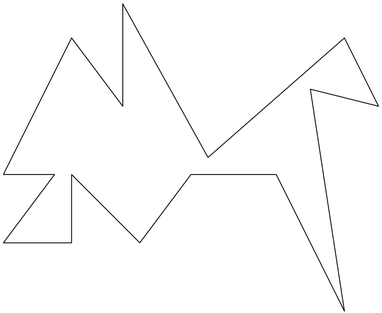
Time-Space Trade-offs for Triangulating a Simple Polygon

Boris Aronov[1] Matias Korman[2] Simon Pratt[3]
André van Renssen[4,5] Marcel Roeloffzen[4,5]

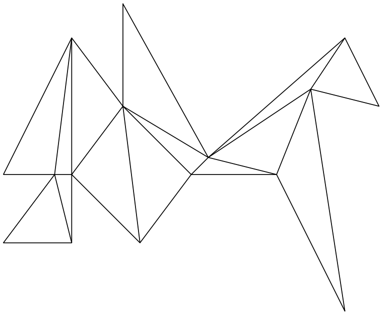
[1]New York University [2]Tohoku University [3]University of Waterloo
[4]National Institute of Informatics
[5]JST, ERATO, Kawarabayashi Large Graph Project

Problem



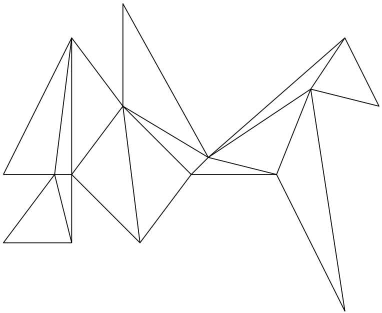
Given a simple polygon:

Problem



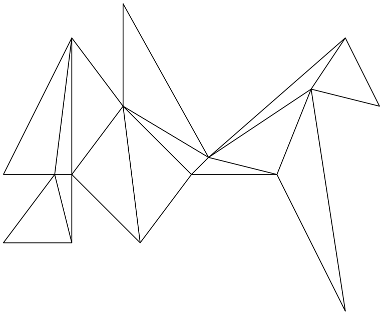
Given a simple polygon: Triangulate!

Why?



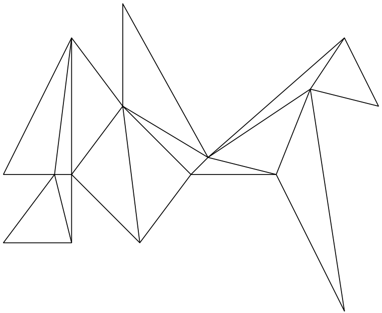
- Don't we already have Chazelle's $O(n)$ time algorithm?

Why?



- Don't we already have Chazelle's $O(n)$ time algorithm?
- Chazelle's algorithm uses $O(n)$ words of space
- What if we only have $O(s < n)$ words of space?

Why?



- Don't we already have Chazelle's $O(n)$ time algorithm?
- Chazelle's algorithm uses $O(n)$ words of space
- What if we only have $O(s < n)$ words of space?
- Ideally, we'd like a time-space trade-off

Model

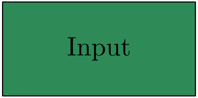
s-Workspace Model:

- Also called constant workspace

Model

s-Workspace Model:

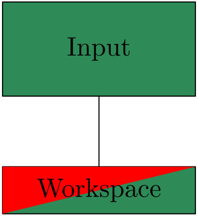
- Also called constant workspace
- Input is read-only, but we have random access



Model

s-Workspace Model:

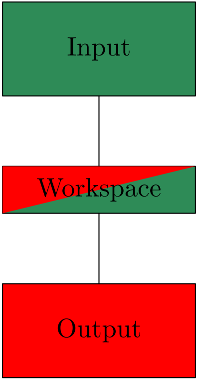
- Also called constant workspace
- Input is read-only, but we have random access
- $O(s)$ words of extra (read/write) space



Model

s-Workspace Model:

