segment (except for the first) starts at the end of the previous segment.
- Unless stated otherwise, we will always assume that all vertices of a polygonal curve are co-planar, i.e., that the polygonal curve is plane. The default plane is $\mathbb{R}^2$.

Polygonal Curve

Definition 1 (Polygonal curve, Dt.: Polygonzug)

Consider the sequence of points $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, for some $d, n \in \mathbb{N}_0$. The *polygonal curve* (or *polygonal chain*, *polygonal profile*) specified by these points ("vertices") is given by $\gamma\colon [0, n] \to \mathbb{R}^d$ with

$$\gamma(t) := p_i + (t - i) \cdot (p_{i+1} - p_i) \quad \text{if } t \in [i, i+1] \text{ for } i \in \{1, 2, \ldots, n-1\}.$$

- Hence, a polygonal curve is a sequence of finitely many vertices connected by straight-line segments such that each segment (except for the first) starts at the end of the previous segment.
- Unless stated otherwise, we will always assume that all vertices of a polygonal curve are co-planar, i.e., that the polygonal curve is plane. The default plane is $\mathbb{R}^2$.
- It is common to extend this definition by allowing $n = 0$, in which case we get a single point.

Definition 2 (Polygon)

For $n \in \mathbb{N}$ with $n \geq 3$, a *polygon* with vertices $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, aka *n-gon*, is a polygonal curve such that $p_0 = p_n$.

Definition 2 (Polygon)

For $n \in \mathbb{N}$ with $n \geq 3$, a *polygon* with vertices $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, aka *n-gon*, is a polygonal curve such that $p_0 = p_n$.

Definition 3 (Simple polygon, Dt.: einfaches Polygon)

An n-gon is *simple* if it admits a simple parametrization.

Definition 2 (Polygon)

For $n \in \mathbb{N}$ with $n \geq 3$, a *polygon* with vertices $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, aka *n-gon*, is a polygonal curve such that $p_0 = p_n$.

Definition 3 (Simple polygon, Dt.: einfaches Polygon)

An n-gon is *simple* if it admits a simple parametrization.

- If a plane polygon $\mathcal{P}$ is simple then, by the Jordan Curve Theorem, it splits the plane into two regions, one of which is bounded.
- In this case it is common to be a bit liberal and use the term “polygon” for either the (simple) polygonal curve $\mathcal{P}$ or for the entire region bounded by $\mathcal{P}$; the actual meaning has to be inferred from the context.

Definition 2 (Polygon)

For $n \in \mathbb{N}$ with $n \geq 3$, a *polygon* with vertices $p_0, p_1, p_2, \ldots, p_n \in \mathbb{R}^d$, aka *n-gon*, is a polygonal curve such that $p_0 = p_n$.

Definition 3 (Simple polygon, Dt.: einfaches Polygon)

An *n*-gon is *simple* if it admits a simple parametrization.

- If a plane polygon $\mathcal{P}$ is simple then, by the Jordan Curve Theorem, it splits the plane into two regions, one of which is bounded.
- In this case it is common to be a bit liberal and use the term "polygon" for either the (simple) polygonal curve $\mathcal{P}$ or for the entire region bounded by $\mathcal{P}$; the actual meaning has to be inferred from the context.
- If $\mathcal{P}$ is regarded to be only the simple polygonal curve then the bounded region (without $\mathcal{P}$ itself) is called the polygon's *interior*, and points within that region are said to be *inside* of $\mathcal{P}$.

Planar Straight-Line Graph

Definition 4 (Planar straight-line graph)

A *planar straight-line graph* (PSLG) is a finite collection of isolated vertices and straight-line segments such that

- each two segments intersect only in vertices shared by both of them,
- no segment passes through a vertex other than one of its two end-points.

Planar Straight-Line Graph

Definition 4 (Planar straight-line graph)

A *planar straight-line graph* (PSLG) is a finite collection of isolated vertices and straight-line segments such that

- each two segments intersect only in vertices shared by both of them,
- no segment passes through a vertex other than one of its two end-points.

- Hence, a PSLG is an embedding of a planar graph such that all its edges are drawn as straight-line segments.

Planar Straight-Line Graph

Definition 4 (Planar straight-line graph)

A *planar straight-line graph* (PSLG) is a finite collection of isolated vertices and straight-line segments such that

- each two segments intersect only in vertices shared by both of them,
- no segment passes through a vertex other than one of its two end-points.

- Hence, a PSLG is an embedding of a planar graph such that all its edges are drawn as straight-line segments.
- Aka: Plane geometric graph.

Planar Straight-Line Graph

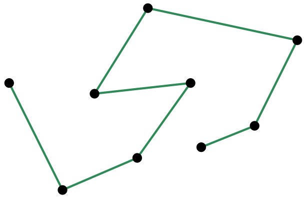
Definition 4 (Planar straight-line graph)

A *planar straight-line graph* (PSLG) is a finite collection of isolated vertices and straight-line segments such that

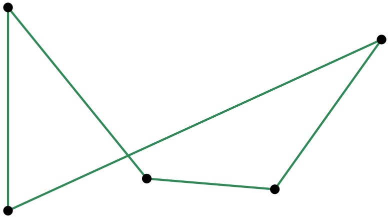
- each two segments intersect only in vertices shared by both of them,
- no segment passes through a vertex other than one of its two end-points.

- Hence, a PSLG is an embedding of a planar graph such that all its edges are drawn as straight-line segments.
- Aka: Plane geometric graph.
- Hence, simple polygonal curves and simple polygons are special PSLGs.
- Of course, Euler's Theorem applies to the faces, edges and vertices of a PSLG.

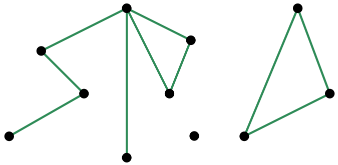
Sample Polygonal Chains and PSLGs



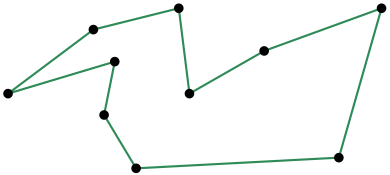
polygonal curve



polygon, not simple



planar straight-line graph



simple polygon

Polygonal Region

Definition 5 (Polygonal region)

A *polygonal region* is a (possibly) multiply-connected but connected subset of $\mathbb{R}^2$ that is bounded by k simple polygons $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, for some $k \in \mathbb{N}$, such that

1. no pair of polygons (seen as curves) intersect,
2. the polygons $\mathcal{P}_2, \ldots, \mathcal{P}_k$ lie in the interior of $\mathcal{P}_1$,
3. for $2 \leq i, j \leq k$, the polygon $\mathcal{P}_i$ does not lie in the interior of the polygon $\mathcal{P}_j$.

The polygon $\mathcal{P}_1$ is called *outer polygon* and the polygons $\mathcal{P}_2, \ldots, \mathcal{P}_k$ are called *islands* or *holes*.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

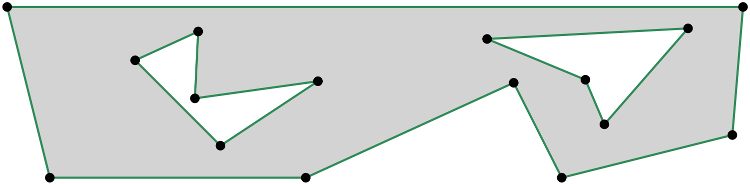
Polygonal Region

Definition 5 (Polygonal region)

A *polygonal region* is a (possibly) multiply-connected but connected subset of $\mathbb{R}^2$ that is bounded by k simple polygons $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, for some $k \in \mathbb{N}$, such that

1. no pair of polygons (seen as curves) intersect,
2. the polygons $\mathcal{P}_2, \ldots, \mathcal{P}_k$ lie in the interior of $\mathcal{P}_1$,
3. for $2 \leq i, j \leq k$, the polygon $\mathcal{P}_i$ does not lie in the interior of the polygon $\mathcal{P}_j$.

The polygon $\mathcal{P}_1$ is called *outer polygon* and the polygons $\mathcal{P}_2, \ldots, \mathcal{P}_k$ are called *islands* or *holes*.



Polyhedron

Definition 6 (Polyhedron, Dt.: Polyeder)

A *polyhedron* in $\mathbb{R}^3$ is either

- a (possibly unbounded) solid given by the intersection of finitely many halfspaces,

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 6 (Polyhedron, Dt.: Polyeder)

A *polyhedron* in $\mathbb{R}^3$ is either

- a (possibly unbounded) solid given by the intersection of finitely many halfspaces, or
- a connected bounded solid whose (closed manifold) boundary is formed by a finite collection of plane polygons ("faces") such that
  1. each vertex is incident to at least three edges and faces,
  2. each edge is shared by exactly two faces,
  3. each two faces intersect only in vertices and edges shared by both of them,
  4. the faces that share a vertex form a cyclic chain of polygons in which every pair of consecutive polygons shares an edge.

Polyhedron

Definition 6 (Polyhedron, Dt.: Polyeder)

A *polyhedron* in $\mathbb{R}^3$ is either

- a (possibly unbounded) solid given by the intersection of finitely many halfspaces, or
- a connected bounded solid whose (closed manifold) boundary is formed by a finite collection of plane polygons (“faces”) such that
  1. each vertex is incident to at least three edges and faces,
  2. each edge is shared by exactly two faces,
  3. each two faces intersect only in vertices and edges shared by both of them,
  4. the faces that share a vertex form a cyclic chain of polygons in which every pair of consecutive polygons shares an edge.

- Note: Plural of “polyhedron” is “polyhedra”.

Definition 6 (Polyhedron, Dt.: Polyeder)

A *polyhedron* in $\mathbb{R}^3$ is either

- a (possibly unbounded) solid given by the intersection of finitely many halfspaces, or
- a connected bounded solid whose (closed manifold) boundary is formed by a finite collection of plane polygons ("faces") such that
  1. each vertex is incident to at least three edges and faces,
  2. each edge is shared by exactly two faces,
  3. each two faces intersect only in vertices and edges shared by both of them,
  4. the faces that share a vertex form a cyclic chain of polygons in which every pair of consecutive polygons shares an edge.

- Note: Plural of "polyhedron" is "polyhedra".
- Recall that Euler's Formula $v - e + f = 2$ holds for the vertices, edges and faces of a polyhedron.

Polyhedron

- Unfortunately, even in $\mathbb{R}^3$ there there is no universal agreement over how to define the analogue to a polygon in $\mathbb{R}^3$ ...
- The situation gets worse once different fields of mathematics and computer science are considered!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Polyhedron

- Unfortunately, even in $\mathbb{R}^3$ there there is no universal agreement over how to define the analogue to a polygon in $\mathbb{R}^3$ ...
- The situation gets worse once different fields of mathematics and computer science are considered!

Grünbaum (1994)

"The Original Sin in the theory of polyhedra goes back to Euclid, ... and many others, ... at each stage ... the writers failed to define what are the polyhedra."

Polyhedron

- Unfortunately, even in $\mathbb{R}^3$ there there is no universal agreement over how to define the analogue to a polygon in $\mathbb{R}^3$ ...
- The situation gets worse once different fields of mathematics and computer science are considered!

Grünbaum (1994)

"The Original Sin in the theory of polyhedra goes back to Euclid, ... and many others, ... at each stage ... the writers failed to define what are the polyhedra."

Polyhedron versus Polytope

1. For convex solids, some authors (in some fields of mathematics) prefer to use the term "polytope" for a bounded polyhedron, whereas "polyhedron" is a generic convex object.
2. From this point of view, a polyhedron is the intersection of a finite number of halfspaces and is defined by its faces whereas a polytope is the convex hull of a finite number of points and is defined by its vertices.

Logarithms

Definition 7 (Logarithm)

The *logarithm* of a positive real number $x \in \mathbb{R}^+$ with respect to a base b, which is a positive real number not equal to 1, is the unique solution y of the equation $b^y = x$. It is denoted by $\log_b x$.

- Hence, it is the exponent by which b must be raised to yield x.
- Common bases:

$$\operatorname{ld} x := \log_2 x \qquad \ln x := \log_e x \quad \text{with} \quad e := \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \approx 2.71828\ldots$$

Lemma 8

Let $x, y, p \in \mathbb{R}^+$ and $b \in \mathbb{R}^+ \setminus \{1\}$.

$$\log_b(xy) = \log_b(x) + \log_b(y) \qquad \log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$

$$\log_b\left(x^p\right) = p\log_b(x) \qquad \log_b\left(\sqrt[p]{x}\right) = \frac{\log_b(x)}{p}$$

Logarithms

Lemma 9 (Change of base)

Let $x \in \mathbb{R}^+$ and $\alpha, \beta \in \mathbb{R}^+ \setminus \{1\}$. Then $\log_\alpha(x)$ and $\log_\beta(x)$ differ only by a multiplicative constant:

$$\log_\alpha(x) = \frac{1}{\log_\beta(\alpha)} \cdot \log_\beta(x)$$

Convention

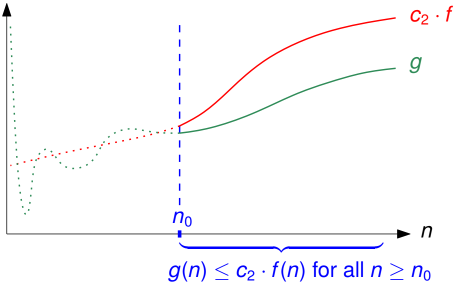
In this course, $\log n$ will always denote the logarithm of n to the base 2, i.e., $\log n := \log_2 n$.

Asymptotic Notation: Big-O

Definition 10 (Big-O, Dt.: Groß-O)

Let $f\colon \mathbb{N} \to \mathbb{R}^+$. Then the set $O(f)$ is defined as

$$O(f) \;:=\; \{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \exists c_2 \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad g(n) \leq c_2 \cdot f(n)\}\,.$$



- Equivalent definition used by some authors:

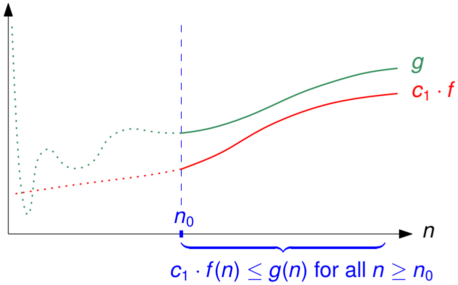
$$O(f) \;:=\; \left\{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \exists c_2 \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad \frac{g(n)}{f(n)} \leq c_2\right\}.$$

Asymptotic Notation: Big-Omega

Definition 11 (Big-Omega, Dt.: Groß-Omega)

Let $f\colon \mathbb{N} \to \mathbb{R}^+$. Then the set $\Omega(f)$ is defined as

$$\Omega(f) \;:=\; \{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \exists c_1 \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad c_1 \cdot f(n) \leq g(n)\}\,.$$



- Equivalently,

$$\Omega(f) \;:=\; \left\{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \exists c_1 \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad c_1 \leq \frac{g(n)}{f(n)}\right\}.$$

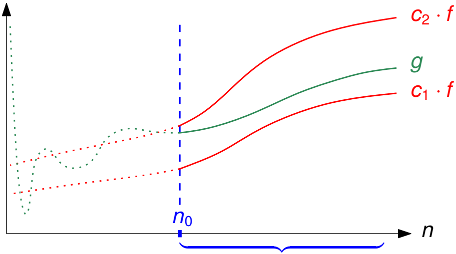
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Asymptotic Notation: Big-Theta

Definition 12 (Big-Theta, Dt.: Groß-Theta)

Let $f\colon \mathbb{N} \to \mathbb{R}^+$. Then the set $\Theta(f)$ is defined as

$$\Theta(f) := \{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \quad c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}.$$



$c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$

which is equivalent to $c_1 \leq \frac{g(n)}{f(n)} \leq c_2$ for all $n \geq n_0$

Asymptotic Notation: Small-Oh and Small-Omega

Definition 13 (Small-Oh, Dt.: Klein-O)

Let $f\colon \mathbb{N} \to \mathbb{R}^+$. Then the set $o(f)$ is defined as

$$o(f) := \{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \forall c \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad g(n) \leq c \cdot f(n)\}.$$

Definition 14 (Small-Omega, Dt.: Klein-Omega)

Let $f\colon \mathbb{N} \to \mathbb{R}^+$. Then the set $\omega(f)$ is defined as

$$\omega(f) := \{g\colon \mathbb{N} \to \mathbb{R}^+ \mid \quad \forall c \in \mathbb{R}^+ \quad \exists n_0 \in \mathbb{N} \quad \forall n \geq n_0 \qquad g(n) \geq c \cdot f(n)\}.$$

- We can extend Defs. 10–14 such that $\mathbb{N}_0$ rather than $\mathbb{N}$ is taken as the domain (Dt.: Definitionsmenge). We can also replace the codomain (Dt.: Zielbereich) $\mathbb{R}^+$ by $\mathbb{R}_0^+$ (or even $\mathbb{R}$) provided that all functions are eventually positive.

Warning

The use of the equality operator "$=$" instead of the set operators "$\in$" or "$\subseteq$" to denote set membership or a subset relation is a *common abuse of notation*.

Master Theorem

Theorem 15

Consider constants $n_0 \in \mathbb{N}$ and $a \in \mathbb{N}$, $b \in \mathbb{R}$ with $b > 1$, and a function $f\colon \mathbb{N} \to \mathbb{R}_0^+$.
Let $T\colon \mathbb{N} \to \mathbb{R}_0^+$ be an eventually non-decreasing function such that

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

for all $n \in \mathbb{N}$ with $n \geq n_0$, where we interpret $T(\frac{n}{b})$ as (a combination of) $T(\lceil \frac{n}{b} \rceil)$ or $T(\lfloor \frac{n}{b} \rfloor)$.
Then we have

$$T \in \begin{cases} \Theta(f) & \text{if } \begin{cases} f \in \Omega(n^{(\log_b a)+\varepsilon}) \text{ for some } \varepsilon \in \mathbb{R}^+, \\ \text{and if the following regularity condition holds} \\ \text{for some } 0 < s < 1 \text{ and all sufficiently large n:} \\ \qquad a \cdot f(^n/_b) \leq s \cdot f(n), \end{cases} \\ \Theta\left(n^{\log_b a} \log n\right) & \text{if } f \in \Theta(n^{\log_b a}), \\ \Theta(n^{\log_b a}) & \text{if } f \in O(n^{(\log_b a)-\varepsilon}) \text{ for some } \varepsilon \in \mathbb{R}^+. \end{cases}$$

- This is a simplified version of the Akra-Bazzi Theorem [Akra&Bazzi 1998].

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Fibonacci Numbers

Definition 16 (Fibonacci numbers)

For all $n \in \mathbb{N}_0$,

$$F_n := \begin{cases} n & \text{if } n \leq 1, \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610

Lemma 17

For $n \in \mathbb{N}$ with $n \geq 2$:

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n \geq \left(\frac{1+\sqrt{5}}{2}\right)^{n-2}$$

- Lots of interesting mathematical properties. For instance,

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi, \quad \text{where } \phi := \frac{1+\sqrt{5}}{2} = 1.618\ldots \text{ is the } \textit{golden ratio}.$$

Catalan Numbers

Definition 18 (Catalan numbers)

For $n \in \mathbb{N}_0$,

$$C_0 := 1 \quad \text{and} \quad C_{n+1} := \sum_{i=0}^{n} C_i \cdot C_{n-i}.$$

n	0	1	2	3	4	5	6	7	8	9	10	11
C_n	1	1	2	5	14	42	132	429	1430	4862	16796	58786

Lemma 19

For $n \in \mathbb{N}_0$,

$$C_n = \frac{1}{n+1} \sum_{i=0}^{n} \binom{n}{i}^2 = \frac{1}{n+1} \binom{2n}{n} \in \Theta\left(\frac{4^n}{n^{1.5}}\right).$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 20 (Harmonic numbers)

For $n \in \mathbb{N}$,

$$H_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}.$$

Lemma 21

The sequence $s\colon \mathbb{N} \to \mathbb{R}$ with

$$s_n := H_n - \ln n$$

is monotonically decreasing and convergent. Its limit is the Euler-Mascheroni constant

$$\gamma := \lim_{n\to+\infty} (H_n - \ln n) \approx 0.5772\ldots,$$

and we have

$$\ln n < H_n - \gamma < \ln(n+1), \qquad \text{i.e.} \quad H_n \in \Theta(\ln) = \Theta(\log).$$

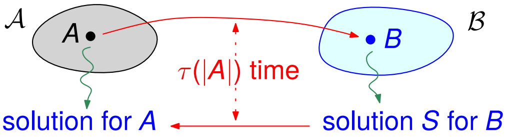
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Reduction of a Problem

Definition 22 (Reduction)

A problem $\mathcal{A}$ can be *reduced* (or *transformed*) to a problem $\mathcal{B}$ if

1. every instance A of $\mathcal{A}$ can be converted to an instance B of $\mathcal{B}$,
2. a solution S for B can be computed, and
3. S can be transformed back into a correct solution for A.



Definition 23

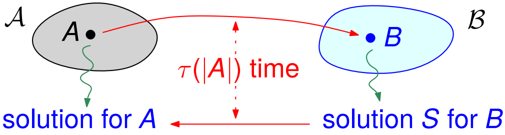
A problem $\mathcal{A}$ is τ-*reducible* to $\mathcal{B}$, denoted by $\mathcal{A} \leq_\tau \mathcal{B}$, if

1. $\mathcal{A}$ can be reduced to $\mathcal{B}$,
2. for any instance A of $\mathcal{A}$, steps 1 and 3 of the reduction can be carried out in at most $\tau(|A|)$ time, where $|A|$ denotes the input size of A.

Transfer of Complexity Bounds

Lemma 24 (Upper bound via reduction)

Suppose that $\mathcal{A}$ is τ-reducible to $\mathcal{B}$ such that the order of the input size is preserved. If problem $\mathcal{B}$ can be solved in $O(T)$ time, then $\mathcal{A}$ can be solved in at most $O(T+\tau)$ time.



Lemma 25 (Lower bound via reduction)

Suppose that $\mathcal{A}$ is τ-reducible to $\mathcal{B}$ such that the order of the input size is preserved. If problem $\mathcal{A}$ is known to require $\Omega(T)$ time, then $\mathcal{B}$ requires at least $\Omega(T-\tau)$ time.

2 Geometric Concepts and Paradigms

- Plane Sweep
- Arrangements
- Point-Line Duality

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

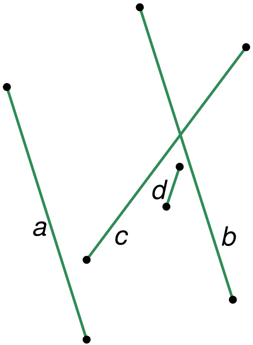
2 Geometric Concepts and Paradigms

- Plane Sweep
  - Line Segment Intersection
  - Bentley-Ottmann Algorithm for Line Segment Detection
  - Boolean Operations on Curvilinear Polygons
- Arrangements
- Point-Line Duality

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Problem: LINESEGMENTINTERSECTION

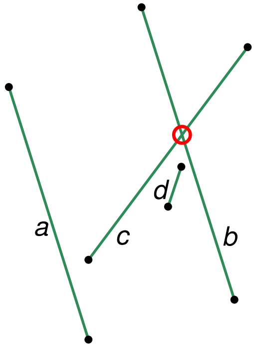
Given: A set S of line segments in $\mathbb{R}^2$.



Problem: LINESEGMENTINTERSECTION

Given: A set S of line segments in $\mathbb{R}^2$.

Decide: Do any two segments of S intersect?



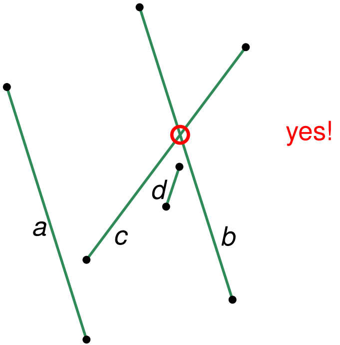
yes!

Problem: LineSegmentIntersection

Given: A set S of line segments in $\mathbb{R}^2$.

Decide: Do any two segments of S intersect?

- LineSegmentIntersection does not require us to find and report one or all intersections.

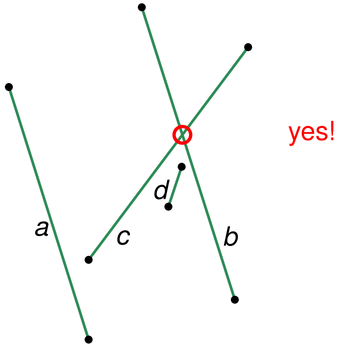


Problem: LineSegmentIntersection

Given: A set S of line segments in $\mathbb{R}^2$.

Decide: Do any two segments of S intersect?

- LineSegmentIntersection does not require us to find and report one or all intersections.
- Still, we explain how all intersections can be found.
- Stopping the algorithm at the first intersection (if one exists) yields an answer to the original problem.



Theorem 26 (Bentley&Ottmann (1979))

All k intersections among n line segments in $\mathbb{R}^2$ can be detected in $O((n+k)\log n)$ time and $O(n)$ space, using a plane-sweep algorithm.

Theorem 26 (Bentley&Ottmann (1979))

All k intersections among n line segments in $\mathbb{R}^2$ can be detected in $O((n+k)\log n)$ time and $O(n)$ space, using a plane-sweep algorithm.

Corollary 27

LINESEGMENTINTERSECTION can be solved in optimal $O(n \log n)$ time and $O(n)$ space for n line segments.

Theorem 26 (Bentley&Ottmann (1979))

All k intersections among n line segments in $\mathbb{R}^2$ can be detected in $O((n+k)\log n)$ time and $O(n)$ space, using a plane-sweep algorithm.

Corollary 27

LINESEGMENTINTERSECTION can be solved in optimal $O(n \log n)$ time and $O(n)$ space for n line segments.

- Note that n line segments may yield $\Theta(n^2)$ many intersections. Hence, $k \in O(n^2)$, and in the worst case the Bentley-Ottmann algorithm runs in $\Theta(n^2 \log n)$ time.

Line Segment Intersection

Theorem 26 (Bentley&Ottmann (1979))

All k intersections among n line segments in $\mathbb{R}^2$ can be detected in $O((n+k)\log n)$ time and $O(n)$ space, using a plane-sweep algorithm.

Corollary 27

LINESEGMENTINTERSECTION can be solved in optimal $O(n \log n)$ time and $O(n)$ space for n line segments.

- Note that n line segments may yield $\Theta(n^2)$ many intersections. Hence, $k \in O(n^2)$, and in the worst case the Bentley-Ottmann algorithm runs in $\Theta(n^2 \log n)$ time.
- [Chazelle&Edelsbrunner (1992)] explain how to detect all k intersections in $O(k + n \log n)$ time, using $O(n+k)$ space.
- [Balaban (1995)] improves this to $O(k + n \log n)$ time and $O(n)$ space.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Line Segment Intersection: General Position Assumed

General position assumed

For the sake of descriptional simplicity, we assume that

- no two end-points or intersections of line segments of S have the same y-coordinate;
- no two line segments overlap;
- no three line segments intersect at the same point;
- no segment is horizontal.

General position assumed

For the sake of descriptional simplicity, we assume that

- no two end-points or intersections of line segments of S have the same y-coordinate;
- no two line segments overlap;
- no three line segments intersect at the same point;
- no segment is horizontal.

- A GPA assumption makes perfect sense since it allows to avoid special cases and, thus, to focus on the essential ideas of a (geometric) algorithm.

Line Segment Intersection: General Position Assumed

General position assumed

For the sake of descriptional simplicity, we assume that

- no two end-points or intersections of line segments of S have the same y-coordinate;
- no two line segments overlap;
- no three line segments intersect at the same point;
- no segment is horizontal.

- A GPA assumption makes perfect sense since it allows to avoid special cases and, thus, to focus on the essential ideas of a (geometric) algorithm.
- Different GPA assumptions might be made depending on the actual application. (See the slides on Voronoi diagrams later in this course!)

Line Segment Intersection: General Position Assumed

General position assumed

For the sake of descriptional simplicity, we assume that

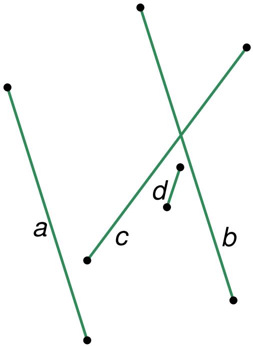
- no two end-points or intersections of line segments of S have the same y-coordinate;
- no two line segments overlap;
- no three line segments intersect at the same point;
- no segment is horizontal.

- A GPA assumption makes perfect sense since it allows to avoid special cases and, thus, to focus on the essential ideas of a (geometric) algorithm.
- Different GPA assumptions might be made depending on the actual application. (See the slides on Voronoi diagrams later in this course!)

Caveat

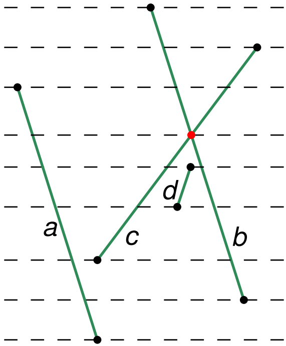
A GPA assumption will not hold for most real-world data. Thus, a GPA assumption may make it necessary to work out all the (possibly subtle) details and to close all (possibly non-trivial) gaps on one's own prior to an actual implementation . . .

Line Segment Intersection: Plane Sweep

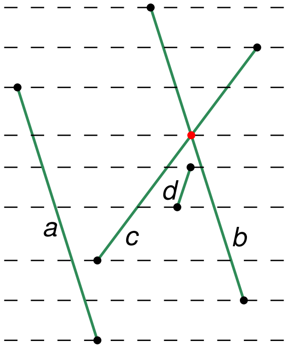


Line Segment Intersection: Plane Sweep

- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.

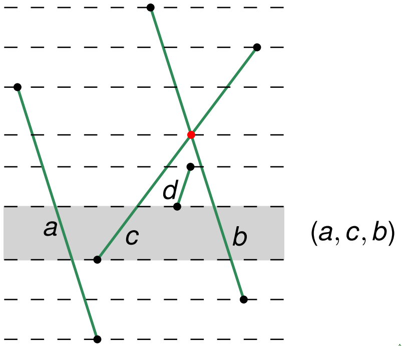


- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n+k$ lines split $\mathbb{R}^2$ into $2n+k+1$ horizontal slabs.



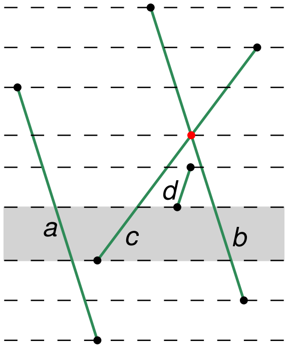
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n + k$ lines split $\mathbb{R}^2$ into $2n + k + 1$ horizontal slabs.
- Note that the left-to-right order of the line segments does not change within a slab.



Line Segment Intersection: Plane Sweep

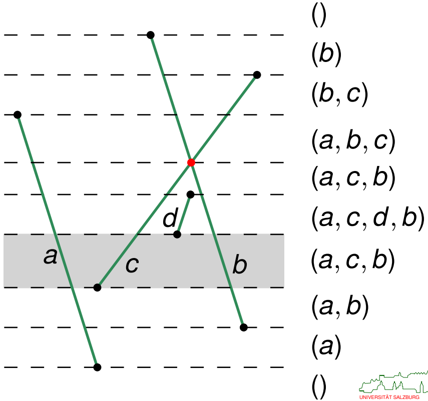
- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n + k$ lines split $\mathbb{R}^2$ into $2n + k + 1$ horizontal slabs.
- Note that the left-to-right order of the line segments does not change within a slab. And this observation holds for all slabs!



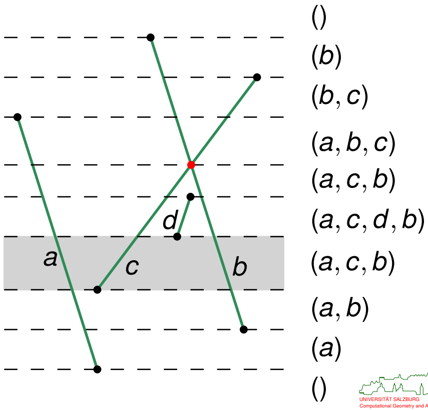
$()$
(b)
(b, c)
(a, b, c)
(a, c, b)
(a, c, d, b)
(a, c, b)
(a, b)
(a)
$()$

Line Segment Intersection: Plane Sweep

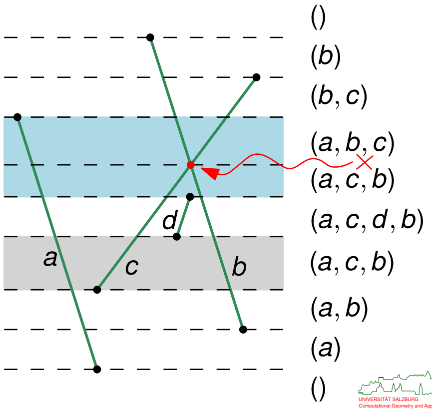
- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n + k$ lines split $\mathbb{R}^2$ into $2n + k + 1$ horizontal slabs.
- Note that the left-to-right order of the line segments does not change within a slab. And this observation holds for all slabs!
- Question: When does the relative order of two line segments change?



- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n + k$ lines split $\mathbb{R}^2$ into $2n + k + 1$ horizontal slabs.
- Note that the left-to-right order of the line segments does not change within a slab. And this observation holds for all slabs!
- Question: When does the relative order of two line segments change?
- Answer: The relative order of two line segments ℓ_1, ℓ_2 changes at the border line of two adjacent slabs if that border line passes through the point of intersection of ℓ_1 and ℓ_2.



- Suppose that we draw horizontal lines through all $2n$ end-points of the line segments and all k intersection points.
- These $2n + k$ lines split $\mathbb{R}^2$ into $2n + k + 1$ horizontal slabs.
- Note that the left-to-right order of the line segments does not change within a slab. And this observation holds for all slabs!
- Question: When does the relative order of two line segments change?
- Answer: The relative order of two line segments ℓ_1, ℓ_2 changes at the border line of two adjacent slabs if that border line passes through the point of intersection of ℓ_1 and ℓ_2.



Lemma 28

Two line segments ℓ_1, ℓ_2 intersect if and only if there exist two adjacent slabs such that ℓ_1, ℓ_2 are neighbors in the left-to-right orders and such that the relative order of ℓ_1, ℓ_2 within the two slabs is different.

Line Segment Intersection: Plane Sweep

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.
  - Halt and update these left-to-right orders whenever necessary.

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.
  - Halt and update these left-to-right orders whenever necessary.

Plane-sweep algorithm (aka: “sweep-line algorithm”)

A *plane-sweep algorithm* uses two data structures:

1. *Event-point schedule*: Sequence of halting positions to be assumed by the sweep line.

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.
  - Halt and update these left-to-right orders whenever necessary.

Plane-sweep algorithm (aka: “sweep-line algorithm”)

A *plane-sweep algorithm* uses two data structures:

1. *Event-point schedule*: Sequence of halting positions to be assumed by the sweep line.

Sweep for line-segment intersection

Plane sweep applied to line-segment intersection detection:

1. Event-point schedule: End-points of all line segments of S and all intersection points, arranged according to ascending y-coordinates. (The sweep is bottom-to-top.)

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Line Segment Intersection: Plane Sweep

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.
  - Halt and update these left-to-right orders whenever necessary.

Plane-sweep algorithm (aka: "sweep-line algorithm")

A *plane-sweep algorithm* uses two data structures:

1. *Event-point schedule*: Sequence of halting positions to be assumed by the sweep line.
2. *Sweep-line status:* Description of the intersection of the sweep line with the geometric object(s) being swept at the current event.

Sweep for line-segment intersection

Plane sweep applied to line-segment intersection detection:

1. Event-point schedule: End-points of all line segments of S and all intersection points, arranged according to ascending y-coordinates. (The sweep is bottom-to-top.)

Line Segment Intersection: Plane Sweep

- Basic idea:
  - Sweep a horizontal line over the line segments and keep track of their left-to-right orders.
  - Halt and update these left-to-right orders whenever necessary.

Plane-sweep algorithm (aka: "sweep-line algorithm")

A *plane-sweep algorithm* uses two data structures:

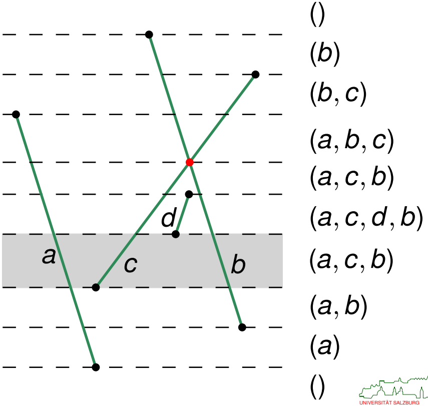
1. *Event-point schedule*: Sequence of halting positions to be assumed by the sweep line.
2. *Sweep-line status:* Description of the intersection of the sweep line with the geometric object(s) being swept at the current event.

Sweep for line-segment intersection

Plane sweep applied to line-segment intersection detection:

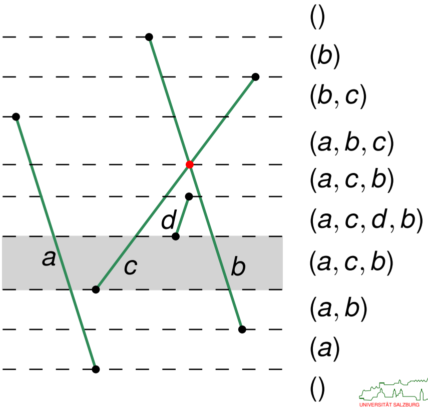
1. Event-point schedule: End-points of all line segments of S and all intersection points, arranged according to ascending y-coordinates. (The sweep is bottom-to-top.)
2. Sweep-line status: Left-to-right sequence of the line segments of S that intersect the sweep line.

1. Initialize a priority queue Q of future events:
   - Every event is associated with a point in $\mathbb{R}^2$ and with the up to two line segments on which it lies.



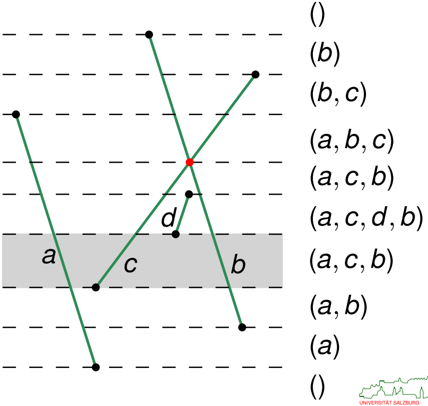
Line Segment Intersection: Plane Sweep

1. Initialize a priority queue Q of future events:
   - Every event is associated with a point in $\mathbb{R}^2$ and with the up to two line segments on which it lies.
   - The events are prioritized according to the points' y-coordinates.



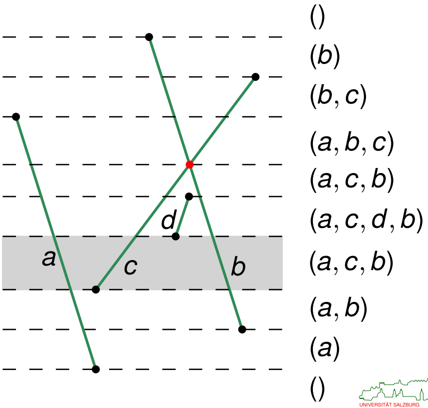
Line Segment Intersection: Plane Sweep

1. Initialize a priority queue Q of future events:
   - Every event is associated with a point in $\mathbb{R}^2$ and with the up to two line segments on which it lies.
   - The events are prioritized according to the points' y-coordinates.
2. Insert all $2n$ end-points of the n line segments into Q.



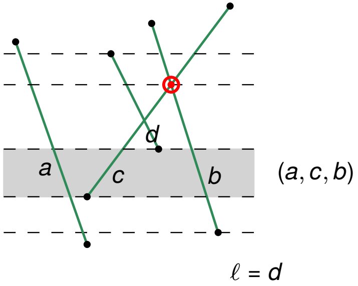
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

1. Initialize a priority queue Q of future events:
   - Every event is associated with a point in $\mathbb{R}^2$ and with the up to two line segments on which it lies.
   - The events are prioritized according to the points' y-coordinates.
2. Insert all $2n$ end-points of the n line segments into Q.
3. Initialize a binary search tree T that will contain those line segments of S which are crossed by the sweep line:
   - The segments are ordered according to the x-coordinates of the crossing points.
   - Initially, T is empty.

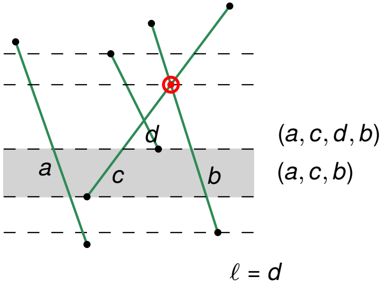


4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:

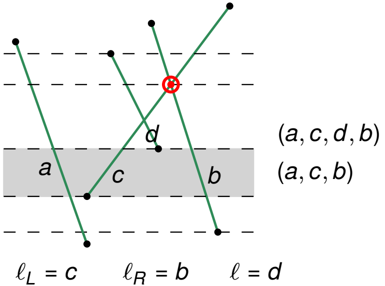
4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   a. If p is the lower end-point of a line segment ℓ:



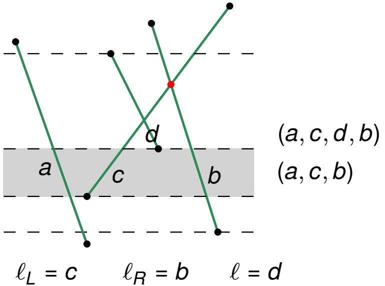
4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   a. If p is the lower end-point of a line segment ℓ:
      i. Insert ℓ into T.



4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   a. If p is the lower end-point of a line segment ℓ:
      i. Insert ℓ into T.
      ii. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)

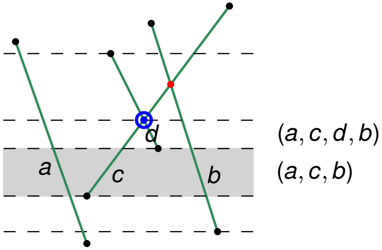


4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   a. If p is the lower end-point of a line segment ℓ:
      i. Insert ℓ into T.
      ii. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)
      iii. If ℓ_L, ℓ_R intersect above y_p then remove the intersection from Q.



Line Segment Intersection: Plane Sweep

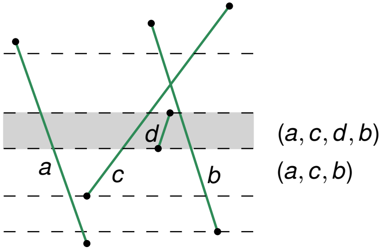
4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   a. If p is the lower end-point of a line segment ℓ:
      i. Insert ℓ into T.
      ii. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)
      iii. If ℓ_L, ℓ_R intersect above y_p then remove the intersection from Q.
      iv. If ℓ_L, ℓ or ℓ, ℓ_R intersect above y_p then insert the intersection(s) into Q.



$\ell_L = c \qquad \ell_R = b \qquad \ell = d$

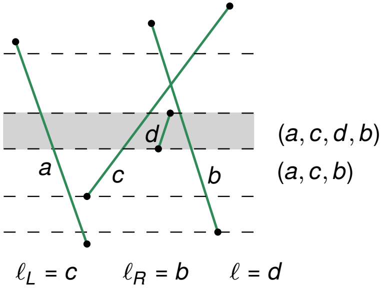
Line Segment Intersection: Plane Sweep

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   b. If p is the upper end-point of a line segment ℓ:



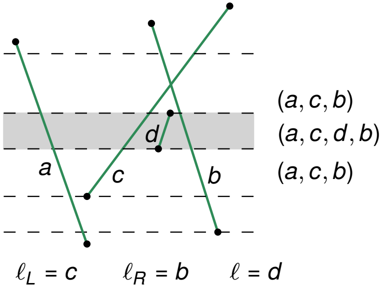
$\ell = d$

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - b. If p is the upper end-point of a line segment ℓ:
     - i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)



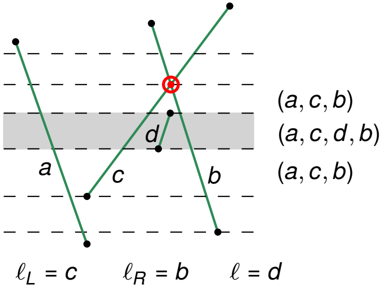
Line Segment Intersection: Plane Sweep

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - b. If p is the upper end-point of a line segment ℓ:
     - i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)
     - ii. Remove ℓ from T.

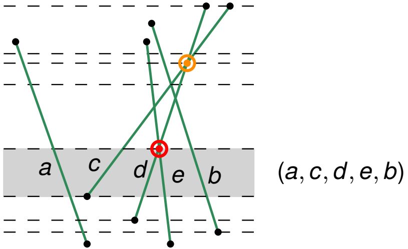


Line Segment Intersection: Plane Sweep

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   b. If p is the upper end-point of a line segment ℓ:
      i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left and right of ℓ, if they exist. (Use T to locate ℓ_L, ℓ_R.)
      ii. Remove ℓ from T.
      iii. If ℓ_L, ℓ_R intersect above y_p then insert the intersection into Q.

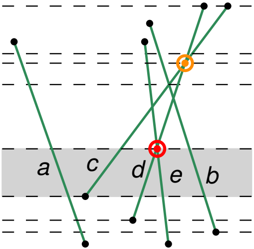


4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - c. If p is a point of intersection of ℓ_1 and ℓ_2:



$\ell_1 = d \qquad \ell_2 = e$

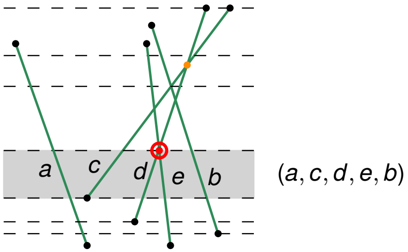
4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - c. If p is a point of intersection of ℓ_1 and ℓ_2:
     - i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left of ℓ_1 and right of ℓ_2, if they exist. (Use T to locate ℓ_L, ℓ_R.)



(a, c, d, e, b)

$\ell_L = c \quad \ell_R = b \quad \ell_1 = d \quad \ell_2 = e$

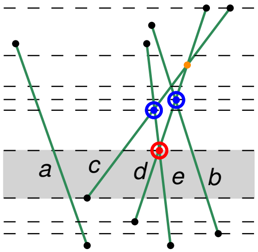
4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - c. If p is a point of intersection of ℓ_1 and ℓ_2:
     - i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left of ℓ_1 and right of ℓ_2, if they exist. (Use T to locate ℓ_L, ℓ_R.)
     - ii. If ℓ_L, ℓ_1 or ℓ_2, ℓ_R intersect above y_p then remove the intersection(s) from Q.



$\ell_L = c \quad \ell_R = b \quad \ell_1 = d \quad \ell_2 = e$

Line Segment Intersection: Plane Sweep

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   - c. If p is a point of intersection of ℓ_1 and ℓ_2:
     - i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left of ℓ_1 and right of ℓ_2, if they exist. (Use T to locate ℓ_L, ℓ_R.)
     - ii. If ℓ_L, ℓ_1 or ℓ_2, ℓ_R intersect above y_p then remove the intersection(s) from Q.
     - iii. If ℓ_1, ℓ_R or ℓ_L, ℓ_2 intersect above y_p then insert the intersection(s) into Q.

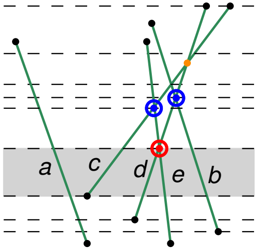


(a, c, d, e, b)

$\ell_L = c$ $\ell_R = b$ $\ell_1 = d$ $\ell_2 = e$

Line Segment Intersection: Plane Sweep

4. While Q is not empty, fetch and remove the next event from Q. Let p be the point associated with that event, and let y_p be its y-coordinate:
   c. If p is a point of intersection of ℓ_1 and ℓ_2:
      i. Let ℓ_L and ℓ_R be the line segments that are immediately to the left of ℓ_1 and right of ℓ_2, if they exist. (Use T to locate ℓ_L, ℓ_R.)
      ii. If ℓ_L, ℓ_1 or ℓ_2, ℓ_R intersect above y_p then remove the intersection(s) from Q.
      iii. If ℓ_1, ℓ_R or ℓ_L, ℓ_2 intersect above y_p then insert the intersection(s) into Q.
      iv. Trade the order of ℓ_1 and ℓ_2 in T.



(a, c, e, d, b)

(a, c, d, e, b)

$\ell_L = c$ $\ell_R = b$ $\ell_1 = d$ $\ell_2 = e$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Correctness:

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.

Line Segment Intersection: Plane Sweep

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.
- Since future intersections between line segments are maintained in the priority queue Q if and only if the line segments currently are neighbors in the left-to-right order, at any given point in time we will never need to do maintain more than $3n - 1$ events in Q.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.
- Since future intersections between line segments are maintained in the priority queue Q if and only if the line segments currently are neighbors in the left-to-right order, at any given point in time we will never need to do maintain more than $3n - 1$ events in Q.
- The algorithm stores up to n line segments in left-to-right order in T.

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.
- Since future intersections between line segments are maintained in the priority queue Q if and only if the line segments currently are neighbors in the left-to-right order, at any given point in time we will never need to do maintain more than $3n - 1$ events in Q.
- The algorithm stores up to n line segments in left-to-right order in T.
- Every event requires a constant number of updates of Q and T.

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.
- Since future intersections between line segments are maintained in the priority queue Q if and only if the line segments currently are neighbors in the left-to-right order, at any given point in time we will never need to do maintain more than $3n - 1$ events in Q.
- The algorithm stores up to n line segments in left-to-right order in T.
- Every event requires a constant number of updates of Q and T.
- If Q and T allow insertions, deletions and searches in logarithmic time then every event is handled in $O(\log n)$ time.
- Any standard balanced binary search tree (e.g., AVL-tree, red-black tree) and any logarithmic-time priority queue (e.g., binary heap) suffices.

Line Segment Intersection: Plane Sweep

Correctness:

- Whenever two line segments ℓ_1, ℓ_2 are neigbors in the sorted left-to-right order of segments, the point of intersection of ℓ_1, ℓ_2 is present in Q, if it exists and has a higher y-coordinate.
- Hence, no future event and, in particular, no point of intersection is missed.

Complexity:

- The algorithm processes a sequence of $2n + k$ events.
- Since future intersections between line segments are maintained in the priority queue Q if and only if the line segments currently are neighbors in the left-to-right order, at any given point in time we will never need to do maintain more than $3n - 1$ events in Q.
- The algorithm stores up to n line segments in left-to-right order in T.
- Every event requires a constant number of updates of Q and T.
- If Q and T allow insertions, deletions and searches in logarithmic time then every event is handled in $O(\log n)$ time.
- Any standard balanced binary search tree (e.g., AVL-tree, red-black tree) and any logarithmic-time priority queue (e.g., binary heap) suffices.
- Summarizing, the Bentley-Ottmann algorithm finds all intersections among n line segments in $O((n + k) \log n)$ time, using $O(n)$ space.

Rotational sweep:

- A line (or ray) rotates about a point.

Rotational sweep:

- A line (or ray) rotates about a point.

Space sweep:

- A plane (which is parallel to one of the coordinate planes) sweeps through 3D space.
- A recursive application of this idea sometimes allows to replace a d-dimensional problem by a series of $(d-1)$-dimensional problems.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Generalizations of the Sweep Paradigm

Rotational sweep:

- A line (or ray) rotates about a point.

Space sweep:

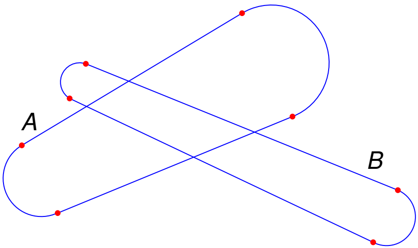
- A plane (which is parallel to one of the coordinate planes) sweeps through 3D space.
- A recursive application of this idea sometimes allows to replace a d-dimensional problem by a series of $(d-1)$-dimensional problems.

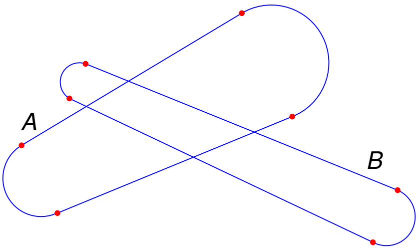
Topological sweep:

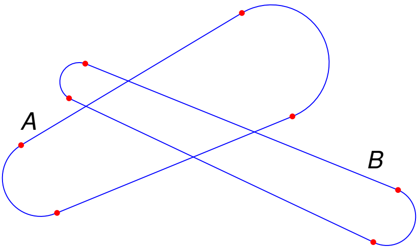
- Edelsbrunner&Guibas (1991).
- A “topological” line is used instead of a straight line.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

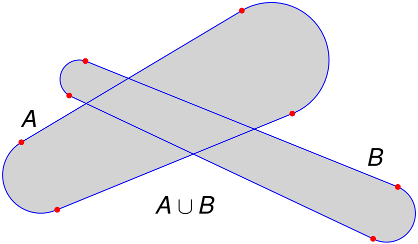
Computing Boolean Operations on Curvilinear Polygons

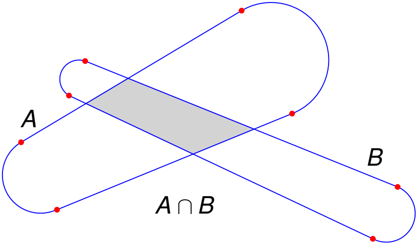


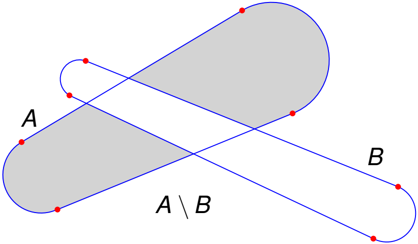




Computing Boolean Operations on Curvilinear Polygons

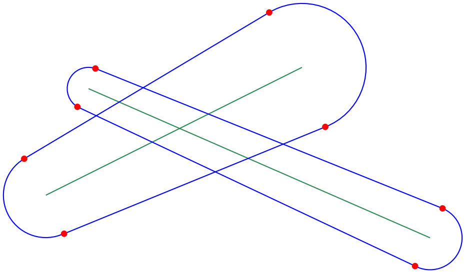




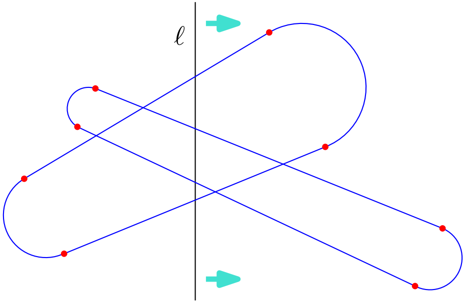


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

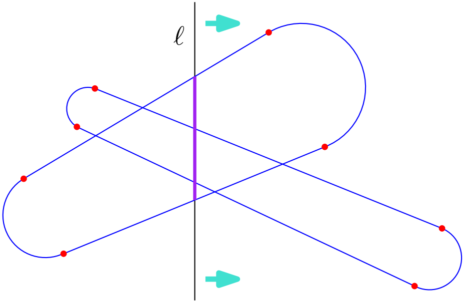
Sweep-Line Algorithm for Boolean Operations



- Consider a vertical line ℓ that sweeps from left to right.

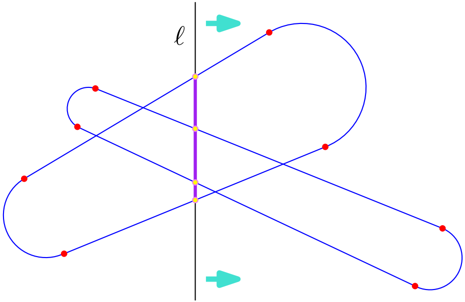


- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,



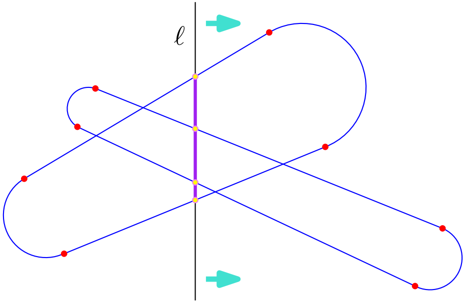
Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



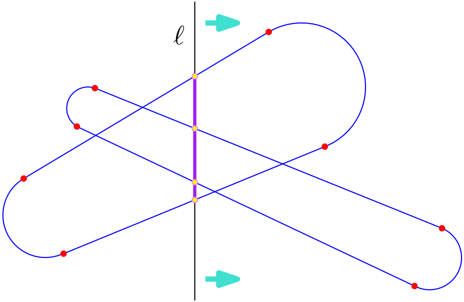
Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



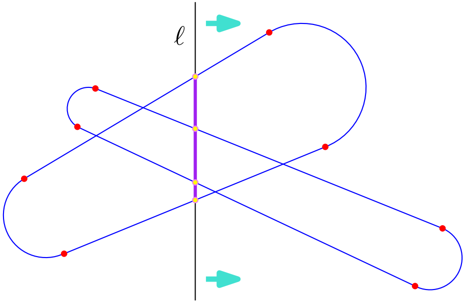
- Q: At which events does the intersection of ℓ with the union change topologically?

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.

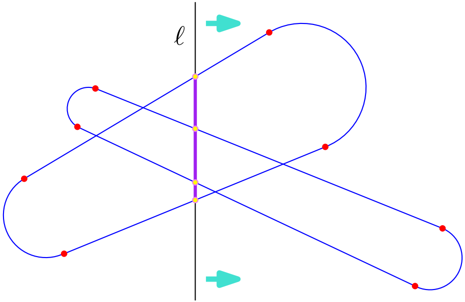
- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?

Sweep-Line Algorithm for Boolean Operations

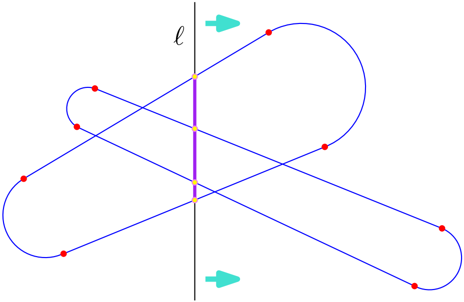
- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

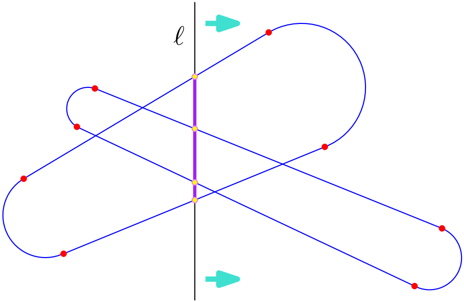


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.
- Put events into a priority queue and process in left-to-right order.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm for Boolean Operations

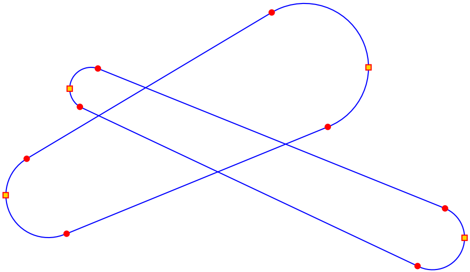
- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

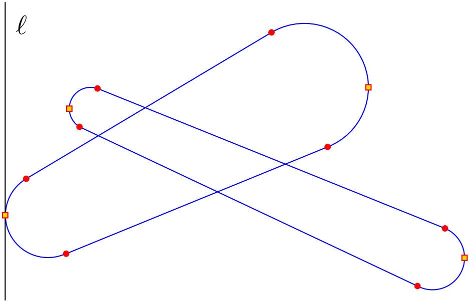


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



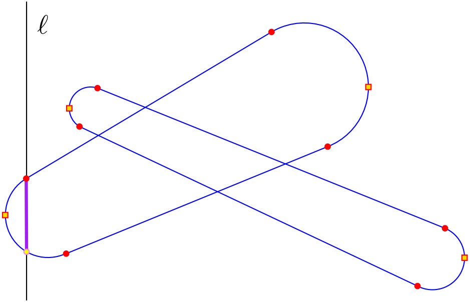
- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



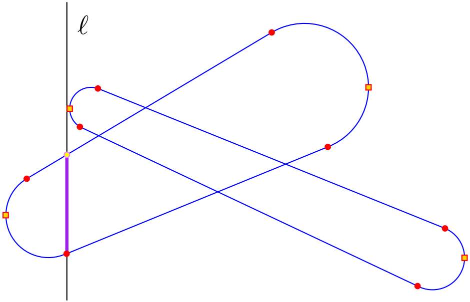
- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

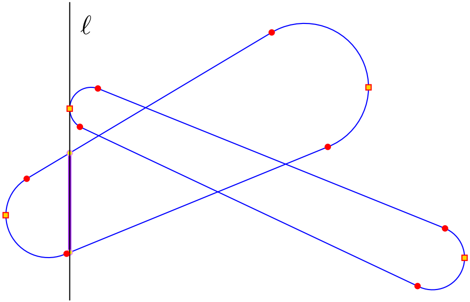


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

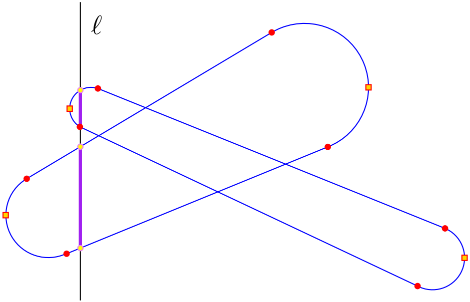


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

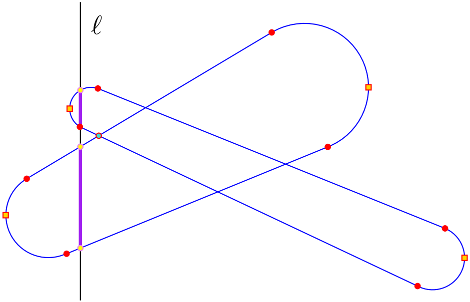


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.

- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.

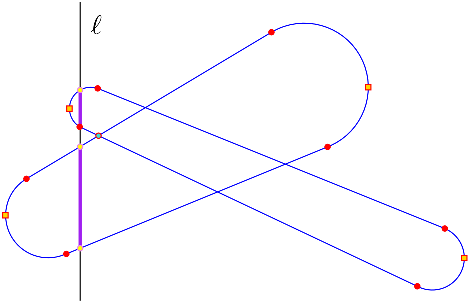


- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.
- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!
- Handling of circular arcs on the same circle requires some care.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm for Boolean Operations

- Consider a vertical line ℓ that sweeps from left to right. Study
  (1) its intersection with the union of the curvilinear polygons,
  (2) the top-to-bottom order of the segments/arcs intersected by ℓ.



- Q: At which events does the intersection of ℓ with the union change topologically?
  A: Whenever ℓ enters or leaves a curvilinear polygon.
- Q: At which events does the top-to-bottom order of the segments/arcs intersected by ℓ change?
  A: Whenever ℓ moves through a vertex or an intersection point.
- Put events into a priority queue and process in left-to-right order.
- Intersection points can be detected on the fly between segments/arcs that are neighbors in the top-to-bottom order!
- Handling of circular arcs on the same circle requires some care.
- Complexity: $O((n+k)\log n)$ for n segments/arcs and k intersection points.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

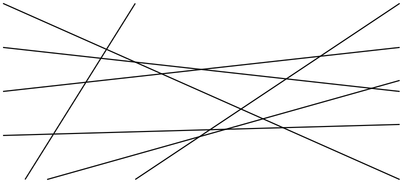
2 Geometric Concepts and Paradigms

- Plane Sweep
- Arrangements
  - Basics
  - Construction
- Point-Line Duality

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 29 (Arrangement)

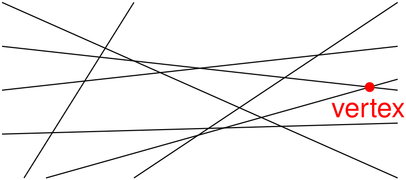
Consider a set L of lines in the plane.



Definition 29 (Arrangement)

Consider a set L of lines in the plane. The *(line) arrangement* $\mathcal{A}(L)$ induced by L is the subdivision of the plane that consists of

vertices: points of intersection of two or more lines of L,

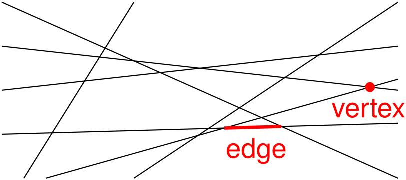


Definition 29 (Arrangement)

Consider a set L of lines in the plane. The *(line) arrangement* $\mathcal{A}(L)$ induced by L is the subdivision of the plane that consists of

vertices: points of intersection of two or more lines of L,

edges: connected components of all lines of L without all vertices,



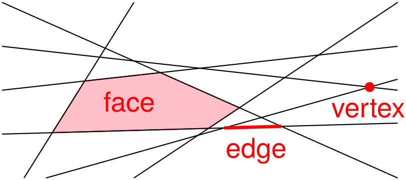
Definition 29 (Arrangement)

Consider a set L of lines in the plane. The *(line) arrangement* $\mathcal{A}(L)$ induced by L is the subdivision of the plane that consists of

vertices: points of intersection of two or more lines of L,

edges: connected components of all lines of L without all vertices,

faces: the connected components of the subset of the plane not intersected by any line of L. Aka: cells.

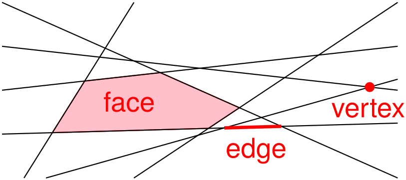


Definition 29 (Arrangement)

Consider a set L of lines in the plane. The *(line) arrangement* $\mathcal{A}(L)$ induced by L is the subdivision of the plane that consists of

- **vertices:** points of intersection of two or more lines of L,
- **edges:** connected components of all lines of L without all vertices,
- **faces:** the connected components of the subset of the plane not intersected by any line of L. Aka: cells.

An arrangement is *simple* if no more than two lines of L intersect at a vertex.



Definition 29 (Arrangement)

Consider a set L of lines in the plane. The *(line) arrangement* $\mathcal{A}(L)$ induced by L is the subdivision of the plane that consists of

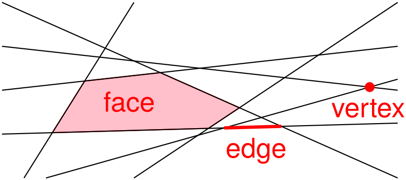
vertices: points of intersection of two or more lines of L,

edges: connected components of all lines of L without all vertices,

faces: the connected components of the subset of the plane not intersected by any line of L. Aka: cells.

An arrangement is *simple* if no more than two lines of L intersect at a vertex.

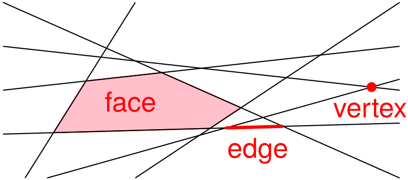
- Arrangements can also be induced by other primitives (e.g., circles) and studied in higher dimensions.



Line Arrangement: Combinatorial Complexity

Lemma 30

Every face of an arrangement is convex.



Line Arrangement: Combinatorial Complexity

Lemma 30

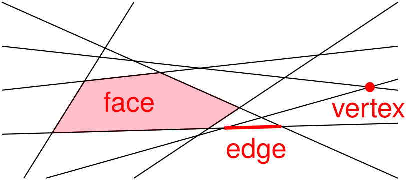
Every face of an arrangement is convex.

Lemma 31

The arrangement induced by a set of n lines has

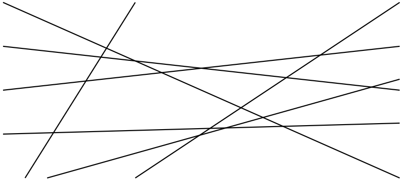
- at most $\binom{n}{2}$ vertices,
- at most n^2 edges,
- at most $\binom{n+1}{2} + 1$ faces,

i.e., its combinatorial complexity is $O(n^2)$. Equality holds for simple arrangements.



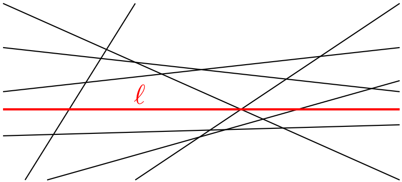
Definition 32

The *zone* of a line $\ell \notin L$ in an arrangement $\mathcal{A}(L)$ of a set L of lines is the set of all faces of $\mathcal{A}(L)$ whose closure is intersected by ℓ.



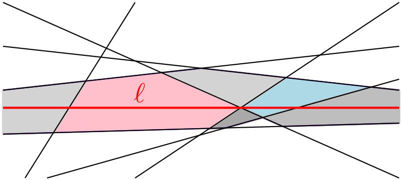
Definition 32

The *zone* of a line $\ell \notin L$ in an arrangement $\mathcal{A}(L)$ of a set L of lines is the set of all faces of $\mathcal{A}(L)$ whose closure is intersected by ℓ.



Definition 32

The *zone* of a line $\ell \notin L$ in an arrangement $\mathcal{A}(L)$ of a set L of lines is the set of all faces of $\mathcal{A}(L)$ whose closure is intersected by ℓ.

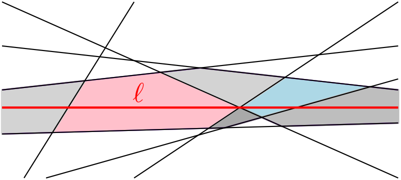


Definition 32

The *zone* of a line $\ell \notin L$ in an arrangement $\mathcal{A}(L)$ of a set L of lines is the set of all faces of $\mathcal{A}(L)$ whose closure is intersected by ℓ.

Theorem 33 (Zone theorem)

The complexity of the zone of a line in an arrangement of n lines is $O(n)$.



Line Arrangement: Combinatorial Complexity

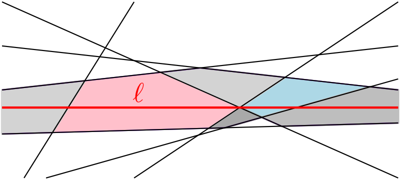
Definition 32

The *zone* of a line $\ell \notin L$ in an arrangement $\mathcal{A}(L)$ of a set L of lines is the set of all faces of $\mathcal{A}(L)$ whose closure is intersected by ℓ.

Theorem 33 (Zone theorem)

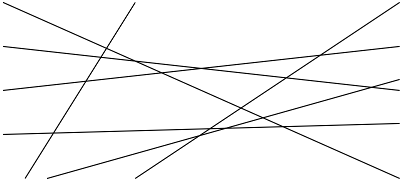
The complexity of the zone of a line in an arrangement of n lines is $O(n)$.

Sketch of Proof : Assume that ℓ is horizontal and construct the zone by inserting the lines of L from left to right along ℓ. Then one can show by induction that each new line adds at most 6 new zone edges. □



Problem: LINEARRANGEMENT

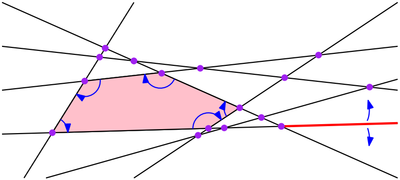
Given: A set L of lines in the plane.



Problem: LINEARRANGEMENT

Given: A set L of lines in the plane.

Compute: A (combinatorial) representation of the arrangement $\mathcal{A}(L)$ that allows to traverse $\mathcal{A}(L)$.



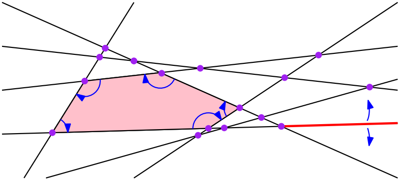
Problem: LINEARRANGEMENT

Given: A set L of lines in the plane.

Compute: A (combinatorial) representation of the arrangement $\mathcal{A}(L)$ that allows to traverse $\mathcal{A}(L)$.

Theorem 34

A combinatorial representation of the arrangement $\mathcal{A}(L)$ of a set L of n lines in the plane can be computed incrementally in time $O(n^2)$.



Line Arrangement: Construction

Problem: LineArrangement

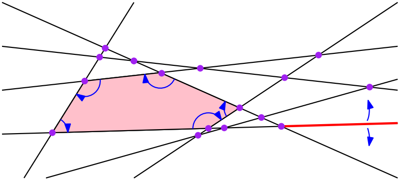
Given: A set L of lines in the plane.

Compute: A (combinatorial) representation of the arrangement $\mathcal{A}(L)$ that allows to traverse $\mathcal{A}(L)$.

Theorem 34

A combinatorial representation of the arrangement $\mathcal{A}(L)$ of a set L of n lines in the plane can be computed incrementally in time $O(n^2)$.

Sketch of Proof: The Zone Theorem 33 implies $O(n)$ complexity per insertion of a line of L. □



2 Geometric Concepts and Paradigms

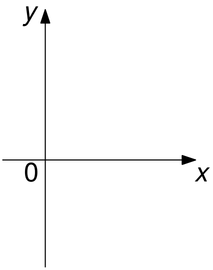
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Point-Line Duality

- We study two incarnations of the plane, both with right-handed Cartesian coordinate systems:

Point-Line Duality

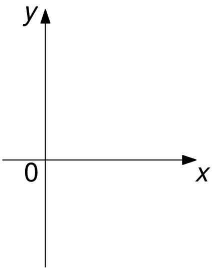
- We study two incarnations of the plane, both with right-handed Cartesian coordinate systems:
    - the *primal plane* with coordinates x, y,



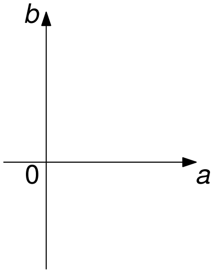
primal plane

Point-Line Duality

- We study two incarnations of the plane, both with right-handed Cartesian coordinate systems:
  - the *primal plane* with coordinates x, y, and
  - the *dual plane* with coordinates a, b.



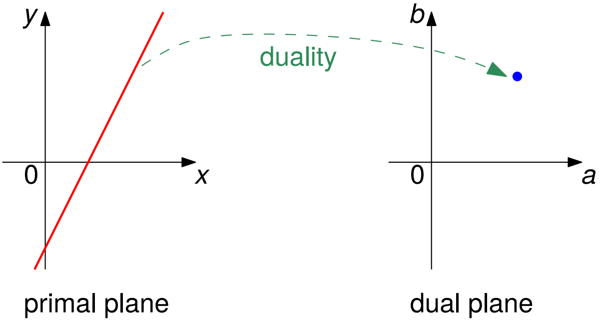
primal plane



dual plane

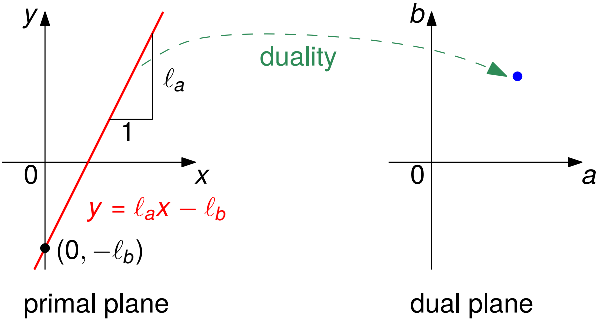
Point-Line Duality

- We study two incarnations of the plane, both with right-handed Cartesian coordinate systems:
  - the *primal plane* with coordinates x, y, and
  - the *dual plane* with coordinates a, b.
- We will identify a line in one plane with a point in the other plane, and vice versa.



Point-Line Duality

- We study two incarnations of the plane, both with right-handed Cartesian coordinate systems:
  - the *primal plane* with coordinates x, y, and
  - the *dual plane* with coordinates a, b.
- We will identify a line in one plane with a point in the other plane, and vice versa.
- Remember: A (non-vertical) line ℓ has the equation $y = \ell_a x - \ell_b$, where ℓ_a models the slope and ℓ_b models the y-intercept of ℓ.

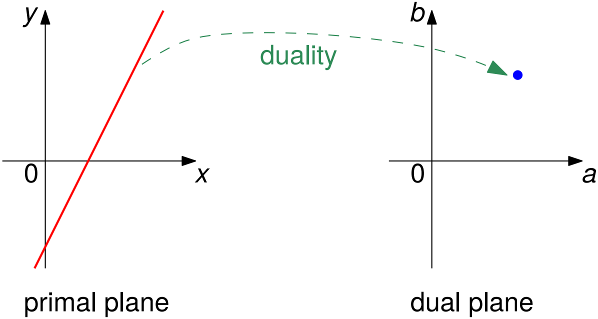


Point-Line Duality: Motivation

Goal

A duality mapping between points and lines in the plane shall allow us to translate theorems and algorithms about

points and lines



Point-Line Duality: Motivation

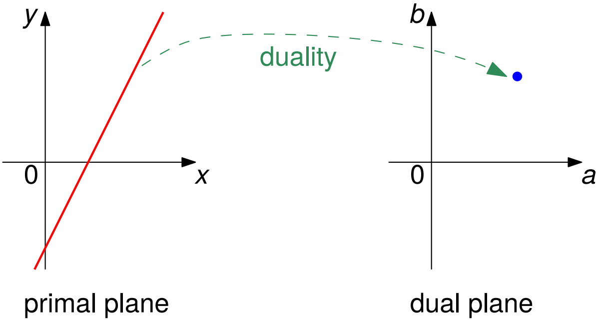
Goal

A duality mapping between points and lines in the plane shall allow us to translate theorems and algorithms about

points and lines

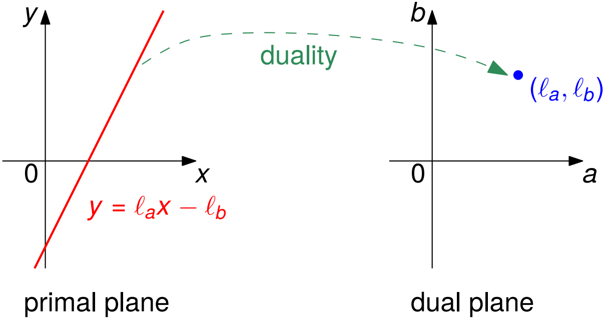
into theorems and algorithms about

lines and points.



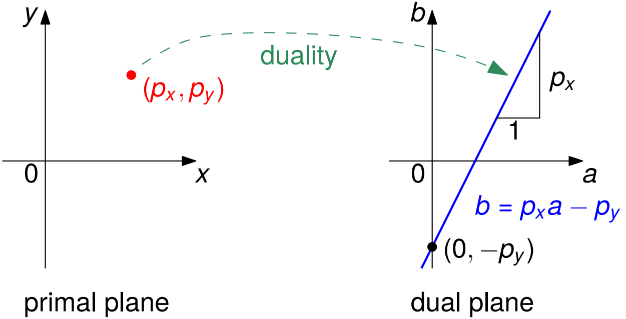
Definition 35 (Point-line duality)

1. Let ℓ be a line in primal space with equation $y = \ell_a x - \ell_b$. We associate with ℓ the point $\ell^\star$ in the dual plane with coordinates (ℓ_a, ℓ_b).



Definition 35 (Point-line duality)

1. Let ℓ be a line in primal space with equation $y = \ell_a x - \ell_b$. We associate with ℓ the point $\ell^\star$ in the dual plane with coordinates (ℓ_a, ℓ_b).
2. Let p be a point in the primal space with coordinates (p_x, p_y). We associate with p the line $p^\star$ in the dual plane with equation $b = p_x a - p_y$.



Point-Line Duality: Properties

- Of course, we can apply the same duality mapping $\star$ to points and lines in the dual plane, and map them to lines and points in the primal plane.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Point-Line Duality: Properties

- Of course, we can apply the same duality mapping $\star$ to points and lines in the dual plane, and map them to lines and points in the primal plane.

Lemma 36 (Self-inverse mapping)

The duality mapping $*$ is self-inverse:
(1) For every point p in the primal plane: $(p^\star)^\star = p$.
(2) For every line ℓ in the primal plane: $(\ell^\star)^\star = \ell$.

Point-Line Duality: Properties

- Of course, we can apply the same duality mapping $\star$ to points and lines in the dual plane, and map them to lines and points in the primal plane.

Lemma 36 (Self-inverse mapping)

The duality mapping $*$ is self-inverse:
(1) For every point p in the primal plane: $(p^\star)^\star = p$.
(2) For every line ℓ in the primal plane: $(\ell^\star)^\star = \ell$.

Proof : (1) For p with coordinates (p_x, p_y) we get the dual line $p^\star$ with equation $b = p_x a - p_y$. This line dualizes to a point $(p^\star)^\star$ with coordinates $(p_x, -(-p_y))$, i.e., to p.

Point-Line Duality: Properties

- Of course, we can apply the same duality mapping $\star$ to points and lines in the dual plane, and map them to lines and points in the primal plane.

Lemma 36 (Self-inverse mapping)

The duality mapping $*$ is self-inverse:
(1) For every point p in the primal plane: $(p^\star)^\star = p$.
(2) For every line ℓ in the primal plane: $(\ell^\star)^\star = \ell$.

Proof: (1) For p with coordinates (p_x, p_y) we get the dual line $p^\star$ with equation $b = p_x a - p_y$. This line dualizes to a point $(p^\star)^\star$ with coordinates $(p_x, -(-p_y))$, i.e., to p.
(2) A line ℓ with equation $y = \ell_a x - \ell_b$ dualizes to the point $\ell^\star$ with coordinates (ℓ_a, ℓ_b), which in turn dualizes to the line $(\ell^\star)^\star$ with equation $y = \ell_a x - \ell_b$, i.e., to ℓ. $\square$

Lemma 37 (Incidence-preserving mapping)

For every point p and every line ℓ in primal space: $p \in \ell$ if and only if $\ell^\star \in p^\star$.

Point-Line Duality: Properties

Lemma 37 (Incidence-preserving mapping)

For every point p and every line ℓ in primal space: $p \in \ell$ if and only if $\ell^\star \in p^\star$.

Proof : Assume that $p \in \ell$, with point coordinates (p_x, p_y) and line equation $y = \ell_a x - \ell_b$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 37 (Incidence-preserving mapping)

For every point p and every line ℓ in primal space: $p \in \ell$ if and only if $\ell^\star \in p^\star$.

Proof: Assume that $p \in \ell$, with point coordinates (p_x, p_y) and line equation $y = \ell_a x - \ell_b$. Then $p_y = \ell_a p_x - \ell_b$. Hence, $\ell_b = p_x \ell_a - p_y$, and $\ell^\star = (\ell_a, \ell_b) \in p^\star$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 37 (Incidence-preserving mapping)

For every point p and every line ℓ in primal space: $p \in \ell$ if and only if $\ell^\star \in p^\star$.

Proof: Assume that $p \in \ell$, with point coordinates (p_x, p_y) and line equation $y = \ell_a x - \ell_b$. Then $p_y = \ell_a p_x - \ell_b$. Hence, $\ell_b = p_x \ell_a - p_y$, and $\ell^\star = (\ell_a, \ell_b) \in p^\star$. Now assume that $\ell^\star \in p^\star$. We get $(p^\star)^\star \in (\ell^\star)^\star$, and by Lem. 36, $p \in \ell$. □

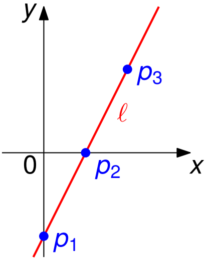
Lemma 37 (Incidence-preserving mapping)

For every point p and every line ℓ in primal space: $p \in \ell$ if and only if $\ell^\star \in p^\star$.

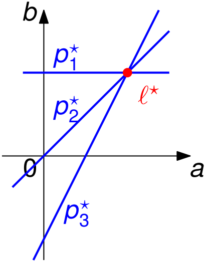
Proof: Assume that $p \in \ell$, with point coordinates (p_x, p_y) and line equation $y = \ell_a x - \ell_b$. Then $p_y = \ell_a p_x - \ell_b$. Hence, $\ell_b = p_x \ell_a - p_y$, and $\ell^\star = (\ell_a, \ell_b) \in p^\star$. Now assume that $\ell^\star \in p^\star$. We get $(p^\star)^\star \in (\ell^\star)^\star$, and by Lem. 36, $p \in \ell$. □

Corollary 38

The points p_1, p_2, p_3 lie on the line ℓ if and only iff the lines $p_1^\star, p_2^\star, p_3^\star$ intersect in the common point $\ell^\star$.



primal plane



dual plane

UNIVERSITÄT SALZBURG Computational Geometry and Applications Lab

Point-Line Duality: Properties

Lemma 39

The signed vertical distance from a point p to a line ℓ equals the signed vertical distance from the point $\ell^\star$ to the line $p^\star$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 39

The signed vertical distance from a point p to a line ℓ equals the signed vertical distance from the point $\ell^\star$ to the line $p^\star$.

- This lemma implies Lem. 37.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 39

The signed vertical distance from a point p to a line ℓ equals the signed vertical distance from the point $\ell^\star$ to the line $p^\star$.

- This lemma implies Lem. 37.

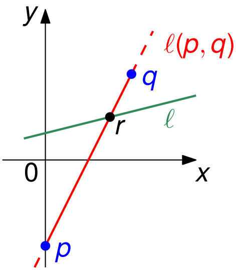
Corollary 40

A point p lies above a line ℓ if and only if the point $\ell^\star$ lies above the line $p^\star$.

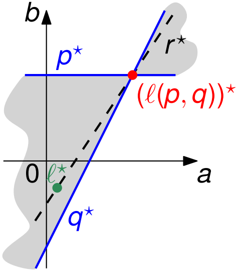
Point-Line Duality: Properties

Corollary 41

A line ℓ intersects the line segment $\overline{pq}$ if and only if the point $\ell^\star$ lies in the "horizontal" double wedge defined by the lines $p^\star$ and $q^\star$. (I.e., the double wedge which does not contain the vertical line through the intersection point of $p^\star$ and $q^\star$.)



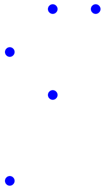
primal plane



dual plane

Problem: COLLINEARITY

Given: A set S of n points in the plane.

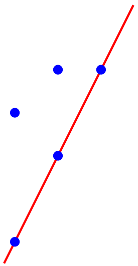


Detecting Collinearity

Problem: COLLINEARITY

Given: A set S of n points in the plane.

Decide: Are any three points of S collinear?



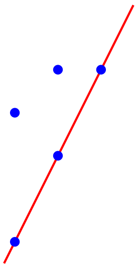
Detecting Collinearity

Problem: COLLINEARITY

Given: A set S of n points in the plane.

Decide: Are any three points of S collinear?

- Naïve algorithm: Check all triples of points of S in $O(n^3)$ time.

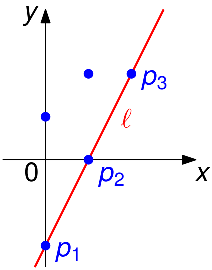


Problem: COLLINEARITY

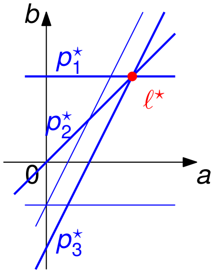
Given: A set S of n points in the plane.

Decide: Are any three points of S collinear?

- Naïve algorithm: Check all triples of points of S in $O(n^3)$ time.
- Better: Recall duality (Cor. 38) and compute the arrangement of the dual lines of the points of S in $O(n^2)$ time.



primal plane

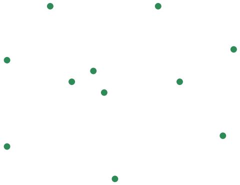


dual plane

Smallest Triangle

Problem: MINIMUMAREATRIANGLE

Given: A set S of n points in the plane.



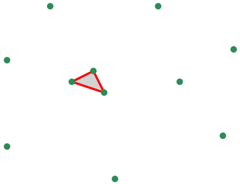
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Smallest Triangle

Problem: MinimumAreaTriangle

Given: A set S of n points in the plane.

Find: The triangle with smallest area whose three vertices are in S.



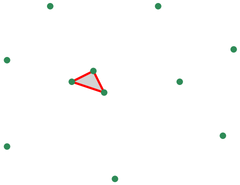
Smallest Triangle

Problem: MINIMUMAREATRIANGLE

Given: A set S of n points in the plane.

Find: The triangle with smallest area whose three vertices are in S.

- A naïve solution evaluates all triples of points of S and, thus, runs in $O(n^3)$ time.



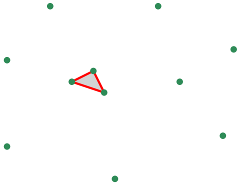
Smallest Triangle

Problem: MINIMUMAREATRIANGLE

Given: A set S of n points in the plane.

Find: The triangle with smallest area whose three vertices are in S.

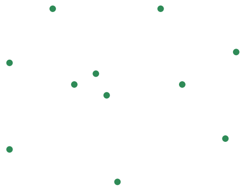
- A naïve solution evaluates all triples of points of S and, thus, runs in $O(n^3)$ time.
- General position assumed: No three points are collinear.



Smallest Triangle: Characterization of Solution

Lemma 42

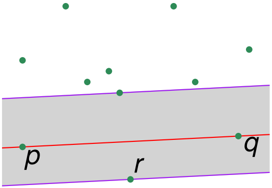
Let $p, q \in S$ with $p \neq q$. Then the point $r \in S$ which forms the smallest triangle $\Delta(p, q, r)$ with fixed base edge $\overline{pq}$ is a point of S which lies on the boundary of the largest empty corridor along the line $\ell(p, q)$.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 42

Let $p, q \in S$ with $p \neq q$. Then the point $r \in S$ which forms the smallest triangle $\Delta(p, q, r)$ with fixed base edge $\overline{pq}$ is a point of S which lies on the boundary of the largest empty corridor along the line $\ell(p, q)$.

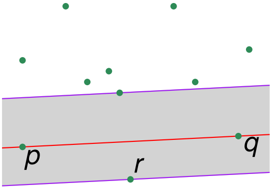


Smallest Triangle: Characterization of Solution

Lemma 42

Let $p, q \in S$ with $p \neq q$. Then the point $r \in S$ which forms the smallest triangle $\Delta(p, q, r)$ with fixed base edge $\overline{pq}$ is a point of S which lies on the boundary of the largest empty corridor along the line $\ell(p, q)$.

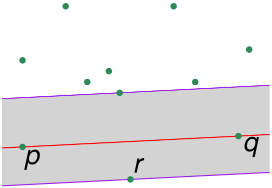
- Let ℓ be the line through p, q.



Lemma 42

Let $p, q \in S$ with $p \neq q$. Then the point $r \in S$ which forms the smallest triangle $\Delta(p, q, r)$ with fixed base edge $\overline{pq}$ is a point of S which lies on the boundary of the largest empty corridor along the line $\ell(p, q)$.

- Let ℓ be the line through p, q.
- Then r lies on a line ℓ_r such that
  - ℓ_r is parallel to ℓ,
  - there is no other line with the same slope through a point of S that lies strictly between ℓ and ℓ_r.



Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,
  - $\ell_r^\star$ lies on the boundary of the same cell of $\mathcal{A}(S^\star)$ as $\ell^\star$,

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,
  - $\ell_r^\star$ lies on the boundary of the same cell of $\mathcal{A}(S^\star)$ as $\ell^\star$,
  - $\ell_r^\star$ lies vertically below or above $\ell^\star$.

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,
  - $\ell_r^\star$ lies on the boundary of the same cell of $\mathcal{A}(S^\star)$ as $\ell^\star$,
  - $\ell_r^\star$ lies vertically below or above $\ell^\star$.
- This implies that we have only two candidate lines parallel to ℓ for any fixed pair p, q, namely the duals of the two points u, v on $\mathcal{A}(S^\star)$ right below or above $\ell^\star$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,
  - $\ell_r^\star$ lies on the boundary of the same cell of $\mathcal{A}(S^\star)$ as $\ell^\star$,
  - $\ell_r^\star$ lies vertically below or above $\ell^\star$.
- This implies that we have only two candidate lines parallel to ℓ for any fixed pair p, q, namely the duals of the two points u, v on $\mathcal{A}(S^\star)$ right below or above $\ell^\star$.
- Thus, it suffices to determine u, v for every vertex w of $\mathcal{A}(S^\star)$.

Smallest Triangle: Interpretation in Dual Space

- This implies for the dual space:
  - $\ell_r^\star$ is on $r^\star$,
  - $\ell_r^\star$ and $\ell^\star$ have the same x-coordinate (since ℓ_r and ℓ have the same slope),
  - no line $s^\star$, for $s \in S$, crosses the line segment with endpoints $\ell^\star$ and $\ell_r^\star$ (since then s would lie between ℓ and ℓ_r).
- Hence,
  - $\ell^\star$ is a vertex of $\mathcal{A}(S^\star)$,
  - $\ell_r^\star$ lies on the boundary of the same cell of $\mathcal{A}(S^\star)$ as $\ell^\star$,
  - $\ell_r^\star$ lies vertically below or above $\ell^\star$.
- This implies that we have only two candidate lines parallel to ℓ for any fixed pair p, q, namely the duals of the two points u, v on $\mathcal{A}(S^\star)$ right below or above $\ell^\star$.
- Thus, it suffices to determine u, v for every vertex w of $\mathcal{A}(S^\star)$.
- Since all faces of $\mathcal{A}(S^\star)$ are convex, this can be done on a face-by-face basis for all vertices of $\mathcal{A}(S^\star)$, in total $O(n^2)$ time.

Theorem 43

MINIMUMAREATRIANGLE can be solved in $O(n^2)$ time for n points.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

3 Geometric Searching

- Introduction
- Point Inclusion

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

3 Geometric Searching

- Introduction
- Point Inclusion

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Point-Inclusion Query: In which “cell” (of, e.g., a map) does a query point lie?

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Point-Inclusion Query: In which “cell” (of, e.g., a map) does a query point lie?

Range Searching:

- **Report Query**: Which points are within a query object (rectangle, circle)?
- **Count Query:** Only the number of points within an object matters.

Point-Inclusion Query: In which “cell” (of, e.g., a map) does a query point lie?

Range Searching:

- **Report Query**: Which points are within a query object (rectangle, circle)?
- **Count Query:** Only the number of points within an object matters.

- Another way to distinguish geometric searching queries:

Single-Shot Query: Only one query per data set.

Repetitive-Mode Query: Many queries per data set; preprocessing may make sense.

Point-Inclusion Query: In which “cell” (of, e.g., a map) does a query point lie?

Range Searching:

- **Report Query**: Which points are within a query object (rectangle, circle)?
- **Count Query:** Only the number of points within an object matters.

- Another way to distinguish geometric searching queries:

Single-Shot Query: Only one query per data set.

Repetitive-Mode Query: Many queries per data set; preprocessing may make sense.

- The complexity of a query is determined relative to four cost measures:
  - query time,
  - preprocessing time,
  - memory consumption,
  - update time (in the case of dynamic data sets).

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

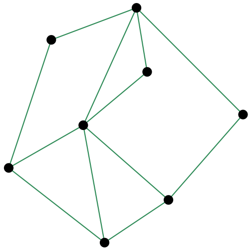
3 Geometric Searching

- Introduction
- Point Inclusion
  - Point-in-Polygon Query
  - Triangulation Refinement Technique

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

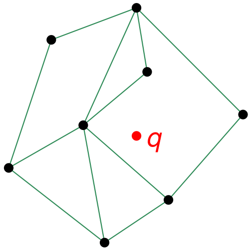
Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$,



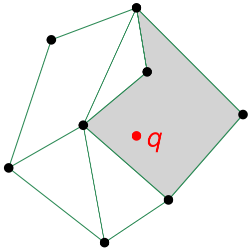
Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.



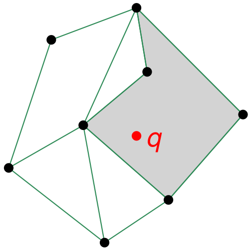
Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.
- Problem: Which face of $\mathcal{G}$ contains q?
- Aka "point location". Obviously, point-inclusion problems can also be studied in higher dimensions.



Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.
- Problem: Which face of $\mathcal{G}$ contains q?
- Aka "point location". Obviously, point-inclusion problems can also be studied in higher dimensions.

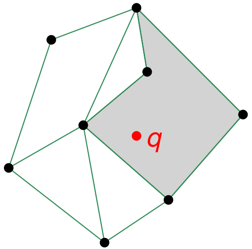


Theorem 44

A brute-force point location within an n-vertex planar subdivision can be carried out in $O(n)$ time, using $O(n)$ space.

Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.
- Problem: Which face of $\mathcal{G}$ contains q?
- Aka "point location". Obviously, point-inclusion problems can also be studied in higher dimensions.



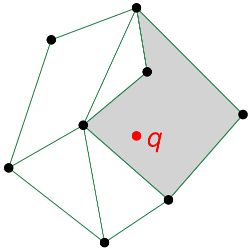
Theorem 44

A brute-force point location within an n-vertex planar subdivision can be carried out in $O(n)$ time, using $O(n)$ space.

Sketch of Proof : An "even-odd" test can be used for the individual point-in-face tests. □

Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.
- Problem: Which face of $\mathcal{G}$ contains q?
- Aka "point location". Obviously, point-inclusion problems can also be studied in higher dimensions.



Theorem 44

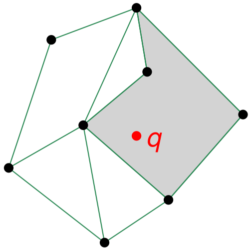
A brute-force point location within an n-vertex planar subdivision can be carried out in $O(n)$ time, using $O(n)$ space.

Sketch of Proof: An "even-odd" test can be used for the individual point-in-face tests. □

- This is not an efficient solution for repetitive-mode queries!

Point Inclusion

- Given: Decomposition of the plane into polygonal regions, as induced by a PSLG $\mathcal{G}$, and a query point q.
- Problem: Which face of $\mathcal{G}$ contains q?
- Aka “point location”. Obviously, point-inclusion problems can also be studied in higher dimensions.



Theorem 44

A brute-force point location within an n-vertex planar subdivision can be carried out in $O(n)$ time, using $O(n)$ space.

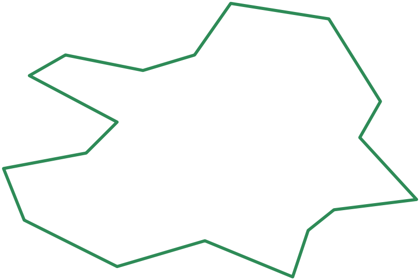
Sketch of Proof : An “even-odd” test can be used for the individual point-in-face tests. □

- This is not an efficient solution for repetitive-mode queries!
- Goal: Create geometric data structure that supports some kind of binary search.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

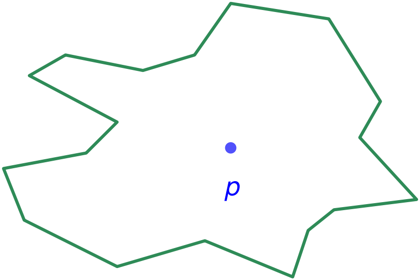
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing:



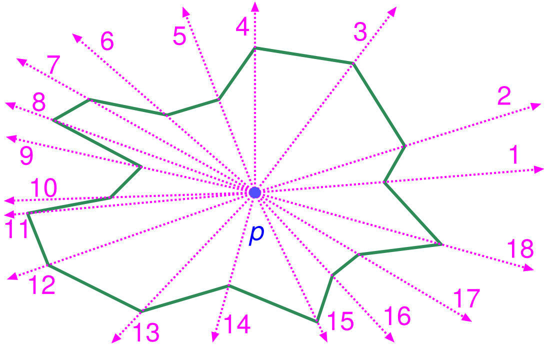
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point *p* within kernel of polygon.



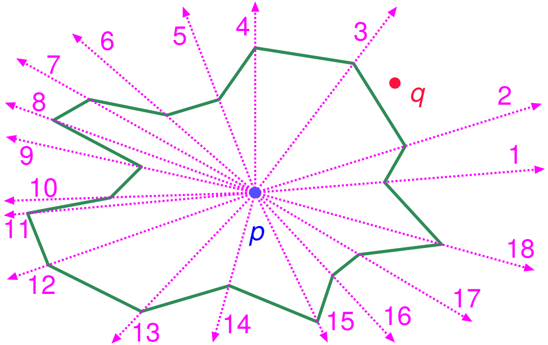
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.