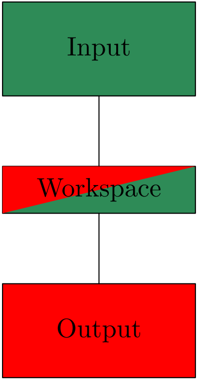
- Also called constant workspace
- Input is read-only, but we have random access
- $O(s)$ words of extra (read/write) space
- Output is write-only, no random access



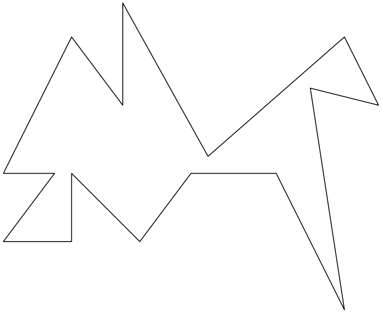
Model

s-Workspace Model:

- Also called constant workspace
- Input is read-only, but we have random access
- $O(s)$ words of extra (read/write) space
- Output is write-only, no random access
- Otherwise, normal word RAM

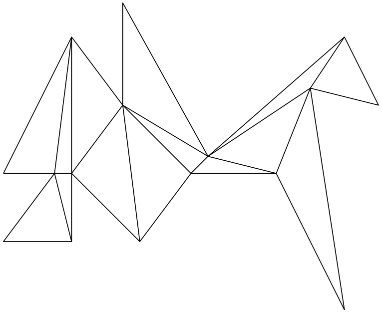


Problem



- Input:
  - A simple polygon
  - Given as vertices in clockwise order: $v_1 v_2, v_2 v_3, \ldots, v_{n-1} v_n$

Problem



- Input:
  - A simple polygon
  - Given as vertices in clockwise order: $v_1 v_2, v_2 v_3, \ldots, v_{n-1} v_n$
- Output:
  - A maximal set of straight, non-intersecting line segments between vertices of the input

Results

- $O\left(n^2/s + n \log s \log^5(n/s)\right)$ expected time, for $s \in O(n)$

Results

- $O\left(n^2/s + n \log s \log^5(n/s)\right)$ expected time, for $s \in O(n)$
- $O\left(n^2/s\right)$ expected time, for "reasonable" $s \in O\left(\frac{n}{\log n \log^5 \log n}\right)$

Tool 1: Chazelle's Algorithm

- Triangulates a simple polygon
- $O(n)$ time
- $O(n)$ space

Credit: *Chazelle, 1991*

Tool 2: Memory-Constrained Triangulation

- Triangulates a simple polygon
- $O\left(n^2\right)$ time
- $O(1)$ space

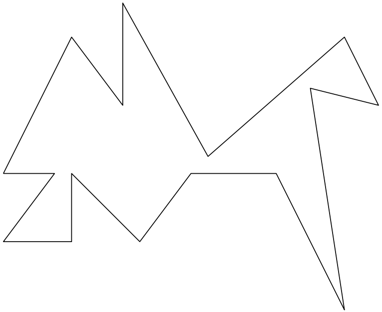
Credit: *Asano et al., 2013*

Tool 3: Har-Peled's Algorithm

- Finds the shortest path between two vertices within a simple polygon
- $O\left(n^2/s + n \log s \log^4(n/s)\right)$ expected time
- $O(s)$ space

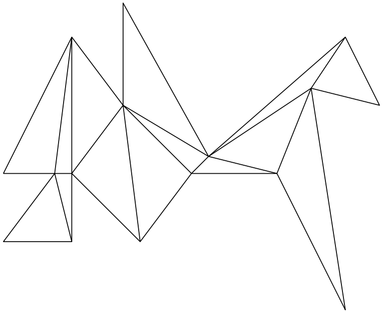
Credit: *Har-Peled, SoCG 2015, JoCG 2016*

Overview



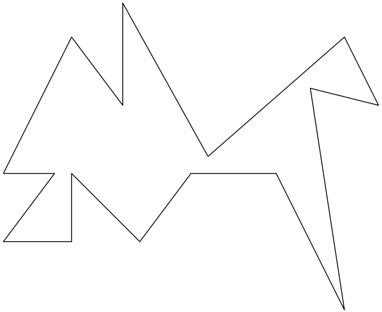
- If polygon fits in available memory:

Overview



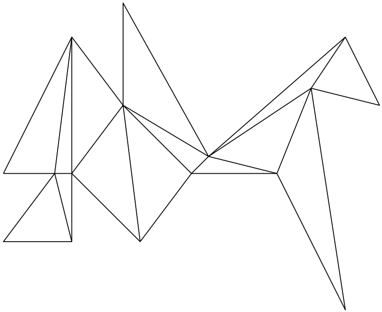
- If polygon fits in available memory: Use Chazelle's algorithm

Overview



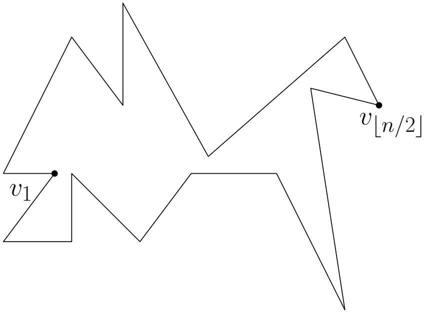
- If polygon fits in available memory: Use Chazelle's algorithm
- Else if s is constant:

Overview



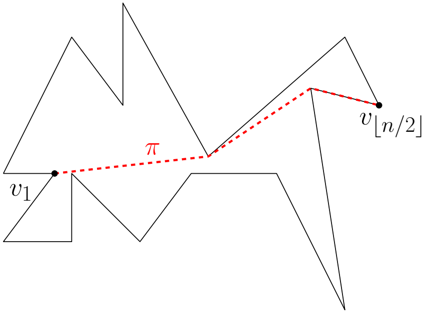
- If polygon fits in available memory: Use Chazelle's algorithm
- Else if s is constant: Use memory-constrained triangulation

Overview



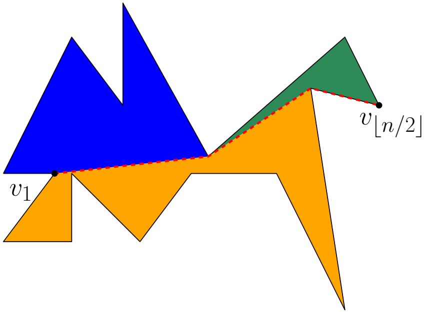
- If polygon fits in available memory: Use Chazelle's algorithm
- Else if s is constant: Use memory-constrained triangulation
- Else:

Overview



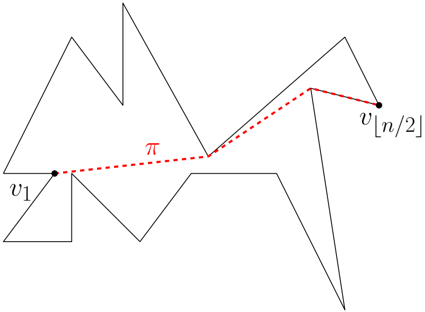
- If polygon fits in available memory: Use Chazelle's algorithm
- Else if s is constant: Use memory-constrained triangulation
- Else:
  - Use Har-Peled's algorithm to find geodesic from v_1 to $v_{\lfloor n/2 \rfloor}$

Overview



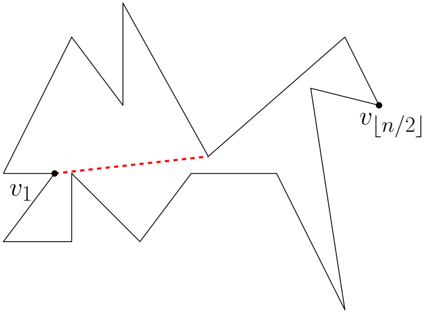
- If polygon fits in available memory: Use Chazelle's algorithm
- Else if s is constant: Use memory-constrained triangulation
- Else:
  - Use Har-Peled's algorithm to find geodesic from v_1 to $v_{\lfloor n/2 \rfloor}$
  - Recurse on subpolygons induced by π, with smaller s