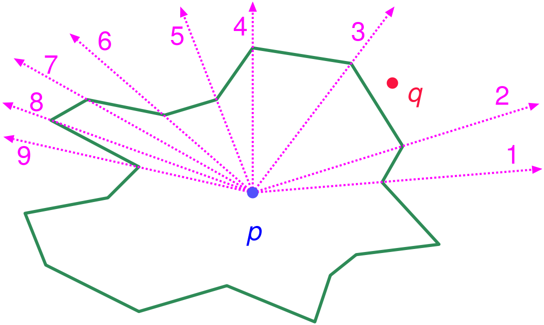
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.
3. For a query point q:



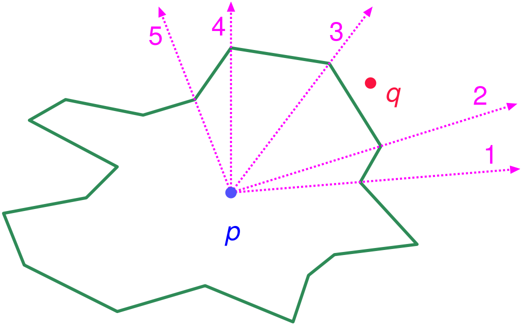
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.
3. For a query point q: Perform binary search.



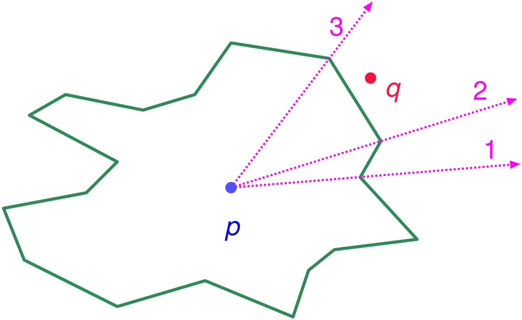
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point *p* within kernel of polygon.
2. Preprocessing: Shoot rays starting at *p* through each vertex.
3. For a query point *q*: Perform binary search.



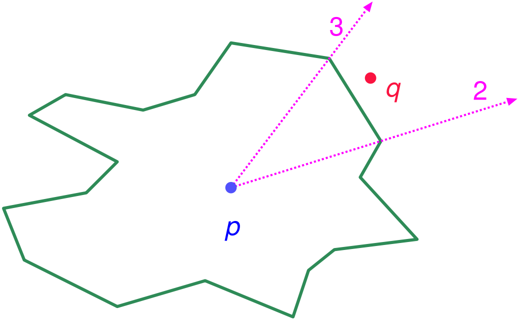
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.
3. For a query point q: Perform binary search.



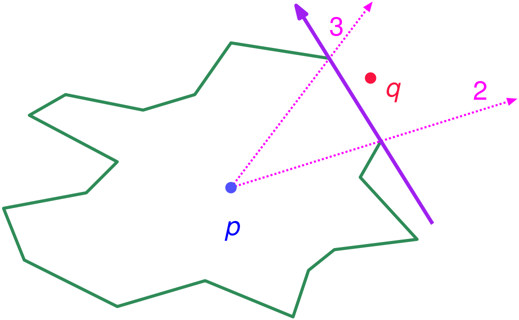
Point-in-Polygon Query for Star-Shaped Polygons

1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.
3. For a query point q: Perform binary search.



Point-in-Polygon Query for Star-Shaped Polygons

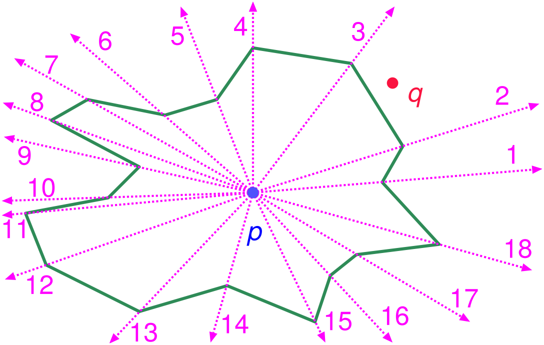
1. Preprocessing: Find point p within kernel of polygon.
2. Preprocessing: Shoot rays starting at p through each vertex.
3. For a query point q: Perform binary search.
4. Determine sidedness relative to one edge of the polygon.



Point-in-Polygon Query for Star-Shaped Polygons

Theorem 45

For an n-vertex star-shaped polygon, a point-location query can be answered in $O(\log n)$ query time, after $O(n)$ preprocessing and within $O(n)$ space.

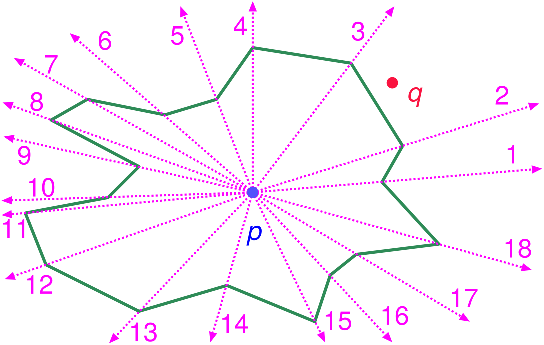


Point-in-Polygon Query for Star-Shaped Polygons

Theorem 45

For an n-vertex star-shaped polygon, a point-location query can be answered in $O(\log n)$ query time, after $O(n)$ preprocessing and within $O(n)$ space.

Sketch of Proof: Determining a point p within the kernel can be seen as a solution of an LP, which can be obtained in $O(n)$ time [Megiddo (1983)]. □



Triangulation Refinement Technique

- Aka: Dobkin-Kirkpatrick Hierarchy [1990] in 3D, based on Kirkpatrick [1983].

Triangulation Refinement Technique

- Aka: Dobkin-Kirkpatrick Hierarchy [1990] in 3D, based on Kirkpatrick [1983].
- For a given n-vertex PSLG $\mathcal{G}$, generate a PSLG $\mathcal{G}'$ with the following properties:
  1. $\mathcal{G}'$ is a super-graph of $\mathcal{G}$,
  2. $\mathcal{G}'$ is a triangulation,

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation Refinement Technique

- Aka: Dobkin-Kirkpatrick Hierarchy [1990] in 3D, based on Kirkpatrick [1983].
- For a given n-vertex PSLG $\mathcal{G}$, generate a PSLG $\mathcal{G}'$ with the following properties:
  1. $\mathcal{G}'$ is a super-graph of $\mathcal{G}$,
  2. $\mathcal{G}'$ is a triangulation,
  3. $\mathcal{G}'$ has a triangular boundary, and, by Euler's formula,
  4. $\mathcal{G}'$ has exactly $3n - 6$ edges.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation Refinement Technique

- Aka: Dobkin-Kirkpatrick Hierarchy [1990] in 3D, based on Kirkpatrick [1983].
- For a given n-vertex PSLG $\mathcal{G}$, generate a PSLG $\mathcal{G}'$ with the following properties:
  1. $\mathcal{G}'$ is a super-graph of $\mathcal{G}$,
  2. $\mathcal{G}'$ is a triangulation,
  3. $\mathcal{G}'$ has a triangular boundary, and, by Euler's formula,
  4. $\mathcal{G}'$ has exactly $3n - 6$ edges.

Triangulation refinement in a nutshell

- Construct hierarchy of triangulations above $\mathcal{G}'$, and set up a directed acyclic search graph $\mathcal{T}$, in time $O(n \log n)$ and space $O(n)$.

Triangulation Refinement Technique

- Aka: Dobkin-Kirkpatrick Hierarchy [1990] in 3D, based on Kirkpatrick [1983].
- For a given n-vertex PSLG $\mathcal{G}$, generate a PSLG $\mathcal{G}'$ with the following properties:
  1. $\mathcal{G}'$ is a super-graph of $\mathcal{G}$,
  2. $\mathcal{G}'$ is a triangulation,
  3. $\mathcal{G}'$ has a triangular boundary, and, by Euler's formula,
  4. $\mathcal{G}'$ has exactly $3n - 6$ edges.

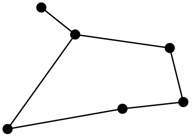
Triangulation refinement in a nutshell

- Construct hierarchy of triangulations above $\mathcal{G}'$, and set up a directed acyclic search graph $\mathcal{T}$, in time $O(n \log n)$ and space $O(n)$.
- Perform point-location queries within $\mathcal{T}$ in time $O(\log n)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation Refinement Technique

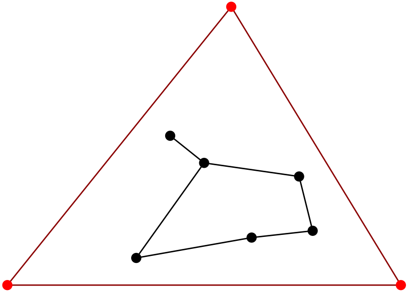
- Convert $\mathcal{G}$ into a triangulation $\mathcal{G}'$ with triangular outer face.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

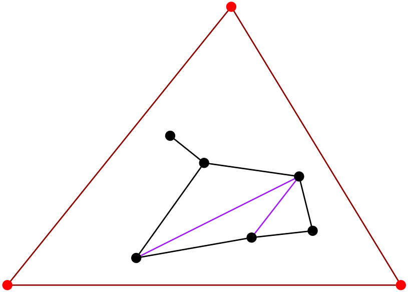
Triangulation Refinement Technique

- Convert $\mathcal{G}$ into a triangulation $\mathcal{G}'$ with triangular outer face.



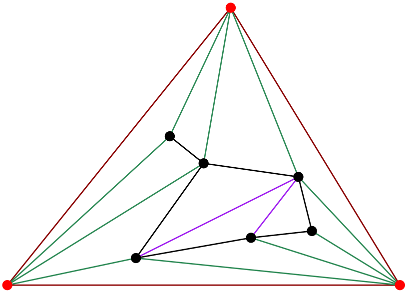
Triangulation Refinement Technique

- Convert $\mathcal{G}$ into a triangulation $\mathcal{G}'$ with triangular outer face.



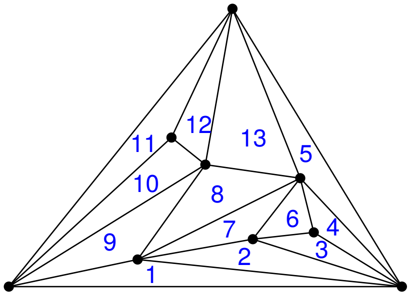
Triangulation Refinement Technique

- Convert $\mathcal{G}$ into a triangulation $\mathcal{G}'$ with triangular outer face.



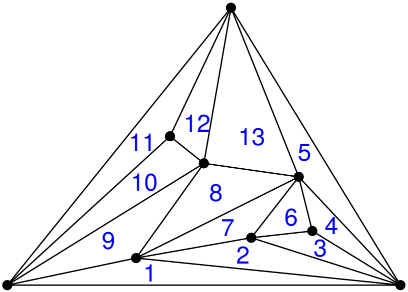
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.

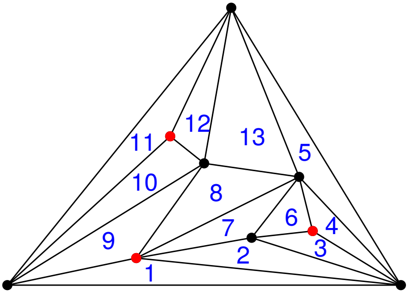


Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:

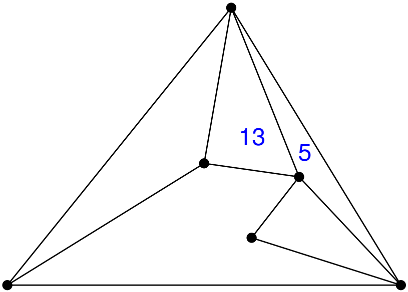


- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.



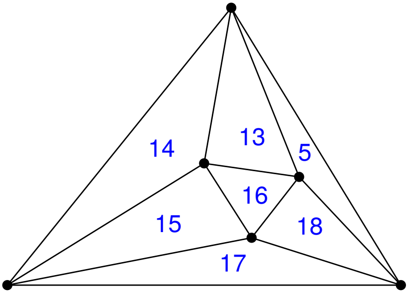
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, ..., S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.



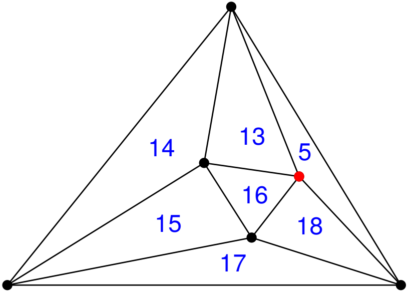
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \dots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



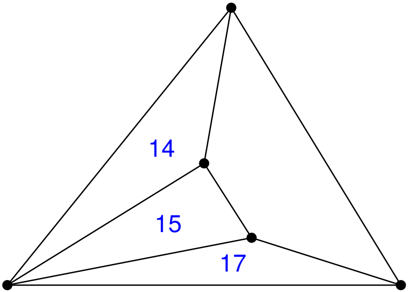
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, ..., S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



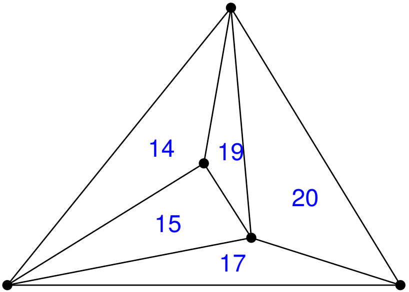
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



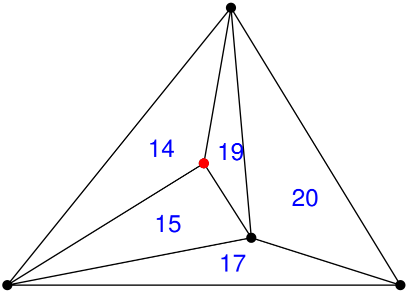
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



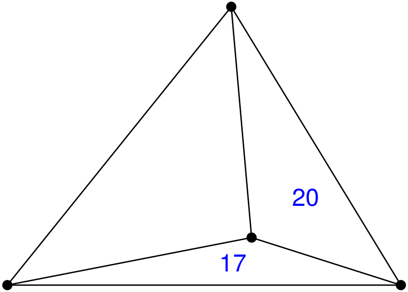
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, ..., S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



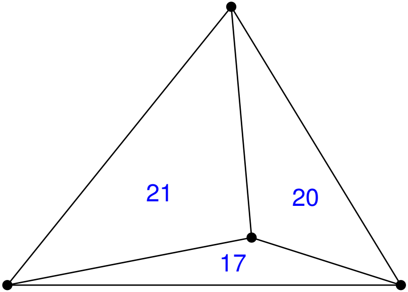
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



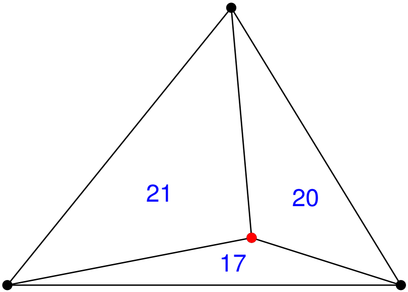
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.



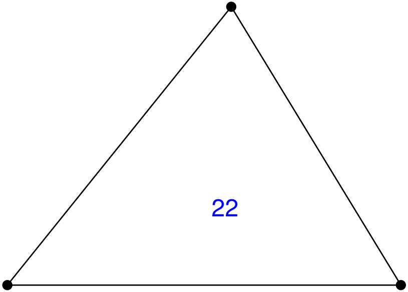
Triangulation Refinement: Hierarchy of Triangulations

- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, ..., S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.

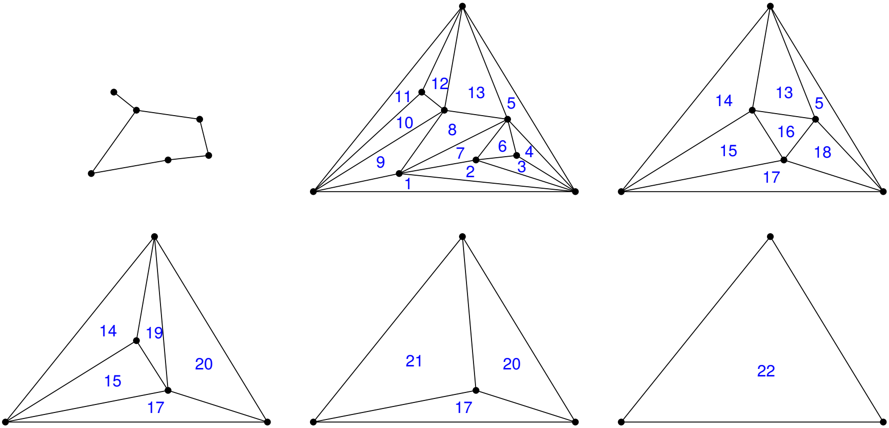


Triangulation Refinement: Hierarchy of Triangulations

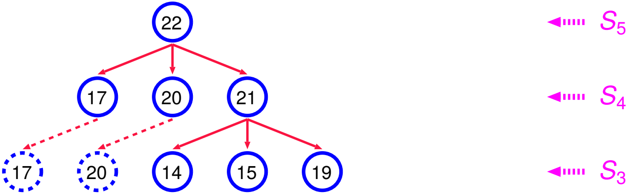
- Consider a triangulated PSLG $\mathcal{G}'$ (with triangular outer face) on n vertices.
- We construct a hierarchy of triangulations $S_1, S_2, \ldots, S_{h(n)}$, where $S_1 := \mathcal{G}'$ and S_i is obtained from S_{i-1} as follows:
  - **Step 1**: Select and remove a maximal independent set of non-boundary vertices of S_{i-1} together with their incident edges.
  - **Step 2**: Re-triangulate the holes arising from the removal of those vertices and edges.
- The final triangulation in the hierarchy, $S_{h(n)}$, is just one triangle.



Triangulation Refinement: Hierarchy of Triangulations

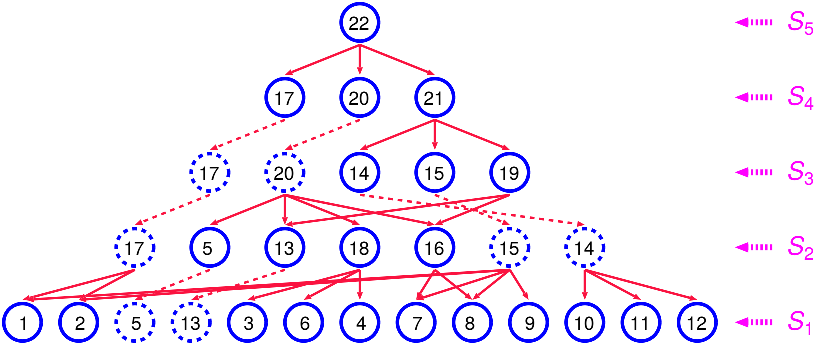


- We set up a directed acyclic search graph $\mathcal{T}$ on $S_1, S_2, \ldots, S_{h(n)}$.
- The graph $\mathcal{T}$ contains an edge from triangle Δ_k to triangle Δ_j if, when constructing triangulation S_i from triangulation S_{i-1}, we have:
  1. Δ_j is removed from S_{i-1} in **Step 1**.
  2. Δ_k is created in S_i in **Step 2**.
  3. $\Delta_j \cap \Delta_k \neq \emptyset$.



Triangulation Refinement: Directed Acyclic Search Graph

- We set up a directed acyclic search graph $\mathcal{T}$ on $S_1, S_2, \ldots, S_{h(n)}$.
- The graph $\mathcal{T}$ contains an edge from triangle Δ_k to triangle Δ_j if, when constructing triangulation S_i from triangulation S_{i-1}, we have:
  1. Δ_j is removed from S_{i-1} in **Step 1**.
  2. Δ_k is created in S_i in **Step 2**.
  3. $\Delta_j \cap \Delta_k \neq \emptyset$.



Triangulation Refinement Technique: Query

- For a query point q, perform a point-in-triangle test of q relative to the root triangle of $\mathcal{T}$.
- If q is
  - **outside** then q lies in the unbounded (exterior) face.
  - **inside** then q is tested for triangle inclusion with each of the descendants of the current node.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation Refinement Technique: Query

- For a query point q, perform a point-in-triangle test of q relative to the root triangle of $\mathcal{T}$.
- If q is
  - **outside** then q lies in the unbounded (exterior) face.
  - **inside** then q is tested for triangle inclusion with each of the descendants of the current node.
- This scheme is applied recursively until a leaf of $\mathcal{T}$ is reached.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation Refinement Technique: Query

- For a query point q, perform a point-in-triangle test of q relative to the root triangle of $\mathcal{T}$.
- If q is
  - **outside** then q lies in the unbounded (exterior) face.
  - **inside** then q is tested for triangle inclusion with each of the descendants of the current node.
- This scheme is applied recursively until a leaf of $\mathcal{T}$ is reached.
- What is the complexity of a query? This depends on
  - the height $h(n)$ of $\mathcal{T}$,
  - the maximum number m of point-in-triangle tests needed per node.

Triangulation Refinement Technique: Query

- For a query point q, perform a point-in-triangle test of q relative to the root triangle of $\mathcal{T}$.
- If q is
  - **outside** then q lies in the unbounded (exterior) face.
  - **inside** then q is tested for triangle inclusion with each of the descendants of the current node.
- This scheme is applied recursively until a leaf of $\mathcal{T}$ is reached.
- What is the complexity of a query? This depends on
  - the height $h(n)$ of $\mathcal{T}$,
  - the maximum number m of point-in-triangle tests needed per node.
- Both terms seem to depend on how we select those vertices of S_{i-1} that will not be part of S_i.
- Goal: Construct $\mathcal{T}$ such that $m = O(1)$ and $h(n) = O(\log n)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Analysis of Triangulation Refinement Technique

- Let N_i denote the number of vertices of triangulation S_i.
- Criterion for selecting vertices that are to be removed:

  Remove a set of non-adjacent vertices of degree less than K,

  where $K := 12$.

Analysis of Triangulation Refinement Technique

- Let N_i denote the number of vertices of triangulation S_i.
- Criterion for selecting vertices that are to be removed:

  Remove a set of non-adjacent vertices of degree less than K,

  where $K := 12$.
- Easy to prove: This criterion allows us to delete at least

  $$\frac{1}{12}\left(\frac{N_{i-1}}{2} - 3\right)$$

  vertices within S_{i-1}.

Analysis of Triangulation Refinement Technique

- Let N_i denote the number of vertices of triangulation S_i.
- Criterion for selecting vertices that are to be removed:

  Remove a set of non-adjacent vertices of degree less than K,

  where $K := 12$.
- Easy to prove: This criterion allows us to delete at least

$$\frac{1}{12}\left(\frac{N_{i-1}}{2} - 3\right)$$

  vertices within S_{i-1}.
- We get:
  - $N_i \le \alpha N_{i-1}$, where $\alpha \approx \frac{23}{24}$.
  - $h(n) \le \lceil \log_{1/\alpha} n \rceil \approx 16 \log n$, and only $O(n)$ triangles need to be stored.
  - Finally, $m = (K - 1) - 2 = 9$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Analysis of Triangulation Refinement Technique

- Let N_i denote the number of vertices of triangulation S_i.
- Criterion for selecting vertices that are to be removed:

  Remove a set of non-adjacent vertices of degree less than K*,*

  where $K := 12$.
- Easy to prove: This criterion allows us to delete at least

$$\frac{1}{12}\left(\frac{N_{i-1}}{2} - 3\right)$$

  vertices within S_{i-1}.
- We get:
  - $N_i \leq \alpha N_{i-1}$, where $\alpha \approx \frac{23}{24}$.
  - $h(n) \leq \lceil \log_{1/\alpha} n \rceil \approx 16 \log n$, and only $O(n)$ triangles need to be stored.
  - Finally, $m = (K - 1) - 2 = 9$.

- Other choices for K yield tighter bounds! E.g., $K := 9$ yields the slightly better bounds $\alpha \approx \frac{17}{18}$ and $12 \log n$ per query, and more elaborate choices for the vertices to be deleted bring down the query complexity to roughly $\frac{9}{2} \log n$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Analysis of Triangulation Refinement Technique

- The key step in the preprocessing is the initial triangulation of the PSLG, which takes $O(n \log n)$ time (or $O(n)$ time if the PSLG is connected and Chazelle's linear-time triangulation algorithm is used).

Analysis of Triangulation Refinement Technique

- The key step in the preprocessing is the initial triangulation of the PSLG, which takes $O(n \log n)$ time (or $O(n)$ time if the PSLG is connected and Chazelle's linear-time triangulation algorithm is used).
- All other triangulation operations can easily be carried out in time linear in the number of vertices involved.

Analysis of Triangulation Refinement Technique

- The key step in the preprocessing is the initial triangulation of the PSLG, which takes $O(n \log n)$ time (or $O(n)$ time if the PSLG is connected and Chazelle's linear-time triangulation algorithm is used).
- All other triangulation operations can easily be carried out in time linear in the number of vertices involved.

Theorem 46 (Kirkpatrick (1983))

For a connected n-vertex PSLG, triangulation refinement supports point-location queries in $O(\log n)$ query time, after $O(n)$ preprocessing and within $O(n)$ space.

Analysis of Triangulation Refinement Technique

- The key step in the preprocessing is the initial triangulation of the PSLG, which takes $O(n \log n)$ time (or $O(n)$ time if the PSLG is connected and Chazelle's linear-time triangulation algorithm is used).
- All other triangulation operations can easily be carried out in time linear in the number of vertices involved.

Theorem 46 (Kirkpatrick (1983))

For a connected n-vertex PSLG, triangulation refinement supports point-location queries in $O(\log n)$ query time, after $O(n)$ preprocessing and within $O(n)$ space.

- Although this point-inclusion algorithm is optimum in terms of the O-notation, it is not very practical and better (but more elaborate) algorithms are known.

4 Convex Hulls

- Basics
- Algorithms
- Convex Hull of Polygons
- Convex Hulls in 3D
- Applications of Convex Hulls

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

4 **Convex Hulls**

- Basics
  - Definition
  - Time Complexity
  - Heuristic for Speeding Up Convex-Hull Computations
- Algorithms
- Convex Hull of Polygons
- Convex Hulls in 3D
- Applications of Convex Hulls

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Linear Combination and Convex Combination

Definition 47 (Linear combination, Dt.: Linearkombination)

Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. A *linear combination* of $p_1, \ldots, p_k$ is given by

$$\sum_{i=1}^{k} \lambda_i \, \mathfrak{p}_i$$

where $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$ are scalars.

Linear Combination and Convex Combination

Definition 47 (Linear combination, Dt.: Linearkombination)

Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. A *linear combination* of $p_1, \ldots, p_k$ is given by

$$\sum_{i=1}^{k} \lambda_i \, \mathfrak{p}_i$$

where $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$ are scalars.

Definition 48 (Convex combination, Dt.: Konvexkombination)

Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. A *convex combination* of $p_1, \ldots, p_k$ is given by

$$\sum_{i=1}^{k} \lambda_i \, \mathfrak{p}_i \quad \text{with} \quad \sum_{i=1}^{k} \lambda_i = 1 \quad \text{and} \quad \forall (1 \leq i \leq k) \;\; \lambda_i \geq 0,$$

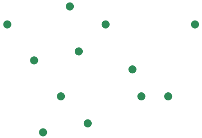
where $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$ are scalars.

Convex Hull

Definition 49 (Convex hull, Dt.: konvexe Hülle)

Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. The *convex hull* of $p_1, \ldots, p_k$ is the set

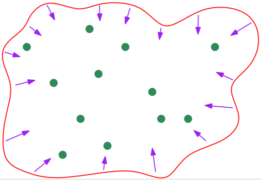
$$\{\sum_{i=1}^{k} \lambda_i \mathfrak{p}_i : \ \lambda_1, \ldots \lambda_k \in \mathbb{R}_0^+ \text{ and } \sum_{i=1}^{k} \lambda_i \ = \ 1\}.$$



Definition 49 (Convex hull, Dt.: konvexe Hülle)

Let p_1, p_2, ..., p_k be k points in $\mathbb{R}^n$. The *convex hull* of p_1, ..., p_k is the set

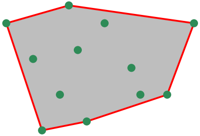
$$\{\sum_{i=1}^{k} \lambda_i \mathfrak{p}_i : \ \lambda_1, \ldots \lambda_k \in \mathbb{R}_0^+ \text{ and } \sum_{i=1}^{k} \lambda_i = 1\}.$$



Definition 49 (Convex hull, Dt.: konvexe Hülle)

Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. The *convex hull* of $p_1, \ldots, p_k$ is the set

$$\{\sum_{i=1}^{k} \lambda_i \mathfrak{p}_i : \lambda_1, \ldots \lambda_k \in \mathbb{R}_0^+ \text{ and } \sum_{i=1}^{k} \lambda_i = 1\}.$$



Definition 49 (Convex hull, Dt.: konvexe Hülle)

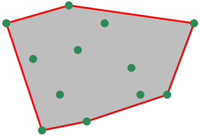
Let $p_1, p_2, \ldots, p_k$ be k points in $\mathbb{R}^n$. The *convex hull* of $p_1, \ldots, p_k$ is the set

$$\{\sum_{i=1}^{k} \lambda_i \mathfrak{p}_i : \lambda_1, \ldots \lambda_k \in \mathbb{R}_0^+ \text{ and } \sum_{i=1}^{k} \lambda_i = 1\}.$$

For a set $S \subseteq \mathbb{R}^n$ (with possibly infinitely many points), the *convex hull* of S is the set

$$\{\sum_{i=1}^{k} \lambda_i \mathfrak{p}_i : k \in \mathbb{N} \text{ and } p_1, p_2, \ldots, p_k \in S \text{ and } \lambda_1, \ldots \lambda_k \in \mathbb{R}_0^+ \text{ and } \sum_{i=1}^{k} \lambda_i = 1\}.$$

The convex hull of S is commonly denoted by $CH(S)$.



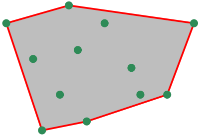
Convexity

Definition 50 (Convex set, Dt.: konvexe Menge)

A set $S \subseteq \mathbb{R}^n$ is called *convex* if for all $p, q \in S$

$$\overline{pq} \subseteq S$$

where $\overline{pq}$ denotes the straight-line segment between p and q.



Convexity

Definition 50 (Convex set, Dt.: konvexe Menge)

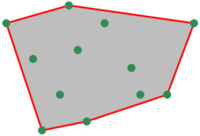
A set $S \subseteq \mathbb{R}^n$ is called *convex* if for all $p, q \in S$

$$\overline{pq} \subseteq S$$

where $\overline{pq}$ denotes the straight-line segment between p and q.

Lemma 51

For $S \subseteq \mathbb{R}^n$, the convex hull $CH(S)$ of S is a convex set.



Definition 50 (Convex set, Dt.: konvexe Menge)

A set $S \subseteq \mathbb{R}^n$ is called *convex* if for all $p, q \in S$

$$\overline{pq} \subseteq S$$

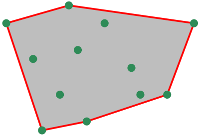
where $\overline{pq}$ denotes the straight-line segment between p and q.

Lemma 51

For $S \subseteq \mathbb{R}^n$, the convex hull $CH(S)$ of S is a convex set.

Lemma 52

For a set S of n points in $\mathbb{R}^2$, the convex hull $CH(S)$ is a convex polygon.

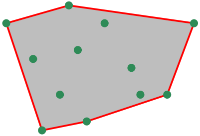


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 53 (Convex superset)

A set $B \subseteq \mathbb{R}^n$ is called a *convex superset* of a set $A \subseteq \mathbb{R}^n$ if

$$A \subseteq B \quad \text{and} \quad B \text{ is convex.}$$



Convexity

Definition 53 (Convex superset)

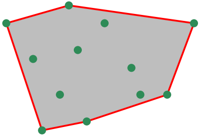
A set $B \subseteq \mathbb{R}^n$ is called a *convex superset* of a set $A \subseteq \mathbb{R}^n$ if

$$A \subseteq B \quad \text{and} \quad B \text{ is convex.}$$

Lemma 54

For $A \subseteq \mathbb{R}^n$, the following definitions are equivalent to Def. 49:

- $CH(A)$ is the smallest convex superset of A.
- $CH(A)$ is the intersection of all convex supersets of A.



Convexity

Definition 53 (Convex superset)

A set $B \subseteq \mathbb{R}^n$ is called a *convex superset* of a set $A \subseteq \mathbb{R}^n$ if

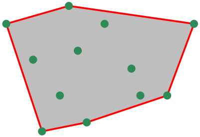
$$A \subseteq B \quad \text{and} \quad B \text{ is convex.}$$

Lemma 54

For $A \subseteq \mathbb{R}^n$, the following definitions are equivalent to Def. 49:

- $CH(A)$ is the smallest convex superset of A.
- $CH(A)$ is the intersection of all convex supersets of A.

- The definition of a convex hull (and of convexity) is readily extended from $\mathbb{R}^n$ to other vector spaces over $\mathbb{R}$.



Complexity of Computing Convex Hulls

Problem: CONVEXHULL

Given: A set S of n points in the plane.

Compute: The convex hull $CH(S)$, as an ordered list of vertices.

Complexity of Computing Convex Hulls

Problem: CONVEXHULL

Given: A set S of n points in the plane.

Compute: The convex hull $CH(S)$, as an ordered list of vertices.

- Question: Can we state a worst-case lower bound on the time complexity of CONVEXHULL, i.e., for computing $CH(S)$?

Problem: CONVEXHULL

Given: A set S of n points in the plane.

Compute: The convex hull $CH(S)$, as an ordered list of vertices.

- Question: Can we state a worst-case lower bound on the time complexity of CONVEXHULL, i.e., for computing $CH(S)$?

Theorem 55

SORTING is linear-time transformable to CONVEXHULL.

Complexity of Computing Convex Hulls

Problem: CONVEXHULL

Given: A set S of n points in the plane.

Compute: The convex hull $CH(S)$, as an ordered list of vertices.

- Question: Can we state a worst-case lower bound on the time complexity of CONVEXHULL, i.e., for computing $CH(S)$?

Theorem 55

SORTING is linear-time transformable to CONVEXHULL.

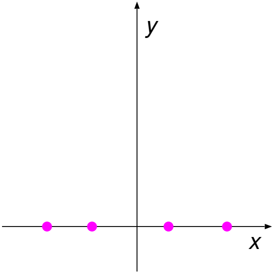
Corollary 56

Solving CONVEXHULL for n points requires at least $\Omega(n \log n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sketch of Proof : of Theorem 55

- Suppose the instance of SORTING is the set of $S' := \{x_1, x_2, ..., x_n\} \subset \mathbb{R}$.

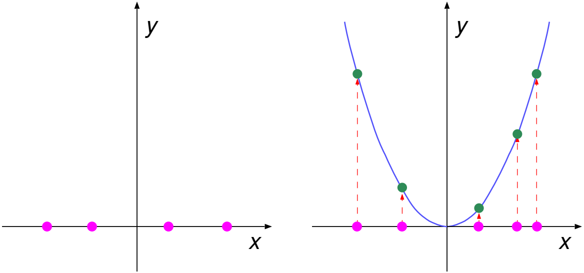


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Reduction From Sorting to Convex Hulls

Sketch of Proof : of Theorem 55

- Suppose the instance of SORTING is the set of $S' := \{x_1, x_2, ..., x_n\} \subset \mathbb{R}$.
- We transform S' into an instance of CONVEXHULL by mapping each real number x_i to the point $(x_i, {x_i}^2)$. All points of the resulting set S of points lie on the parabola $y = x^2$.

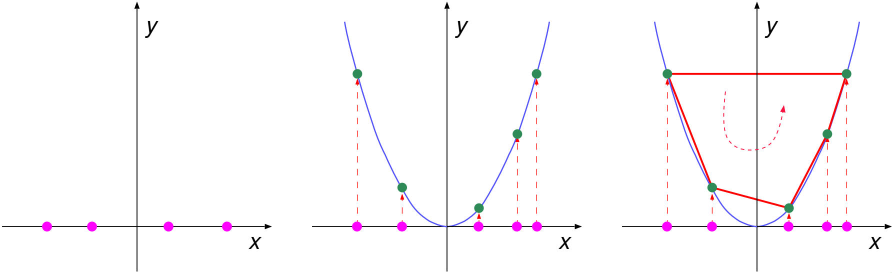


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Reduction From Sorting to Convex Hulls

Sketch of Proof : of Theorem 55

- Suppose the instance of SORTING is the set of $S' := \{x_1, x_2, ..., x_n\} \subset \mathbb{R}$.
- We transform S' into an instance of CONVEXHULL by mapping each real number x_i to the point $(x_i, {x_i}^2)$. All points of the resulting set S of points lie on the parabola $y = x^2$.
- The convex hull of S contains a list of vertices sorted by x-coordinates.
- One pass through this list will find the smallest element. The sorted numbers can be obtained by a second pass through this list.



Complexity of Computing Convex Hulls

- The $\Omega(n \log n)$ lower bound also applies if only the unordered set of hull vertices is sought. (But the proof becomes a bit trickier . . .)

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Complexity of Computing Convex Hulls

- The $\Omega(n \log n)$ lower bound also applies if only the unordered set of hull vertices is sought. (But the proof becomes a bit trickier . . .)
- If also the size h of the output is considered (in addition to the input size n), then one can prove the lower bound $\Omega(n \log h)$.

Complexity of Computing Convex Hulls

- The $\Omega(n \log n)$ lower bound also applies if only the unordered set of hull vertices is sought. (But the proof becomes a bit trickier ...)
- If also the size h of the output is considered (in addition to the input size n), then one can prove the lower bound $\Omega(n \log h)$.
- This lower bound is matched by a “marriage-before-conquest” algorithm (Kirkpatrick&Seidel) and by Chan’s algorithm. Chan’s algorithm is simpler and also extends to 3D.

Theorem 57 (Kirkpatrick&Seidel (1986), Chan (1996))

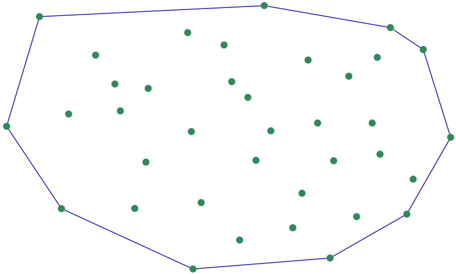
The convex hull of n points in the plane can be computed in $O(n \log h)$ time and within $O(n)$ storage, where h denotes the number of vertices of $CH(S)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Discarding Internal Points

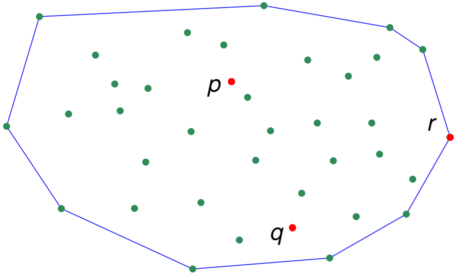
Lemma 58

Consider three points $p, q, r \in CH(S)$.



Lemma 58

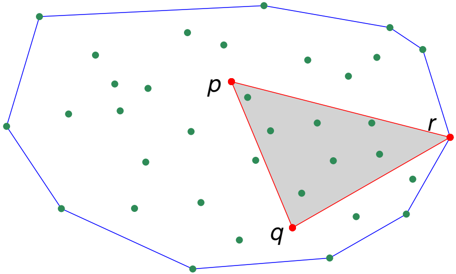
Consider three points $p, q, r \in CH(S)$.



Discarding Internal Points

Lemma 58

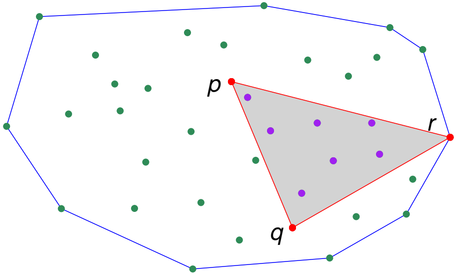
Consider three points $p, q, r \in CH(S)$. Then every point q that lies strictly within $\Delta(p, q, r)$ is internal to $CH(S)$.



Discarding Internal Points

Lemma 58

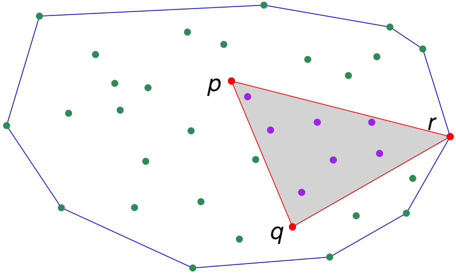
Consider three points $p, q, r \in CH(S)$. Then every point q that lies strictly within $\Delta(p, q, r)$ is internal to $CH(S)$.



- In particular, no point strictly within $\Delta(p, q, r)$ can be a vertex of the convex hull.

Lemma 58

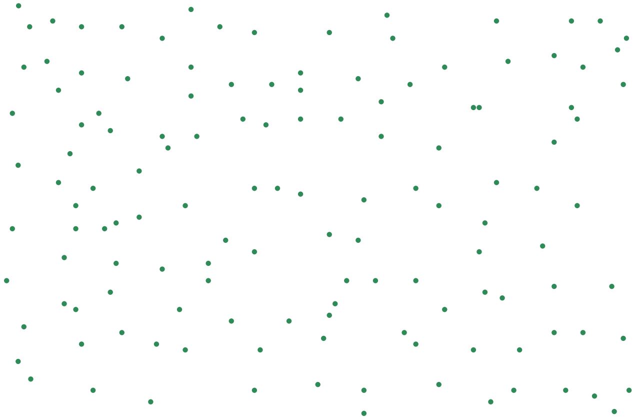
Consider three points $p, q, r \in CH(S)$. Then every point q that lies strictly within $\Delta(p, q, r)$ is internal to $CH(S)$.



- In particular, no point strictly within $\Delta(p, q, r)$ can be a vertex of the convex hull.
- This lemma can be generalized to any convex quadrangle (or polygon) whose vertices lie within $CH(S)$.

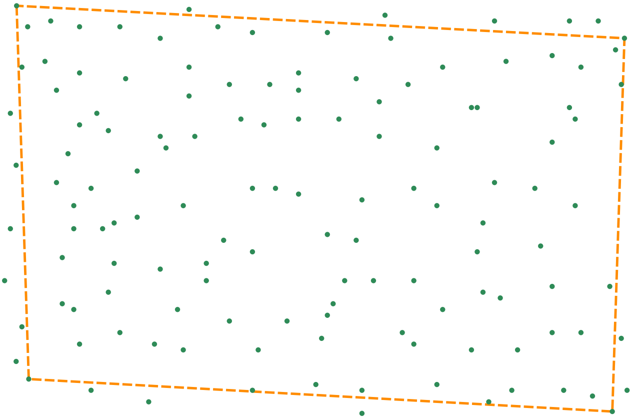
Discarding Internal Points: Interior Elimination

- Discard all points within a large (axis-aligned) rectangle.



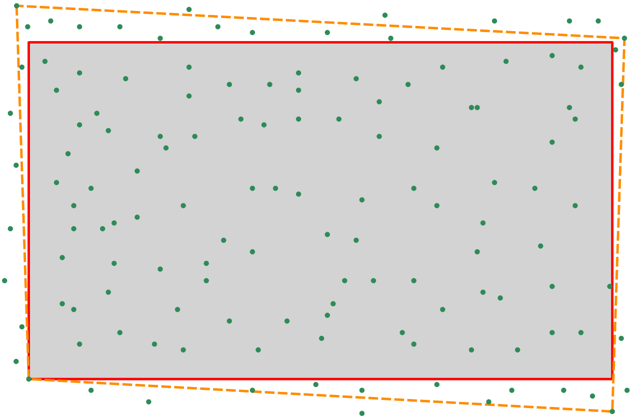
Discarding Internal Points: Interior Elimination

- Discard all points within a large (axis-aligned) rectangle.



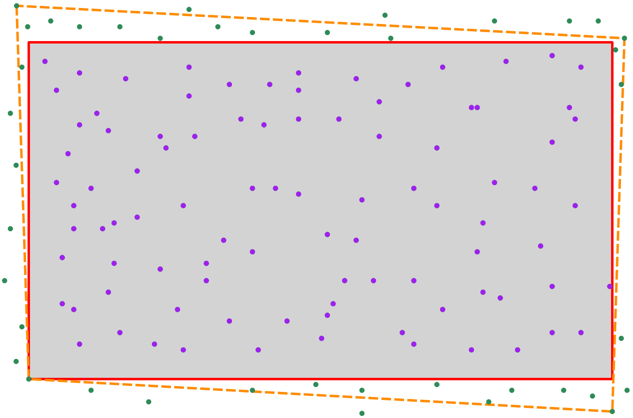
Discarding Internal Points: Interior Elimination

- Discard all points within a large (axis-aligned) rectangle.



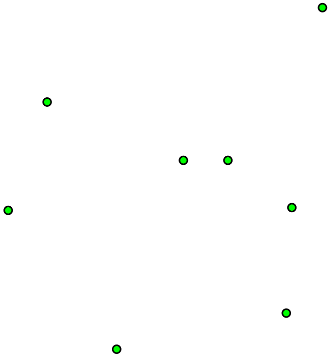
Discarding Internal Points: Interior Elimination

- Discard all points within a large (axis-aligned) rectangle.
- Heuristic improvement; does not change worst-case complexity.



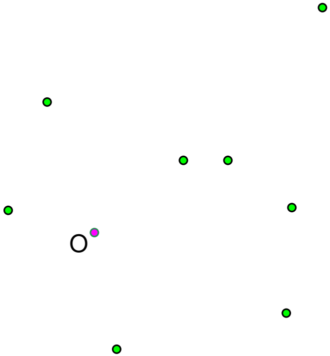
4 Convex Hulls

Graham’s Scan



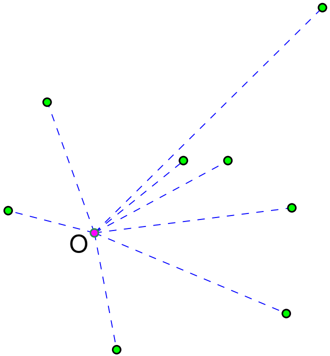
Graham's Scan

- Find a point O internal to $CH(S)$, e.g, the center of three points of S.

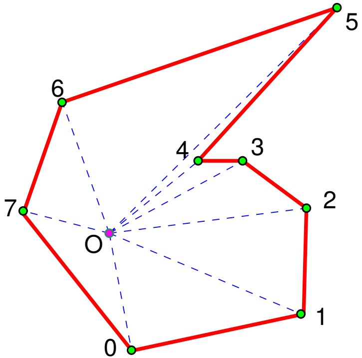


Graham's Scan

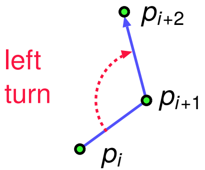
- Find a point O internal to $CH(S)$, e.g, the center of three points of S.
- Sort the n points of S lexicographically on
  1. polar angle relative to O,
  2. distance from O.



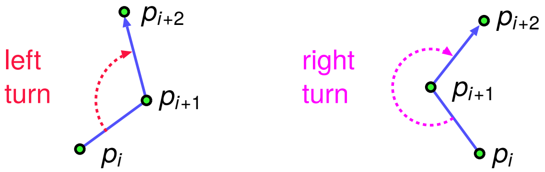
- Find a point O internal to $CH(S)$, e.g, the center of three points of S.
- Sort the n points of S lexicographically on
  1. polar angle relative to O,
  2. distance from O.
- Choose a point $p_0 \in S$ guaranteed to be a vertex of $CH(S)$, and re-number the points.



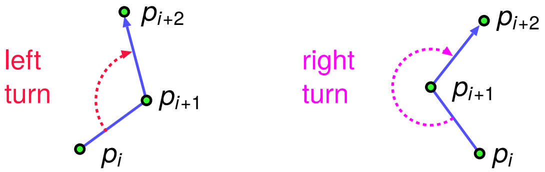
- CCW scan algorithm: The algorithm repeatedly examines triangles defined by triples of consecutive points $\triangle(p_i, p_{i+1}, p_{i+2})$:
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a left turn, advance to $\triangle(p_{i+1}, p_{i+2}, p_{i+3})$.



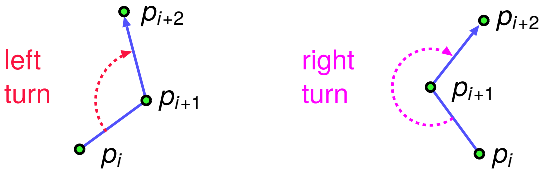
- CCW scan algorithm: The algorithm repeatedly examines triangles defined by triples of consecutive points $\triangle(p_i, p_{i+1}, p_{i+2})$:
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a left turn, advance to $\triangle(p_{i+1}, p_{i+2}, p_{i+3})$.
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a right turn, eliminate p_{i+1} from S and backtrack to $\triangle(p_{i-1}, p_i, p_{i+2})$.



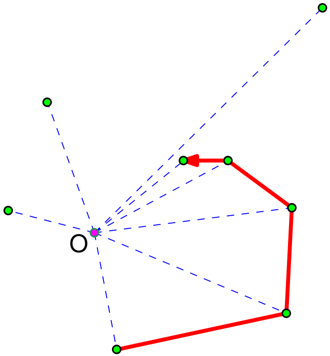
- CCW scan algorithm: The algorithm repeatedly examines triangles defined by triples of consecutive points $\triangle(p_i, p_{i+1}, p_{i+2})$:
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a left turn, advance to $\triangle(p_{i+1}, p_{i+2}, p_{i+3})$.
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a right turn, eliminate p_{i+1} from S and backtrack to $\triangle(p_{i-1}, p_i, p_{i+2})$.
  - If p_i, p_{i+1}, p_{i+2} are collinear then eliminate p_{i+1} and advance to $\triangle(p_i, p_{i+2}, p_{i+3})$.



- CCW scan algorithm: The algorithm repeatedly examines triangles defined by triples of consecutive points $\triangle(p_i, p_{i+1}, p_{i+2})$:
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a left turn, advance to $\triangle(p_{i+1}, p_{i+2}, p_{i+3})$.
  - If $\triangle(p_i, p_{i+1}, p_{i+2})$ is a right turn, eliminate p_{i+1} from S and backtrack to $\triangle(p_{i-1}, p_i, p_{i+2})$.
  - If p_i, p_{i+1}, p_{i+2} are collinear then eliminate p_{i+1} and advance to $\triangle(p_i, p_{i+2}, p_{i+3})$.
  - Scan ends when it returns to p_0.

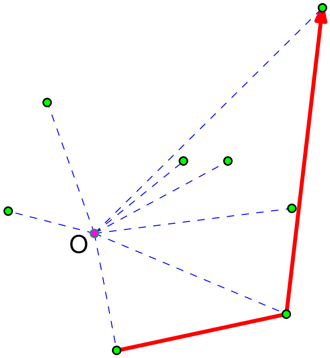


- Backtracking may occur more than once in succession, eliminating a sequence of points.

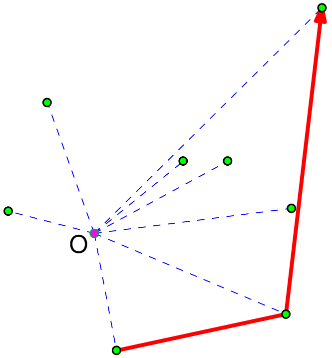


Graham's Scan: Advancing and Backtracking

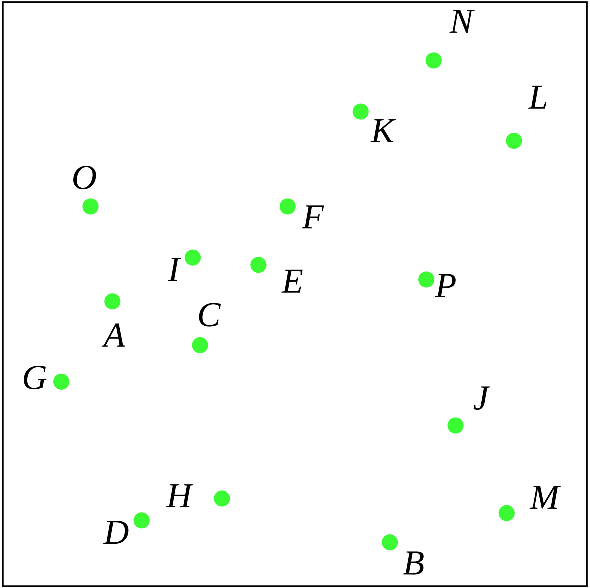
- Backtracking may occur more than once in succession, eliminating a sequence of points.



- Backtracking may occur more than once in succession, eliminating a sequence of points.
- Backtracking sure to stop at p_0.



Animation of Graham's Scan



Analysis of Graham's Scan

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

Proof :

1. Find a point O within $CH(S)$: $O(1)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Analysis of Graham's Scan

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

Proof :

1. Find a point O within $CH(S)$: $O(1)$.
2. Sort the points relative to O in polar coordinates: $O(n \log n)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Analysis of Graham's Scan

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

Proof :

1. Find a point O within $CH(S)$: $O(1)$.
2. Sort the points relative to O in polar coordinates: $O(n \log n)$.
3. Find a point on $CH(S)$: $O(n)$.

Analysis of Graham's Scan

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

Proof :

1. Find a point O within $CH(S)$: $O(1)$.
2. Sort the points relative to O in polar coordinates: $O(n \log n)$.
3. Find a point on $CH(S)$: $O(n)$.
4. Scan algorithm: Use amortized analysis to argue $O(n)$.

□

Theorem 59 (Complexity of Graham's Scan)

Graham's Scan computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

Proof :

1. Find a point O within $CH(S)$: $O(1)$.
2. Sort the points relative to O in polar coordinates: $O(n \log n)$.
3. Find a point on $CH(S)$: $O(n)$.
4. Scan algorithm: Use amortized analysis to argue $O(n)$.

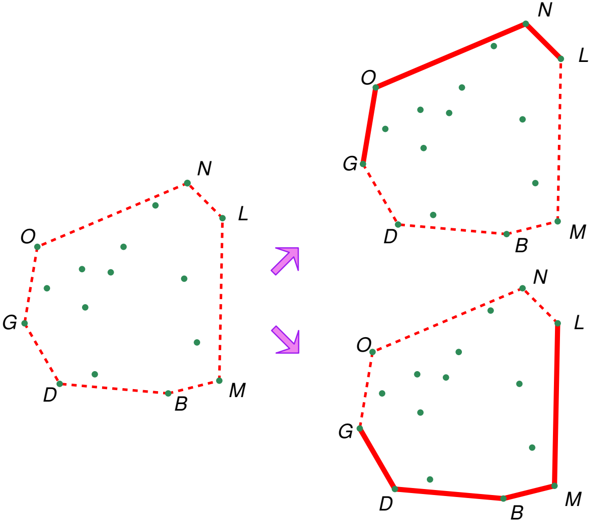
□

Corollary 60

Graham's Scan computes the convex hull of a star-shaped polygon in linear time.

Practice-Minded Simplification of Graham's Scan

- Compute upper and lower convex hull separately: Then a conventional lexicographical sort with respect to *x*-coordinates (and *y*-coordinates) suffices.



Divide-and-Conquer Convex Hull

1. If $|S| \leq k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.

Divide-and-Conquer Convex Hull

1. If $|S| \leq k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.
2. Partition the set S arbitrarily into two subsets S_1 and S_2 of approximately equal sizes.

Divide-and-Conquer Convex Hull

1. If $|S| \leq k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.
2. Partition the set S arbitrarily into two subsets S_1 and S_2 of approximately equal sizes.
3. Recursively find the convex hulls $CH(S_1)$ and $CH(S_2)$.

Divide-and-Conquer Convex Hull

1. If $|S| \leq k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.
2. Partition the set S arbitrarily into two subsets S_1 and S_2 of approximately equal sizes.
3. Recursively find the convex hulls $CH(S_1)$ and $CH(S_2)$.
4. Merge the two hulls together to form $CH(S)$.

Divide-and-Conquer Convex Hull

1. If $|S| \le k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.
2. Partition the set S arbitrarily into two subsets S_1 and S_2 of approximately equal sizes.
3. Recursively find the convex hulls $CH(S_1)$ and $CH(S_2)$.
4. Merge the two hulls together to form $CH(S)$.

Observations

- The convex hull of the union of the two subsets is the same as the convex hull of the union of the convex hulls of the two subsets.

Divide-and-Conquer Convex Hull

1. If $|S| \leq k_0$, where k_0 is a small integer (e.g., $k_0 = 3$), then construct the convex hull $CH(S)$ directly by some method and stop, else go to Step 2.
2. Partition the set S arbitrarily into two subsets S_1 and S_2 of approximately equal sizes.
3. Recursively find the convex hulls $CH(S_1)$ and $CH(S_2)$.
4. Merge the two hulls together to form $CH(S)$.

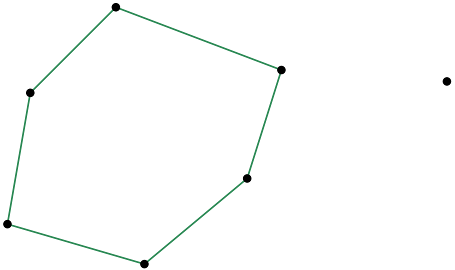
Observations

- The convex hull of the union of the two subsets is the same as the convex hull of the union of the convex hulls of the two subsets.
- Computing the convex hull of $CH(S_1) \cup CH(S_2)$ is relatively simple since $CH(S_1)$ and $CH(S_2)$ are convex polygons P_1, P_2 and, thus, have a natural ordering of their vertices.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

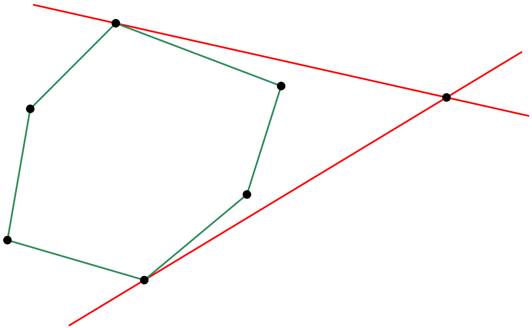
Definition 61 (Supporting line, Dt.: Stützgerade)

A *supporting line* of a convex polygon P is a straight line ℓ passing through a vertex of P such that the interior of P lies entirely to one side of ℓ.



Definition 61 (Supporting line, Dt.: Stützgerade)

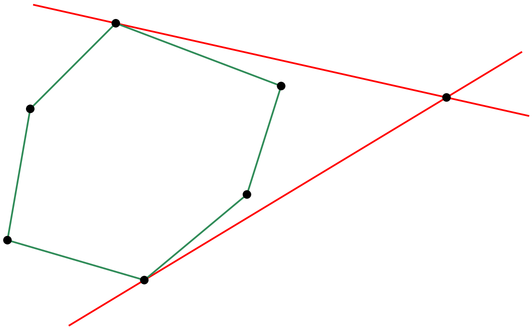
A *supporting line* of a convex polygon P is a straight line ℓ passing through a vertex of P such that the interior of P lies entirely to one side of ℓ.



Definition 61 (Supporting line, Dt.: Stützgerade)

A *supporting line* of a convex polygon P is a straight line ℓ passing through a vertex of P such that the interior of P lies entirely to one side of ℓ.

- This definition is readily generalized to general convex sets.

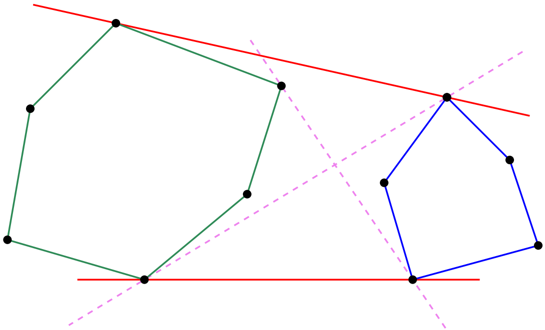


Divide-and-Conquer Convex Hull: Supporting Lines

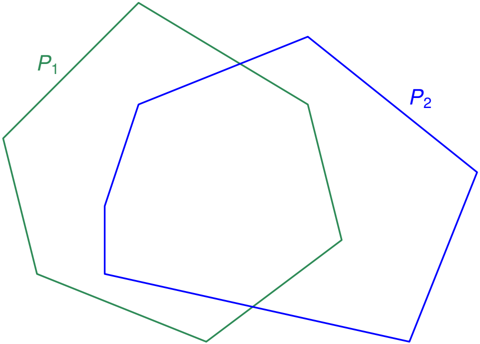
Definition 61 (Supporting line, Dt.: Stützgerade)

A *supporting line* of a convex polygon P is a straight line ℓ passing through a vertex of P such that the interior of P lies entirely to one side of ℓ.

- This definition is readily generalized to general convex sets.
- Two convex polygons P_1 and P_2, where no polygon is entirely contained within the other polygon, have up to four *common supporting lines*.

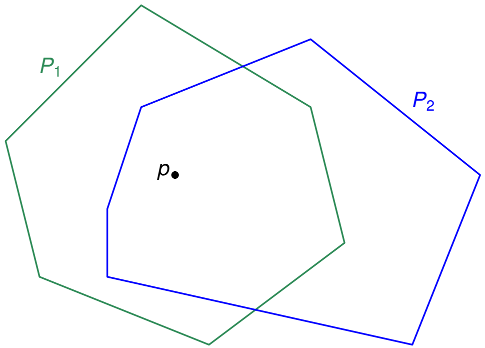


1. Find a point p that is internal to P_1; e.g., the centroid. Note that this point p will be internal to $CH(P_1 \cup P_2)$.



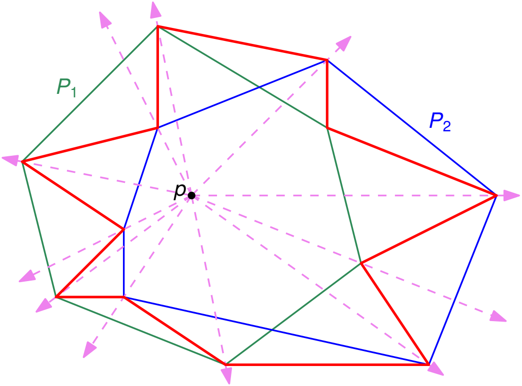
Divide-and-Conquer Convex Hull: Merge

1. Find a point p that is internal to P_1; e.g., the centroid. Note that this point p will be internal to $CH(P_1 \cup P_2)$.
2. Determine whether or not p is internal to P_2.



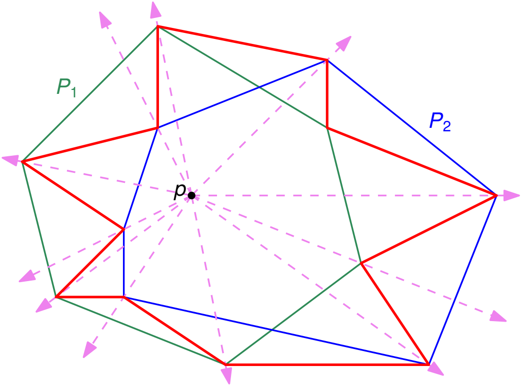
Divide-and-Conquer Convex Hull: Merge

1. Find a point p that is internal to P_1; e.g., the centroid. Note that this point p will be internal to $CH(P_1 \cup P_2)$.
2. Determine whether or not p is internal to P_2.
3. Case: Point p is internal to P_2:
   Merge P_1 and P_2 into one polygon that is star-shaped, with p within its kernel.

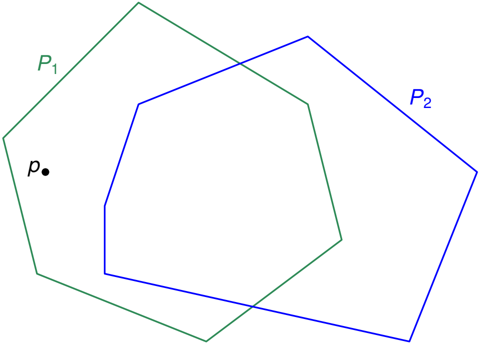


Divide-and-Conquer Convex Hull: Merge

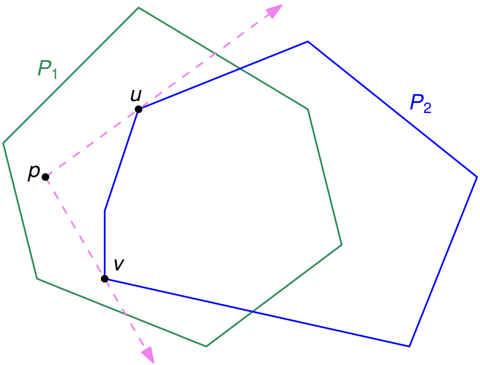
1. Find a point p that is internal to P_1; e.g., the centroid. Note that this point p will be internal to $CH(P_1 \cup P_2)$.
2. Determine whether or not p is internal to P_2.
3. Case: Point p is internal to P_2:
   Merge P_1 and P_2 into one polygon that is star-shaped, with p within its kernel.
4. Apply Graham's Scan to the resulting polygon.



3. Case: Point p is not internal to P_2:
   (a) Find vertices u and v on P_2 such that $\overline{pu}$ and $\overline{pv}$ are supporting lines of P_2.
   (b) Split P_2 into two chains at u and v.
   (c) Merge P_1 and one chain of P_2 into one polygon that is star-shaped, with p within its kernel.



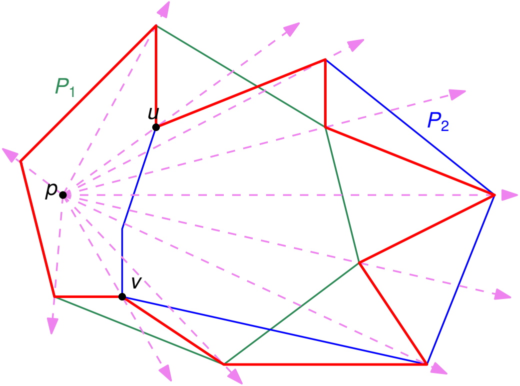
3. Case: Point p is not internal to P_2:
(a) Find vertices u and v on P_2 such that $\overline{pu}$ and $\overline{pv}$ are supporting lines of P_2.
(b) Split P_2 into two chains at u and v.
(c) Merge P_1 and one chain of P_2 into one polygon that is star-shaped, with p within its kernel.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

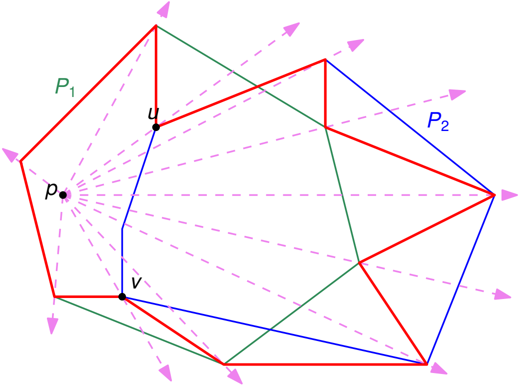
Divide-and-Conquer Convex Hull: Merge

3. Case: Point p is not internal to P_2:
   (a) Find vertices u and v on P_2 such that $\overline{pu}$ and $\overline{pv}$ are supporting lines of P_2.
   (b) Split P_2 into two chains at u and v.
   (c) Merge P_1 and one chain of P_2 into one polygon that is star-shaped, with p within its kernel.



Divide-and-Conquer Convex Hull: Merge

3. Case: Point p is not internal to P_2:
   (a) Find vertices u and v on P_2 such that $\overline{pu}$ and $\overline{pv}$ are supporting lines of P_2.
   (b) Split P_2 into two chains at u and v.
   (c) Merge P_1 and one chain of P_2 into one polygon that is star-shaped, with p within its kernel.
4. Apply Graham's Scan to the resulting polygon.



Divide-and-Conquer Convex Hull: Analysis

- If polygon P_1 has n_1 vertices and polygon P_2 has n_2 vertices, then the merge algorithm computes $CH(P_1 \cup P_2)$ in $O(n_1 + n_2)$ time.
- Obviously, an $O(n)$ merge yields an $O(n \log n)$ time bound for this divide-and-conquer algorithm.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- If polygon P_1 has n_1 vertices and polygon P_2 has n_2 vertices, then the merge algorithm computes $CH(P_1 \cup P_2)$ in $O(n_1 + n_2)$ time.
- Obviously, an $O(n)$ merge yields an $O(n \log n)$ time bound for this divide-and-conquer algorithm.

Theorem 62 (Complexity of divide&conquer convex hull)

The divide&conquer algorithm computes the convex hull of n points in the plane in $O(n \log n)$ time and within $O(n)$ storage.

4 Convex Hulls

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hull of a Simple Polygon

- Given is the sequence $(p_1, p_2, ..., p_n)$ of n points in $\mathbb{R}^2$ which form the vertices of a simple polygon P.
- Obviously, $CH(P)$ can be computed in $O(n \log n)$ time.
- Can we do any better?

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hull of a Simple Polygon

- Given is the sequence $(p_1, p_2, ..., p_n)$ of n points in $\mathbb{R}^2$ which form the vertices of a simple polygon P.
- Obviously, $CH(P)$ can be computed in $O(n \log n)$ time.
- Can we do any better? Note: The lower bound for computing the convex hull of n points does not carry over to this problem!

Convex Hull of a Simple Polygon

- Given is the sequence $(p_1, p_2, ..., p_n)$ of n points in $\mathbb{R}^2$ which form the vertices of a simple polygon P.
- Obviously, $CH(P)$ can be computed in $O(n \log n)$ time.
- Can we do any better? Note: The lower bound for computing the convex hull of n points does not carry over to this problem!
- Recall that Graham's Scan runs in linear time when applied to a star-shaped polygon.
- Thus, the fact that the points are vertices of a polygon can be expected to help when designing a linear-time algorithm.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hull of a Simple Polygon

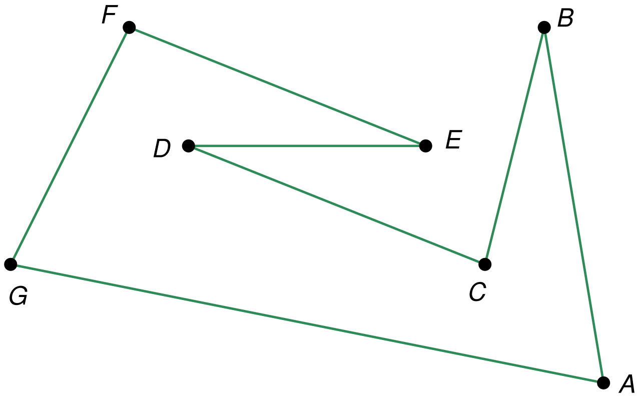
- Given is the sequence $(p_1, p_2, ..., p_n)$ of n points in $\mathbb{R}^2$ which form the vertices of a simple polygon P.
- Obviously, $CH(P)$ can be computed in $O(n \log n)$ time.
- Can we do any better? Note: The lower bound for computing the convex hull of n points does not carry over to this problem!
- Recall that Graham's Scan runs in linear time when applied to a star-shaped polygon.
- Thus, the fact that the points are vertices of a polygon can be expected to help when designing a linear-time algorithm.

Caveats

1. Several invalid linear-time "algorithms" were published in the early days of computational geometry.
2. Graham's Scan does not work properly for arbitrary simple polygons!

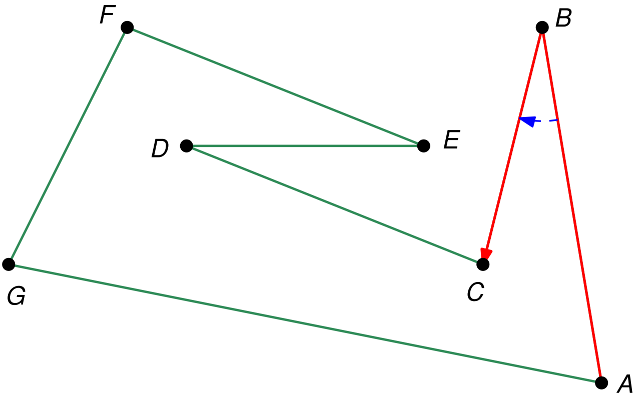
Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



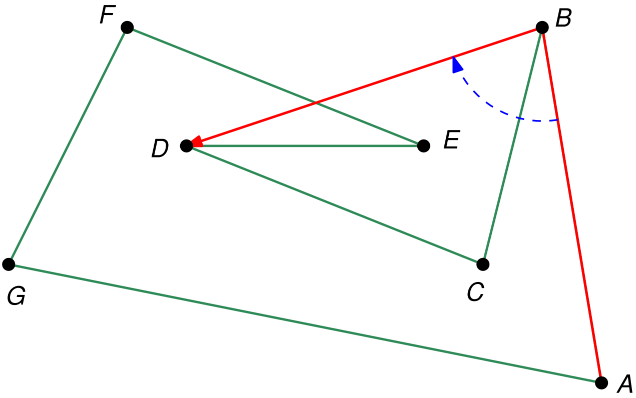
Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



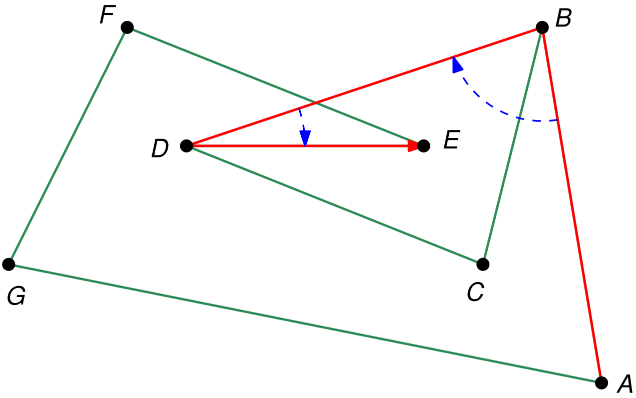
Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



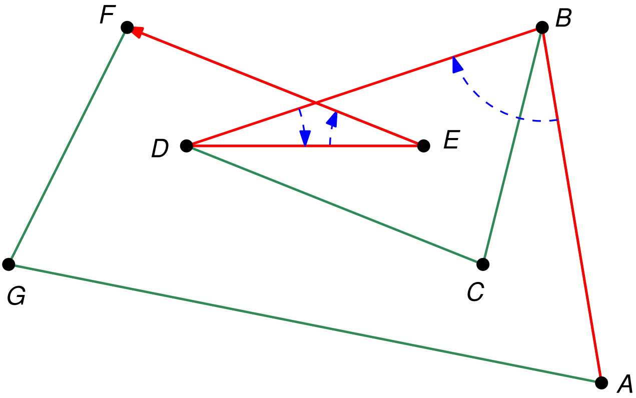
Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



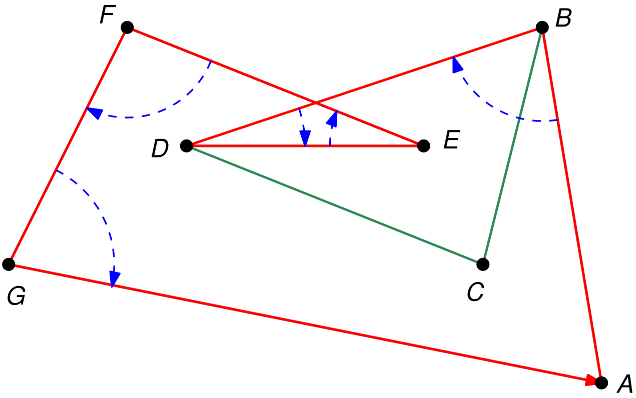
Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



Convex Hull of a Simple Polygon

- Graham's Scan does not work properly for arbitrary simple polygons!



Convex Hull of a Simple Polygon: Melkman's Algorithm

- Melkman's algorithm (1987) operates on a double-ended queue ("deque") $< d_b, \dots, d_t >$, with $d_b = d_t$; the d_i's will represent vertices of the convex hull.
- Deque operations:
  - Push(v) increments t by one, and inserts v at the new top;
  - Pop(d_t) deletes the top element and decrements t by one;
  - Insert(v) decrements b by one, and inserts v at the new bottom;
  - Delete(d_b) deletes the bottom element and increments b by one.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hull of a Simple Polygon: Melkman's Algorithm

- Melkman's algorithm (1987) operates on a double-ended queue ("deque") $< d_b, \ldots, d_t >$, with $d_b = d_t$; the d_i's will represent vertices of the convex hull.
- Deque operations:
  - Push(v) increments t by one, and inserts v at the new top;
  - Pop(d_t) deletes the top element and decrements t by one;
  - Insert(v) decrements b by one, and inserts v at the new bottom;
  - Delete(d_b) deletes the bottom element and increments b by one.
- Melkman's algorithm incrementally computes the convex hull of the polygon by adding one vertex at a time.
- A deque D is used to maintain the vertices of the convex hull constructed so far in CW order.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

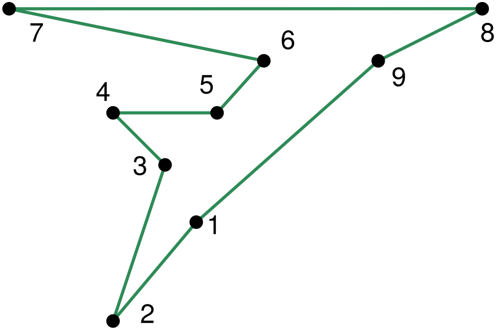
Convex Hull of a Simple Polygon: Melkman's Algorithm

- Melkman's algorithm (1987) operates on a double-ended queue ("deque") $< d_b, \ldots, d_t >$, with $d_b = d_t$; the d_i's will represent vertices of the convex hull.
- Deque operations:
  - Push(v) increments t by one, and inserts v at the new top;
  - Pop(d_t) deletes the top element and decrements t by one;
  - Insert(v) decrements b by one, and inserts v at the new bottom;
  - Delete(d_b) deletes the bottom element and increments b by one.
- Melkman's algorithm incrementally computes the convex hull of the polygon by adding one vertex at a time.
- A deque D is used to maintain the vertices of the convex hull constructed so far in CW order.
- The input polygon needs to be oriented CW.
- In the pseudo-code the vertices are retrieved online from "input", and an actual implementation needs to check for an end of the input data.

Algorithm *Melkman's Algorithm*

$t \leftarrow -1$; $b \leftarrow 0$; (∗ The current convex hull is maintained in the deque D ∗)
$v_1 \leftarrow$ input; $v_2 \leftarrow$ input; $v_3 \leftarrow$ input; (∗ Obtain vertices in CW order ∗)
if $\det(v_1, v_2, v_3) < 0$ **then** (∗ Initialize D ∗)
  Push(v_1); Push(v_2);
else
  Push(v_2); Push(v_1);
Push(v_3); Insert(v_3);
repeat
  repeat
    $v \leftarrow$ input;
  until $\det(d_b, d_{b+1}, v) > 0$ **or** $\det(d_{t-1}, d_t, v) > 0$ (∗ Skip v if interior to D ∗)
  while $\det(d_{t-1}, d_t, v) > 0$ **do**
    Pop(d_t); (∗ Delete interior vertices from top of D ∗)
  Push(v); (∗ Insert v at top of D ∗)
  while $\det(d_b, d_{b+1}, v) > 0$ **do**
    Delete(d_b); (∗ Delete interior vertices from bottom of D ∗)
  Insert(v);
until input is empty.

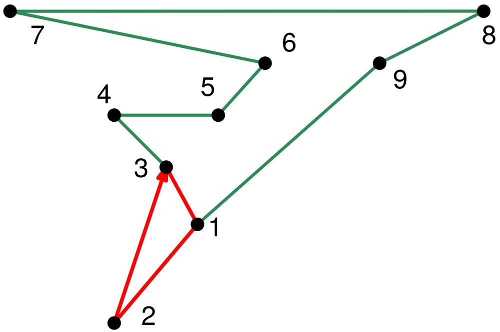
Animation of Melkman's Algorithm



b ← → t

-2	-1	0	1	2	3

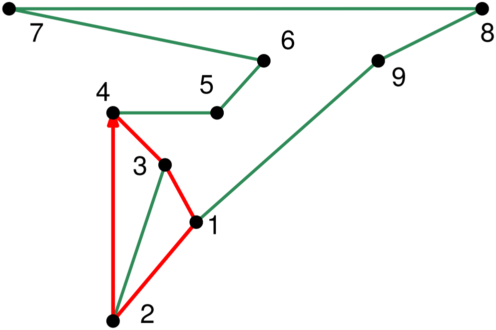
Animation of Melkman's Algorithm



← b t →

-2	-1	0	1	2	3
	3	1	2	3	

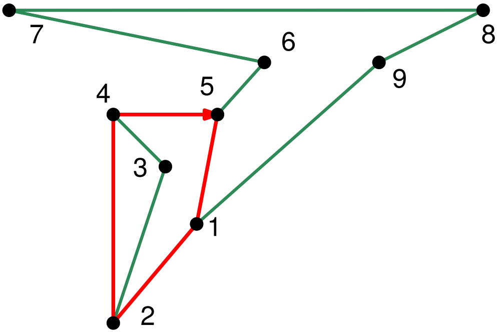
Animation of Melkman's Algorithm



← *b* *t* →

-2	-1	0	1	2	3
	3	1	2	3	
4	3	1	2	4	

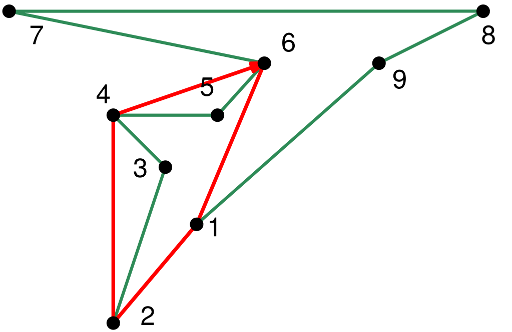
Animation of Melkman's Algorithm



b ← → *t*

-2	-1	0	1	2	3
	3	1	2	3	
4	3	1	2	4	
	5	1	2	4	5

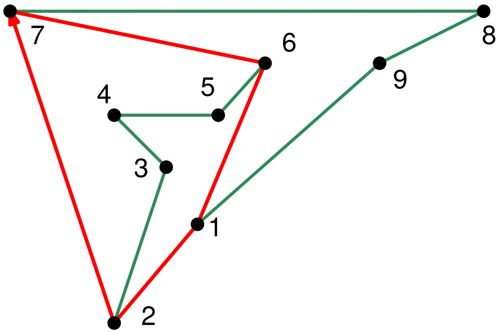
Animation of Melkman's Algorithm



b ← → *t*

-2	-1	0	1	2	3
	3	1	2	3	
4	3	1	2	4	
	5	1	2	4	5
	6	1	2	4	6

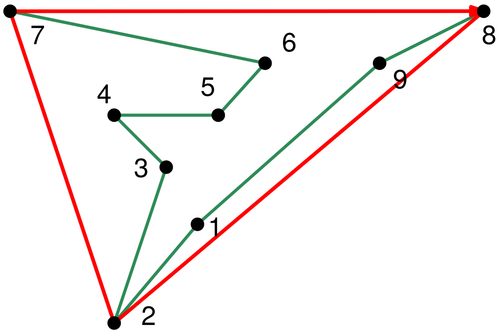
Animation of Melkman's Algorithm



$\xleftarrow{b}$ $\xrightarrow{t}$

-2	-1	0	1	2	3
	3	1	2	3	
4	3	1	2	4	
	5	1	2	4	5
	6	1	2	4	6
7	6	1	2	7	

Animation of Melkman's Algorithm



b ←   → *t*

-2	-1	0	1	2	3
	3	1	2	3	
4	3	1	2	4	
	5	1	2	4	5
	6	1	2	4	6
7	6	1	2	7	
		8	2	7	8

Theorem 63 (Melkman (1987))

Melkman's algorithm computes the convex hull of a simple n-vertex polygon in $O(n)$ time.

Convex Hull of a Simple Polygon: Analysis of Melkman's Algorithm

Theorem 63 (Melkman (1987))

Melkman's algorithm computes the convex hull of a simple n-vertex polygon in $O(n)$ time.

Proof : Similar to the analysis of Graham's Scan:

- Each vertex of the polygon is classified as either interior or exterior to the current hull in $O(1)$ time.
- If vertex v_i is exterior to the current hull then k_i other vertices may end up being deleted, with $O(1)$ time per each vertex that is deleted.
- Since $\sum_{i=1}^{n} k_i \leq n - 3$, the entire algorithm runs in $O(n)$ time.

$\square$

Theorem 63 (Melkman (1987))

Melkman's algorithm computes the convex hull of a simple n-vertex polygon in $O(n)$ time.

Proof : Similar to the analysis of Graham's Scan:

- Each vertex of the polygon is classified as either interior or exterior to the current hull in $O(1)$ time.
- If vertex v_i is exterior to the current hull then k_i other vertices may end up being deleted, with $O(1)$ time per each vertex that is deleted.
- Since $\sum_{i=1}^{n} k_i \leq n - 3$, the entire algorithm runs in $O(n)$ time.

$\square$

- The first correct linear-time convex-hull algorithm for polygons is due to McCallum&Avis (1979).

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

4 Convex Hulls

- Basics
- Algorithms
- Convex Hull of Polygons
- Convex Hulls in 3D
- Applications of Convex Hulls

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.
- Divide-and-conquer algorithm for computing convex hulls in 3D:
  1. Sort (and re-number) the points of S according to their x-coordinates.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.
- Divide-and-conquer algorithm for computing convex hulls in 3D:
  1. Sort (and re-number) the points of S according to their x-coordinates.
  2. Partition the set S into two subsets:
     $S_1 := \{p_1, p_2, \ldots, p_{\lfloor n/2 \rfloor}\}$, and
     $S_2 := \{p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n\}$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.
- Divide-and-conquer algorithm for computing convex hulls in 3D:
  1. Sort (and re-number) the points of S according to their x-coordinates.
  2. Partition the set S into two subsets:
     $S_1 := \{p_1, p_2, \ldots, p_{\lfloor n/2 \rfloor}\}$, and
     $S_2 := \{p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n\}$.
  3. Recursively find the convex hulls $P_1 := CH(S_1)$ and $P_2 := CH(S_2)$.

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.
- Divide-and-conquer algorithm for computing convex hulls in 3D:
  1. Sort (and re-number) the points of S according to their x-coordinates.
  2. Partition the set S into two subsets:
     $S_1 := \{p_1, p_2, \ldots, p_{\lfloor n/2 \rfloor}\}$, and
     $S_2 := \{p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n\}$.
  3. Recursively find the convex hulls $P_1 := CH(S_1)$ and $P_2 := CH(S_2)$.
  4. Merge P_1 and P_2 together to form $CH(S)$.

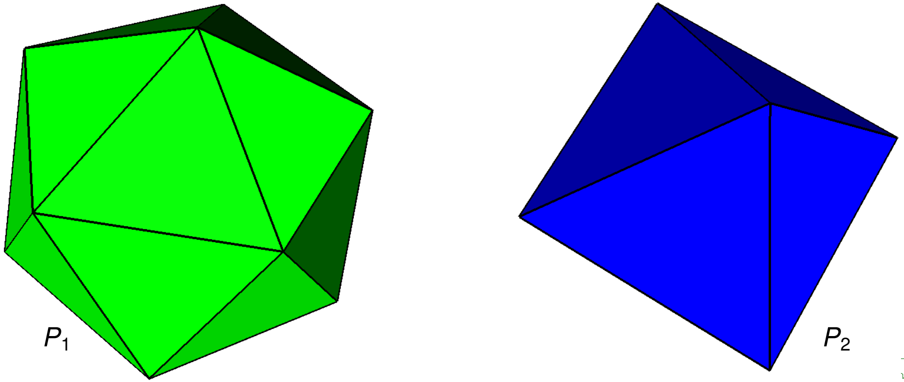
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Convex Hulls in 3D

- Consider a set S of n points $\{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^3$. We want to compute $CH(S)$.
- Easy to prove: $CH(S)$ is a convex polyhedron with at most $3n - 6$ edges.
- The $\Omega(n \log n)$ lower bound extends trivially from 2D to 3D.
- Divide-and-conquer algorithm for computing convex hulls in 3D:
  1. Sort (and re-number) the points of S according to their x-coordinates.
  2. Partition the set S into two subsets:
     $S_1 := \{p_1, p_2, \ldots, p_{\lfloor n/2 \rfloor}\}$, and
     $S_2 := \{p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n\}$.
  3. Recursively find the convex hulls $P_1 := CH(S_1)$ and $P_2 := CH(S_2)$.
  4. Merge P_1 and P_2 together to form $CH(S)$.
- In order to assist the merge, during all steps of the divide-and-conquer algorithm we maintain convex hulls of the point sets projected to 2D.

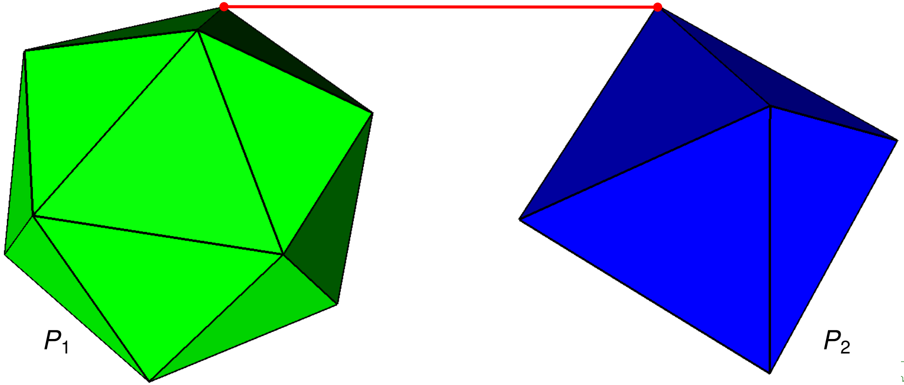
Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to "gift wrapping":



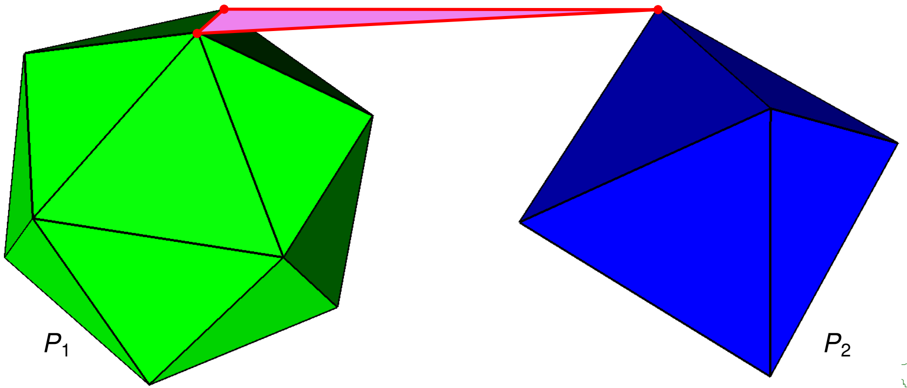
Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to "gift wrapping":
  1. Find a supporting edge e of the projections of P_1 and P_2 to 2D.



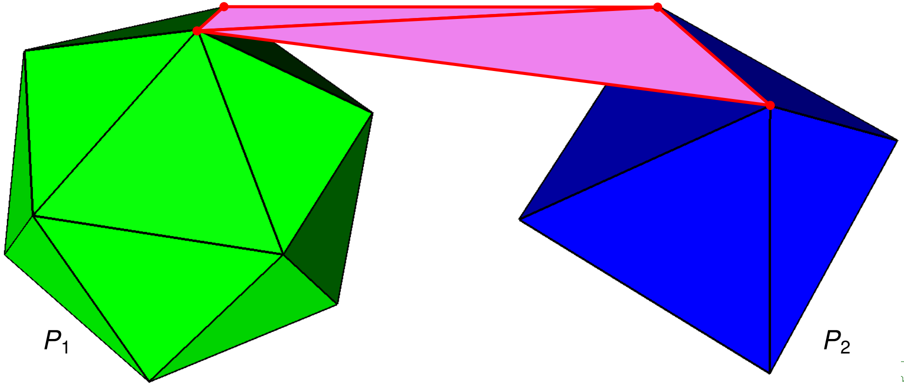
Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to "gift wrapping":
  1. Find a supporting edge e of the projections of P_1 and P_2 to 2D.
  2. Construct the first merge facet of the hull by "wrapping" a 2D plane through e around P_1 and P_2.

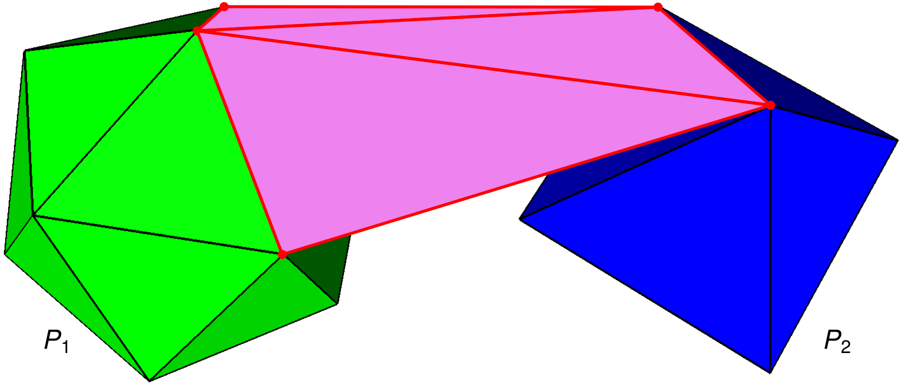


Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to "gift wrapping":
  1. Find a supporting edge *e* of the projections of P_1 and P_2 to 2D.
  2. Construct the first merge facet of the hull by "wrapping" a 2D plane through *e* around P_1 and P_2.
  3. Find a neighboring facet of the hull, again by "wrapping".

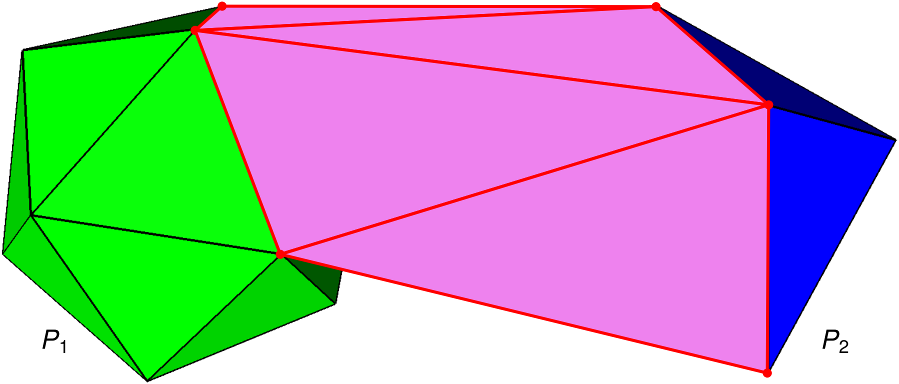


- The key idea of the merge step is similar to “gift wrapping”:
  1. Find a supporting edge e of the projections of P_1 and P_2 to 2D.
  2. Construct the first merge facet of the hull by "wrapping" a 2D plane through e around P_1 and P_2.
  3. Find a neighboring facet of the hull, again by “wrapping”.
  4. Continue from each facet to its neighbor until all merge facets are found.



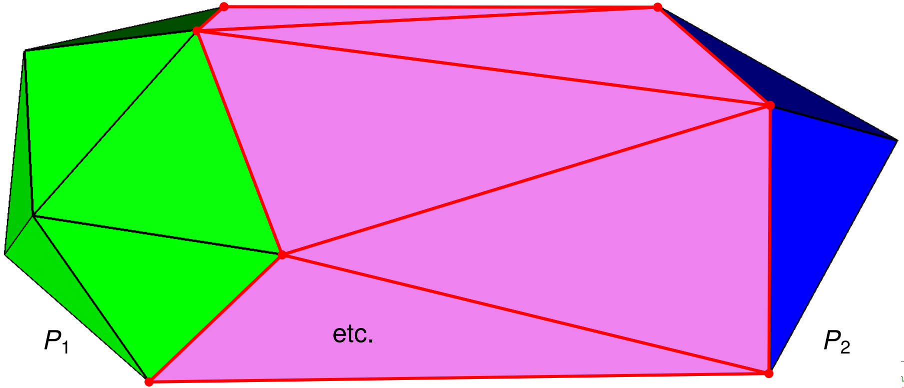
Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to “gift wrapping”:
  1. Find a supporting edge e of the projections of P_1 and P_2 to 2D.
  2. Construct the first merge facet of the hull by "wrapping" a 2D plane through e around P_1 and P_2.
  3. Find a neighboring facet of the hull, again by “wrapping”.
  4. Continue from each facet to its neighbor until all merge facets are found.



Convex Hulls in 3D: Divide-and-Conquer Algorithm

- The key idea of the merge step is similar to “gift wrapping”:
  1. Find a supporting edge e of the projections of P_1 and P_2 to 2D.
  2. Construct the first merge facet of the hull by "wrapping" a 2D plane through e around P_1 and P_2.
  3. Find a neighboring facet of the hull, again by “wrapping”.
  4. Continue from each facet to its neighbor until all merge facets are found.



Lemma 64

The merge step of the divide&conquer algorithm for computing the convex hull of n points in $\mathbb{R}^3$ can be carried out in $O(n)$ time.

Convex Hulls in 3D: Divide-and-Conquer Algorithm

Lemma 64

The merge step of the divide&conquer algorithm for computing the convex hull of n points in $\mathbb{R}^3$ can be carried out in $O(n)$ time.

Sketch of Proof : Each new facet runs through the last constructed edge e and through an endpoint of another edge e' either on P_1 or on P_2, where e and e' share a common endpoint. □

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 64

The merge step of the divide&conquer algorithm for computing the convex hull of n points in $\mathbb{R}^3$ can be carried out in $O(n)$ time.

Sketch of Proof : Each new facet runs through the last constructed edge e and through an endpoint of another edge e' either on P_1 or on P_2, where e and e' share a common endpoint. $\square$

Theorem 65

The full convex hull of n points in $\mathbb{R}^3$ can be computed in $O(n \log n)$ time.

Lemma 64

The merge step of the divide&conquer algorithm for computing the convex hull of n points in $\mathbb{R}^3$ can be carried out in $O(n)$ time.

Sketch of Proof : Each new facet runs through the last constructed edge e and through an endpoint of another edge e' either on P_1 or on P_2, where e and e' share a common endpoint. $\square$

Theorem 65

The full convex hull of n points in $\mathbb{R}^3$ can be computed in $O(n \log n)$ time.

Theorem 66 (Seidel (1984))

The computation of the convex hull of a star-shaped polyhedron in $\mathbb{R}^3$ with n vertices requires $\Omega(n \log n)$ time in the worst case.

Convex Hulls in Higher Dimensions

Theorem 67 (Seidel (1981))

The convex hull of n points in $\mathbb{R}^d$ can have $\Omega(n^{\lfloor d/2 \rfloor})$ facets.

Theorem 67 (Seidel (1981))

The convex hull of n points in $\mathbb{R}^d$ can have $\Omega(n^{\lfloor d/2 \rfloor})$ facets.

Theorem 68 (Chazelle (1993))

The convex hull of n points in $\mathbb{R}^d$ can be computed in $O(n \log n + n^{\lfloor d/2 \rfloor})$ time.

4 Convex Hulls

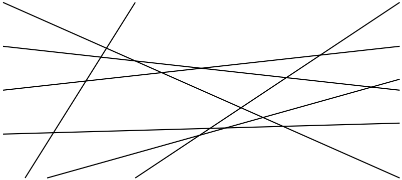
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Application of Convex Hulls: Lower Envelope

Definition 69 (Lower envelope)

Let L be a set of n lines with equations

$$y = k_1 x - d_1, \quad y = k_2 x - d_2, \quad \ldots, \quad y = k_n x - d_n.$$



Sample Application of Convex Hulls: Lower Envelope

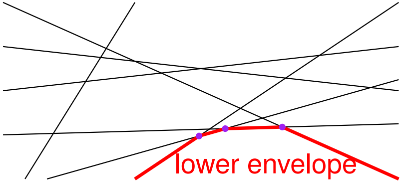
Definition 69 (Lower envelope)

Let L be a set of n lines with equations

$$y = k_1 x - d_1, \quad y = k_2 x - d_2, \quad \ldots, \quad y = k_n x - d_n.$$

Then the *lower envelope* $\mathcal{L}_L$ of L is the function $\mathcal{L}_L \colon \mathbb{R} \to \mathbb{R}$ with

$$\mathcal{L}_L(x) := \min_{1 \le i \le n} (k_i x - d_i).$$



Definition 69 (Lower envelope)

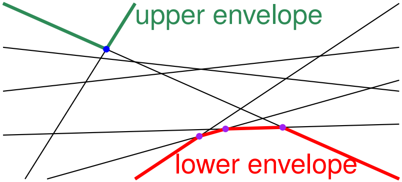
Let L be a set of n lines with equations

$$y = k_1 x - d_1, \quad y = k_2 x - d_2, \quad \ldots, \quad y = k_n x - d_n.$$

Then the *lower envelope* $\mathcal{L}_L$ of L is the function $\mathcal{L}_L \colon \mathbb{R} \to \mathbb{R}$ with

$$\mathcal{L}_L(x) := \min_{1 \leq i \leq n} (k_i x - d_i).$$

Similarly for the upper envelope $\mathcal{U}_L$.



Sample Application of Convex Hulls: Lower Envelope

Definition 69 (Lower envelope)

Let L be a set of n lines with equations

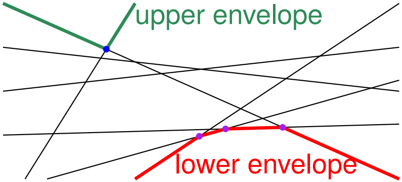
$$y = k_1 x - d_1, \quad y = k_2 x - d_2, \quad \ldots, \quad y = k_n x - d_n.$$

Then the *lower envelope* $\mathcal{L}_L$ of L is the function $\mathcal{L}_L : \mathbb{R} \to \mathbb{R}$ with

$$\mathcal{L}_L(x) := \min_{1 \le i \le n} (k_i x - d_i).$$

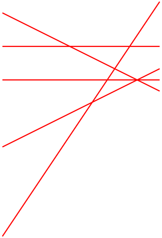
Similarly for the upper envelope $\mathcal{U}_L$.

- Note that a line of L may belong to both the lower and the upper envelope.



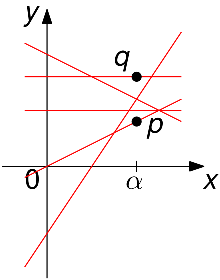
Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q.



Lemma 70

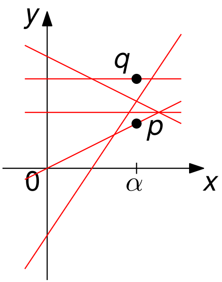
Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q.



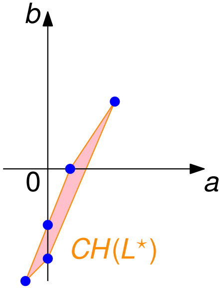
primal plane

Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q.



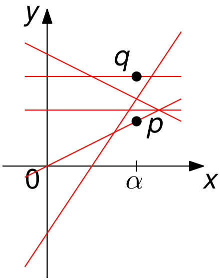
primal plane



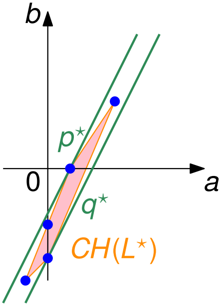
dual plane

Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q. Then the lines $p^\star$ and $q^\star$ support $CH(L^\star)$.



primal plane

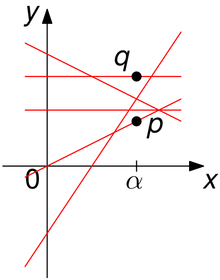


dual plane

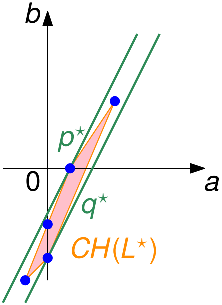
Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q. Then the lines $p^\star$ and $q^\star$ support $CH(L^\star)$.

Proof: By Cor. 40, all points of $L^\star$ are below or on the line $p^\star$. □



primal plane

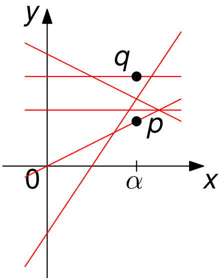


dual plane

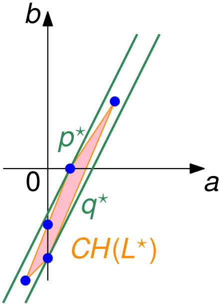
Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q. Then the lines $p^\star$ and $q^\star$ support $CH(L^\star)$.

Proof: By Cor. 40, all points of $L^\star$ are below or on the line $p^\star$. Furthermore, since p is on the lower envelope and, thus, on a line of L, the line $p^\star$ must pass through one of the points of $L^\star$. Hence, $p^\star$ supports $CH(L^\star)$ and lies above it. □



primal plane

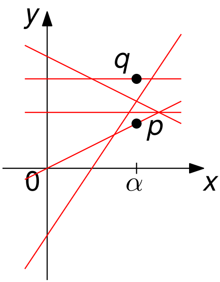


dual plane

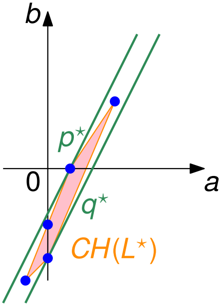
Lemma 70

Let L be a set of lines. For $\alpha \in \mathbb{R}$ arbitrary but fixed let $\beta^- := \mathcal{L}_L(\alpha)$ and $\beta^+ := \mathcal{U}_L(\alpha)$. Let (α, β^-) be the coordinates of the point p and (α, β^+) be the coordinates of the point q. Then the lines $p^\star$ and $q^\star$ support $CH(L^\star)$.

Proof: By Cor. 40, all points of $L^\star$ are below or on the line $p^\star$. Furthermore, since p is on the lower envelope and, thus, on a line of L, the line $p^\star$ must pass through one of the points of $L^\star$. Hence, $p^\star$ supports $CH(L^\star)$ and lies above it. Similarly for $q^\star$. □



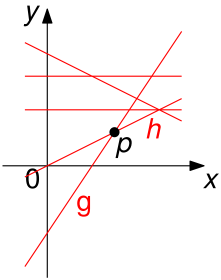
primal plane



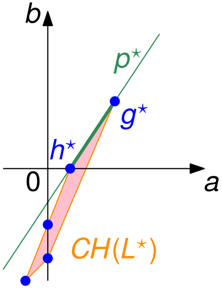
dual plane

Lemma 71

Let L be a set of lines. Then p is a vertex of the lower envelope of L if and only if $p^\star$ contains an edge on the (upper) convex hull $CH(L^\star)$.



primal plane

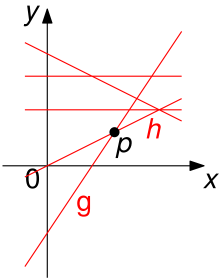


dual plane

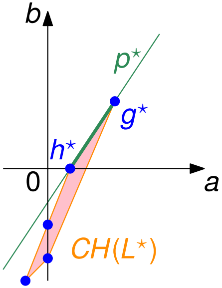
Lemma 71

Let L be a set of lines. Then p is a vertex of the lower envelope of L if and only if $p^\star$ contains an edge on the (upper) convex hull $CH(L^\star)$.

Proof: If p is a vertex of the lower envelope of L, then it is given by the intersection of two lines g and h. □



primal plane

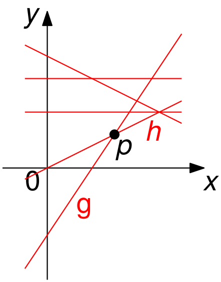


dual plane

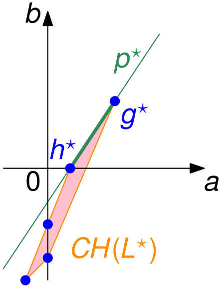
Lemma 71

Let L be a set of lines. Then p is a vertex of the lower envelope of L if and only if $p^\star$ contains an edge on the (upper) convex hull $CH(L^\star)$.

Proof: If p is a vertex of the lower envelope of L, then it is given by the intersection of two lines g and h. By Lem 70, all points of $L^\star$ lie below or on $p^\star$. Furthermore, $p^\star$ passes through $g^\star$ and $h^\star$. Hence, $p^\star$ contains an edge of $CH(L^\star)$. □



primal plane



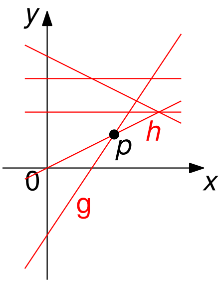
dual plane

Sample Application of Convex Hulls: Lower Envelope

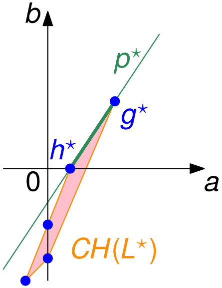
Lemma 71

Let L be a set of lines. Then p is a vertex of the lower envelope of L if and only if $p^\star$ contains an edge on the (upper) convex hull $CH(L^\star)$.

Proof : If p is a vertex of the lower envelope of L, then it is given by the intersection of two lines g and h. By Lem 70, all points of $L^\star$ lie below or on $p^\star$. Furthermore, $p^\star$ passes through $g^\star$ and $h^\star$. Hence, $p^\star$ contains an edge of $CH(L^\star)$. The other direction is argued similarly. □



primal plane



dual plane

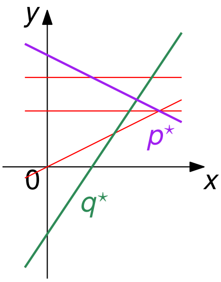
Theorem 72

The lower (or upper) envelope of a set L of n lines in the plane can be computed in $O(n \log n)$.

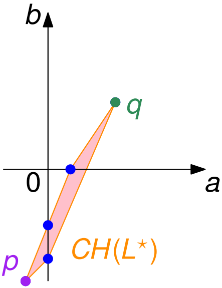
Theorem 72

The lower (or upper) envelope of a set L of n lines in the plane can be computed in $O(n \log n)$.

- The y-extreme points p, q of $CH(L^\star)$ correspond to the two lines which appear on both the upper and lower envelope of L and which contain the four infinite rays of these envelopes.



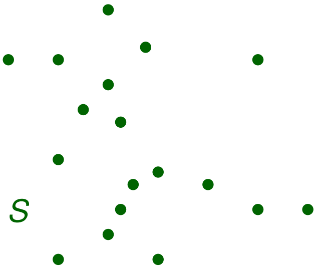
primal plane



dual plane

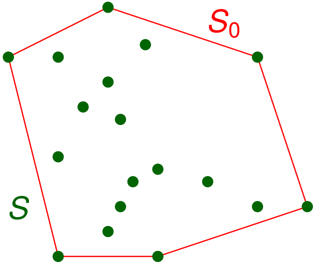
Sample Application of Convex Hulls: Onion Layers

- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.



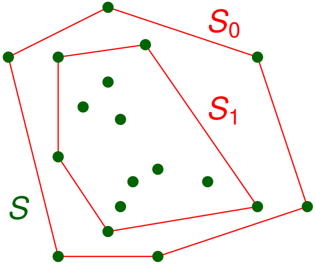
Sample Application of Convex Hulls: Onion Layers

- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.
- Let $S_0 \subseteq S$ be the set of all vertices of $CH(S)$.
- The points of S_0 are said to have *depth* 0.



Sample Application of Convex Hulls: Onion Layers

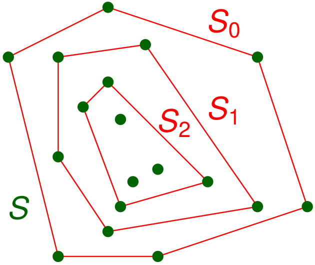
- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.
- Let $S_0 \subseteq S$ be the set of all vertices of $CH(S)$.
- The points of S_0 are said to have *depth* 0.



- Now let $S := S \setminus S_0$, and re-consider $CH(S)$.
- All points of S that are on $CH(S)$ are said to have *depth* 1, and are assigned to S_1.

Sample Application of Convex Hulls: Onion Layers

- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.
- Let $S_0 \subseteq S$ be the set of all vertices of $CH(S)$.
- The points of S_0 are said to have *depth* 0.

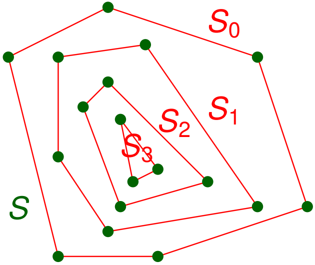


- Now let $S := S \setminus S_0$, and re-consider $CH(S)$.
- All points of S that are on $CH(S)$ are said to have *depth* 1, and are assigned to S_1.
- Similarly for depths $2, 3, \ldots, k$, where $S_k \neq \emptyset$ and $S_{k+1} = \emptyset$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Application of Convex Hulls: Onion Layers

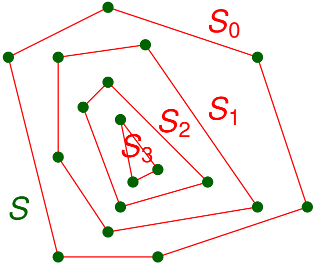
- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.
- Let $S_0 \subseteq S$ be the set of all vertices of $CH(S)$.
- The points of S_0 are said to have *depth* 0.



- Now let $S := S \setminus S_0$, and re-consider $CH(S)$.
- All points of S that are on $CH(S)$ are said to have *depth* 1, and are assigned to S_1.
- Similarly for depths $2, 3, \ldots, k$, where $S_k \neq \emptyset$ and $S_{k+1} = \emptyset$.

Sample Application of Convex Hulls: Onion Layers

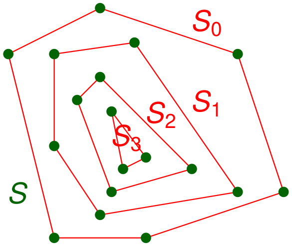
- Consider a set S of n points in $\mathbb{R}^2$, with general position assumed.
- Let $S_0 \subseteq S$ be the set of all vertices of $CH(S)$.
- The points of S_0 are said to have *depth* 0.



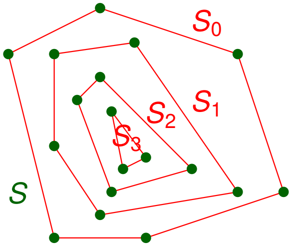
- Now let $S := S \setminus S_0$, and re-consider $CH(S)$.
- All points of S that are on $CH(S)$ are said to have *depth* 1, and are assigned to S_1.
- Similarly for depths $2, 3, \ldots, k$, where $S_k \neq \emptyset$ and $S_{k+1} = \emptyset$.
- The sets $S_0, S_1, S_2, \ldots$ are called *shells* or *onion layers* or *convex layers* of S.

Lemma 73

It takes $\Omega(n \log n)$ time to compute all depths of n points in $\mathbb{R}^2$.



Sample Application of Convex Hulls: Onion Layers

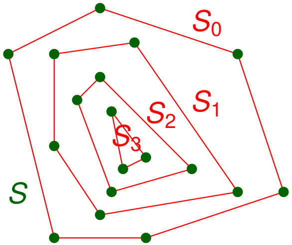


Lemma 73

It takes $\Omega(n \log n)$ time to compute all depths of n points in $\mathbb{R}^2$.

Theorem 74 (Chazelle (1985))

All depths of n points in $\mathbb{R}^2$, together with their onion layers, can be computed in time $O(n \log n)$.



Lemma 73

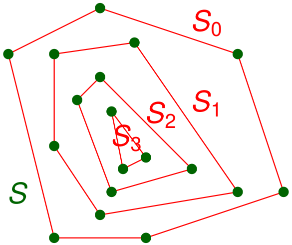
It takes $\Omega(n \log n)$ time to compute all depths of n points in $\mathbb{R}^2$.

Theorem 74 (Chazelle (1985))

All depths of n points in $\mathbb{R}^2$, together with their onion layers, can be computed in time $O(n \log n)$.

- Statistics: The points of $S_k, S_{k-1}, S_{k-2}, \ldots$ lie close to the "center" of S, and computing their mean tends to discard "outliers", thus yielding a more robust statistical estimator of the mean of S than the mean of all point samples.

Sample Application of Convex Hulls: Onion Layers



Lemma 73

It takes $\Omega(n \log n)$ time to compute all depths of n points in $\mathbb{R}^2$.

Theorem 74 (Chazelle (1985))

All depths of n points in $\mathbb{R}^2$, together with their onion layers, can be computed in time $O(n \log n)$.

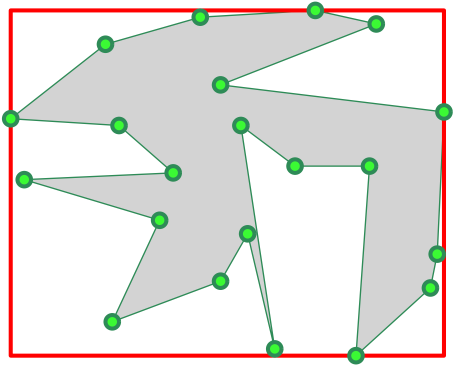
- Statistics: The points of $S_k, S_{k-1}, S_{k-2}, \ldots$ lie close to the "center" of S, and computing their mean tends to discard "outliers", thus yielding a more robust statistical estimator of the mean of S than the mean of all point samples.
- Rendering: Onion layers can be used to generate Hamiltonian triangulations.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Application of Convex Hulls: Kinetic AABB

Definition 75 (AABB)

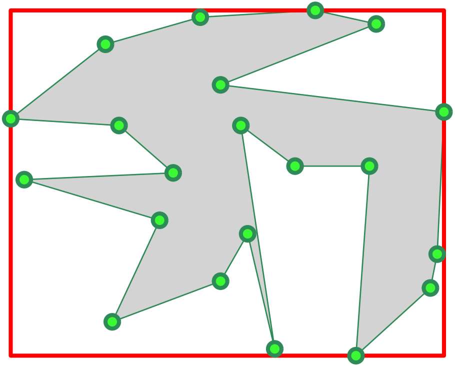
The *(axis-aligned) bounding box* (AABB) of a set $S \subset \mathbb{R}^d$, denoted by $AABB(S)$, is the smallest box (with sides parallel to the coordinate planes) which contains S.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 75 (AABB)

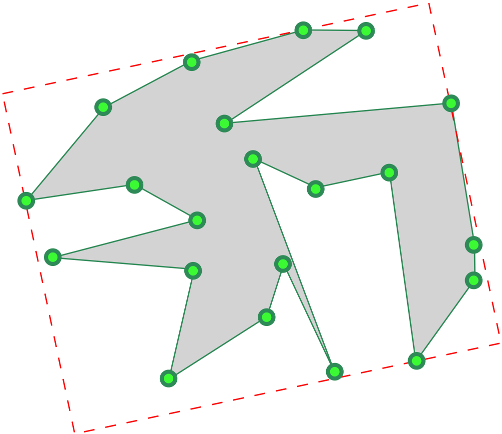
The *(axis-aligned) bounding box* (AABB) of a set $S \subset \mathbb{R}^d$, denoted by $AABB(S)$, is the smallest box (with sides parallel to the coordinate planes) which contains S.



- If S can be described by a set of n vertices then $AABB(S)$ can be computed in $O(d \cdot n)$ time in a straightforward manner.

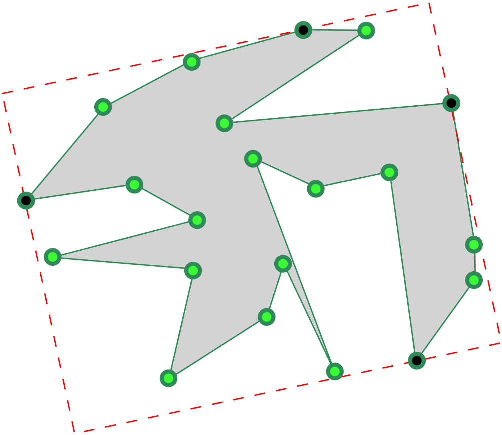
Sample Application of Convex Hulls: Kinetic AABB

- What happens if S moves?

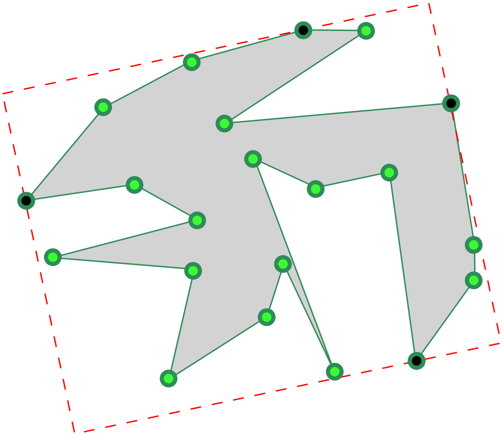


Sample Application of Convex Hulls: Kinetic AABB

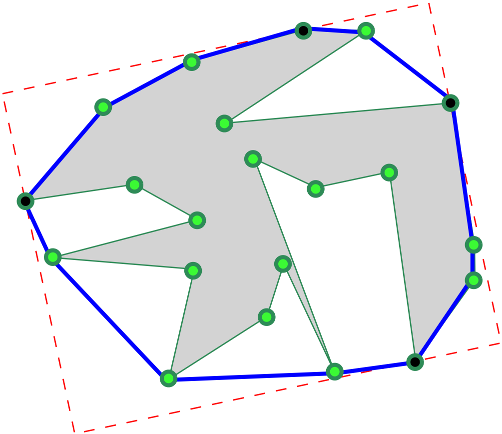
- What happens if S moves? We observe that $AABB(S)$ equals $AABB(CH(S))$: up to six vertices $v_1, v_2, \ldots, v_6$ of $CH(S)$ determine $AABB(S)$ in $\mathbb{R}^3$.



- What happens if S moves? We observe that $AABB(S)$ equals $AABB(CH(S))$: up to six vertices $v_1, v_2, \ldots, v_6$ of $CH(S)$ determine $AABB(S)$ in $\mathbb{R}^3$.
- Goal: Avoid re-scanning all vertices of S in order to re-compute the axis-aligned bounding box from scratch.

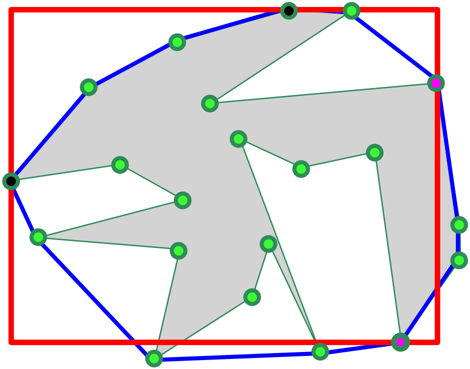


- What happens if S moves? We observe that $AABB(S)$ equals $AABB(CH(S))$: up to six vertices $v_1, v_2, \ldots, v_6$ of $CH(S)$ determine $AABB(S)$ in $\mathbb{R}^3$.
- We can exploit coherence by applying a hill-climbing algorithm, starting at each of these six vertices (resp. four vertices in 2D).



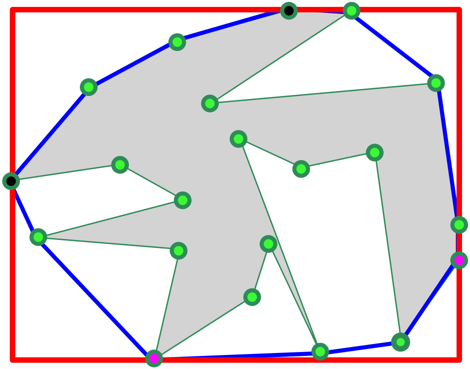
Sample Application of Convex Hulls: Kinetic AABB

- What happens if S moves? We observe that $AABB(S)$ equals $AABB(CH(S))$: up to six vertices $v_1, v_2, \ldots, v_6$ of $CH(S)$ determine $AABB(S)$ in $\mathbb{R}^3$.
- Hill-climbing means to move from one vertex to a neighboring vertex of $CH(S)$ if it has a smaller/larger x-coordinate, y-coordinate, ...



Sample Application of Convex Hulls: Kinetic AABB

- What happens if S moves? We observe that $AABB(S)$ equals $AABB(CH(S))$: up to six vertices $v_1, v_2, \ldots, v_6$ of $CH(S)$ determine $AABB(S)$ in $\mathbb{R}^3$.
- If S has moved only a little then few steps of the hill-climbing algorithm will suffice. Of course, this scheme can be extended to k-dops.



5 Voronoi Diagrams of Points

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

5 **Voronoi Diagrams of Points**

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

A Set of Proximity Problems

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

ALLNEARESTNEIGHBORS: Determine the "nearest neighbor" (point of minimum distance within S) for each point in S.

A Set of Proximity Problems

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

ALLNEARESTNEIGHBORS: Determine the "nearest neighbor" (point of minimum distance within S) for each point in S.

EUCLIDEANMINIMUMSPANNINGTREE (EMST): Construct a tree of minimum total (Euclidean) length whose vertices are the points of S. (No Steiner points allowed.)

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

A Set of Proximity Problems

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

ALLNEARESTNEIGHBORS: Determine the "nearest neighbor" (point of minimum distance within S) for each point in S.

EUCLIDEANMINIMUMSPANNINGTREE (EMST): Construct a tree of minimum total (Euclidean) length whose vertices are the points of S. (No Steiner points allowed.)

MAXIMUMEMPTYCIRCLE: Find a circle with largest radius which does not contain a point of S in its interior and whose center lies within $CH(S)$.

A Set of Proximity Problems

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

ALLNEARESTNEIGHBORS: Determine the "nearest neighbor" (point of minimum distance within S) for each point in S.

EUCLIDEANMINIMUMSPANNINGTREE (EMST): Construct a tree of minimum total (Euclidean) length whose vertices are the points of S. (No Steiner points allowed.)

MAXIMUMEMPTYCIRCLE: Find a circle with largest radius which does not contain a point of S in its interior and whose center lies within $CH(S)$.

TRIANGULATION: Join the points in S by non-intersecting straight-line segments so that every region internal to the convex hull of S is a triangle.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

A Set of Proximity Problems

Problem: Proximity

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

ClosestPair: Determine two points of S whose mutual distance is smallest.

AllNearestNeighbors: Determine the "nearest neighbor" (point of minimum distance within S) for each point in S.

EuclideanMinimumSpanningTree (EMST): Construct a tree of minimum total (Euclidean) length whose vertices are the points of S. (No Steiner points allowed.)

MaximumEmptyCircle: Find a circle with largest radius which does not contain a point of S in its interior and whose center lies within $CH(S)$.

Triangulation: Join the points in S by non-intersecting straight-line segments so that every region internal to the convex hull of S is a triangle.

NearestNeighborSearch: Given a query point q, determine which point $p \in S$ is closest to q.

A Set of Proximity Problems

Problem: PROXIMITY

Given: A set $S := \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^2$ under the Euclidean metric.

CLOSESTPAIR: Determine two points of S whose mutual distance is smallest.

ALLNEARESTNEIGHBORS: Determine the “nearest neighbor” (point of minimum distance within S) for each point in S.

EUCLIDEANMINIMUMSPANNINGTREE (EMST): Construct a tree of minimum total (Euclidean) length whose vertices are the points of S. (No Steiner points allowed.)

MAXIMUMEMPTYCIRCLE: Find a circle with largest radius which does not contain a point of S in its interior and whose center lies within $CH(S)$.

TRIANGULATION: Join the points in S by non-intersecting straight-line segments so that every region internal to the convex hull of S is a triangle.

NEARESTNEIGHBORSEARCH: Given a query point q, determine which point $p \in S$ is closest to q.

- Unless stated explicitly otherwise, we will always deal with the Euclidean metric.

Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

- NearestNeighborSearch: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.

Theorem 76

NEARESTNEIGHBORSEARCH among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; CLOSESTPAIR, ALLNEARESTNEIGHBORS, EMST, MAXIMUMEMPTYCIRCLE and TRIANGULATION all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

- NEARESTNEIGHBORSEARCH: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.
- CLOSESTPAIR: ELEMENTUNIQUENESS, which requires $\Omega(n \log n)$ time (in the ACT model of computation), is linearly reducible to CLOSESTPAIR.

Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

- NearestNeighborSearch: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.
- ClosestPair: ElementUniqueness, which requires $\Omega(n \log n)$ time (in the ACT model of computation), is linearly reducible to ClosestPair.
- AllNearestNeighbors: ClosestPair is linearly reducible to AllNearestNeighbors.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

- NearestNeighborSearch: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.
- ClosestPair: ElementUniqueness, which requires $\Omega(n \log n)$ time (in the ACT model of computation), is linearly reducible to ClosestPair.
- AllNearestNeighbors: ClosestPair is linearly reducible to AllNearestNeighbors.
- EMST: ClosestPair is linearly reducible to EMST. (Also, Sorting can be reduced linearly to EMST.)

Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

- NearestNeighborSearch: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.
- ClosestPair: ElementUniqueness, which requires $\Omega(n \log n)$ time (in the ACT model of computation), is linearly reducible to ClosestPair.
- AllNearestNeighbors: ClosestPair is linearly reducible to AllNearestNeighbors.
- EMST: ClosestPair is linearly reducible to EMST. (Also, Sorting can be reduced linearly to EMST.)
- MaximumEmptyCircle in 1D solves MaxGap, which establishes the $\Omega(n \log n)$ lower bound.

Lower Bounds

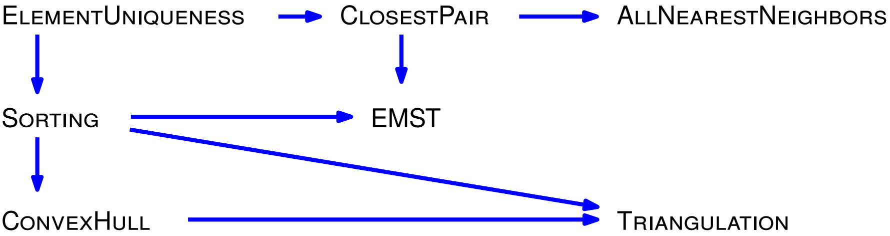
Theorem 76

NearestNeighborSearch among n points in $\mathbb{R}^2$ has an $\Omega(\log n)$ lower bound; ClosestPair, AllNearestNeighbors, EMST, MaximumEmptyCircle and Triangulation all have $\Omega(n \log n)$ lower bounds (in the ACT model of computation).

Sketch of Proof :

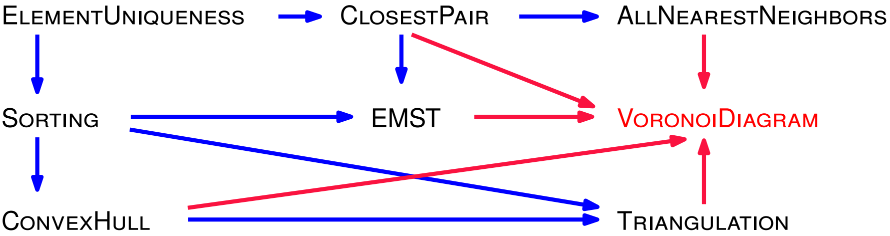
- NearestNeighborSearch: standard information-theoretic arguments yield $\Omega(\log n)$ comparisons.
- ClosestPair: ElementUniqueness, which requires $\Omega(n \log n)$ time (in the ACT model of computation), is linearly reducible to ClosestPair.
- AllNearestNeighbors: ClosestPair is linearly reducible to AllNearestNeighbors.
- EMST: ClosestPair is linearly reducible to EMST. (Also, Sorting can be reduced linearly to EMST.)
- MaximumEmptyCircle in 1D solves MaxGap, which establishes the $\Omega(n \log n)$ lower bound.
- Triangulation: ConvexHull is linearly reducible to Triangulation.

- Thus, we have $\Omega(n \log n)$ lower bounds due to a variety of reductions.



Lower Bounds: Summary of Reductions

- Thus, we have $\Omega(n \log n)$ lower bounds due to a variety of reductions.
- If Voronoi diagram is available then these problems can be solved in $O(n)$ time!

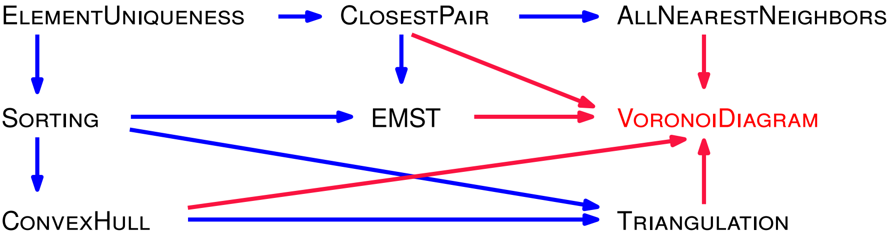


Lower Bounds: Summary of Reductions

- Thus, we have $\Omega(n \log n)$ lower bounds due to a variety of reductions.
- If Voronoi diagram is available then these problems can be solved in $O(n)$ time!

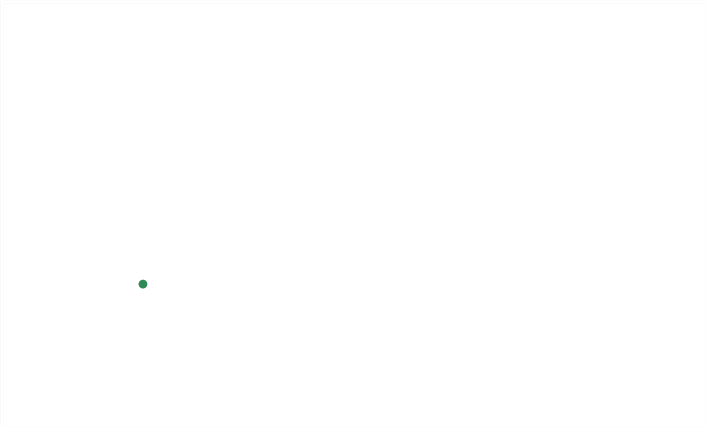
Theorem 77

The computation of the Voronoi diagram of n points in $\mathbb{R}^2$ requires $\Omega(n \log n)$ time.



Prairie fire

Let's ignite a fire in a grassland, and watch it spread out. In an idealized setting — uniform grassland, no wind — the fire wavefronts will form concentric circles!



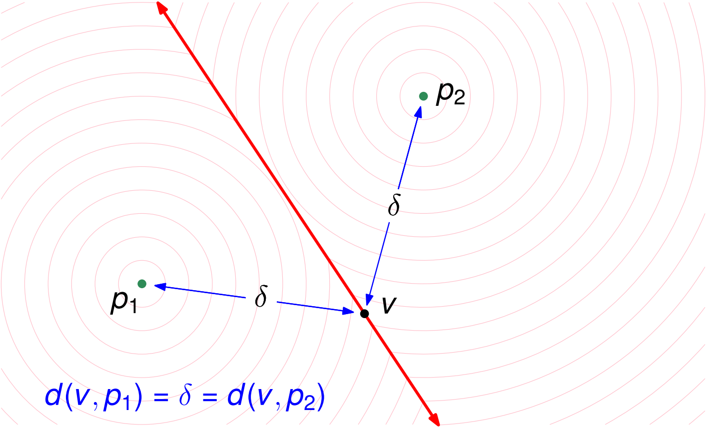
Voronoi Diagram of Points

Prairie fire

Now ignite two fires simultaneously: As the fire wavefronts meet — which propagate at the same speeds! — the bisector line between the two fire sites is traced out.

Prairie fire

Now ignite two fires simultaneously: As the fire wavefronts meet — which propagate at the same speeds! — the bisector line between the two fire sites is traced out.



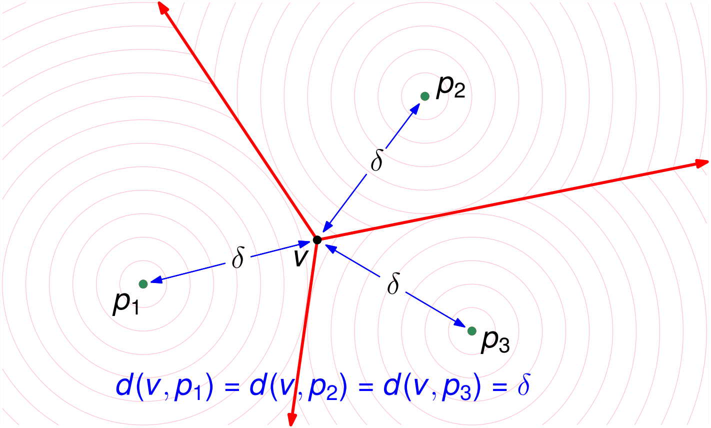
Prairie fire

We repeat the experiment with three fires ignited simultaneously: Again, the fire wavefronts trace out the bisectors between the fire sites as they meet.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

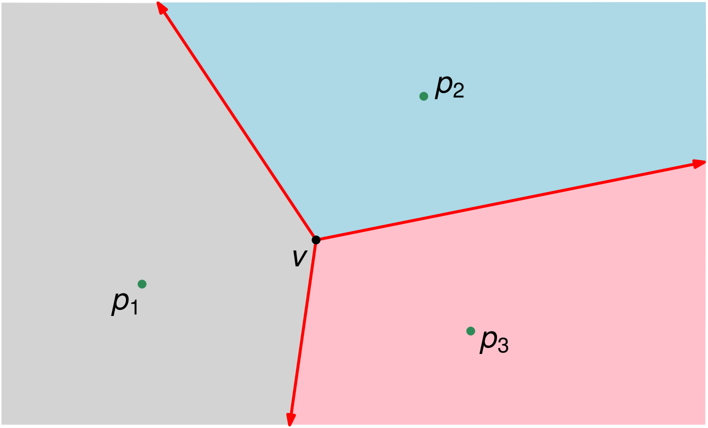
Prairie fire

We repeat the experiment with three fires ignited simultaneously: Again, the fire wavefronts trace out the bisectors between the fire sites as they meet.



Voronoi regions

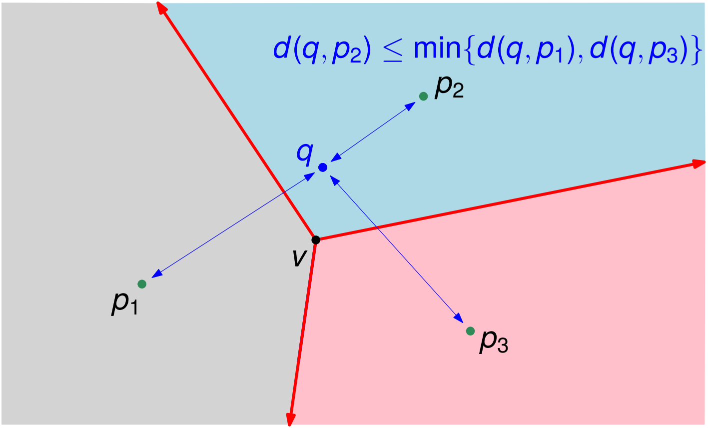
The red bisectors defined by the three fires partition the plane into Voronoi regions:



Voronoi Diagram: Motivation

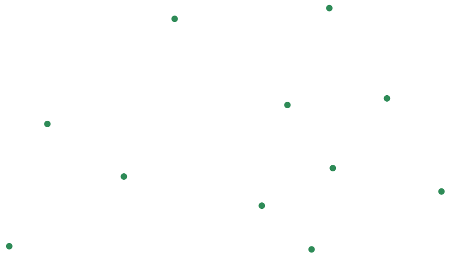
Voronoi regions

The red bisectors defined by the three fires partition the plane into Voronoi regions: Each region is the loci of points q closer to its defining fire site than to any other fire.



Voronoi Diagram: Definition

- Consider a set $S := \{p_1, p_2, \cdots, p_n\}$ of n distinct points in $\mathbb{R}^2$ and denote the Euclidean distance by $d(\cdot, \cdot)$, with $d(q, S) := \min\{d(q, p) : p \in S\}$.



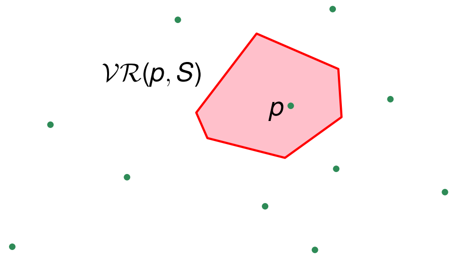
Voronoi Diagram: Definition

- Consider a set $S := \{p_1, p_2, \cdots, p_n\}$ of n distinct points in $\mathbb{R}^2$ and denote the Euclidean distance by $d(\cdot,\cdot)$, with $d(q,S) := \min\{d(q,p) : p \in S\}$.

Definition 78 (Voronoi region, Dt.: Voronoi-Zelle)

The *Voronoi region* (VR, aka "Voronoi cell") of a point $p \in S$ is the locus of points of $\mathbb{R}^2$ whose distance to p is not greater than the distance to any other point of S:

$$\mathcal{VR}(p,S) := \{q \in \mathbb{R}^2 : \ d(q,p) \le d(q,S)\}.$$



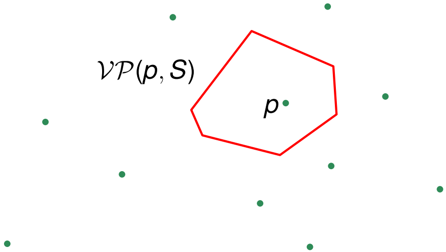
Voronoi Diagram: Definition

Definition 79 (Voronoi polygon)

The *Voronoi polygon* (VP) of $p \in S$ is defined as

$$\mathcal{VP}(p, S) := \partial\,\mathcal{VR}(p, S).$$

The segments of a Voronoi polygon are called *Voronoi edges*.

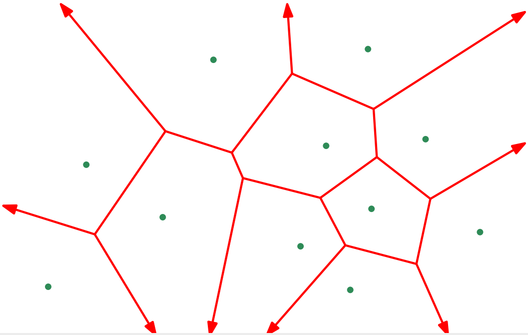


Voronoi Diagram: Definition

Definition 80 (Voronoi diagram)

The *Voronoi diagram* (VD) of S is defined as

$$\mathcal{VD}(S) := \bigcup_{1 \le i \le n} \mathcal{VP}(p_i, S).$$

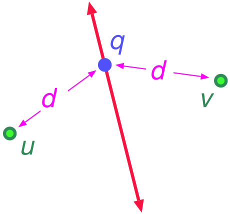


Voronoi Diagram: Definition

Definition 81 (Bisector)

The *bisector* of two points $u, v \in \mathbb{R}^2$ is the set of points of $\mathbb{R}^2$ which are equidistant to u and v:

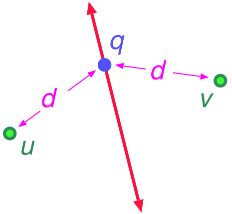
$$b(u, v) := \{q \in \mathbb{R}^2 : \ d(u, q) = d(v, q)\}.$$



Definition 81 (Bisector)

The *bisector* of two points $u, v \in \mathbb{R}^2$ is the set of points of $\mathbb{R}^2$ which are equidistant to u and v:

$$b(u, v) := \{q \in \mathbb{R}^2 : d(u, q) = d(v, q)\}.$$



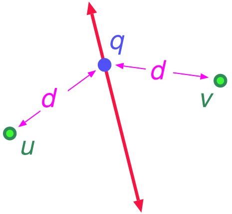
- A Voronoi edge always lies on a bisector. Thus, points on a Voronoi edge are equidistant to two points of S.

Voronoi Diagram: Definition

Definition 81 (Bisector)

The *bisector* of two points $u, v \in \mathbb{R}^2$ is the set of points of $\mathbb{R}^2$ which are equidistant to u and v:

$$b(u, v) := \{q \in \mathbb{R}^2 : \; d(u, q) = d(v, q)\}.$$



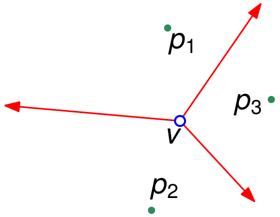
- A Voronoi edge always lies on a bisector. Thus, points on a Voronoi edge are equidistant to two points of S.

Lemma 82

For $p \in S$ we get $\mathcal{VP}(p, S) = \{q \in \mathbb{R}^2 : d(q, p) = d(q, S \setminus \{p\})\}$.

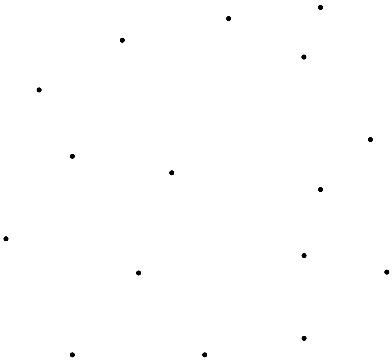
Definition 83 (Voronoi node, Dt.: Voronoi-Knoten)

Intersections of Voronoi edges are called *Voronoi nodes*.



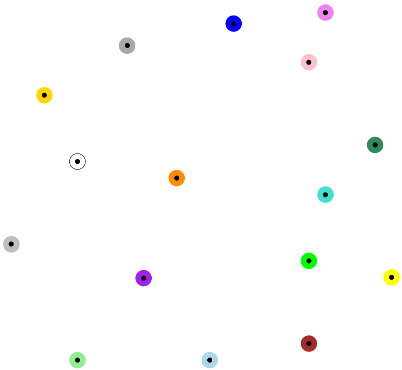
Sample Voronoi Diagram

- Input set S of points.



Sample Voronoi Diagram

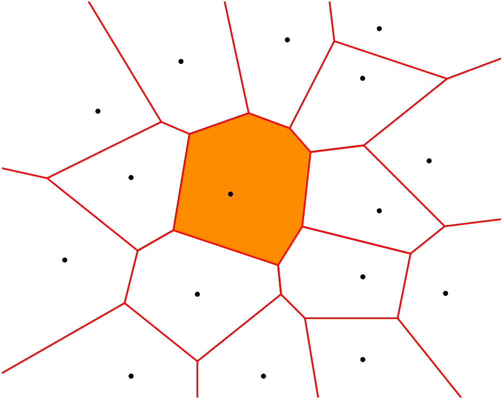
- Input set S of points, wavefronts.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

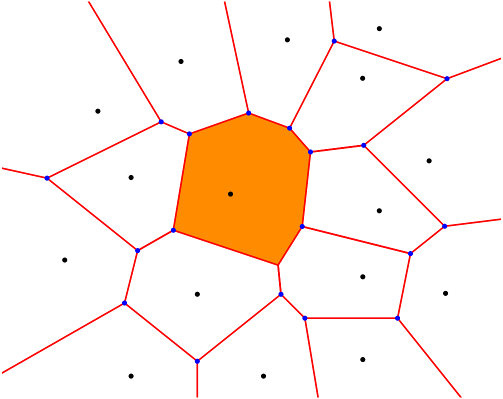
Sample Voronoi Diagram

- Input set S of points, Voronoi diagram and one Voronoi region.



Sample Voronoi Diagram

- Input set S of points, Voronoi diagram and one Voronoi region, Voronoi nodes.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Historical Remarks

- René Descartes (1596–1650) drew Voronoi-like diagrams to illustrate the subdivision of space by celestial bodies [Descartes 1644].
- Gustav Lejeune Dirichlet (1805–1859) provided the first formal definition of Voronoi diagrams in two dimensions [Dirichlet 1850].

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Historical Remarks

- René Descartes (1596–1650) drew Voronoi-like diagrams to illustrate the subdivision of space by celestial bodies [Descartes 1644].
- Gustav Lejeune Dirichlet (1805–1859) provided the first formal definition of Voronoi diagrams in two dimensions [Dirichlet 1850].
- Georgy Feodosevich Voronoi (1868–1908) generalized them to *n* dimensions [Voronoi 1908].
  - Several other Latin spellings of his name: Voronoï, Voronoy, Woronoi.
  - Born at Zhuravky (near Kyiv).
  - Studied at Saint Petersburg University as a student of Andrey Markov.
  - Professor at the University of Warsaw.
  - Students (among others): Boris Delaunay (Kyiv) and Wacław Sierpiński (Warsaw).

Lemma 84

The Voronoi region $\mathcal{VR}(p_i, S)$ is the intersection of half-planes defined by bisectors between $p_i \in S$ and the other points of S:

$$\mathcal{VR}(p_i, S) = \bigcap_{\substack{1 \le j \le n \\ j \ne i}} H(p_i, p_j),$$

where $H(p_i, p_j)$ is the half-plane that contains p_i and is bounded by $b(p_i, p_j)$.

Lemma 84

The Voronoi region $\mathcal{VR}(p_i, S)$ is the intersection of half-planes defined by bisectors between $p_i \in S$ and the other points of S:

$$\mathcal{VR}(p_i, S) = \bigcap_{\substack{1 \le j \le n \\ j \neq i}} H(p_i, p_j),$$

where $H(p_i, p_j)$ is the half-plane that contains p_i and is bounded by $b(p_i, p_j)$.

Corollary 85

Every Voronoi region is a convex polygonal area.

Lemma 84

The Voronoi region $\mathcal{VR}(p_i, S)$ is the intersection of half-planes defined by bisectors between $p_i \in S$ and the other points of S:

$$\mathcal{VR}(p_i, S) = \bigcap_{\substack{1 \le j \le n \\ j \ne i}} H(p_i, p_j),$$

where $H(p_i, p_j)$ is the half-plane that contains p_i and is bounded by $b(p_i, p_j)$.

Corollary 85

Every Voronoi region is a convex polygonal area.

Lemma 86

Every point of S has its own Voronoi region that is not empty.

Voronoi Diagram: Properties

Lemma 84

The Voronoi region $\mathcal{VR}(p_i, S)$ is the intersection of half-planes defined by bisectors between $p_i \in S$ and the other points of S:

$$\mathcal{VR}(p_i, S) = \bigcap_{\substack{1 \le j \le n \\ j \ne i}} H(p_i, p_j),$$

where $H(p_i, p_j)$ is the half-plane that contains p_i and is bounded by $b(p_i, p_j)$.

Corollary 85

Every Voronoi region is a convex polygonal area.

Lemma 86

Every point of S has its own Voronoi region that is not empty.

Lemma 87

The (topological) interiors of Voronoi regions of distinct points of S are disjoint.

Voronoi Diagram: Properties

General position assumed (GPA)

No four points of S are co-circular!

Voronoi Diagram: Properties

General position assumed (GPA)

No four points of S are co-circular!

Lemma 88

A Voronoi node is the common intersection of exactly three Voronoi edges. It is equidistant to the three points of S which lie in the Voronoi regions it belongs to.

Voronoi Diagram: Properties

General position assumed (GPA)

No four points of S are co-circular!

Lemma 88

A Voronoi node is the common intersection of exactly three Voronoi edges. It is equidistant to the three points of S which lie in the Voronoi regions it belongs to.

Corollary 89

A Voronoi diagram is a 3-regular (plane) graph.

Voronoi Diagram: Properties

General position assumed (GPA)

No four points of S are co-circular!

Lemma 88

A Voronoi node is the common intersection of exactly three Voronoi edges. It is equidistant to the three points of S which lie in the Voronoi regions it belongs to.

Corollary 89

A Voronoi diagram is a 3-regular (plane) graph.

Lemma 90

The disk D centered at a Voronoi node v that passes through the node's three equidistant points $p_1, p_2, p_3 \in S$ contains no other points of S in its interior.

Lemma 91

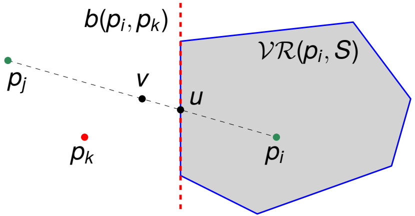
For $p_i \in S$, every nearest neighbor of p_i defines an edge of $\mathcal{VP}(p_i, S)$.

Lemma 91

For $p_i \in S$, every nearest neighbor of p_i defines an edge of $\mathcal{VP}(p_i, S)$.

Proof :

- Let $p_j \in S$ be a nearest neighbor of p_i, and let v be their midpoint.
- Suppose that v does not lie on the boundary of $\mathcal{VR}(p_i, S)$. Then it has to lie outside of $\mathcal{VP}(p_i, S)$!

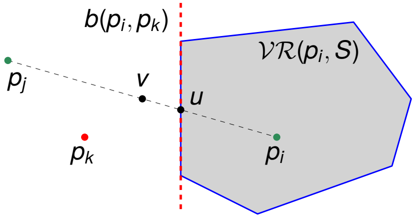


Lemma 91

For $p_i \in S$, every nearest neighbor of p_i defines an edge of $\mathcal{VP}(p_i, S)$.

Proof :

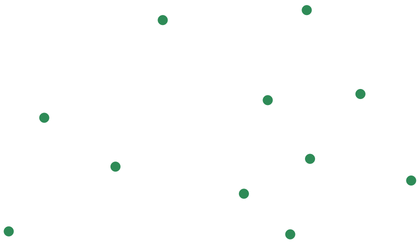
- Let $p_j \in S$ be a nearest neighbor of p_i, and let v be their midpoint.
- Suppose that v does not lie on the boundary of $\mathcal{VR}(p_i, S)$. Then it has to lie outside of $\mathcal{VP}(p_i, S)$!

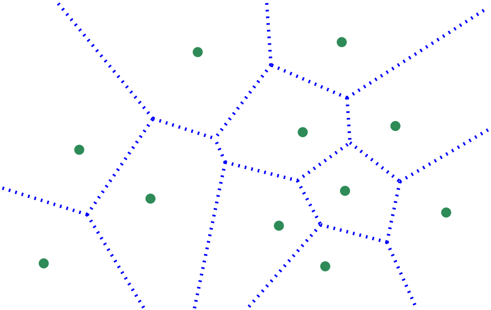


- Then the line segment $\overline{p_i v}$ would intersect some edge of $\mathcal{VP}(p_i, S)$. Assume that it intersects the bisector of $\overline{p_i p_k}$ in the point u. Now $|\overline{p_i u}| < |\overline{p_i v}|$, and therefore $|\overline{p_i p_k}| \leq 2|\overline{p_i u}| < 2|\overline{p_i v}| = |\overline{p_i p_j}|$, and we would have p_k closer to p_i than p_j, which is a contradiction.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Delaunay Triangulation: Definition and Properties

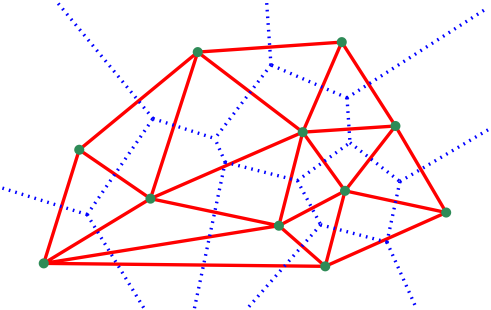




Definition 92 (Delaunay triangulation)

A *Delaunay triangulation* (DT), $\mathcal{DT}(S)$, of S is a plane geometric graph that is *dual* to the Voronoi diagram of S:

- The nodes of the graph are given by the points of S.
- Two points are connected by a line segment, and form an edge of $\mathcal{DT}(S)$, exactly if they share a Voronoi edge of $\mathcal{VD}(S)$.

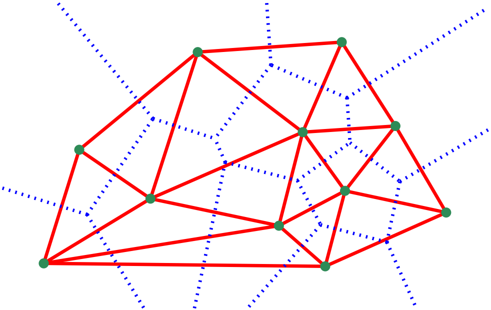


Delaunay Triangulation: Definition and Properties

Definition 92 (Delaunay triangulation)

A *Delaunay triangulation* (DT), $\mathcal{DT}(S)$, of S is a plane geometric graph that is *dual* to the Voronoi diagram of S:

- The nodes of the graph are given by the points of S.
- Two points are connected by a line segment, and form an edge of $\mathcal{DT}(S)$, exactly if they share a Voronoi edge of $\mathcal{VD}(S)$.

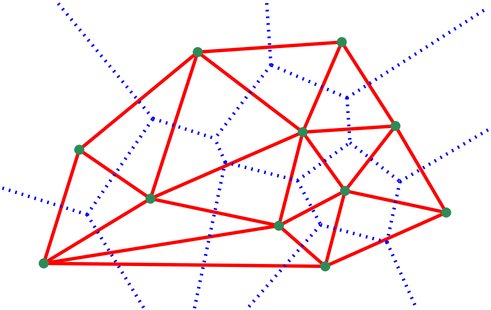


- Named after Boris Nikolaevich Delaunay (1890–1980).

Lemma 93

The structure $\mathcal{DT}(S)$ does indeed form a triangulation of S.

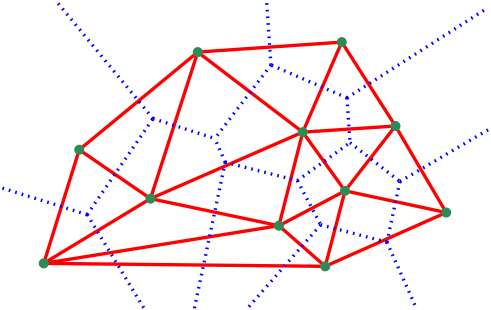
- Thus, the interior faces of $\mathcal{DT}(S)$ are triangles that are defined by triples of S, with each face corresponding to exactly one node of $\mathcal{VD}(S)$.



Lemma 93

The structure $\mathcal{DT}(S)$ does indeed form a triangulation of S.

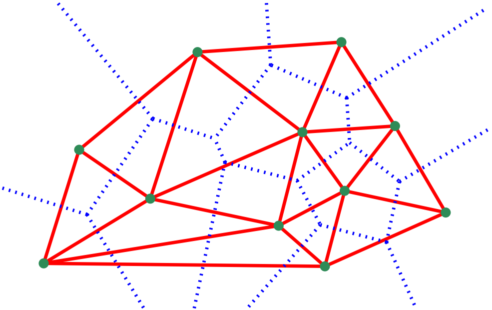
- Thus, the interior faces of $\mathcal{DT}(S)$ are triangles that are defined by triples of S, with each face corresponding to exactly one node of $\mathcal{VD}(S)$.
- By definition, every edge of the Delaunay triangulation has a corresponding edge in the Voronoi diagram.
- $\mathcal{DT}(S)$ is called the *straight-line dual* of $\mathcal{VD}(S)$.



Lemma 93

The structure $\mathcal{DT}(S)$ does indeed form a triangulation of S.

- Thus, the interior faces of $\mathcal{DT}(S)$ are triangles that are defined by triples of S, with each face corresponding to exactly one node of $\mathcal{VD}(S)$.
- By definition, every edge of the Delaunay triangulation has a corresponding edge in the Voronoi diagram.
- $\mathcal{DT}(S)$ is called the *straight-line dual* of $\mathcal{VD}(S)$.

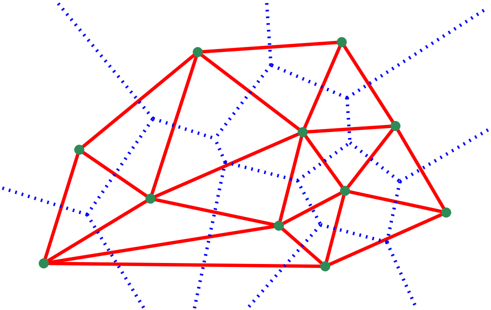


- Note: An edge of $\mathcal{DT}(S)$ need not intersect its dual Voronoi edge.

Lemma 93

The structure $\mathcal{DT}(S)$ does indeed form a triangulation of S.

- Thus, the interior faces of $\mathcal{DT}(S)$ are triangles that are defined by triples of S, with each face corresponding to exactly one node of $\mathcal{VD}(S)$.
- By definition, every edge of the Delaunay triangulation has a corresponding edge in the Voronoi diagram.
- $\mathcal{DT}(S)$ is called the *straight-line dual* of $\mathcal{VD}(S)$.



- Note: An edge of $\mathcal{DT}(S)$ need not intersect its dual Voronoi edge.
- If no four points of S are co-circular then its Delaunay triangulation is unique.

Complexity of Voronoi Diagram and Delaunay Triangulation

Lemma 94

The Delaunay triangulation of n points has at most $3n - 6$ edges and at most $2n - 4$ faces (for all $n \geq 3$).

Lemma 94

The Delaunay triangulation of n points has at most $3n - 6$ edges and at most $2n - 4$ faces (for all $n \geq 3$).

Proof : Recall that a Delaunay triangulation forms a connected planar graph on n nodes, where every bounded face is bounded by exactly three edges.

Lemma 94

The Delaunay triangulation of n points has at most $3n - 6$ edges and at most $2n - 4$ faces (for all $n \geq 3$).

Proof: Recall that a Delaunay triangulation forms a connected planar graph on n nodes, where every bounded face is bounded by exactly three edges. Hence, Euler's formula $V - E + F = 2$ can be applied, with $V := n$, and we get

$$E \leq 3V - 6 \qquad \text{and} \qquad F \leq 2V - 4 \qquad \text{and} \qquad F \leq \frac{2}{3}E.$$

□

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Complexity of Voronoi Diagram and Delaunay Triangulation

Lemma 94

The Delaunay triangulation of n points has at most $3n - 6$ edges and at most $2n - 4$ faces (for all $n \geq 3$).

Proof : Recall that a Delaunay triangulation forms a connected planar graph on n nodes, where every bounded face is bounded by exactly three edges. Hence, Euler's formula $V - E + F = 2$ can be applied, with $V := n$, and we get

$$E \leq 3V - 6 \qquad \text{and} \qquad F \leq 2V - 4 \qquad \text{and} \qquad F \leq \frac{2}{3}E. \qquad \square$$

- We conclude that

$\mathcal{DT}$: $\leq 3n - 6$ edges and thus $\mathcal{VD}$: $\leq 3n - 6$ edges,
$\mathcal{DT}$: $\leq 2n - 4$ faces and thus $\mathcal{VD}$: $\leq 2n - 5$ nodes.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 94

The Delaunay triangulation of n points has at most $3n-6$ edges and at most $2n-4$ faces (for all $n \geq 3$).

Proof: Recall that a Delaunay triangulation forms a connected planar graph on n nodes, where every bounded face is bounded by exactly three edges. Hence, Euler's formula $V - E + F = 2$ can be applied, with $V := n$, and we get

$$E \leq 3V - 6 \qquad \text{and} \qquad F \leq 2V - 4 \qquad \text{and} \qquad F \leq \frac{2}{3}E. \qquad \square$$

- We conclude that

$\mathcal{DT}$: $\leq 3n-6$ edges and thus $\mathcal{VD}$: $\leq 3n-6$ edges,
$\mathcal{DT}$: $\leq 2n-4$ faces and thus $\mathcal{VD}$: $\leq 2n-5$ nodes.

Lemma 95

The Voronoi diagram of n points has at most $3n-6$ edges and at most $2n-5$ nodes.

Lemma 94

The Delaunay triangulation of n points has at most $3n-6$ edges and at most $2n-4$ faces (for all $n \geq 3$).

Proof: Recall that a Delaunay triangulation forms a connected planar graph on n nodes, where every bounded face is bounded by exactly three edges. Hence, Euler's formula $V - E + F = 2$ can be applied, with $V := n$, and we get

$$E \leq 3V - 6 \qquad \text{and} \qquad F \leq 2V - 4 \qquad \text{and} \qquad F \leq \frac{2}{3}E. \qquad \square$$

- We conclude that

$$\begin{array}{llll} \mathcal{DT}: & \leq 3n-6 \text{ edges} & \text{and thus} & \mathcal{VD}: \quad \leq 3n-6 \text{ edges,} \\ \mathcal{DT}: & \leq 2n-4 \text{ faces} & \text{and thus} & \mathcal{VD}: \quad \leq 2n-5 \text{ nodes.} \end{array}$$

Lemma 95

The Voronoi diagram of n points has at most $3n-6$ edges and at most $2n-5$ nodes.

Corollary 96

A Voronoi polygon has at most $n-1$ edges, but only six edges on average.

- The fact that the Voronoi polygons of nearest neighbors always have a Voronoi edge in common implies that it is sufficient to check all points in adjacent Voronoi regions to find a nearest neighbor of a point $p_i \in S$.
- Thus, knowledge of the Voronoi diagram helps to solve ClosestPair and AllNearestNeighbors in $O(n)$ time.

Proximity Problems Solved by Voronoi Diagrams

- The fact that the Voronoi polygons of nearest neighbors always have a Voronoi edge in common implies that it is sufficient to check all points in adjacent Voronoi regions to find a nearest neighbor of a point $p_i \in S$.
- Thus, knowledge of the Voronoi diagram helps to solve CLOSESTPAIR and ALLNEARESTNEIGHBORS in $O(n)$ time.
- The Voronoi polygon of $p_i \in S$ is unbounded if and only if p_i is a vertex of the convex hull of the set S. (Proof: See Preparata&Shamos.) This means that the vertices of $CH(S)$ are those points of S which have unbounded Voronoi polygons.
- Thus, knowledge of the Voronoi diagram allows to solve CONVEXHULL in $O(n)$ time.

Proximity Problems Solved by Voronoi Diagrams

- The fact that the Voronoi polygons of nearest neighbors always have a Voronoi edge in common implies that it is sufficient to check all points in adjacent Voronoi regions to find a nearest neighbor of a point $p_i \in S$.
- Thus, knowledge of the Voronoi diagram helps to solve CLOSESTPAIR and ALLNEARESTNEIGHBORS in $O(n)$ time.
- The Voronoi polygon of $p_i \in S$ is unbounded if and only if p_i is a vertex of the convex hull of the set S. (Proof: See Preparata&Shamos.) This means that the vertices of $CH(S)$ are those points of S which have unbounded Voronoi polygons.
- Thus, knowledge of the Voronoi diagram allows to solve CONVEXHULL in $O(n)$ time.
- A MAXIMUMEMPTYCIRCLE can be found in $O(n)$ time by scanning all nodes of the Voronoi diagram; see later.

Proximity Problems Solved by Voronoi Diagrams

- The fact that the Voronoi polygons of nearest neighbors always have a Voronoi edge in common implies that it is sufficient to check all points in adjacent Voronoi regions to find a nearest neighbor of a point $p_i \in S$.
- Thus, knowledge of the Voronoi diagram helps to solve CLOSESTPAIR and ALLNEARESTNEIGHBORS in $O(n)$ time.
- The Voronoi polygon of $p_i \in S$ is unbounded if and only if p_i is a vertex of the convex hull of the set S. (Proof: See Preparata&Shamos.) This means that the vertices of $CH(S)$ are those points of S which have unbounded Voronoi polygons.
- Thus, knowledge of the Voronoi diagram allows to solve CONVEXHULL in $O(n)$ time.
- A MAXIMUMEMPTYCIRCLE can be found in $O(n)$ time by scanning all nodes of the Voronoi diagram; see later.
- After $O(n)$ preprocessing for building a search data structure of size $O(n)$ on top of the Voronoi diagram, NEARESTNEIGHBORSEARCH queries can be handled in $O(\log n)$ time. (However, the constants are high — better techniques are known for point sites!)

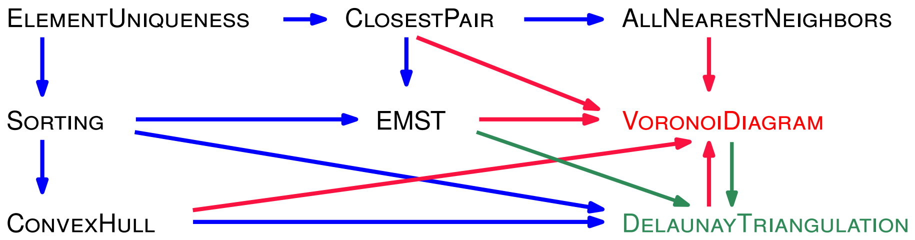
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Lemma 97

The Voronoi diagram of n points in $\mathbb{R}^2$ can be obtained in $O(n)$ time from the Delaunay triangulation, and the Delaunay triangulation can be obtained in $O(n)$ time from the Voronoi diagram.

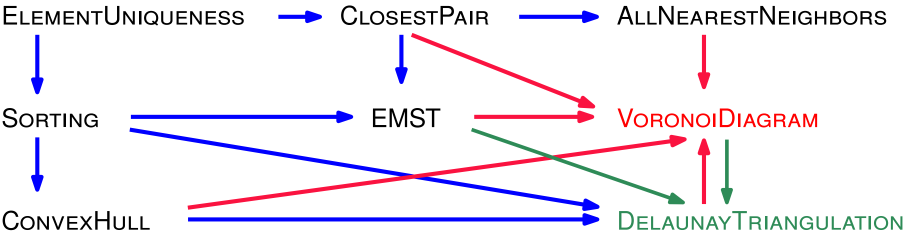
Lemma 97

The Voronoi diagram of n points in $\mathbb{R}^2$ can be obtained in $O(n)$ time from the Delaunay triangulation, and the Delaunay triangulation can be obtained in $O(n)$ time from the Voronoi diagram.



Lemma 97

The Voronoi diagram of n points in $\mathbb{R}^2$ can be obtained in $O(n)$ time from the Delaunay triangulation, and the Delaunay triangulation can be obtained in $O(n)$ time from the Voronoi diagram.



Theorem 98 (Chazelle 1993)

The Voronoi diagram of n points in $\mathbb{R}^d$ can be computed in optimal $O(n \log n + n^{\lceil \frac{d}{2} \rceil})$ time.

5 Voronoi Diagrams of Points

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

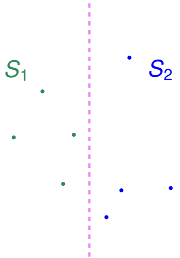
Divide&Conquer Algorithm

- Preprocessing: Sort the points of S by x-coordinates. This takes $O(n \log n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Divide&Conquer Algorithm

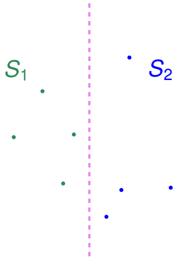
- Preprocessing: Sort the points of S by x-coordinates. This takes $O(n \log n)$ time.
- Divide:
  - Divide S into two subsets S_1 and S_2 of roughly equal size such that the points in S_1 lie to the left and the points in S_2 lie to the right of a vertical line.
  - This step can be carried out in $O(n)$ time.

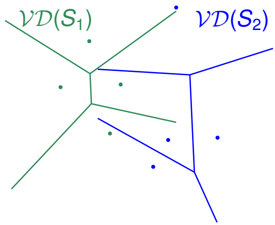


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Divide&Conquer Algorithm

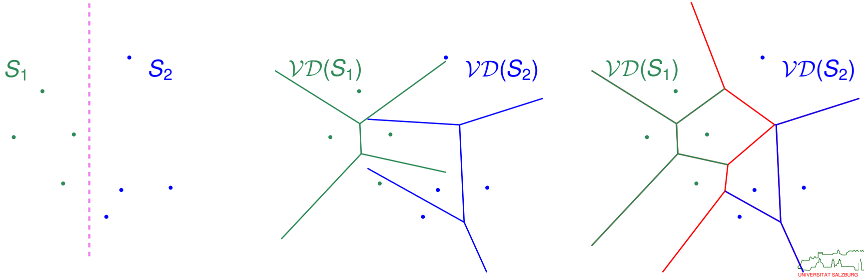
- Preprocessing: Sort the points of S by x-coordinates. This takes $O(n \log n)$ time.
- Divide:
  - Divide S into two subsets S_1 and S_2 of roughly equal size such that the points in S_1 lie to the left and the points in S_2 lie to the right of a vertical line.
  - This step can be carried out in $O(n)$ time.
- Conquer (aka "Merge"):
  - Assume that $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$ are known.



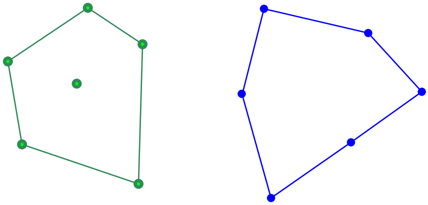


Divide&Conquer Algorithm

- Preprocessing: Sort the points of S by x-coordinates. This takes $O(n \log n)$ time.
- Divide:
  - Divide S into two subsets S_1 and S_2 of roughly equal size such that the points in S_1 lie to the left and the points in S_2 lie to the right of a vertical line.
  - This step can be carried out in $O(n)$ time.
- Conquer (aka “Merge”):
  - Assume that $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$ are known.
  - Clip those parts of $\mathcal{VD}(S_1)$ that lie to the “right” of a so-called *dividing chain*.
  - Analogously for $\mathcal{VD}(S_2)$.

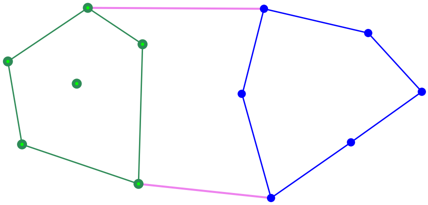


Divide&Conquer Algorithm: Merge

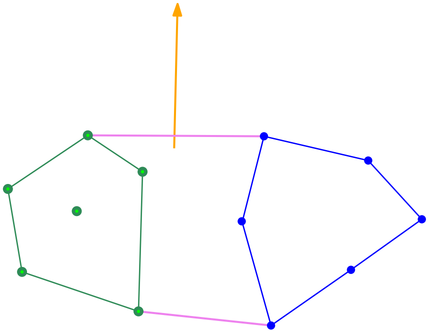


Divide&Conquer Algorithm: Merge

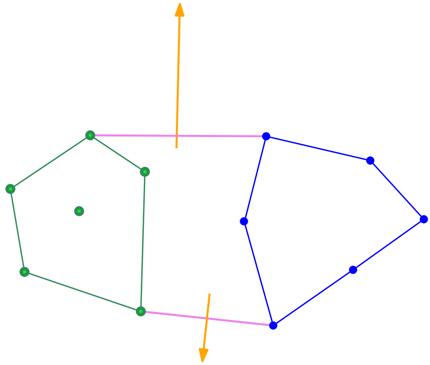
1. Find upper and lower supporting edges of $CH(S_1)$ and $CH(S_2)$ in order to form the convex hull $CH(S)$.



1. Find upper and lower supporting edges of $CH(S_1)$ and $CH(S_2)$ in order to form the convex hull $CH(S)$.
   - Bisector (ray) defined by upper bridge of convex hull is part of the dividing chain.

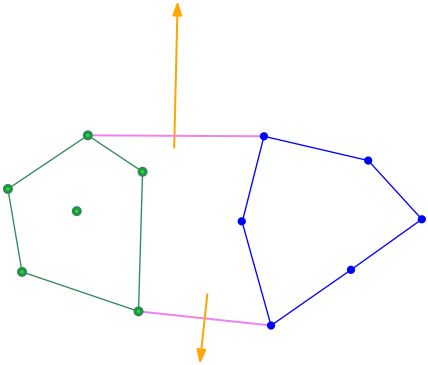


1. Find upper and lower supporting edges of $CH(S_1)$ and $CH(S_2)$ in order to form the convex hull $CH(S)$.
   - Bisector (ray) defined by upper bridge of convex hull is part of the dividing chain.
   - Bisector (ray) defined by lower bridge of convex hull is part of the dividing chain.



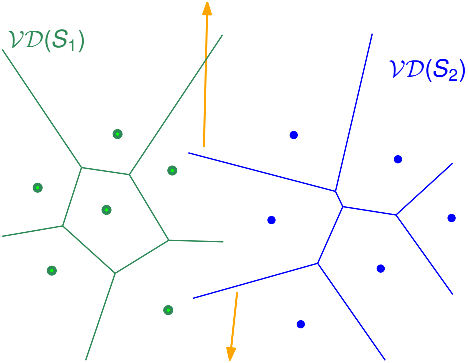
Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:



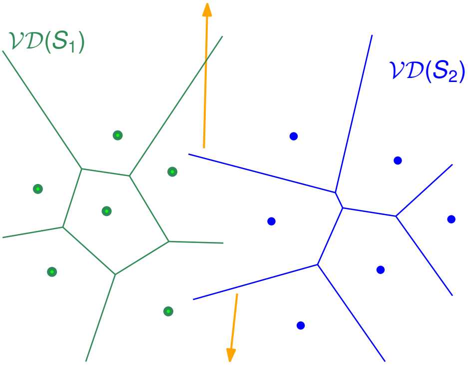
Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:



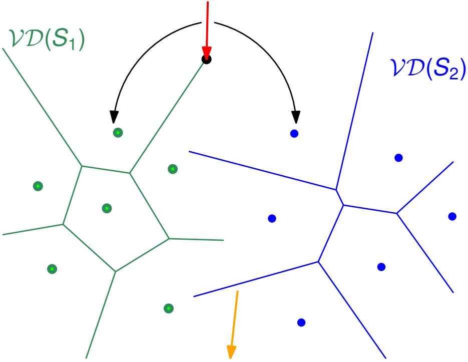
Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:
   - Start by walking down along the upper ray.
   - Intersect the ray with $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$.



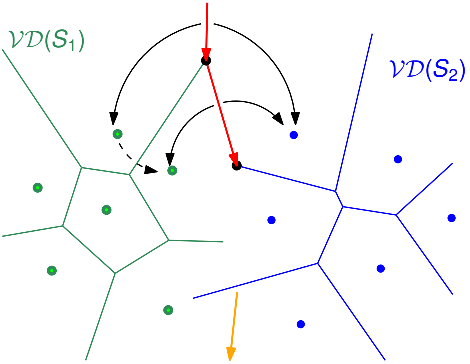
Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:
   - Start by walking down along the upper ray.
   - Intersect the ray with $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$.
   - Pick the first intersection as new Voronoi node.

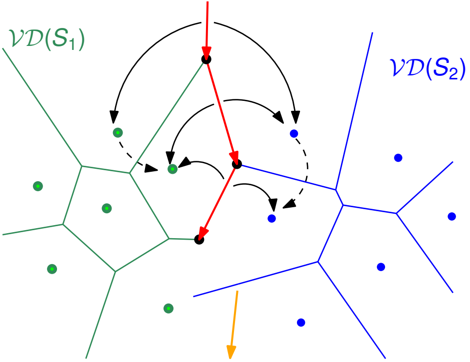


Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:
   - Start by walking down along the upper ray.
   - Intersect the ray with $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$.
   - Pick the first intersection as new Voronoi node.
   - The next ray is the new bisector originating at this node.

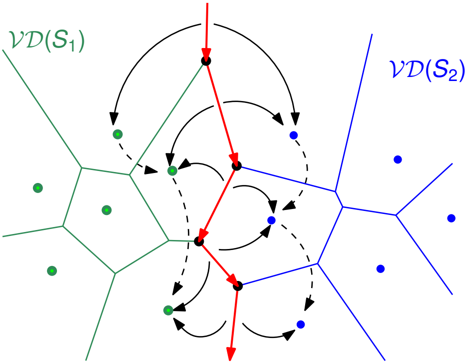


2. Build dividing chain from top to bottom:
   - Start by walking down along the upper ray.
   - Intersect the ray with $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$.
   - Pick the first intersection as new Voronoi node.
   - The next ray is the new bisector originating at this node.
   - Continue this jagged walk until the lower ray is reached.



Divide&Conquer Algorithm: Merge

2. Build dividing chain from top to bottom:
   - Start by walking down along the upper ray.
   - Intersect the ray with $\mathcal{VD}(S_1)$ and $\mathcal{VD}(S_2)$.
   - Pick the first intersection as new Voronoi node.
   - The next ray is the new bisector originating at this node.
   - Continue this jagged walk until the lower ray is reached.



Lemma 99

The merge can be carried out in $O(n)$ time, based on the Shamos-Hoey scanning scheme that prevents Voronoi edges from being searched for an intersection for more than a constant number of times.

Lemma 99

The merge can be carried out in $O(n)$ time, based on the Shamos-Hoey scanning scheme that prevents Voronoi edges from being searched for an intersection for more than a constant number of times.

- If the merge is carried out in linear time then we get a familiar recurrence relation for the time T:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n),$$

and therefore

$$T \in O(n \log n).$$

Divide&Conquer Algorithm: Complexity Analysis

Lemma 99

The merge can be carried out in $O(n)$ time, based on the Shamos-Hoey scanning scheme that prevents Voronoi edges from being searched for an intersection for more than a constant number of times.

- If the merge is carried out in linear time then we get a familiar recurrence relation for the time T:

  $$T(n) = 2T\left(\frac{n}{2}\right) + O(n),$$

  and therefore

  $$T \in O(n \log n).$$

Theorem 100

The divide&conquer algorithm computes $\mathcal{VD}(S)$ for a set S of n points in optimal $O(n \log n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction

- We compute the Voronoi diagram $\mathcal{VD}(S)$ of a set $S := \{p_1, p_2, \ldots, p_n\}$ of n points by inserting the i-th point p_i into $\mathcal{VD}(\{p_1, p_2, \ldots, p_{i-1}\})$, for $1 \le i \le n$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction

- We compute the Voronoi diagram $\mathcal{VD}(S)$ of a set $S := \{p_1, p_2, \ldots, p_n\}$ of n points by inserting the i-th point p_i into $\mathcal{VD}(\{p_1, p_2, \ldots, p_{i-1}\})$, for $1 \leq i \leq n$.
- If we could achieve constant complexity per insertion then a linear algorithm would result:
  $\longrightarrow$ Best case: $O(n)$.
- An insertion could, however, affect all other sites:
  $\longrightarrow$ Worst case: $O(n^2)$, or even worse.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction

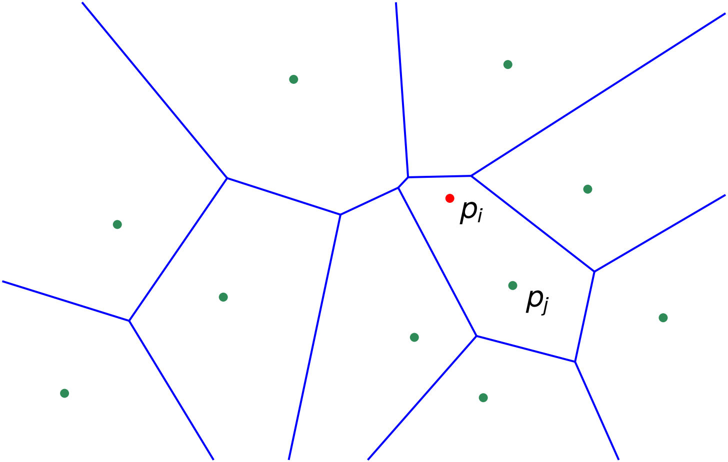
- We compute the Voronoi diagram $\mathcal{VD}(S)$ of a set $S := \{p_1, p_2, \ldots, p_n\}$ of n points by inserting the i-th point p_i into $\mathcal{VD}(\{p_1, p_2, \ldots, p_{i-1}\})$, for $1 \leq i \leq n$.
- If we could achieve constant complexity per insertion then a linear algorithm would result:
  $\longrightarrow$ Best case: $O(n)$.
- An insertion could, however, affect all other sites:
  $\longrightarrow$ Worst case: $O(n^2)$, or even worse.
- Since, on average, every Voronoi region is bounded by six Voronoi edges there is reason to hope that a close-to-linear time complexity can be achieved.

Incremental Construction

- We compute the Voronoi diagram $\mathcal{VD}(S)$ of a set $S := \{p_1, p_2, \ldots, p_n\}$ of n points by inserting the i-th point p_i into $\mathcal{VD}(\{p_1, p_2, \ldots, p_{i-1}\})$, for $1 \leq i \leq n$.
- If we could achieve constant complexity per insertion then a linear algorithm would result:
  $\longrightarrow$ Best case: $O(n)$.
- An insertion could, however, affect all other sites:
  $\longrightarrow$ Worst case: $O(n^2)$, or even worse.
- Since, on average, every Voronoi region is bounded by six Voronoi edges there is reason to hope that a close-to-linear time complexity can be achieved.
- Let $S' := \{p_1, p_2, \ldots, p_{i-1}\}$.

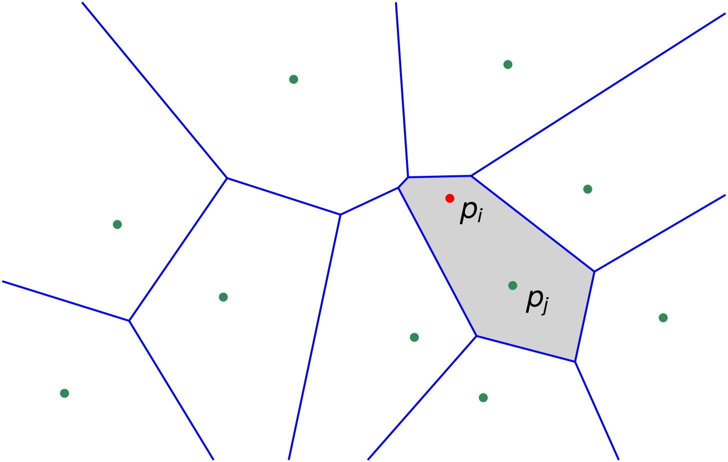
Incremental Construction: Basic Algorithm

1. Nearest-neighbor search among S': Determine $1 \leq j < i$ such that the new point p_i lies in $\mathcal{VR}(p_j, S')$.



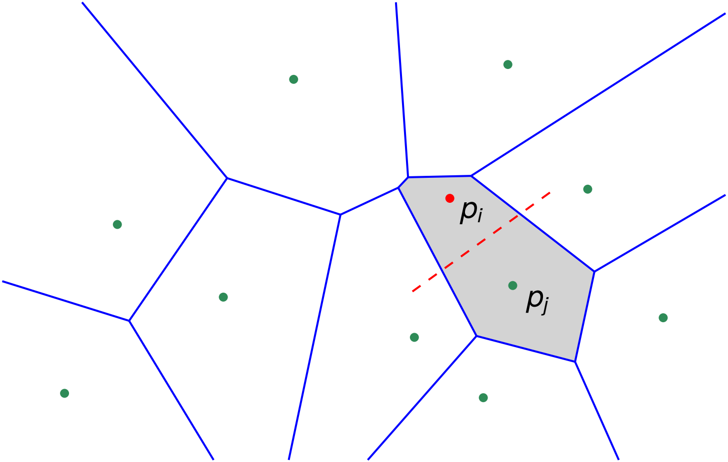
Incremental Construction: Basic Algorithm

1. Nearest-neighbor search among S': Determine $1 \leq j < i$ such that the new point p_i lies in $\mathcal{VR}(p_j, S')$.



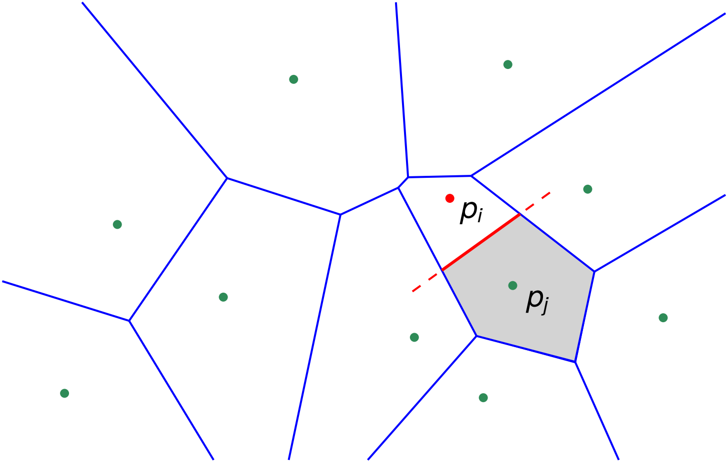
Incremental Construction: Basic Algorithm

2. Construct the bisector $b(p_i, p_j)$ between p_i and p_j, intersect it with $\mathcal{VP}(p_j, S')$, and clip that portion of $\mathcal{VP}(p_j, S')$ which is closer to p_i than to p_j.



Incremental Construction: Basic Algorithm

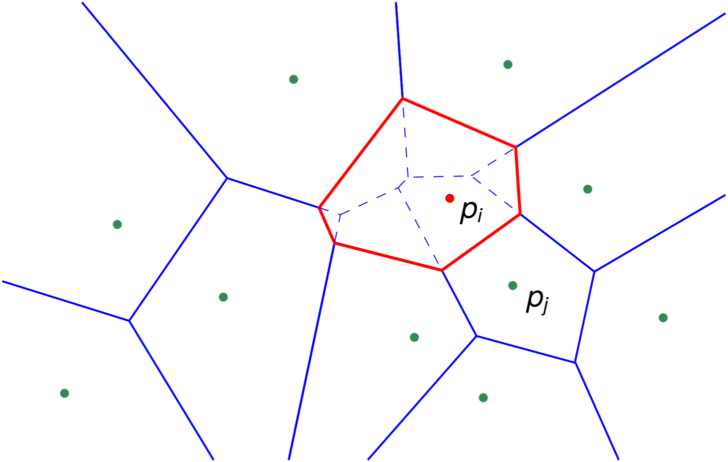
2. Construct the bisector $b(p_i, p_j)$ between p_i and p_j, intersect it with $\mathcal{VP}(p_j, S')$, and clip that portion of $\mathcal{VP}(p_j, S')$ which is closer to p_i than to p_j.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

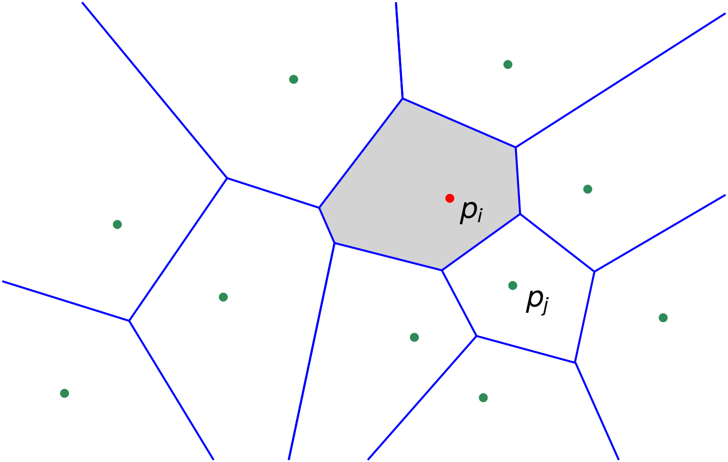
Incremental Construction: Basic Algorithm

3. Generate $\mathcal{VP}(p_i, \{p_1, p_2, \ldots, p_i\})$ by a circular scan around p_i, similar to the construction of the dividing chain in the divide&conquer algorithm.

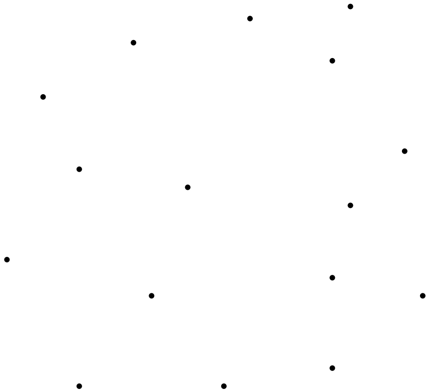


Incremental Construction: Basic Algorithm

- The scan is finished once it returns to $\mathcal{VR}(p_j, S')$.

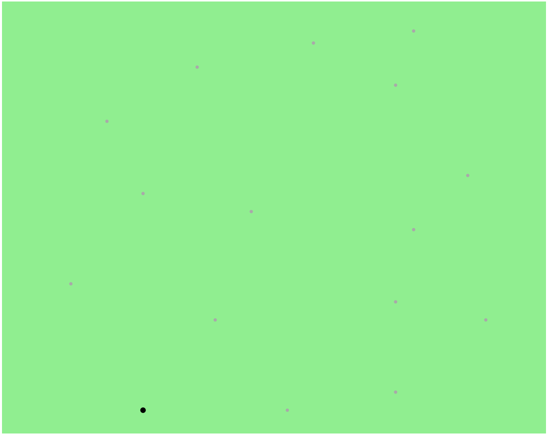


- Incremental construction of the Voronoi diagram of a set of points.



Incremental Construction: Animation

- Incremental construction of the Voronoi diagram of a set of points. Insert points into VD under construction, one at a time, in random order.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction: Complexity

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.

Incremental Construction: Complexity

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.
- It is fairly easy to pick n points and number them "appropriately" such that the insertion of the i-th point requires the generation of $O(i)$ Voronoi edges!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction: Complexity

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.
- It is fairly easy to pick n points and number them "appropriately" such that the insertion of the i-th point requires the generation of $O(i)$ Voronoi edges!
- But randomization comes to our rescue, and one can prove the following result.

Theorem 101

Randomized incremental construction allows to compute the Voronoi diagram of n points in $O(n \log n)$ expected time.

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.
- It is fairly easy to pick n points and number them "appropriately" such that the insertion of the i-th point requires the generation of $O(i)$ Voronoi edges!
- But randomization comes to our rescue, and one can prove the following result.

Theorem 101

Randomized incremental construction allows to compute the Voronoi diagram of n points in $O(n \log n)$ expected time.

Sketch of Proof: A search data structure ("history DAG") can be maintained that has $O(n)$ expected size, and supports nearest-neighbor queries in $O(\log n)$ expected time. Updates can be done in $O(1)$ expected time. □

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction: Complexity

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.
- It is fairly easy to pick n points and number them “appropriately” such that the insertion of the i-th point requires the generation of $O(i)$ Voronoi edges!
- But randomization comes to our rescue, and one can prove the following result.

Theorem 101

Randomized incremental construction allows to compute the Voronoi diagram of n points in $O(n \log n)$ expected time.

Sketch of Proof: A search data structure (“history DAG”) can be maintained that has $O(n)$ expected size, and supports nearest-neighbor queries in $O(\log n)$ expected time. Updates can be done in $O(1)$ expected time. □

- The actual proof of these claims relies on backwards analysis.
- This result is independent of the point distribution, as long as the insertion order is random!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Incremental Construction: Complexity

- The complexity mainly depends on the complexity of the nearest-neighbor search and on the number of edges generated/deleted during the scan.
- It is fairly easy to pick n points and number them "appropriately" such that the insertion of the i-th point requires the generation of $O(i)$ Voronoi edges!
- But randomization comes to our rescue, and one can prove the following result.

Theorem 101

Randomized incremental construction allows to compute the Voronoi diagram of n points in $O(n \log n)$ expected time.

Sketch of Proof: A search data structure ("history DAG") can be maintained that has $O(n)$ expected size, and supports nearest-neighbor queries in $O(\log n)$ expected time. Updates can be done in $O(1)$ expected time. □

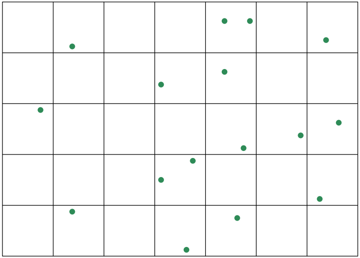
- The actual proof of these claims relies on backwards analysis.
- This result is independent of the point distribution, as long as the insertion order is random!
- This is a nice result seen from a theoretical point of view, but an actual implementation of the search structure would require a bit of work . . .

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Hashing for Nearest-Neighbor Searching

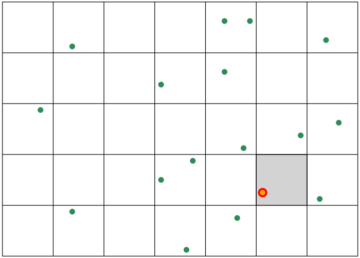
- The bounding box of S (or of a slightly larger region that contains S) is partitioned into rectangular cells of uniform size by means of a regular grid.

- The bounding box of S (or of a slightly larger region that contains S) is partitioned into rectangular cells of uniform size by means of a regular grid.
- For every cell c, all points of $\{p_1, p_2, \ldots, p_{i-1}\}$ that lie in c are stored with c. (Alternatively, only one point is stored per cell.)



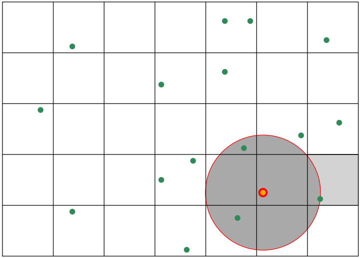
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.



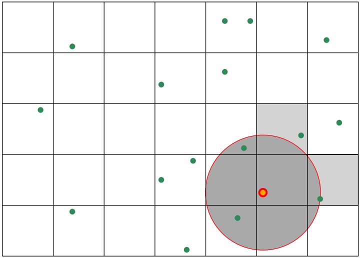
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.



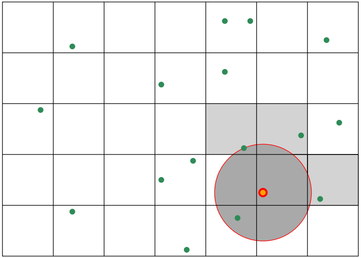
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.
  - We continue searching in c and in those cells around c which are intersected by a disk D with radius δ centered at p_i.



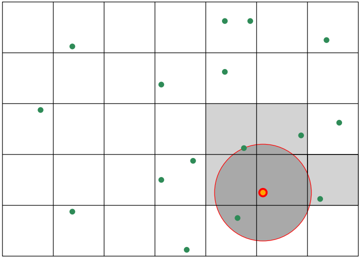
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.
  - We continue searching in c and in those cells around c which are intersected by a disk D with radius δ centered at p_i.
  - Whenever a point of $\{p_1, p_2, \ldots, p_{i-1}\}$ that is closer to p_i is found, we reduce δ appropriately.



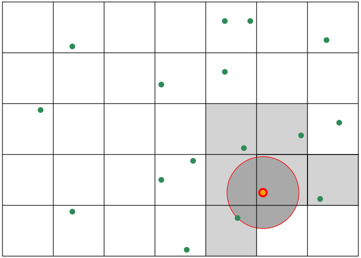
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.
  - We continue searching in c and in those cells around c which are intersected by a disk D with radius δ centered at p_i.
  - Whenever a point of $\{p_1, p_2, \ldots, p_{i-1}\}$ that is closer to p_i is found, we reduce δ appropriately.



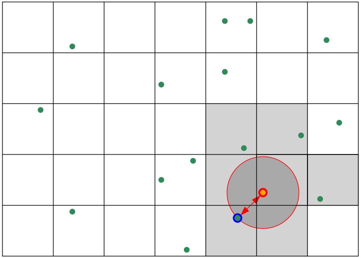
Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.
  - We continue searching in c and in those cells around c which are intersected by a disk D with radius δ centered at p_i.
  - Whenever a point of $\{p_1, p_2, \ldots, p_{i-1}\}$ that is closer to p_i is found, we reduce δ appropriately.



Geometric Hashing for Nearest-Neighbor Searching

- To find the point p_j closest to point p_i:
  - Determine the cell c in which p_i lies.
  - By searching in c (and possibly in its neighboring cells, if c is empty), we find a first candidate for the nearest neighbor.
  - Let δ be the distance from p_i to this point.
  - We continue searching in c and in those cells around c which are intersected by a disk D with radius δ centered at p_i.
  - Whenever a point of $\{p_1, p_2, \ldots, p_{i-1}\}$ that is closer to p_i is found, we reduce δ appropriately.
  - The search stops once no unsearched cell exists that is intersected by D.



- What is a suitable resolution of the grid?

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Hashing for Nearest-Neighbor Searching

- What is a suitable resolution of the grid? There is no universally valid answer. In any case, the grid should not use more than $O(n)$ memory!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Hashing for Nearest-Neighbor Searching

- What is a suitable resolution of the grid? There is no universally valid answer. In any case, the grid should not use more than $O(n)$ memory!

Personal experience

- Grids of the form $(w \cdot \sqrt{n}) \times (h \cdot \sqrt{n})$ seem to work nicely, with $w \cdot h = c$ for some constant c.
- The parameters w, h are chosen to adapt the resolution of the grid to the aspect ratio of the bounding box of the points.
- By experiment: $1 \leq c \leq 2$.

Geometric Hashing for Nearest-Neighbor Searching

- What is a suitable resolution of the grid? There is no universally valid answer. In any case, the grid should not use more than $O(n)$ memory!

Personal experience

- Grids of the form $(w \cdot \sqrt{n}) \times (h \cdot \sqrt{n})$ seem to work nicely, with $w \cdot h = c$ for some constant c.
- The parameters w, h are chosen to adapt the resolution of the grid to the aspect ratio of the bounding box of the points.
- By experiment: $1 \leq c \leq 2$.

- This basic scheme can be tuned considerably:
  - Switch to multi-level hashing or to kd-trees if a small sample of the points indicates that the points are distributed highly non-uniformly.
  - Adapt the grid resolution and re-hash if the number of points stored changes significantly due to insertions and deletions of points.

Geometric Hashing for Nearest-Neighbor Searching

- What is a suitable resolution of the grid? There is no universally valid answer. In any case, the grid should not use more than $O(n)$ memory!

Personal experience

- Grids of the form $(w \cdot \sqrt{n}) \times (h \cdot \sqrt{n})$ seem to work nicely, with $w \cdot h = c$ for some constant c.
- The parameters w, h are chosen to adapt the resolution of the grid to the aspect ratio of the bounding box of the points.
- By experiment: $1 \leq c \leq 2$.