First Challenge



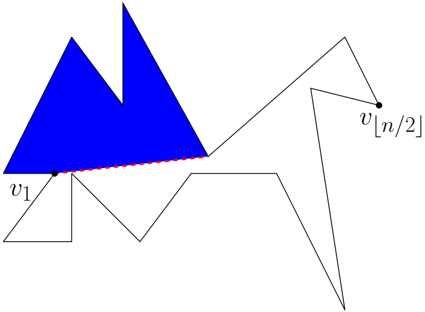
- Problem:
  - π might not fit into available memory

First Challenge



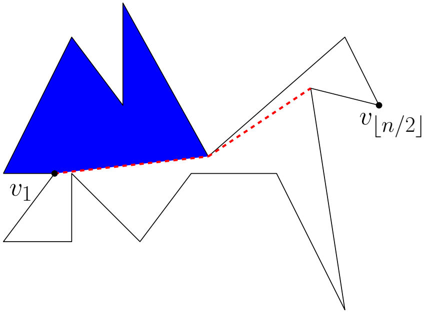
- Problem:
  - π might not fit into available memory
- Solution:
  - "Pause" Har-Peled after each edge

First Challenge



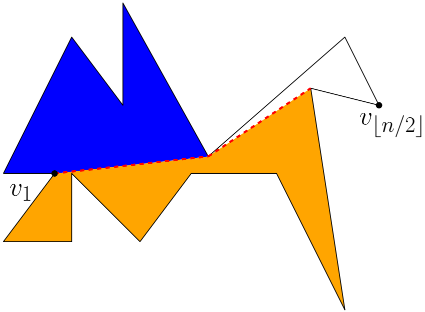
- Problem:
  - π might not fit into available memory
- Solution:
  - "Pause" Har-Peled after each edge
  - Recurse on subpolygons as they are created

First Challenge



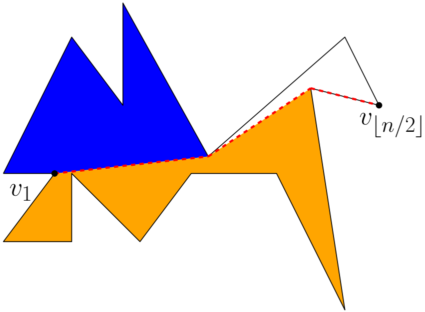
- Problem:
  - π might not fit into available memory
- Solution:
  - “Pause” Har-Peled after each edge
  - Recurse on subpolygons as they are created

First Challenge



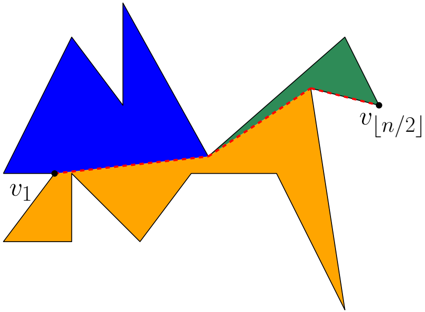
- Problem:
  - π might not fit into available memory
- Solution:
  - "Pause" Har-Peled after each edge
  - Recurse on subpolygons as they are created

First Challenge



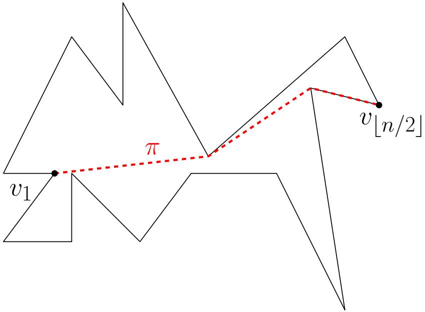
- Problem:
  - π might not fit into available memory
- Solution:
  - "Pause" Har-Peled after each edge
  - Recurse on subpolygons as they are created