- This basic scheme can be tuned considerably:
  - Switch to multi-level hashing or to kd-trees if a small sample of the points indicates that the points are distributed highly non-uniformly.
  - Adapt the grid resolution and re-hash if the number of points stored changes significantly due to insertions and deletions of points.
- Hash-based nearest-neighbor searching will work best for points that are distributed uniformly, and will fail miserably if all points end up in one cell!

- What is a suitable resolution of the grid? There is no universally valid answer. In any case, the grid should not use more than $O(n)$ memory!

Personal experience

- Grids of the form $(w \cdot \sqrt{n}) \times (h \cdot \sqrt{n})$ seem to work nicely, with $w \cdot h = c$ for some constant c.
- The parameters w, h are chosen to adapt the resolution of the grid to the aspect ratio of the bounding box of the points.
- By experiment: $1 \leq c \leq 2$.

- This basic scheme can be tuned considerably:
  - Switch to multi-level hashing or to kd-trees if a small sample of the points indicates that the points are distributed highly non-uniformly.
  - Adapt the grid resolution and re-hash if the number of points stored changes significantly due to insertions and deletions of points.
- Hash-based nearest-neighbor searching will work best for points that are distributed uniformly, and will fail miserably if all points end up in one cell!
- Still, personal experience tells me that (tuned) geometric hashing works extremely well even for point sets that are distributed highly irregularly!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm

- Can a sweep-line algorithm be applied to compute the Voronoi diagram?

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Can a sweep-line algorithm be applied to compute the Voronoi diagram?
- Principal problem: When a top-down sweep line reaches the top-most vertex of $\mathcal{VP}(p_i, S)$, then it has not yet moved over p_i!
- Thus, the information on the corresponding point site is missing when a Voronoi polygon is first encountered and Voronoi nodes are to be computed.

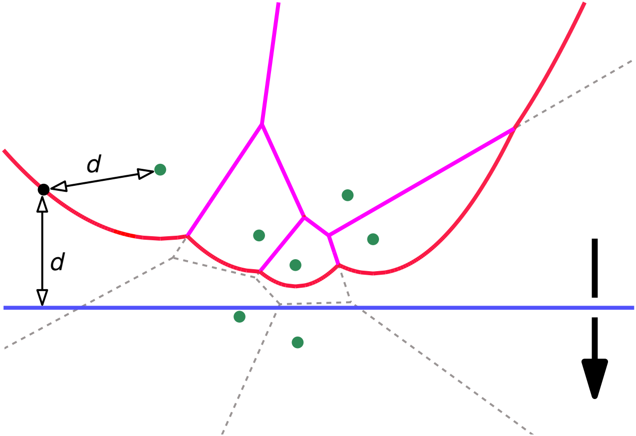
Sweep-Line Algorithm

- Can a sweep-line algorithm be applied to compute the Voronoi diagram?
- Principal problem: When a top-down sweep line reaches the top-most vertex of $\mathcal{VP}(p_i, S)$, then it has not yet moved over p_i!
- Thus, the information on the corresponding point site is missing when a Voronoi polygon is first encountered and Voronoi nodes are to be computed.
- This problem is independent of the sweep direction chosen.
- Hence, for quite some time it was assumed that the sweep-line paradigm is not applicable to Voronoi diagrams.

Sweep-Line Algorithm

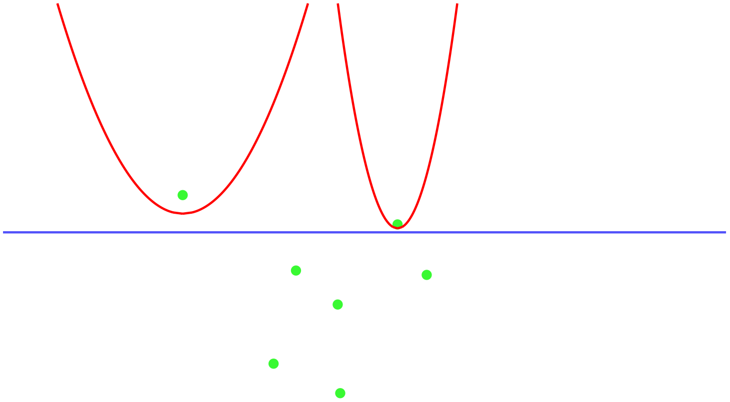
- Can a sweep-line algorithm be applied to compute the Voronoi diagram?
- Principal problem: When a top-down sweep line reaches the top-most vertex of $\mathcal{VP}(p_i, S)$, then it has not yet moved over p_i!
- Thus, the information on the corresponding point site is missing when a Voronoi polygon is first encountered and Voronoi nodes are to be computed.
- This problem is independent of the sweep direction chosen.
- Hence, for quite some time it was assumed that the sweep-line paradigm is not applicable to Voronoi diagrams.
- W.l.o.g., we move the sweep line ℓ from top to bottom.
- Remarkable idea (by S. Fortune): Rather than keeping the actual intersection of the Voronoi diagram with ℓ, we maintain information on that part of the Voronoi diagram of the points above ℓ that is guaranteed not to be affected by points below ℓ.

- The part of the Voronoi diagram that will not change any more as the sweep line continues to move downwards lies above the so-called *beach line* formed by the lower envelope of parabolic arcs: Each parabolic arc is defined by ℓ and by a point above ℓ.



Animation of Sweep-Line Algorithm

- The beach line moves downwards as the sweep-line is moved from top to bottom. A full sweep reveals the complete Voronoi diagram.



Sweep-Line Algorithm: Events

- The following two events need to be considered for the event-point schedule:
  1. Site event:
     - The sweep line ℓ passes through an input point, and a new parabolic arc needs to be inserted into the beach line.

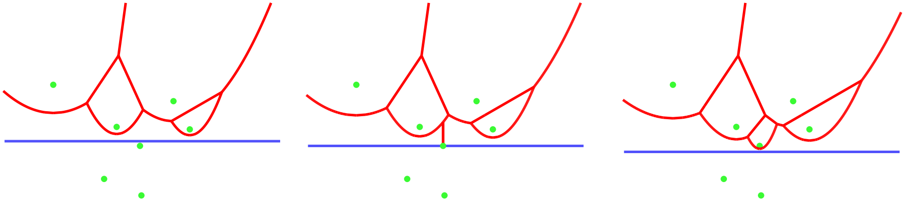
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm: Events

- The following two events need to be considered for the event-point schedule:
  1. Site event:
     - The sweep line ℓ passes through an input point, and a new parabolic arc needs to be inserted into the beach line.
  2. Circle event:
     - A parabolic arc of the beach line vanishes, i.e., degenerates to a point v, and a new Voronoi node has to be inserted at v.
     - What does this mean for the sweep line ℓ? What is the appropriate y-position of ℓ to catch this event?

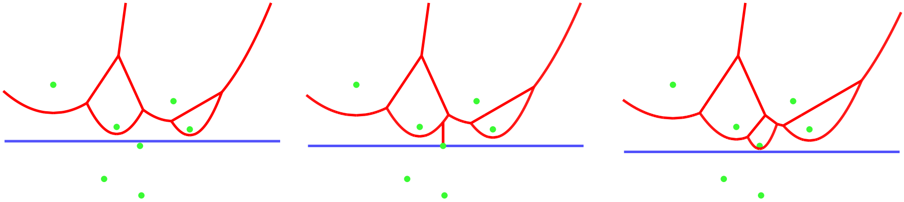
Sweep-Line Algorithm: Site Event

- If the sweep line ℓ passes through an input point then a new parabolic arc needs to be inserted into the beach line. Initially, this arc is degenerate.



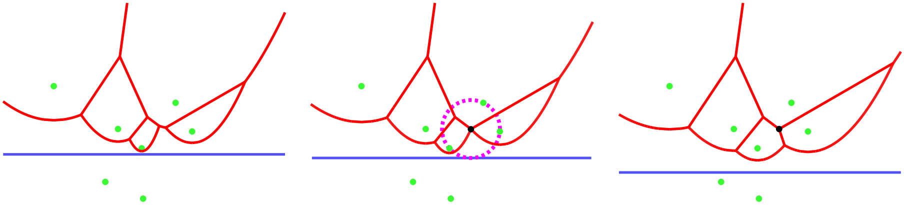
Sweep-Line Algorithm: Site Event

- If the sweep line ℓ passes through an input point then a new parabolic arc needs to be inserted into the beach line. Initially, this arc is degenerate.

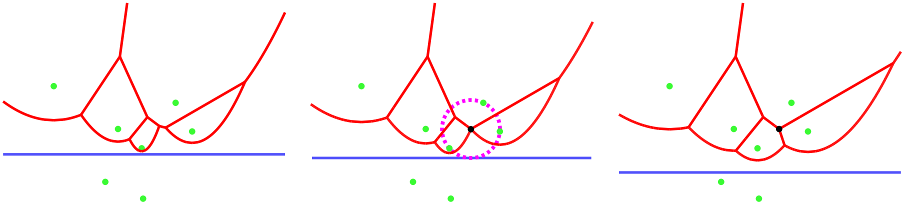


- This event occurs whenever the sweep line ℓ passes through an input point p_i.
- It is responsible for the initialization of a new Voronoi region that will become $\mathcal{VR}(p_i, S)$.

- If a parabolic arc of the beach line degenerates to a point *v* then a new Voronoi node needs to be inserted at *v*.

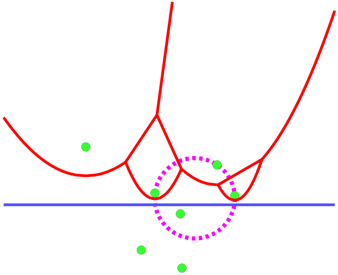


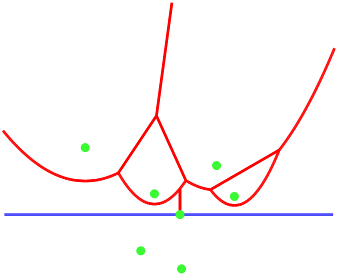
- If a parabolic arc of the beach line degenerates to a point v then a new Voronoi node needs to be inserted at v.



- A circle event occurs when the sweep line ℓ passes over the south pole of a circle through the three defining input points p_i, p_j, p_k of three consecutive parabolic arcs of the beach line.
- The center v of such a circle is equidistant to p_i, p_j, p_k and also to ℓ; it becomes a new node of the Voronoi diagram.

- Not all scheduled circle events correspond to valid new Voronoi nodes: A circle event has to be processed only if its defining three parabolic arcs still are consecutive members of the beach line at the time when the sweep line ℓ passes over the south pole of the circle.





UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm: Event-Point Schedule and Sweep-Line Status

- All input points are stored in sorted order (according to y-coordinates) in the event-point schedule.
- Whenever three parabolic arcs become consecutive for the first time — when a site event occurs — the y-coordinate of the corresponding circle event is inserted into the event-point schedule at the appropriate place.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm: Event-Point Schedule and Sweep-Line Status

- All input points are stored in sorted order (according to y-coordinates) in the event-point schedule.
- Whenever three parabolic arcs become consecutive for the first time — when a site event occurs — the y-coordinate of the corresponding circle event is inserted into the event-point schedule at the appropriate place.
- Parabolic arcs have to be inserted into the beach line when processing site events, and have to be deleted when processing circle events.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sweep-Line Algorithm: Event-Point Schedule and Sweep-Line Status

- All input points are stored in sorted order (according to y-coordinates) in the event-point schedule.
- Whenever three parabolic arcs become consecutive for the first time — when a site event occurs — the y-coordinate of the corresponding circle event is inserted into the event-point schedule at the appropriate place.
- Parabolic arcs have to be inserted into the beach line when processing site events, and have to be deleted when processing circle events.
- Both structures are best represented as balanced binary search trees, since this allows logarithmic insertion/deletion.

Sweep-Line Algorithm: Analysis

Lemma 102

The beach line is monotone with respect to the x-axis.

Lemma 103

An arc can appear on the beach line only through a site event.

Corollary 104

The beach line is a sequence of at most $2n - 1$ parabolic arcs.

Lemma 105

An arc can disappear from the beach line only through a circle event.

Theorem 106 (Fortune (1986))

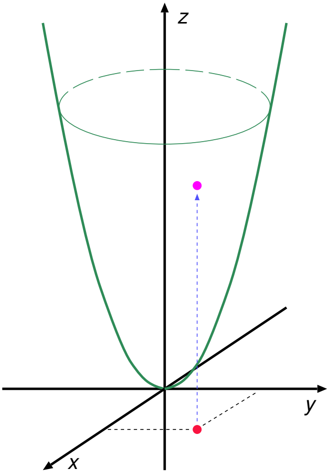
A sweep-line algorithm computes the Voronoi diagram of n points in $O(n \log n)$ time, using $O(n)$ storage.

Construction via Lifting to 3D

- Consider the transformation that maps a point $p = (p_x, p_y)$ to the non-vertical plane $h(p) \equiv z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{R}^3$.

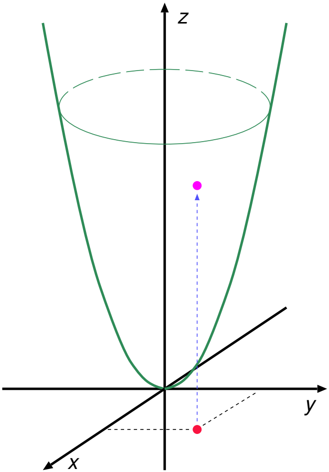
Construction via Lifting to 3D

- Consider the transformation that maps a point $p = (p_x, p_y)$ to the non-vertical plane $h(p) \equiv z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{R}^3$.
- This plane is tangent to the unit paraboloid $z = x^2 + y^2$ at the point $(p_x, p_y, p_x^2 + p_y^2)$.



Construction via Lifting to 3D

- Consider the transformation that maps a point $p = (p_x, p_y)$ to the non-vertical plane $h(p) \equiv z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{R}^3$.
- This plane is tangent to the unit paraboloid $z = x^2 + y^2$ at the point $(p_x, p_y, p_x^2 + p_y^2)$.
- Let $h^+(p)$ be the half-space induced by $h(p)$ which contains the unit paraboloid.

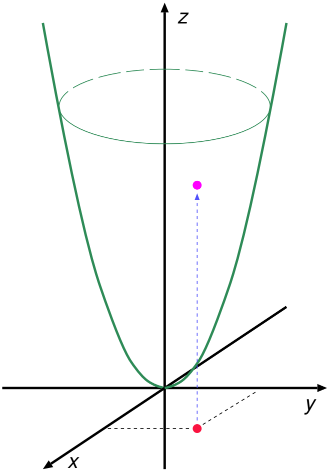


Construction via Lifting to 3D

- Consider the transformation that maps a point $p = (p_x, p_y)$ to the non-vertical plane $h(p) \equiv z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{R}^3$.
- This plane is tangent to the unit paraboloid $z = x^2 + y^2$ at the point $(p_x, p_y, p_x^2 + p_y^2)$.
- Let $h^+(p)$ be the half-space induced by $h(p)$ which contains the unit paraboloid.

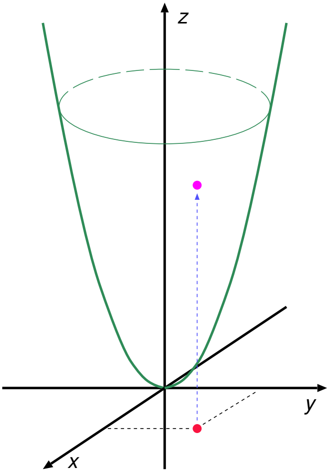
Theorem 107

For $S := \{p_1, p_2, \ldots, p_n\}$, consider the convex polyhedron $\mathcal{P} := \cap_{1 \leq i \leq n} h^+(p_i)$. The normal projection of the vertices and edges of $\mathcal{P}$ onto the xy-plane yields $\mathcal{VD}(S)$.



Construction via Lifting to 3D

- Consider the transformation that maps a point $p = (p_x, p_y)$ to the non-vertical plane $h(p) \equiv z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{R}^3$.
- This plane is tangent to the unit paraboloid $z = x^2 + y^2$ at the point $(p_x, p_y, p_x^2 + p_y^2)$.
- Let $h^+(p)$ be the half-space induced by $h(p)$ which contains the unit paraboloid.



Theorem 107

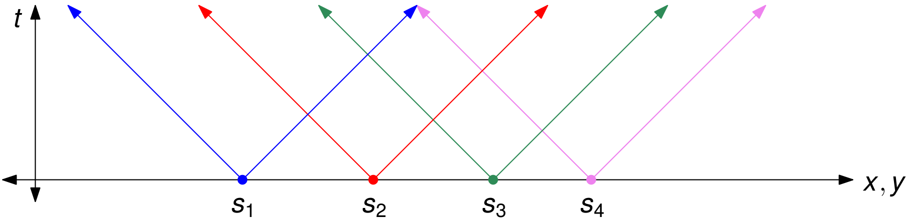
For $S := \{p_1, p_2, \ldots, p_n\}$, consider the convex polyhedron $\mathcal{P} := \cap_{1 \le i \le n} h^+(p_i)$. The normal projection of the vertices and edges of $\mathcal{P}$ onto the xy-plane yields $\mathcal{VD}(S)$.

Corollary 108

This lifting allows to construct Voronoi diagrams in $O(n \log n)$ time.

Edelsbrunner&Seidel (1986)

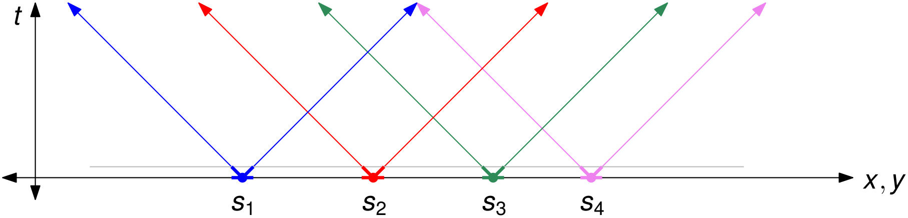
1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.



Voronoi Diagram as Minimization Diagram

Edelsbrunner&Seidel (1986)

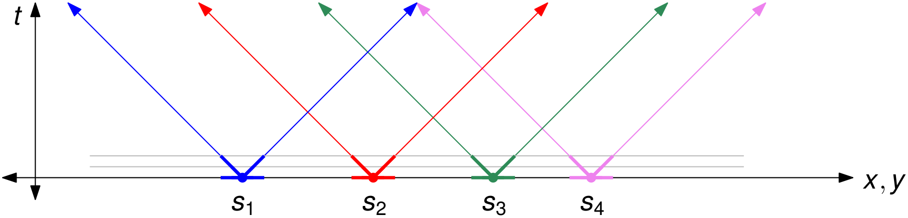
1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.
2. Every cross-section of a site's pyramid corresponds to a wavefront of the site: A point $p \in \mathbb{R}^3$ with coordinates (x, y, t) lies on the pyramid of site s if the point $p_{xy} \in \mathbb{R}^2$ with coordinates (x, y) is at weighted distance t from s.



Voronoi Diagram as Minimization Diagram

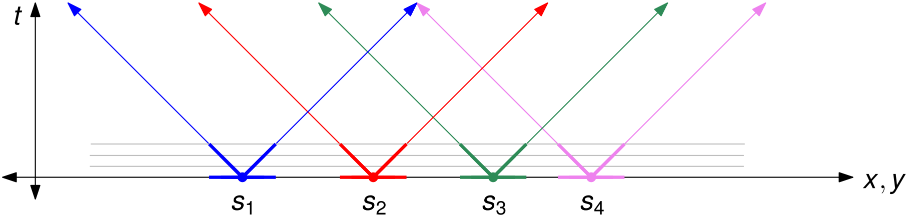
Edelsbrunner&Seidel (1986)

1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.
2. Every cross-section of a site's pyramid corresponds to a wavefront of the site: A point $p \in \mathbb{R}^3$ with coordinates (x, y, t) lies on the pyramid of site s if the point $p_{xy} \in \mathbb{R}^2$ with coordinates (x, y) is at weighted distance t from s.



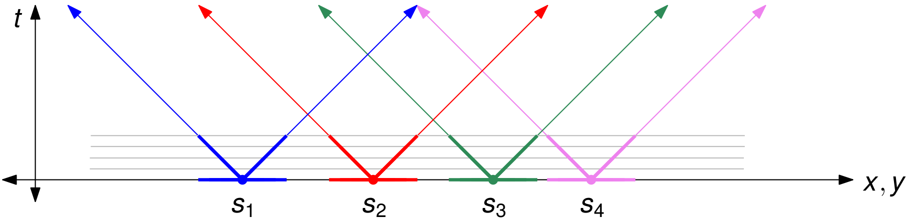
Edelsbrunner&Seidel (1986)

1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.
2. Every cross-section of a site's pyramid corresponds to a wavefront of the site: A point $p \in \mathbb{R}^3$ with coordinates (x, y, t) lies on the pyramid of site s if the point $p_{xy} \in \mathbb{R}^2$ with coordinates (x, y) is at weighted distance t from s.



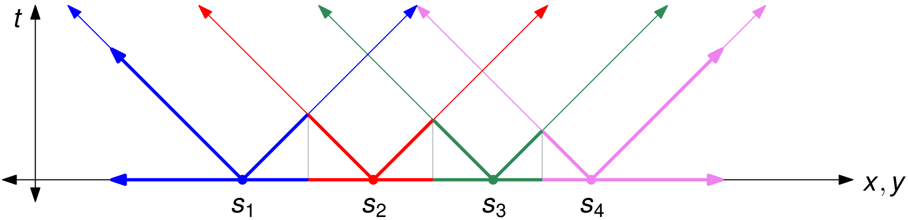
Edelsbrunner&Seidel (1986)

1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.
2. Every cross-section of a site's pyramid corresponds to a wavefront of the site: A point $p \in \mathbb{R}^3$ with coordinates (x, y, t) lies on the pyramid of site s if the point $p_{xy} \in \mathbb{R}^2$ with coordinates (x, y) is at weighted distance t from s.

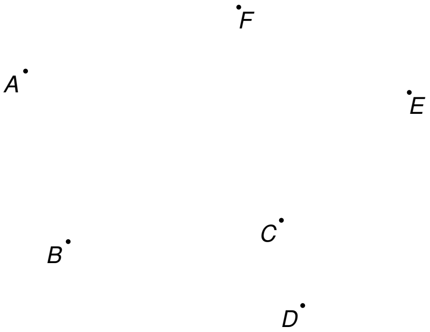


Edelsbrunner&Seidel (1986)

1. For each site: Construct (in $\mathbb{R}^3$) one upside-down, infinitely tall, right pyramid whose apex coincides with the site's location.
2. Every cross-section of a site's pyramid corresponds to a wavefront of the site: A point $p \in \mathbb{R}^3$ with coordinates (x, y, t) lies on the pyramid of site s if the point $p_{xy} \in \mathbb{R}^2$ with coordinates (x, y) is at weighted distance t from s.
3. Then the minimization diagram of all pyramids matches the Voronoi diagram.

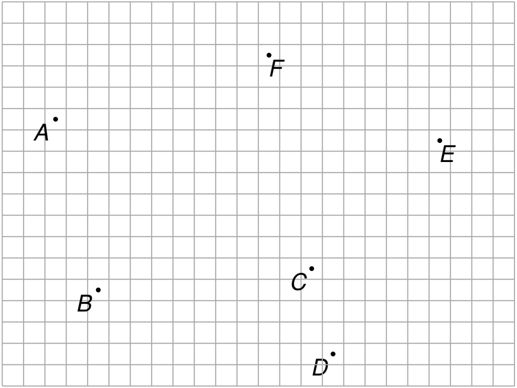


- For a input points given



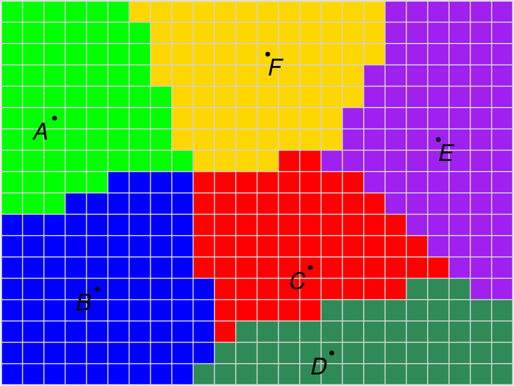
Discrete Voronoi Diagram

- For a input points given, a regular grid is constructed over a super-set of their bounding box.



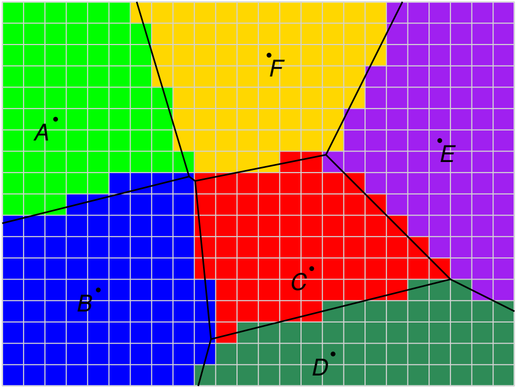
Discrete Voronoi Diagram

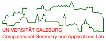
- For a input points given, a regular grid is constructed over a super-set of their bounding box.
- Then discrete Voronoi regions are determined by deciding on a cell-by-cell basis which input point is closest.



Discrete Voronoi Diagram

- For a input points given, a regular grid is constructed over a super-set of their bounding box.
- Then discrete Voronoi regions are determined by deciding on a cell-by-cell basis which input point is closest.
- The Voronoi diagram is extracted from the grid.





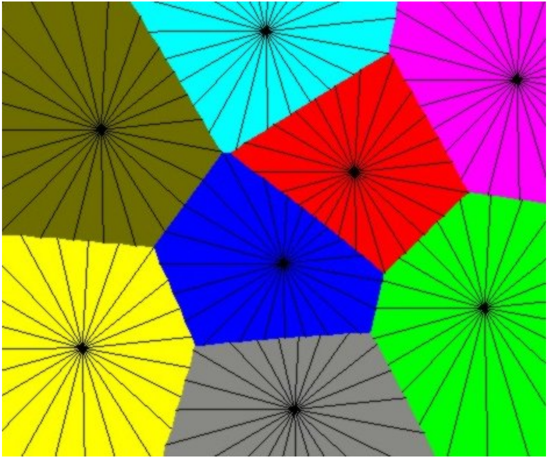
Approximate Voronoi Diagram by Means of Graphics Hardware

- Regard $\mathbb{R}^2$ as the *xy*-plane of $\mathbb{R}^3$, and construct upright circular unit cones at every point of S. (All cones point upwards, are of the same size and form the same angle with the *xy*-plane!) Assign a unique color to every cone.

Approximate Voronoi Diagram by Means of Graphics Hardware

Hoff et al. (1999)

Look at the cones from below the *xy*-plane, and use graphics hardware to render them. This yields a subdivision of the *xy*-plane into approximate Voronoi regions.



5 Voronoi Diagrams of Points

- Definition and Properties
- Algorithms
- Generalizations
  - Additively-Weighted Voronoi Diagram
  - Multiplicatively-Weighted Voronoi Diagram
  - L_1 and L_∞ Voronoi Diagram
  - Power Diagram
  - Higher-Order Voronoi Diagram
  - Farthest-Point Voronoi Diagram
  - Centroidal Voronoi Diagram
- Applications

Generalizations of Voronoi Diagrams

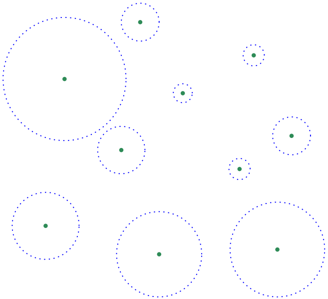
- The definition of a Voronoi region allows generalizations in three different directions.

$$\mathcal{VR}(p, S) := \{q \in \mathbb{R}^2 : d(q, p) \leq d(q, S)\}$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

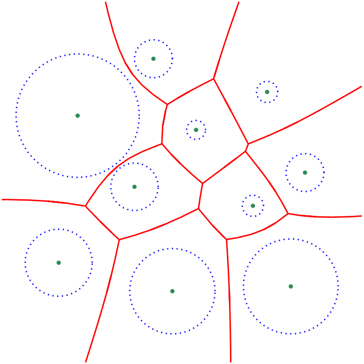
Additively-Weighted Voronoi Diagram

- We define the distance of a point q to a site p_i as $d(q, p_i) - w_i$, where $d(\cdot, \cdot)$ denotes the standard Euclidean distance and where w_i is non-negative.



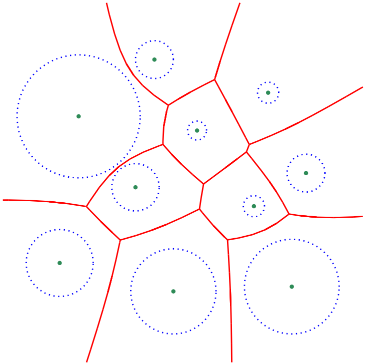
Additively-Weighted Voronoi Diagram

- We define the distance of a point q to a site p_i as $d(q, p_i) - w_i$, where $d(\cdot, \cdot)$ denotes the standard Euclidean distance and where w_i is non-negative.
- The resulting diagram is called *Appollonius diagram* or *additively-weighted VD*.



Additively-Weighted Voronoi Diagram

- We define the distance of a point q to a site p_i as $d(q, p_i) - w_i$, where $d(\cdot, \cdot)$ denotes the standard Euclidean distance and where w_i is non-negative.
- The resulting diagram is called *Appollonius diagram* or *additively-weighted VD*.
- It can be seen as the Voronoi diagram of circles with radii w_i; all its edges are hyperbolic arcs (and straight-line segments).



Multiplicatively-Weighted Voronoi Diagram of Points

Weighted Prairie fire

- Unweighted: Each wavefront propagates at the same speed.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

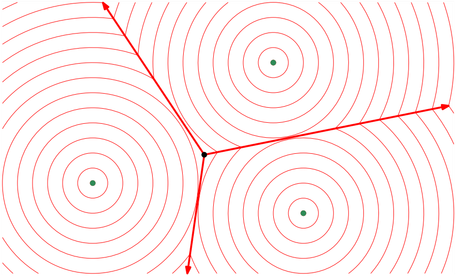
Multiplicatively-Weighted Voronoi Diagram of Points

Weighted Prairie fire

- Unweighted: Each wavefront propagates at the same speed.

Weighted Prairie fire

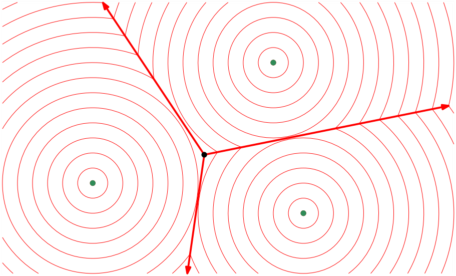
- Unweighted: Each wavefront propagates at the same speed.
- Weighted: The speed of a wavefront is proportional to the weight of the fire site.



Multiplicatively-Weighted Voronoi Diagram of Points

Weighted Prairie fire

- Unweighted: Each wavefront propagates at the same speed.
- Weighted: The speed of a wavefront is proportional to the weight of the fire site.

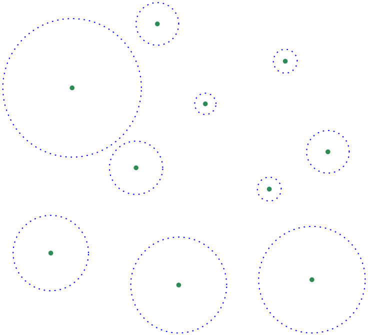


Multiplicatively-Weighted Voronoi Diagram

- We define the weighted distance of a point q to a site p as

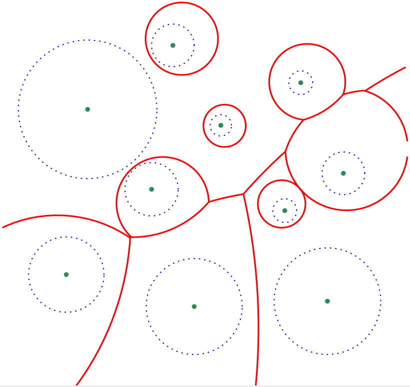
$$d_w(p, q) := \frac{d(p, q)}{w(p)},$$

where $d(\cdot, \cdot)$ denotes the standard Euclidean distance and where $w(p) \in \mathbb{R}^+$.



Multiplicatively-Weighted Voronoi Diagram

- We define the weighted distance of a point q to a site p as
$$d_w(p,q) := \frac{d(p,q)}{w(p)},$$
where $d(\cdot,\cdot)$ denotes the standard Euclidean distance and where $w(p) \in \mathbb{R}^+$.
- The resulting diagram is called the *multiplicatively-weighted Voronoi diagram*.

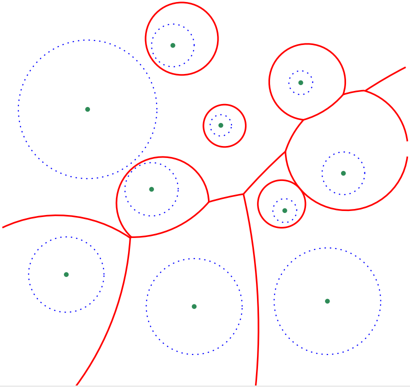


Multiplicatively-Weighted Voronoi Diagram

- We define the weighted distance of a point q to a site p as

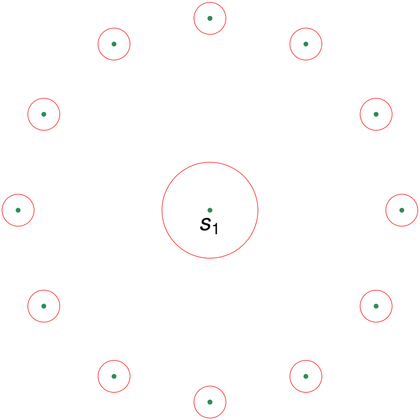
$$d_w(p, q) := \frac{d(p, q)}{w(p)},$$

where $d(\cdot, \cdot)$ denotes the standard Euclidean distance and where $w(p) \in \mathbb{R}^+$.
- The resulting diagram is called the *multiplicatively-weighted Voronoi diagram*.
- All its edges are circular arcs (and straight-line segments).



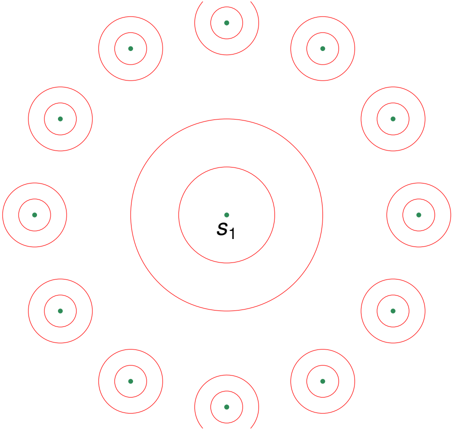
Multiplicatively-Weighted Voronoi Diagram of Points

- Note that the Voronoi regions of (higher-weighted) sites may be disconnected.



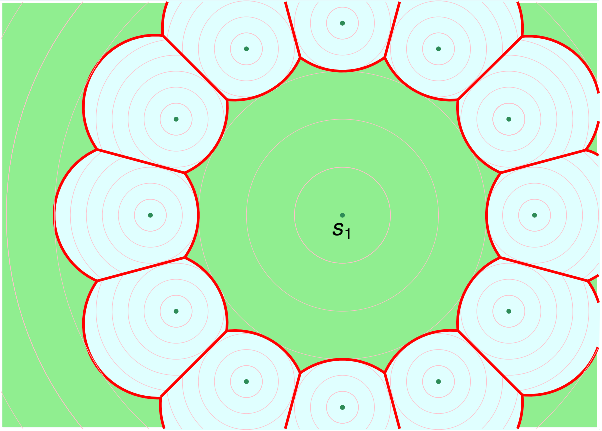
Multiplicatively-Weighted Voronoi Diagram of Points

- Note that the Voronoi regions of (higher-weighted) sites may be disconnected.



Multiplicatively-Weighted Voronoi Diagram of Points

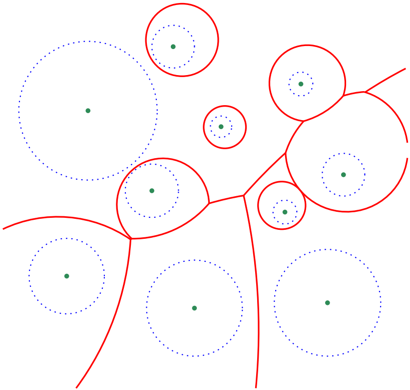
- Note that the Voronoi regions of (higher-weighted) sites may be disconnected.



Multiplicatively-Weighted Voronoi Diagram of Points

Aurenhammer&Edelsbrunner (1984)

The multiplicatively-weighted VD of n points is computed in $\Theta(n^2)$ time.



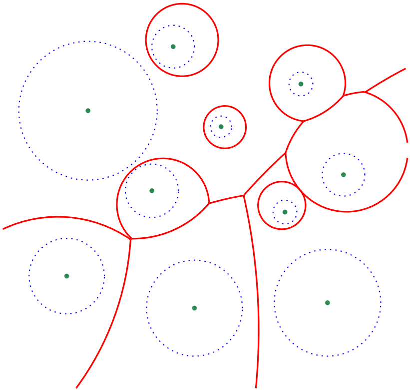
Multiplicatively-Weighted Voronoi Diagram of Points

Aurenhammer&Edelsbrunner (1984)

The multiplicatively-weighted VD of n points is computed in $\Theta(n^2)$ time.

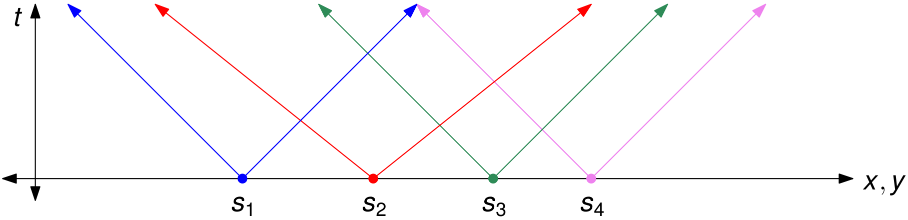
Held&de Lorenzo (2020)

The multiplicatively-weighted VD of n points is computed in $O(n \log^4 n)$ expected time.



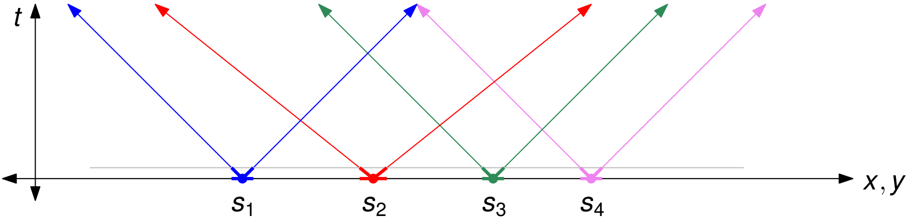
Multiplicatively-Weighted Voronoi Diagram as Minimization Diagram

- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.

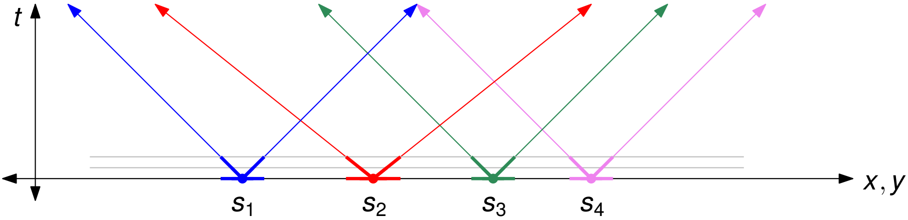


Multiplicatively-Weighted Voronoi Diagram as Minimization Diagram

- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.

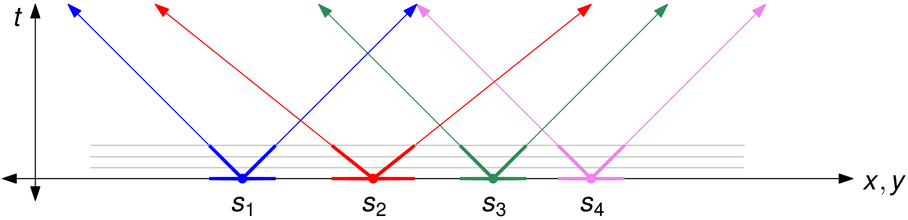


- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.



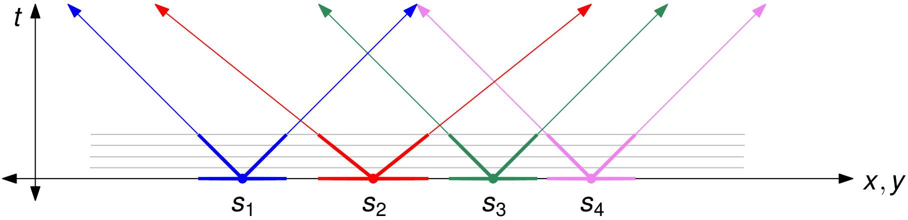
Multiplicatively-Weighted Voronoi Diagram as Minimization Diagram

- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.



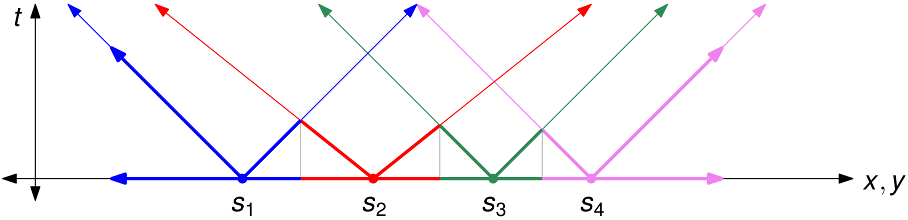
Multiplicatively-Weighted Voronoi Diagram as Minimization Diagram

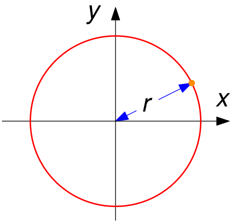
- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.



Multiplicatively-Weighted Voronoi Diagram as Minimization Diagram

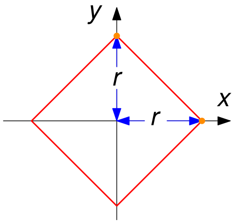
- The approach by Edelsbrunner&Seidel (1986) is also applicable to multiplicatively-weighted Voronoi diagrams: The inclinations of the lateral surfaces are chosen indirectly proportional to the weights of the points.



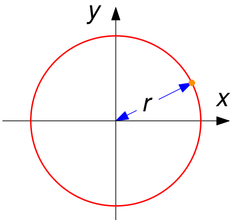


$$L_2 : \quad \sqrt{x^2 + y^2} = r$$

L_1 and L_∞ Voronoi Diagram

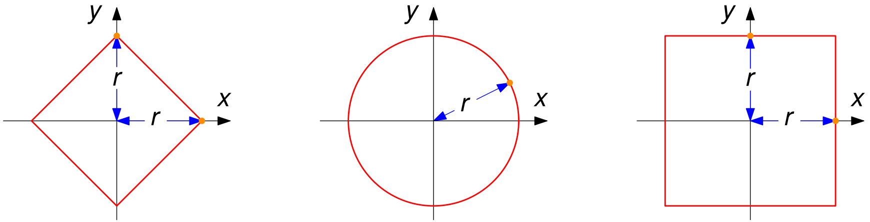


$L_1: \quad |x| + |y| = r$



$L_2: \quad \sqrt{x^2 + y^2} = r$

L_1 and L_∞ Voronoi Diagram

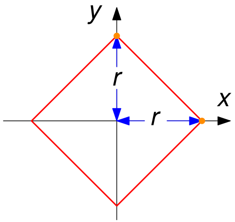


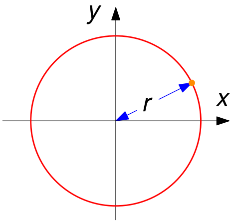
$L_1: \quad |x| + |y| = r$

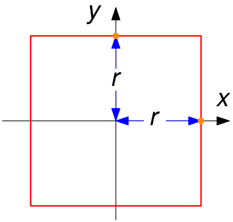
$L_2: \quad \sqrt{x^2 + y^2} = r$

$L_\infty: \quad \max\{|x|, |y|\} = r$

L_1 and L_∞ Voronoi Diagram



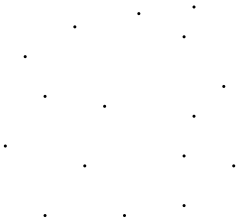


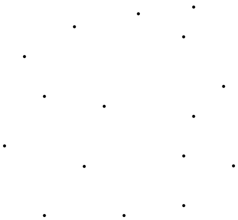


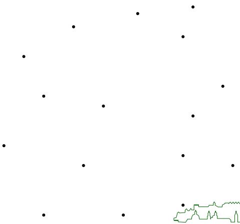
$L_1: \quad |x| + |y| = r$

$L_2: \quad \sqrt{x^2 + y^2} = r$

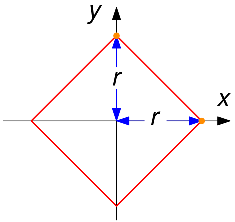
$L_\infty: \quad \max\{|x|, |y|\} = r$



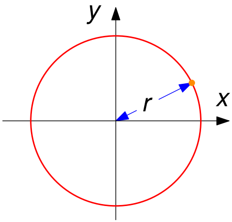




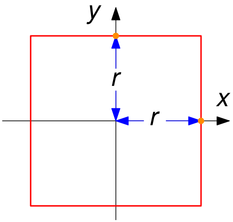
L_1 and L_∞ Voronoi Diagram



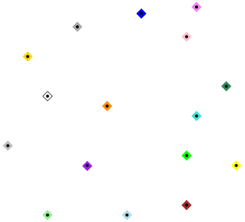
L_1 : $|x| + |y| = r$

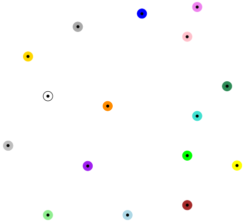


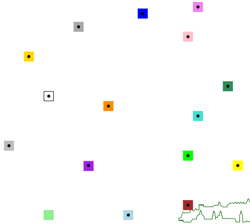
L_2 : $\sqrt{x^2 + y^2} = r$



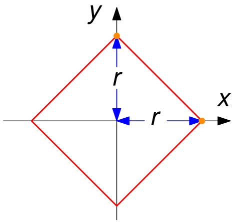
L_∞ : $\max\{|x|, |y|\} = r$

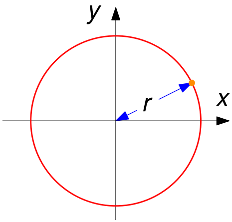


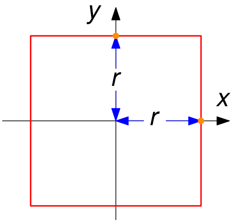




UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab



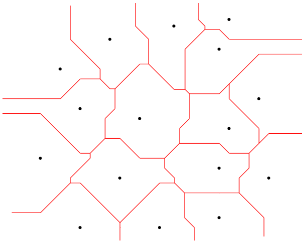


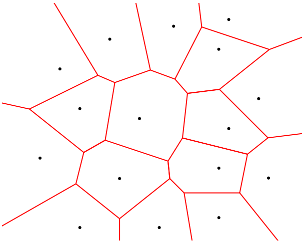


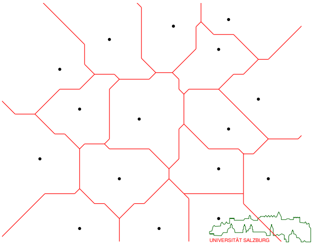
$L_1: \quad |x| + |y| = r$

$L_2: \quad \sqrt{x^2 + y^2} = r$

$L_\infty: \quad \max\{|x|, |y|\} = r$





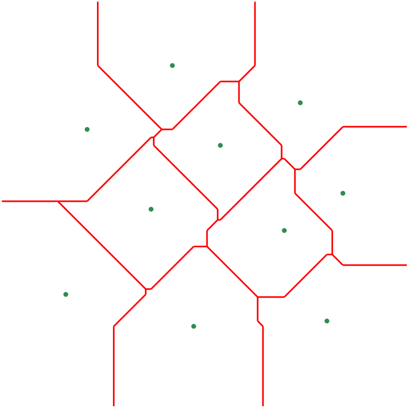


L_1 Voronoi Diagram

L_1 Voronoi Diagram

Eder&Held (2019)

The combinatorial complexity of the VD of n multiplicatively-weighted points in the L_1 norm has a $\Theta(n^2)$ worst-case bound. All its bisectors are polygonal curves of constant complexity. It can be computed by an incremental algorithm in $O(n^2 \log n)$ time.

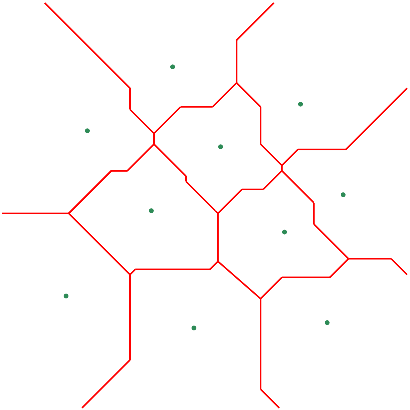


L_∞ Voronoi Diagram

L_∞ Voronoi Diagram

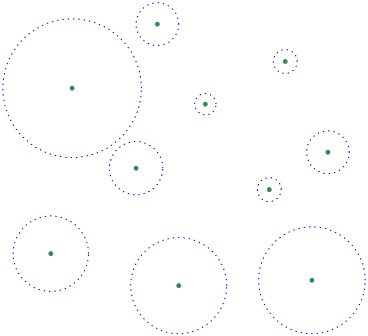
Eder&Held (2019)

The combinatorial complexity of the Voronoi diagram of n multiplicatively-weighted points, axis-aligned rectangular boxes and straight-line segments in the L_∞ norm has a tight $\Theta(n^2)$ worst-case bound. All its bisectors are polygonal curves of constant complexity. It can be computed by an incremental algorithm in $O(n^2\alpha(n)\log n)$ time.



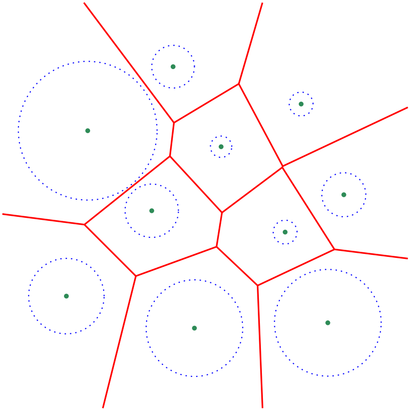
Power Diagram

- We are again given a set of sites p_i with non-negative weights w_i. Then the power of a point q from p_i is defined as $d(q, p_i)^2 - w_i^2$.



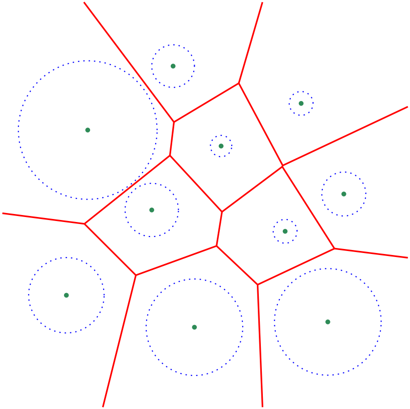
Power Diagram

- We are again given a set of sites p_i with non-negative weights w_i. Then the power of a point q from p_i is defined as $d(q, p_i)^2 - w_i^2$.
- The resulting diagram is called *power diagram*.



Power Diagram

- We are again given a set of sites p_i with non-negative weights w_i. Then the power of a point q from p_i is defined as $d(q, p_i)^2 - w_i^2$.
- The resulting diagram is called *power diagram*.
- All its edges are straight-line segments.

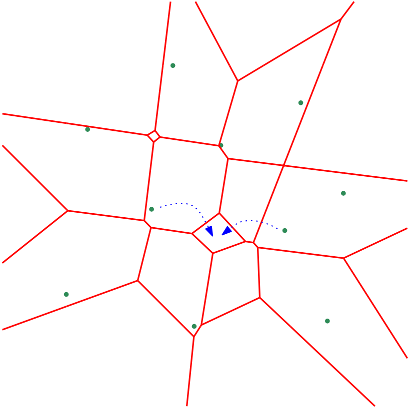


Higher-Order Voronoi Diagram

- We are given a set S of n points $p_1, p_2, \ldots, p_n$, and consider the standard Euclidean distance.

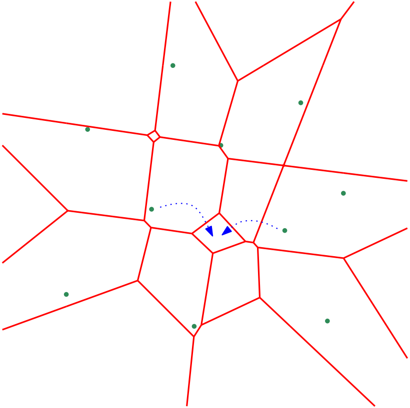
Higher-Order Voronoi Diagram

- We are given a set S of n points $p_1, p_2, \ldots, p_n$, and consider the standard Euclidean distance.
- The *second-order Voronoi diagram* is a partition of the plane such that each Voronoi region is the locus of points closer to two distinct sites p_i, p_j than to any other site of S.



Higher-Order Voronoi Diagram

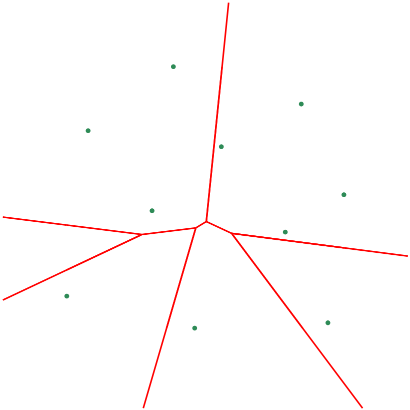
- We are given a set S of n points $p_1, p_2, \ldots, p_n$, and consider the standard Euclidean distance.
- The *second-order Voronoi diagram* is a partition of the plane such that each Voronoi region is the locus of points closer to two distinct sites p_i, p_j than to any other site of S. Similar for higher orders.



Farthest-Point Voronoi Diagram

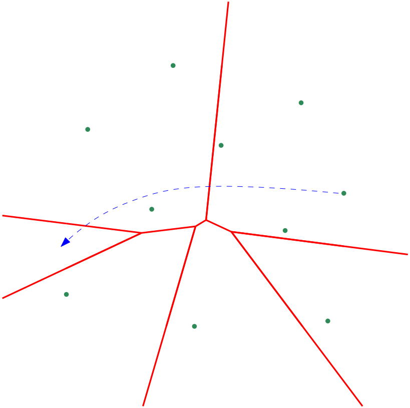
- Again we are given a set S of n points $p_1, p_2, \ldots, p_n$.

- Again we are given a set S of n points $p_1, p_2, \ldots, p_n$.
- The *farthest-point Voronoi diagram* is a partition of the plane such that each Voronoi region is the locus of points which have the same point of S as the farthest point.



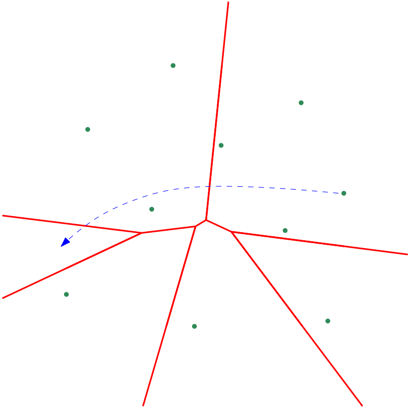
Farthest-Point Voronoi Diagram

- Again we are given a set S of n points $p_1, p_2, \ldots, p_n$.
- The *farthest-point Voronoi diagram* is a partition of the plane such that each Voronoi region is the locus of points which have the same point of S as the farthest point. The farthest-point VD is the $(n-1)$-st order VD. A point p_i of S has a region in the farthest-point VD if and only if p_i is a vertex of $CH(S)$.



Farthest-Point Voronoi Diagram

- Again we are given a set S of n points $p_1, p_2, \dots, p_n$.
- The *farthest-point Voronoi diagram* is a partition of the plane such that each Voronoi region is the locus of points which have the same point of S as the farthest point. The farthest-point VD is the $(n-1)$-st order VD. A point p_i of S has a region in the farthest-point VD if and only if p_i is a vertex of $CH(S)$.
- It can be computed in $O(n \log n)$ time.



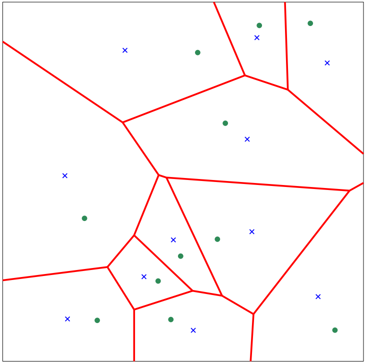
Definition 109 (Centroidal Voronoi Diagram (CVD))

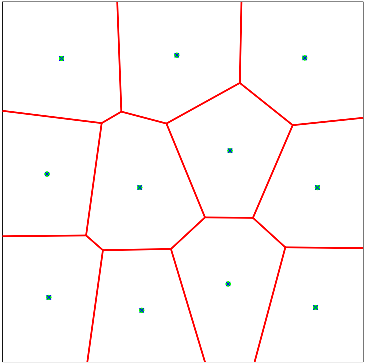
A Voronoi diagram of a set of points is called *centroidal* if the points are also centroids of the Voronoi regions, i.e., centers of mass with respect to a given density function.

Centroidal Voronoi Diagram

Definition 109 (Centroidal Voronoi Diagram (CVD))

A Voronoi diagram of a set of points is called *centroidal* if the points are also centroids of the Voronoi regions, i.e., centers of mass with respect to a given density function.

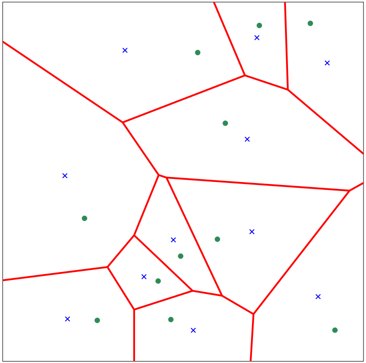


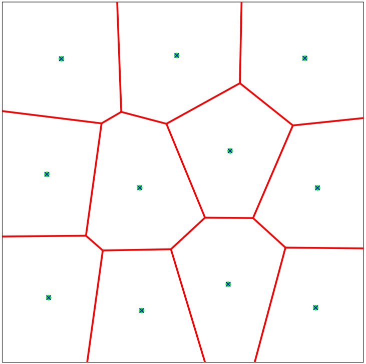


Centroidal Voronoi Diagram

Definition 109 (Centroidal Voronoi Diagram (CVD))

A Voronoi diagram of a set of points is called *centroidal* if the points are also centroids of the Voronoi regions, i.e., centers of mass with respect to a given density function.





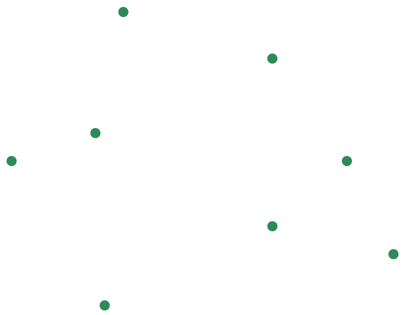
- Applications of CVDs: data compression, image segmentation, mesh generation, modeling of territorial behavior of animals, etc.

5 Voronoi Diagrams of Points

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

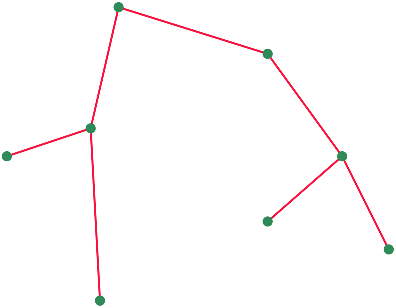
Euclidean Minimum Spanning Tree

- Consider a set $S := \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ of n points, and assume that we want to compute a Euclidean minimum spanning tree (EMST) of S.



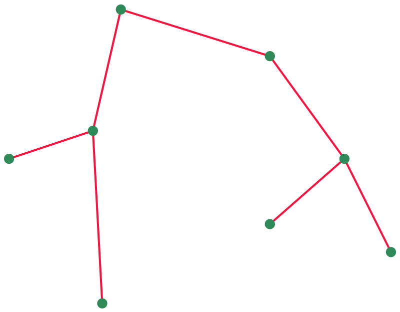
Euclidean Minimum Spanning Tree

- Consider a set $S := \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ of n points, and assume that we want to compute a Euclidean minimum spanning tree (EMST) of S.



Euclidean Minimum Spanning Tree

- Consider a set $S := \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ of n points, and assume that we want to compute a Euclidean minimum spanning tree (EMST) of S.



- Note: An EMST is unique if all inter-point distances on S are distinct.

Euclidean Minimum Spanning Tree

- Obviously, we could apply standard techniques of graph theory by computing an EMST on the weighted graph $\mathcal{G} := (V, E)$, where $V := S$ and $E := S \times S$, and where the Euclidean length of an edge is taken as its weight.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Euclidean Minimum Spanning Tree

- Obviously, we could apply standard techniques of graph theory by computing an EMST on the weighted graph $\mathcal{G} := (V, E)$, where $V := S$ and $E := S \times S$, and where the Euclidean length of an edge is taken as its weight.

Lemma 110 (Jarnik (1930), Prim (1957), Dijkstra (1959))

Assume that $\mathcal{G}$ is connected, and let V_1, V_2 be a partition of V. There is a minimum spanning tree of $\mathcal{G}$ which contains the shortest of the edges with one terminal in V_1 and the other in V_2.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Obviously, we could apply standard techniques of graph theory by computing an EMST on the weighted graph $\mathcal{G} := (V, E)$, where $V := S$ and $E := S \times S$, and where the Euclidean length of an edge is taken as its weight.

Lemma 110 (Jarnik (1930), Prim (1957), Dijkstra (1959))

Assume that $\mathcal{G}$ is connected, and let V_1, V_2 be a partition of V. There is a minimum spanning tree of $\mathcal{G}$ which contains the shortest of the edges with one terminal in V_1 and the other in V_2.

- *Prim's algorithm* starts with a small tree $\mathcal{T}$ and grows it until it contains all nodes of $\mathcal{G}$. Initially, $\mathcal{T}$ contains just one arbitrary node of V. At each stage one node not yet in $\mathcal{T}$ but closest to (a node of) $\mathcal{T}$ is added to $\mathcal{T}$. Prim's algorithm can be implemented to run in $O(|V|^2)$ time.

Euclidean Minimum Spanning Tree

- Obviously, we could apply standard techniques of graph theory by computing an EMST on the weighted graph $\mathcal{G} := (V, E)$, where $V := S$ and $E := S \times S$, and where the Euclidean length of an edge is taken as its weight.

Lemma 110 (Jarnik (1930), Prim (1957), Dijkstra (1959))

Assume that $\mathcal{G}$ is connected, and let V_1, V_2 be a partition of V. There is a minimum spanning tree of $\mathcal{G}$ which contains the shortest of the edges with one terminal in V_1 and the other in V_2.

- *Prim's algorithm* starts with a small tree $\mathcal{T}$ and grows it until it contains all nodes of $\mathcal{G}$. Initially, $\mathcal{T}$ contains just one arbitrary node of V. At each stage one node not yet in $\mathcal{T}$ but closest to (a node of) $\mathcal{T}$ is added to $\mathcal{T}$. Prim's algorithm can be implemented to run in $O(|V|^2)$ time.
- *Kruskal's algorithm* begins with a spanning forest, where each forest is initialized with one node of V. It repeatedly joins two trees together by picking the shortest edge between them until a spanning tree of the entire graph is obtained. Kruskal's algorithm can be implemented to run in $O(|E| \log |E|)$ time.

- Obviously, we could apply standard techniques of graph theory by computing an EMST on the weighted graph $\mathcal{G} := (V, E)$, where $V := S$ and $E := S \times S$, and where the Euclidean length of an edge is taken as its weight.

Lemma 110 (Jarnik (1930), Prim (1957), Dijkstra (1959))

Assume that $\mathcal{G}$ is connected, and let V_1, V_2 be a partition of V. There is a minimum spanning tree of $\mathcal{G}$ which contains the shortest of the edges with one terminal in V_1 and the other in V_2.

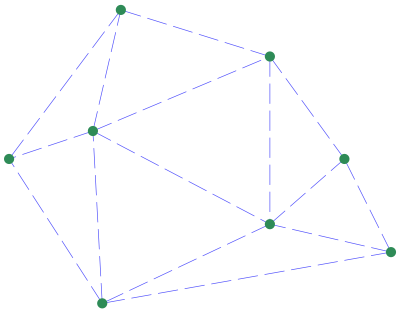
- *Prim's algorithm* starts with a small tree $\mathcal{T}$ and grows it until it contains all nodes of $\mathcal{G}$. Initially, $\mathcal{T}$ contains just one arbitrary node of V. At each stage one node not yet in $\mathcal{T}$ but closest to (a node of) $\mathcal{T}$ is added to $\mathcal{T}$. Prim's algorithm can be implemented to run in $O(|V|^2)$ time.
- *Kruskal's algorithm* begins with a spanning forest, where each forest is initialized with one node of V. It repeatedly joins two trees together by picking the shortest edge between them until a spanning tree of the entire graph is obtained. Kruskal's algorithm can be implemented to run in $O(|E| \log |E|)$ time.
- Can we do any better than $O(n^2)$ when computing EMSTs?

Euclidean Minimum Spanning Tree

- Can we do any better than $O(n^2)$ when computing EMSTs?

Lemma 111

The EMST of a set S of points is a sub-graph of $\mathcal{DT}(S)$.

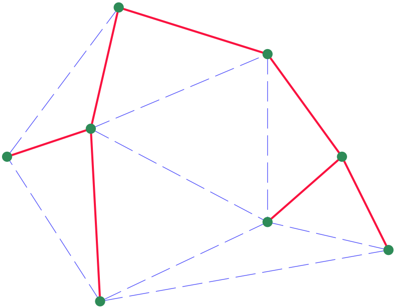


Euclidean Minimum Spanning Tree

- Can we do any better than $O(n^2)$ when computing EMSTs?

Lemma 111

The EMST of a set S of points is a sub-graph of $\mathcal{DT}(S)$.



Euclidean Minimum Spanning Tree

- Thus, there is no need to consider the complete graph, K_n, on S.
- Rather, we can apply Kruskal's algorithm to $\mathcal{DT}(S)$, and obtain an $O(n \log n)$ algorithm for computing EMSTs.

Euclidean Minimum Spanning Tree

- Thus, there is no need to consider the complete graph, K_n, on S.
- Rather, we can apply Kruskal's algorithm to $\mathcal{DT}(S)$, and obtain an $O(n \log n)$ algorithm for computing EMSTs.

Lemma 112

An EMST of S can be computed in time $O(n \log n)$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Euclidean Minimum Spanning Tree

- Thus, there is no need to consider the complete graph, K_n, on S.
- Rather, we can apply Kruskal's algorithm to $\mathcal{DT}(S)$, and obtain an $O(n \log n)$ algorithm for computing EMSTs.

Lemma 112

An EMST of S can be computed in time $O(n \log n)$.

Theorem 113

An EMST of S can be computed from the Delaunay triangulation of S in time $O(n)$.

Euclidean Minimum Spanning Tree

- Thus, there is no need to consider the complete graph, K_n, on S.
- Rather, we can apply Kruskal's algorithm to $\mathcal{DT}(S)$, and obtain an $O(n \log n)$ algorithm for computing EMSTs.

Lemma 112

An EMST of S can be computed in time $O(n \log n)$.

Theorem 113

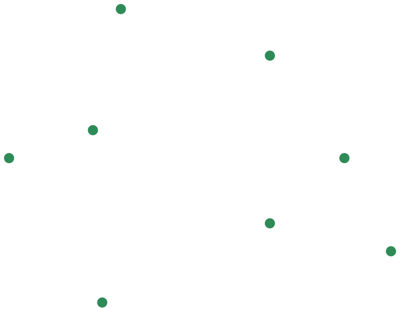
An EMST of S can be computed from the Delaunay triangulation of S in time $O(n)$.

Sketch of Proof: Observe that $\mathcal{DT}(S)$ is a planar graph, and use Cheriton and Tarjan's "clean-up refinement" of Kruskal's algorithm. $\square$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Approximate Traveling Salesman Tour

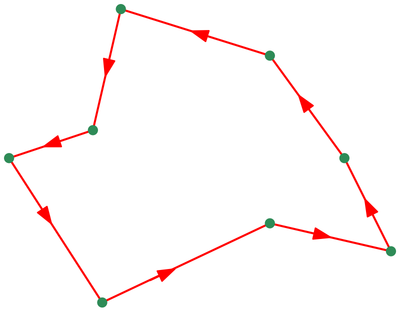
- The EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP) asks to compute a shortest closed path on $S \subset \mathbb{R}^2$ that visits all n points of S.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Approximate Traveling Salesman Tour

- The EUCLIDEANTRAVELINGSALESMANPROBLEM (ETSP) asks to compute a shortest closed path on $S \subset \mathbb{R}^2$ that visits all n points of S.



Approximate Traveling Salesman Tour

Theorem 114

TSP is $\mathcal{NP}$-complete, and ETSP is $\mathcal{NP}$-hard.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 114

TSP is $\mathcal{NP}$-complete, and ETSP is $\mathcal{NP}$-hard.

- Intuitively, ETSP ought to be $\mathcal{NP}$-complete, too.
- And, indeed, the $\mathcal{NP}$-completeness of ETSP is claimed in many publications . . .

Theorem 114

TSP is $\mathcal{NP}$-complete, and ETSP is $\mathcal{NP}$-hard.

- Intuitively, ETSP ought to be $\mathcal{NP}$-complete, too.
- And, indeed, the $\mathcal{NP}$-completeness of ETSP is claimed in many publications ...
- However, this claim is wrong! (The title of [Papadimitriou (1977)] is misleading!) ETSP, and several other optimization problems involving Euclidean distance, are not known to be in $\mathcal{NP}$ due to a "technical twist": For ETSP, the length of a tour on n points is a sum of n square roots. Comparing this sum to a number c may require very high precision, and no polynomial-time algorithm is known for solving this problem. (E.g., repeated squaring of n square roots may lead to numbers that need 2^n bits to store.)
- Open problem: Can the sum of n square roots of integers be compared to another integer in polynomial time?

Approximate Solution for Euclidean Traveling Salesman Problem

- Let *OPT* be the true length of a TSP tour, and let *APX* be the length of an approximate solution.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Let *OPT* be the true length of a TSP tour, and let *APX* be the length of an approximate solution.

Definition 115 (Constant-factor approximation)

An approximation algorithm provides a *constant-factor approximation* (for TSP) if a constant $c \in \mathbb{R}^+$ exists such that $APX \leq c \cdot OPT$ holds for all inputs.

Approximate Solution for Euclidean Traveling Salesman Problem

- Let *OPT* be the true length of a TSP tour, and let *APX* be the length of an approximate solution.

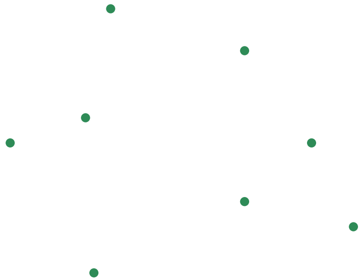
Definition 115 (Constant-factor approximation)

An approximation algorithm provides a *constant-factor approximation* (for TSP) if a constant $c \in \mathbb{R}^+$ exists such that $APX \leq c \cdot OPT$ holds for all inputs.

- Simple constant-factor approximations to ETSP:
  - Doubling-the-EMST heuristic: $c = 2$; runs in $O(n \log n)$ time.
  - Christofides' heuristic [1976]: $c = 3/2$; runs in $O(n^3)$ time.

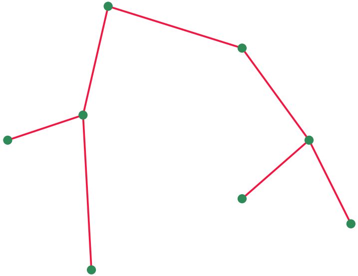
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Approximate ETSP: Doubling-the-EMST Heuristic



Approximate ETSP: Doubling-the-EMST Heuristic

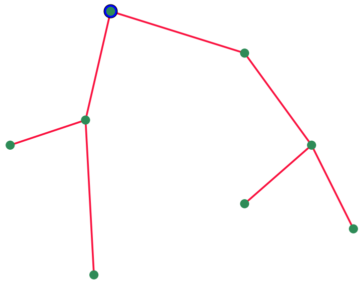
1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

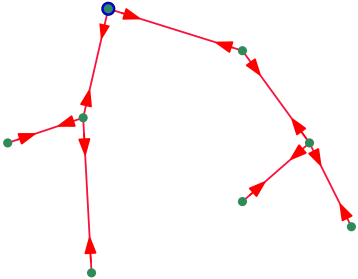
Approximate ETSP: Doubling-the-EMST Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Select an arbitrary node v of $\mathcal{T}(S)$ as root.



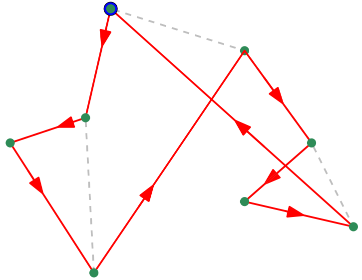
Approximate ETSP: Doubling-the-EMST Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Select an arbitrary node v of $\mathcal{T}(S)$ as root.
3. Compute a (pre-order-like) traversal of $\mathcal{T}(S)$ rooted at v to obtain a tour $\mathcal{C}(S)$.

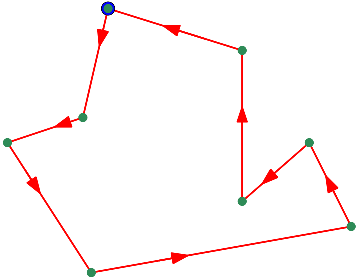


Approximate ETSP: Doubling-the-EMST Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Select an arbitrary node v of $\mathcal{T}(S)$ as root.
3. Compute a (pre-order-like) traversal of $\mathcal{T}(S)$ rooted at v to obtain a tour $\mathcal{C}(S)$.
4. By-pass points already visited, thus shortening $\mathcal{C}(S)$.



1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Select an arbitrary node v of $\mathcal{T}(S)$ as root.
3. Compute a (pre-order-like) traversal of $\mathcal{T}(S)$ rooted at v to obtain a tour $\mathcal{C}(S)$.
4. By-pass points already visited, thus shortening $\mathcal{C}(S)$.
5. Apply 2-opt moves (at additional computational cost).



Approximate ETSP: Doubling-the-EMST Heuristic

- Time complexity: $O(n \log n)$ for computing the EMST $\mathcal{T}(S)$.
- Factor of approximation: $c = 2$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

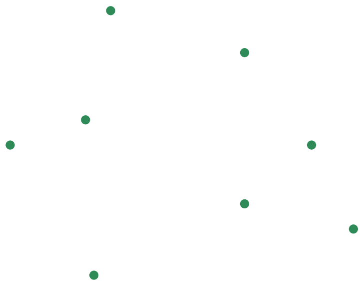
Approximate ETSP: Doubling-the-EMST Heuristic

- Time complexity: $O(n \log n)$ for computing the EMST $\mathcal{T}(S)$.
- Factor of approximation: $c = 2$.

Theorem 116

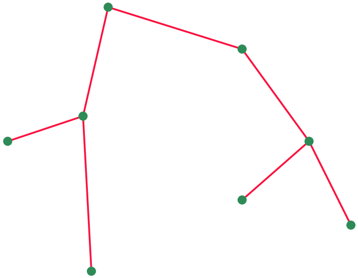
The doubling-the-EMST heuristic computes a tour on n points within $O(n \log n)$ time that is at most 100% longer than the shortest tour.

Approximate ETSP: Christofides' Heuristic

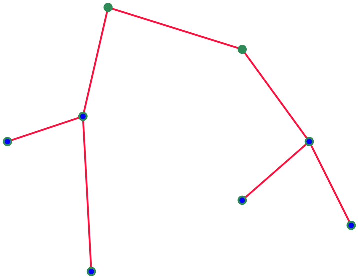


Approximate ETSP: Christofides' Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.

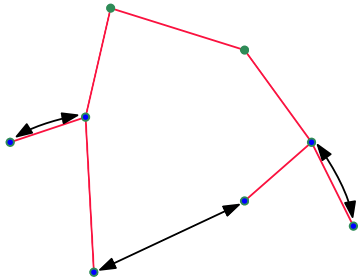


1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Get a minimum Euclidean matching $\mathcal{M}$ on the vertices of odd degree in $\mathcal{T}(S)$.

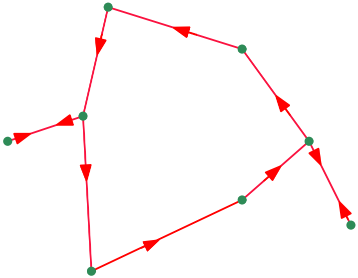


Approximate ETSP: Christofides' Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Get a minimum Euclidean matching $\mathcal{M}$ on the vertices of odd degree in $\mathcal{T}(S)$.

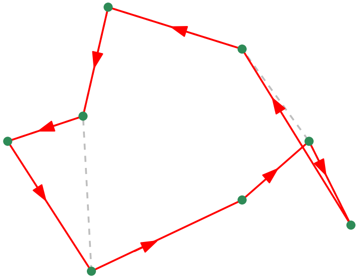


1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Get a minimum Euclidean matching $\mathcal{M}$ on the vertices of odd degree in $\mathcal{T}(S)$.
3. Compute an Eulerian tour $\mathcal{C}$ on $\mathcal{T} \cup \mathcal{M}$.



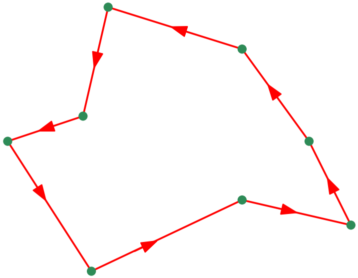
Approximate ETSP: Christofides' Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Get a minimum Euclidean matching $\mathcal{M}$ on the vertices of odd degree in $\mathcal{T}(S)$.
3. Compute an Eulerian tour $\mathcal{C}$ on $\mathcal{T} \cup \mathcal{M}$.
4. By-pass points already visited, thus shortening $\mathcal{C}$.



Approximate ETSP: Christofides' Heuristic

1. Compute the Euclidean minimum spanning tree $\mathcal{T}(S)$ of S.
2. Get a minimum Euclidean matching $\mathcal{M}$ on the vertices of odd degree in $\mathcal{T}(S)$.
3. Compute an Eulerian tour $\mathcal{C}$ on $\mathcal{T} \cup \mathcal{M}$.
4. By-pass points already visited, thus shortening $\mathcal{C}$.
5. Apply 2-opt moves (at additional computational cost).



Approximate ETSP: Christofides' Heuristic

- Time complexity: $O(n^3)$ for computing the Euclidean matching.
- Factor of approximation: $c = \frac{3}{2}$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Approximate ETSP: Christofides' Heuristic

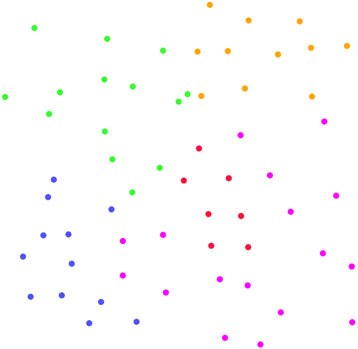
- Time complexity: $O(n^3)$ for computing the Euclidean matching.
- Factor of approximation: $c = \frac{3}{2}$.

Theorem 117

Christofides' heuristic computes a tour on n points within $O(n^3)$ time that is at most 50% longer than the shortest tour.

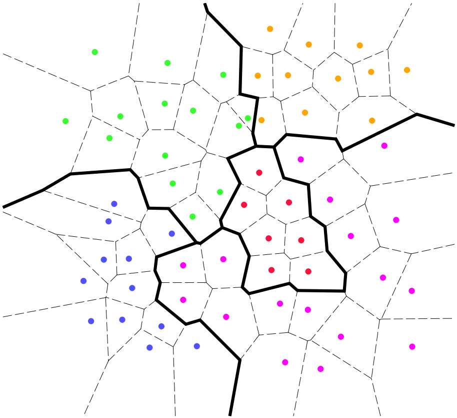
Statistical Classification and Shape Estimation

- Given are sets of differently colored points in the plane. What is a suitable partition of the plane according to the colors of the points?



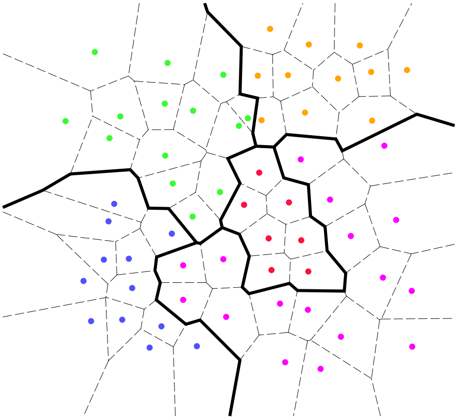
Statistical Classification and Shape Estimation

- Given are sets of differently colored points in the plane. What is a suitable partition of the plane according to the colors of the points?
- Well-known idea: Compute the Voronoi diagram and color every Voronoi region with its point's color.



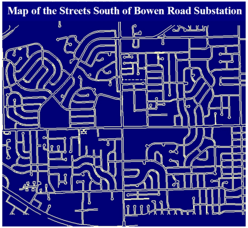
Estimating Electrical Distribution Boundaries

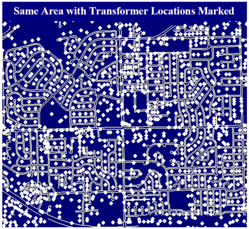
- TXU Energy (Dallas, TX, USA):
  - Which area is serviced by a particular electric device?
  - How can we display (feeder-level) statistical information within a geographic context?

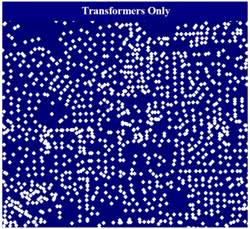


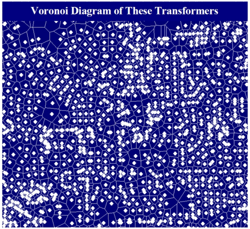
Estimating Electrical Distribution Boundaries

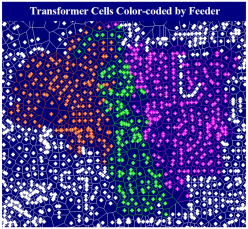
- [Held&Williamson (2004)] generate distribution boundaries as boundaries of unions of Voronoi regions of basic devices (e.g., transformers) and integrate them into TXU's geographic information system.







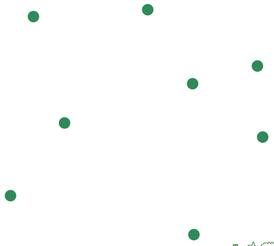




Natural-Neighbor Interpolation

Problem: INTERPOLATION

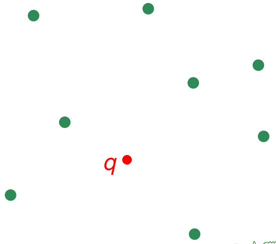
Given: A set S of $m + 1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) “data” $v_0, v_1, \ldots, v_m$,



Natural-Neighbor Interpolation

Problem: INTERPOLATION

Given: A set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$.

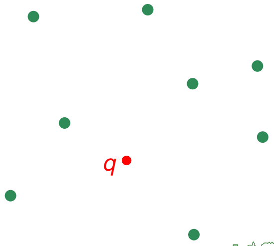


Natural-Neighbor Interpolation

Problem: INTERPOLATION

Given: A set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) “data” $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$.

Compute: An estimate $f(q)$ of the data at q, obtained by interpolation of $v_0, v_1, \ldots, v_m$.



Natural-Neighbor Interpolation

Problem: INTERPOLATION

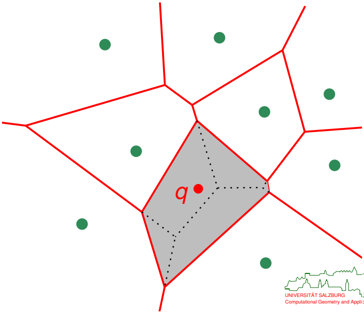
Given: A set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$.

Compute: An estimate $f(q)$ of the data at q, obtained by interpolation of $v_0, v_1, \ldots, v_m$.

Natural-neighbor interpolation (NNI)

[Sibson (1981)]: Use ratios of Voronoi areas as weights $\lambda_i(q)$ in the interpolation:

$$f(q) := \sum_{i=0}^{m} v_i \cdot \lambda_i(q).$$



Natural-Neighbor Interpolation

Problem: INTERPOLATION

Given: A set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$.

Compute: An estimate $f(q)$ of the data at q, obtained by interpolation of $v_0, v_1, \ldots, v_m$.

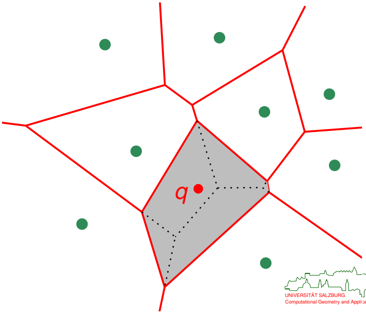
Natural-neighbor interpolation (NNI)

[Sibson (1981)]: Use ratios of Voronoi areas as weights $\lambda_i(q)$ in the interpolation:

$$f(q) := \sum_{i=0}^{m} v_i \cdot \lambda_i(q).$$

Natural-neighbor extrapolation

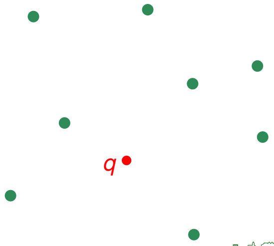
[Bobach et al. (2009)]: NNI outside of convex hull.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 118 (NNI)

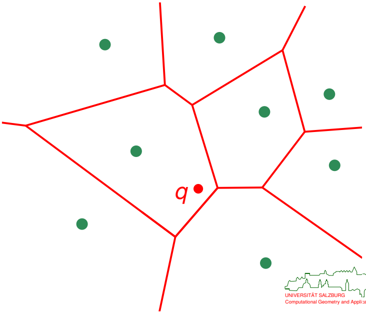
Consider a set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$. Let $S' := S \cup \{q\}$.



Natural-Neighbor Interpolation

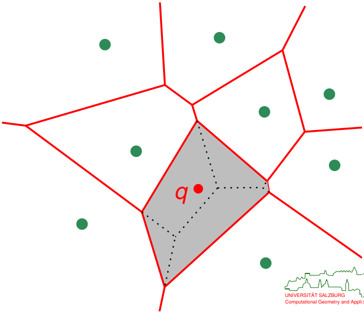
Definition 118 (NNI)

Consider a set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$. Let $S' := S \cup \{q\}$.



Definition 118 (NNI)

Consider a set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$. Let $S' := S \cup \{q\}$.



Definition 118 (NNI)

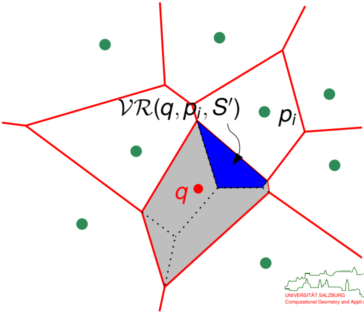
Consider a set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$. Let $S' := S \cup \{q\}$. Then

$$f(q) := \sum_{i=0}^{m} v_i \cdot \lambda_i(q)$$

gives the interpolated data for q obtained by *natural-neighbor interpolation* (NNI), with

$$\lambda_i(q) := \frac{|\mathcal{VR}(q, p_i, S')|}{|\mathcal{VR}(q, S')|},$$

where $|\mathcal{VR}(q, S')|$ denotes the area of the Voronoi region of q within S', and $|\mathcal{VR}(q, p_i, S')|$ corresponds to the area of the second-order Voronoi region of q and p_i within S'.



Definition 118 (NNI)

Consider a set S of $m+1$ sites $p_0, p_1, \ldots, p_m \in \mathbb{R}^2$ with associated (scalar or vector-valued) "data" $v_0, v_1, \ldots, v_m$, and $q \in CH(S)$. Let $S' := S \cup \{q\}$. Then

$$f(q) := \sum_{i=0}^{m} v_i \cdot \lambda_i(q)$$

gives the interpolated data for q obtained by *natural-neighbor interpolation* (NNI), with

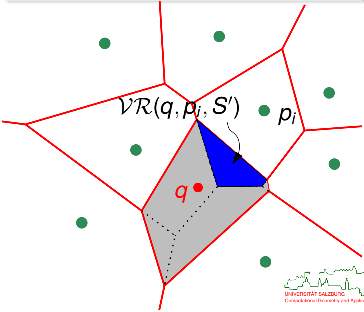
$$\lambda_i(q) := \frac{|\mathcal{VR}(q, p_i, S')|}{|\mathcal{VR}(q, S')|},$$

where $|\mathcal{VR}(q, S')|$ denotes the area of the Voronoi region of q within S', and $|\mathcal{VR}(q, p_i, S')|$ corresponds to the area of the second-order Voronoi region of q and p_i within S'.

Theorem 119

Sibson's NNI interpolant is

- C^0 if $q \in S$,
- C^1 if q lies on a Delaunay circle of S, and
- C^∞ otherwise.



Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Laser sintering is a manufacturing process used in rapid prototyping:
  - A laser is used to manufacture a part by sintering powder-based materials layer by layer.
  - Small-series production is possible.
  - Snap fits and living hinges can be produced.

Images courtesy of EOS GmbH

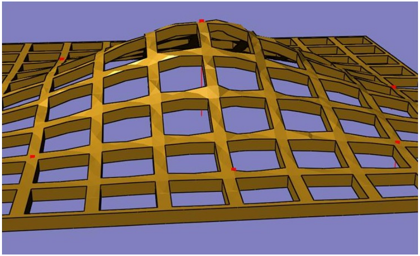
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Major problem:
  - The laser-induced heating and subsequent cooling down of the material may cause the “warpage” phenomenon.
  - Warpage is the result of a change in the morphology of the molten powder: amorphous to part-crystalline.
  - Crystalline regions have a higher density than the amorphous regions, leading to a loss of volume.
  - Different layers undergo different loss in volume, leading to inter-layer tension.

Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Major problem:
  - The laser-induced heating and subsequent cooling down of the material may cause the “warpage” phenomenon.
  - Warpage is the result of a change in the morphology of the molten powder: amorphous to part-crystalline.
  - Crystalline regions have a higher density than the amorphous regions, leading to a loss of volume.
  - Different layers undergo different loss in volume, leading to inter-layer tension.
  - This tension may result in a bimetallic effect: “curl”.



Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Bold idea: Apply a pre-deformation in order to manufacture an inversely deformed part!

Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Bold idea: Apply a pre-deformation in order to manufacture an inversely deformed part!
- It seems natural to use interpolation — but how shall we interpolate vectors on the surface of a polyhedron?

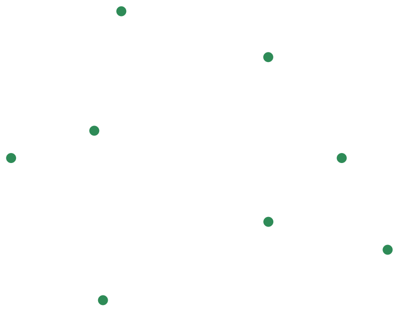
Improved Laser Sintering Based on Natural-Neighbor Interpolation

- Bold idea: Apply a pre-deformation in order to manufacture an inversely deformed part!
- It seems natural to use interpolation — but how shall we interpolate vectors on the surface of a polyhedron?
- [Held&Pfligersdorffer (2009)]: Pre-deformation by means of approximate natural-neighbor interpolation (NNI) helps to reduce warpage by 90%.

Improved Laser Sintering Based on Natural-Neighbor Interpolation

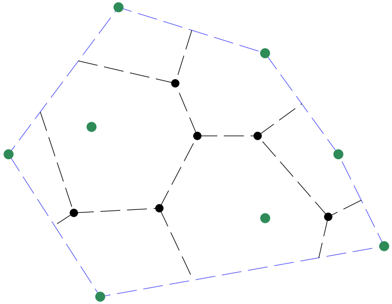
- Bold idea: Apply a pre-deformation in order to manufacture an inversely deformed part!
- It seems natural to use interpolation — but how shall we interpolate vectors on the surface of a polyhedron?
- [Held&Pfligersdorffer (2009)]: Pre-deformation by means of approximate natural-neighbor interpolation (NNI) helps to reduce warpage by 90%.
- Pre-deformation works neatly for reasonably triangulated parts and a reasonable number of deformation vectors.

Maximum Empty Circle



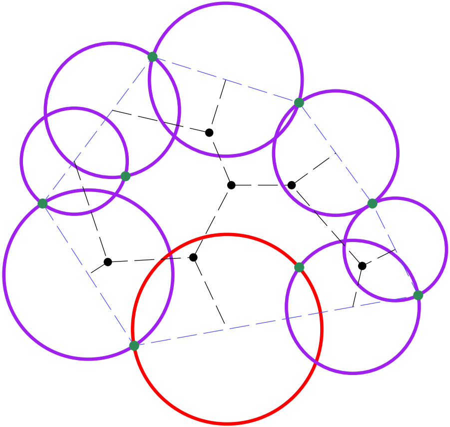
Maximum Empty Circle

1. Restrict $\mathcal{VD}(S)$ to $CH(S)$.



Maximum Empty Circle

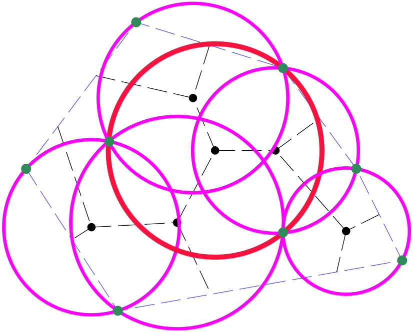
1. Restrict $\mathcal{VD}(S)$ to $CH(S)$.
2. Determine the largest circle centered at an intersection of $\mathcal{VD}(S)$ and $CH(S)$.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

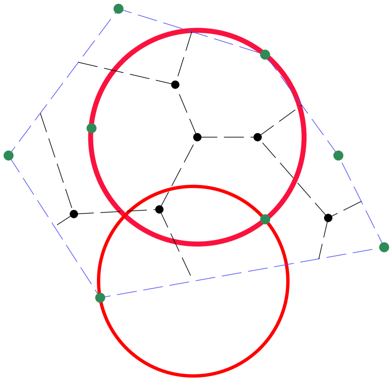
Maximum Empty Circle

1. Restrict $\mathcal{VD}(S)$ to $CH(S)$.
2. Determine the largest circle centered at an intersection of $\mathcal{VD}(S)$ and $CH(S)$.
3. Determine the largest circle centered at an interior node of $\mathcal{VD}(S)$.



Maximum Empty Circle

1. Restrict $\mathcal{VD}(S)$ to $CH(S)$.
2. Determine the largest circle centered at an intersection of $\mathcal{VD}(S)$ and $CH(S)$.
3. Determine the largest circle centered at an interior node of $\mathcal{VD}(S)$.
4. Pick the largest circle among those two categories of circles.



Definition 120 (Hausdorff distance)

Let A, B be two non-empty subsets of $\mathbb{R}^d$. The *directed Hausdorff distance*, $\mathrm{h}(A, B)$, from A to B (relative to the standard Euclidean distance $d(.,.)$) is defined as

$$\mathrm{h}(A, B) := \sup_{a \in A} \left(\inf_{b \in B} d(a, b) \right).$$

Definition 120 (Hausdorff distance)

Let A, B be two non-empty subsets of $\mathbb{R}^d$. The *directed Hausdorff distance*, $\mathrm{h}(A, B)$, from A to B (relative to the standard Euclidean distance $d(.,.)$) is defined as

$$\mathrm{h}(A, B) := \sup_{a \in A} \left(\inf_{b \in B} d(a, b) \right).$$

The *(symmetric) Hausdorff distance*, $\mathrm{H}(A, B)$, between A and B is defined as

$$\mathrm{H}(A, B) := \max \left\{ \mathrm{h}(A, B), \mathrm{h}(B, A) \right\}.$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 120 (Hausdorff distance)

Let A, B be two non-empty subsets of $\mathbb{R}^d$. The *directed Hausdorff distance*, $\mathrm{h}(A, B)$, from A to B (relative to the standard Euclidean distance $d(.,.)$) is defined as

$$\mathrm{h}(A, B) := \sup_{a \in A} \left(\inf_{b \in B} d(a, b) \right).$$

The *(symmetric) Hausdorff distance*, $\mathrm{H}(A, B)$, between A and B is defined as

$$\mathrm{H}(A, B) := \max \left\{ \mathrm{h}(A, B), \mathrm{h}(B, A) \right\}.$$

- Introduced by Felix Hausdorff in 1914.
- If both A and B are bounded then $\mathrm{H}(A, B)$ is guaranteed to be finite.
- For compact sets we can replace $\inf$ by $\min$ and $\sup$ by $\max$.

Definition 120 (Hausdorff distance)

Let A, B be two non-empty subsets of $\mathbb{R}^d$. The *directed Hausdorff distance*, $\mathrm{h}(A, B)$, from A to B (relative to the standard Euclidean distance $d(.,.)$) is defined as

$$\mathrm{h}(A, B) := \sup_{a \in A} \left(\inf_{b \in B} d(a, b) \right).$$

The *(symmetric) Hausdorff distance*, $\mathrm{H}(A, B)$, between A and B is defined as

$$\mathrm{H}(A, B) := \max \left\{ \mathrm{h}(A, B), \mathrm{h}(B, A) \right\}.$$

- Introduced by Felix Hausdorff in 1914.
- If both A and B are bounded then $\mathrm{H}(A, B)$ is guaranteed to be finite.
- For compact sets we can replace $\inf$ by $\min$ and $\sup$ by $\max$.

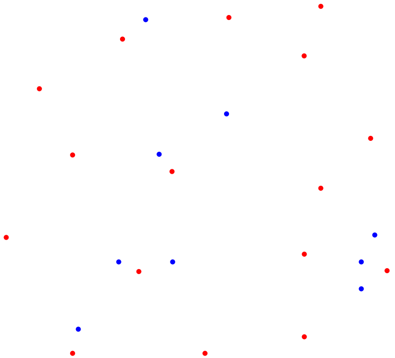
Theorem 121

The Hausdorff distance between two finite sets S_1, S_2 of points in $\mathbb{R}^2$ can be computed in $O(n \log n)$ time, where $n := \max\{|S_1|, |S_2|\}$.

Hausdorff Distance

Sketch of Proof of Theorem 121 :

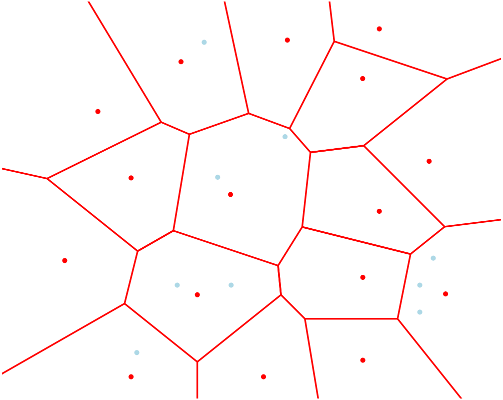
1. Consider sets S_1 and S_2 of blue and red points.



Sketch of Proof of Theorem 121 :

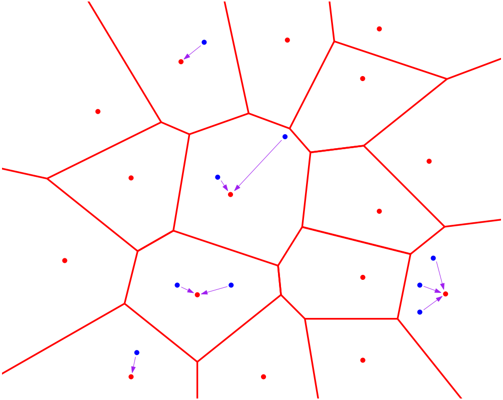
1. Consider sets S_1 and S_2 of blue and red points.
2. Compute $\mathcal{VD}(S_2)$.

□



Sketch of Proof of Theorem 121 :

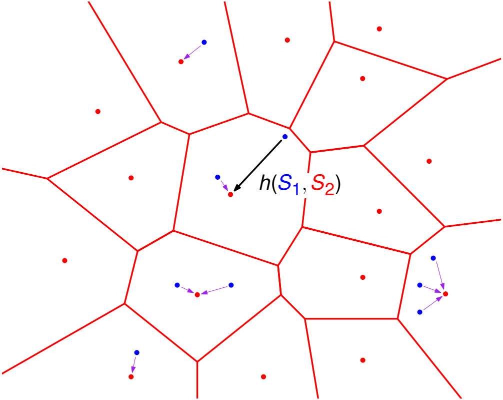
1. Consider sets S_1 and S_2 of blue and red points.
2. Compute $\mathcal{VD}(S_2)$.
3. Locate each point of S_1 within the Voronoi regions of $\mathcal{VD}(S_2)$. □



Sketch of Proof of Theorem 121 :

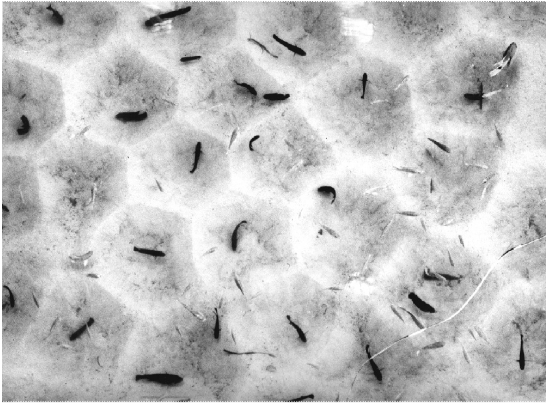
1. Consider sets S_1 and S_2 of blue and red points.
2. Compute $\mathcal{VD}(S_2)$.
3. Locate each point of S_1 within the Voronoi regions of $\mathcal{VD}(S_2)$.
4. The maximum distance yields $h(S_1, S_2)$.

□



Centroidal Voronoi Diagrams and Territorial Behavior

- Tilapia mossambica (Dt.: Weißkehl-Buntbarsch):
  - The male fishes dig nesting pits into sandy grounds.
  - The centers and corners of the pits are adjusted until the final configuration resembles a centroidal Voronoi diagram.



[Image credit: G. Barlow, “Hexagonal Territories”, Animal Behavior 22:876–878, 1974.]

Evaporation as a Massively Parallel Algorithm?

- Salar de Atacama in the Chilean Andes: 3 000 km^2, average elevation about 2 300 m asl., 3 500 milliliters annual evaporation, only a few milliliters of annual rainfall.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

6 Skeletal Structures

- Voronoi Diagram of Points, Line Segments and Circular Arcs
- Straight Skeleton
- Applications

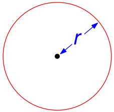
6 Skeletal Structures

- Voronoi Diagram of Points, Line Segments and Circular Arcs
  - Definitions and Properties
  - State of the Art
  - Randomized Incremental Construction
- Straight Skeleton
- Applications

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

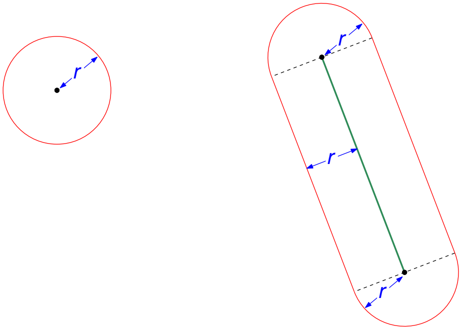
Generalizing the Wavefront

- The wavefront of a point is a circle of radius r, for some non-negative value of r.

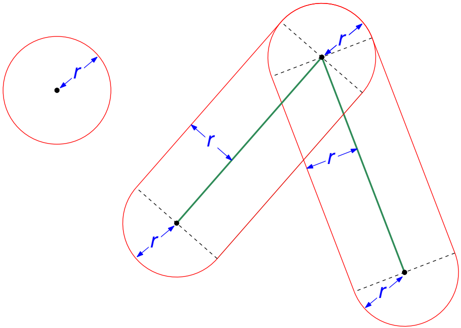


Generalizing the Wavefront

- The wavefront of a point is a circle of radius r, for some non-negative value of r.
- The wavefront of a straight-line segment is a box with semi-circular caps.

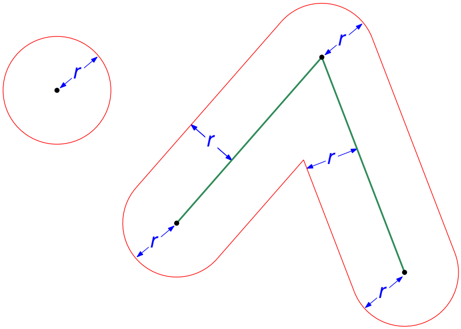


- The wavefront of a point is a circle of radius r, for some non-negative value of r.
- The wavefront of a straight-line segment is a box with semi-circular caps.
- Of course, individual portions of the wavefront may interact again.



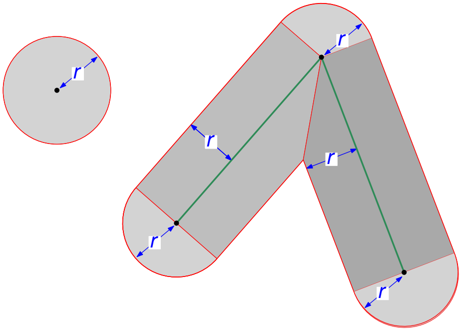
Generalizing the Wavefront

- The wavefront of a point is a circle of radius r, for some non-negative value of r.
- The wavefront of a straight-line segment is a box with semi-circular caps.
- Of course, individual portions of the wavefront may interact again.



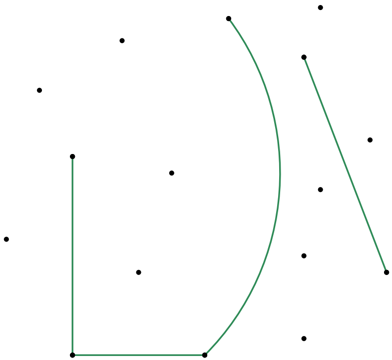
Generalizing the Wavefront

- The wavefront of a point is a circle of radius r, for some non-negative value of r.
- The wavefront of a straight-line segment is a box with semi-circular caps.
- Of course, individual portions of the wavefront may interact again.
- It is natural to split up the wavefront into parts according to the input items that emitted them.



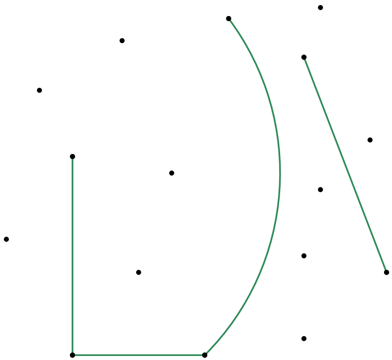
Voronoi Diagram of Points, Line Segments and Arcs: Wavefront

- Consider a set S of n points, straight-line segments, and circular arcs (“sites”).



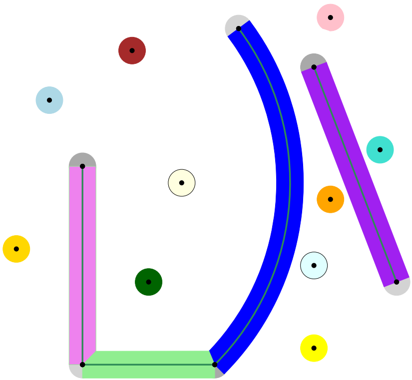
Voronoi Diagram of Points, Line Segments and Arcs: Wavefront

- Consider a set S of n points, straight-line segments, and circular arcs ("sites").
- For technical reasons we assume that all end-points of all segments and arcs are members of S. Furthermore, the segments and arcs are allowed to intersect only at common end-points. Such a set of sites is called "*admissible*".



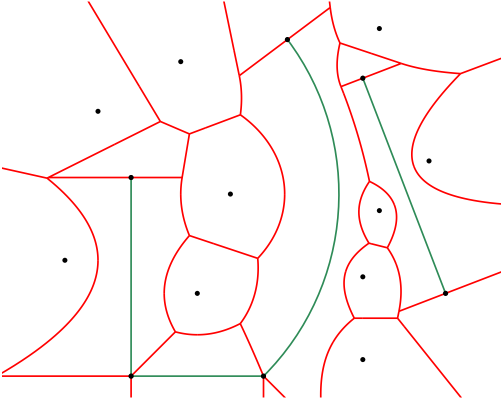
Voronoi Diagram of Points, Line Segments and Arcs: Wavefront

- Consider a set S of n points, straight-line segments, and circular arcs ("sites").
- For technical reasons we assume that all end-points of all segments and arcs are members of S. Furthermore, the segments and arcs are allowed to intersect only at common end-points. Such a set of sites is called "*admissible*".
- Now perform a (generalized) wavefront propagation.



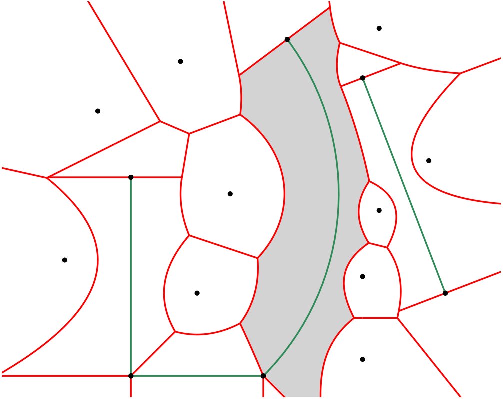
Voronoi Diagram of Points, Line Segments and Arcs

- Intuitively, the Voronoi diagram of S partitions the Euclidean plane into regions that are closer to one site than to any other.



Voronoi Diagram of Points, Line Segments and Arcs

- Intuitively, the Voronoi diagram of S partitions the Euclidean plane into regions that are closer to one site than to any other.
- Natural generalization of Voronoi diagrams of points, but Voronoi regions are now bounded by conics and need not be convex.

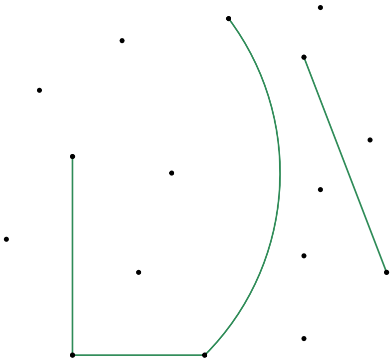


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Voronoi Diagram of Points, Line Segments and Arcs

Problem: GeneralizedVoronoiDiagram

Given: Admissible set S of points, line segments and circular arcs in 2D.

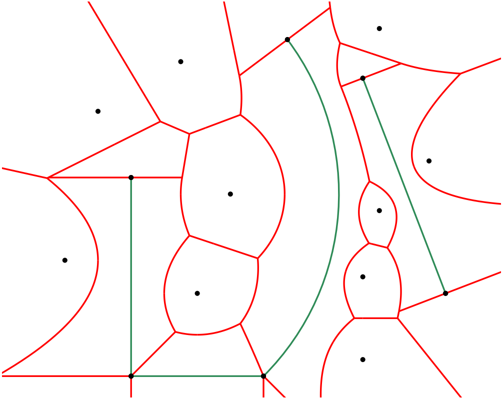


Voronoi Diagram of Points, Line Segments and Arcs

Problem: GENERALIZEDVORONOIDIAGRAM

Given: Admissible set S of points, line segments and circular arcs in 2D.

Compute: Voronoi diagram $\mathcal{VD}(S)$ under the Euclidean distance $d(\cdot, \cdot)$.

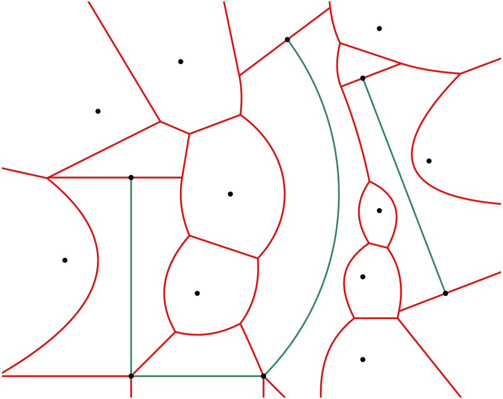


Problem: GENERALIZEDVORONOIDIAGRAM

Given: Admissible set S of points, line segments and circular arcs in 2D.

Compute: Voronoi diagram $\mathcal{VD}(S)$ under the Euclidean distance $d(\cdot,\cdot)$.

- Formal definition requires some technicalities . . . [Yap (1987), Held (1991)]



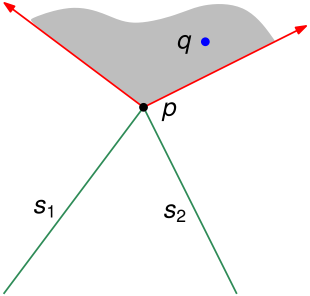
Voronoi Diagram of Points, Line Segments and Arcs: Technical Problem

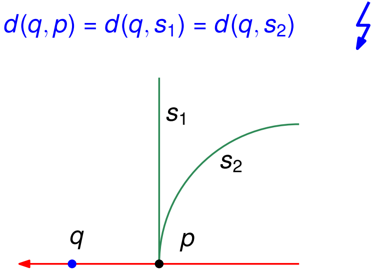
- Consider an admissible set S of n points, line segments and circular arcs as input sites in 2D, and two sites $s_1, s_2 \in S$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Voronoi Diagram of Points, Line Segments and Arcs: Technical Problem

- Consider an admissible set S of n points, line segments and circular arcs as input sites in 2D, and two sites $s_1, s_2 \in S$.
- Problem: We need to avoid "two-dimensional" and "non-intuitive" bisectors.

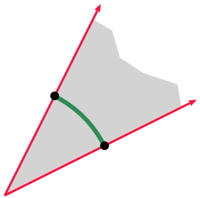




Definition 122 (Cone of influence)

The *cone of influence*, $\mathcal{CI}(s)$, of

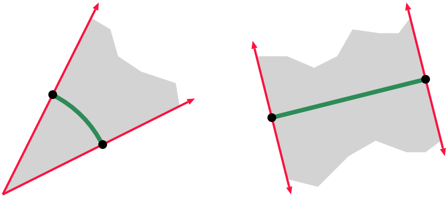
- a circular arc s is the closure of the cone bounded by the pair of rays originating in the arc's center and extending through its endpoints;



Definition 122 (Cone of influence)

The *cone of influence*, $\mathcal{CI}(s)$, of

- a circular arc s is the closure of the cone bounded by the pair of rays originating in the arc's center and extending through its endpoints;
- a straight-line segment s is the closure of the strip bounded by the normals through its endpoints;

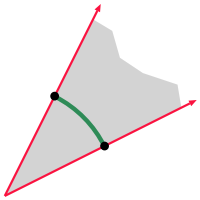
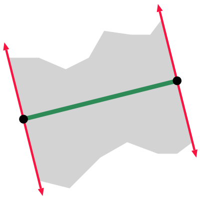


Voronoi Diagram of Points, Line Segments and Arcs: Definitions

Definition 122 (Cone of influence)

The *cone of influence*, $\mathcal{CI}(s)$, of

- a circular arc s is the closure of the cone bounded by the pair of rays originating in the arc's center and extending through its endpoints;
- a straight-line segment s is the closure of the strip bounded by the normals through its endpoints;
- a point s is the entire plane.

Definition 123 ((Generalized) Voronoi region)

The *(generalized) Voronoi region* of $s_i \in S$ relative to S is defined as

$$\mathcal{VR}(s_i, S) := \mathrm{cl}\{q \in \mathrm{int}\,\mathcal{CI}(s_i) : d(q, s_i) \leq d(q, S)\}.$$

Definition 123 ((Generalized) Voronoi region)

The *(generalized) Voronoi region* of $s_i \in S$ relative to S is defined as

$$\mathcal{VR}(s_i, S) := \mathrm{cl}\{q \in \mathrm{int}\, \mathcal{CI}(s_i) : d(q, s_i) \leq d(q, S)\}.$$

- It is common to drop the attribute “generalized” if the meaning is clear.

© M. Held (Univ. Salzburg)

Definition 123 ((Generalized) Voronoi region)

The *(generalized) Voronoi region* of $s_i \in S$ relative to S is defined as

$$\mathcal{VR}(s_i, S) := \mathrm{cl}\{q \in \mathrm{int}\,\mathcal{CI}(s_i) : d(q, s_i) \leq d(q, S)\}.$$

- It is common to drop the attribute “generalized” if the meaning is clear.

Definition 124 ((Generalized) Voronoi polygon)

The *(generalized) Voronoi polygon* of $s_i \in S$ relative to S is defined as

$$\mathcal{VP}(s_i, S) := \partial\,\mathcal{VR}(s_i, S).$$

Voronoi Diagram of Points, Line Segments and Arcs: Definitions

Definition 123 ((Generalized) Voronoi region)

The *(generalized) Voronoi region* of $s_i \in S$ relative to S is defined as

$$\mathcal{VR}(s_i, S) := \mathrm{cl}\{q \in \mathrm{int}\,\mathcal{CI}(s_i) : d(q, s_i) \le d(q, S)\}.$$

- It is common to drop the attribute "generalized" if the meaning is clear.

Definition 124 ((Generalized) Voronoi polygon)

The *(generalized) Voronoi polygon* of $s_i \in S$ relative to S is defined as

$$\mathcal{VP}(s_i, S) := \partial\,\mathcal{VR}(s_i, S).$$

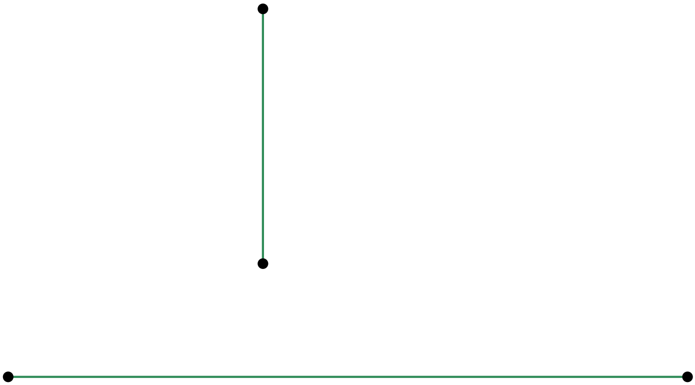
Definition 125 ((Generalized) Voronoi diagram)

The *(generalized) Voronoi diagram* of S is defined as

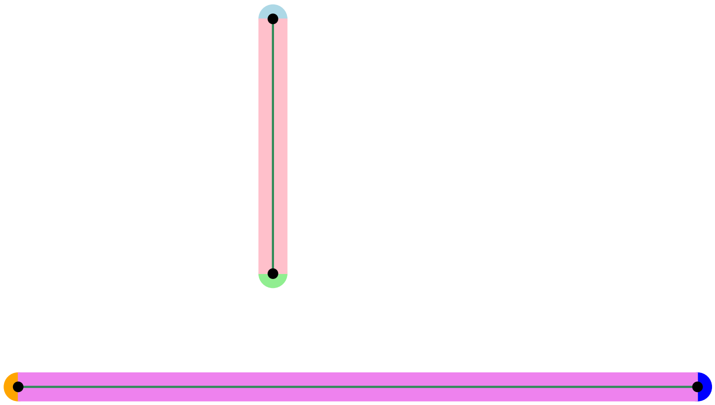
$$\mathcal{VD}(S) := \bigcup_{1 \le i \le n} \mathcal{VP}(s_i, S).$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

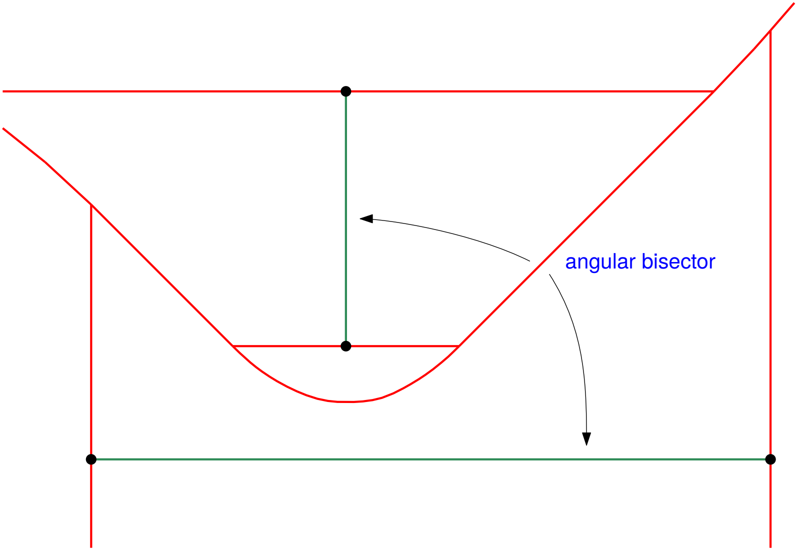
Sample Voronoi Diagram Dissected



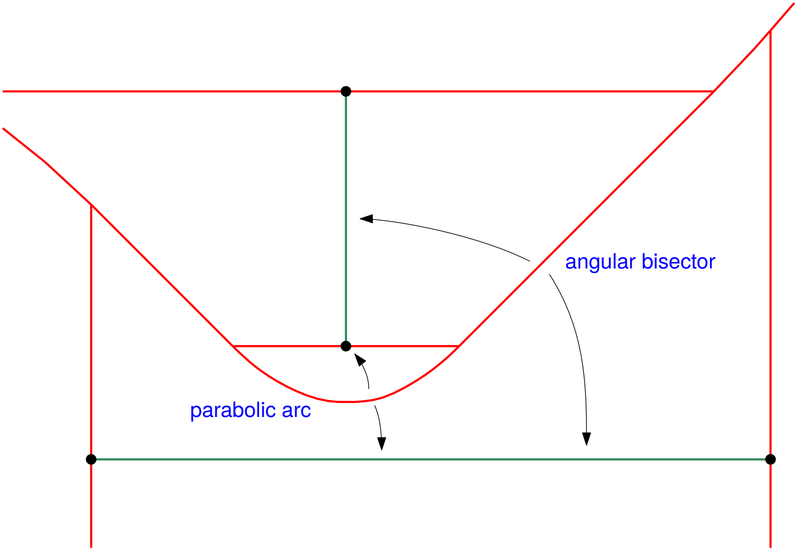
Sample Voronoi Diagram Dissected



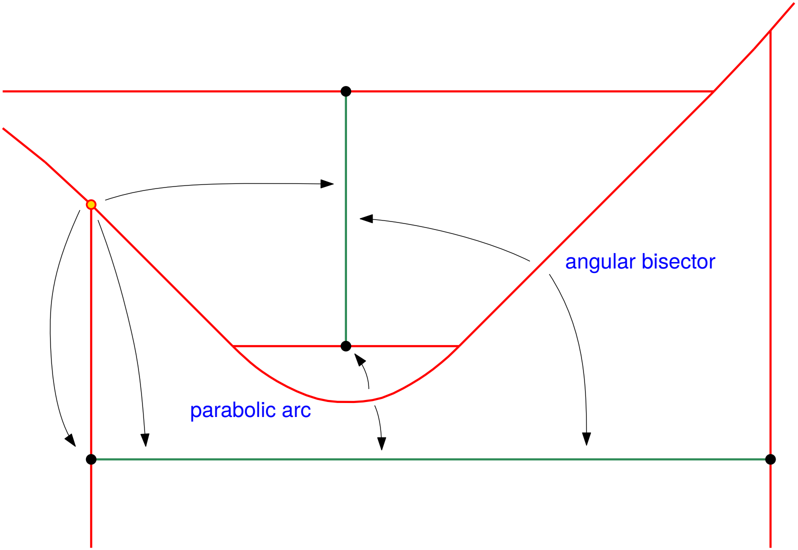
Sample Voronoi Diagram Dissected



Sample Voronoi Diagram Dissected

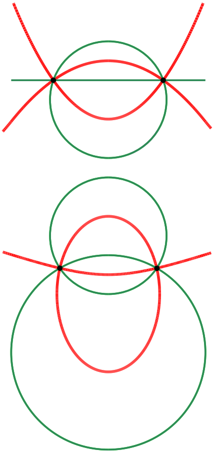


Sample Voronoi Diagram Dissected



Lemma 126

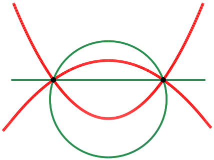
The structure $\mathcal{VD}(S)$ is a planar graph and consists of $O(n)$ parabolic, hyperbolic, elliptic and straight-line edges.

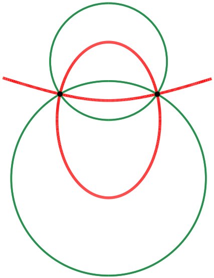


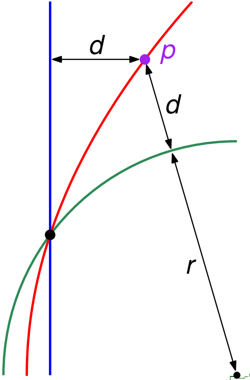
Bisectors

Lemma 126

The structure $\mathcal{VD}(S)$ is a planar graph and consists of $O(n)$ parabolic, hyperbolic, elliptic and straight-line edges.

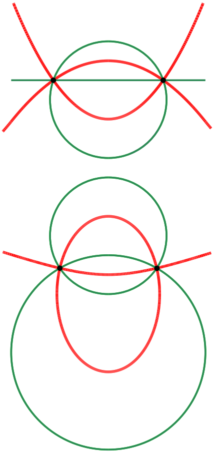


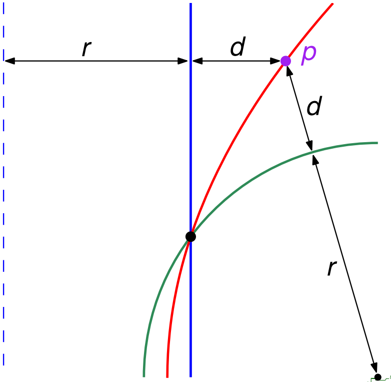




Lemma 126

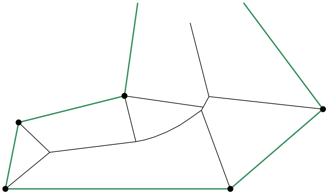
The structure $\mathcal{VD}(S)$ is a planar graph and consists of $O(n)$ parabolic, hyperbolic, elliptic and straight-line edges.





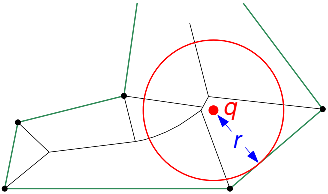
Definition 127 (Clearance)

The *clearance* of a point q relative to S is the radius r of the largest disk ("*clearance disk*") centered at q which does not contain any site of S in its interior.



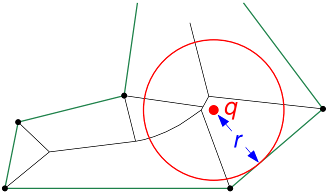
Definition 127 (Clearance)

The *clearance* of a point q relative to S is the radius r of the largest disk ("*clearance disk*") centered at q which does not contain any site of S in its interior.



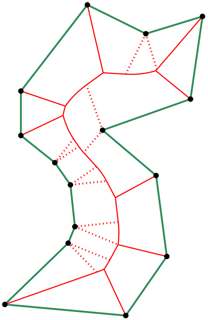
Definition 127 (Clearance)

The *clearance* of a point q relative to S is the radius r of the largest disk ("*clearance disk*") centered at q which does not contain any site of S in its interior.



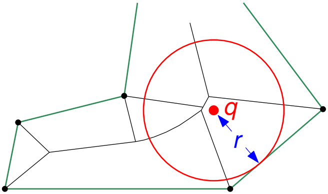
Definition 128 (Medial axis)

A point in the interior of a (multiply-connected) polygonal region belongs to the *medial axis* (MA) of the region if and only if its clearance disk touches the boundary in at least two disjoint points.



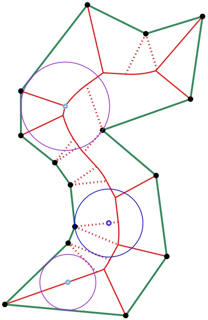
Definition 127 (Clearance)

The *clearance* of a point q relative to S is the radius r of the largest disk ("*clearance disk*") centered at q which does not contain any site of S in its interior.



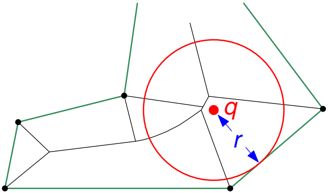
Definition 128 (Medial axis)

A point in the interior of a (multiply-connected) polygonal region belongs to the *medial axis* (MA) of the region if and only if its clearance disk touches the boundary in at least two disjoint points.



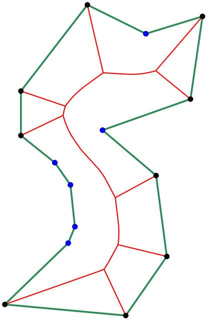
Definition 127 (Clearance)

The *clearance* of a point q relative to S is the radius r of the largest disk ("*clearance disk*") centered at q which does not contain any site of S in its interior.



Definition 128 (Medial axis)

A point in the interior of a (multiply-connected) polygonal region belongs to the *medial axis* (MA) of the region if and only if its clearance disk touches the boundary in at least two disjoint points.



Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

Theorem 129 (Fortune (1987))

The Voronoi diagram of n points and straight-line segments can be constructed in $O(n \log n)$ time by means of a sweep-line algorithm.

Theorem 130 (Yap (1987))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log n)$ time by means of a divide&conquer algorithm.

Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

Theorem 129 (Fortune (1987))

The Voronoi diagram of n points and straight-line segments can be constructed in $O(n \log n)$ time by means of a sweep-line algorithm.

Theorem 130 (Yap (1987))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log n)$ time by means of a divide&conquer algorithm.

Theorem 131 (Aichholzer et alii (2009))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log^2 n)$ expected time by means of randomization combined with a divide&conquer algorithm.

Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

Theorem 129 (Fortune (1987))

The Voronoi diagram of n points and straight-line segments can be constructed in $O(n \log n)$ time by means of a sweep-line algorithm.

Theorem 130 (Yap (1987))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log n)$ time by means of a divide&conquer algorithm.

Theorem 131 (Aichholzer et alii (2009))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log^2 n)$ expected time by means of randomization combined with a divide&conquer algorithm.

Theorem 132 (Held&Huber (2009), based on Held (2001))

The Voronoi diagram of n points, straight-line segments and circular arcs can be constructed in $O(n \log n)$ expected time by means of randomized incremental construction.

Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

- Several other $O(n \log n)$ expected-time algorithms for polygons and/or line segments . . .
- What about Voronoi diagrams of polygons? Can one achieve $o(n \log n)$?

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

- Several other $O(n \log n)$ expected-time algorithms for polygons and/or line segments ...
- What about Voronoi diagrams of polygons? Can one achieve $o(n \log n)$?

Theorem 133 (Aggarwal et alii (1989))

The Voronoi diagram of a convex polygon can be constructed in linear time.

Voronoi Diagram of Points, Line Segments and Arcs: State of the Art

- Several other $O(n \log n)$ expected-time algorithms for polygons and/or line segments ...
- What about Voronoi diagrams of polygons? Can one achieve $o(n \log n)$?

Theorem 133 (Aggarwal et alii (1989))

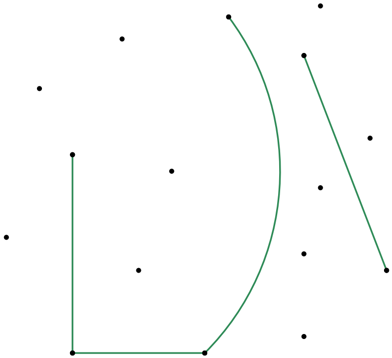
The Voronoi diagram of a convex polygon can be constructed in linear time.

Theorem 134 (Chin et alii (1999))

The Voronoi diagram of a simple polygon can be constructed in linear time.

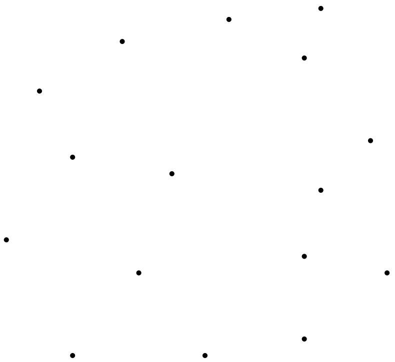
Randomized Incremental Construction

- How can we construct the Voronoi diagram of these sites?



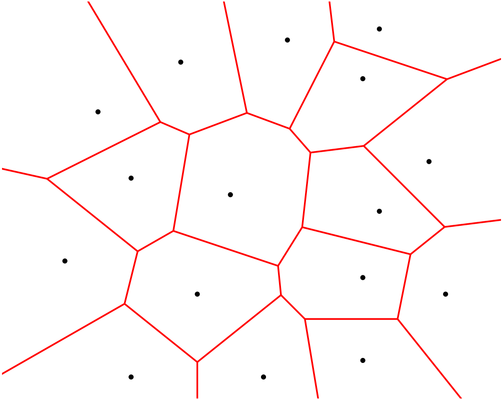
Randomized Incremental Construction

- [Held (2001), Held&Huber (2009)]: Start with the vertices of S.



Randomized Incremental Construction

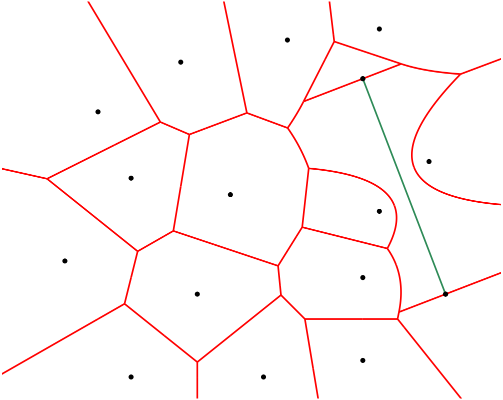
- [Held (2001), Held&Huber (2009)]: Start with the vertices of S, and compute their Voronoi diagram. (E.g., use randomized incremental construction.)

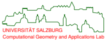


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Randomized Incremental Construction

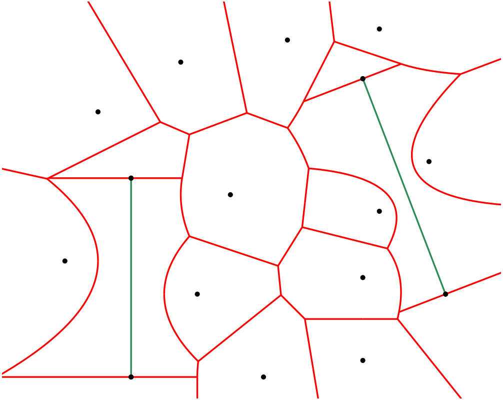
- [Held (2001), Held&Huber (2009)]: Start with the vertices of S, and compute their Voronoi diagram. (E.g., use randomized incremental construction.) Insert segments.





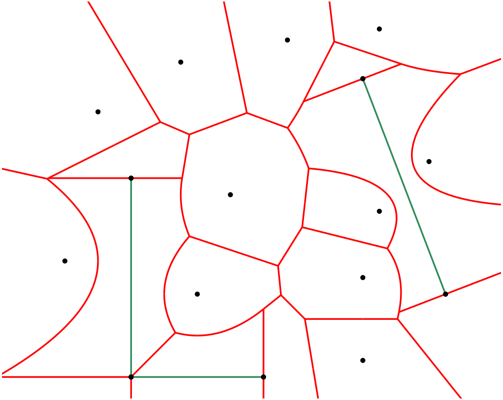
Randomized Incremental Construction

- [Held (2001), Held&Huber (2009)]: Start with the vertices of S, and compute their Voronoi diagram. (E.g., use randomized incremental construction.) Insert segments, in random order.

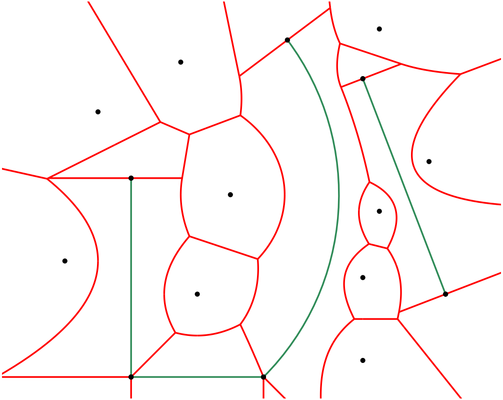


Randomized Incremental Construction

- [Held (2001), Held&Huber (2009)]: Start with the vertices of S, and compute their Voronoi diagram. (E.g., use randomized incremental construction.) Insert segments, in random order, one after the other.

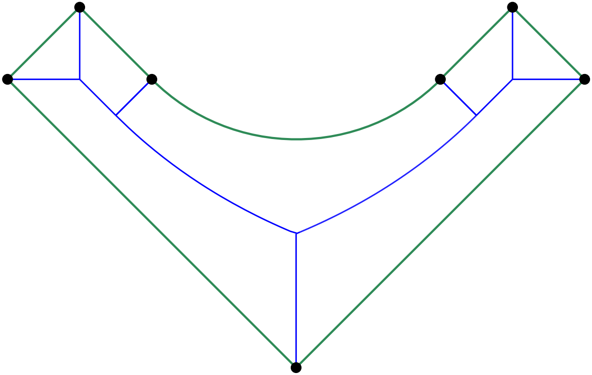


- [Held (2001), Held&Huber (2009)]: Start with the vertices of S, and compute their Voronoi diagram. (E.g., use randomized incremental construction.) Insert segments, in random order, one after the other. Same for the arcs.



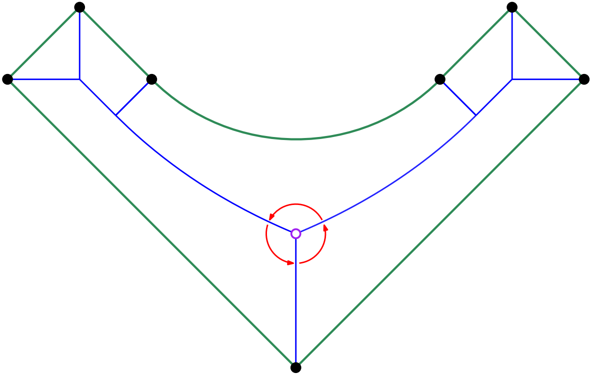
Storing a Voronoi Diagram: Topological Data

- Any standard way to represent a planar graph is good enough to store the topology of a Voronoi diagram.



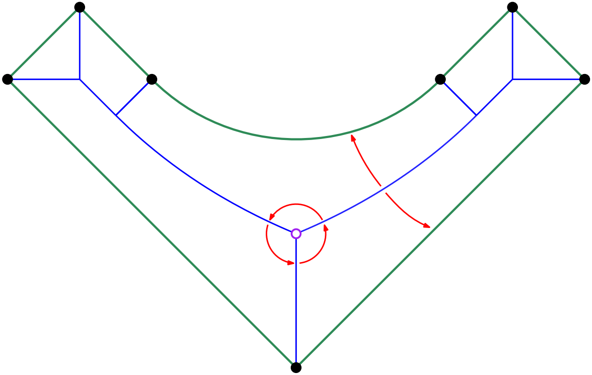
Storing a Voronoi Diagram: Topological Data

- Any standard way to represent a planar graph is good enough to store the topology of a Voronoi diagram.
- E.g., we
  - store CCW pointers around all nodes,



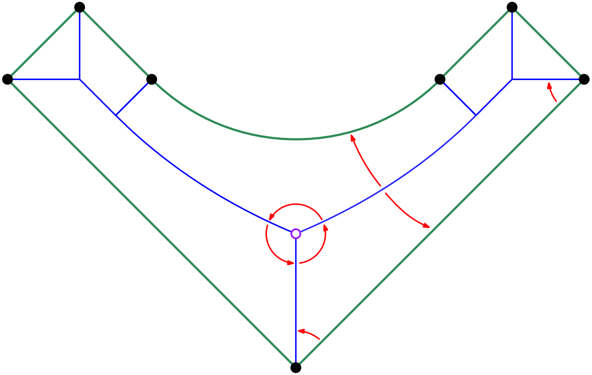
Storing a Voronoi Diagram: Topological Data

- Any standard way to represent a planar graph is good enough to store the topology of a Voronoi diagram.
- E.g., we
  - store CCW pointers around all nodes,
  - store pointers to the two defining sites of every edge,



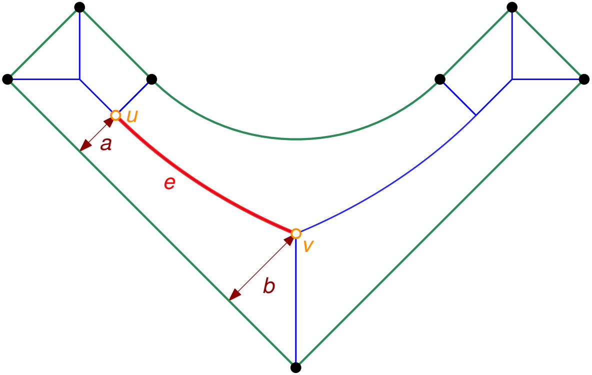
Storing a Voronoi Diagram: Topological Data

- Any standard way to represent a planar graph is good enough to store the topology of a Voronoi diagram.
- E.g., we
  - store CCW pointers around all nodes,
  - store pointers to the two defining sites of every edge,
  - store pointers to the first and last edge of a site's Voronoi region.



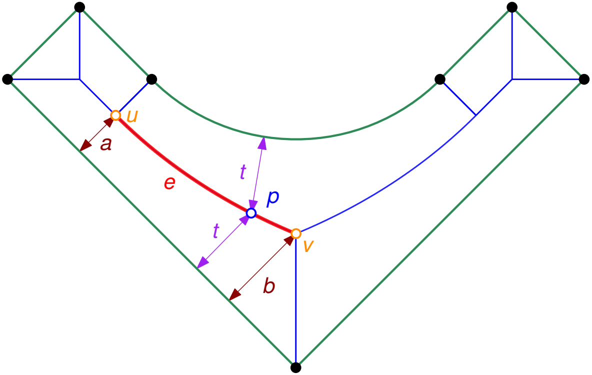
Storing a Voronoi Diagram: Numerical Data

- We assign a clearance-based parameterization $f\colon [a, b] \to \mathbb{R}^2$ to every edge e, where a is the minimum and b is the maximum clearance of points of e.



Storing a Voronoi Diagram: Numerical Data

- We assign a clearance-based parameterization $f\colon [a, b] \to \mathbb{R}^2$ to every edge e, where a is the minimum and b is the maximum clearance of points of e.
- The coordinates of a point p of e with clearance t are obtained by evaluating f: we have $p = f(t)$.

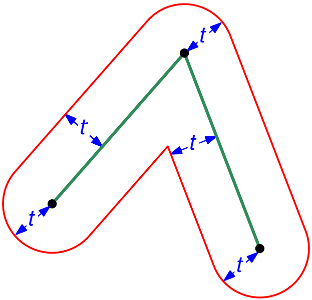


6 Skeletal Structures

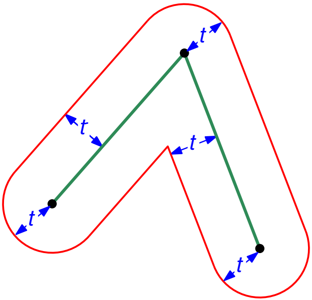
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

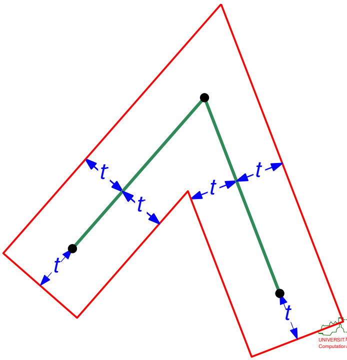
Straight Skeletons: Constant-Radius vs. Mitered Wavefront

- Voronoi diagram: constant-radius wavefront.



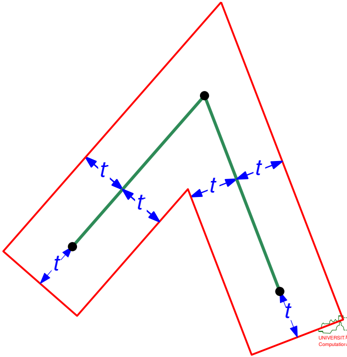
- Voronoi diagram: constant-radius wavefront.
- Straight skeleton: mitered wavefront.





Straight Skeletons: Motivation

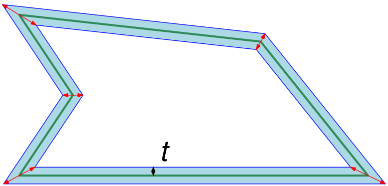
Aichholzer&Alberts&Aurenhammer&Gärtner (1995)



Straight Skeletons: Motivation

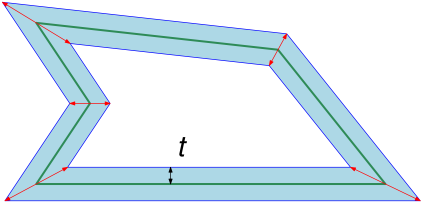
Aichholzer&Alberts&Aurenhammer&Gärtner (1995)

- Self-parallel mitered offsetting of input polygon $\mathcal{P}$ yields wavefront $\mathcal{WF}(\mathcal{P}, t)$ for offset distance t.



Aichholzer&Alberts&Aurenhammer&Gärtner (1995)

- Self-parallel mitered offsetting of input polygon $\mathcal{P}$ yields wavefront $\mathcal{WF}(\mathcal{P}, t)$ for offset distance t.
- Wavefront propagation: Shrinking process via continued self-parallel offsetting with unit speed, where offset distance t equals time.

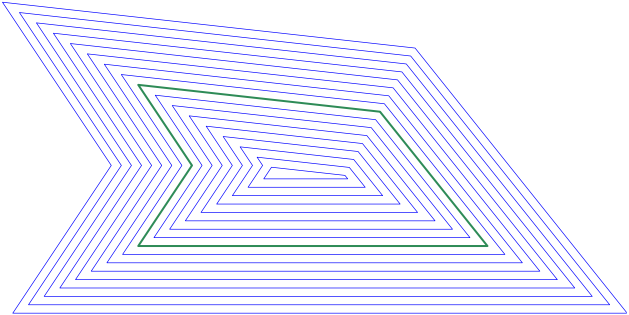


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Straight Skeletons: Motivation

Aichholzer&Alberts&Aurenhammer&Gärtner (1995)

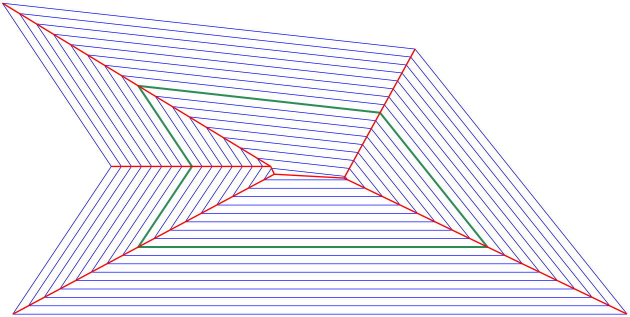
- Self-parallel mitered offsetting of input polygon $\mathcal{P}$ yields wavefront $\mathcal{WF}(\mathcal{P}, t)$ for offset distance t.
- Wavefront propagation: Shrinking process via continued self-parallel offsetting with unit speed, where offset distance t equals time.



Straight Skeletons: Motivation

Aichholzer&Alberts&Aurenhammer&Gärtner (1995)

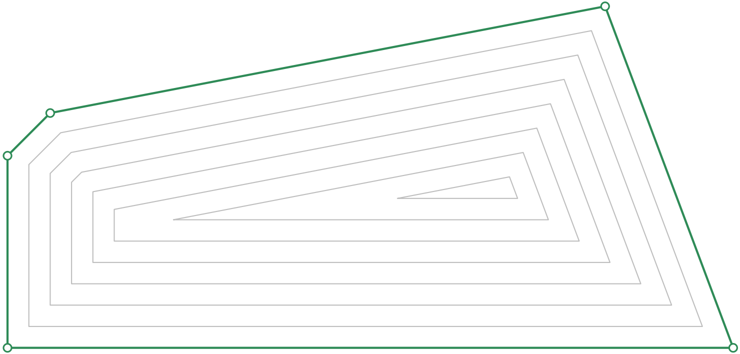
- Self-parallel mitered offsetting of input polygon $\mathcal{P}$ yields wavefront $\mathcal{WF}(\mathcal{P}, t)$ for offset distance t.
- Wavefront propagation: Shrinking process via continued self-parallel offsetting with unit speed, where offset distance t equals time.
- Straight skeleton $\mathcal{SK}(\mathcal{P})$ is union of traces of wavefront vertices.



Change of Wavefront Topology

Edge event

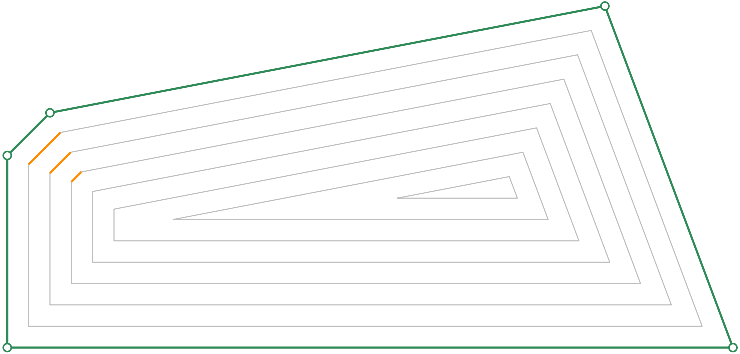
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.



Change of Wavefront Topology

Edge event

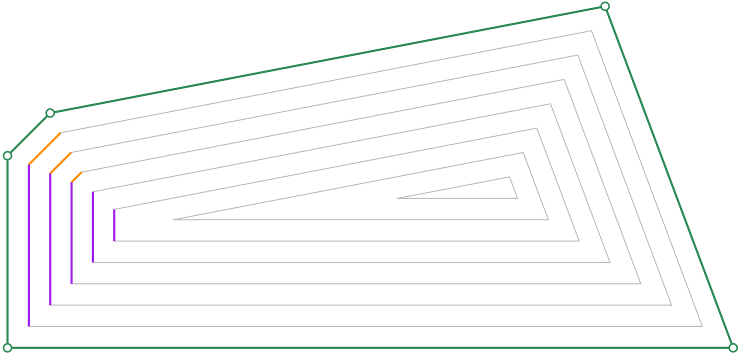
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.



Change of Wavefront Topology

Edge event

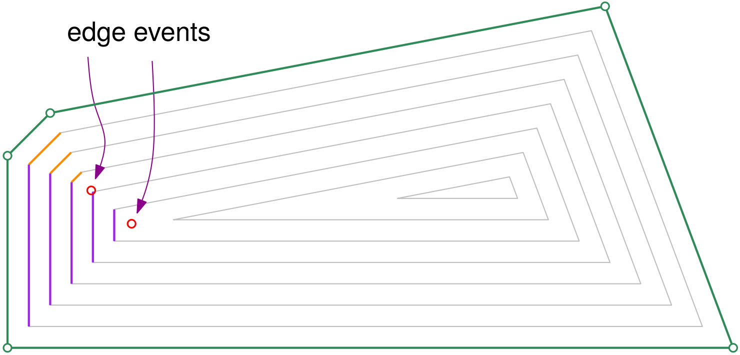
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.



Change of Wavefront Topology

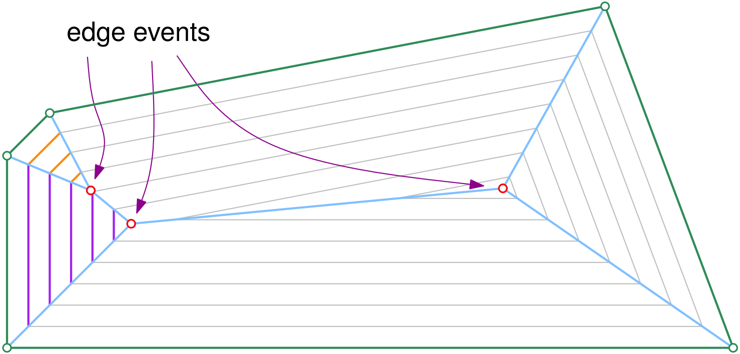
Edge event

- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.
- *Edge event*: an edge of $\mathcal{WF}(\mathcal{P}, t)$ vanishes.



Edge event

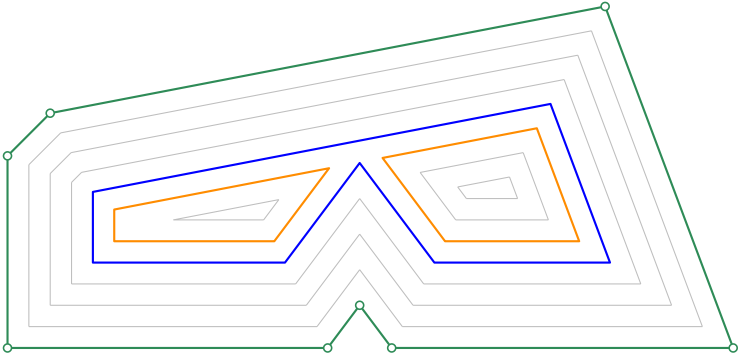
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.
- *Edge event*: an edge of $\mathcal{WF}(\mathcal{P}, t)$ vanishes.
- Such a change of topology of $\mathcal{WF}(\mathcal{P}, t)$ corresponds to a *node* of $\mathcal{SK}(\mathcal{P})$.



Change of Wavefront Topology

Split event

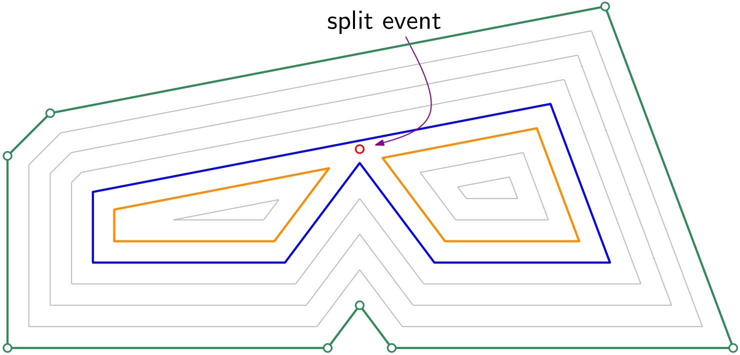
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.



Change of Wavefront Topology

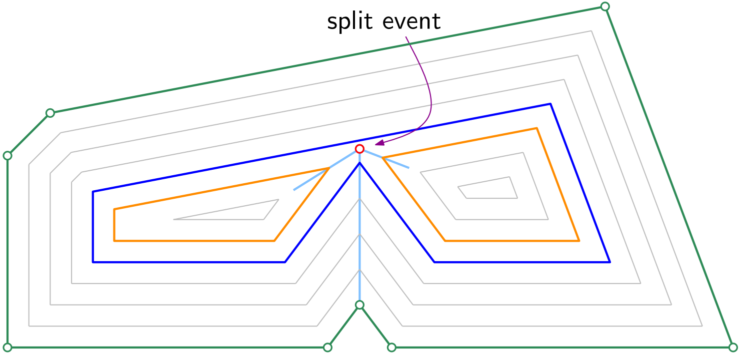
Split event

- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.
- *Split event*: Wavefront $\mathcal{WF}(\mathcal{P}, t)$ splits into two parts.



Split event

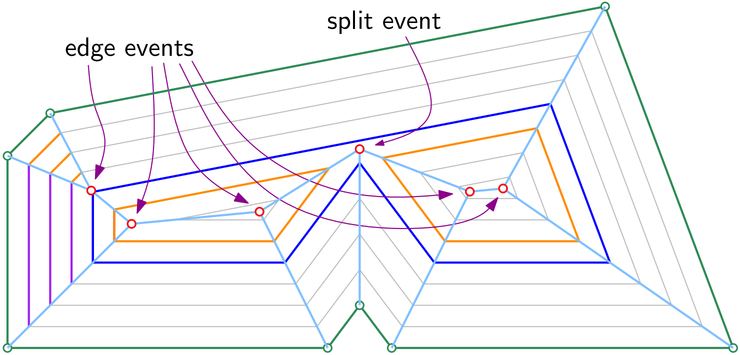
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.
- *Split event*: Wavefront $\mathcal{WF}(\mathcal{P}, t)$ splits into two parts.
- Also split events correspond to *nodes* of $\mathcal{SK}(\mathcal{P})$.

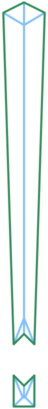


Change of Wavefront Topology

Split event

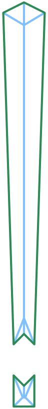
- Topology of wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes over time.
- *Split event*: Wavefront $\mathcal{WF}(\mathcal{P}, t)$ splits into two parts.
- Also split events correspond to *nodes* of $\mathcal{SK}(\mathcal{P})$.



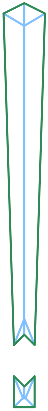


Definition 135

The *straight skeleton* $\mathcal{SK}(\mathcal{P})$ of a polygon $\mathcal{P}$ is given by the union of traces of wavefront vertices of $\mathcal{P}$ over the entire wavefront propagation process.



Straight Skeleton of Polygon

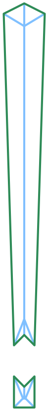


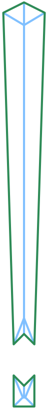
Definition 135

The *straight skeleton* $\mathcal{SK}(\mathcal{P})$ of a polygon $\mathcal{P}$ is given by the union of traces of wavefront vertices of $\mathcal{P}$ over the entire wavefront propagation process.

Lemma 136

1. The topology of the wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes with time/distance t due to edge and split events.



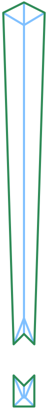


Definition 135

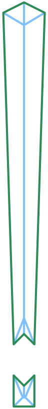
The *straight skeleton* $\mathcal{SK}(\mathcal{P})$ of a polygon $\mathcal{P}$ is given by the union of traces of wavefront vertices of $\mathcal{P}$ over the entire wavefront propagation process.

Lemma 136

1. The topology of the wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes with time/distance t due to edge and split events.
2. These events correspond to nodes of $\mathcal{SK}(\mathcal{P})$.



Straight Skeleton of Polygon

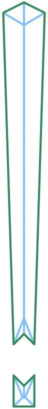


Definition 135

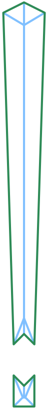
The *straight skeleton* $\mathcal{SK}(\mathcal{P})$ of a polygon $\mathcal{P}$ is given by the union of traces of wavefront vertices of $\mathcal{P}$ over the entire wavefront propagation process.

Lemma 136

1. The topology of the wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes with time/distance t due to edge and split events.
2. These events correspond to nodes of $\mathcal{SK}(\mathcal{P})$.
3. No metric-based definition of straight skeletons exists.



Straight Skeleton of Polygon

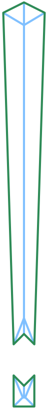


Definition 135

The *straight skeleton* $\mathcal{SK}(\mathcal{P})$ of a polygon $\mathcal{P}$ is given by the union of traces of wavefront vertices of $\mathcal{P}$ over the entire wavefront propagation process.

Lemma 136

1. The topology of the wavefront $\mathcal{WF}(\mathcal{P}, t)$ changes with time/distance t due to edge and split events.
2. These events correspond to nodes of $\mathcal{SK}(\mathcal{P})$.
3. No metric-based definition of straight skeletons exists.
4. If $\mathcal{P}$ has n segments then $\mathcal{SK}(\mathcal{P})$ consists of $O(n)$ nodes and $O(n)$ straight-line edges.

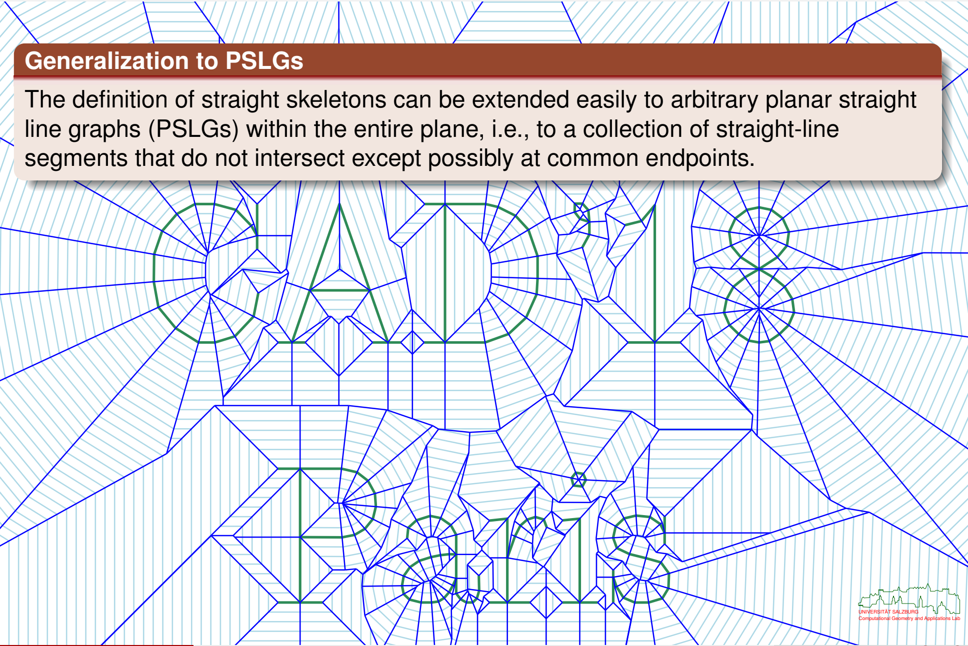


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Straight Skeleton of a Planar Straight-Line Graph

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Straight Skeleton of a Planar Straight-Line Graph

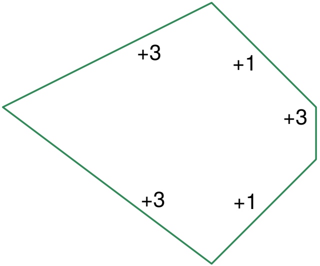


Generalization to PSLGs

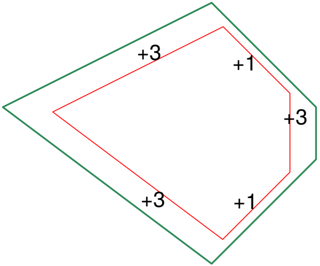
The definition of straight skeletons can be extended easily to arbitrary planar straight line graphs (PSLGs) within the entire plane, i.e., to a collection of straight-line segments that do not intersect except possibly at common endpoints.

Weighted Straight Skeletons

- Multiplicative weights: Edges move at different speeds

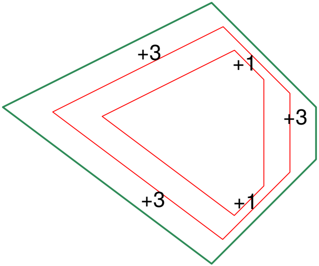


- Multiplicative weights: Edges move at different speeds



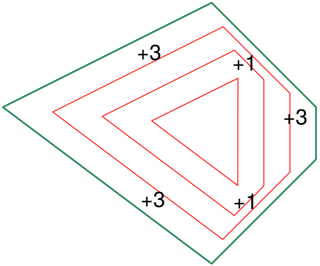
Weighted Straight Skeletons

- Multiplicative weights: Edges move at different speeds

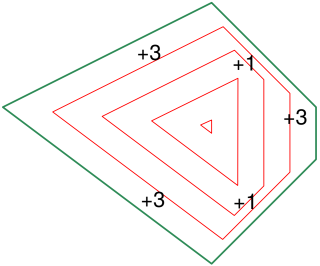


Weighted Straight Skeletons

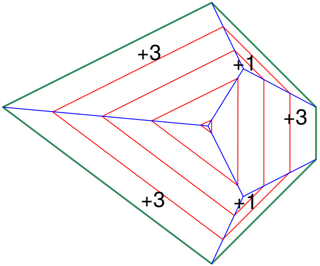
- Multiplicative weights: Edges move at different speeds



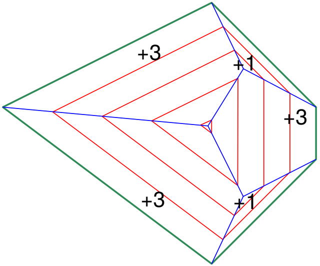
- Multiplicative weights: Edges move at different speeds

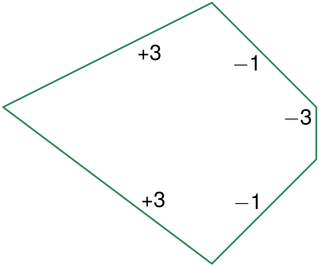


- Multiplicative weights: Edges move at different speeds

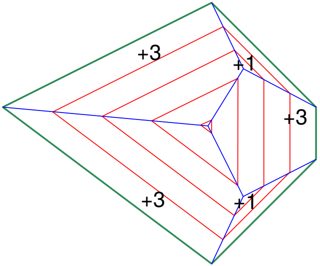


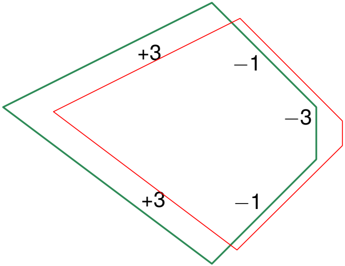
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.





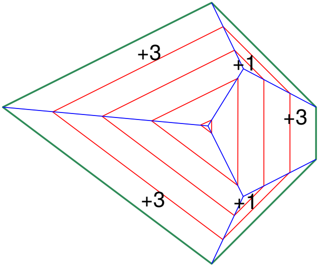
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.

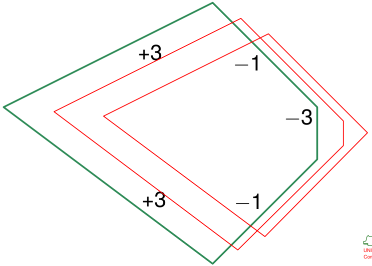




Weighted Straight Skeletons

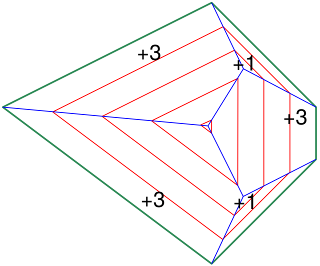
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.

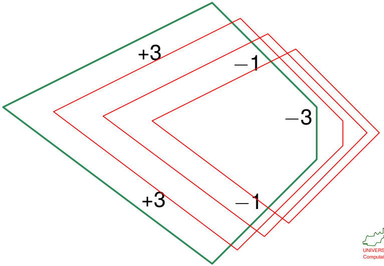




Weighted Straight Skeletons

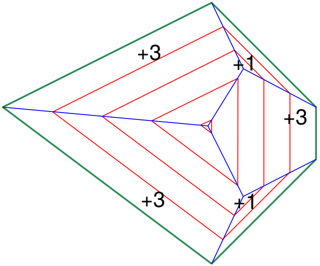
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.

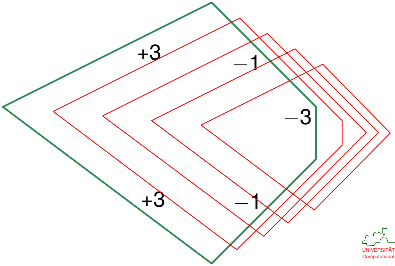




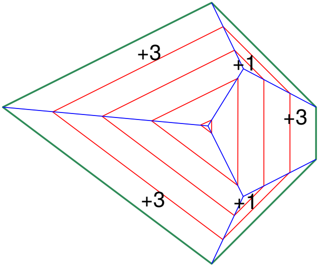
Weighted Straight Skeletons

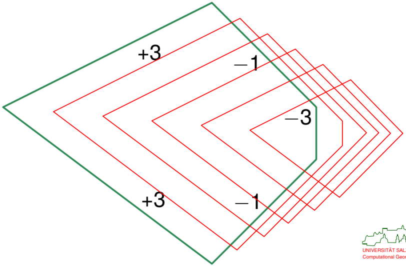
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.





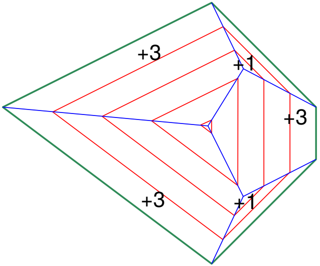
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.

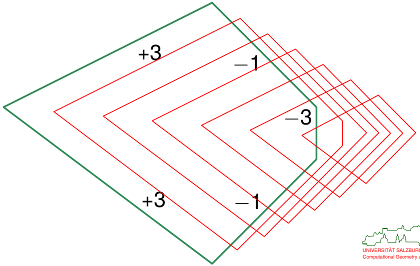




Weighted Straight Skeletons

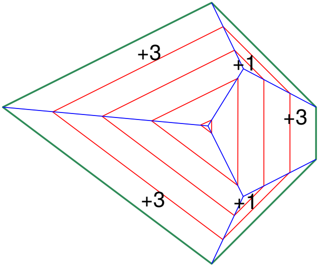
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.

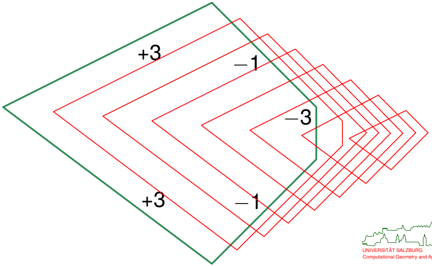




Weighted Straight Skeletons

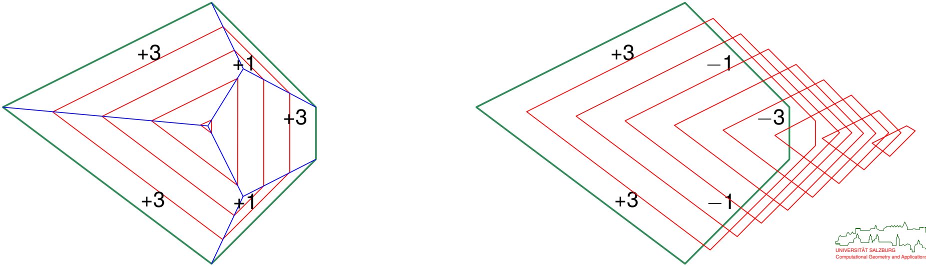
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.





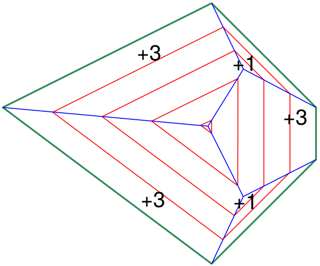
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

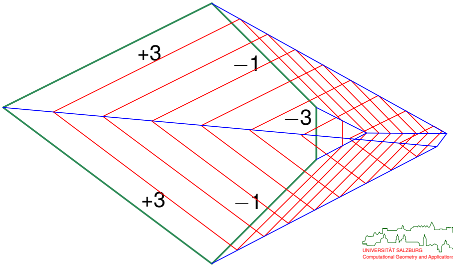
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.



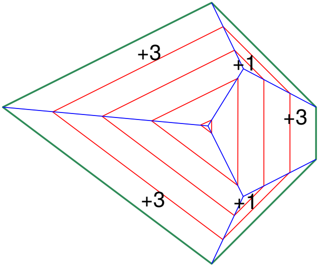
Weighted Straight Skeletons

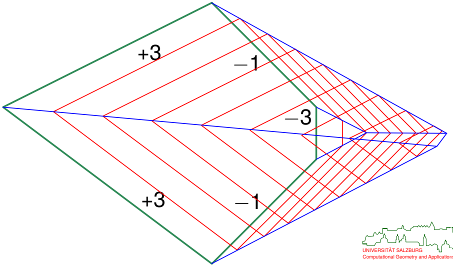
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.





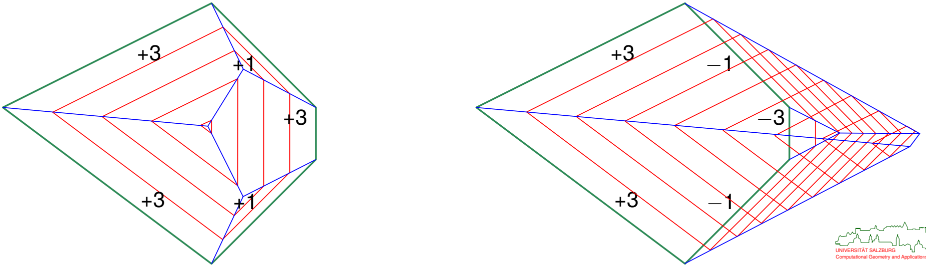
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.
- [Barequet et alii (2008)]: Weighted straight skeletons in 2D can be used for computing a straight skeleton in the interior of a polyhedron in 3D.





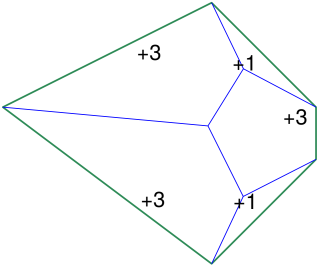
Weighted Straight Skeletons

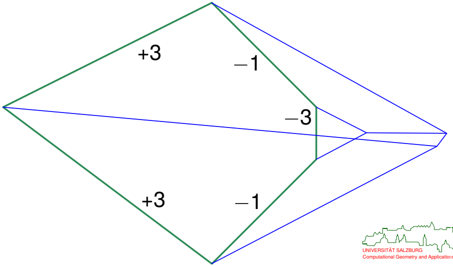
- Multiplicative weights: Edges move at different speeds, possibly even at negative speeds.
- [Barequet et alii (2008)]: Weighted straight skeletons in 2D can be used for computing a straight skeleton in the interior of a polyhedron in 3D.
- Which of the properties of the straight skeleton (planarity, tree structure, faces are monotone) carry over to weighted straight skeletons?



Theorem 137 (Biedl et alii (2014))

The geometric, graph-theoretical, and combinatorial properties of multiplicatively weighted straight skeletons are identical to unweighted straight-skeletons if all weights are positive. If negative weights are allowed then the weighted straight skeleton of even a convex polygon may contain crossings and cycles.





Theorem 138 (Aichholzer et alii (1995))

The straight skeleton of a simple n-gon with r reflex vertices can be computed in $O(nr \log n)$ time.

Straight Skeleton: State of the Art

Theorem 138 (Aichholzer et alii (1995))

The straight skeleton of a simple n-gon with r reflex vertices can be computed in $O(nr \log n)$ time.

Theorem 139 (Aichholzer&Aurenhammer (1998))

A wavefront-propagation can be used to compute the straight skeleton of an n-vertex PSLG in $O(n^3 \log n)$ time.

Straight Skeleton: State of the Art

Theorem 138 (Aichholzer et alii (1995))

The straight skeleton of a simple n-gon with r reflex vertices can be computed in $O(nr \log n)$ time.

Theorem 139 (Aichholzer&Aurenhammer (1998))

A wavefront-propagation can be used to compute the straight skeleton of an n-vertex PSLG in $O(n^3 \log n)$ time.

Theorem 140 (Eppstein&Erickson (1999))

Efficient closest-pair data structures can be combined in a hierarchical fashion to achieve an $O(n^{17/11+\varepsilon})$ time and space complexity for computing the straight skeleton of an n-vertex PSLG.

Theorem 141 (Cheng&Vigneron (2007))

Based on $1/\sqrt{r}$ cuttings, the straight skeleton of a simple n-gon with r reflex vertices can be computed in expected time $O(n \log^2 n + r\sqrt{r} \log r)$.

Theorem 142 (Huber&Held (2012))

A straight-skeleton algorithm based on motorcycle-graph computations can be engineered to run in $O(n \log n)$ time and $O(n)$ space for practical n-vertex PSLGs. However, its worst-case complexity is $O(n^2 \log n)$.

Straight Skeleton: State of the Art

Theorem 142 (Huber&Held (2012))

A straight-skeleton algorithm based on motorcycle-graph computations can be engineered to run in $O(n \log n)$ time and $O(n)$ space for practical n-vertex PSLGs. However, its worst-case complexity is $O(n^2 \log n)$.

Theorem 143 (Palfrader&Held&Huber (2012))

A wavefront-propagation based on kinetic triangulations can be engineered to run in $O(n \log n)$ time and $O(n)$ space for practical n-vertex PSLGs. In particular, only $O(n)$ flip events occur in practice. However, its worst-case complexity is $O(n^3 \log n)$.

Straight Skeleton: State of the Art

Theorem 142 (Huber&Held (2012))

A straight-skeleton algorithm based on motorcycle-graph computations can be engineered to run in $O(n \log n)$ time and $O(n)$ space for practical n-vertex PSLGs. However, its worst-case complexity is $O(n^2 \log n)$.

Theorem 143 (Palfrader&Held&Huber (2012))

A wavefront-propagation based on kinetic triangulations can be engineered to run in $O(n \log n)$ time and $O(n)$ space for practical n-vertex PSLGs. In particular, only $O(n)$ flip events occur in practice. However, its worst-case complexity is $O(n^3 \log n)$.

Theorem 144 (Biedl et alii (2014))

The weighted straight skeleton of an n-vertex convex polygon (with positive multiplicative weights) admits a non-procedural characterization and can be computed in $O(n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

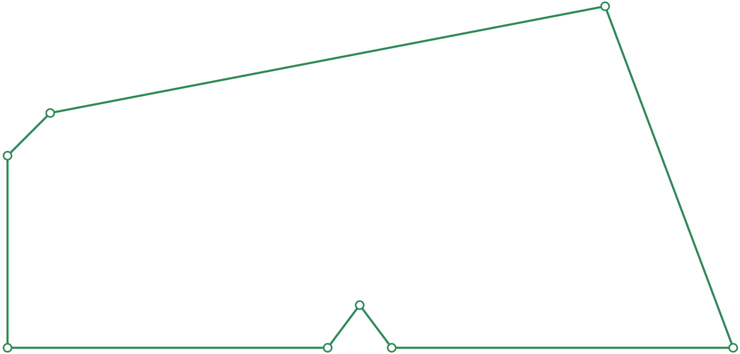
Theorem 145 (Vigneron&Yan (2013))

A motorcycle-graph based algorithm allows to compute the straight skeleton of a non-degenerate polygon with n vertices and h holes in time $O(n\sqrt{h+1}\log^2 n + n^{4/3+\varepsilon})$, for any $\varepsilon > 0$. If all coordinates are $O(\log n)$-bit rationals then the straight skeleton of a simple polygon can be computed in $O(n\log^3 n)$ expected time.

Theorem 146 (Cheng&Mencel&Vigneron (2014))

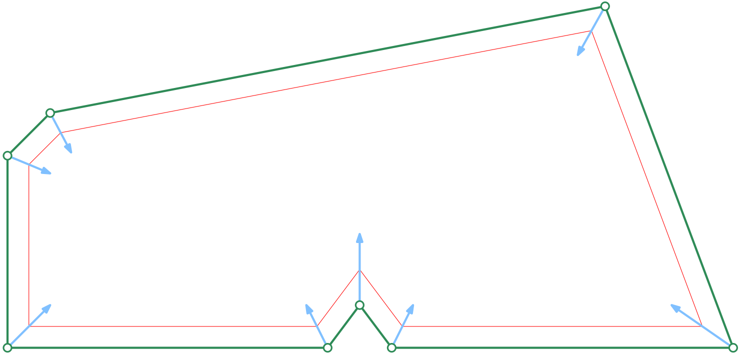
A motorcycle-graph based algorithm allows to compute the straight skeleton of a non-degenerate polygon with n vertices, with r vertices being reflex, in time $O(n\log n\log r + r^{4/3+\varepsilon})$, for any $\varepsilon > 0$. For degenerate input the time increases to $O(n\log n\log r + r^{17/11+\varepsilon})$.

Wavefront Propagation for Computing Straight Skeletons



Basic idea

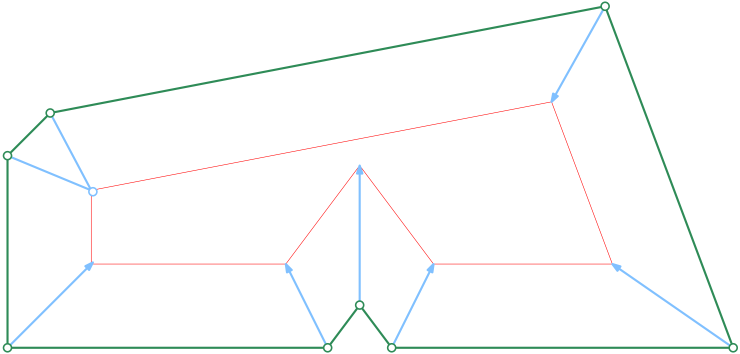
- Simulate the wavefront propagation.



Wavefront Propagation for Computing Straight Skeletons

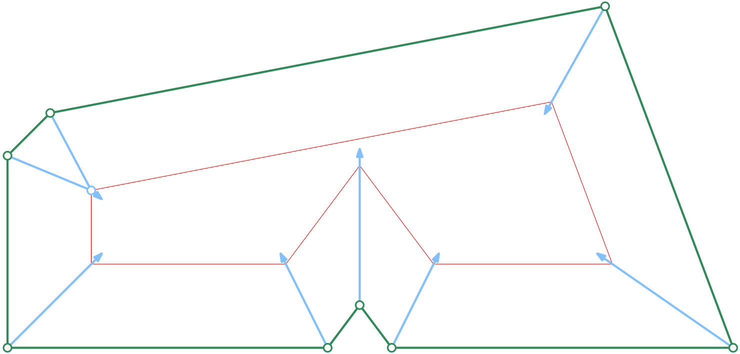
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?



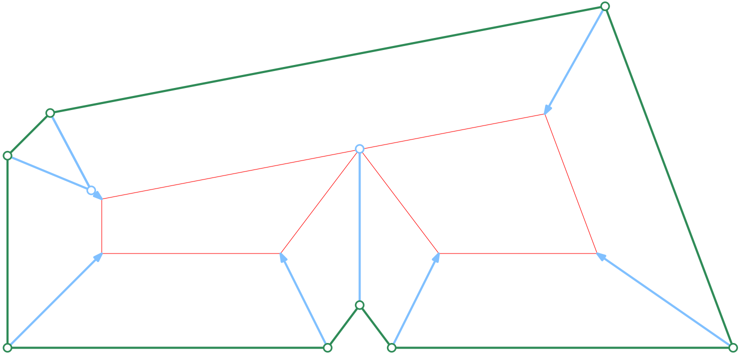
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



Basic idea

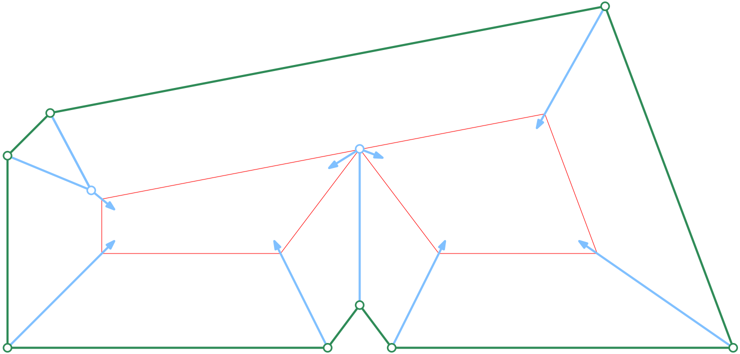
- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



Wavefront Propagation for Computing Straight Skeletons

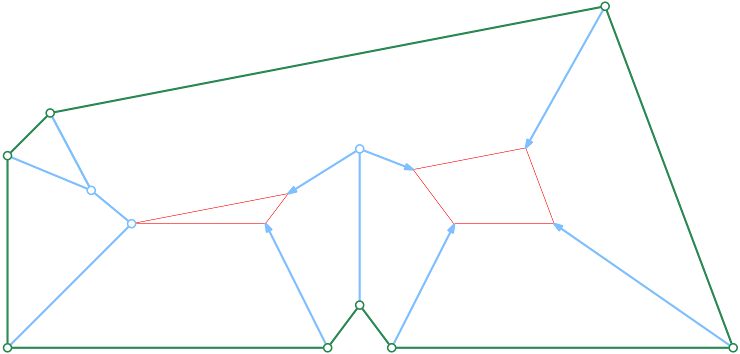
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



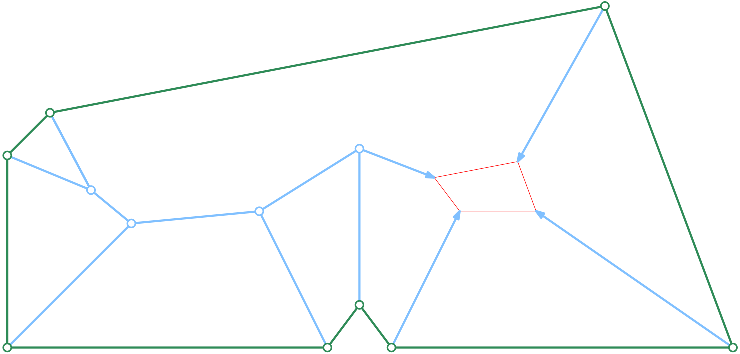
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



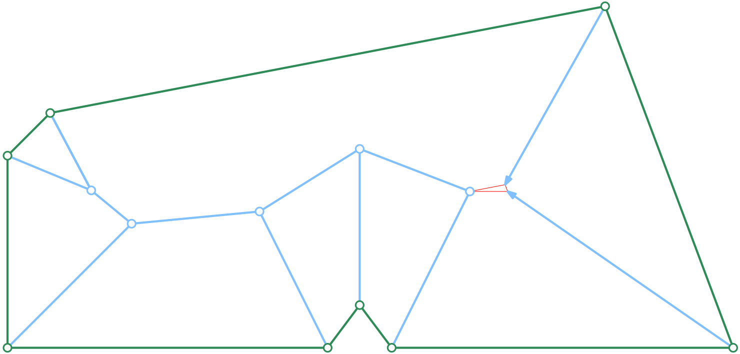
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



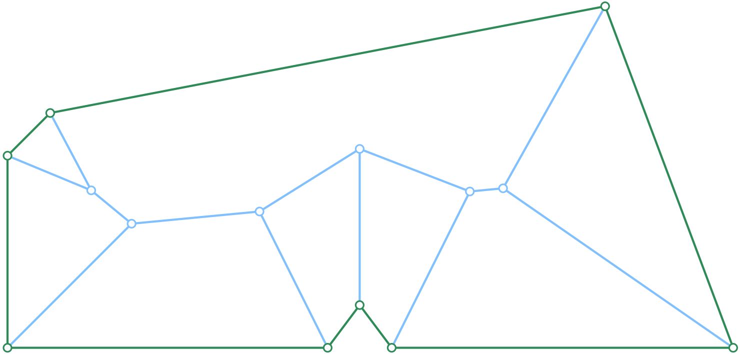
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



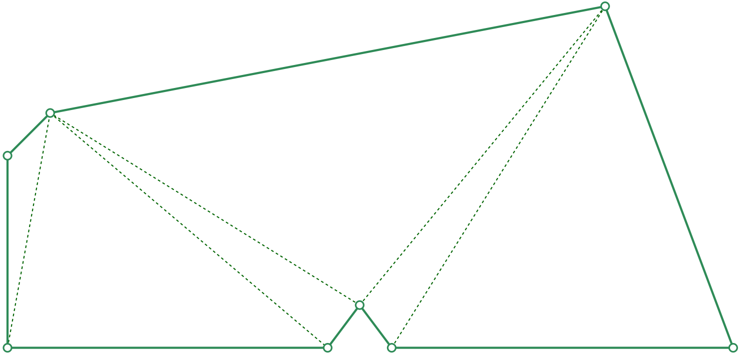
Basic idea

- Simulate the wavefront propagation.
- Problem: When will the next event happen? Which event?
- If we can solve this problem then we can construct straight skeletons.



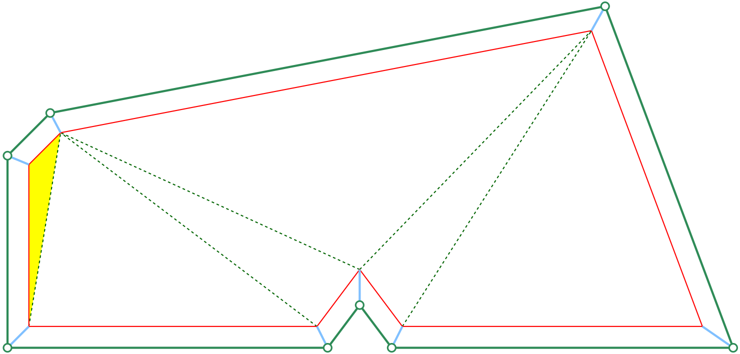
Aichholzer&Aurenhammer (1998)

- Maintain a kinetic triangulation of (the interior of) the wavefront.



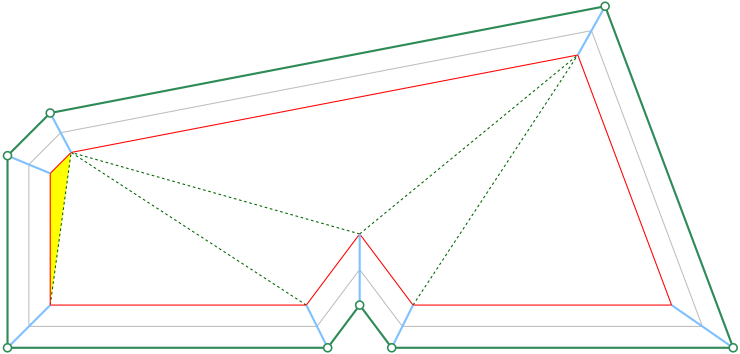
Aichholzer&Aurenhammer (1998)

- Maintain a kinetic triangulation of (the interior of) the wavefront.



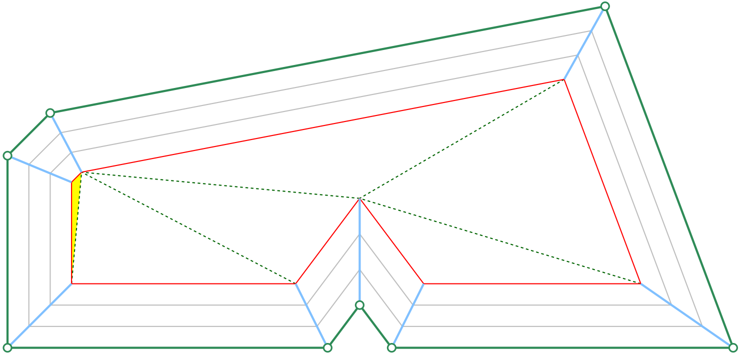
Aichholzer&Aurenhammer (1998)

- Maintain a kinetic triangulation of (the interior of) the wavefront.



Aichholzer&Aurenhammer (1998)

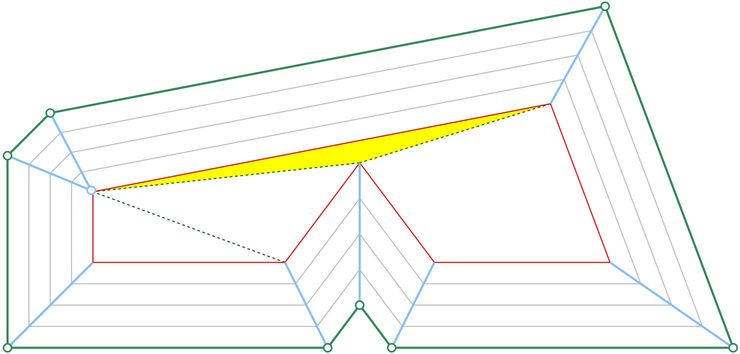
- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.



Triangulation-Based Algorithm

Aichholzer&Aurenhammer (1998)

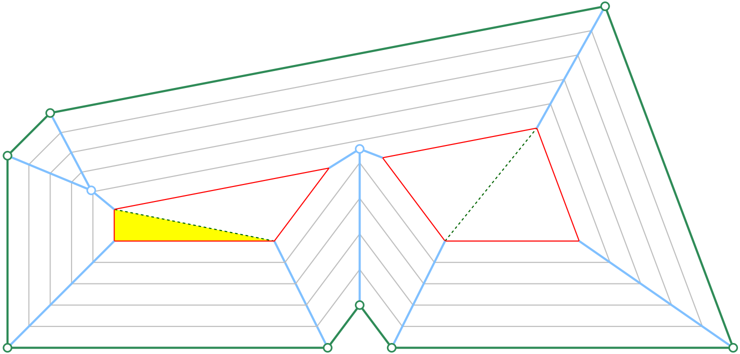
- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.



Triangulation-Based Algorithm

Aichholzer&Aurenhammer (1998)

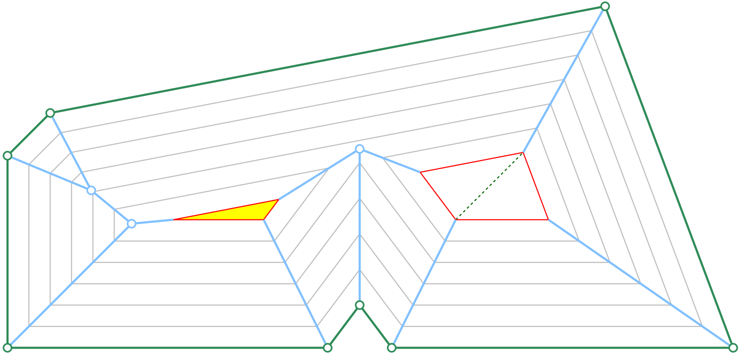
- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.
- A triangle collapses when its area becomes zero.



Triangulation-Based Algorithm

Aichholzer&Aurenhammer (1998)

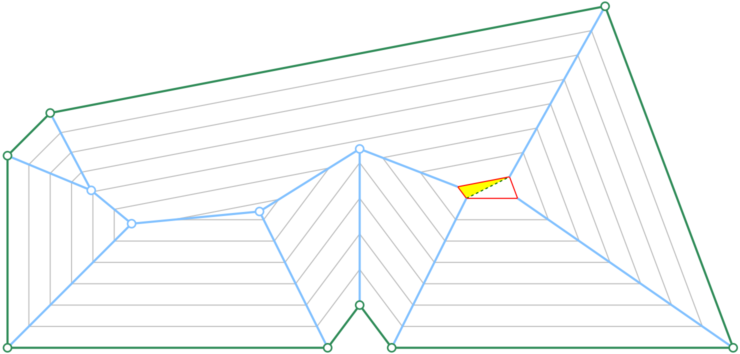
- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.
- A triangle collapses when its area becomes zero.



Triangulation-Based Algorithm

Aichholzer&Aurenhammer (1998)

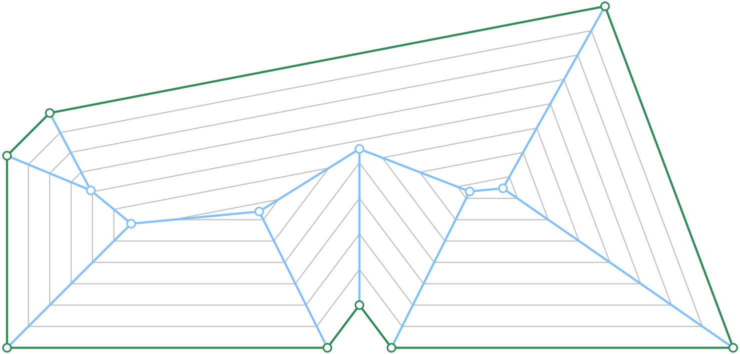
- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.
- A triangle collapses when its area becomes zero.



Triangulation-Based Algorithm

Aichholzer&Aurenhammer (1998)

- Maintain a kinetic triangulation of (the interior of) the wavefront.
- Collapsing triangles witness edge and split events.
- A triangle collapses when its area becomes zero.



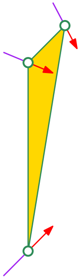
Triangulation-Based Algorithm

Algorithmic insight

Collapsing triangles witness edge and split events.

Algorithmic insight

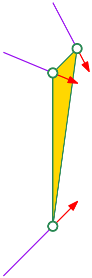
Collapsing triangles witness edge and split events.



Triangulation-Based Algorithm

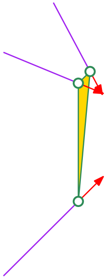
Algorithmic insight

Collapsing triangles witness edge and split events.



Algorithmic insight

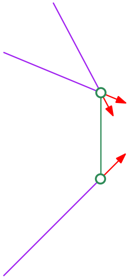
Collapsing triangles witness edge and split events.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Algorithmic insight

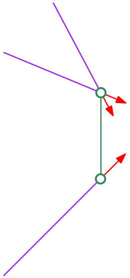
Collapsing triangles witness edge and split events.



Algorithmic insight

Collapsing triangles witness edge and split events.

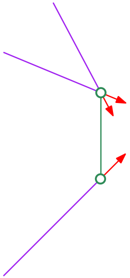
- Compute collapse times of triangles.



Algorithmic insight

Collapsing triangles witness edge and split events.

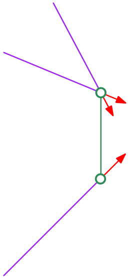
- Compute collapse times of triangles.
- That is, determine when the area of a triangle becomes zero.



Algorithmic insight

Collapsing triangles witness edge and split events.

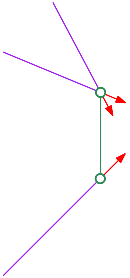
- Compute collapse times of triangles.
- That is, determine when the area of a triangle becomes zero.
- Maintain a priority queue of collapse events.



Algorithmic insight

Collapsing triangles witness edge and split events.

- Compute collapse times of triangles.
- That is, determine when the area of a triangle becomes zero.
- Maintain a priority queue of collapse events.
- Update triangulation and priority queue as required upon events.

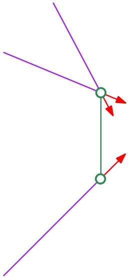


Triangulation-Based Algorithm

Algorithmic insight

Collapsing triangles witness edge and split events.

- Compute collapse times of triangles.
- That is, determine when the area of a triangle becomes zero.
- Maintain a priority queue of collapse events.
- Update triangulation and priority queue as required upon events.



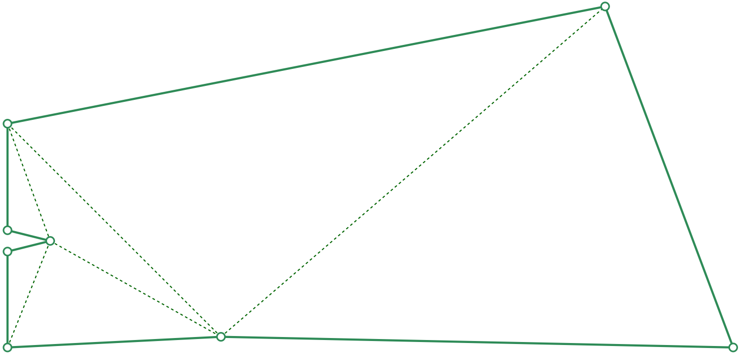
Wavefront propagation based on kinetic triangulations . . .

. . . allows to determine all events and to compute straight skeletons.

Triangulation-Based Algorithm

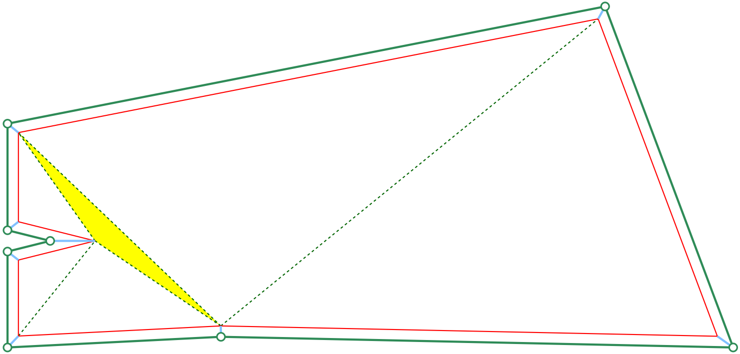
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.