First Challenge



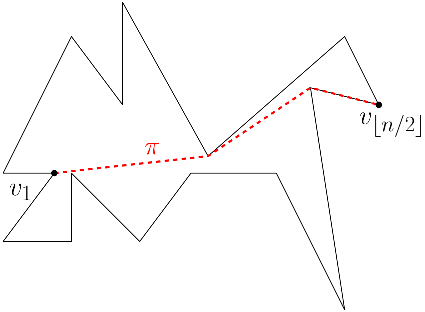
- Problem:
  - π might not fit into available memory
- Solution:
  - "Pause" Har-Peled after each edge
  - Recurse on subpolygons as they are created

Second Challenge



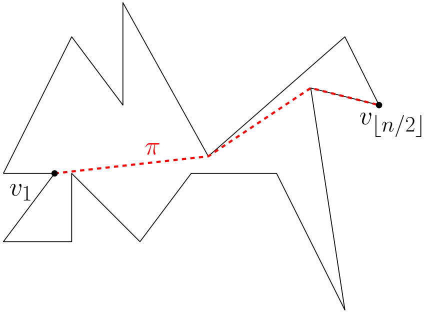
- Problem:
  - A subpolygon can be bounded by more than a single edge of π

Second Challenge



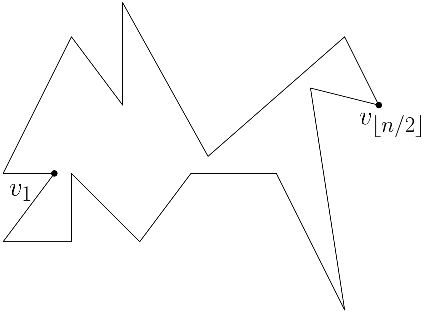
- Problem:
  - A subpolygon can be bounded by more than a single edge of π
  - We need to store a list of edges on π

Second Challenge

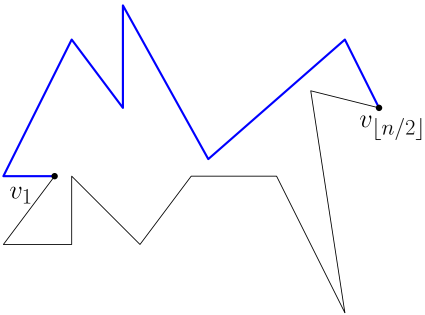


- Problem:
  - A subpolygon can be bounded by more than a single edge of π
  - We need to store a list of edges on π
  - This list of edges might not fit in memory

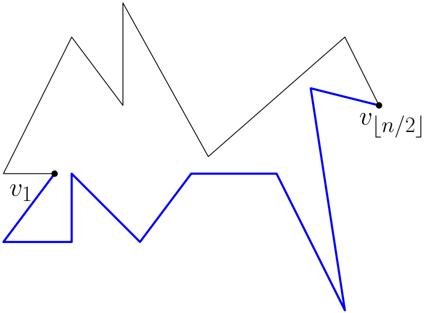
Alternating Diagonals



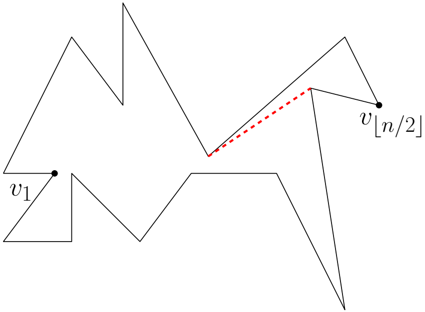
Alternating Diagonals



Alternating Diagonals



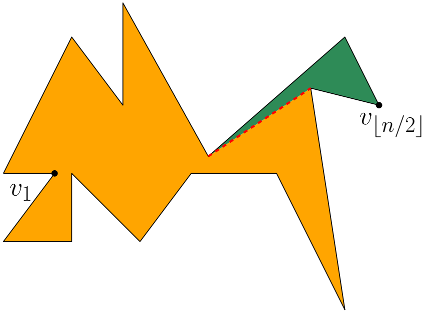
Alternating Diagonals



Alternating diagonal:

- An edge with one vertex on the "top" and one on the "bottom" of the polygon

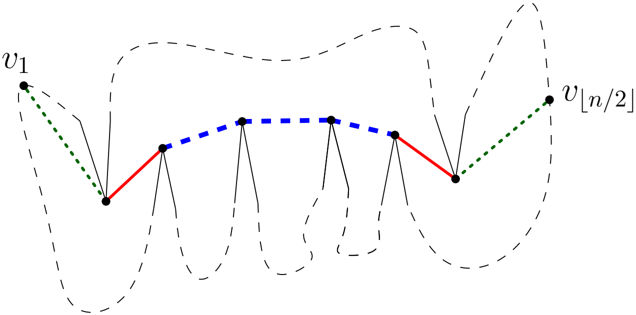
Alternating Diagonals



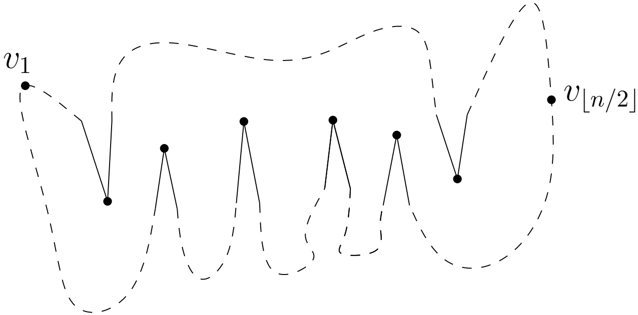
Alternating diagonal:

- An edge with one vertex on the "top" and one on the "bottom" of the polygon
- In other words, an edge which separates v_1 and $v_{\lfloor n/2 \rfloor}$ into different induced subpolygons

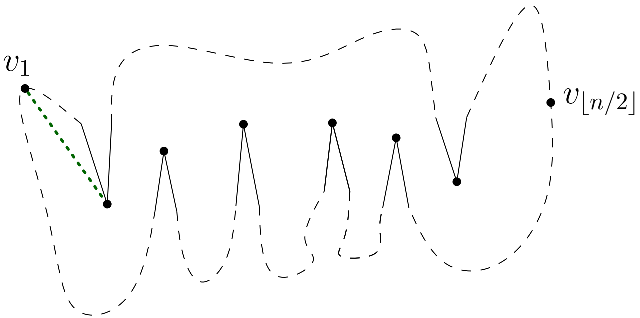
More Details



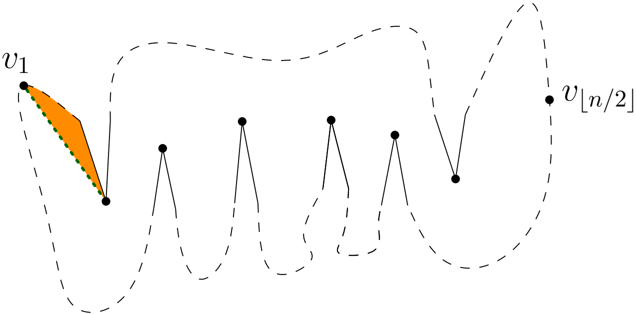
More Details



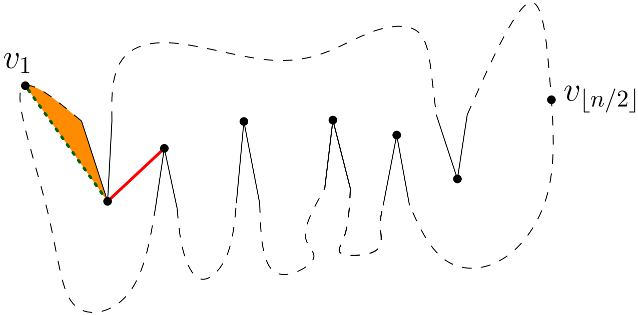
More Details



More Details

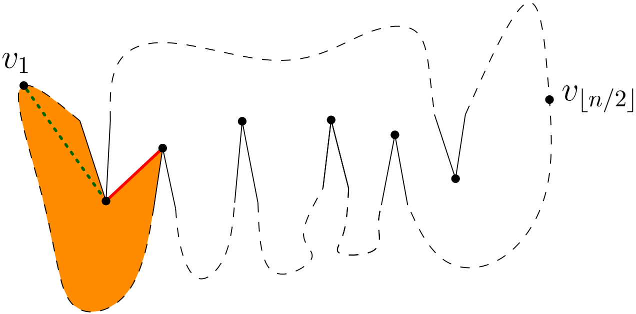


More Details



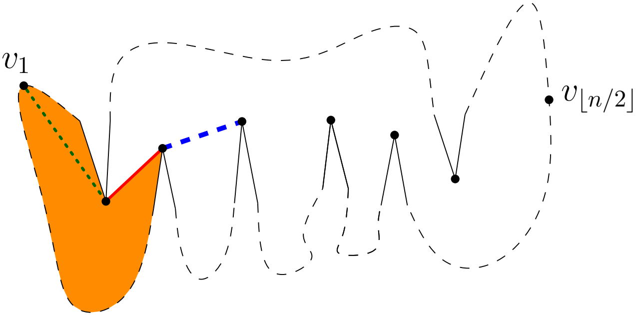
- Store:
  - Last alternating diagonal a_c

More Details



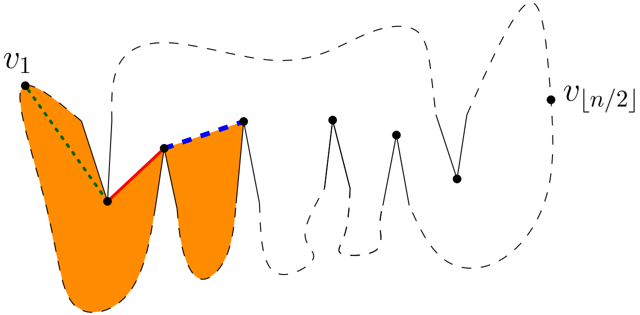
- Store:
  - Last alternating diagonal a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



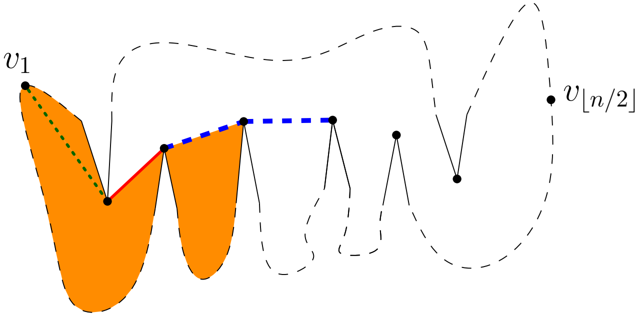
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



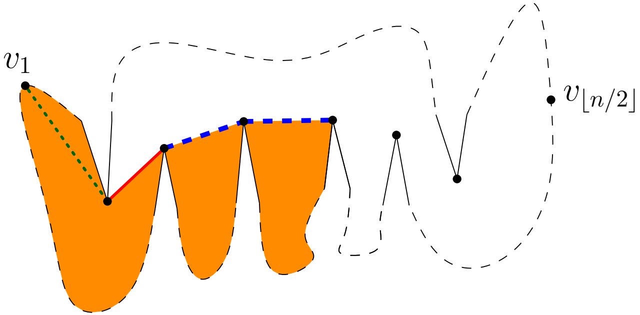
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



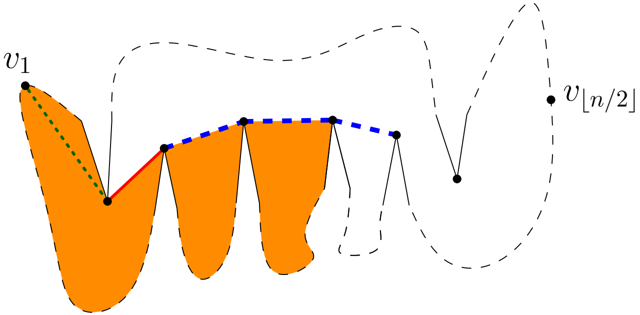
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