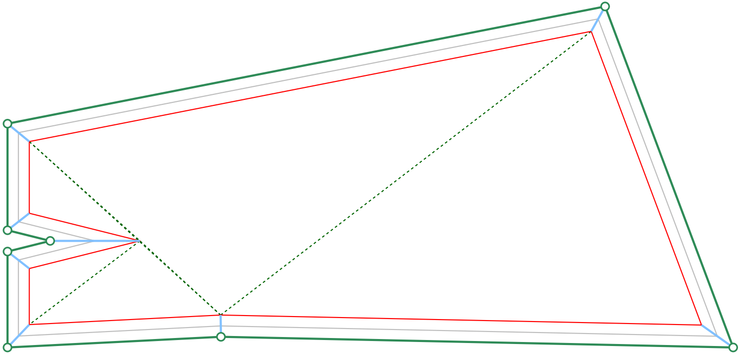
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.



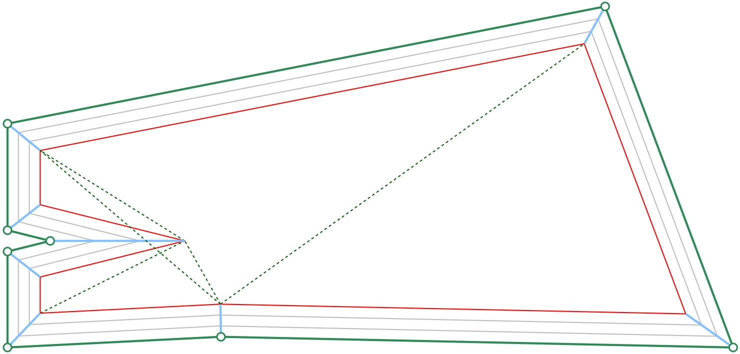
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.



Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!

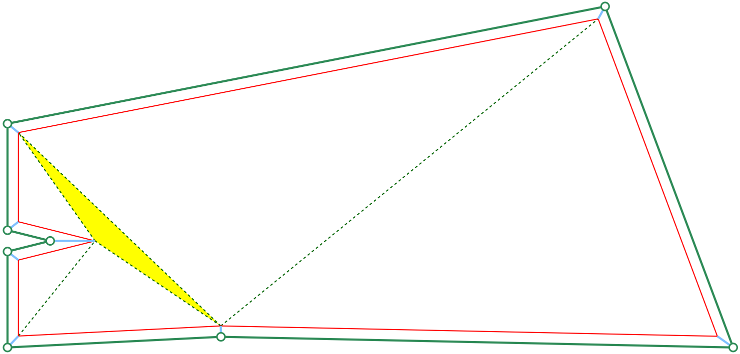


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation-Based Algorithm

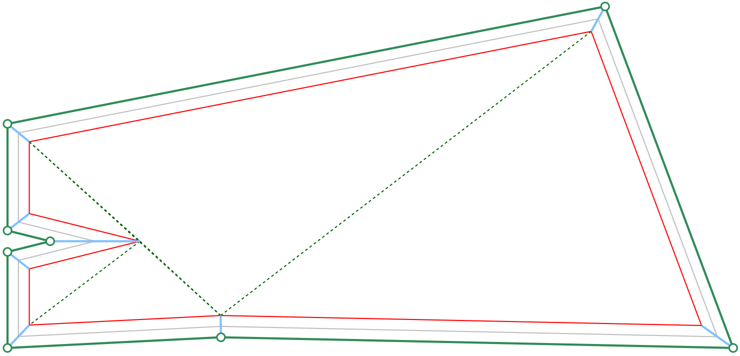
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.



Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.

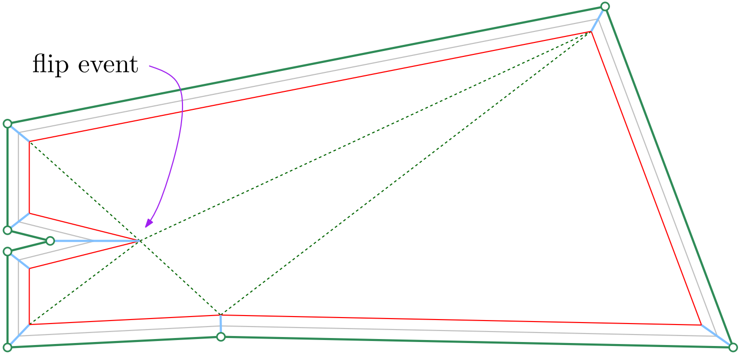


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Triangulation-Based Algorithm

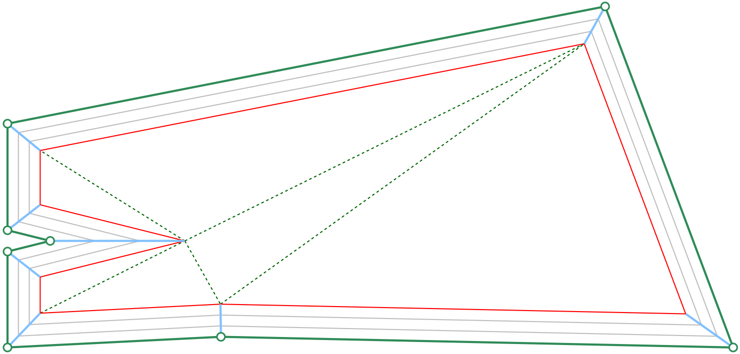
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.



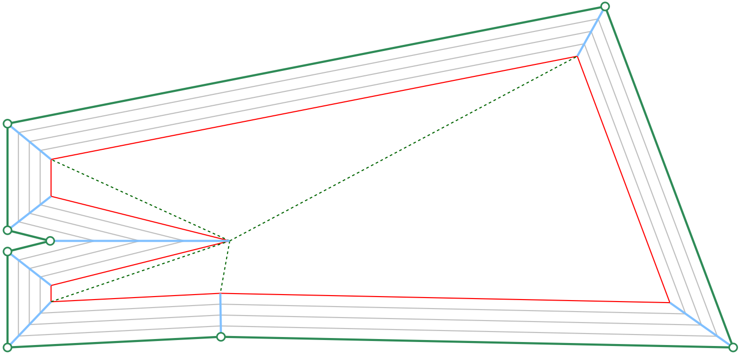
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.



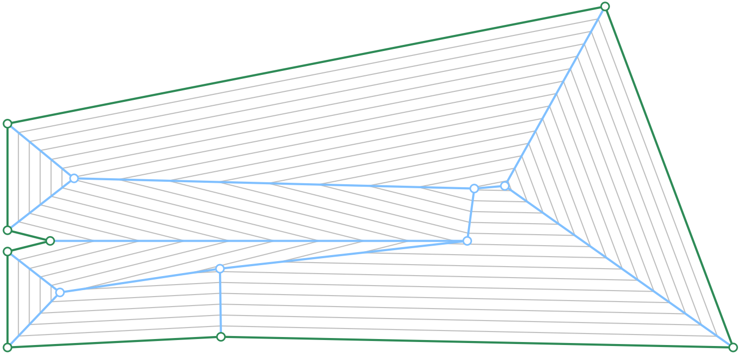
Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.



Flip events

- Caveat: Not all collapses witness changes in the wavefront topology.
- Such collapses cannot be ignored!
- Rather these collapes need special processing: *flip events*.

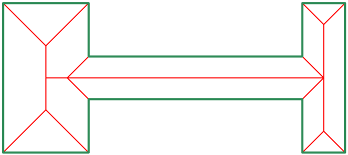


Implementation of Triangulation-Based Algorithm

- Long way to go from the theoretical sketch by Aichholzer&Aurenhammer (1998) to an actual implementation . . .

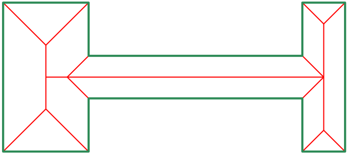
Implementation of Triangulation-Based Algorithm

- Long way to go from the theoretical sketch by Aichholzer&Aurenhammer (1998) to an actual implementation . . .
- [Palfrader&Held&Huber (2012)]: Need to avoid flip-event loops.
- [Palfrader&Held (2015)]: Need to handle degeneracies that cause multiple simultaneous events.



Implementation of Triangulation-Based Algorithm

- Long way to go from the theoretical sketch by Aichholzer&Aurenhammer (1998) to an actual implementation . . .
- [Palfrader&Held&Huber (2012)]: Need to avoid flip-event loops.
- [Palfrader&Held (2015)]: Need to handle degeneracies that cause multiple simultaneous events.
- [Palfrader&Held (2015)]: Need to detect and classify simultaneous events reliably on a standard floating-point arithmetic.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Implementation of Triangulation-Based Algorithm: SURFER

SURFER

Straight-skeleton algorithm, based on kinetic triangulations and standard floating-point arithmetic, implemented in C and named SURFER.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Implementation of Triangulation-Based Algorithm: SURFER

SURFER

Straight-skeleton algorithm, based on kinetic triangulations and standard floating-point arithmetic, implemented in C and named SURFER.

Experimental result [Palfrader&Held (2015)]

SURFER runs in $O(n \log n)$ time for n-vertex PSLGs. In particular, only a (small) linear number of flip events occur.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Implementation of Triangulation-Based Algorithm: SURFER

SURFER

Straight-skeleton algorithm, based on kinetic triangulations and standard floating-point arithmetic, implemented in C and named SURFER.

Experimental result [Palfrader&Held (2015)]

SURFER runs in $O(n \log n)$ time for n-vertex PSLGs. In particular, only a (small) linear number of flip events occur.

How many flip events can occur in the worst case?

This is an open problem! Trivial upper bound is $\Theta(n^3)$, but only (highly contrived) inputs with $\Theta(n^2)$ flips are known.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

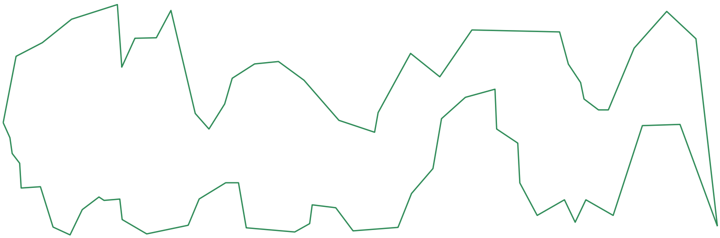
Straight Skeleton of Monotone Polygons

- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?

- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

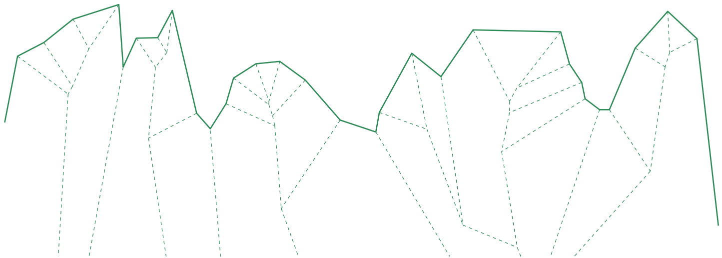
The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.



- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

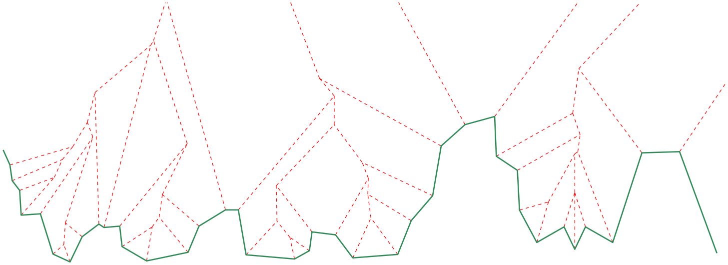
The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.



- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.

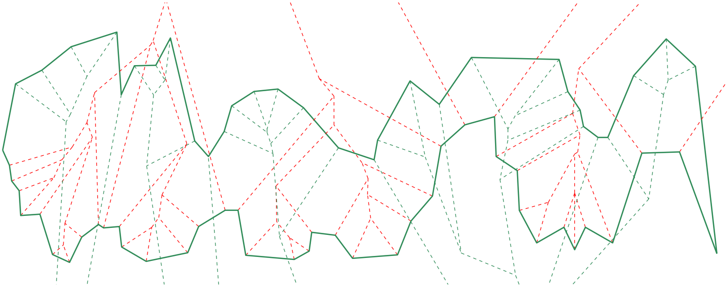


Straight Skeleton of Monotone Polygons

- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

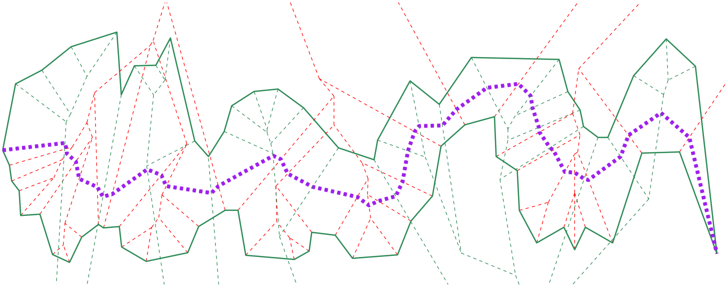
The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.



- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.

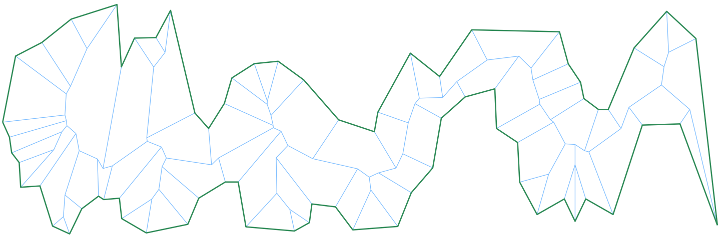


Straight Skeleton of Monotone Polygons

- The (positively weighted) straight skeleton of a convex polygon can be computed in $O(n)$ time. (Recall Thm. 144 by [Biedl et al. (2014)].)
- Can we also do better for other specific input classes?
- Yes!

Theorem 147 (Biedl et alii (2015))

The straight skeleton of an n-vertex monotone polygon can be computed in $O(n \log n)$ time.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

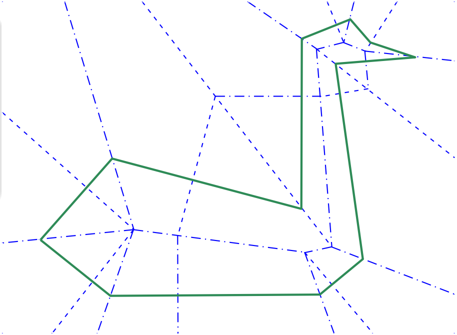
6 Skeletal Structures

- Voronoi Diagram of Points, Line Segments and Circular Arcs
- Straight Skeleton
- Applications
  - Origami and Cut-and-Fold Problems
  - Offsetting/Buffering
  - Generation of Tool Paths
  - Maximum Inscribed Circle
  - Finding a Gouge-Free Path
  - Approximation and Simplification of Curves
  - Topologically Consistent Watermarking
  - Roofs, Terrains, Chamfers and Fillets
  - Voronoi Diagrams in Structural Design

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 148 (Demaine et alii (1999))

Every polygon can be cut out of a sheet of paper by one straight cut after adequate folding. The folding creases can be constructed by a straight-skeleton-based algorithm.

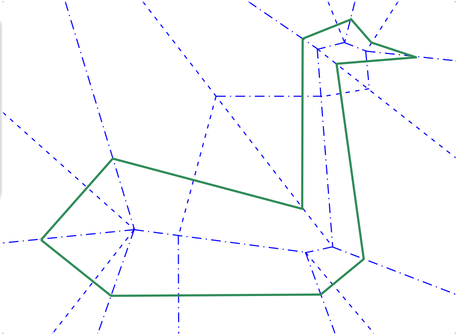


[Image courtesy of Erik Demaine]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 148 (Demaine et alii (1999))

Every polygon can be cut out of a sheet of paper by one straight cut after adequate folding. The folding creases can be constructed by a straight-skeleton-based algorithm.

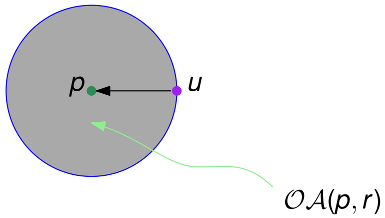


[Image courtesy of Erik Demaine]

- Other applications of straight skeletons comprise
  - design of pop-up cards [Sugihara (2013)];
  - shape reconstruction and contour interpolation [Oliva et alii (1996)];
  - computing centerlines of roads and area collapsing in GIS maps [Haunert&Sester (2008)].

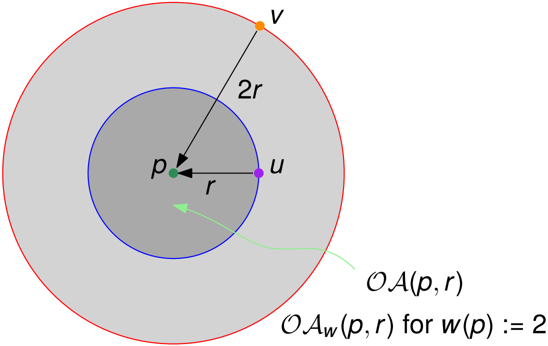
Unweighted and Weighted Offsets of Point

- The *unweighted offset area* $\mathcal{OA}(p, r)$ of the point p of $\mathbb{R}^2$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose unweighted distance $d(u, p)$ to p is at most r.

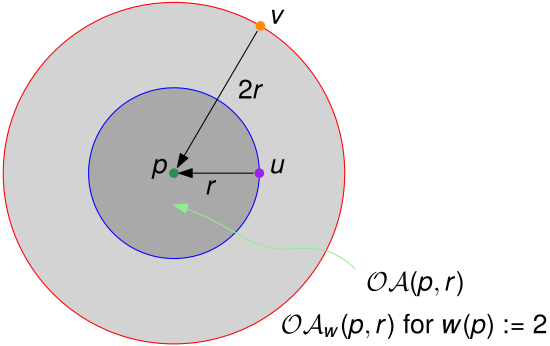


Unweighted and Weighted Offsets of Point

- The *unweighted offset area* $\mathcal{OA}(p, r)$ of the point p of $\mathbb{R}^2$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose unweighted distance $d(u, p)$ to p is at most r.
- The *weighted offset area* $\mathcal{OA}_w(p, r)$ of the point p of $\mathbb{R}^2$ for offset value $r \geq 0$ and weight $w(p) > 0$ is the set of all points v of $\mathbb{R}^2$ whose weighted distance $d_w(v, p)$ to p is at most r.



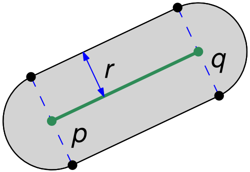
- The *unweighted offset area* $\mathcal{OA}(p, r)$ of the point p of $\mathbb{R}^2$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose unweighted distance $d(u, p)$ to p is at most r.
- The *weighted offset area* $\mathcal{OA}_w(p, r)$ of the point p of $\mathbb{R}^2$ for offset value $r \geq 0$ and weight $w(p) > 0$ is the set of all points v of $\mathbb{R}^2$ whose weighted distance $d_w(v, p)$ to p is at most r.
- For $w(p) := 1$ we get standard offsetting.



Unweighted and Weighted Offsets of Line Segment

- The *unweighted offset area* $\mathcal{OA}(\overline{pq}, r)$ of the straight-line segment $\overline{pq}$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose minimum unweighted distance to a point v of $\overline{pq}$ is at most r:

$$\mathcal{OA}(\overline{pq}, r) := \{u \in \mathbb{R}^2 \colon \min_{v \in \overline{pq}} d(u, v) \leq r\}$$

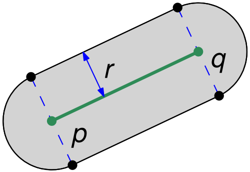


Unweighted and Weighted Offsets of Line Segment

- The *unweighted offset area* $\mathcal{OA}(\overline{pq}, r)$ of the straight-line segment $\overline{pq}$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose minimum unweighted distance to a point v of $\overline{pq}$ is at most r:

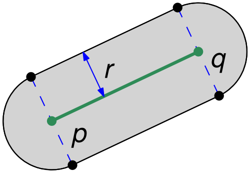
$$\mathcal{OA}(\overline{pq}, r) := \{u \in \mathbb{R}^2 : \min_{v \in \overline{pq}} d(u, v) \leq r\}$$

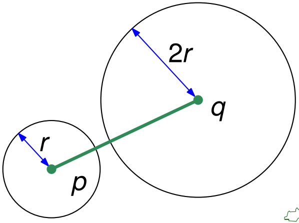
- This definition is generalized easily to circular arcs and to offsets of (curvilinear) polygons.



Unweighted and Weighted Offsets of Line Segment

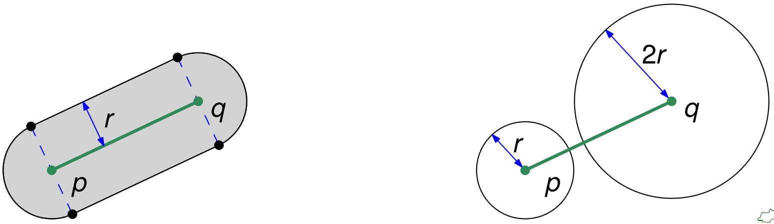
- Suppose that p has weight $w(p) > 0$ and q has weight $w(q) > 0$, possibly with $w(p) \neq w(q)$. (In the figure, $w(p) := 1$ and $w(q) := 2$.)





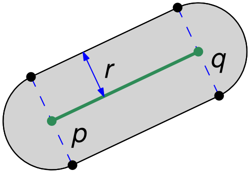
Unweighted and Weighted Offsets of Line Segment

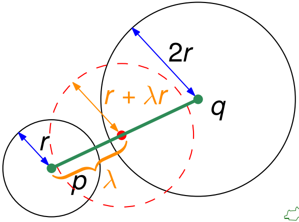
- Suppose that p has weight $w(p) > 0$ and q has weight $w(q) > 0$, possibly with $w(p) \neq w(q)$. (In the figure, $w(p) := 1$ and $w(q) := 2$.)
- For $0 \leq \lambda \leq 1$, the weight of a point $(1-\lambda)p + \lambda q$ on $\overline{pq}$ is given by $(1-\lambda)w(p) + \lambda w(q)$, i.e., by linear interpolation of the weights of p and q.



Unweighted and Weighted Offsets of Line Segment

- Suppose that p has weight $w(p) > 0$ and q has weight $w(q) > 0$, possibly with $w(p) \neq w(q)$. (In the figure, $w(p) := 1$ and $w(q) := 2$.)
- For $0 \leq \lambda \leq 1$, the weight of a point $(1-\lambda)p + \lambda q$ on $\overline{pq}$ is given by $(1-\lambda)w(p) + \lambda w(q)$, i.e., by linear interpolation of the weights of p and q.

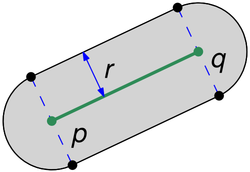


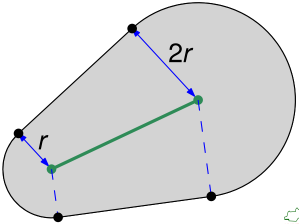


Unweighted and Weighted Offsets of Line Segment

- Suppose that p has weight $w(p) > 0$ and q has weight $w(q) > 0$, possibly with $w(p) \neq w(q)$. (In the figure, $w(p) := 1$ and $w(q) := 2$.)
- For $0 \leq \lambda \leq 1$, the weight of a point $(1-\lambda)p + \lambda q$ on $\overline{pq}$ is given by $(1-\lambda)w(p) + \lambda w(q)$, i.e., by linear interpolation of the weights of p and q.
- Then the *variable-radius offset area* $\mathcal{OA}_v(\overline{pq}, r)$ of the straight-line segment $\overline{pq}$ for offset value $r \geq 0$ is the set of all points u of $\mathbb{R}^2$ whose minimum weighted distance to a (weighted) point v of $\overline{pq}$ is at most r:

$$\mathcal{OA}_v(\overline{pq}, r) := \{u \in \mathbb{R}^2 : \min_{v \in \overline{pq}} d_w(u, v) \leq r\}$$

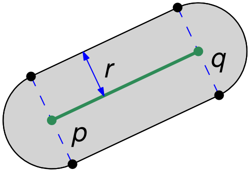


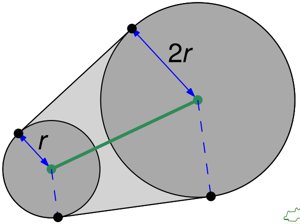


Unweighted and Weighted Offsets of Line Segment

[Held&Huber&Palfrader (2016)]

The *variable-radius offset area* $\mathcal{OA}_v(\overline{pq}, r)$ of the straight-line segment $\overline{pq}$ for offset value $r \geq 0$ is given by the convex hull of $\mathcal{OA}_w(p, r) \cup \mathcal{OA}_w(q, r)$. Thus, $\mathcal{OA}_v(\overline{pq}, r)$ is bounded by up to two straight-line segments and up to two circular arcs.

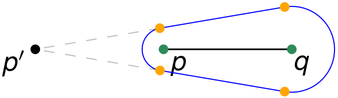




Unweighted and Weighted Offsets of Line Segment

[Held&Huber&Palfrader (2016)]

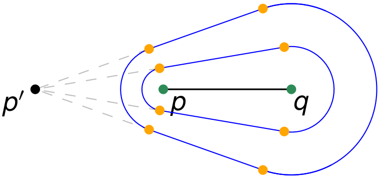
All supporting lines of segments of $\mathcal{OA}_v(\overline{pq}, r)$ meet in a point p'.



Unweighted and Weighted Offsets of Line Segment

[Held&Huber&Palfrader (2016)]

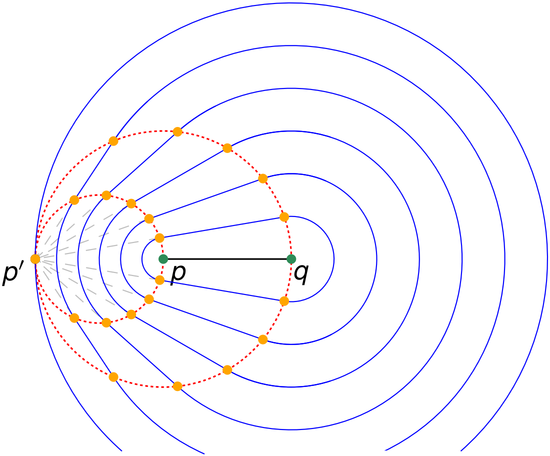
All supporting lines of segments of $\mathcal{OA}_v(\overline{pq}, r)$ meet in a point p'.



Unweighted and Weighted Offsets of Line Segment

[Held&Huber&Palfrader (2016)]

All supporting lines of segments of $\mathcal{OA}_v(\overline{pq}, r)$ meet in a point p'.



Minkowski Sum and Difference

- Let A, B be sets, and a, b denote points of A respectively B.
- We define the translation of A by the vector b as

$$A_b := \{a + b : a \in A\}.$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Minkowski Sum and Difference

- Let A, B be sets, and a, b denote points of A respectively B.
- We define the translation of A by the vector b as

$$A_b := \{a + b : a \in A\}.$$

- [Hadwiger (1950)]: The *Minkowski sum* of A and B is defined as

$$A \oplus B := \bigcup_{b \in B} A_b.$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Minkowski Sum and Difference

- Let A, B be sets, and a, b denote points of A respectively B.
- We define the translation of A by the vector b as

$$A_b := \{a + b : a \in A\}.$$

- [Hadwiger (1950)]: The *Minkowski sum* of A and B is defined as

$$A \oplus B := \bigcup_{b \in B} A_b.$$

- [Hadwiger (1950)]: The *Minkowski difference* of A and B is defined as

$$A \ominus B := \bigcap_{b \in B} A_{-b}.$$

Minkowski Sum and Difference

- Let A, B be sets, and a, b denote points of A respectively B.
- We define the translation of A by the vector b as

$$A_b := \{a + b : a \in A\}.$$

- [Hadwiger (1950)]: The *Minkowski sum* of A and B is defined as

$$A \oplus B := \bigcup_{b \in B} A_b.$$

- [Hadwiger (1950)]: The *Minkowski difference* of A and B is defined as

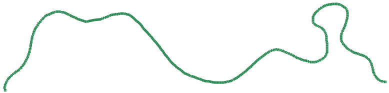
$$A \ominus B := \bigcap_{b \in B} A_{-b}.$$

- Note: In general, $(A \oplus B) \ominus B \neq A$.

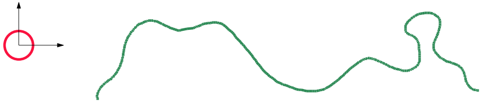
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Minkowski Sum for Offsetting

- Let A be a curve,

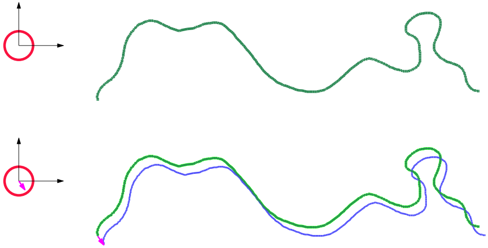


- Let A be a curve, and B be a circular disk centered at the origin. What is $A \oplus B$?



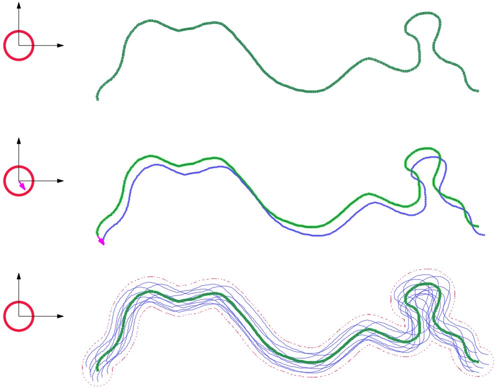
Minkowski Sum for Offsetting

- Let A be a curve, and B be a circular disk centered at the origin. What is $A \oplus B$?



Minkowski Sum for Offsetting

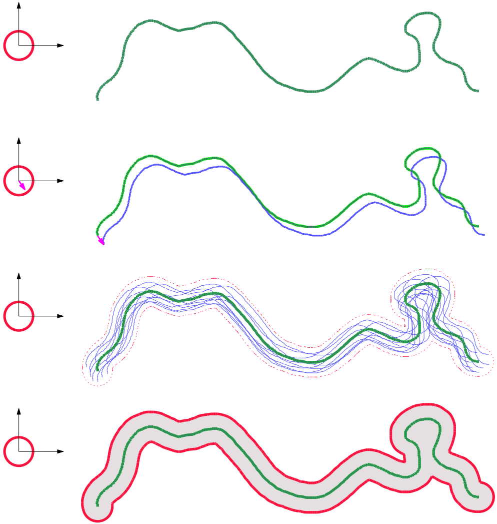
- Let A be a curve, and B be a circular disk centered at the origin. What is $A \oplus B$?



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Minkowski Sum for Offsetting

- Let A be a curve, and B be a circular disk centered at the origin. What is $A \oplus B$?

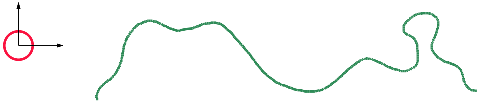


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Minkowski Sum for Offsetting

Lemma 149

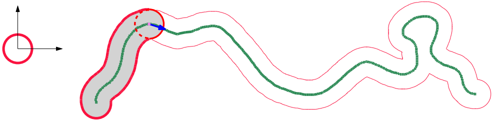
The Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is the area swept by a disk with radius r whose center is moved along A. That is, it is the (unweighted) offset area of A for offset distance r.



Minkowski Sum for Offsetting

Lemma 149

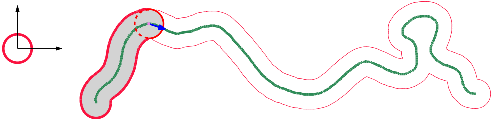
The Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is the area swept by a disk with radius r whose center is moved along A. That is, it is the (unweighted) offset area of A for offset distance r.



Minkowski Sum for Offsetting

Lemma 149

The Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is the area swept by a disk with radius r whose center is moved along A. That is, it is the (unweighted) offset area of A for offset distance r.

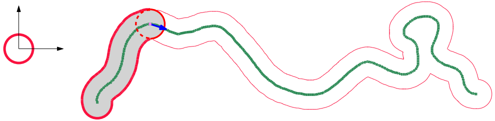


Lemma 150

The boundary of the Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is traced out by the center of a disk with radius r that is "rolled" along A.

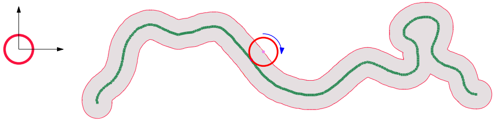
Lemma 149

The Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is the area swept by a disk with radius r whose center is moved along A. That is, it is the (unweighted) offset area of A for offset distance r.



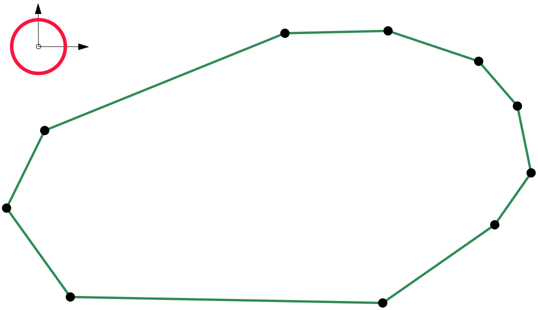
Lemma 150

The boundary of the Minkowski sum $A \oplus B$ of a curve A and a circular disk B (with radius r) centered at the origin is traced out by the center of a disk with radius r that is "rolled" along A.



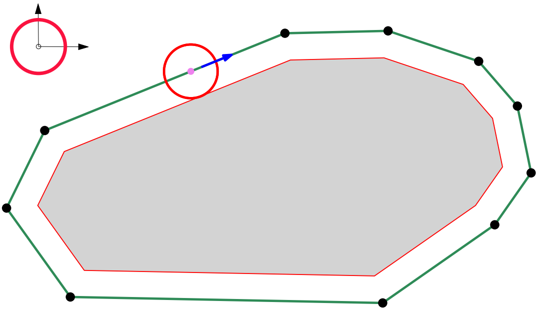
Minkowski Difference for Offsetting

- Let A be a polygon, and B be a circular disk centered at the origin. What is $A \ominus B$?



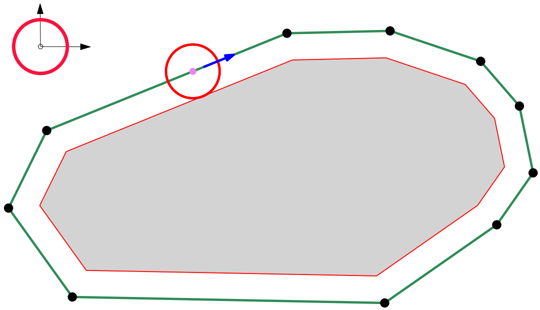
Minkowski Difference for Offsetting

- Let A be a polygon, and B be a circular disk centered at the origin. What is $A \ominus B$?



Minkowski Difference for Offsetting

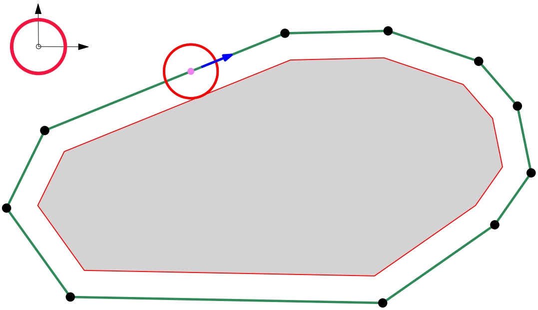
- Let A be a polygon, and B be a circular disk centered at the origin. What is $A \ominus B$?



- Hence, it is the interior offset area of the polygon.

Minkowski Difference for Offsetting

- Let A be a polygon, and B be a circular disk centered at the origin. What is $A \ominus B$?



- Hence, it is the interior offset area of the polygon.
- Offsets, i.e., Minkowski sums and differences of an area A with a circular disk B centered at the origin, are also called *buffers* (in GIS) and *dilation*/*erosion* (in image processing).

Buffering in Geographical Information Systems

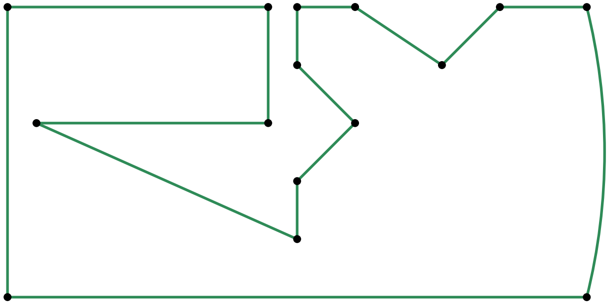
- Sample GIS application: Identify the portion of the territorial waters of Malta that is within some nautical miles of the baseline (coast) of Malta.

Buffering in Geographical Information Systems

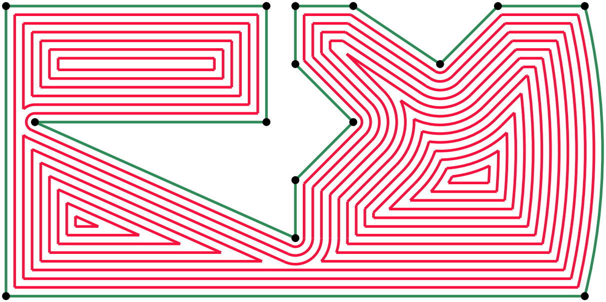
- Sample GIS application: Identify the portion of the territorial waters of Malta that is within some nautical miles of the baseline (coast) of Malta.

Computation of Offset Patterns

- How can we compute offset patterns reliably and efficiently?

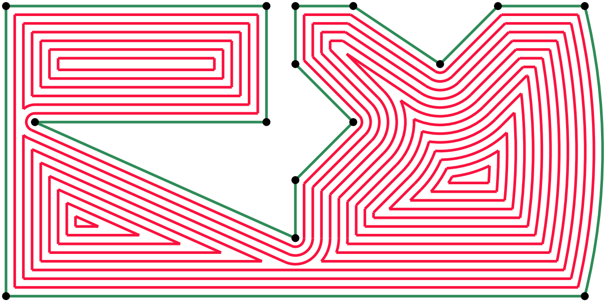


- How can we compute offset patterns reliably and efficiently?
- Note: The boundary of an offset may contain circular arcs even if the input is purely polygonal.



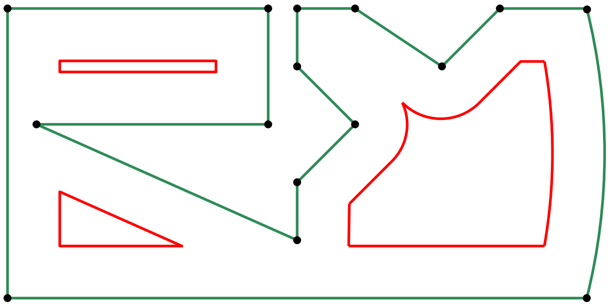
Computation of Offset Patterns

- How can we compute offset patterns reliably and efficiently?
- Note: The boundary of an offset may contain circular arcs even if the input is purely polygonal.
- Note: Offsetting may cause topological changes!

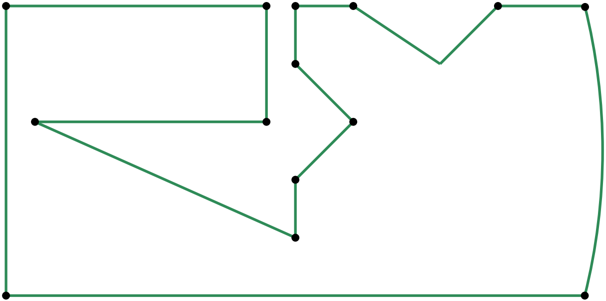


Computation of Offset Patterns

- How can we compute offset patterns reliably and efficiently?
- Note: The boundary of an offset may contain circular arcs even if the input is purely polygonal.
- Note: Offsetting may cause topological changes!
- How can we compute even just one individual offset?

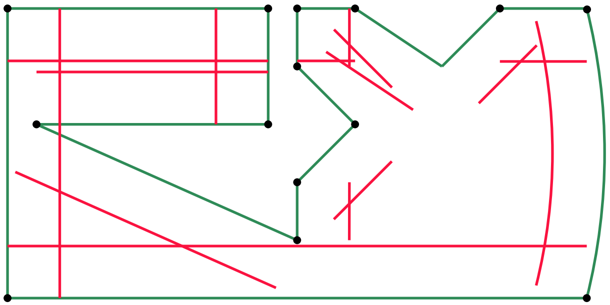


Conventional Offsetting

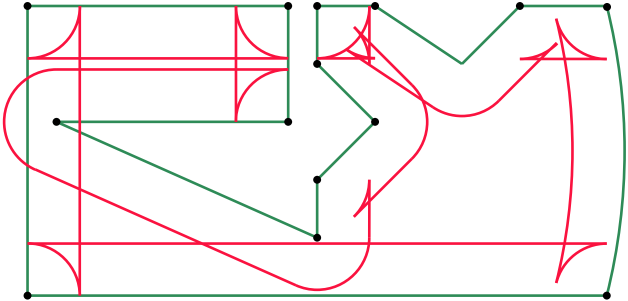


Conventional Offsetting

1. First, one computes offset elements for every input element.

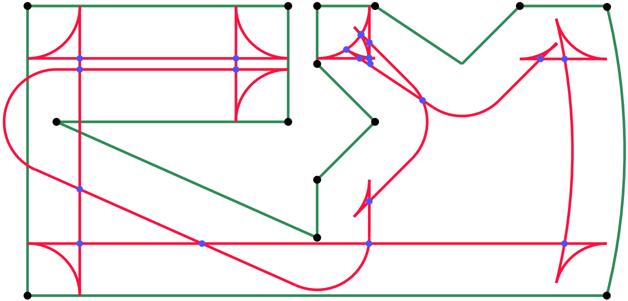


1. First, one computes offset elements for every input element.
2. In order to get one closed loop, trimming arcs are inserted.



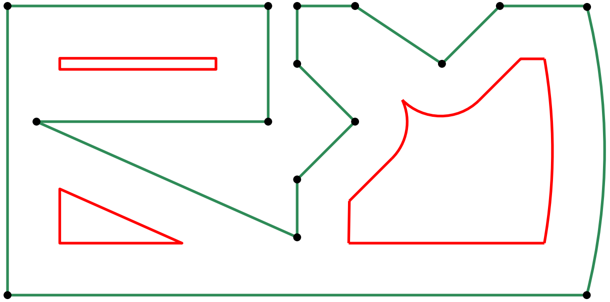
Conventional Offsetting

1. First, one computes offset elements for every input element.
2. In order to get one closed loop, trimming arcs are inserted.
3. Next, all self-intersections are determined.

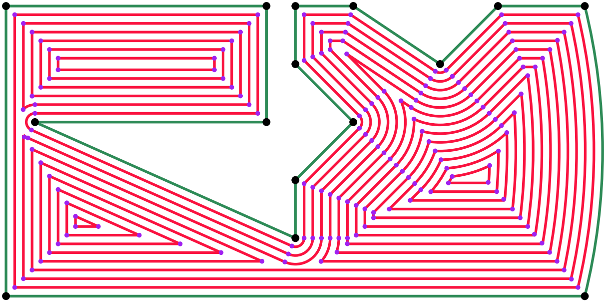


Conventional Offsetting

1. First, one computes offset elements for every input element.
2. In order to get one closed loop, trimming arcs are inserted.
3. Next, all self-intersections are determined.
4. Finally, all incorrect loops of the offset are removed.

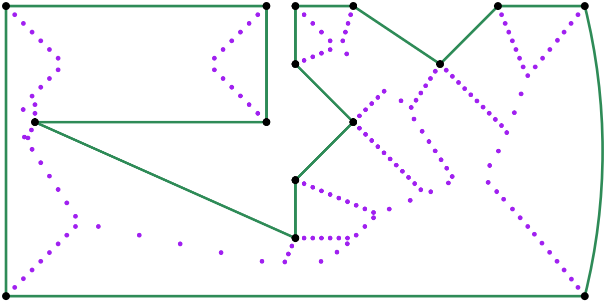


- We start with analyzing the positions of the end-points of the offset segments.

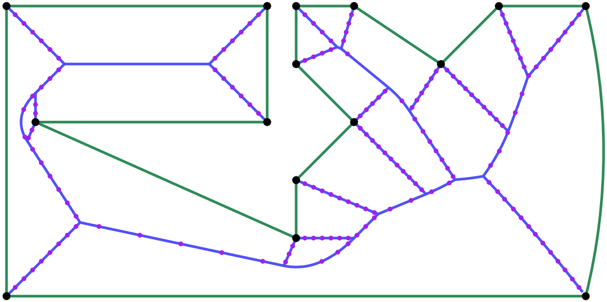


Voronoi-Based Offsetting

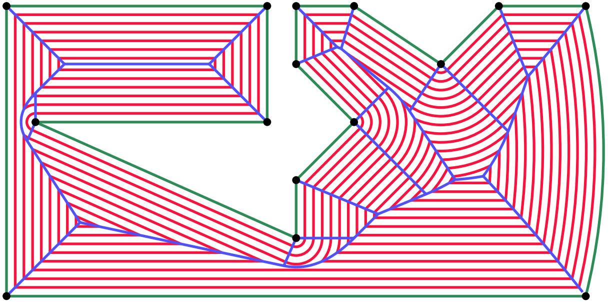
- We start with analyzing the positions of the end-points of the offset segments.
- This looks familiar!

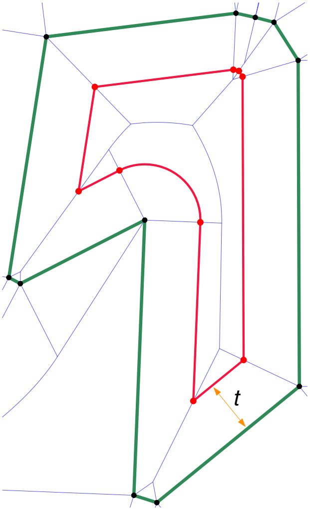


- We start with analyzing the positions of the end-points of the offset segments.
- This looks familiar!
- Indeed, all end-points of offset segments lie on the Voronoi diagram!



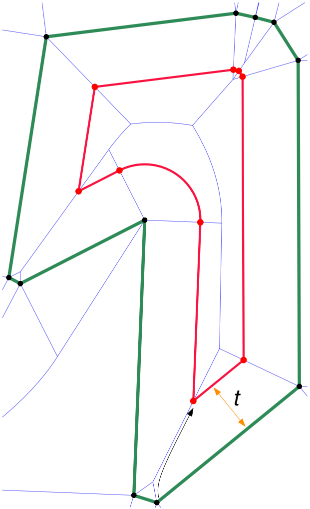
- We start with analyzing the positions of the end-points of the offset segments.
- This looks familiar!
- Indeed, all end-points of offset segments lie on the Voronoi diagram!
- A linear-time scan of the Voronoi diagram reveals the end-points of one offset.





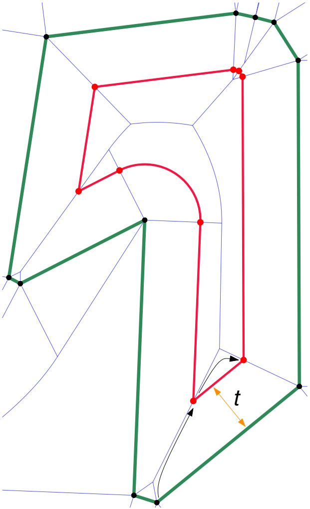
Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.



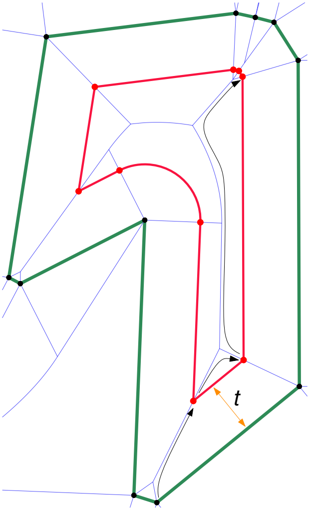
Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.



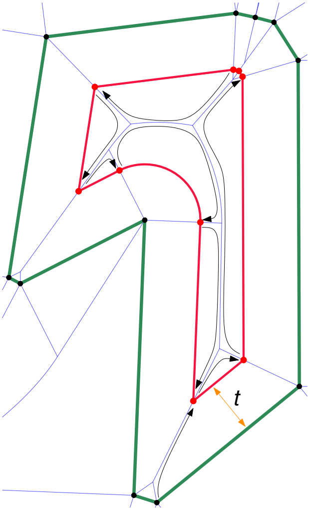
Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.



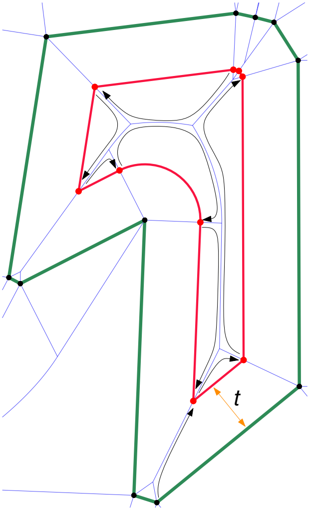
Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.



Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.

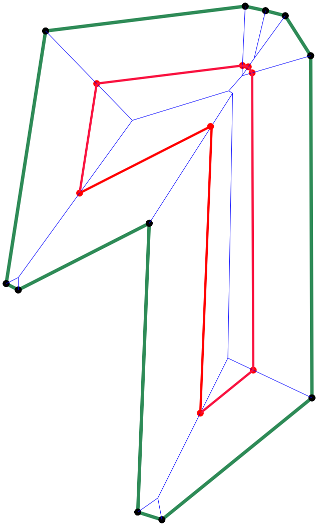


Theorem 151 (Persson (1978), Held (1991))

Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. If $\mathcal{VD}(S)$ is known then all offset curves of S at offset t can be determined in $O(n)$ time.

Corollary 152

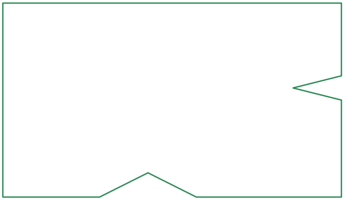
Let S be an admissible set of sites, and $t \in \mathbb{R}^+$. Then all offset curves of S at offset t can be determined in $O(n \log n)$ time.



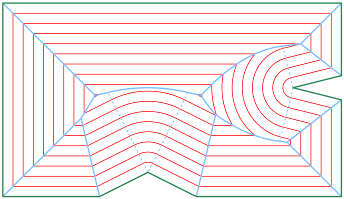
Theorem 153 (Palfrader&Held (2014))

Let S a PSLG, and $t \in \mathbb{R}^+$. If $\mathcal{SK}(S)$ is known then all mitered offset curves of S at offset t can be determined in $O(n)$ time.

Comparison of Offsets: Constant-Radius vs. Mitered Offsets



Comparison of Offsets: Constant-Radius vs. Mitered Offsets

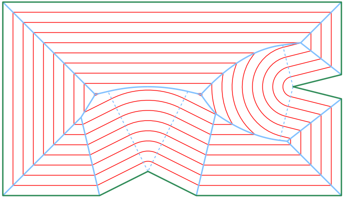


Voronoi diagram and rounded offsets

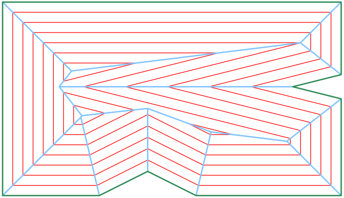
Comparison of Offsets: Constant-Radius vs. Mitered Offsets

Held&Palfrader (2015)

Computing just one mitered offset via an SK is faster than standard mitered offsetting.



Voronoi diagram and rounded offsets

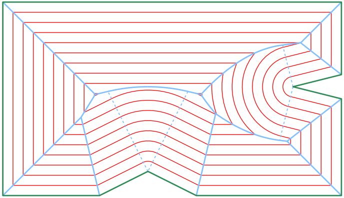


Straight skeleton and mitered offsets

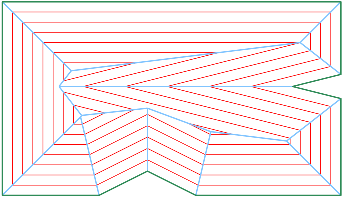
Comparison of Offsets: Constant-Radius vs. Mitered Offsets

Held&Palfrader (2015)

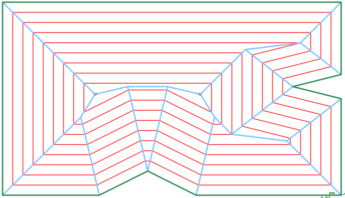
Computing just one mitered offset via an SK is faster than standard mitered offsetting.



Voronoi diagram and rounded offsets



Straight skeleton and mitered offsets

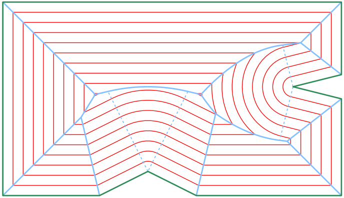


Straight skeleton and beveled offsets

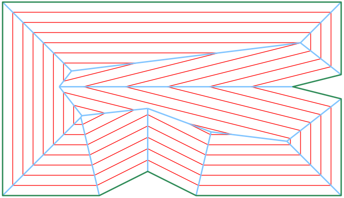
Comparison of Offsets: Constant-Radius vs. Mitered Offsets

Held&Palfrader (2015)

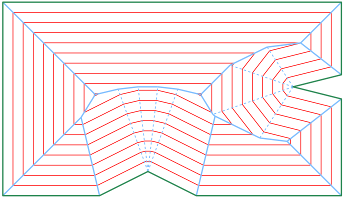
Computing just one mitered offset via an SK is faster than standard mitered offsetting.



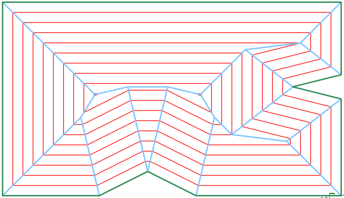
Voronoi diagram and rounded offsets



Straight skeleton and mitered offsets



Linear axis and multi-segment bevels



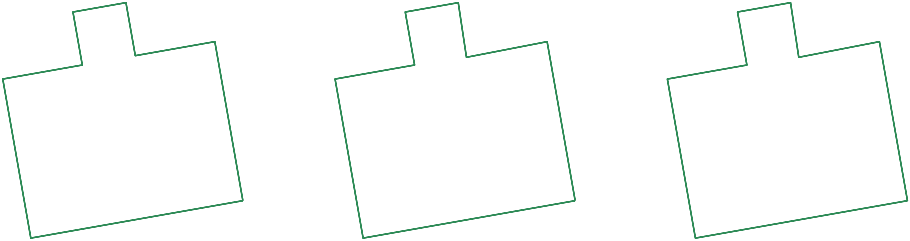
Straight skeleton and beveled offsets

Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton drastically.

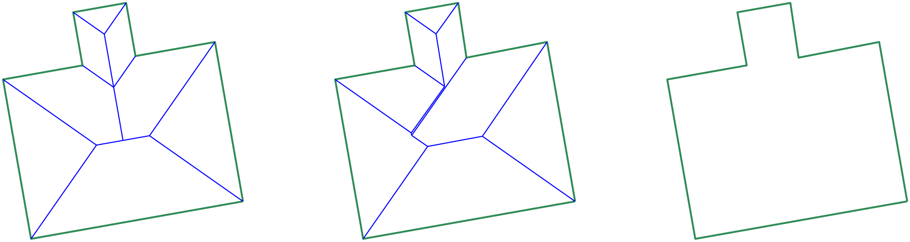
Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton drastically.
- From left to right:
  - symmetric shape
  - perturbed shape
  - perturbed shape



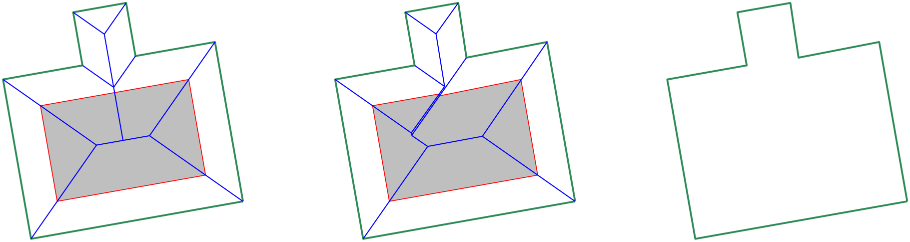
Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton drastically.
- From left to right:
  - symmetric shape and its straight skeleton
  - perturbed shape and its straight skeleton
  - perturbed shape



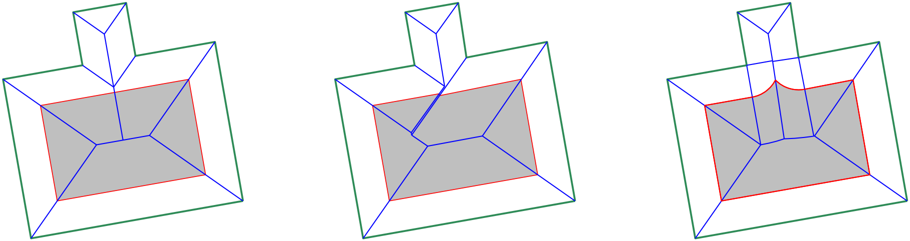
Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton drastically.
- From left to right:
  - symmetric shape and its straight skeleton and mitered offset,
  - perturbed shape and its straight skeleton and mitered offset,
  - perturbed shape



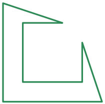
Sensitivity of Straight Skeleton to Perturbations of the Input

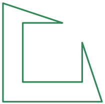
- Even a small perturbation of the input may suffice to change the straight skeleton drastically.
- From left to right:
  - symmetric shape and its straight skeleton and mitered offset,
  - perturbed shape and its straight skeleton and mitered offset,
  - perturbed shape and its Voronoi diagram and constant-radius offset.

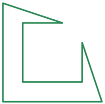


Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton and the resulting mitered offsets drastically.
- From left to right:
  - symmetric shape
  - perturbed shape
  - perturbed shape

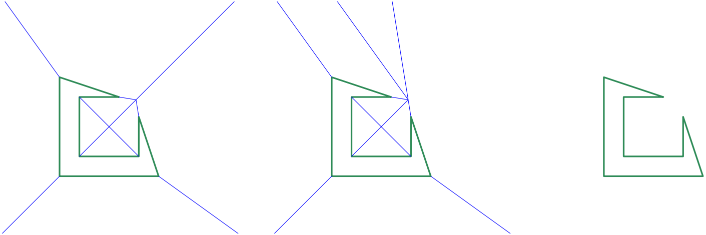






UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

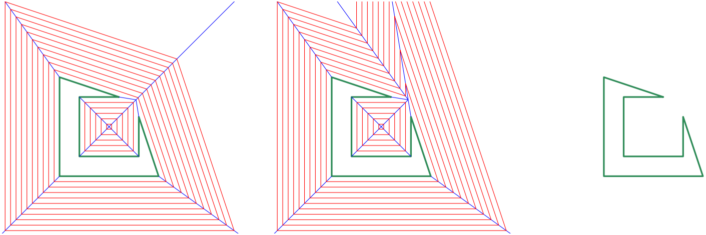
- Even a small perturbation of the input may suffice to change the straight skeleton and the resulting mitered offsets drastically.
- From left to right:
  - symmetric shape and its straight skeleton
  - perturbed shape and its straight skeleton
  - perturbed shape



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sensitivity of Straight Skeleton to Perturbations of the Input

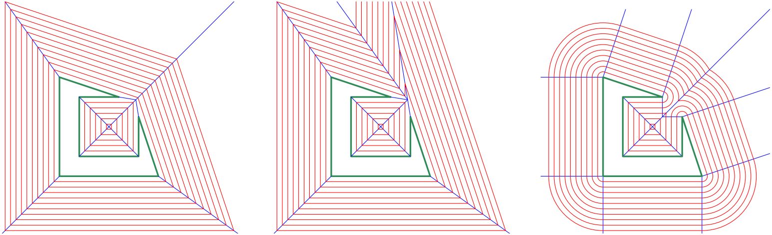
- Even a small perturbation of the input may suffice to change the straight skeleton and the resulting mitered offsets drastically.
- From left to right:
  - symmetric shape and its straight skeleton and mitered offsets,
  - perturbed shape and its straight skeleton and mitered offsets,
  - perturbed shape



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sensitivity of Straight Skeleton to Perturbations of the Input

- Even a small perturbation of the input may suffice to change the straight skeleton and the resulting mitered offsets drastically.
- From left to right:
  - symmetric shape and its straight skeleton and mitered offsets,
  - perturbed shape and its straight skeleton and mitered offsets,
  - perturbed shape and its Voronoi diagram and constant-radius offsets.



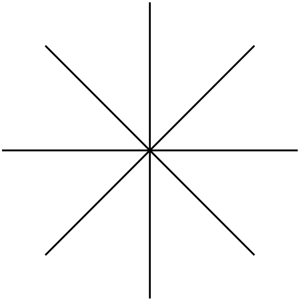
Variable-Radius Offsets: Brush Stroke

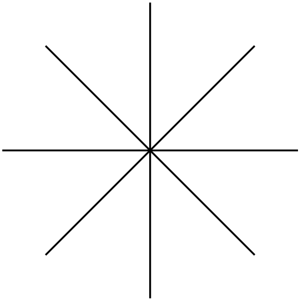
Uniform pressure

Constant uniform width of the shape.

Non-uniform pressure

Varying width of the shape.





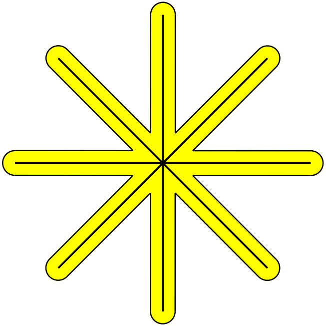
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Uniform pressure

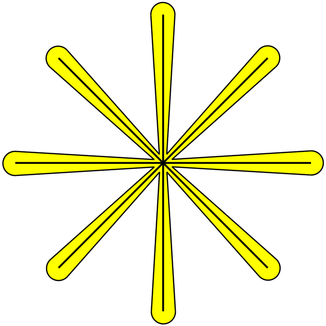
Constant uniform width of the shape.

Non-uniform pressure

Varying width of the shape.



constant-radius offset



variable-radius offset

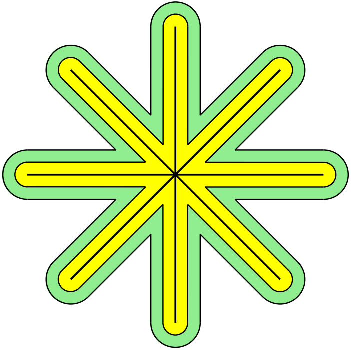
Variable-Radius Offsets: Brush Stroke

Uniform pressure

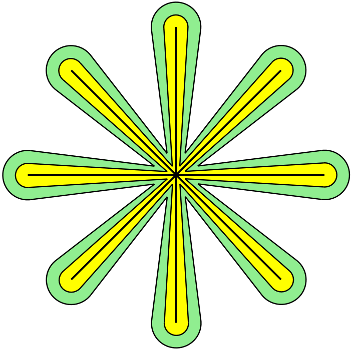
Constant uniform width of the shape.

Non-uniform pressure

Varying width of the shape.



constant-radius offset



variable-radius offset

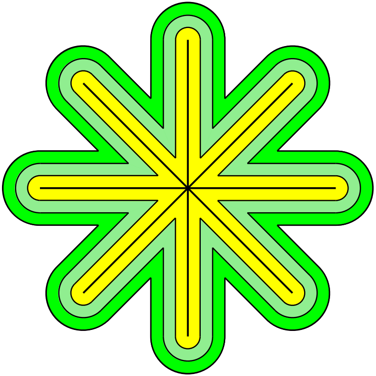
Variable-Radius Offsets: Brush Stroke

Uniform pressure

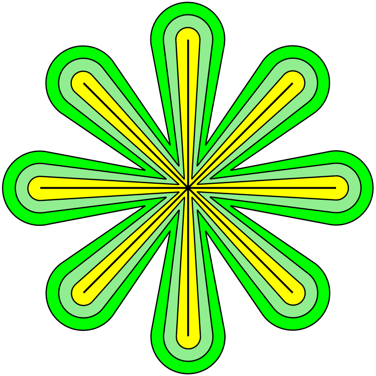
Constant uniform width of the shape.

Non-uniform pressure

Varying width of the shape.



constant-radius offset



variable-radius offset

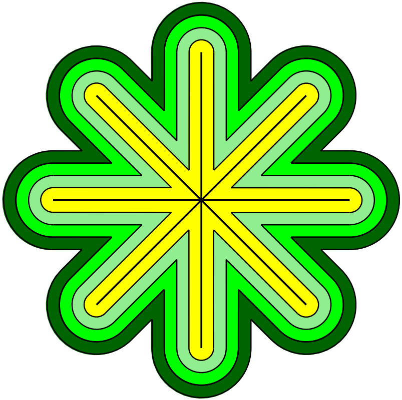
Variable-Radius Offsets: Brush Stroke

Uniform pressure

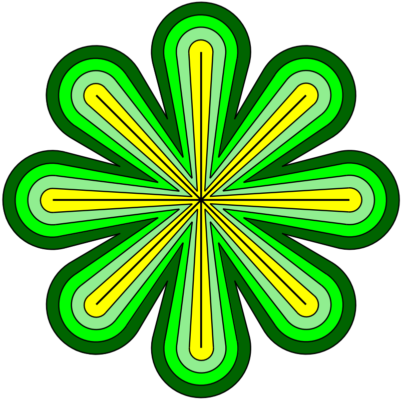
Constant uniform width of the shape.

Non-uniform pressure

Varying width of the shape.



constant-radius offset



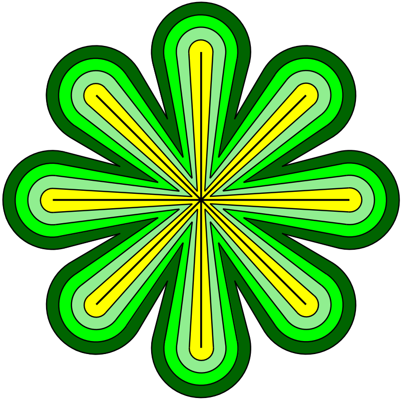
variable-radius offset

Brush stroke

Standard application in computer-assisted calligraphy.

Non-uniform pressure

Varying width of the shape.



variable-radius offset

Variable-Radius Offsets: Applications

Brush stroke

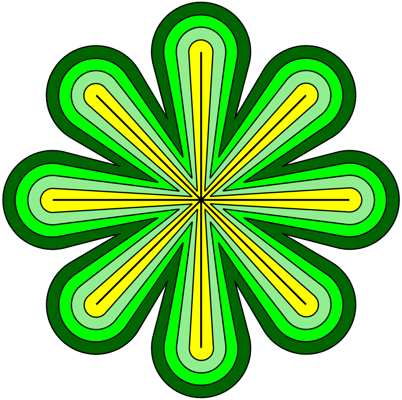
Standard application in computer-assisted calligraphy.

Shoe and garment design

- Ornamentary stitches need not run in a perfectly parallel manner.
- Scaling a basic shape need not necessarily be uniform.

Non-uniform pressure

Varying width of the shape.



variable-radius offset

Brush stroke

Standard application in computer-assisted calligraphy.

Shoe and garment design

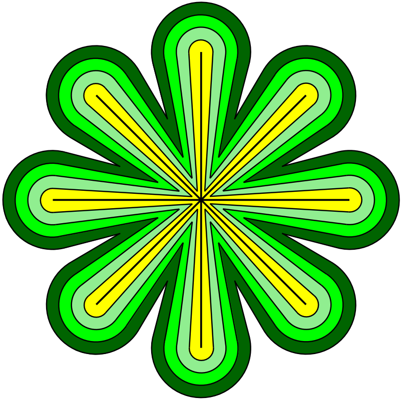
- Ornamentary stitches need not run in a perfectly parallel manner.
- Scaling a basic shape need not necessarily be uniform.

Image manipulation

Patent “Retrograde Curve Filtering for Variable Offset Curves” granted to Adobe Inc. in April 2014.

Non-uniform pressure

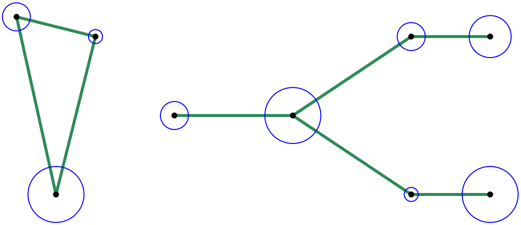
Varying width of the shape.



variable-radius offset

Held&Huber&Palfrader (2015)

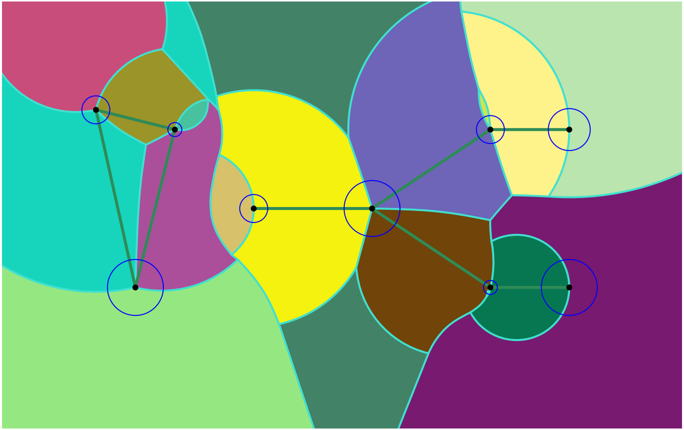
Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



Variable-Radius Offsets and Variable-Radius Voronoi Diagram

Held&Huber&Palfrader (2015)

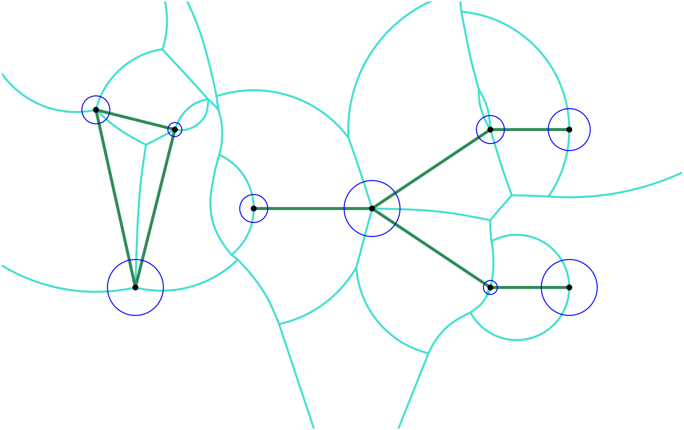
Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



Variable-Radius Offsets and Variable-Radius Voronoi Diagram

Held&Huber&Palfrader (2015)

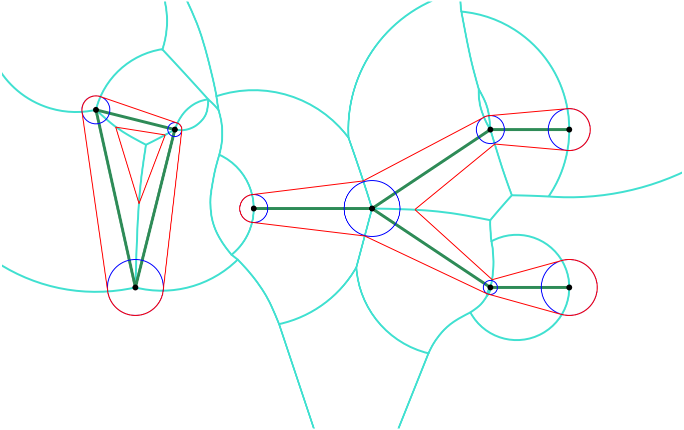
Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



Variable-Radius Offsets and Variable-Radius Voronoi Diagram

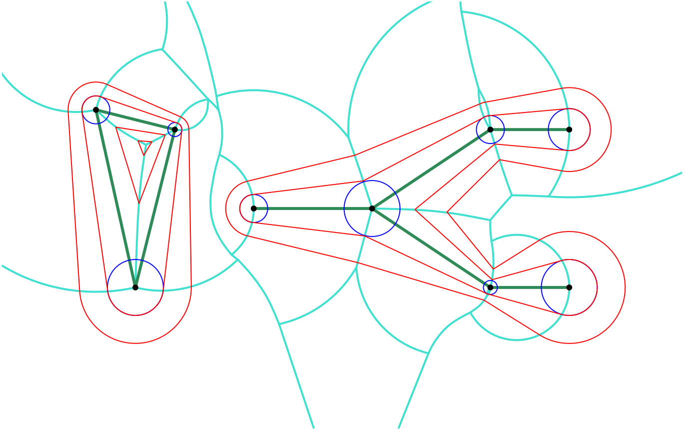
Held&Huber&Palfrader (2015)

Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



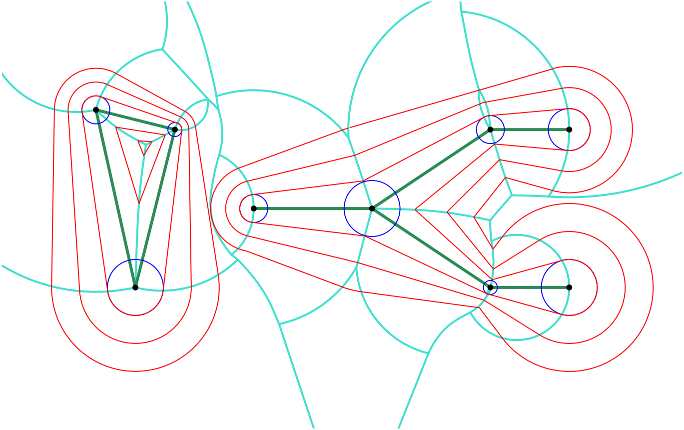
Held&Huber&Palfrader (2015)

Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



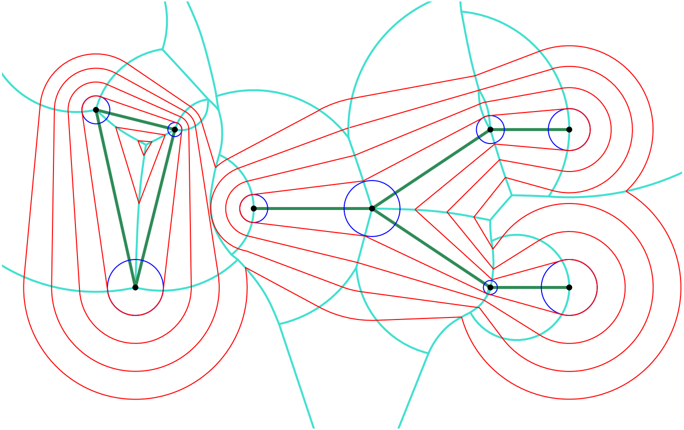
Held&Huber&Palfrader (2015)

Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



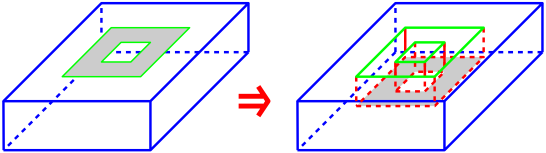
Held&Huber&Palfrader (2015)

Assign non-negative weights to the vertices of a PSLG, and interpolate weights linearly along segments. Then the variable-radius Voronoi diagram induced by the resulting weighted distance supports variable-radius offsets.



Pocket Machining

Pocket: Interior recess that is cut into the surface of a workpiece.

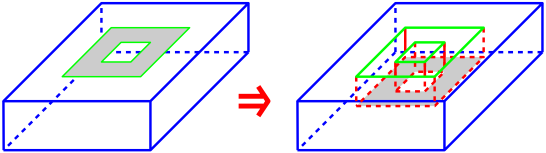


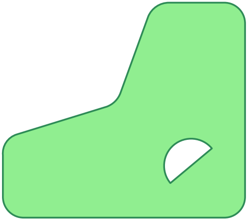
Tool Paths for Pocket Machining

Pocket Machining

Pocket: Interior recess that is cut into the surface of a workpiece.

Tool: Can be regarded as a cylinder that rotates.

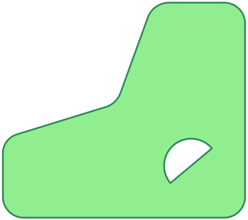




Geometry of a pocket

- 2D area,
- straight-line segments and circular arcs as boundary elements,
- may contain islands.

Tool Paths for Pocket Machining



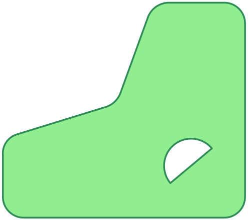
Geometry of a pocket

- 2D area,
- straight-line segments and circular arcs as boundary elements,
- may contain islands.

Geometry of a tool

- circular disk.

Tool Paths for Pocket Machining



Geometry of a pocket

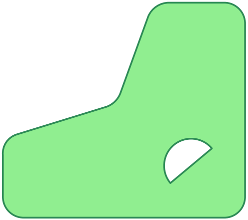
- 2D area,
- straight-line segments and circular arcs as boundary elements,
- may contain islands.

Geometry of a tool

- circular disk.

The goal is . . .

- . . . to compute a “good” tool path.



Geometry of a pocket

- 2D area,
- straight-line segments and circular arcs as boundary elements,
- may contain islands.

Geometry of a tool

- circular disk.

The goal is . . .

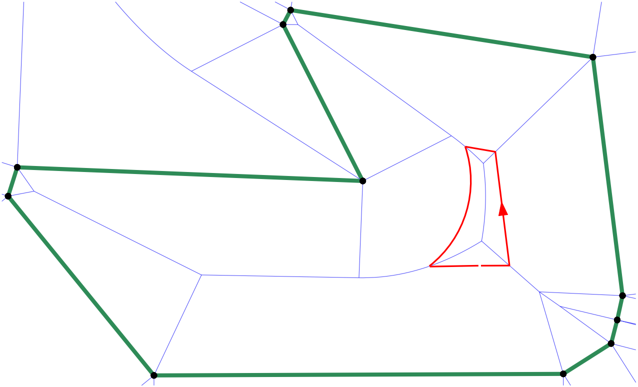
- . . . to compute a "good" tool path.

- Similar path planning problems arise in many other applications that require "coverage" of an area by a disk-shaped object, e.g., layered manufacturing, spray painting, aerial surveillance.

Voronoi-Based Generation of Tool Path

Persson (1978), Held (1991)

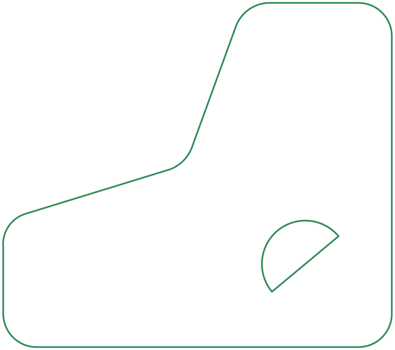
- Family of offset curves forms a tool path.
- Tool path computed by means of Voronoi diagram.



Voronoi-Based Generation of Tool Path

Pros of offset-based machining

- Offset curves can be computed easily (based on Voronoi diagram).
- Reasonably short tool path.



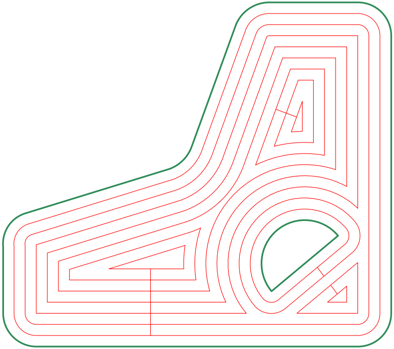
Voronoi-Based Generation of Tool Path

Pros of offset-based machining

- Offset curves can be computed easily (based on Voronoi diagram).
- Reasonably short tool path.

Cons of offset-based machining

- Sharp corners.
- Highly varying material removal rate.
- Might require tool retractions.
- Not suitable for high-speed machining.



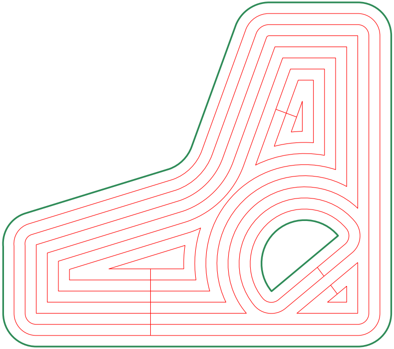
Voronoi-Based Generation of Tool Path

Pros of offset-based machining

- Offset curves can be computed easily (based on Voronoi diagram).
- Reasonably short tool path.

Cons of offset-based machining

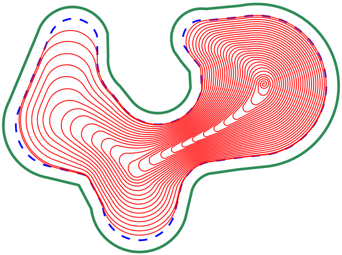
- Sharp corners.
- Highly varying material removal rate.
- Might require tool retractions.
- Not suitable for high-speed machining.



High-speed machining (HSM)

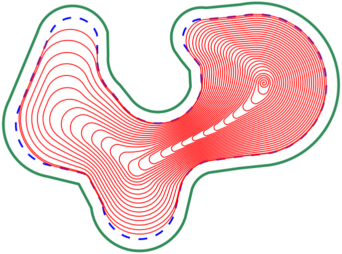
Faster tool movement requires

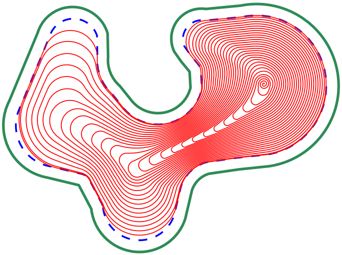
- smooth tool paths,
- low variation of material removal rate.

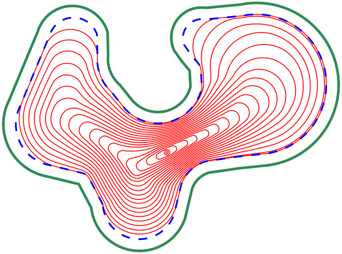


Held&Spielberger (2009)

- Smooth spiral path.
- Handle general areas without islands.







Held&Spielberger (2009)

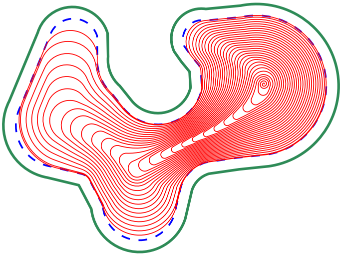
- Smooth spiral path.
- Handle general areas without islands.

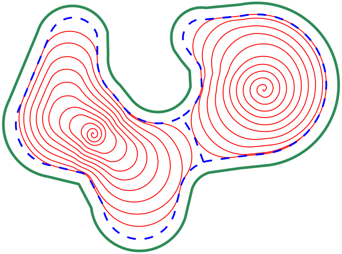
Held&Spielberger (2013)

- Optimization of the start point of the spiral tool path.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Voronoi-Based Generation of Smooth Spiral Paths





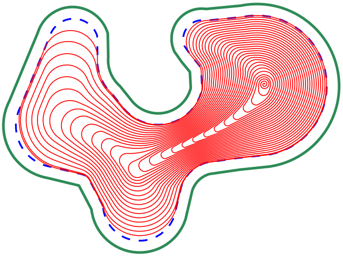
Held&Spielberger (2009)

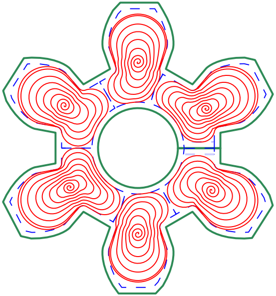
- Smooth spiral path.
- Handle general areas without islands.

Held&Spielberger (2013)

- Optimization of the start point of the spiral tool path.
- Decomposition of "complex" areas.

Voronoi-Based Generation of Smooth Spiral Paths





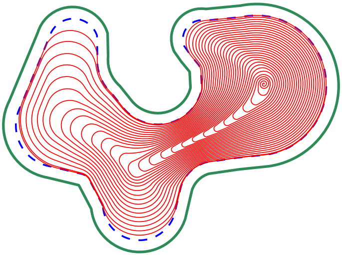
Held&Spielberger (2009)

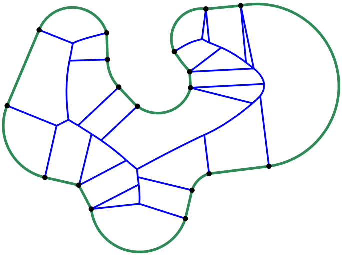
- Smooth spiral path.
- Handle general areas without islands.

Held&Spielberger (2013)

- Optimization of the start point of the spiral tool path.
- Decomposition of "complex" areas.
- Handle areas with islands.

Voronoi-Based Generation of Smooth Spiral Paths





Held&Spielberger (2009)

- Smooth spiral path.
- Handle general areas without islands.

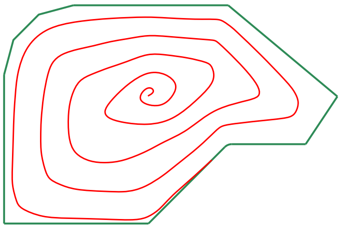
Held&Spielberger (2013)

- Optimization of the start point of the spiral tool path.
- Decomposition of "complex" areas.
- Handle areas with islands.

Algorithmic vehicle

- Voronoi diagram of area boundary.

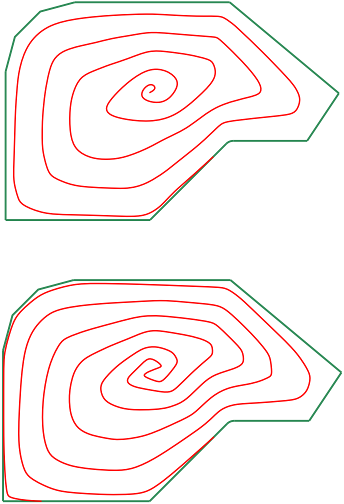
Voronoi-Based Generation of Smooth Multi-Spiral Paths



Held&de Lorenzo (2018)

- Simplified approach to computing a smooth spiral path.

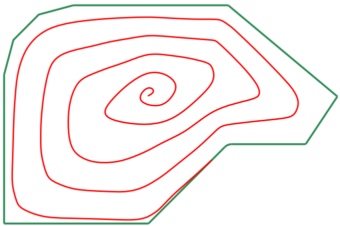
Voronoi-Based Generation of Smooth Multi-Spiral Paths

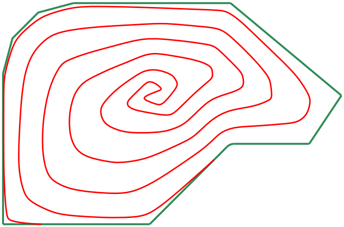


Held&de Lorenzo (2018)

- Simplified approach to computing a smooth spiral path.
- Double spiral that starts and ends on the boundary.

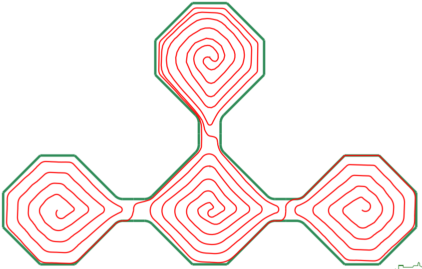
Voronoi-Based Generation of Smooth Multi-Spiral Paths





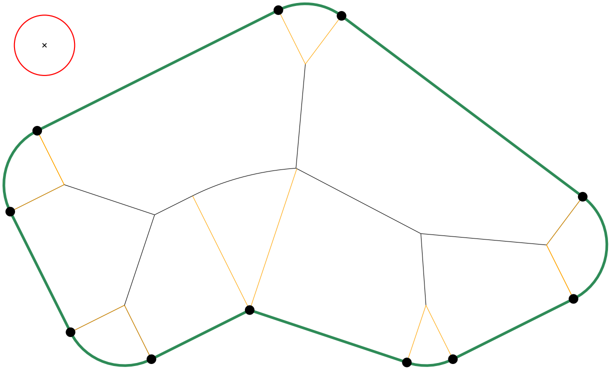
Held&de Lorenzo (2018)

- Simplified approach to computing a smooth spiral path.
- Double spiral that starts and ends on the boundary.
- Double spirals linked to one multi-spiral path.

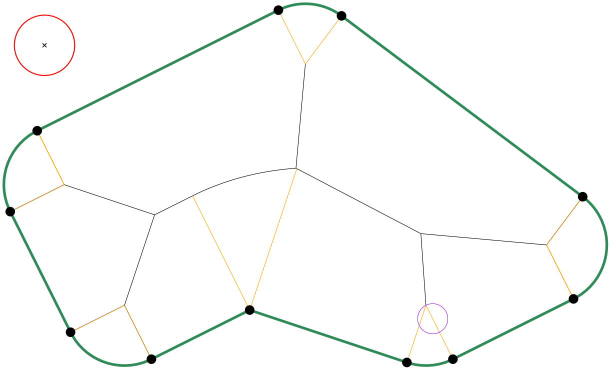


Paths for High-Speed Machining

- [Elber&Cohen&Drake (2005)]: “Medial axis transform toward high-speed machining of pockets” (MATHSM). They use the medial axis of a pocket to compute clearance disks that form “machining circles”.

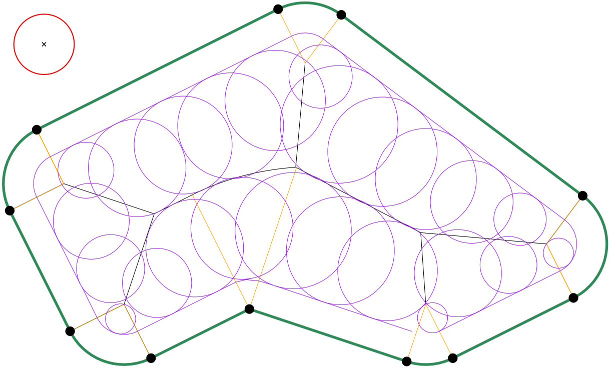


- [Elber&Cohen&Drake (2005)]: “Medial axis transform toward high-speed machining of pockets” (MATHSM). They use the medial axis of a pocket to compute clearance disks that form “machining circles”.
- The path is an alternating series of “machining circles” and tangential “transition elements” between pairs of machining circles.

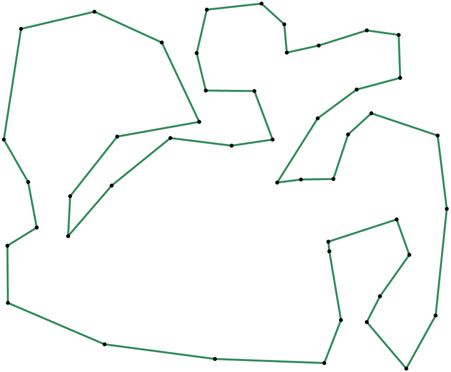


Paths for High-Speed Machining

- [Elber&Cohen&Drake (2005)]: “Medial axis transform toward high-speed machining of pockets” (MATHSM). They use the medial axis of a pocket to compute clearance disks that form “machining circles”.
- The path is an alternating series of “machining circles” and tangential “transition elements” between pairs of machining circles.
- [Held&Pfeiffer (2024)]: MATHSM extended such that the engagement angle is controlled.

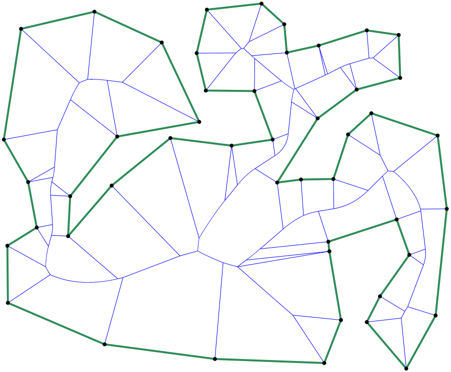


Maximum Inscribed Circle



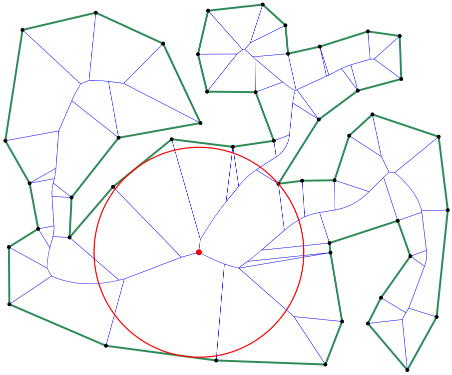
Maximum Inscribed Circle

- Similarly to the computation of a maximum empty circle, scanning the Voronoi nodes interior to a polygon yields a maximum inscribed circle in $O(n)$ time.

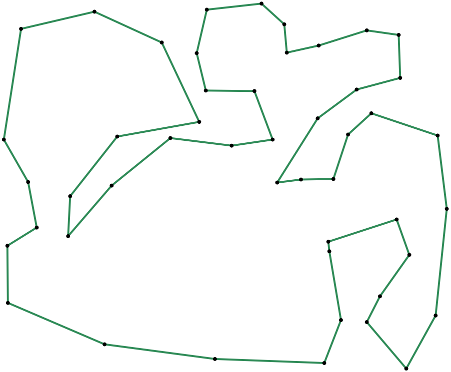


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Similarly to the computation of a maximum empty circle, scanning the Voronoi nodes interior to a polygon yields a maximum inscribed circle in $O(n)$ time.

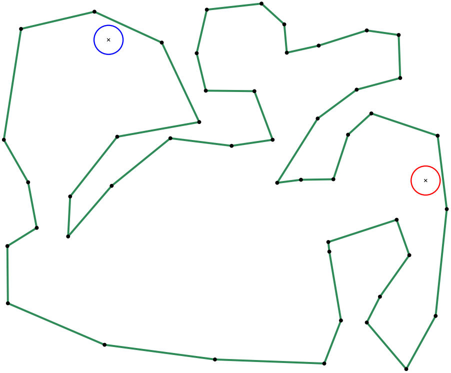


Finding a Gouge-Free Path



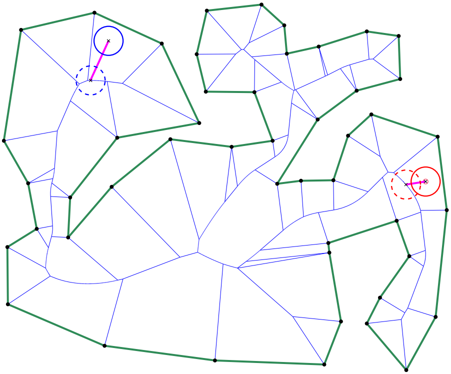
Finding a Gouge-Free Path

- Can we move the disk within the polygon from the blue to the red position?



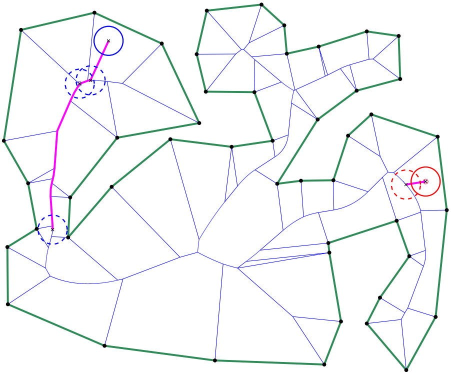
Finding a Gouge-Free Path

- Can we move the disk within the polygon from the blue to the red position?
- [Ó'Dúnlaing&Yap (1985)]: Retraction method:
  - Project red and blue centers onto the Voronoi diagram.



Finding a Gouge-Free Path

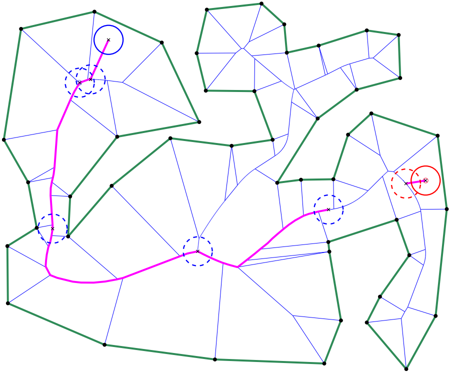
- Can we move the disk within the polygon from the blue to the red position?
- [Ó'Dúnlaing&Yap (1985)]: Retraction method:
  - Project red and blue centers onto the Voronoi diagram.
  - Scan the Voronoi diagram to find a way from blue to red.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Finding a Gouge-Free Path

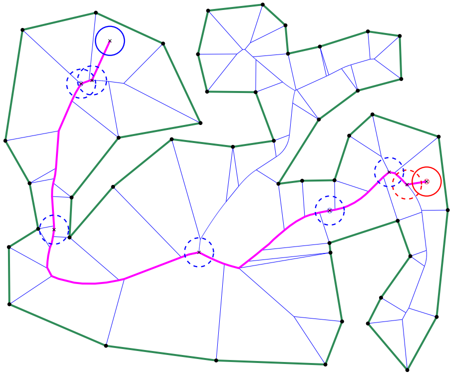
- Can we move the disk within the polygon from the blue to the red position?
- [Ó'Dúnlaing&Yap (1985)]: Retraction method:
  - Project red and blue centers onto the Voronoi diagram.
  - Scan the Voronoi diagram to find a way from blue to red.
  - Make sure to check the clearance while moving through a bottleneck.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Finding a Gouge-Free Path

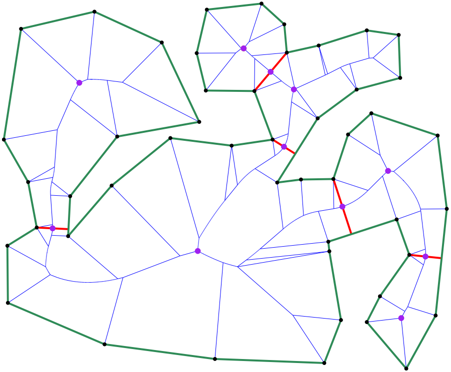
- Can we move the disk within the polygon from the blue to the red position?
- [Ó'Dúnlaing&Yap (1985)]: Retraction method:
  - Project red and blue centers onto the Voronoi diagram.
  - Scan the Voronoi diagram to find a way from blue to red.
  - Make sure to check the clearance while moving through a bottleneck.
- Indeed, this disk can be moved from blue to red!



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

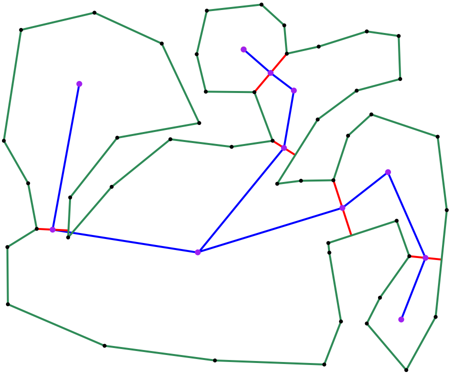
Bottlenecks and Locally Inner-Most Points

- A linear-time scan of the VD reveals all bottlenecks and locally inner-most points.



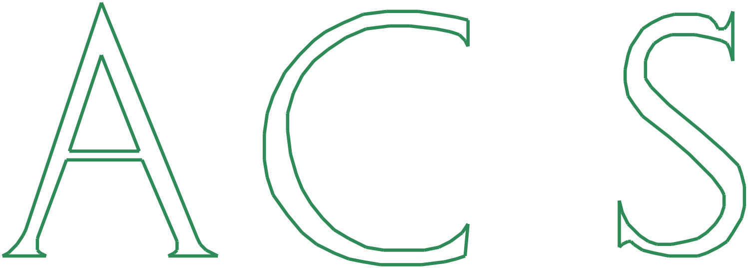
Bottlenecks and Locally Inner-Most Points

- To save time, a graph search is performed on the graph of offset-connected areas.



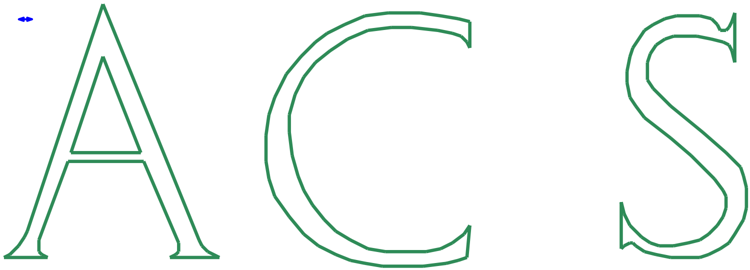
Informal problem statement

- For a set $\mathcal{P}$ of planar (polygonal or curvilinear) profiles



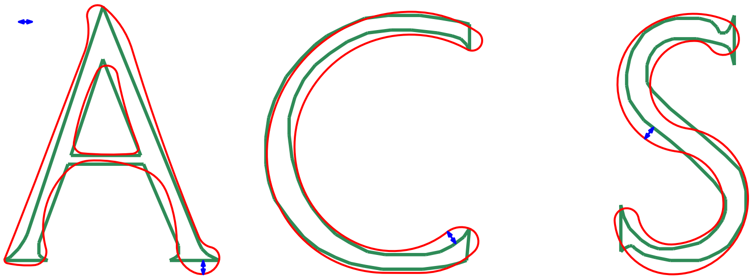
Informal problem statement

- For a set $\mathcal{P}$ of planar (polygonal or curvilinear) profiles
- and an approximation threshold given,



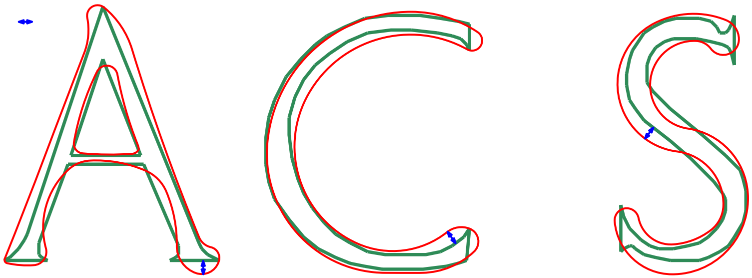
Informal problem statement

- For a set $\mathcal{P}$ of planar (polygonal or curvilinear) profiles
- and an approximation threshold given,
- compute an approximation such that the approximation threshold is not exceeded.



Informal problem statement

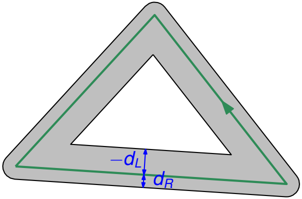
- For a set $\mathcal{P}$ of planar (polygonal or curvilinear) profiles
- and an approximation threshold given,
- compute an approximation such that the approximation threshold is not exceeded.

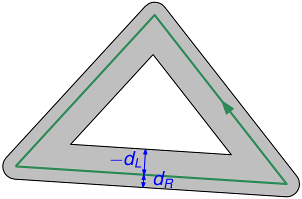


- Real-world applications: smoothing of tool paths, simplification of contours derived from scanning, recovery of “linearized” PCB data.

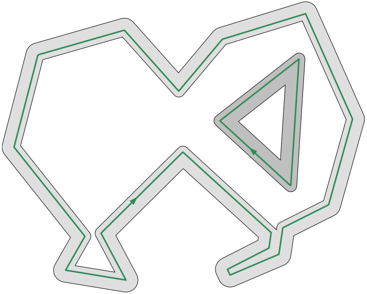
Approximation: Specifying a Tolerance

- Intuitively, for an input profile $P \in \mathcal{P}$ we seek a tolerance zone, $\mathcal{TZ}(P, d_L, d_R)$, of P with left tolerance d_L and right tolerance d_R.

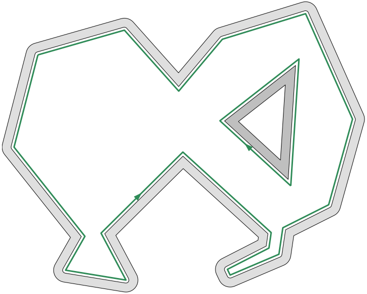




- Intuitively, for an input profile $P \in \mathcal{P}$ we seek a tolerance zone, $\mathcal{TZ}(P, d_L, d_R)$, of P with left tolerance d_L and right tolerance d_R.
- Non-trivial tolerances classified as
  - *symmetric* if $-d_L = d_R > 0$,
  - *asymmetric* if $d_L < 0 \leq d_R$ or $d_L \leq 0 < d_R$, and
  - *one-sided* if $d_L < d_R < 0$ or $0 < d_L <_r$.



asymmetric with $d_L < 0 < d_R$

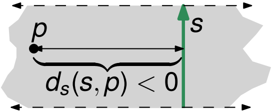


one-sided with $0 < d_L < d_R$

Definition 154 (Signed distance)

The *signed distance*, $d_s(s,p)$, of a point $p \in \mathcal{CI}(s)$

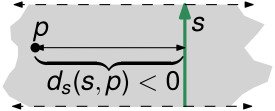
- to an oriented straight-line segment or circular arc s of $\mathcal{P}$ is given by the standard (Euclidean) distance of p to s, multiplied by -1 if p is on the left side of the supporting line or circle of s,

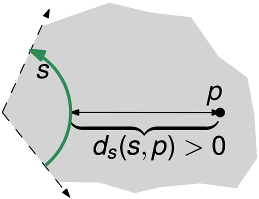


Definition 154 (Signed distance)

The *signed distance*, $d_s(s,p)$, of a point $p \in \mathcal{CI}(s)$

- to an oriented straight-line segment or circular arc s of $\mathcal{P}$ is given by the standard (Euclidean) distance of p to s, multiplied by -1 if p is on the left side of the supporting line or circle of s,

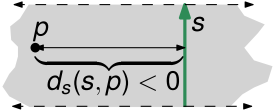


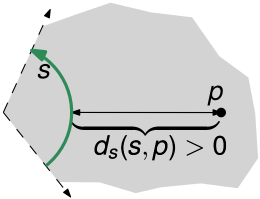


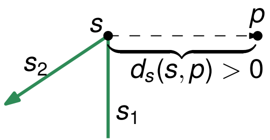
Definition 154 (Signed distance)

The *signed distance*, $d_s(s, p)$, of a point $p \in \mathcal{CI}(s)$

- to an oriented straight-line segment or circular arc s of $\mathcal{P}$ is given by the standard (Euclidean) distance of p to s, multiplied by -1 if p is on the left side of the supporting line or circle of s,
- to a vertex s of $\mathcal{P}$ we take the standard distance between p and s, and multiply it by -1 if the ray from s to p is locally on the left side of s_1 and s_2, where s_1 and s_2 are the sites of $\mathcal{P}$ that share s as a common vertex.





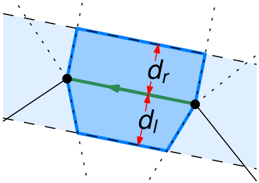


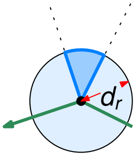
Tolerance Zone

Definition 155 (Tolerance zone)

The tolerance zone of a site s of $\mathcal{P}$ is defined as

$$\mathcal{TZ}_{site}(s, \mathcal{P}, d_L, d_R) := \{p \in \mathcal{VR}(\mathcal{P}, s) : d_L < d_s(s, p) < d_R\}.$$





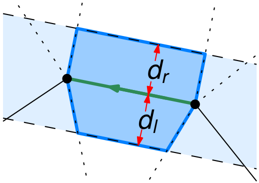
Definition 155 (Tolerance zone)

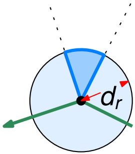
The tolerance zone of a site s of $\mathcal{P}$ is defined as

$$\mathcal{TZ}_{site}(s, \mathcal{P}, d_L, d_R) := \{p \in \mathcal{VR}(\mathcal{P}, s) : d_L < d_s(s, p) < d_R\}.$$

The tolerance zone of $\mathcal{P}$ is defined as the union of all tolerance zones of all sites:

$$\mathcal{TZ}(\mathcal{P}, d_L, d_R) := \bigcup_{s \in \mathcal{P}} \mathcal{TZ}_{site}(s, \mathcal{P}, d_L, d_R)\,.$$

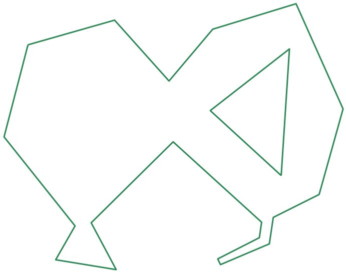




Problem Statement

Given: Input

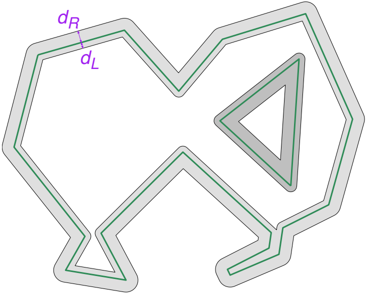
- Set $\mathcal{P}$ of (open or closed) polygonal profiles that do not intersect pairwise;



Problem Statement

Given: Input

- Set $\mathcal{P}$ of (open or closed) polygonal profiles that do not intersect pairwise;
- Left approximation tolerance d_L and right approximation tolerance d_R, with $d_L < d_R$.



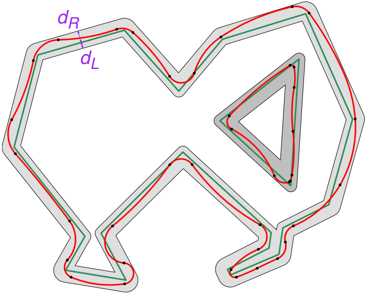
Problem Statement

Given: Input

- Set $\mathcal{P}$ of (open or closed) polygonal profiles that do not intersect pairwise;
- Left approximation tolerance d_L and right approximation tolerance d_R, with $d_L < d_R$.

Compute: Approximation $\mathcal{A}$ of $\mathcal{P}$ such that

- $\mathcal{A}$ consists of G^k curves, for some $k \in \mathbb{N}$,
- all curves of $\mathcal{A}$ are simple and pairwise disjoint,



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

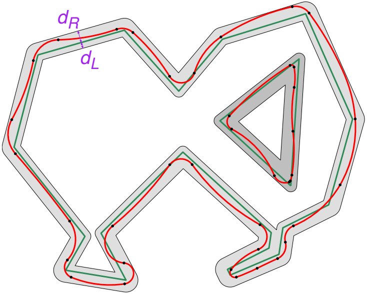
Problem Statement

Given: Input

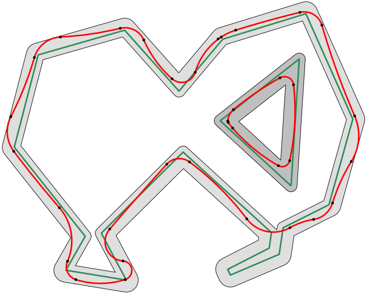
- Set $\mathcal{P}$ of (open or closed) polygonal profiles that do not intersect pairwise;
- Left approximation tolerance d_L and right approximation tolerance d_R, with $d_L < d_R$.

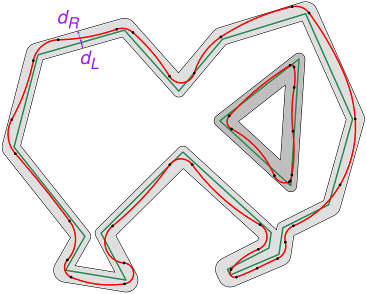
Compute: Approximation $\mathcal{A}$ of $\mathcal{P}$ such that

- $\mathcal{A}$ consists of G^k curves, for some $k \in \mathbb{N}$,
- all curves of $\mathcal{A}$ are simple and pairwise disjoint,
- $\mathcal{A} \subset \mathcal{TZ}(\mathcal{P}, d_L, d_R)$,
- $\mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, -d_L)$ if requested by user,
- topology of $\mathcal{A}$ matches topology of $\mathcal{P}$.



- Omitting the second condition $\mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, -d_L)$ makes a difference!





Tolerance Zone and Distance Measures

- Assume $-d_L = d_R > 0$. We have

$$\mathcal{A} \subset \mathcal{TZ}(\mathcal{P}, -d_R, d_R) \wedge \mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, d_R) \implies \mathrm{H}(\mathcal{A}, \mathcal{P}) \le d_R,$$

where $\mathrm{H}(\mathcal{A}, \mathcal{P})$ denotes the Hausdorff distance between $\mathcal{A}$ and $\mathcal{P}$.

Tolerance Zone and Distance Measures

- Assume $-d_L = d_R > 0$. We have

$$\mathcal{A} \subset \mathcal{TZ}(\mathcal{P}, -d_R, d_R) \wedge \mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, d_R) \implies \mathrm{H}(\mathcal{A}, \mathcal{P}) \leq d_R,$$

  where $\mathrm{H}(\mathcal{A}, \mathcal{P})$ denotes the Hausdorff distance between $\mathcal{A}$ and $\mathcal{P}$.

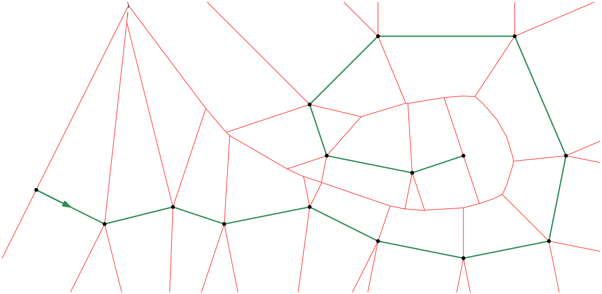
- Assume $-d_L = d_R > 0$. If each approximation curve $A \in \mathcal{A}$ is “monotone” relative to its corresponding input curve $P \in \mathcal{P}$, then

$$\mathcal{A} \subset \mathcal{TZ}(\mathcal{P}, -d_R, d_R) \wedge \mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, d_R) \implies \mathrm{Fr}(A, P) \leq d_R,$$

  where $\mathrm{Fr}(A, P)$ denotes the Fréchet distance between A and P, for each $A \in \mathcal{A}$ and corresponding $P \in \mathcal{P}$.

VD-Based Computation of the Tolerance Zone

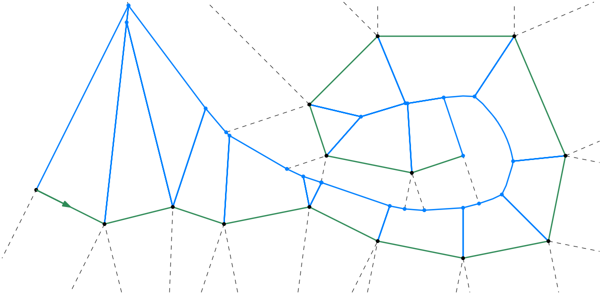
- [Held&Heimlich (2008), Held&Kaaser (2014)] use the Voronoi diagram of the input to compute the boundary of the tolerance zone.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

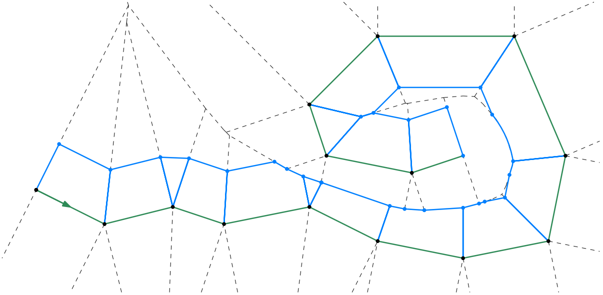
VD-Based Computation of the Tolerance Zone

- [Held&Heimlich (2008), Held&Kaaser (2014)] use the Voronoi diagram of the input to compute the boundary of the tolerance zone.
- Tolerance zone computation for an input profile:
  1. Collect all nodes of Voronoi cells left of the profile.

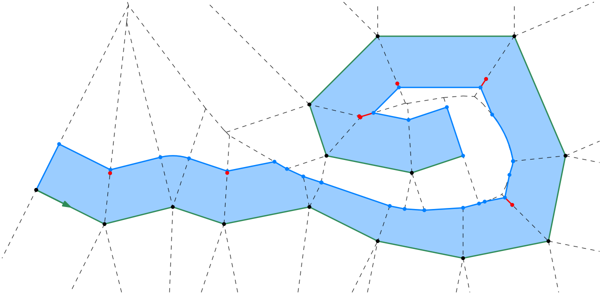


VD-Based Computation of the Tolerance Zone

- [Held&Heimlich (2008), Held&Kaaser (2014)] use the Voronoi diagram of the input to compute the boundary of the tolerance zone.
- Tolerance zone computation for an input profile:
  1. Collect all nodes of Voronoi cells left of the profile.
  2. Skip nodes that are further away than d_L from the profile.

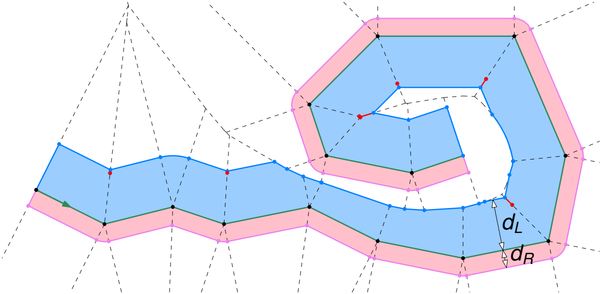


- [Held&Heimlich (2008), Held&Kaaser (2014)] use the Voronoi diagram of the input to compute the boundary of the tolerance zone.
- Tolerance zone computation for an input profile:
  1. Collect all nodes of Voronoi cells left of the profile.
  2. Skip nodes that are further away than d_L from the profile.
  3. Remove trees within the tolerance zone and add spikes.



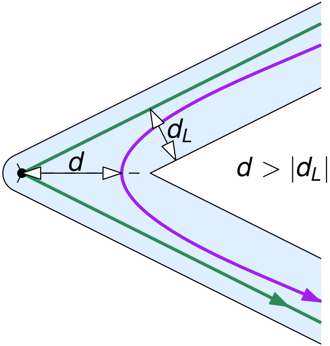
VD-Based Computation of the Tolerance Zone

- [Held&Heimlich (2008), Held&Kaaser (2014)] use the Voronoi diagram of the input to compute the boundary of the tolerance zone.
- Tolerance zone computation for an input profile:
  1. Collect all nodes of Voronoi cells left of the profile.
  2. Skip nodes that are further away than d_L from the profile.
  3. Remove trees within the tolerance zone and add spikes.
  4. Repeat this procedure for the right side of the profile w.r.t. d_R.



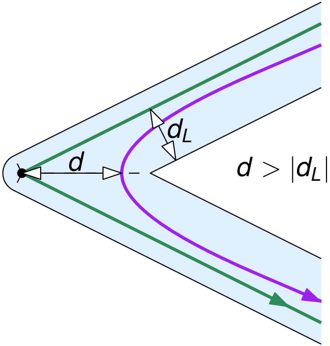
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

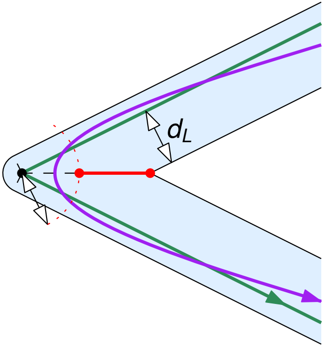
- How can we guarantee $\mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, -d_L)$?



Offset Spikes

- How can we guarantee $\mathcal{P} \subset \mathcal{TZ}(\mathcal{A}, -d_R, -d_L)$?
- *Offset spikes* ensure that the directed Hausdorff distance from the input to the approximation curve does not exceed the user-specified maximum tolerance.
- Spikes are formed by portions of the Voronoi diagram; they can be computed in linear time.





Theorem 156 (Held&Heimlich (2008))

Let n denote the number of vertices of a set $\mathcal{P}$ of polygonal profiles. Then a G^1 biarc approximation or a polygonal approximation, within an (asymmetric) user-specified tolerance that preserves the topology of $\mathcal{P}$, can be computed in $O(n \log n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 156 (Held&Heimlich (2008))

Let n denote the number of vertices of a set $\mathcal{P}$ of polygonal profiles. Then a G^1 biarc approximation or a polygonal approximation, within an (asymmetric) user-specified tolerance that preserves the topology of $\mathcal{P}$, can be computed in $O(n \log n)$ time.

Theorem 157 (Maier&Pisinger (2013))

Let n denote the number of vertices of one closed polygon P, and assume that a tolerance zone is given. Then a G^1 biarc approximation of P that uses the minimum number of biarcs (relative to the tolerance zone) can be computed in $O(n^3)$ time.

Results on Voronoi-Based Approximation

Theorem 156 (Held&Heimlich (2008))

Let n denote the number of vertices of a set $\mathcal{P}$ of polygonal profiles. Then a G^1 biarc approximation or a polygonal approximation, within an (asymmetric) user-specified tolerance that preserves the topology of $\mathcal{P}$, can be computed in $O(n \log n)$ time.

Theorem 157 (Maier&Pisinger (2013))

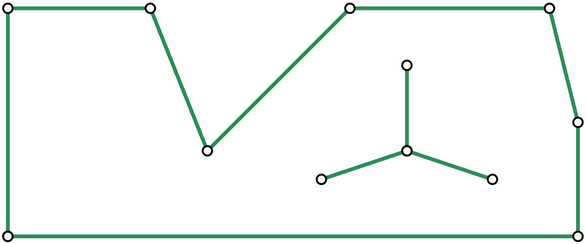
Let n denote the number of vertices of one closed polygon P, and assume that a tolerance zone is given. Then a G^1 biarc approximation of P that uses the minimum number of biarcs (relative to the tolerance zone) can be computed in $O(n^3)$ time.

Theorem 158 (Held&Kaaser (2014))

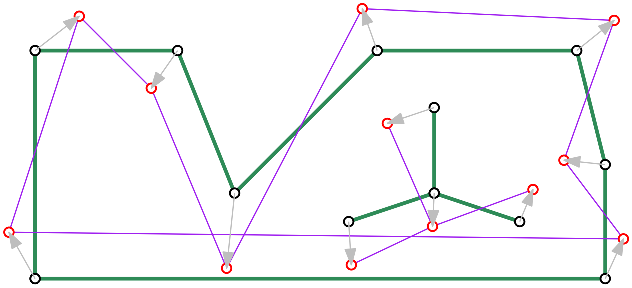
Let n denote the number of vertices of a set $\mathcal{P}$ of polygonal profiles. Then a C^2 approximation by uniform cubic B-splines within an (asymmetric) user-specified tolerance that preserves the topology of $\mathcal{P}$ can be computed in $O(n \log n)$ time.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

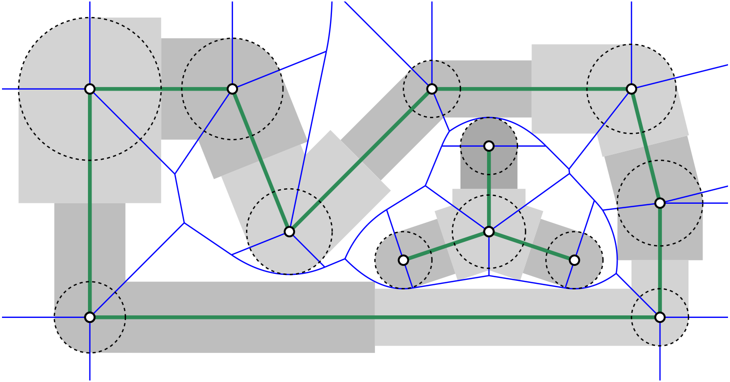
- Watermarking techniques for vector graphics dislocate vertices in order to embed imperceptible, yet detectable, statistical features into the input data.



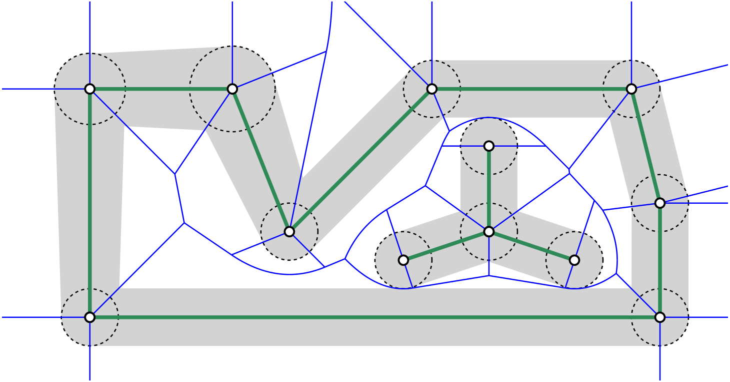
- Watermarking techniques for vector graphics dislocate vertices in order to embed imperceptible, yet detectable, statistical features into the input data.
- Obvious problem: One needs to guarantee that the introduction of a watermark preserves the input topology.



- [Huber et al. (2014)] compute for each vertex a disk-shaped maximum perturbation region (MPR), based on the Voronoi diagram of the input.

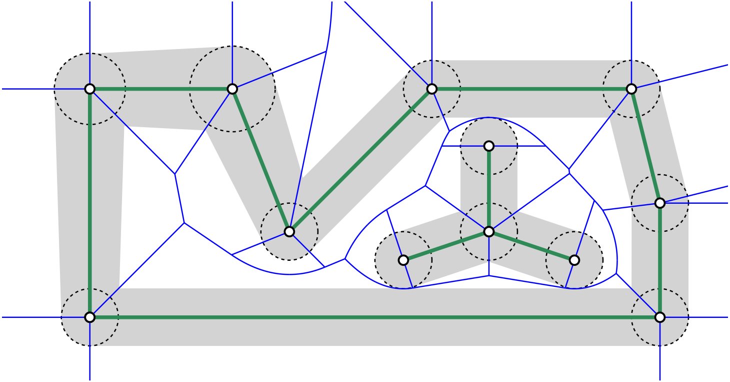


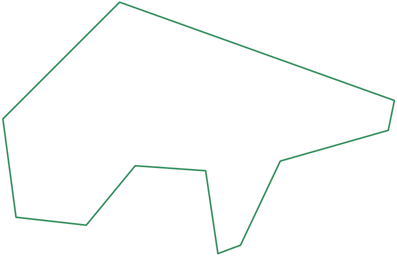
- [Huber et al. (2014)] compute for each vertex a disk-shaped maximum perturbation region (MPR), based on the Voronoi diagram of the input.
- Perturbing the vertices within their MPRs causes the edges to stay within their hoses and allows to preserve the input topology.

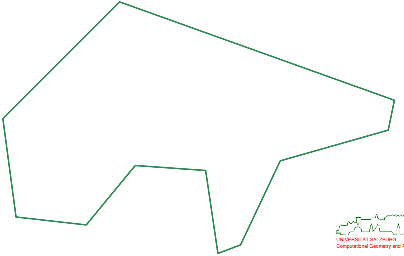


Topology-Preserving Watermarking of Vector Graphics

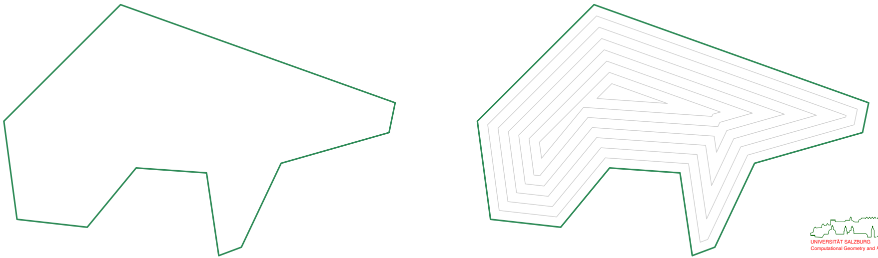
- [Huber et al. (2014)] compute for each vertex a disk-shaped maximum perturbation region (MPR), based on the Voronoi diagram of the input.
- Perturbing the vertices within their MPRs causes the edges to stay within their hoses and allows to preserve the input topology.
- This scheme can be extended to cover straight-line segments and circular arcs.





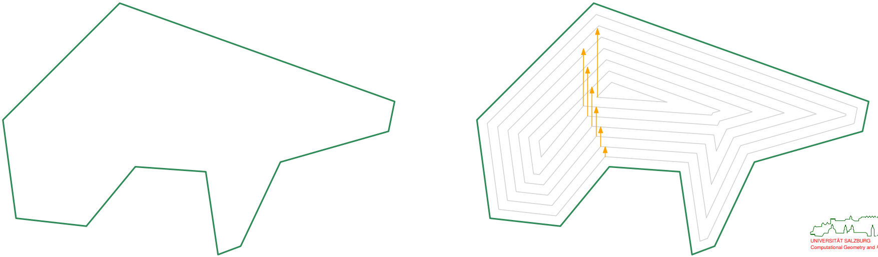


Straight Skeleton: Roof Model



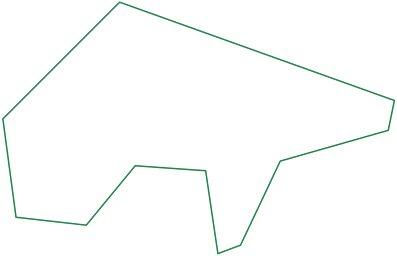
Roof model via lifted wavefronts

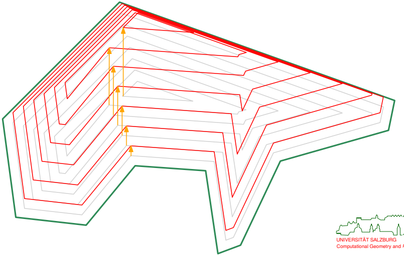
We lift a wavefront $\mathcal{WF}(P, t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P, t) \times \{t\}$.



Roof model via lifted wavefronts

We lift a wavefront $\mathcal{WF}(P, t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P, t) \times \{t\}$.





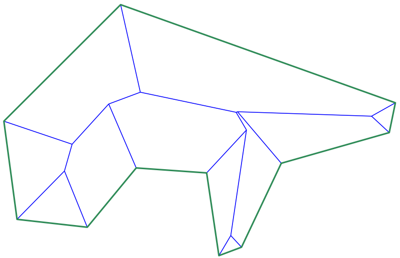
Straight Skeleton: Roof Model

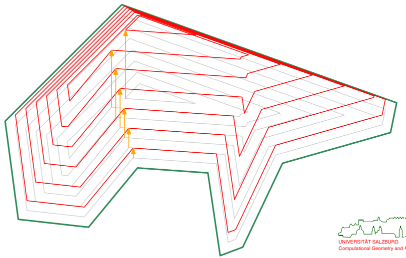
Roof model via lifted wavefronts

We lift a wavefront $\mathcal{WF}(P,t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P,t) \times \{t\}$.

Roof model

Alternatively, a point p of the straight skeleton of polygon P, with coordinates (p_x, p_y), is lifted to a point in 3D with coordinates (p_x, p_y, t) if the orthogonal boundary clearance of p is t.



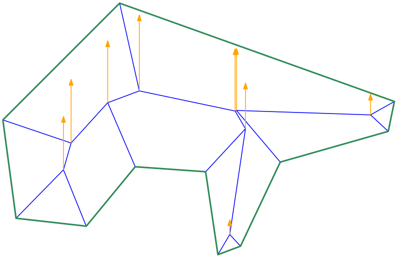


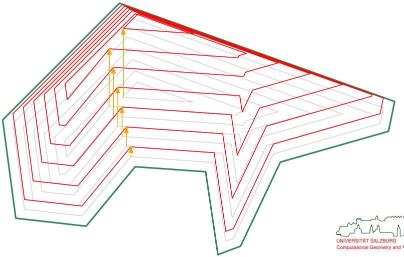
Roof model via lifted wavefronts

We lift a wavefront $\mathcal{WF}(P, t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P, t) \times \{t\}$.

Roof model

Alternatively, a point p of the straight skeleton of polygon P, with coordinates (p_x, p_y), is lifted to a point in 3D with coordinates (p_x, p_y, t) if the orthogonal boundary clearance of p is t.





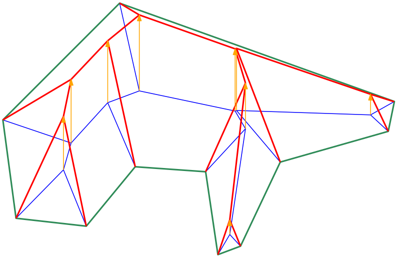
Straight Skeleton: Roof Model

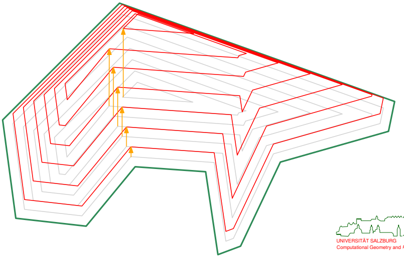
Roof model via lifted wavefronts

We lift a wavefront $\mathcal{WF}(P, t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P, t) \times \{t\}$.

Roof model

Alternatively, a point p of the straight skeleton of polygon P, with coordinates (p_x, p_y), is lifted to a point in 3D with coordinates (p_x, p_y, t) if the orthogonal boundary clearance of p is t.



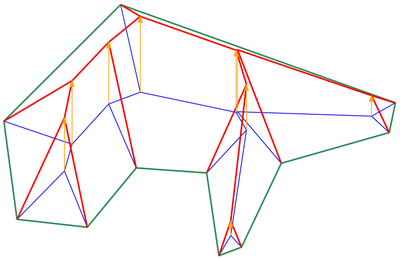


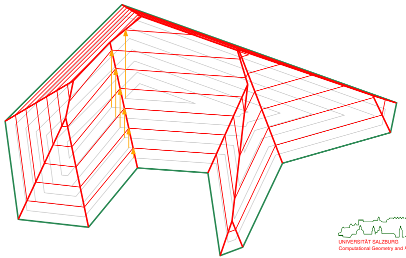
Roof model via lifted wavefronts

We lift a wavefront $\mathcal{WF}(P, t)$ of P for the orthogonal boundary clearance t to z-coordinate t: We get $\mathcal{WF}(P, t) \times \{t\}$.

Roof model

Alternatively, a point p of the straight skeleton of polygon P, with coordinates (p_x, p_y), is lifted to a point in 3D with coordinates (p_x, p_y, t) if the orthogonal boundary clearance of p is t.





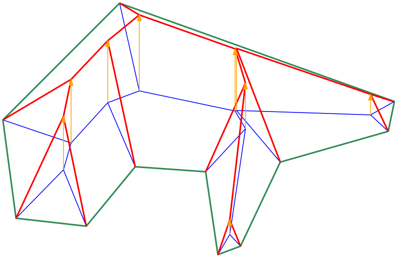
Straight Skeleton: Properties of Roof Model

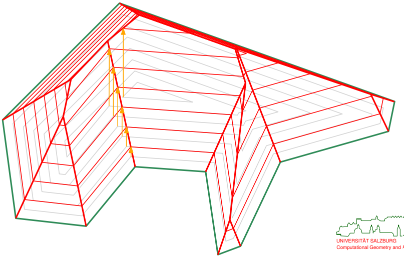
Properties

- The *roof*

$$\mathcal{R}(P) := \bigcup_{t \geq 0} (\mathcal{WF}(P,t) \times \{t\})$$

is a piecewise-linear and continuous surface.





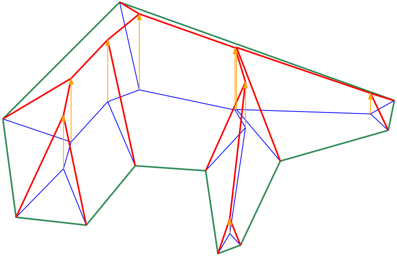
Properties

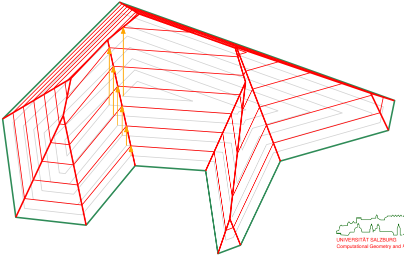
- The *roof*

$$\mathcal{R}(P) := \bigcup_{t \geq 0} (\mathcal{WF}(P, t) \times \{t\})$$

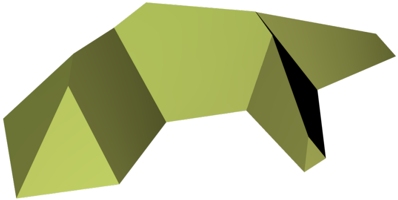
is a piecewise-linear and continuous surface.

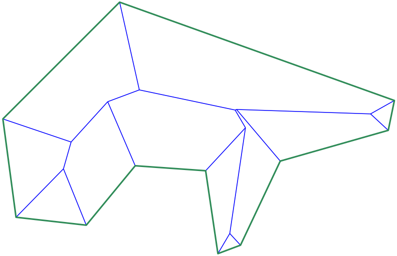
- It is monotone relative to the *xy*-plane.

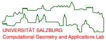




Straight Skeleton and Voronoi Diagram: Roof Model

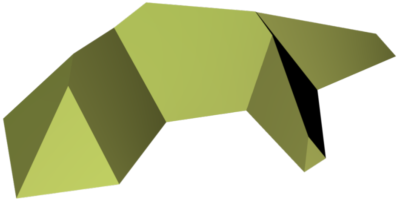


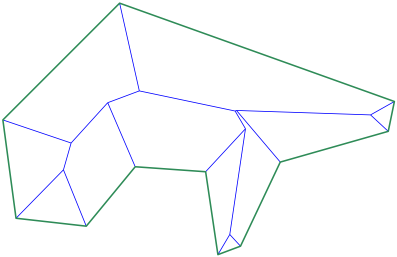


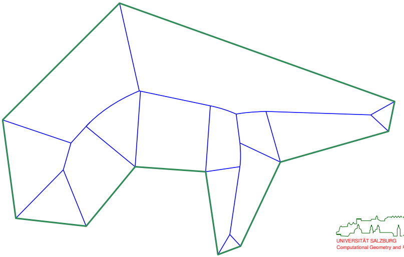


Straight Skeleton and Voronoi Diagram: Roof Model

- The same lifting approach can also be applied to Voronoi diagrams, thereby generating a roof for a Voronoi diagram.

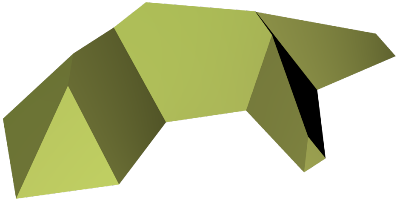


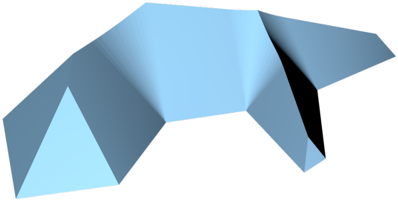


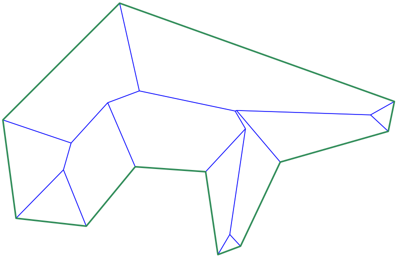


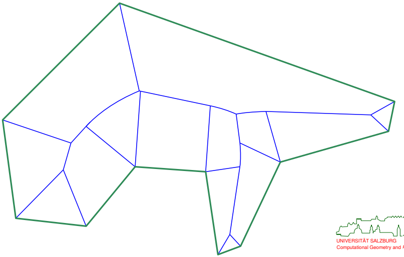
Straight Skeleton and Voronoi Diagram: Roof Model

- The same lifting approach can also be applied to Voronoi diagrams, thereby generating a roof for a Voronoi diagram.

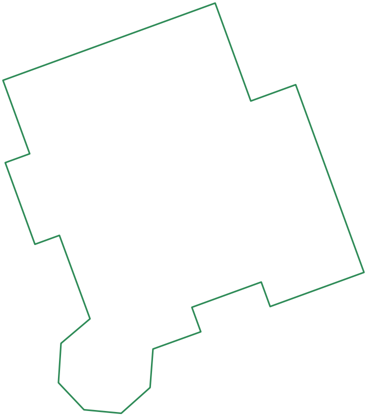




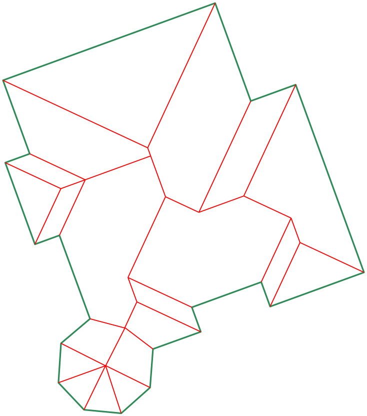




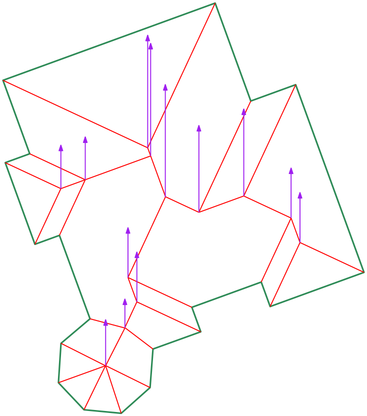
- Footprint.



- Footprint. Straight skeleton.

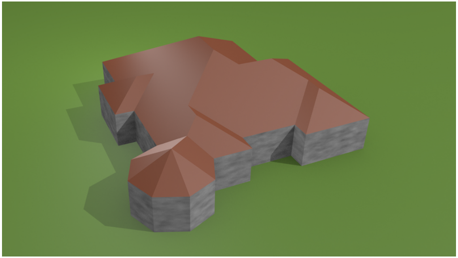


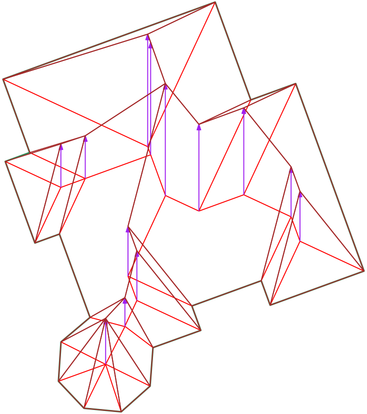
- Footprint. Straight skeleton. Lift to 3D.



Roofs as Skeletal Structures Lifted to 3D

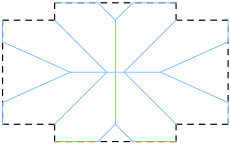
- Footprint. Straight skeleton. Lift to 3D. Roof.

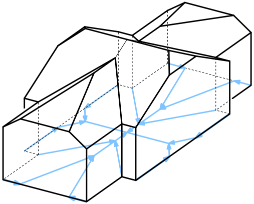


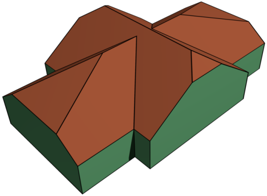


Held&Palfrader (2016)

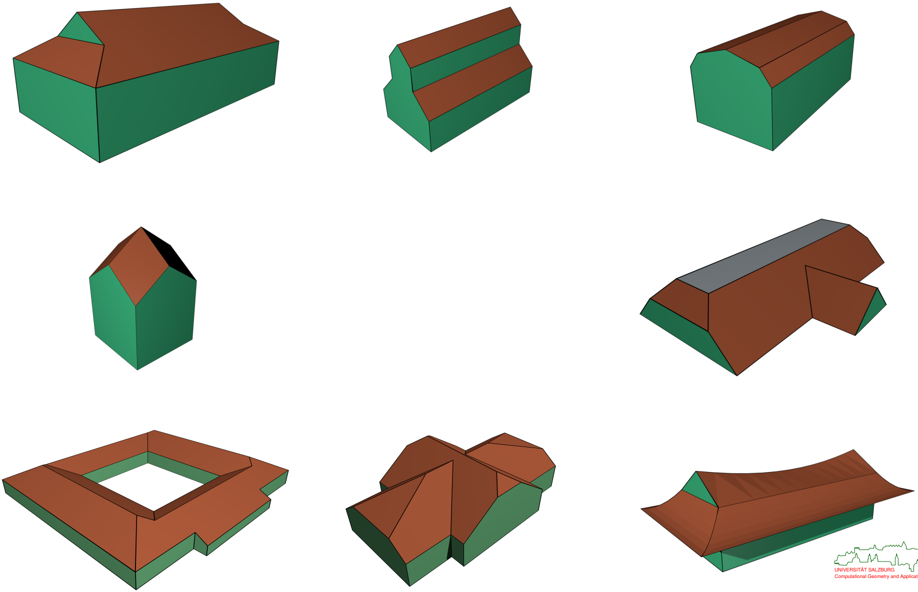
Additive and multiplicative weights support the automatic generation of realistic complex roofs based on the footprints of buildings.



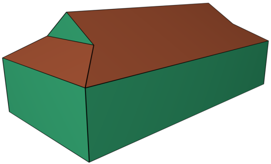


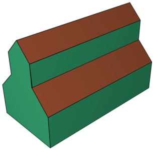


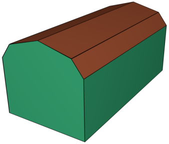
Complex Roofs for Urban Modeling and Reconstruction

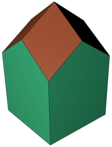


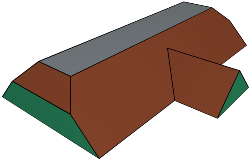
Complex Roofs for Urban Modeling and Reconstruction

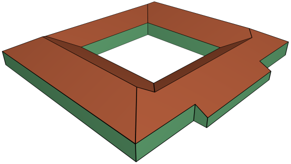


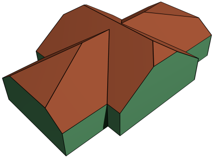


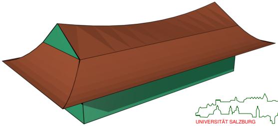




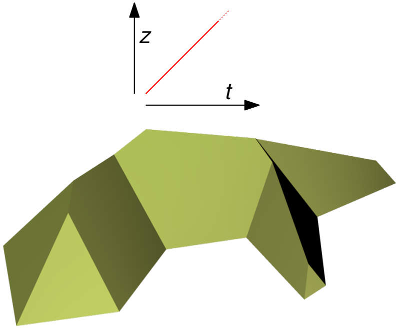








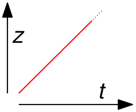
Generalizing the Roof Based on Straight Skeleton

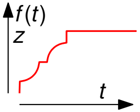


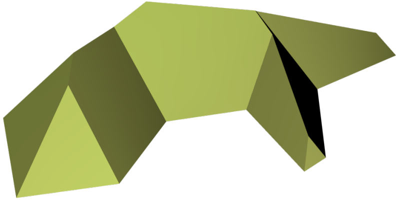
Generalizing the Roof Based on Straight Skeleton

Generalized Roof

We use a (continuous) height "function" f to obtain a scalar field on P,





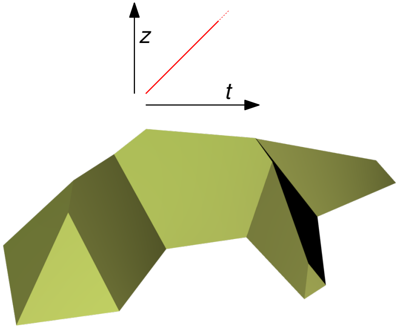


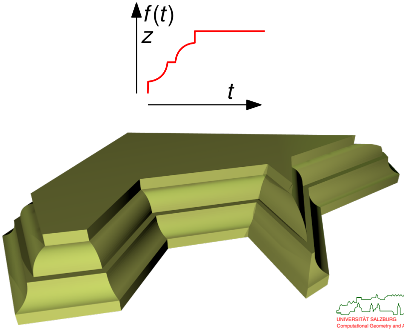
Generalizing the Roof Based on Straight Skeleton

Generalized Roof

We use a (continuous) height "function" f to obtain a scalar field on P, thereby generalizing the roof $\mathcal{R}(P)$ to a surface $\mathcal{T}_f(P)$:

$$\mathcal{T}_f(P) := \bigcup_{t \geq 0} (\mathcal{WF}(P,t) \times \{f(t)\}).$$



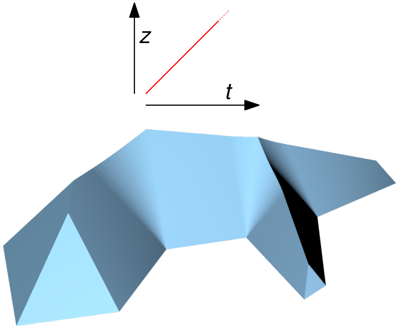


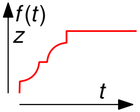
Generalizing the Roof Based on Voronoi Diagram

Generalized Roof

We use a (continuous) height "function" f to obtain a scalar field on P, thereby generalizing the roof $\mathcal{R}(P)$ to a surface $\mathcal{T}_f(P)$:

$$\mathcal{T}_f(P) := \bigcup_{t \geq 0} (\mathcal{WF}(P, t) \times \{f(t)\}).$$



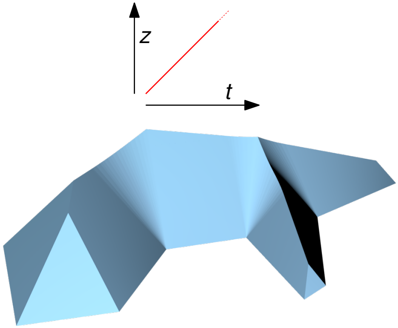


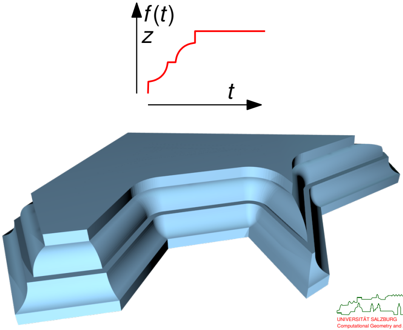
Generalizing the Roof Based on Voronoi Diagram

Generalized Roof

We use a (continuous) height "function" f to obtain a scalar field on P, thereby generalizing the roof $\mathcal{R}(P)$ to a surface $\mathcal{T}_f(P)$:

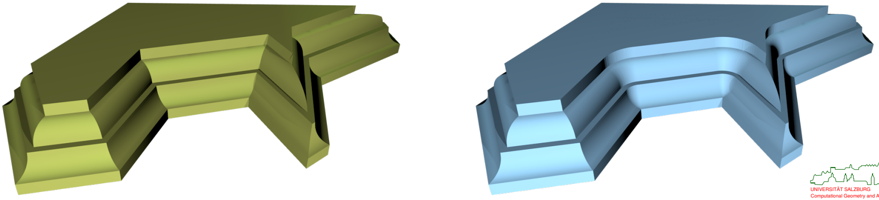
$$\mathcal{T}_f(P) := \bigcup_{t \geq 0} (\mathcal{WF}(P, t) \times \{f(t)\}).$$





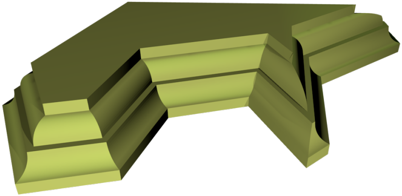
Properties

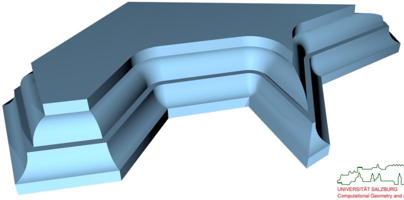
- The generalized roof $\mathcal{T}_f(P)$ is monotone relative to the *xy*-plane.



Properties

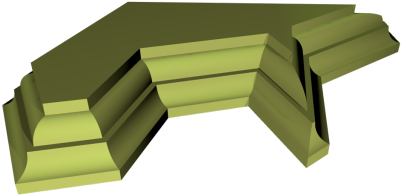
- The generalized roof $\mathcal{T}_f(P)$ is monotone relative to the *xy*-plane.
- If f is continuous then also $\mathcal{T}_f(P)$ is continuous.

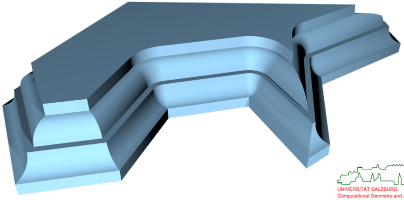




Properties

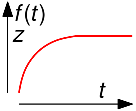
- The generalized roof $\mathcal{T}_f(P)$ is monotone relative to the *xy*-plane.
- If f is continuous then also $\mathcal{T}_f(P)$ is continuous.
- A face of $\mathcal{T}_f(P)$ is a ruled surface if it is incident to an edge of P, and a surface of revolution if it is incident to a reflex vertex.

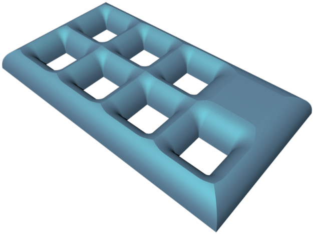




Held&Palfrader (2018)

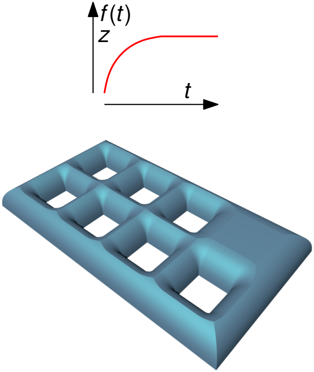
Such a generalization of the function that “lifts” a Voronoi diagram or (weighted) straight skeleton to 3D supports the generation of complex chamfers and fillets.

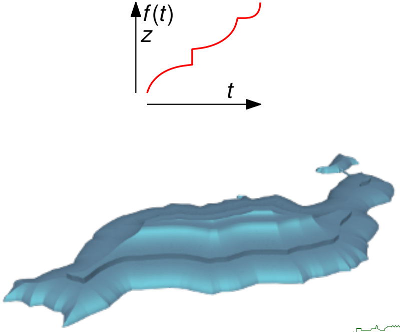




Held&Palfrader (2018)

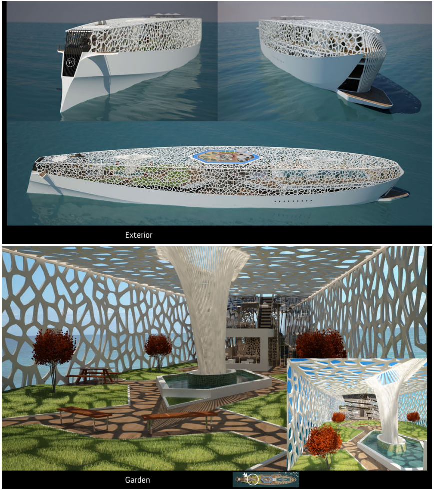
Such a generalization of the function that "lifts" a Voronoi diagram or (weighted) straight skeleton to 3D supports the generation of complex chamfers and fillets.





Voronoi Diagrams in Structural Design

- CNN (16-Aug-2011): "Stunning superyacht design inspired by nature's hidden patterns".



[Images courtesy of Hyun-Seok Kim]

7 Triangulations

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

7 Triangulations

- Basics
  - Definitions
  - Types of Triangulations
  - Constrained Triangulations
- Computing Constrained Triangulations
- Triangulations in 3D
- Applications of Triangulations

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 159 (Triangulation of a point set)

A *triangulation* of a set S of n points of $\mathbb{R}^2$ is a subdivision of the convex hull $CH(S)$ into triangles such that

1. the set of vertices of the triangles matches S,
2. no pair of triangles intersects except in a common vertex or edge.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 159 (Triangulation of a point set)

A *triangulation* of a set S of n points of $\mathbb{R}^2$ is a subdivision of the convex hull $CH(S)$ into triangles such that

1. the set of vertices of the triangles matches S,
2. no pair of triangles intersects except in a common vertex or edge.

Definition 160 (Triangulation of a polygon)

A *polygon triangulation* is a subdivision of a (simple plane) polygon P into triangles such that

1. the set of vertices of the triangles matches the vertices of P,
2. no pair of triangles intersects except in a common vertex or edge.

Definition 159 (Triangulation of a point set)

A *triangulation* of a set S of n points of $\mathbb{R}^2$ is a subdivision of the convex hull $CH(S)$ into triangles such that

1. the set of vertices of the triangles matches S,
2. no pair of triangles intersects except in a common vertex or edge.

Definition 160 (Triangulation of a polygon)

A *polygon triangulation* is a subdivision of a (simple plane) polygon P into triangles such that

1. the set of vertices of the triangles matches the vertices of P,
2. no pair of triangles intersects except in a common vertex or edge.

- Similarly for points/polyhedra in $\mathbb{R}^3$ and a subdivision into tetrahedra.

Definition 159 (Triangulation of a point set)

A *triangulation* of a set S of n points of $\mathbb{R}^2$ is a subdivision of the convex hull $CH(S)$ into triangles such that

1. the set of vertices of the triangles matches S,
2. no pair of triangles intersects except in a common vertex or edge.

Definition 160 (Triangulation of a polygon)

A *polygon triangulation* is a subdivision of a (simple plane) polygon P into triangles such that

1. the set of vertices of the triangles matches the vertices of P,
2. no pair of triangles intersects except in a common vertex or edge.

- Similarly for points/polyhedra in $\mathbb{R}^3$ and a subdivision into tetrahedra.
- For $d > 3$ it is standard to resort to the terms "simplex" and "simplicial complex" to define triangulations in $\mathbb{R}^d$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

What is a Good Triangulation?

- Let S be a set of n points in $\mathbb{R}^2$. Several options to demand additional properties for a triangulation of S.

What is a Good Triangulation?

- Let S be a set of n points in $\mathbb{R}^2$. Several options to demand additional properties for a triangulation of S.

Definition 161 (Locally Delaunay)

A triangulation $\mathcal{T}(\mathcal{S})$ of S is *locally Delaunay* if for every pair of adjacent triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$ of $\mathcal{T}(\mathcal{S})$ the Delaunay triangulation of a, b, c, d includes these two triangles.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

What is a Good Triangulation?

- Let S be a set of n points in $\mathbb{R}^2$. Several options to demand additional properties for a triangulation of S.

Definition 161 (Locally Delaunay)

A triangulation $\mathcal{T}(\mathcal{S})$ of S is *locally Delaunay* if for every pair of adjacent triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$ of $\mathcal{T}(\mathcal{S})$ the Delaunay triangulation of a, b, c, d includes these two triangles.

Lemma 162 (Fortune (1992))

A triangulation of S is a Delaunay triangulation of S if and only if it is locally Delaunay.

What is a Good Triangulation?

- Let S be a set of n points in $\mathbb{R}^2$. Several options to demand additional properties for a triangulation of S.

Definition 161 (Locally Delaunay)

A triangulation $\mathcal{T}(\mathcal{S})$ of S is *locally Delaunay* if for every pair of adjacent triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$ of $\mathcal{T}(\mathcal{S})$ the Delaunay triangulation of a, b, c, d includes these two triangles.

Lemma 162 (Fortune (1992))

A triangulation of S is a Delaunay triangulation of S if and only if it is locally Delaunay.

Lemma 163 (Sibson (1978))

The minimum internal angle of the triangles of $\mathcal{DT}$(S) is maximum over all triangulations of S.

What is a Good Triangulation?

- Let S be a set of n points in $\mathbb{R}^2$. Several options to demand additional properties for a triangulation of S.

Definition 161 (Locally Delaunay)

A triangulation $\mathcal{T}(\mathcal{S})$ of S is *locally Delaunay* if for every pair of adjacent triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$ of $\mathcal{T}(\mathcal{S})$ the Delaunay triangulation of a, b, c, d includes these two triangles.

Lemma 162 (Fortune (1992))

A triangulation of S is a Delaunay triangulation of S if and only if it is locally Delaunay.

Lemma 163 (Sibson (1978))

The minimum internal angle of the triangles of $\mathcal{DT}$(S) is maximum over all triangulations of S.

Lemma 164 (Lambert (1994))

The (arithmetic) mean inradius of the triangles of $\mathcal{DT}$(S) is maximum over all triangulations of S.

What is a Good Triangulation?

Definition 165 (Hamiltonian triangulation)

A triangulation of S is a *Hamiltonian triangulation* if the dual graph of the triangulation admits a Hamiltonian path.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

What is a Good Triangulation?

Definition 165 (Hamiltonian triangulation)

A triangulation of S is a *Hamiltonian triangulation* if the dual graph of the triangulation admits a Hamiltonian path.

Lemma 166 (Arkin et alii (1996))

A Hamiltonian triangulation of S can be computed in $O(n \log n)$ time.

What is a Good Triangulation?

Definition 165 (Hamiltonian triangulation)

A triangulation of S is a *Hamiltonian triangulation* if the dual graph of the triangulation admits a Hamiltonian path.

Lemma 166 (Arkin et alii (1996))

A Hamiltonian triangulation of S can be computed in $O(n \log n)$ time.

Definition 167 (Minimum-weight triangulation)

A triangulation of S is a *minimum-weight triangulation* (MWT) if the sum of the lengths of the triangulation edges is minimum over all triangulations.

What is a Good Triangulation?

Definition 165 (Hamiltonian triangulation)

A triangulation of S is a *Hamiltonian triangulation* if the dual graph of the triangulation admits a Hamiltonian path.

Lemma 166 (Arkin et alii (1996))

A Hamiltonian triangulation of S can be computed in $O(n \log n)$ time.

Definition 167 (Minimum-weight triangulation)

A triangulation of S is a *minimum-weight triangulation* (MWT) if the sum of the lengths of the triangulation edges is minimum over all triangulations.

Theorem 168 (Mulzer&Rote (2006))

Computing a minimum-weight triangulation is $\mathcal{NP}$-hard.

Theorem 169 (Remy&Steger (2009))

For any $\varepsilon > 0$, a minimum-weight triangulation can be approximated with approximation factor $1 + \varepsilon$ in time $2^{O((\log n)^c)}$ for some fixed $c \in \mathbb{R}^+$.

Counting Triangulations

- No tight bounds are known for the minimum and the maximum number of (different straight-edge) triangulations that n points in 2D may admit.
- For 20 points, the best known minimum is 20 662 980, and the best known maximum is 918 462 742 512 [Aicholzer et alii (2001–2003)].

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Counting Triangulations

- No tight bounds are known for the minimum and the maximum number of (different straight-edge) triangulations that n points in 2D may admit.
- For 20 points, the best known minimum is 20 662 980, and the best known maximum is 918 462 742 512 [Aicholzer et alii (2001–2003)].

Lemma 170 (Goldbach&Euler (1751), Lamé (1838))

If $(n+2)$ points are in convex position then the number of different triangulations is given by the n-th Catalan number C_n.

Counting Triangulations

- No tight bounds are known for the minimum and the maximum number of (different straight-edge) triangulations that n points in 2D may admit.
- For 20 points, the best known minimum is 20 662 980, and the best known maximum is 918 462 742 512 [Aicholzer et alii (2001–2003)].

Lemma 170 (Goldbach&Euler (1751), Lamé (1838))

If $(n+2)$ points are in convex position then the number of different triangulations is given by the n-th Catalan number C_n.

Lemma 171 (Sharir&Sheffer (2009))

Every set of n points in the plane admits at most 30^n different triangulations.

Lemma 172 (Aichholzer et alii (2016))

Every set of n points — GPA! — in the plane admits at least $\Omega(2.631^n)$ triangulations.

Lemma 173 (Dumitrescu et alii (2010))

There exist sets of n points in the plane which admit at least $\Omega(8.65^n)$ triangulations.

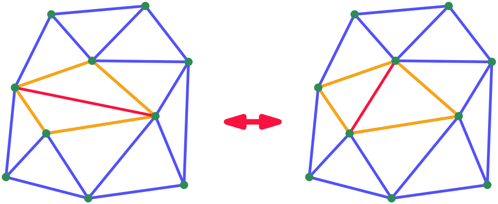
Triangulations are Related via Edge Flips

Definition 174 (Edge flip, Dt.: Kantenaustausch)

An *edge flip* is a local operation on a triangulation that replaces one diagonal of a convex quadrilateral (formed by two neighboring triangles) with the other diagonal.

Definition 174 (Edge flip, Dt.: Kantenaustausch)

An *edge flip* is a local operation on a triangulation that replaces one diagonal of a convex quadrilateral (formed by two neighboring triangles) with the other diagonal.



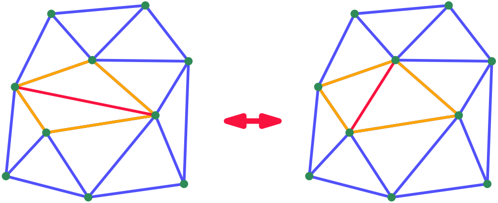
Triangulations are Related via Edge Flips

Definition 174 (Edge flip, Dt.: Kantenaustausch)

An *edge flip* is a local operation on a triangulation that replaces one diagonal of a convex quadrilateral (formed by two neighboring triangles) with the other diagonal.

Lemma 175 (Bern&Eppstein (1992))

$O(n^2)$ edge-flipping operations suffice to transform any triangulation of n points (in $\mathbb{R}^2$) into a Delaunay triangulation.



Triangulations are Related via Edge Flips

Lemma 176 (Lubiw&Pathak (2012))

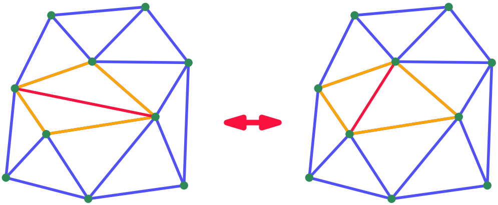
Minimizing the flip distance between triangulations of point sets is $\mathcal{NP}$-hard.

Lemma 177 (Aichholzer&Mulzer&Pilz (2013))

Minimizing the flip distance between triangulations of a polygon is $\mathcal{NP}$-hard.

Lemma 178 (Pilz (2014))

Minimizing the flip distance between triangulations of point sets is APX-hard; i.e., no polynomial-time constant factor approximation exists unless $\mathcal{P} = \mathcal{NP}$.

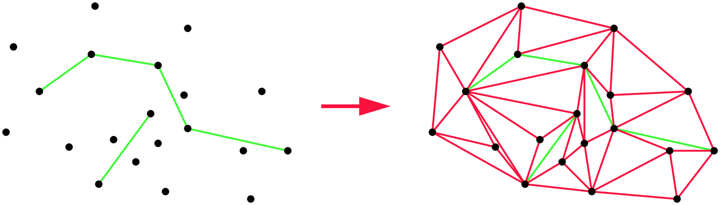


Constrained Triangulation

Definition 179 (Constrained triangulation)

A triangulation $\mathcal{T}$ forms a *constrained triangulation* of an admissible set S of vertices and line segments in $\mathbb{R}^2$ if

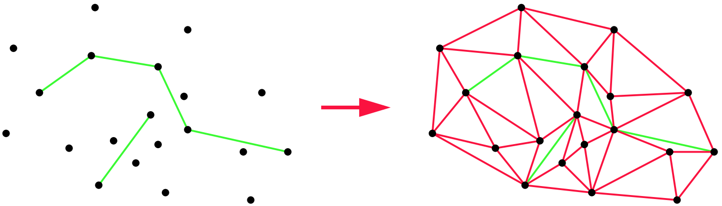
1. $\mathcal{T}$ is a triangulation of the convex hull of all vertices of S,
2. all line segments of S are edges of $\mathcal{T}$.



Definition 180 (Constrained Delaunay triangulation)

A triangulation $\mathcal{T}$ forms a *constrained Delaunay triangulation* (CDT) of an admissible set S of vertices and line segments if

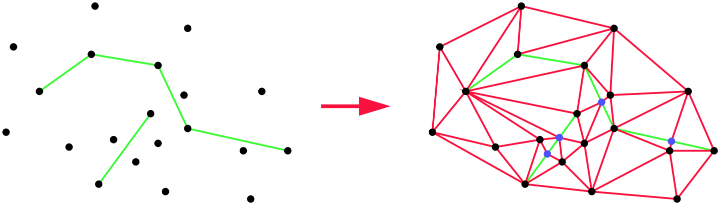
1. $\mathcal{T}$ is a constrained triangulation of S, and
2. no triangle Δ of $\mathcal{T}$ contains a vertex of S in its circumcircle that is visible from Δ.



Definition 181 (Conforming triangulation)

A triangulation $\mathcal{T}$ forms a *conforming triangulation* of an admissible set S of vertices and line segments if

1. $\mathcal{T}$ is a triangulation of the convex hull of all vertices of S and (possibly) of some additional Steiner points,
2. all line segments of S are represented by unions of edges of $\mathcal{T}$.



7 Triangulations

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 182 (Diagonal)

For vertices p and q of a simple polygon P, the line segment $\overline{pq}$ forms a *diagonal* of P if $\overline{pq}$ lies completely in the interior of P, except for the vertices p and q.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Definition 182 (Diagonal)

For vertices p and q of a simple polygon P, the line segment $\overline{pq}$ forms a *diagonal* of P if $\overline{pq}$ lies completely in the interior of P, except for the vertices p and q.

Lemma 183

Every simple polygon with $n \geq 4$ vertices contains a diagonal.

Definition 182 (Diagonal)

For vertices p and q of a simple polygon P, the line segment $\overline{pq}$ forms a *diagonal* of P if $\overline{pq}$ lies completely in the interior of P, except for the vertices p and q.

Lemma 183

Every simple polygon with $n \geq 4$ vertices contains a diagonal.

Corollary 184

Every simple polygon with n vertices can be partitioned into $n-2$ triangles by inserting $n-3$ appropriate diagonals.

Computing Constrained Triangulations

Definition 182 (Diagonal)

For vertices p and q of a simple polygon P, the line segment $\overline{pq}$ forms a *diagonal* of P if $\overline{pq}$ lies completely in the interior of P, except for the vertices p and q.

Lemma 183

Every simple polygon with $n \geq 4$ vertices contains a diagonal.

Corollary 184

Every simple polygon with n vertices can be partitioned into $n-2$ triangles by inserting $n-3$ appropriate diagonals.

Lemma 185

A regularization procedure can be used to partition a simple polygon with n vertices in $O(n \log n)$ time into a set of monotone polygons.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Computing Constrained Triangulations

Definition 182 (Diagonal)

For vertices p and q of a simple polygon P, the line segment $\overline{pq}$ forms a *diagonal* of P if $\overline{pq}$ lies completely in the interior of P, except for the vertices p and q.

Lemma 183

Every simple polygon with $n \geq 4$ vertices contains a diagonal.

Corollary 184

Every simple polygon with n vertices can be partitioned into $n - 2$ triangles by inserting $n - 3$ appropriate diagonals.

Lemma 185

A regularization procedure can be used to partition a simple polygon with n vertices in $O(n \log n)$ time into a set of monotone polygons.

Corollary 186

A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Corollary 188 (Chazelle (1991))

In $O(n)$ time we can check whether a polygon is simple.

Computing Constrained Triangulations

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Corollary 188 (Chazelle (1991))

In $O(n)$ time we can check whether a polygon is simple.

Theorem 189 (Clarkson&Tarjan&Wyk (1989), Seidel (1991))

A simple polygon with n vertices can be triangulated in expected $O(n \log^* n)$ time by means of randomization.

Computing Constrained Triangulations

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Corollary 188 (Chazelle (1991))

In $O(n)$ time we can check whether a polygon is simple.

Theorem 189 (Clarkson&Tarjan&Wyk (1989), Seidel (1991))

A simple polygon with n vertices can be triangulated in expected $O(n \log^* n)$ time by means of randomization.

Theorem 190 (Amato&Goodrich&Ramos (2001))

A simple polygon with n vertices can be triangulated in expected $O(n)$ time by means of randomization.

Computing Constrained Triangulations

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Corollary 188 (Chazelle (1991))

In $O(n)$ time we can check whether a polygon is simple.

Theorem 189 (Clarkson&Tarjan&Wyk (1989), Seidel (1991))

A simple polygon with n vertices can be triangulated in expected $O(n \log^* n)$ time by means of randomization.

Theorem 190 (Amato&Goodrich&Ramos (2001))

A simple polygon with n vertices can be triangulated in expected $O(n)$ time by means of randomization.

- The algorithms by Chazelle and Amato et alii are considered impractical.

Computing Constrained Triangulations

Theorem 187 (Chazelle (1991))

A simple polygon with n vertices can be triangulated in optimal $O(n)$ time.

Corollary 188 (Chazelle (1991))

In $O(n)$ time we can check whether a polygon is simple.

Theorem 189 (Clarkson&Tarjan&Wyk (1989), Seidel (1991))

A simple polygon with n vertices can be triangulated in expected $O(n \log^* n)$ time by means of randomization.

Theorem 190 (Amato&Goodrich&Ramos (2001))

A simple polygon with n vertices can be triangulated in expected $O(n)$ time by means of randomization.

- The algorithms by Chazelle and Amato et alii are considered impractical.
- The implementation of Seidel's algorithm by Narkhede&Manocha (1995) is surprisingly slow.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Computing Constrained Delaunay Triangulations

Theorem 191 (Chew (1989))

A constrained Delaunay triangulation of an admissible set S of n vertices and straight-line segments can be computed in optimal $O(n \log n)$ time.

Computing Constrained Delaunay Triangulations

Theorem 191 (Chew (1989))

A constrained Delaunay triangulation of an admissible set S of n vertices and straight-line segments can be computed in optimal $O(n \log n)$ time.

Theorem 192 (Chin&Wang (1998))

A constrained Delaunay triangulation of a simple polygon with n vertices can be computed in optimal $O(n)$ time.

Computing Constrained Delaunay Triangulations

Theorem 191 (Chew (1989))

A constrained Delaunay triangulation of an admissible set S of n vertices and straight-line segments can be computed in optimal $O(n \log n)$ time.

Theorem 192 (Chin&Wang (1998))

A constrained Delaunay triangulation of a simple polygon with n vertices can be computed in optimal $O(n)$ time.

- The algorithm by Chin&Wang is far too complicated to be of practical relevance.

Polygon Triangulation via Ear-Clipping

Definition 193 (Ear)

Three consecutive vertices (u, v, w) of a simple polygon P form an *ear* of P if $\overline{uw}$ constitutes a diagonal of P.

Polygon Triangulation via Ear-Clipping

Definition 193 (Ear)

Three consecutive vertices (u, v, w) of a simple polygon P form an *ear* of P if $\overline{uw}$ constitutes a diagonal of P.

Lemma 194 (Meisters (1975))

Every simple polygon with four or more vertices has at least two non-overlapping ears.

Polygon Triangulation via Ear-Clipping

Definition 193 (Ear)

Three consecutive vertices (u, v, w) of a simple polygon P form an *ear* of P if $\overline{uw}$ constitutes a diagonal of P.

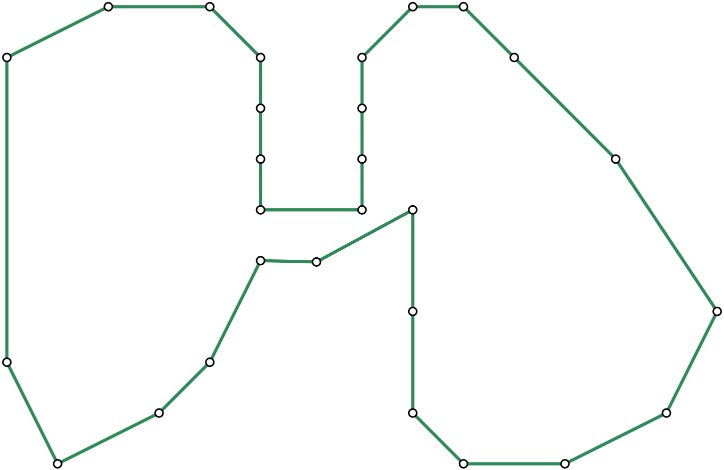
Lemma 194 (Meisters (1975))

Every simple polygon with four or more vertices has at least two non-overlapping ears.

- Simple triangulation algorithm:
  - Find an ear of P and clip it.
  - Repeat the ear clipping until only one triangle is left.
- An ear-clipping operation transforms an n-gon into an $(n-1)$-gon.

Animation of Ear Clipping

- Find an ear, and clip it. Keep clipping ears, until triangulation is finished.



Polygon Triangulation via Ear-Clipping

- Complexity of naïve algorithm: $O(n^3)$.
  - $O(n)$ time to check whether a triple of consecutive vertices forms an ear.
  - $O(n)$ many checks to find next ear.
  - $O(n)$ many ears needed.

Polygon Triangulation via Ear-Clipping

- Complexity of naïve algorithm: $O(n^3)$.
  - $O(n)$ time to check whether a triple of consecutive vertices forms an ear.
  - $O(n)$ many checks to find next ear.
  - $O(n)$ many ears needed.
- Observation: The clipping of one ear can change the earity status of at most two other triples of vertices of P.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Polygon Triangulation via Ear-Clipping

- Complexity of naïve algorithm: $O(n^3)$.
  - $O(n)$ time to check whether a triple of consecutive vertices forms an ear.
  - $O(n)$ many checks to find next ear.
  - $O(n)$ many ears needed.
- Observation: The clipping of one ear can change the earity status of at most two other triples of vertices of P.
- Thus, the overall complexity can be reduced to $O(n^2)$.

Lemma 195

Ear clipping computes a triangulation of a simple n-gon in $O(n^2)$ time.

Polygon Triangulation via Ear-Clipping

Lemma 196

Three consecutive vertices (u, v, w) of P form an ear of P if and only if

1. v is a convex vertex,
2. the triangle $\Delta(u, v, w)$ contains no reflex vertex of P, except for u or w if they are reflex.

Lemma 196

Three consecutive vertices (u, v, w) of P form an ear of P if and only if

1. v is a convex vertex,
2. the triangle $\Delta(u, v, w)$ contains no reflex vertex of P, except for u or w if they are reflex.

Corollary 197

Ear clipping computes a triangulation of a simple n-gon in $O(n \cdot r)$ time, where r is the number of its reflex vertices.

Polygon Triangulation via Ear-Clipping

Lemma 196

Three consecutive vertices (u, v, w) of P form an ear of P if and only if

1. v is a convex vertex,
2. the triangle $\Delta(u, v, w)$ contains no reflex vertex of P, except for u or w if they are reflex.

Corollary 197

Ear clipping computes a triangulation of a simple n-gon in $O(n \cdot r)$ time, where r is the number of its reflex vertices.

- [Held (2001)]: A triangulation algorithm based on ear-clipping and geometric hashing can be engineered to run in near-linear time, beating implementations of theoretically better algorithms on thousands of synthetic and real-world data sets. $\longrightarrow$ "Fast Industrial-Strength Triangulation" (FIST).

Polygon Triangulation via Ear-Clipping

Lemma 196

Three consecutive vertices (u, v, w) of P form an ear of P if and only if

1. v is a convex vertex,
2. the triangle $\Delta(u, v, w)$ contains no reflex vertex of P, except for u or w if they are reflex.

Corollary 197

Ear clipping computes a triangulation of a simple n-gon in $O(n \cdot r)$ time, where r is the number of its reflex vertices.

- [Held (2001)]: A triangulation algorithm based on ear-clipping and geometric hashing can be engineered to run in near-linear time, beating implementations of theoretically better algorithms on thousands of synthetic and real-world data sets. $\longrightarrow$ "Fast Industrial-Strength Triangulation" (FIST).
- [Eder&Held&Palfrader (2018)]: A coarse-grain parallelization of FIST's ear-clipping algorithm achieves a speedup of about 2–3 for four threads and about 3–4 for eight threads. Also parallel edge flipping to obtain a CDT is possible.

7 Triangulations

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Caveats for 3D Triangulations: Number of Tetrahedra

Theorem 198

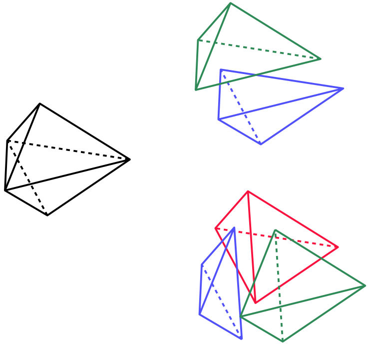
The number of tetrahedra contained in a triangulation of points in $\mathbb{R}^3$ may vary.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Caveats for 3D Triangulations: Number of Tetrahedra

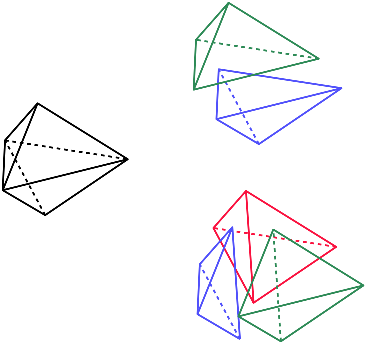
Theorem 198

The number of tetrahedra contained in a triangulation of points in $\mathbb{R}^3$ may vary.



Theorem 198

The number of tetrahedra contained in a triangulation of points in $\mathbb{R}^3$ may vary.



Theorem 199

A triangulation of n points in $\mathbb{R}^3$ can have $\Theta(n^2)$ many tetrahedra.

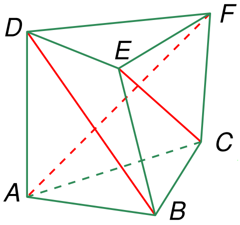
Theorem 200

Not every polyhedron can be triangulated.

Theorem 200

Not every polyhedron can be triangulated.

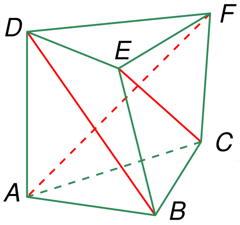
Proof: The triangle $\Delta(D, E, F)$ of Schönhardt's polyhedron (Math. Annalen, 1928) is rotated relative to $\Delta(A, B, C)$, causing the three red edges BD, CE and AF to become reflex. $\square$



Theorem 200

Not every polyhedron can be triangulated.

Proof : The triangle $\Delta(D, E, F)$ of Schönhardt's polyhedron (Math. Annalen, 1928) is rotated relative to $\Delta(A, B, C)$, causing the three red edges BD, CE and AF to become reflex. □



Theorem 201 (Ruppert&Seidel (1992))

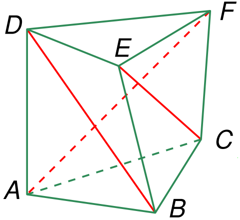
It is $\mathcal{NP}$-complete to determine whether a polyhedron requires Steiner points for triangulation. (And this result holds even for star-shaped polyhedra!)

Caveats for 3D Triangulations: Does a Triangulation Exist?

Theorem 200

Not every polyhedron can be triangulated.

Proof : The triangle $\Delta(D, E, F)$ of Schönhardt's polyhedron (Math. Annalen, 1928) is rotated relative to $\Delta(A, B, C)$, causing the three red edges BD, CE and AF to become reflex. □



Theorem 201 (Ruppert&Seidel (1992))

It is $\mathcal{NP}$-complete to determine whether a polyhedron requires Steiner points for triangulation. (And this result holds even for star-shaped polyhedra!)

Theorem 202 (Barequet et al. (1996))

It is $\mathcal{NP}$-complete to determine whether a non-plane polygon in $\mathbb{R}^3$ has a non-intersecting triangulation.

Caveats for 3D Triangulations: Many Steiner Points Required

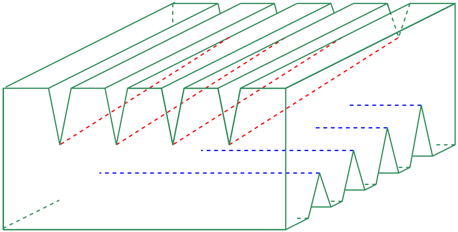
Theorem 203 (Chazelle (1984))

There exist polyhedra with n vertices that require $\Omega(n^2)$ Steiner points.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theorem 203 (Chazelle (1984))

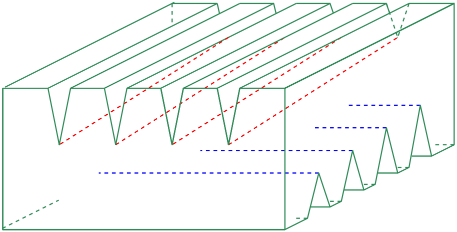
There exist polyhedra with n vertices that require $\Omega(n^2)$ Steiner points.



Caveats for 3D Triangulations: Many Steiner Points Required

Theorem 203 (Chazelle (1984))

There exist polyhedra with n vertices that require $\Omega(n^2)$ Steiner points.



Theorem 204 (Chazelle&Palios (1990))

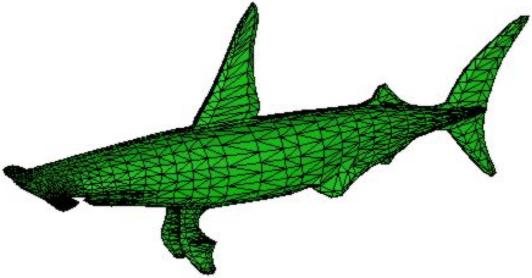
Every simple polyhedron with n vertices and r reflex edges can be triangulated using $O(n + r^2)$ Steiner points and a total of $O(n + r^2)$ tetrahedra.

7 Triangulations

- Basics
- Computing Constrained Triangulations
- Triangulations in 3D
- Applications of Triangulations
  - Rendering
  - Triangulated Irregular Network
  - Visibility Determination
  - Minimum Convex Decomposition
  - Topologically Consistent Watermarking

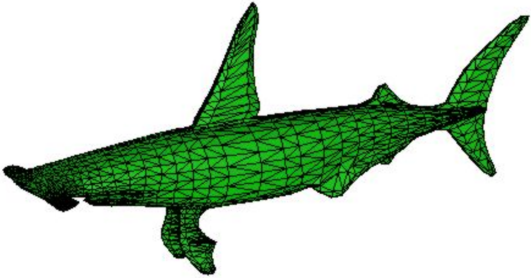
Rendering

- Graphics hardware is best at handling triangles rather than more general geometric primitives. Thus, the surfaces of 3D models need to be triangulated.



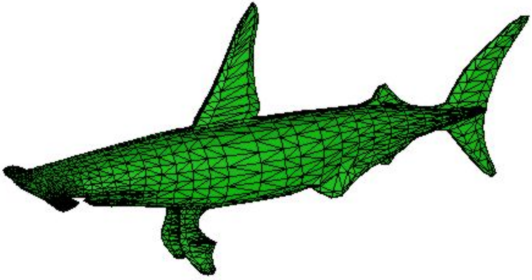
Rendering

- Graphics hardware is best at handling triangles rather than more general geometric primitives. Thus, the surfaces of 3D models need to be triangulated.
- Euler's Theorem applies to the faces, edges and vertices of a polyhedron.



Rendering

- Graphics hardware is best at handling triangles rather than more general geometric primitives. Thus, the surfaces of 3D models need to be triangulated.
- Euler's Theorem applies to the faces, edges and vertices of a polyhedron.

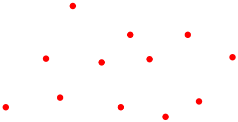


Caveat

3D polyhedral models used for graphics purposes tend to exhibit all types of "problems"!

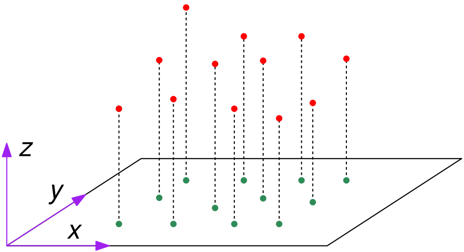
Triangulated Irregular Network

- Given is a set of sample points on a 3D terrain. How can we model the actual terrain surface by interpolating those points?



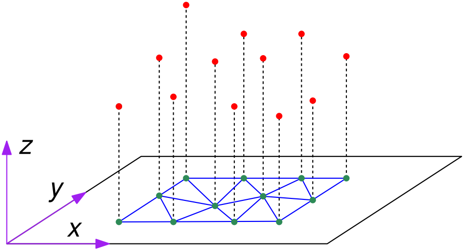
Triangulated Irregular Network

- Given is a set of sample points on a 3D terrain. How can we model the actual terrain surface by interpolating those points?
- Natural approach:
  1. Project the points onto 2D (by discarding their z-coordinate).



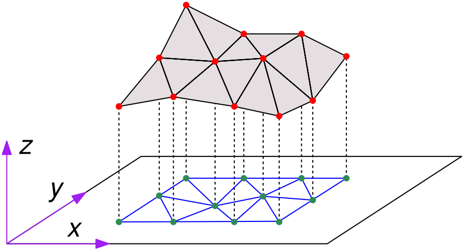
Triangulated Irregular Network

- Given is a set of sample points on a 3D terrain. How can we model the actual terrain surface by interpolating those points?
- Natural approach:
  1. Project the points onto 2D (by discarding their z-coordinate).
  2. Compute a triangulation $\mathcal{T}$ of the projected points.



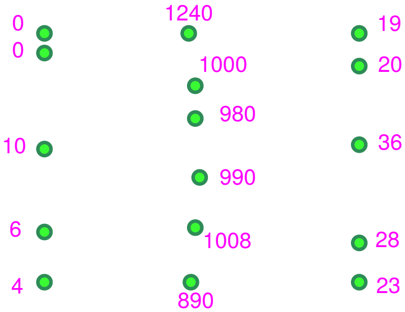
Triangulated Irregular Network

- Given is a set of sample points on a 3D terrain. How can we model the actual terrain surface by interpolating those points?
- Natural approach:
  1. Project the points onto 2D (by discarding their z-coordinate).
  2. Compute a triangulation $\mathcal{T}$ of the projected points.
  3. Lift $\mathcal{T}$ back to 3D.
- Known to the GIS community as *triangulated irregular network* (TIN).



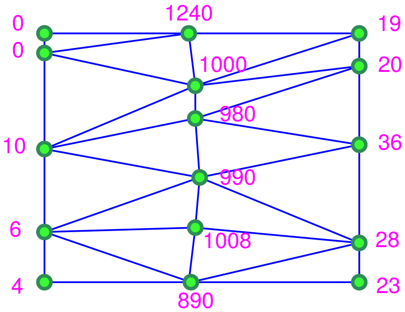
Triangulated Irregular Network

- Note: The shape of the terrain depends heavily on the 2D triangulation!



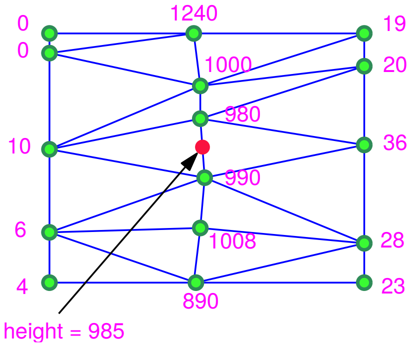
Triangulated Irregular Network

- Note: The shape of the terrain depends heavily on the 2D triangulation!

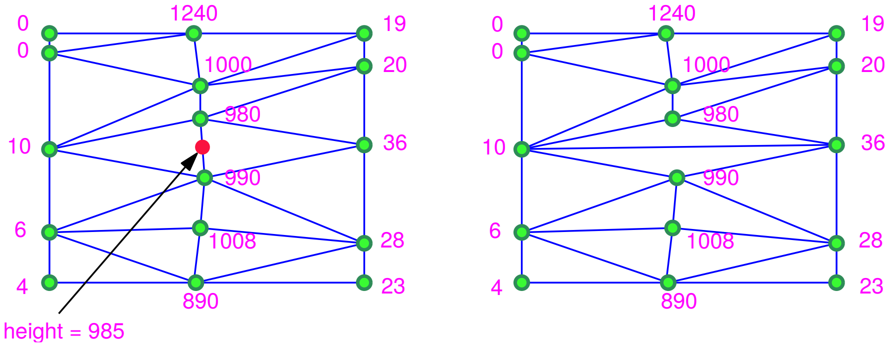


Triangulated Irregular Network

- Note: The shape of the terrain depends heavily on the 2D triangulation!

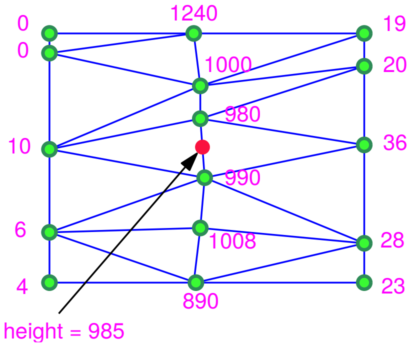


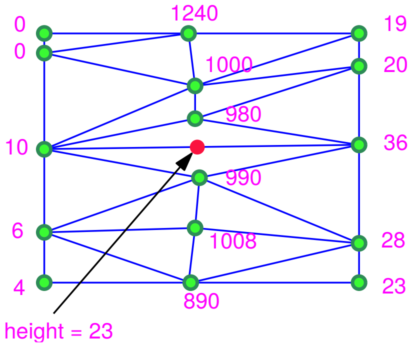
- Note: The shape of the terrain depends heavily on the 2D triangulation!



Triangulated Irregular Network

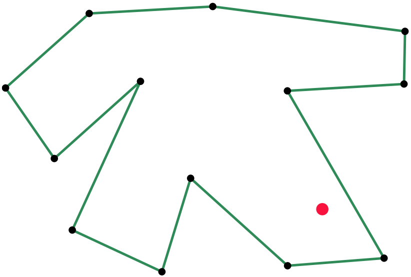
- Note: The shape of the terrain depends heavily on the 2D triangulation!



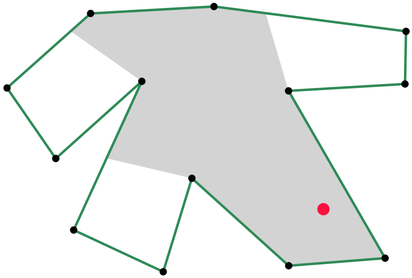


Visibility Determination Within Triangulated Environments

- Which portion of the green polygon is visible from the red point?

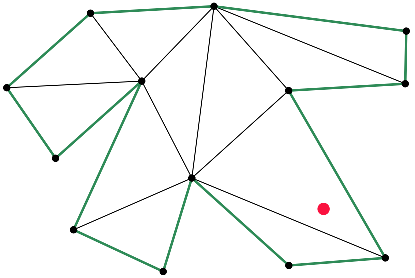


- Which portion of the green polygon is visible from the red point?

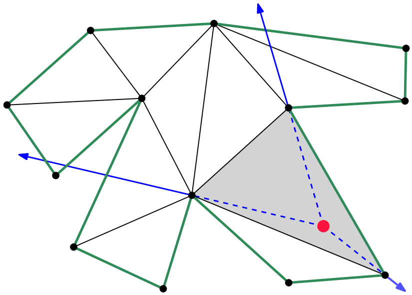


Visibility Determination Within Triangulated Environments

- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.

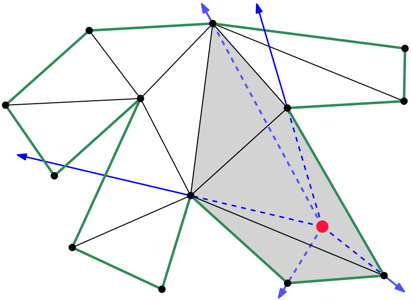


- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.
- The triangle that contains the red point is illuminated.



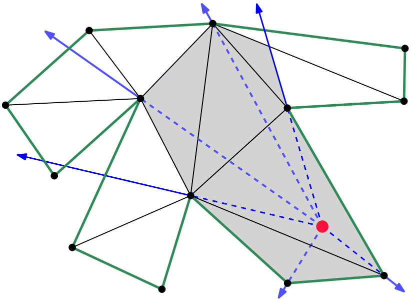
Visibility Determination Within Triangulated Environments

- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.
- The triangle that contains the red point is illuminated.
- Neighboring triangles are illuminated and new visibility rays generated.



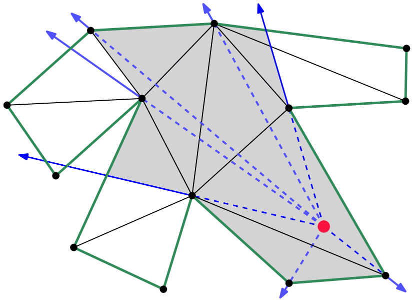
Visibility Determination Within Triangulated Environments

- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.
- The triangle that contains the red point is illuminated.
- Neighboring triangles are illuminated and new visibility rays generated.

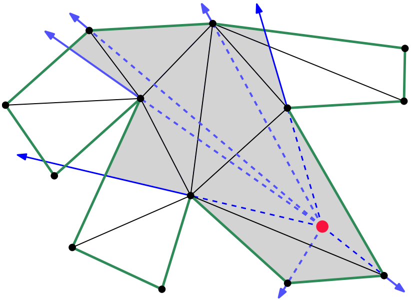


Visibility Determination Within Triangulated Environments

- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.
- The triangle that contains the red point is illuminated.
- Neighboring triangles are illuminated and new visibility rays generated.

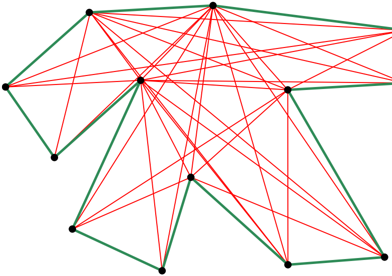


- Which portion of the green polygon is visible from the red point?
- We start with an arbitrary triangulation of the polygon.
- The triangle that contains the red point is illuminated.
- Neighboring triangles are illuminated and new visibility rays generated.
- All triangles which are at least partially visible have been traversed.



Definition 205 (Visibility graph, Dt.: Sichtbarkeitsgraph)

The *visibility graph* inside an n-gon P consists of the vertices of P as nodes which are connected by edges if the vertices can see each other.

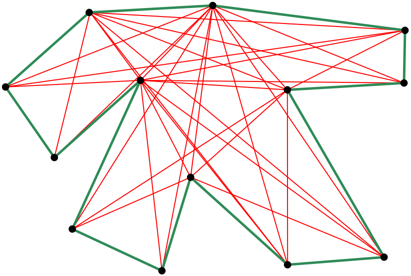


Definition 205 (Visibility graph, Dt.: Sichtbarkeitsgraph)

The *visibility graph* inside an n-gon P consists of the vertices of P as nodes which are connected by edges if the vertices can see each other.

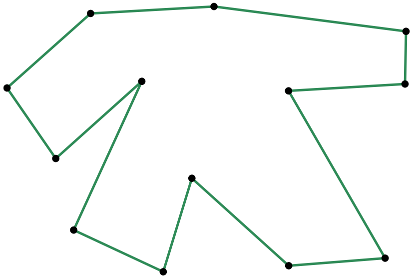
Theorem 206 (Ghosh&Mount (1991))

The full visibility graph inside an n-gon can be computed in time $O(|E| + n \log n)$, where $|E|$ is the size of the visibility graph.



Problem: ArtGalleryProblem

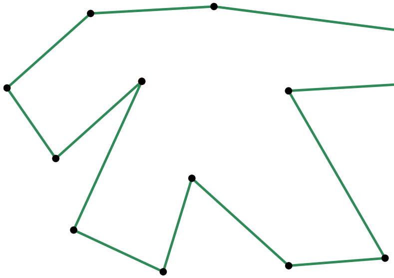
Given: A simple polygon P.



Problem: ArtGalleryProblem

Given: A simple polygon P.

Select: A minimum number of vertices of P such that every point of P can be seen from at least one of the vertices selected.

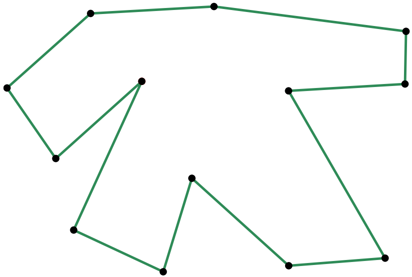


Problem: ArtGalleryProblem

Given: A simple polygon P.

Select: A minimum number of vertices of P such that every point of P can be seen from at least one of the vertices selected.

- Posed by Klee in 1973.

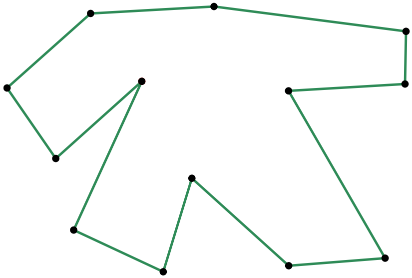


Problem: ArtGalleryProblem

Given: A simple polygon P.

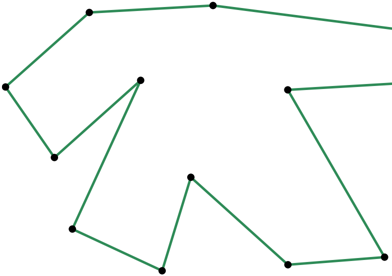
Select: A minimum number of vertices of P such that every point of P can be seen from at least one of the vertices selected.

- Posed by Klee in 1973.
- The problem is $\mathcal{NP}$-hard.



Theorem 207 (Chvátal (1975))

To guard a simple polygon with n vertices, $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary.

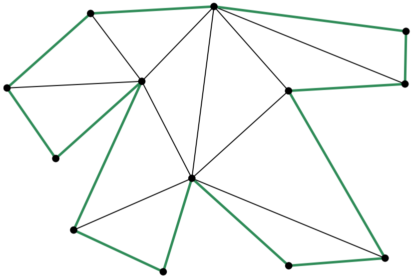


Theorem 207 (Chvátal (1975))

To guard a simple polygon with n vertices, $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary.

Sketch of Proof by Fisk (1978) : Consider an (arbitrary) triangulation of the polygon.

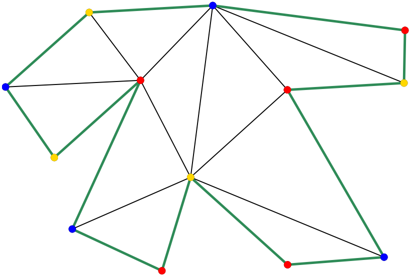
□



Theorem 207 (Chvátal (1975))

To guard a simple polygon with n vertices, $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary.

Sketch of Proof by Fisk (1978) : Consider an (arbitrary) triangulation of the polygon. One can show that the vertices of the triangulation graph are 3-colorable. Of course, vertices of one color form a valid set of guards. □

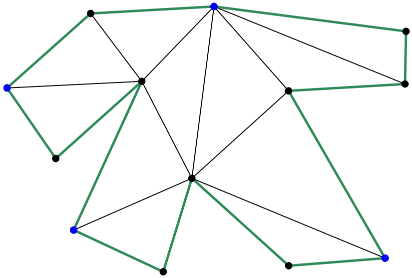


Guarding an Art Gallery

Theorem 207 (Chvátal (1975))

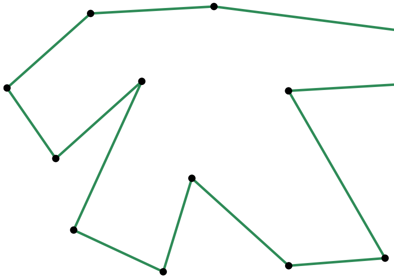
To guard a simple polygon with n vertices, $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary.

Sketch of Proof by Fisk (1978) : Consider an (arbitrary) triangulation of the polygon. One can show that the vertices of the triangulation graph are 3-colorable. Of course, vertices of one color form a valid set of guards. Hence, the color with the fewest vertices yields a valid set of at most $\lfloor n/3 \rfloor$ guards. □



Problem: MinimumConvexDecomposition

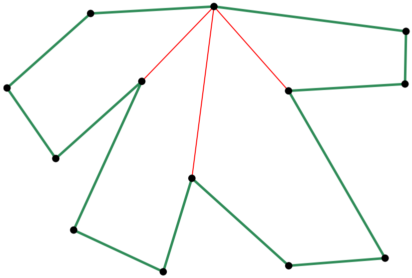
Given: A simple polygon P.



Problem: MINIMUMCONVEXDECOMPOSITION

Given: A simple polygon P.

Compute: A minimum number of polygonal convex areas whose vertices match the vertices of P, whose interiors are disjoint and whose union equals P.



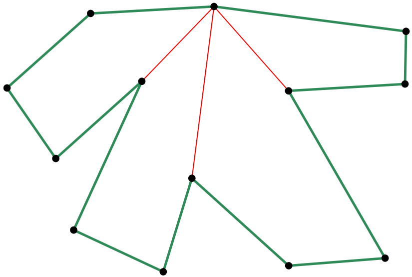
Problem: MinimumConvexDecomposition

Given: A simple polygon P.

Compute: A minimum number of polygonal convex areas whose vertices match the vertices of P, whose interiors are disjoint and whose union equals P.

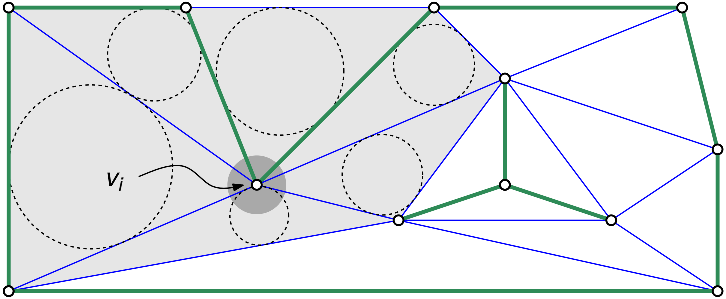
Theorem 208 (Hertel&Mehlhorn (1983))

If a triangulation of P is given then an approximate convex decomposition with at most four times the minimum number of convex pieces can be obtained in linear time.



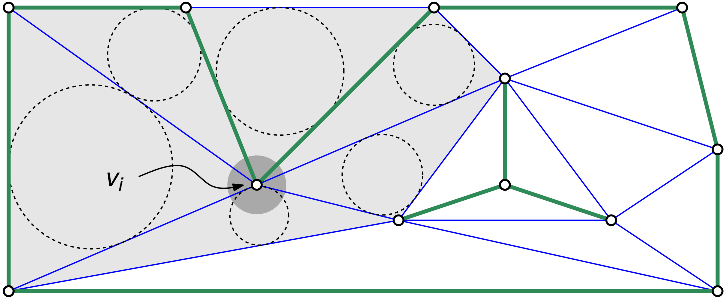
Topology-Preserving Watermarking of Vector Graphics

- [Huber et al. (2014)] compute for each vertex a disk-shaped maximum perturbation region (MPR), based on the radii of the inscribed circles of a constrained triangulation of the input.
- Perturbing the vertices within their MPRs causes the edges to stay within their hoses and allows to preserve the input topology.



Topology-Preserving Watermarking of Vector Graphics

- [Huber et al. (2014)] compute for each vertex a disk-shaped maximum perturbation region (MPR), based on the radii of the inscribed circles of a constrained triangulation of the input.
- Perturbing the vertices within their MPRs causes the edges to stay within their hoses and allows to preserve the input topology.
- This scheme can be extended to 3D.



8 Robustness Problems and Real-World Issues

- Theory and Practice
- Introduction to Robustness Problems
- Approaches to Achieving Robustness
- Improving the Reliability of Floating-Point Code
- Real-World Applications and Experiences

8 Robustness Problems and Real-World Issues

Theory and Practice

- Voronoi diagrams of points and segments and/or arcs are easy to understand, but notorious for being difficult to implement reliably. No surprise that very few decent Voronoi codes are known.

Theory and Practice

- Voronoi diagrams of points and segments and/or arcs are easy to understand, but notorious for being difficult to implement reliably. No surprise that very few decent Voronoi codes are known.
- The situation is similar for many other algorithms of computational geometry. That is, there is a gap between theory and practice . . .

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Theory and Practice

- Voronoi diagrams of points and segments and/or arcs are easy to understand, but notorious for being difficult to implement reliably. No surprise that very few decent Voronoi codes are known.
- The situation is similar for many other algorithms of computational geometry. That is, there is a gap between theory and practice . . .

Benjamin Brewster ("The Yale Literary Magazine" 1882)

In theory, there is no difference between theory and practice. In practice, there is.

Marie von Ebner-Eschenbach (1893)

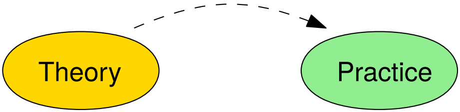
Theorie und Praxis sind eins wie Seele und Leib, und wie Seele und Leib liegen sie großenteils miteinander in Streit.

Jan L.A. van de Snepscheut

The difference between theory and practice is larger in practice than the difference between theory and practice in theory.

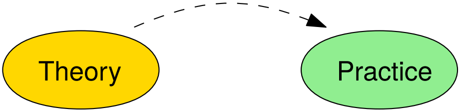
Theory Into Practice

- What we have.



Theory Into Practice

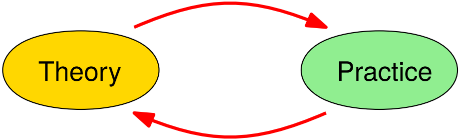
- What we have.



Ayn Rand (Russian-born American writer and philosopher)

Those who say that theory and practice are two unrelated realms are fools in one and scoundrels in the other.

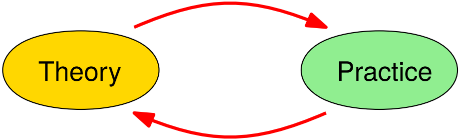
- What we have. What we'd need . . .



Ayn Rand (Russian-born American writer and philosopher)

Those who say that theory and practice are two unrelated realms are fools in one and scoundrels in the other.

- What we have. What we'd need . . .



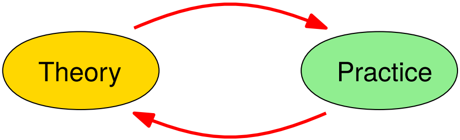
Ayn Rand (Russian-born American writer and philosopher)

Those who say that theory and practice are two unrelated realms are fools in one and scoundrels in the other.

Folklore ☻

Theory is when you know everything but nothing works. Practice is when everything works but no one knows why.

- What we have. What we'd need . . .



Ayn Rand (Russian-born American writer and philosopher)

Those who say that theory and practice are two unrelated realms are fools in one and scoundrels in the other.

Folklore ☻

Theory is when you know everything but nothing works. Practice is when everything works but no one knows why. However, we combine theory and practice: Nothing works and no one knows why.

8 Robustness Problems and Real-World Issues

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Arithmetic

- Computers employ floating-point arithmetic to perform real arithmetic.
- No matter how many bits are used, floating-point arithmetic represents a number by a fixed-length binary mantissa and an exponent of fixed size.

Floating-Point Arithmetic

- Computers employ floating-point arithmetic to perform real arithmetic.
- No matter how many bits are used, floating-point arithmetic represents a number by a fixed-length binary mantissa and an exponent of fixed size. E.g., according to the IEEE 754 standard, we have 23 bits for the mantissa and 8 bits for the exponent in case of 32-bit floats, and 52 bits and 11 bits in case of 64-bit doubles.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Arithmetic

- Computers employ floating-point arithmetic to perform real arithmetic.
- No matter how many bits are used, floating-point arithmetic represents a number by a fixed-length binary mantissa and an exponent of fixed size. E.g., according to the IEEE 754 standard, we have 23 bits for the mantissa and 8 bits for the exponent in case of 32-bit floats, and 52 bits and 11 bits in case of 64-bit doubles.
- Thus, only a finite number of values within a finite sub-interval of $\mathbb{R}$ can be represented accurately; all other values have to be rounded to the closest number that is representable.

Floating-Point Arithmetic

- Computers employ floating-point arithmetic to perform real arithmetic.
- No matter how many bits are used, floating-point arithmetic represents a number by a fixed-length binary mantissa and an exponent of fixed size. E.g., according to the IEEE 754 standard, we have 23 bits for the mantissa and 8 bits for the exponent in case of 32-bit floats, and 52 bits and 11 bits in case of 64-bit doubles.
- Thus, only a finite number of values within a finite sub-interval of $\mathbb{R}$ can be represented accurately; all other values have to be rounded to the closest number that is representable.
- The IEEE 754 standard for floating-point arithmetic knows four different rounding modes. The first mode is the default; the others are called directed roundings.

  Round to Nearest

  Round towards 0

  Round towards $+\infty$

  Round towards $-\infty$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Arithmetic

- Computers employ floating-point arithmetic to perform real arithmetic.
- No matter how many bits are used, floating-point arithmetic represents a number by a fixed-length binary mantissa and an exponent of fixed size. E.g., according to the IEEE 754 standard, we have 23 bits for the mantissa and 8 bits for the exponent in case of 32-bit floats, and 52 bits and 11 bits in case of 64-bit doubles.
- Thus, only a finite number of values within a finite sub-interval of $\mathbb{R}$ can be represented accurately; all other values have to be rounded to the closest number that is representable.
- The IEEE 754 standard for floating-point arithmetic knows four different rounding modes. The first mode is the default; the others are called directed roundings.
  Round to Nearest
  Round towards 0
  Round towards $+\infty$
  Round towards $-\infty$

Chuck Allison

Floating-point numbers are not real numbers [...]. Real numbers have infinite precision and are therefore continuous and non-lossy; floating-point numbers have limited precision, so they are finite, and they resemble "badly behaved" integers, because they are not evenly spaced throughout their range.

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.

  - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
  - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.

  - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
  - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!
  - What do we get if we assign $2^{24} + 1 = 16777217$ to a 32-bit float?

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.
  - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
  - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!
  - What do we get if we assign $2^{24} + 1 = 16777217$ to a 32-bit float? We get 16777216!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.
  - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
  - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!
  - What do we get if we assign $2^{24} + 1 = 16777217$ to a 32-bit float? We get 16777216!
  - Integers between 2^n and 2^{n+1} round to a multiple of 2^{n-23} when assigned to a float.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.
  - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
  - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!
  - What do we get if we assign $2^{24} + 1 = 16777217$ to a 32-bit float? We get 16777216!
  - Integers between 2^n and 2^{n+1} round to a multiple of 2^{n-23} when assigned to a float.

  Round-off error: It arises from rounding results of floating-point computations during an algorithm.
  - E.g., $\sqrt{2}$ cannot be represented exactly since $\sqrt{2}$ is an irrational number.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Floating-Point Errors

- There are two sources of error for floating-point computations: input error and round-off error.

  Input error: It arises from reading/assigning a value to a floating-point variable.
    - It is well-known that $\frac{1}{3}$ cannot be represented by a finite sum of powers of 10.
    - Similarly, 0.1 cannot be represented by a finite sum of powers of 2!
    - What do we get if we assign $2^{24} + 1 = 16777217$ to a 32-bit float? We get 16777216!
    - Integers between 2^n and 2^{n+1} round to a multiple of 2^{n-23} when assigned to a float.

  Round-off error: It arises from rounding results of floating-point computations during an algorithm.
    - E.g., $\sqrt{2}$ cannot be represented exactly since $\sqrt{2}$ is an irrational number.
- While one can instruct the C command `printf` to print, say, 57 digits after the decimal separator, one will "only" get the digits of the closest value that is representable:

$$1/3 = 0.333333333333333314829616256247390992939472198486328125000$$

$$1/10 = 0.100000000000000005551115123125782702118158340454101562500$$

Machine Precision

- The round-off error is bounded in terms of the *machine precision*, ε, which is the smallest value satisfying

$$|fp(a \circ b) - (a \circ b)| \leq \varepsilon |a \circ b|$$

for all floating-point numbers a, b and any of the four operations $+, -, \cdot, /$ instead of $\circ$, for which $a \circ b$ does not cause an underflow or an overflow.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Machine Precision

- The round-off error is bounded in terms of the *machine precision*, ε, which is the smallest value satisfying

$$|fp(a \circ b) - (a \circ b)| \leq \varepsilon |a \circ b|$$

for all floating-point numbers a, b and any of the four operations $+, -, \cdot, /$ instead of $\circ$, for which $a \circ b$ does not cause an underflow or an overflow.

- On IEEE-754 machines, $\varepsilon = 2^{-23} \approx 1.19 \cdot 10^{-7}$ for floats, and $\varepsilon = 2^{-52} \approx 2.22 \cdot 10^{-16}$ for doubles.
- On some exotic platform, ε can be determined approximately by finding the smallest positive value x such that $1 + x \neq 1$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Machine Precision

- The round-off error is bounded in terms of the *machine precision*, ε, which is the smallest value satisfying

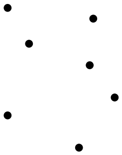
$$|fp(a \circ b) - (a \circ b)| \leq \varepsilon |a \circ b|$$

  for all floating-point numbers a, b and any of the four operations $+, -, \cdot, /$ instead of $\circ$, for which $a \circ b$ does not cause an underflow or an overflow.
- On IEEE-754 machines, $\varepsilon = 2^{-23} \approx 1.19 \cdot 10^{-7}$ for floats, and $\varepsilon = 2^{-52} \approx 2.22 \cdot 10^{-16}$ for doubles.
- On some exotic platform, ε can be determined approximately by finding the smallest positive value x such that $1 + x \neq 1$.
- Note: Some compilers promote floats to doubles!

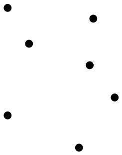
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

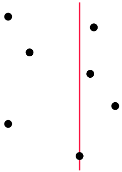
Geometric Predicates

Sorting



Sorting

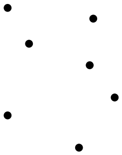




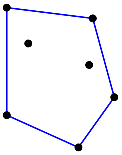
coordinate comparison

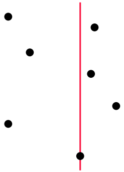
$x_2 - x_1 > 0?$

Sorting



Convex Hull

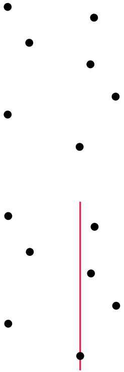




coordinate comparison

$x_2 - x_1 > 0?$

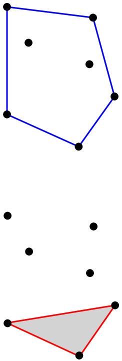
Sorting



coordinate comparison

$$x_2 - x_1 > 0?$$

Convex Hull



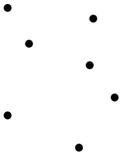
sidedness test

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0?$$

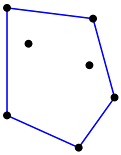
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Geometric Predicates

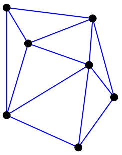
Sorting

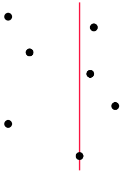


Convex Hull



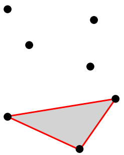
Delaunay Triangulation





coordinate comparison

$x_2 - x_1 > 0?$

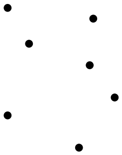


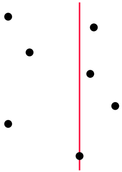
sidedness test

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0?$$

Geometric Predicates

Sorting

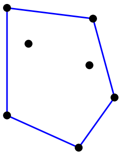


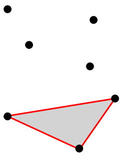


coordinate comparison

$$x_2 - x_1 > 0?$$

Convex Hull

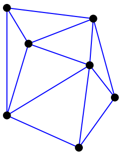


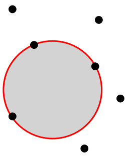


sidedness test

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0?$$

Delaunay Triangulation





incircle test

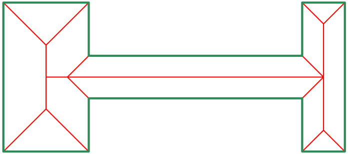
$$\begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{vmatrix} > 0?$$

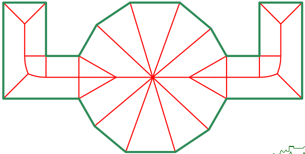
Degeneracy

- Degeneracies are caused by the special position of two or more geometric objects.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

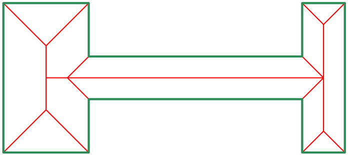
- Degeneracies are caused by the special position of two or more geometric objects. E.g.:
  - Two line segments that overlap partially rather than being disjoint or intersecting in a point.
  - Polygon edges that are parallel.
  - A Voronoi node of degree greater than three.

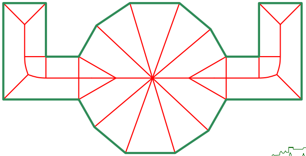




Degeneracy

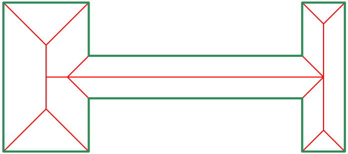
- Degeneracies are caused by the special position of two or more geometric objects. E.g.:
  - Two line segments that overlap partially rather than being disjoint or intersecting in a point.
  - Polygon edges that are parallel.
  - A Voronoi node of degree greater than three.
- Note: Degeneracies may be intentional, and real-world data should be assumed to be degenerate!

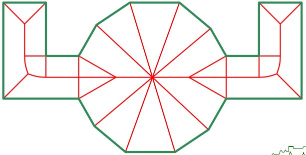




Degeneracy

- Degeneracies are caused by the special position of two or more geometric objects. E.g.:
  - Two line segments that overlap partially rather than being disjoint or intersecting in a point.
  - Polygon edges that are parallel.
  - A Voronoi node of degree greater than three.
- Note: Degeneracies may be intentional, and real-world data should be assumed to be degenerate!
- The net result of degeneracies is a vastly increased number of so-called *special cases*.

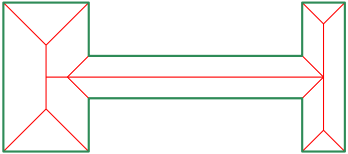


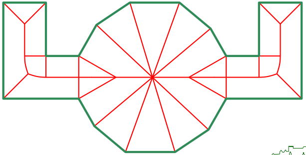


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Degeneracy Versus Numerical Precision

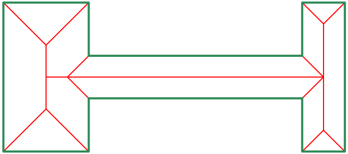
- How can we handle the special cases correctly?
- How can we even be sure that all special cases have been modeled?

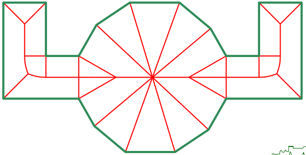




Degeneracy Versus Numerical Precision

- How can we handle the special cases correctly?
- How can we even be sure that all special cases have been modeled?
- Typically, a degeneracy occurs if a predicate evaluates to zero.
- And, typically, predicates are evaluated by floating-point arithmetic.





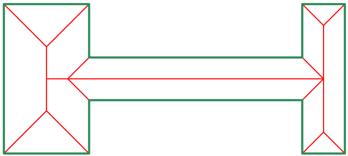
Degeneracy Versus Numerical Precision

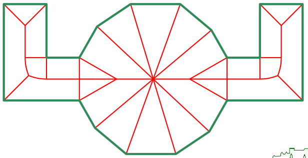
- How can we handle the special cases correctly?
- How can we even be sure that all special cases have been modeled?
- Typically, a degeneracy occurs if a predicate evaluates to zero.
- And, typically, predicates are evaluated by floating-point arithmetic.

If a predicate evaluates to a value close to zero . . .

Is it a special case or simply a numerical inaccuracy??

- The mere fact that a degeneracy cannot be classified reliably on a floating-point arithmetic complicates matters significantly.





UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Robustness Problem: Failure of Basic Mathematical Implications

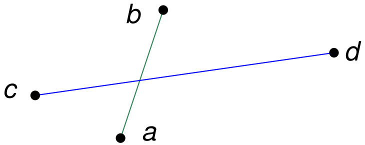
- Suppose that we are given two line segments $\overline{ab}$ and $\overline{cd}$ in the plane such that

$$c_x < a_x < b_x < d_x \qquad a_y < c_y < d_y < b_y.$$

Sample Robustness Problem: Failure of Basic Mathematical Implications

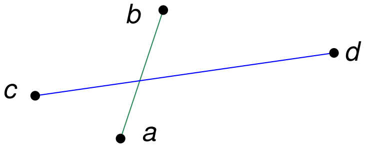
- Suppose that we are given two line segments $\overline{ab}$ and $\overline{cd}$ in the plane such that

$$c_x < a_x < b_x < d_x \qquad a_y < c_y < d_y < b_y.$$



- Suppose that we are given two line segments $\overline{ab}$ and $\overline{cd}$ in the plane such that

$$c_x < a_x < b_x < d_x \qquad a_y < c_y < d_y < b_y.$$

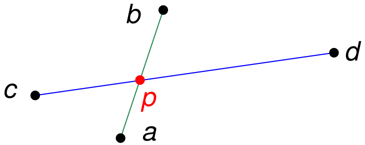


- It is easy to see that the two line segments intersect, without a or b lying on $\overline{cd}$ and without c or d lying on $\overline{ab}$. In particular, the line segments cannot overlap. Hence, the two line segments intersect in a point.

Sample Robustness Problem: Failure of Basic Mathematical Implications

- Suppose that we are given two line segments $\overline{ab}$ and $\overline{cd}$ in the plane such that

$$c_x < a_x < b_x < d_x \qquad a_y < c_y < d_y < b_y.$$



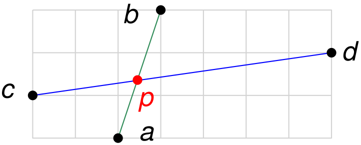
- It is easy to see that the two line segments intersect, without a or b lying on $\overline{cd}$ and without c or d lying on $\overline{ab}$. In particular, the line segments cannot overlap. Hence, the two line segments intersect in a point.
- Let $p := \overline{ab} \cap \overline{cd}$. Are the following inequalities guaranteed to be true?

$$a_x < p_x < b_x \qquad a_y < p_y < b_y \qquad c_x < p_x < d_x \qquad c_y < p_y < d_y$$

Sample Robustness Problem: Failure of Basic Mathematical Implications

- Suppose that we are given two line segments $\overline{ab}$ and $\overline{cd}$ in the plane such that

$$c_x < a_x < b_x < d_x \qquad a_y < c_y < d_y < b_y.$$



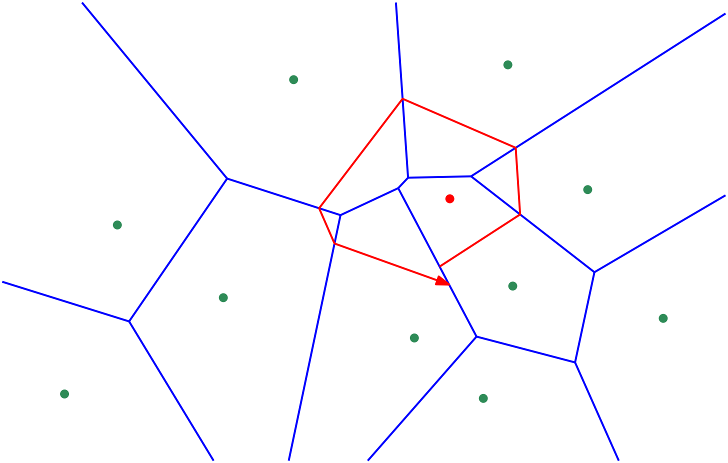
- It is easy to see that the two line segments intersect, without a or b lying on $\overline{cd}$ and without c or d lying on $\overline{ab}$. In particular, the line segments cannot overlap. Hence, the two line segments intersect in a point.
- Let $p := \overline{ab} \cap \overline{cd}$. Are the following inequalities guaranteed to be true?

$$a_x < p_x < b_x \qquad a_y < p_y < b_y \qquad c_x < p_x < d_x \qquad c_y < p_y < d_y$$

- Yes in theory, but no on a floating-point arithmetic!

Sample Robustness Problem: Lack of Global Consistency

- Local consistency need not imply global consistency.



- [Kettner et alii 2006] study the standard determinant-based orientation predicate on IEEE 754 floating-point arithmetic to check the sidedness of $(p_x + x \cdot u, p_y + y \cdot u)$ relative to two points q, r, for $0 \leq x, y \leq 255$ and with $u := 2^{-53}$:

$$\text{sign}\det \begin{pmatrix} p_x + x \cdot u & p_y + y \cdot u & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{pmatrix} \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} 0 \;?$$

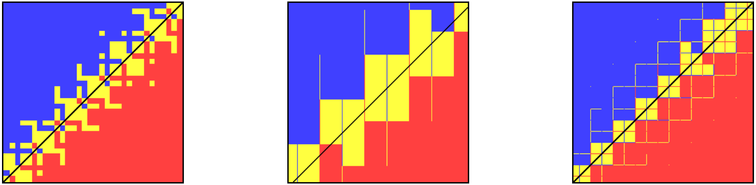
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Robustness Problem: Incorrect Orientation Predicate

- [Kettner et alii 2006] study the standard determinant-based orientation predicate on IEEE 754 floating-point arithmetic to check the sidedness of $(p_x + x \cdot u, p_y + y \cdot u)$ relative to two points q, r, for $0 \le x, y \le 255$ and with $u := 2^{-53}$:

$$\operatorname{sign}\det\begin{pmatrix} p_x + x\cdot u & p_y + y \cdot u & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{pmatrix} \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} 0 \;?$$

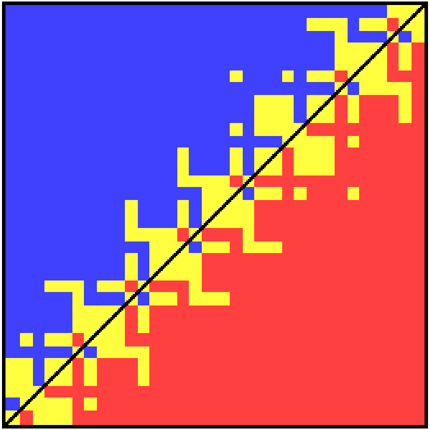
- The resulting 256×256 array of signs (as a function of x, y) is color-coded: A yellow (red, blue) pixel indicates collinear (negative, positive, resp.) orientation.
- The black line indicates the line through q and r.
- Note the sign inversions!



[Image credit: www.mpi-inf.mpg.de/~kettner/proj/NonRobust/]

- [Kettner et alii 2006]: A yellow (red, blue) pixel indicates collinear (negative, positive, resp.) orientation.

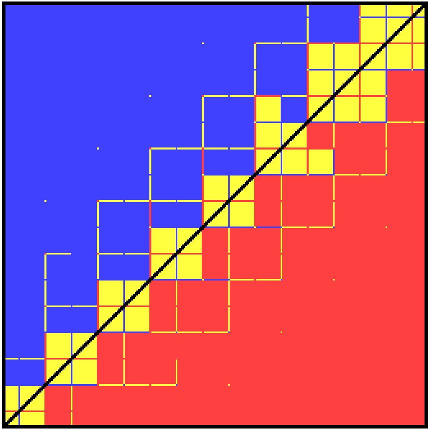
$$p := \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \quad q := \begin{pmatrix} 12 \\ 12 \end{pmatrix} \quad r := \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$



Sample Robustness Problem: Incorrect Orientation Predicate

- [Kettner et alii 2006]: A yellow (red, blue) pixel indicates collinear (negative, positive, resp.) orientation.

$$p := \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \quad q := \begin{pmatrix} 8.8000000000000007 \\ 8.8000000000000007 \end{pmatrix} \quad r := \begin{pmatrix} 12.1 \\ 12.1 \end{pmatrix}$$



8 Robustness Problems and Real-World Issues

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Approaches to Improving Robustness

- Several approaches have been proposed in recent years:
  - Error analysis.
  - Exact arithmetic (on integers, rationals, or even algebraic numbers).
  - Exact geometric computing.
  - Floating-point filters.
  - Symbolic perturbation.
  - Epsilon threshholds and sophisticated tolerancing.

Approaches to Improving Robustness

- Several approaches have been proposed in recent years:
  - Error analysis.
  - Exact arithmetic (on integers, rationals, or even algebraic numbers).
  - Exact geometric computing.
  - Floating-point filters.
  - Symbolic perturbation.
  - Epsilon threshholds and sophisticated tolerancing.
- No agreement on the best approach . . .
- All methods have shortcomings, typically also limited applicability — or they suffer from inefficiency!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Exact Arithmetic

- Input is assumed to be exact.
- Compute the numerical value of every predicate exactly, based on exact number types.
- Exact computation is possible if all numerical values are algebraic. (This is the case for most current problems in computational geometry.)

Exact Arithmetic

- Input is assumed to be exact.
- Compute the numerical value of every predicate exactly, based on exact number types.
- Exact computation is possible if all numerical values are algebraic. (This is the case for most current problems in computational geometry.)
- Note: We may no longer assume that each arithmetic operation takes constant time!
- Note: Constructors complicate the situation significantly!

Bigint: Arbitrary-precision integers.

- Usually based on Karatsuba's multiplication algorithm, with $\Theta(b^{\log_2 3}) \equiv \Theta(b^{1.58\cdots})$ complexity for multiplying b-bit numbers.
- See, e.g., GNU's GMP library `http://gmplib.org/`.

Exact Number Types

Bigint: Arbitrary-precision integers.

- Usually based on Karatsuba's multiplication algorithm, with $\Theta(b^{\log_2 3}) \equiv \Theta(b^{1.58\dots})$ complexity for multiplying b-bit numbers.
- See, e.g., GNU's GMP library `http://gmplib.org/`.

Bigrational: Quotients of bigints.

- Standard rational arithmetic.
- It is important to reduce fractions frequently.
- Euclid's algorithm can be used for finding common factors.
- See, e.g., GMP's `mpq_t` number type.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Exact Number Types

Bigint: Arbitrary-precision integers.

- Usually based on Karatsuba's multiplication algorithm, with $\Theta(b^{\log_2 3}) \equiv \Theta(b^{1.58\cdots})$ complexity for multiplying b-bit numbers.
- See, e.g., GNU's GMP library `http://gmplib.org/`.

Bigrational: Quotients of bigints.

- Standard rational arithmetic.
- It is important to reduce fractions frequently.
- Euclid's algorithm can be used for finding common factors.
- See, e.g., GMP's `mpq_t` number type.

Algebraic numbers: Roots of polynomials with bigint coefficients.

- Complexity of deciding equality grows with the number of operators allowed.
- [Caviness (1967)] proved undecidability of functional equivalence for moderately complicated terms.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Exact Number Types

Bigint: Arbitrary-precision integers.

- Usually based on Karatsuba's multiplication algorithm, with $\Theta(b^{\log_2 3}) \equiv \Theta(b^{1.58\ldots})$ complexity for multiplying b-bit numbers.
- See, e.g., GNU's GMP library `http://gmplib.org/`.

Bigrational: Quotients of bigints.

- Standard rational arithmetic.
- It is important to reduce fractions frequently.
- Euclid's algorithm can be used for finding common factors.
- See, e.g., GMP's `mpq_t` number type.

Algebraic numbers: Roots of polynomials with bigint coefficients.

- Complexity of deciding equality grows with the number of operators allowed.
- [Caviness (1967)] proved undecidability of functional equivalence for moderately complicated terms.

Homogeneous coordinates: Not exactly a number type.

- Can often be used to avoid divisions.
- It is important to reduce fractions frequently.
- Most predicates can be expressed directly in terms of homogeneous coordinates.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Do we really need exact arithmetic for solving geometric problems?

Exact Geometric Computing

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.
- Error analysis is needed to come up with *root bounds*: A positive number r forms a root bound for a predicate p if $|p(x)| > r$ guarantees that the evaluation of p for x yields the correct sign.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Exact Geometric Computing

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.
- Error analysis is needed to come up with *root bounds*: A positive number r forms a root bound for a predicate p if $|p(x)| > r$ guarantees that the evaluation of p for x yields the correct sign.
- Main problems:
  - Arbitrary-precision arithmetic is needed.
  - Tight root bounds are difficult to obtain.
  - Constructors may cause the bit-length to grow tremendously.

Exact Geometric Computing

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.
- Error analysis is needed to come up with *root bounds*: A positive number r forms a root bound for a predicate p if $|p(x)| > r$ guarantees that the evaluation of p for x yields the correct sign.
- Main problems:
  - Arbitrary-precision arithmetic is needed.
  - Tight root bounds are difficult to obtain.
  - Constructors may cause the bit-length to grow tremendously.
- The use of floating-point filters may help to keep the increase in CPU time consumption a bit more moderate:
  - Work with floating-point arithmetic and check whether it gives the right answer.
  - Resort to extended-precision or exact arithmetic if the answer is incorrect.

Exact Geometric Computing

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.
- Error analysis is needed to come up with *root bounds*: A positive number r forms a root bound for a predicate p if $|p(x)| > r$ guarantees that the evaluation of p for x yields the correct sign.
- Main problems:
  - Arbitrary-precision arithmetic is needed.
  - Tight root bounds are difficult to obtain.
  - Constructors may cause the bit-length to grow tremendously.
- The use of floating-point filters may help to keep the increase in CPU time consumption a bit more moderate:
  - Work with floating-point arithmetic and check whether it gives the right answer.
  - Resort to extended-precision or exact arithmetic if the answer is incorrect.
  - Great example: Shewchuk's fine (and efficient!) CDT code "Triangle" [Shewchuk (1996)].

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Exact Geometric Computing

- Do we really need exact arithmetic for solving geometric problems?
- Not quite! Or, at least, not always! $\longrightarrow$ Exact geometric computing (EGC).
- Input is assumed to be exact.
- Basic idea: Make sure that the sign of every predicate is evaluated correctly.
- Error analysis is needed to come up with *root bounds*: A positive number r forms a root bound for a predicate p if $|p(x)| > r$ guarantees that the evaluation of p for x yields the correct sign.
- Main problems:
  - Arbitrary-precision arithmetic is needed.
  - Tight root bounds are difficult to obtain.
  - Constructors may cause the bit-length to grow tremendously.
- The use of floating-point filters may help to keep the increase in CPU time consumption a bit more moderate:
  - Work with floating-point arithmetic and check whether it gives the right answer.
  - Resort to extended-precision or exact arithmetic if the answer is incorrect.
  - Great example: Shewchuk's fine (and efficient!) CDT code "Triangle" [Shewchuk (1996)].
- Software libraries that provide support for EGC: CORE, CGAL, and Mörig's `RealAlgebraic` data type.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.
- Roughly, each coordinate x is replaced by $x + f(\varepsilon)$, where ε is unknown but very small and the perturbation function f is simple, e.g, a polynomial.
- Symbolic perturbation transforms the result of a numerical predicate into a polynomial in ε, whose sign is given by the sign of the first non-zero coefficient.
- For several important types of predicates perturbation functions can be chosen such that all degeneracies are resolved.

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.
- Roughly, each coordinate x is replaced by $x + f(\varepsilon)$, where ε is unknown but very small and the perturbation function f is simple, e.g, a polynomial.
- Symbolic perturbation transforms the result of a numerical predicate into a polynomial in ε, whose sign is given by the sign of the first non-zero coefficient.
- For several important types of predicates perturbation functions can be chosen such that all degeneracies are resolved.
- Main problems:
  - Exact arithmetic is required for evaluating the predicates.
  - Constructors are often disallowed.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.
- Roughly, each coordinate x is replaced by $x + f(\varepsilon)$, where ε is unknown but very small and the perturbation function f is simple, e.g, a polynomial.
- Symbolic perturbation transforms the result of a numerical predicate into a polynomial in ε, whose sign is given by the sign of the first non-zero coefficient.
- For several important types of predicates perturbation functions can be chosen such that all degeneracies are resolved.
- Main problems:
  - Exact arithmetic is required for evaluating the predicates.
  - Constructors are often disallowed.
  - Computational or combinatorial complexities may change substantially. E.g., the arrangement of n lines that intersect in one common point has $\Theta(n)$ complexity whereas an arrangement of the perturbed lines will have $\Theta(n^2)$ complexity.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.
- Roughly, each coordinate x is replaced by $x + f(\varepsilon)$, where ε is unknown but very small and the perturbation function f is simple, e.g, a polynomial.
- Symbolic perturbation transforms the result of a numerical predicate into a polynomial in ε, whose sign is given by the sign of the first non-zero coefficient.
- For several important types of predicates perturbation functions can be chosen such that all degeneracies are resolved.
- Main problems:
  - Exact arithmetic is required for evaluating the predicates.
  - Constructors are often disallowed.
  - Computational or combinatorial complexities may change substantially. E.g., the arrangement of n lines that intersect in one common point has $\Theta(n)$ complexity whereas an arrangement of the perturbed lines will have $\Theta(n^2)$ complexity.
  - The output computed does not correspond to the actual input.
  - Intentional degeneracies in the input are "resolved", too!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Symbolic Perturbation

- Aka: Simulation of simplicity [Edelsbrunner&Mücke (1990)].
- Basic idea: Perturb input such that no degeneracies occur.
- Roughly, each coordinate x is replaced by $x + f(\varepsilon)$, where ε is unknown but very small and the perturbation function f is simple, e.g, a polynomial.
- Symbolic perturbation transforms the result of a numerical predicate into a polynomial in ε, whose sign is given by the sign of the first non-zero coefficient.
- For several important types of predicates perturbation functions can be chosen such that all degeneracies are resolved.
- Main problems:
  - Exact arithmetic is required for evaluating the predicates.
  - Constructors are often disallowed.
  - Computational or combinatorial complexities may change substantially. E.g., the arrangement of n lines that intersect in one common point has $\Theta(n)$ complexity whereas an arrangement of the perturbed lines will have $\Theta(n^2)$ complexity.
  - The output computed does not correspond to the actual input.
  - Intentional degeneracies in the input are "resolved", too!
  - Similar to EGC codes, it is extremely difficult to interface a code based on SoS with a code based on conventional floating-point arithmetic.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Epsilon Thresholds

- Topological decisions are based on the results of floating-point computations, which are prone to round-off errors.
- Threshold-based comparison:

$$(a =_\varepsilon b) \quad :\Longleftrightarrow \quad (|a - b| \leq \varepsilon),$$

for some positive value of ε.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Epsilon Thresholds

- Topological decisions are based on the results of floating-point computations, which are prone to round-off errors.
- Threshold-based comparison:

$$(a =_\varepsilon b) \quad :\Longleftrightarrow \quad (|a - b| \leq \varepsilon),$$

for some positive value of ε.

Caveat!

Threshold-based comparisons are not transitive: $a =_\varepsilon b$ and $b =_\varepsilon c$ need not imply $a =_\varepsilon c$.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Epsilon Thresholds

- Topological decisions are based on the results of floating-point computations, which are prone to round-off errors.
- Threshold-based comparison:

$$(a =_\varepsilon b) \quad :\Longleftrightarrow \quad (|a - b| \leq \varepsilon),$$

for some positive value of ε.

Caveat!

Threshold-based comparisons are not transitive: $a =_\varepsilon b$ and $b =_\varepsilon c$ need not imply $a =_\varepsilon c$.

- This gap between theory and practice once again has important and severe consequences for the actual coding practice when implementing geometric algorithms:

Epsilon Thresholds

- Topological decisions are based on the results of floating-point computations, which are prone to round-off errors.
- Threshold-based comparison:

$$(a =_\varepsilon b) \quad :\Longleftrightarrow \quad (|a - b| \leq \varepsilon),$$

for some positive value of ε.

Caveat!

Threshold-based comparisons are not transitive: $a =_\varepsilon b$ and $b =_\varepsilon c$ need not imply $a =_\varepsilon c$.

- This gap between theory and practice once again has important and severe consequences for the actual coding practice when implementing geometric algorithms:

No guarantee for success!

1. The correctness proof of a mathematical algorithm does not extend to the program, and the program can fail on seemingly appropriate input data.

Epsilon Thresholds

- Topological decisions are based on the results of floating-point computations, which are prone to round-off errors.
- Threshold-based comparison:

$$(a =_\varepsilon b) \quad :\Longleftrightarrow \quad (|a - b| \le \varepsilon),$$

  for some positive value of ε.

Caveat!

Threshold-based comparisons are not transitive: $a =_\varepsilon b$ and $b =_\varepsilon c$ need not imply $a =_\varepsilon c$.

- This gap between theory and practice once again has important and severe consequences for the actual coding practice when implementing geometric algorithms:

No guarantee for success!

1. The correctness proof of a mathematical algorithm does not extend to the program, and the program can fail on seemingly appropriate input data.
2. Local consistency need not imply global consistency.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned}\det(p,q,r) &= \det(q,r,p) = \det(r,p,q)\\ &= -\det(q,p,r) = -\det(p,r,q) = -\det(r,q,p).\end{aligned}$$

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned}\det(p, q, r) &= \det(q, r, p) = \det(r, p, q) \\ &= -\det(q, p, r) = -\det(p, r, q) = -\det(r, q, p).\end{aligned}$$

- Do not resort to multiple precision thresholds! At most two thresholds: One to avoid divisions by zero, and another threshold to catch "nearly zero" numbers.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned}\det(p, q, r) &= \det(q, r, p) = \det(r, p, q) \\ &= -\det(q, p, r) = -\det(p, r, q) = -\det(r, q, p).\end{aligned}$$

- Do not resort to multiple precision thresholds! At most two thresholds: One to avoid divisions by zero, and another threshold to catch "nearly zero" numbers.
- Epsilon-based comparisons need to be relative to the absolute values of the numbers to be compared, or the input has to be scaled (by performing shifts!) to fit into the unit square/cube prior to the actual computation.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned} \det(p,q,r) &= \det(q,r,p) = \det(r,p,q) \\ &= -\det(q,p,r) = -\det(p,r,q) = -\det(r,q,p). \end{aligned}$$

- Do not resort to multiple precision thresholds! At most two thresholds: One to avoid divisions by zero, and another threshold to catch "nearly zero" numbers.
- Epsilon-based comparisons need to be relative to the absolute values of the numbers to be compared, or the input has to be scaled (by performing shifts!) to fit into the unit square/cube prior to the actual computation.
- Use iterations as back-up for analytical solutions to equations. If at all possible, use methods that bracket the solution sought!

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned} \det(p,q,r) &= \det(q,r,p) = \det(r,p,q) \\ &= -\det(q,p,r) = -\det(p,r,q) = -\det(r,q,p). \end{aligned}$$

- Do not resort to multiple precision thresholds! At most two thresholds: One to avoid divisions by zero, and another threshold to catch "nearly zero" numbers.
- Epsilon-based comparisons need to be relative to the absolute values of the numbers to be compared, or the input has to be scaled (by performing shifts!) to fit into the unit square/cube prior to the actual computation.
- Use iterations as back-up for analytical solutions to equations. If at all possible, use methods that bracket the solution sought!
- Algebraically equivalent terms need not be equally reliable on a floating-point arithmetic.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Standard Tricks for Achieving Reliable Floating-Point Code

- Try to perform all numerical computations relative to the original input data.
- All floating-point computations need to be consistent! In particular, make sure that different calls of the same function with the "same" input will yield exactly the same output. E.g., when computing 3×3 determinants, we need

$$\begin{aligned} \det(p,q,r) &= \det(q,r,p) = \det(r,p,q) \\ &= -\det(q,p,r) = -\det(p,r,q) = -\det(r,q,p). \end{aligned}$$

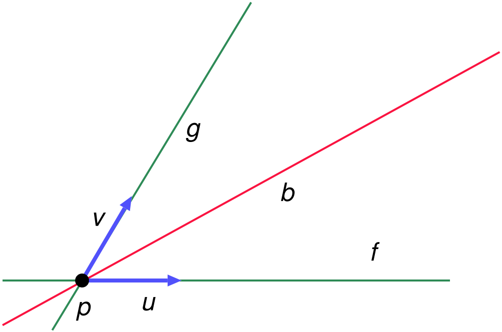
- Do not resort to multiple precision thresholds! At most two thresholds: One to avoid divisions by zero, and another threshold to catch "nearly zero" numbers.
- Epsilon-based comparisons need to be relative to the absolute values of the numbers to be compared, or the input has to be scaled (by performing shifts!) to fit into the unit square/cube prior to the actual computation.
- Use iterations as back-up for analytical solutions to equations. If at all possible, use methods that bracket the solution sought!
- Algebraically equivalent terms need not be equally reliable on a floating-point arithmetic. E.g., consider the quadratic equation $x^2 + px + q = 0$ and compare

$$\left\{ \begin{array}{l} x_1 := -\frac{1}{2}(p + \sqrt{p^2 - 4q}) \\ x_2 := -\frac{1}{2}(p - \sqrt{p^2 - 4q}) \end{array} \right\} \quad \text{to} \quad \left\{ \begin{array}{l} x_1 := -\frac{1}{2}(p + \operatorname{sign}(p)\sqrt{p^2 - 4q}) \\ x_2 := q/x_1 \end{array} \right\}$$

if $|q|$ is small.

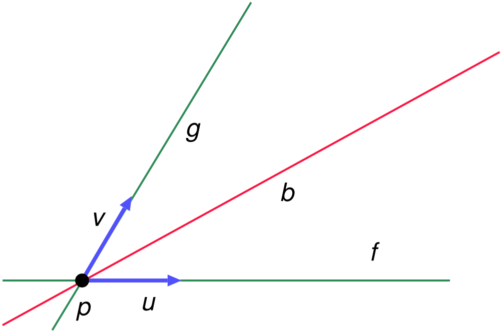
Case Study: Computing Voronoi Bisectors

- Algorithm for computing a bisector b between two lines f and g which are not parallel:
  - Compute their point of intersection: p.
  - Compute unit direction vectors u and v of f, g.
  - Then a parametrization of b is given by $p + t \cdot (u + v)$.



Case Study: Computing Voronoi Bisectors

- Algorithm for computing a bisector b between two lines f and g which are not parallel:
  - Compute their point of intersection: p.
  - Compute unit direction vectors u and v of f, g.
  - Then a parametrization of b is given by $p + t \cdot (u + v)$.
- This "natural" approach to computing b becomes completely infeasible if f and g are nearly parallel. (In that case the computation of p will become *very* unreliable!)

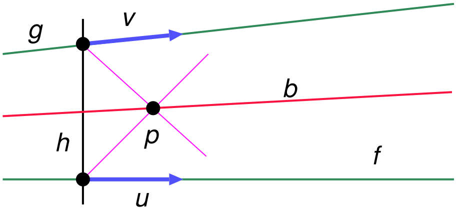


- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

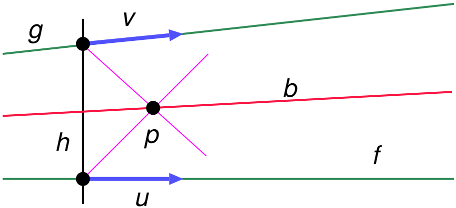
Case Study: Computing Voronoi Bisectors

- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:
  - Compute a line h that is normal on f.

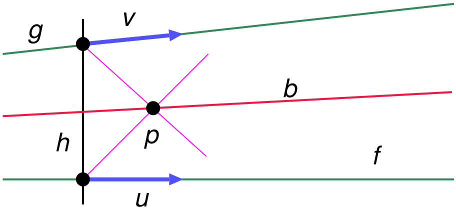


Case Study: Computing Voronoi Bisectors

- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:
  - Compute a line h that is normal on f.
  - Compute the bisector b_1 between h and f, and the bisector b_2 between h and g.

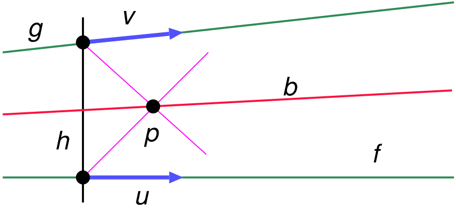


- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:
  - Compute a line h that is normal on f.
  - Compute the bisector b_1 between h and f, and the bisector b_2 between h and g.
  - Compute their point of intersection: p.

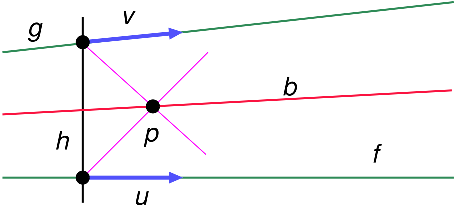


Case Study: Computing Voronoi Bisectors

- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:
  - Compute a line h that is normal on f.
  - Compute the bisector b_1 between h and f, and the bisector b_2 between h and g.
  - Compute their point of intersection: p.
  - Compute unit direction vectors u and v of f, g.
  - Then a parametrization of b is given by $p + t \cdot (u + v)$.

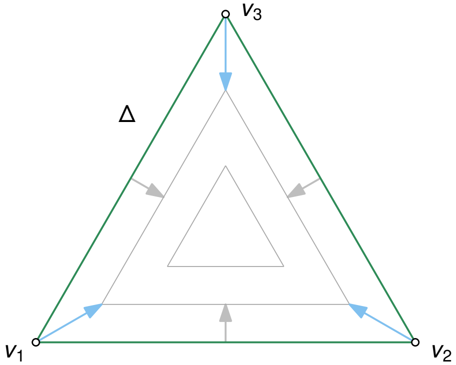


- Algorithm for computing a bisector b between two lines f and g which are roughly parallel:
  - Compute a line h that is normal on f.
  - Compute the bisector b_1 between h and f, and the bisector b_2 between h and g.
  - Compute their point of intersection: p.
  - Compute unit direction vectors u and v of f, g.
  - Then a parametrization of b is given by $p + t \cdot (u + v)$.



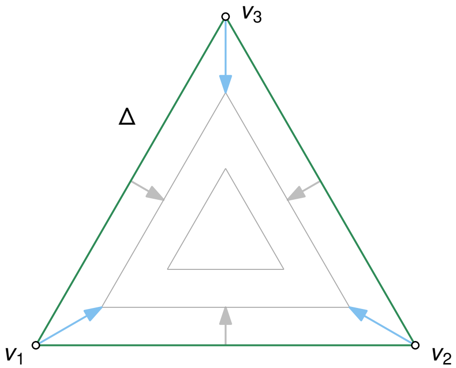
- Note: All intersections are defined by pairs of lines that are roughly perpendicular.

- Suppose that all three vertices v_1, v_2, v_3 of a triangle Δ move along straight-line paths (modeled as functions of time), and that we are interested in knowing when the triangle collapses. (That is, when it has zero area.)



Case Study: Collapse Time of Kinetic Triangle

- Suppose that all three vertices v_1, v_2, v_3 of a triangle Δ move along straight-line paths (modeled as functions of time), and that we are interested in knowing when the triangle collapses. (That is, when it has zero area.)



- How can we determine the time(s) of collapse?

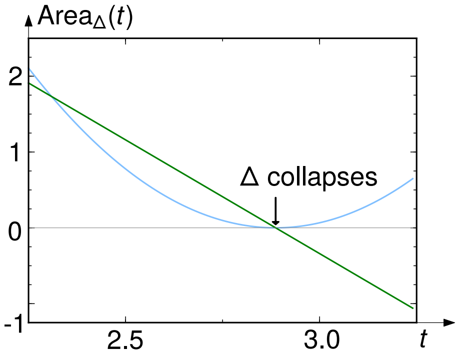
- Which option is better?
  1. Check the signed area of the triangle, e.g., as obtained by means of determinant computations.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Which option is better?
  1. Check the signed area of the triangle, e.g., as obtained by means of determinant computations.
  2. Check the signed distance of a kinetic vertex to its opposite edge.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- Which option is better?
  1. Check the signed area of the triangle, e.g., as obtained by means of determinant computations.
  2. Check the signed distance of a kinetic vertex to its opposite edge.



Topology-Oriented Computation

- First used by Sugihara et alii [1992, 2000].
- Basic idea:
  1. Define topological criteria that the output has to meet.

- First used by Sugihara et alii [1992, 2000].
- Basic idea:
  1. Define topological criteria that the output has to meet.
  2. Use floating-point computations to choose among different geometric set-ups if two or more set-ups fulfill all topological criteria.

Topology-Oriented Computation

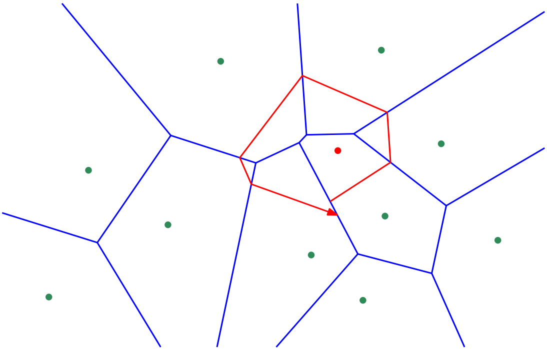
- First used by Sugihara et alii [1992, 2000].
- Basic idea:
  1. Define topological criteria that the output has to meet.
  2. Use floating-point computations to choose among different geometric set-ups if two or more set-ups fulfill all topological criteria.
- Main problem: For many problems it is difficult to formulate meaningful topological criteria.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Topology-Oriented Computation

- First used by Sugihara et alii [1992, 2000].
- Basic idea:
  1. Define topological criteria that the output has to meet.
  2. Use floating-point computations to choose among different geometric set-ups if two or more set-ups fulfill all topological criteria.
- Main problem: For many problems it is difficult to formulate meaningful topological criteria.
- Sample application: When inserting a new site into a Voronoi diagram (during an incremental construction), the portion of the old Voronoi diagram to be deleted forms a tree.

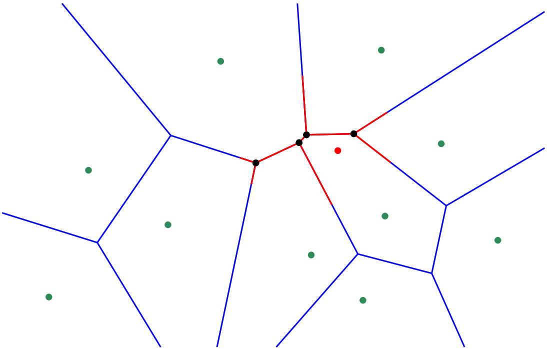
- The portion of the Voronoi diagram to be deleted forms a tree.



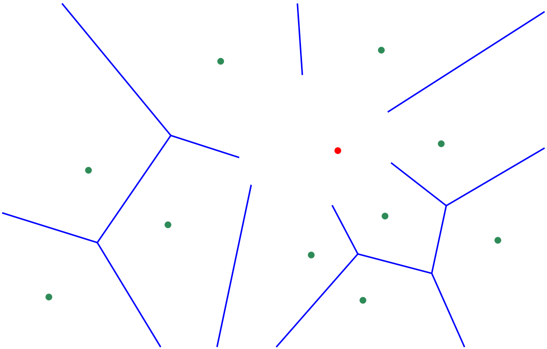
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Topology-Oriented Computation

- The portion of the Voronoi diagram to be deleted forms a tree.
- We start at the closest Voronoi node and scan the Voronoi diagram for nodes to be deleted, making sure that no cycle occurs.

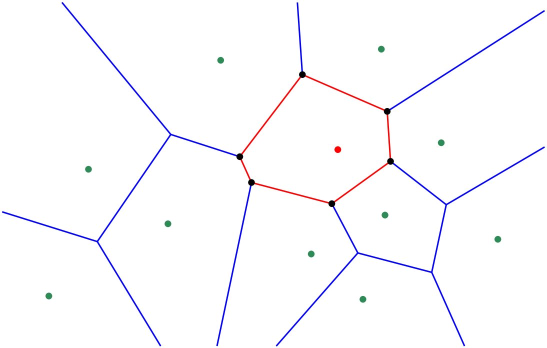


- The portion of the Voronoi diagram to be deleted forms a tree.
- We start at the closest Voronoi node and scan the Voronoi diagram for nodes to be deleted, making sure that no cycle occurs.
- All Voronoi nodes found by this scan are deleted, and their incident bisectors are shortened appropriately.



Topology-Oriented Computation

- The portion of the Voronoi diagram to be deleted forms a tree.
- We start at the closest Voronoi node and scan the Voronoi diagram for nodes to be deleted, making sure that no cycle occurs.
- All Voronoi nodes found by this scan are deleted, and their incident bisectors are shortened appropriately.
- All that remains to be done is to compute the new Voronoi nodes, which form the corners of the new Voronoi polygon, and to link them in cyclic order.



Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \leq \varepsilon$?", for some positive value of ε.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \leq \varepsilon$?", for some positive value of ε.
- Just what is an appropriate value for ε?

Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \leq \varepsilon$?", for some positive value of ε.
- Just what is an appropriate value for ε?
- Asking the user of a geometric code to choose an appropriate precision threshold comes close to asking for witch-craft.
- To make the situation worse, experience tells me that one fixed precision threshold will rarely suffice to handle all sorts of different input data.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \leq \varepsilon$?", for some positive value of ε.
- Just what is an appropriate value for ε?
- Asking the user of a geometric code to choose an appropriate precision threshold comes close to asking for witch-craft.
- To make the situation worse, experience tells me that one fixed precision threshold will rarely suffice to handle all sorts of different input data.
- Alternate approach:
  - A natural lower bound ε_{min} for a suitable threshold is given by the machine precision of the machine used.

Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \leq \varepsilon$?", for some positive value of ε.
- Just what is an appropriate value for ε?
- Asking the user of a geometric code to choose an appropriate precision threshold comes close to asking for witch-craft.
- To make the situation worse, experience tells me that one fixed precision threshold will rarely suffice to handle all sorts of different input data.
- Alternate approach:
  - A natural lower bound ε_{min} for a suitable threshold is given by the machine precision of the machine used.
  - The user specifies the maximum distance that two points (out of the unit square) may be apart in order to allow the code to treat them as one point. This yields an upper bound ε_{max}.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Relaxation of Epsilon Thresholds

- Common practice: Use epsilon thresholds for comparisons with respect to zero.
- The test "$x = 0$?" is replaced by the test "$|x| \le \varepsilon$?", for some positive value of ε.
- Just what is an appropriate value for ε?
- Asking the user of a geometric code to choose an appropriate precision threshold comes close to asking for witch-craft.
- To make the situation worse, experience tells me that one fixed precision threshold will rarely suffice to handle all sorts of different input data.
- Alternate approach:
  - A natural lower bound ε_{min} for a suitable threshold is given by the machine precision of the machine used.
  - The user specifies the maximum distance that two points (out of the unit square) may be apart in order to allow the code to treat them as one point. This yields an upper bound ε_{max}.
  - The code varies ε at its own discretion, always attempting to succeed with the smallest ε possible, i.e., with maximal precision.

Algorithm *Typical Computational Unit*

$\varepsilon = \varepsilon_{min}$; (∗ set epsilon to maximum precision ∗)
repeat
  $x = \text{ComputeData}(\varepsilon)$; (∗ compute some data ∗)
  success = CheckConditions(x, ε); (∗ check topological/numerical conditions ∗)
  if (not success) **then**
    $\varepsilon = 10 \cdot \varepsilon$; (∗ relaxation of epsilon threshold ∗)
    reset data structures appropriately;
until (success OR $\varepsilon > \varepsilon_{max}$);
$\varepsilon = \varepsilon_{min}$; (∗ make sure to reset epsilon ∗)
if (not success) **then**
  illegal = CheckInput(); (∗ check locally for soundness of input ∗)
  if (illegal) **then**
    clean data locally; (∗ fix the problem in the input data ∗)
    restart computation from scratch;
  else
    $x = \text{DesperateMode}()$; (∗ time to hope for the best ∗)

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.
  - Do not use any operation which is not defined for all floating-point numbers.

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace "correct" or "optimum" by "best possible".
  - Do not use any operation which is not defined for all floating-point numbers.
  - Any violation of a topological condition is "cured" by forcing its validity. E.g., a cycle of Voronoi nodes can be broken by inserting a new dummy node on one of the edges of the cycle.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.
  - Do not use any operation which is not defined for all floating-point numbers.
  - Any violation of a topological condition is “cured” by forcing its validity. E.g., a cycle of Voronoi nodes can be broken by inserting a new dummy node on one of the edges of the cycle.
  - If numerical data does not fulfill all numerical conditions then accept the data whichever meets most of them.

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.
  - Do not use any operation which is not defined for all floating-point numbers.
  - Any violation of a topological condition is “cured” by forcing its validity. E.g., a cycle of Voronoi nodes can be broken by inserting a new dummy node on one of the edges of the cycle.
  - If numerical data does not fulfill all numerical conditions then accept the data whichever meets most of them.
- Net result: A code that resorts to desperate mode will never get stuck, crash or loop. (Well, at least in theory . . .)

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.
  - Do not use any operation which is not defined for all floating-point numbers.
  - Any violation of a topological condition is “cured” by forcing its validity. E.g., a cycle of Voronoi nodes can be broken by inserting a new dummy node on one of the edges of the cycle.
  - If numerical data does not fulfill all numerical conditions then accept the data whichever meets most of them.
- Net result: A code that resorts to desperate mode will never get stuck, crash or loop. (Well, at least in theory . . .)
- Whether or not the output still is of practical use despite of desperate mode depends on the application and the type of problem that caused desperate mode.

Desperate Mode

- What shall we do if computing x would require an invalid operation, e.g., a division by zero or taking the square root of a negative number?
- After all, x may be needed for subsequent computations! E.g., if a new Voronoi node is to be inserted into the Voronoi diagram then the code may subsequently need to access the coordinates of this node.
- Desperate mode: Replace “correct” or “optimum” by “best possible”.
  - Do not use any operation which is not defined for all floating-point numbers.
  - Any violation of a topological condition is “cured” by forcing its validity. E.g., a cycle of Voronoi nodes can be broken by inserting a new dummy node on one of the edges of the cycle.
  - If numerical data does not fulfill all numerical conditions then accept the data whichever meets most of them.
- Net result: A code that resorts to desperate mode will never get stuck, crash or loop. (Well, at least in theory . . .)
- Whether or not the output still is of practical use despite of desperate mode depends on the application and the type of problem that caused desperate mode.
- Personal experience: Desperate mode works remarkably well for “incremental” algorithms!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Desperate Mode and Epsilon Relaxation: Caveats

- The relaxation of epsilon thresholds does not come for free: Repeatedly computing some data will increase the CPU time consumption.

Desperate Mode and Epsilon Relaxation: Caveats

- The relaxation of epsilon thresholds does not come for free: Repeatedly computing some data will increase the CPU time consumption.
- Always attempt to perform a computation as it would be performed normally, irrelevant of whether or not desperate mode was used once before.

Desperate Mode and Epsilon Relaxation: Caveats

- The relaxation of epsilon thresholds does not come for free: Repeatedly computing some data will increase the CPU time consumption.
- Always attempt to perform a computation as it would be performed normally, irrelevant of whether or not desperate mode was used once before.
- Keep track of how often your code resorts to an actual relaxation: Frequent use of epsilon relaxation is a hint that the numerical reliability of your code is less than ideal.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- The relaxation of epsilon thresholds does not come for free: Repeatedly computing some data will increase the CPU time consumption.
- Always attempt to perform a computation as it would be performed normally, irrelevant of whether or not desperate mode was used once before.
- Keep track of how often your code resorts to an actual relaxation: Frequent use of epsilon relaxation is a hint that the numerical reliability of your code is less than ideal.

Warning

The availability of such a multi-phase recovery process can easily conceal genuine algorithmic flaws or bugs!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- The relaxation of epsilon thresholds does not come for free: Repeatedly computing some data will increase the CPU time consumption.
- Always attempt to perform a computation as it would be performed normally, irrelevant of whether or not desperate mode was used once before.
- Keep track of how often your code resorts to an actual relaxation: Frequent use of epsilon relaxation is a hint that the numerical reliability of your code is less than ideal.

Warning

The availability of such a multi-phase recovery process can easily conceal genuine algorithmic flaws or bugs!

Important advice

Always test your code with desperate mode being disabled! Robustness without desperate mode is a must for *all* tests on *your* test data!

Adding an MPFR Backend

- GNU's MPFR library (`www.mpfr.org`) is a C library for multiple-precision floating-point computations.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Adding an MPFR Backend

- GNU's MPFR library (www.mpfr.org) is a C library for multiple-precision floating-point computations.
- Canonical adaptions:
  - Precision threshold ε needs to depend on MPFR precision.
  - [Held&Huber (2013)] use a heuristic formula:

$$\varepsilon := \varepsilon_{\text{fp}} \cdot 2^{-100\cdot(\sqrt{\text{prec}/53}-1)},$$

  where ε_{fp} is the machine precision.

Adding an MPFR Backend

- GNU's MPFR library (`www.mpfr.org`) is a C library for multiple-precision floating-point computations.
- Canonical adaptions:
  - Precision threshold ε needs to depend on MPFR precision.
  - [Held&Huber (2013)] use a heuristic formula:

$$\varepsilon := \varepsilon_{\text{fp}} \cdot 2^{-100 \cdot \left(\sqrt{\text{prec}/53} - 1\right)},$$

    where ε_{fp} is the machine precision.
- Subtle problem encountered: `mpfr_set_default_prec` does not change existing variables.
  - Global variables are not adjusted.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

8 Robustness Problems and Real-World Issues

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Real-World Data

- Real-world data often means no quality at all:
  - raster-to-vector conversions,
  - data cleaned up manually or “visually”,
  - data preprocessed by some dubious program of unknown origin,
  - data comes from “an important customer” or from “God knows where”.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Real-World Data

- Real-world data often means no quality at all:
  - raster-to-vector conversions,
  - data cleaned up manually or “visually”,
  - data preprocessed by some dubious program of unknown origin,
  - data comes from “an important customer” or from “God knows where”.
- As a consequence:
  - all sorts of degeneracies,
  - self-intersections,
  - tiny gaps in supposedly closed contours.

Real-World Data

- Real-world data often means no quality at all:
  - raster-to-vector conversions,
  - data cleaned up manually or “visually”,
  - data preprocessed by some dubious program of unknown origin,
  - data comes from “an important customer” or from “God knows where”.
- As a consequence:
  - all sorts of degeneracies,
  - self-intersections,
  - tiny gaps in supposedly closed contours.

Advice

Be prepared for troubles — general position must not be assumed!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Real-World Data

- Real-world data often means no quality at all:
  - raster-to-vector conversions,
  - data cleaned up manually or “visually”,
  - data preprocessed by some dubious program of unknown origin,
  - data comes from “an important customer” or from “God knows where”.
- As a consequence:
  - all sorts of degeneracies,
  - self-intersections,
  - tiny gaps in supposedly closed contours.

Advice

Be prepared for troubles — general position must not be assumed!

If the implementation ends up in an invalid algorithmic state . . .

Is it due to

- a bug in the implementation,
- a genuine numerical problem, or
- invalid input data?

Data size:

- Data sizes vary substantially from a few hundred segments/arcs in a machining application to a few million segments in a GIS application, or even tens of millions of segments/arcs in the PCB business if arcs are approximated by segments.

Data size:

- Data sizes vary substantially from a few hundred segments/arcs in a machining application to a few million segments in a GIS application, or even tens of millions of segments/arcs in the PCB business if arcs are approximated by segments.

Efficiency requirements:

- Efficiency requirements vary substantially from real-time map generation on a smart phone to minutes of CPU time allowed on some high-end machine.
- In general, linear space complexity and a close-to-linear time complexity is expected.

Industrial Requirements

Data size:

- Data sizes vary substantially from a few hundred segments/arcs in a machining application to a few million segments in a GIS application, or even tens of millions of segments/arcs in the PCB business if arcs are approximated by segments.

Efficiency requirements:

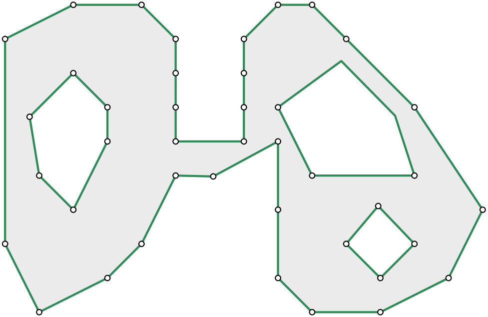
- Efficiency requirements vary substantially from real-time map generation on a smart phone to minutes of CPU time allowed on some high-end machine.
- In general, linear space complexity and a close-to-linear time complexity is expected.

Parallelization requirements:

- Exactly one inquiry concerning GPU-based codes so far.
- Only moderate interest in multi-core computing and multi-threaded implementations.

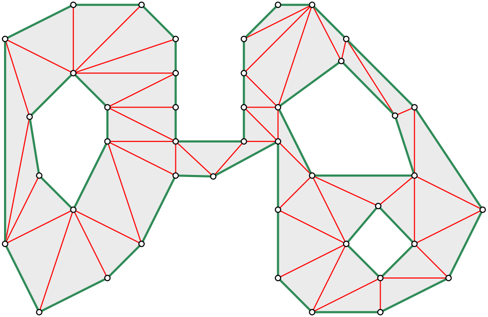
Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,



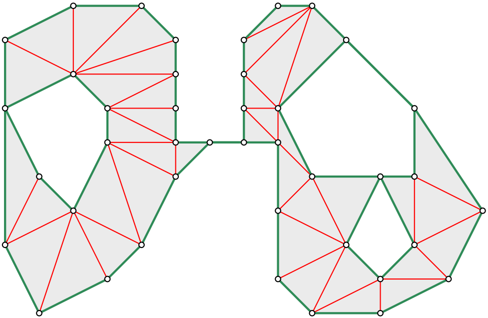
Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.



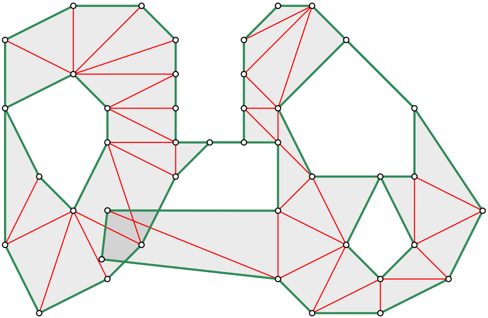
Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,

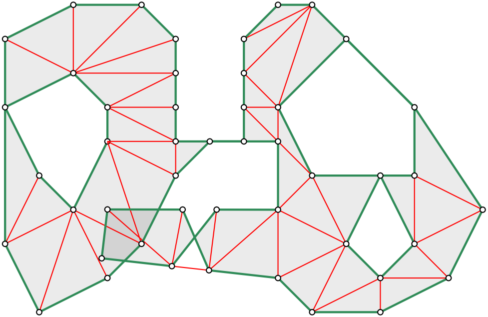


Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,

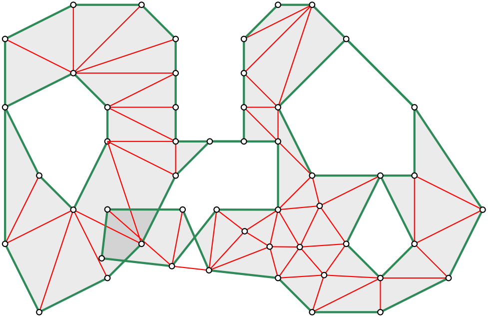


- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,



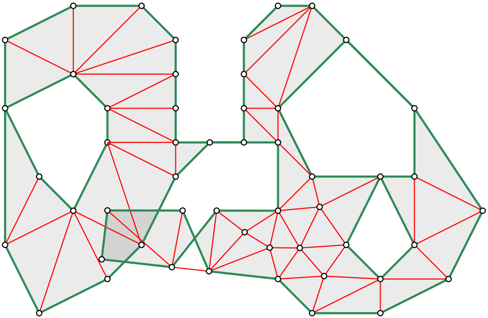
Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,
  - Steiner points.



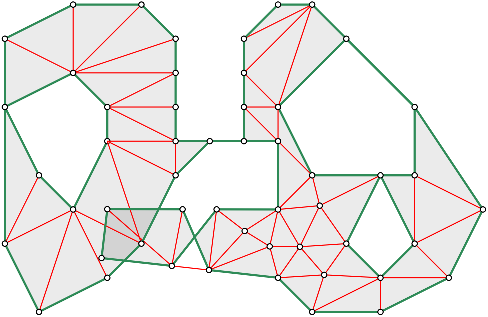
Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,
  - Steiner points.
- No Delaunay triangulation, but heuristics to generate "decent" triangles.
- Runs in close-to-linear time in practice.



Software for Triangulating Polygons: FIST

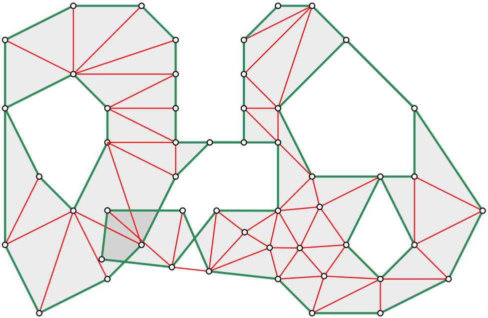
- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,
  - Steiner points.
- No Delaunay triangulation, but heuristics to generate "decent" triangles.
- Runs in close-to-linear time in practice.



- ANSI C code based on IEEE 754 floating-point arithmetic, with careful engineering to ensure reliability. Thread-safe.

Software for Triangulating Polygons: FIST

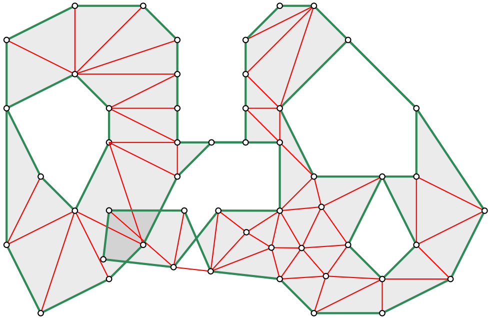
- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,
  - Steiner points.
- No Delaunay triangulation, but heuristics to generate “decent” triangles.
- Runs in close-to-linear time in practice.



- ANSI C code based on IEEE 754 floating-point arithmetic, with careful engineering to ensure reliability. Thread-safe.
- [Eder&Held&Palfrader (2018)]: Heuristics for coarse-grain parallelization.

Software for Triangulating Polygons: FIST

- [Held (2001)]: FIST triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation.
- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input,
  - Steiner points.
- No Delaunay triangulation, but heuristics to generate “decent” triangles.
- Runs in close-to-linear time in practice.



- ANSI C code based on IEEE 754 floating-point arithmetic, with careful engineering to ensure reliability. Thread-safe.
- [Eder&Held&Palfrader (2018)]: Heuristics for coarse-grain parallelization.
- Typical applications in industry: Triangulation of (very) large GIS datasets, triangulation of “planar” faces of 3D models.

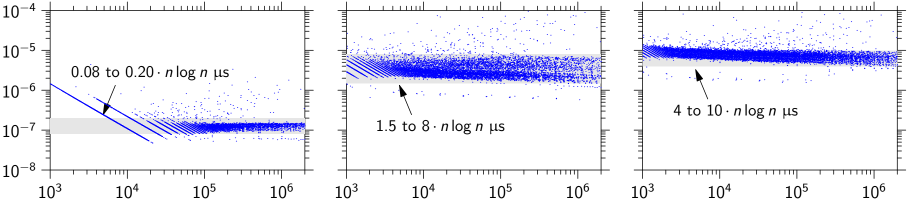
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Experimental Results for Triangulations

- 21 175 polygons with and without holes.
- Six arithmetic configurations:
  - fistFp, fistShew, fistCore, fistMp{53, 212, 1000}

Experimental Results for Triangulations

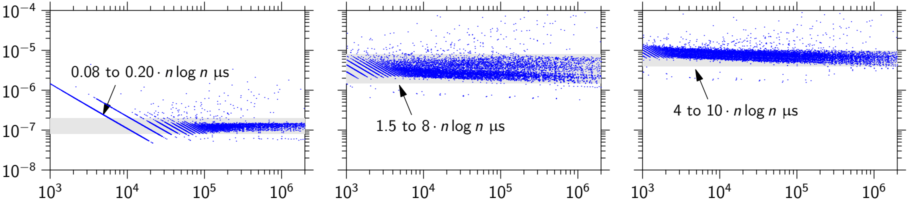
- 21 175 polygons with and without holes.
- Six arithmetic configurations:
  - fistFp, fistShew, fistCore, fistMp{53, 212, 1000}



Runtime per seconds divided by $n \log n$, for fistFp, fistMp212, fistCore.

Experimental Results for Triangulations

- 21 175 polygons with and without holes.
- Six arithmetic configurations:
  - fistFp, fistShew, fistCore, fistMp{53, 212, 1000}
- Conclusion:
  - Shewchuck's predicates have negligible impact on speed.
  - fistMp* about 24 times slower than fistFp.
  - fistCore about 50 times slower than fistFp.



Runtime per seconds divided by $n \log n$, for fistFp, fistMp212, fistCore.

Held&Mann (2011)

Correctness of FIST triangulations with floating-point arithmetic?

- Verification code:
  - Bentley-Ottmann algorithm, implemented with exact `mpq_t` from GMP.

Experimental Results for Triangulations

Held&Mann (2011)

Correctness of FIST triangulations with floating-point arithmetic?

- Verification code:
  - Bentley-Ottmann algorithm, implemented with exact `mpq_t` from GMP.
- **Scenario 1**:
  - We interprete `0.1` in input files as $\frac{1}{10}$.
  - 4.9% of all results faulty, uniformly across all non-CORE configurations.
    - Including fistShew.
  - But: Error only visible at huge zoom factors.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Held&Mann (2011)

Correctness of FIST triangulations with floating-point arithmetic?

- Verification code:
  - Bentley-Ottmann algorithm, implemented with exact `mpq_t` from GMP.
- **Scenario 1**:
  - We interprete `0.1` in input files as $\frac{1}{10}$.
  - 4.9% of all results faulty, uniformly across all non-CORE configurations.
    - Including fistShew.
  - But: Error only visible at huge zoom factors.
- **Scenario 2**:
  - We take `0.1` as closest floating-point number using `atof()`.
  - No errors found!

Held&Mann (2011)

Correctness of FIST triangulations with floating-point arithmetic?

- Verification code:
  - Bentley-Ottmann algorithm, implemented with exact `mpq_t` from GMP.
- **Scenario 1**:
  - We interprete `0.1` in input files as $\frac{1}{10}$.
  - 4.9% of all results faulty, uniformly across all non-CORE configurations.
    - Including fistShew.
  - But: Error only visible at huge zoom factors.
- **Scenario 2**:
  - We take `0.1` as closest floating-point number using `atof()`.
  - No errors found!

Conclusion

Non-exactness need not be a practical issue in pure floating-point applications.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Software for Computing Voronoi Diagrams: CGAL

- Segment-Delaunay-Graph implemented within CGAL by [Karavelas (2004)].
- Input:
  - Points and straight-line segments;
  - Input sites may intersect arbitrarily;
  - No support for circular arcs.

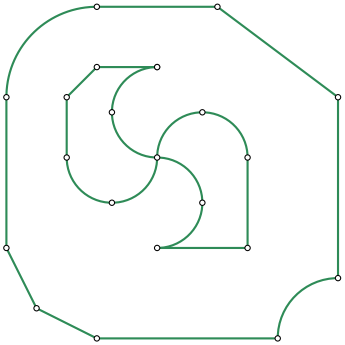
Software for Computing Voronoi Diagrams: CGAL

- Segment-Delaunay-Graph implemented within CGAL by [Karavelas (2004)].
- Input:
  - Points and straight-line segments;
  - Input sites may intersect arbitrarily;
  - No support for circular arcs.
- Incremental construction.
- Complexity:
  - $O((n+m)\log^2 n)$ expected time, where n is the number of sites and m is the number of intersections.
  - $O((n+m)\log n)$ in practice.
- CPU-time consumption is claimed to be mostly insensitive to the number type used.

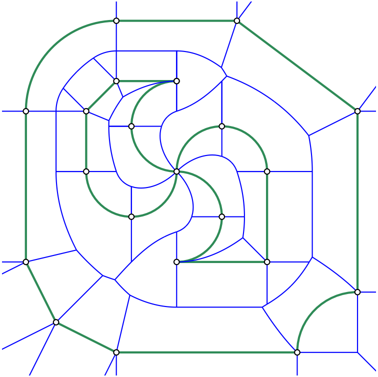
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Software for Computing Voronoi Diagrams: VRONI and ARCVRONI

- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input "sites"),

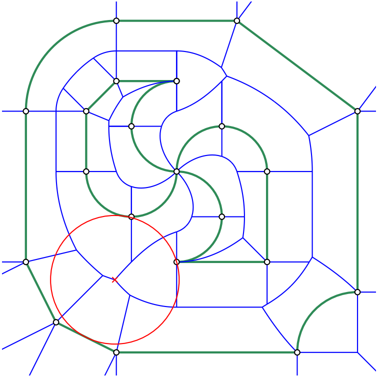


- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input “sites”),
  - based on randomized incremental construction and a topology-oriented approach.



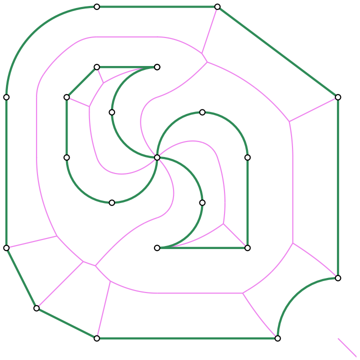
Software for Computing Voronoi Diagrams: VRONI and ARCVRONI

- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input “sites”),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,

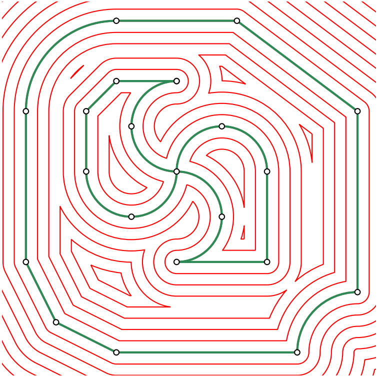


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

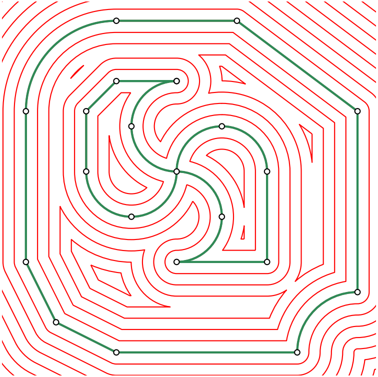
- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input “sites”),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,
  - medial axis,



- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input "sites"),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,
  - medial axis, and
  - offset curves.

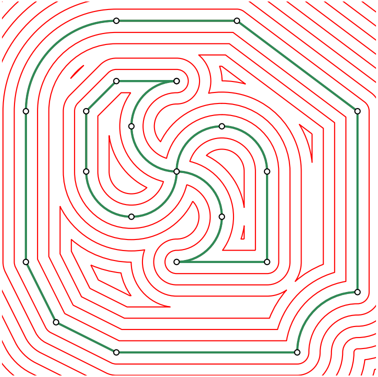


- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input "sites"),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,
  - medial axis, and
  - offset curves.
- Complexity:
  - $O(n)$ space complexity, where n is the number of sites.
  - $O(n \log n)$ expected time.



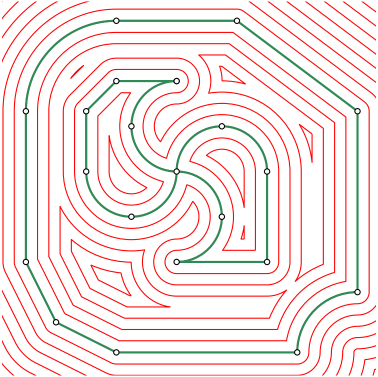
Software for Computing Voronoi Diagrams: VRONI and ARCVRONI

- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input “sites”),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,
  - medial axis, and
  - offset curves.
- Complexity:
  - $O(n)$ space complexity, where n is the number of sites.
  - $O(n \log n)$ expected time.
- Based on standard IEEE 754 floating-point arithmetic, with careful engineering, epsilon relaxation and desperate mode.



Software for Computing Voronoi Diagrams: VRONI and ARCVRONI

- [Held (2001), Held&Huber (2009)]: VRONI/ARCVRONI computes Voronoi diagrams
  - of points, straight-line segments and circular arcs (input “sites”),
  - based on randomized incremental construction and a topology-oriented approach.
- Also computes
  - maximum-inscribed circle,
  - medial axis, and
  - offset curves.
- Complexity:
  - $O(n)$ space complexity, where n is the number of sites.
  - $O(n \log n)$ expected time.
- Based on standard IEEE 754 floating-point arithmetic, with careful engineering, epsilon relaxation and desperate mode.



- Typical applications in industry: generation of tool paths (e.g., for machining or sintering), generation of buffers in GIS applications.

Codes tested:

- CGAL 3.8 (cgvdEx, cgvdFp):
  - `CORE::Expr` as predicate kernel.
  - `Segment_Delaunay_graph_filtered_traits_2` template parameter to the underlying segment Delaunay graph class.
  - Graphics disabled, input stream-lined, own timing routine added.
  - Compiled with `g++ -O2`.
- VRONI 6.0 (vroniFp, vroniMp$\{53, 212, 1000\}$):
  - Also compiled with `g++ -O2`.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Codes tested:

- CGAL 3.8 (cgvdEx, cgvdFp):
  - `CORE::Expr` as predicate kernel.
  - `Segment_Delaunay_graph_filtered_traits_2` template parameter to the underlying segment Delaunay graph class.
  - Graphics disabled, input stream-lined, own timing routine added.
  - Compiled with `g++ -O2`.
- VRONI 6.0 (vroniFp, vroniMp$\{53, 212, 1000\}$):
  - Also compiled with `g++ -O2`.

Test platform:

- 3.33GHz i7 CPU X 980; 24GB RAM; 64bit Ubuntu.
- All timings given in CPU microseconds.

Experimental Results for Voronoi Diagrams

Codes tested:

- CGAL 3.8 (cgvdEx, cgvdFp):
  - `CORE::Expr` as predicate kernel.
  - `Segment_Delaunay_graph_filtered_traits_2` template parameter to the underlying segment Delaunay graph class.
  - Graphics disabled, input stream-lined, own timing routine added.
  - Compiled with `g++ -O2`.
- VRONI 6.0 (vroniFp, vroniMp$\{53, 212, 1000\}$):
  - Also compiled with `g++ -O2`.

Test platform:

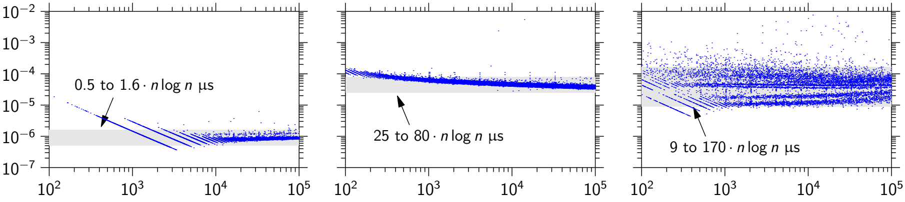
- 3.33GHz i7 CPU X 980; 24GB RAM; 64bit Ubuntu.
- All timings given in CPU microseconds.

Test data:

- Synthetic and real-world data; circular arcs approximated by polygonal chains.
- 18 787 data sets tested, with $200 \leq \#(\text{segs}) \leq 100\,000$.

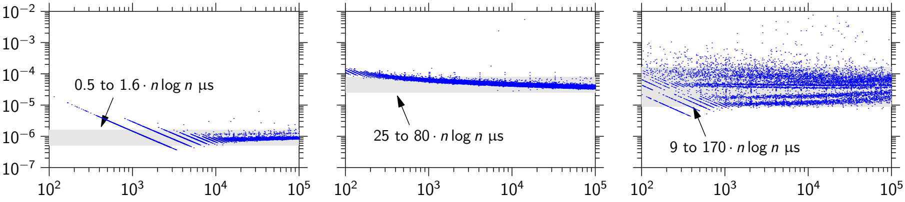
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Experimental Results for Voronoi Diagrams



Runtime per seconds divided by $n \log n$, for vroniFp, vroniMp212, cgvdEx.

- Conclusion:
  - vroniFp about 50–60 times faster than vroniMp*.
  - vroniFp about 20–100 times faster than cgvd*.
  - cgvdFp only 1.5 times faster than cgvdEx.
    - Crashed on 937 datasets due to floating-point exception.
  - On average, cgvdEx is slightly faster than vroniMp*.
    - cgvdEx timings vary by a factor of 20.
    - A few cgvdEx results were numerically clearly wrong.



Runtime per seconds divided by $n \log n$, for vroniFp, vroniMp212, cgvdEx.

Experimental Results for Voronoi Diagrams

- [Held&Mann (2011)]: Implemented a verifier based on GMP's `mpq_t` data type.
- Brute-force all-pairs distance computations used.
- Hence, verification was limited to small inputs with up to 2 000 segments.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Experimental Results for Voronoi Diagrams

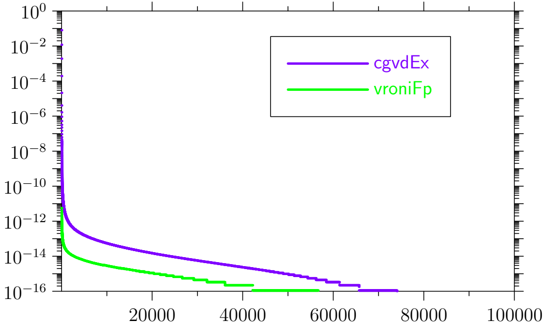
- [Held&Mann (2011)]: Implemented a verifier based on GMP's `mpq_t` data type.
- Brute-force all-pairs distance computations used.
- Hence, verification was limited to small inputs with up to 2 000 segments.

Deviation: Difference in the distances of a node to its defining sites.

Violation: Another site is closer to a node than defining sites.

Experimental Results for Voronoi Diagrams

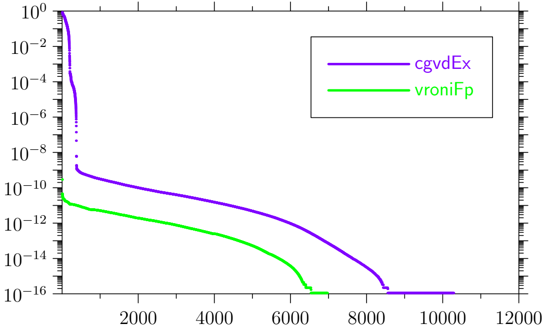
- Deviation: Difference in the distances of a node to its defining sites.



	VRONI	CGAL
maximum:	$7.25 \cdot 10^{-8}$	$8.04 \cdot 10^{-1}$
median:	$2.22 \cdot 10^{-16}$	$3.10 \cdot 10^{-15}$

Experimental Results for Voronoi Diagrams

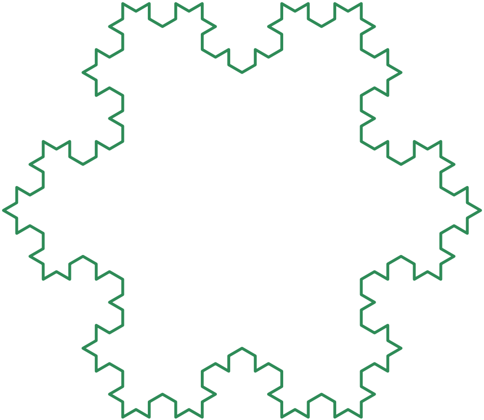
- Violation: Another site is closer to a node than defining sites.



	VRONI	CGAL
maximum:	$1.75 \cdot 10^{-6}$	$9.14 \cdot 10^{-1}$
median:	$3.87 \cdot 10^{-14}$	$3.94 \cdot 10^{-12}$

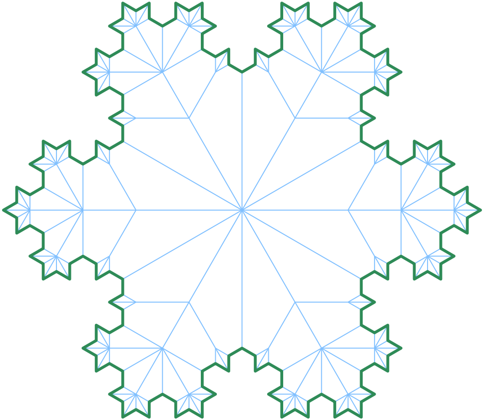
Software for Computing Straight Skeletons: SURFER and SURFER2

- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,

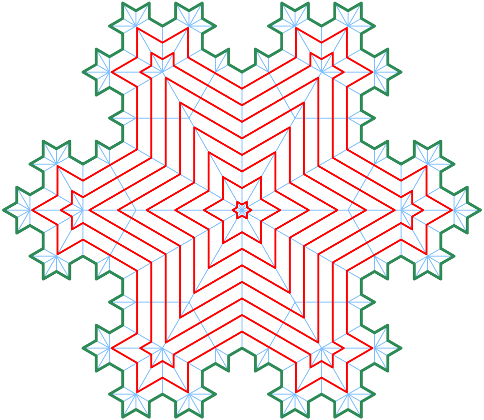


Software for Computing Straight Skeletons: SURFER and SURFER2

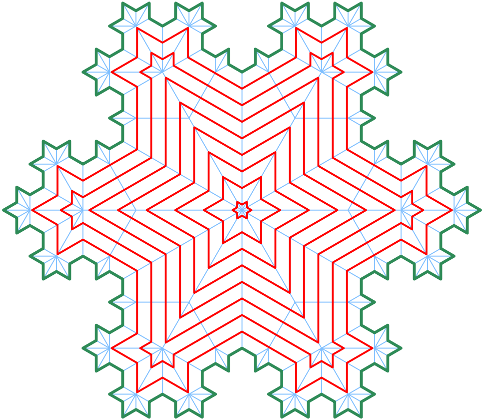
- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,
  - simulate the wavefront propagation based on kinetic triangulations.



- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,
  - simulate the wavefront propagation based on kinetic triangulations.
- SURFER also computes mitered offset curves.

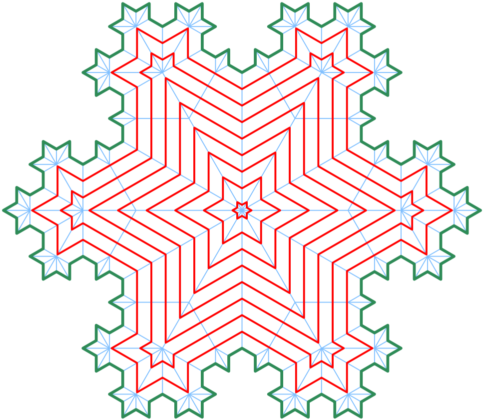


- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,
  - simulate the wavefront propagation based on kinetic triangulations.
- SURFER also computes mitered offset curves.
- SURFER2 handles weighted straight skeletons, too.



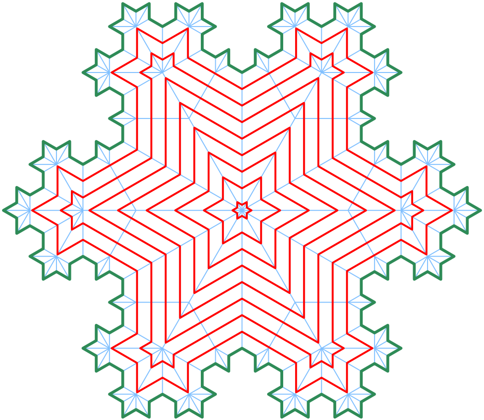
Software for Computing Straight Skeletons: SURFER and SURFER2

- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,
  - simulate the wavefront propagation based on kinetic triangulations.
- SURFER also computes mitered offset curves.
- SURFER2 handles weighted straight skeletons, too.
- Complexity:
  - $O(n)$ space complexity, where n is the number of segments.
  - $O(n \log n)$ average time complexity, $O(n^3 \log n)$ worst-case time complexity.



Software for Computing Straight Skeletons: SURFER and SURFER2

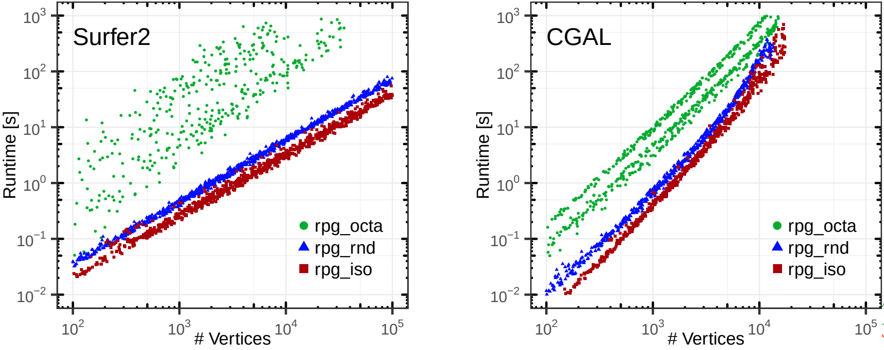
- [Held&Palfrader (2012–2020)]: SURFER, SURFER2
  - compute straight skeletons of planar straight-line graphs,
  - simulate the wavefront propagation based on kinetic triangulations.
- SURFER also computes mitered offset curves.
- SURFER2 handles weighted straight skeletons, too.
- Complexity:
  - $O(n)$ space complexity, where n is the number of segments.
  - $O(n \log n)$ average time complexity, $O(n^3 \log n)$ worst-case time complexity.
- SURFER is based on standard IEEE 754 floating-point arithmetic.



- SURFER2 is based on exact geometric computing.

Eder&Held&Palfrader (2020)

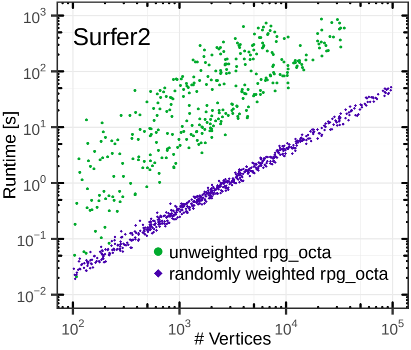
- SURFER2: Based on CGAL's exact-predicates-exact-constructions algebraic kernel with square root, which is backed by CORE's `Core::Expr` exact number type.
- CGAL 5.0: Based on exact-predicates-inexact-constructions algebraic kernel.

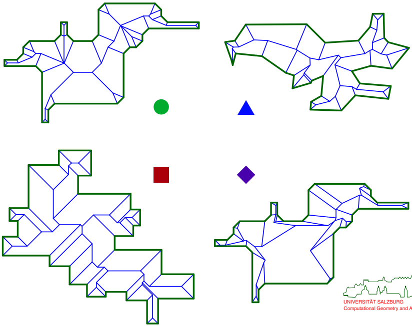


Eder&Held&Palfrader (2020)

Multiple events that occur simultaneously have a significant impact on the practical runtime of SURFER2 if the `CORE::Expr` number type is used!

- That is, using the `CORE::Expr` number type forces one to abandon the concept of unit-cost comparisons!





The End!

I hope that you enjoyed this course, and I wish you all the best for your future studies.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab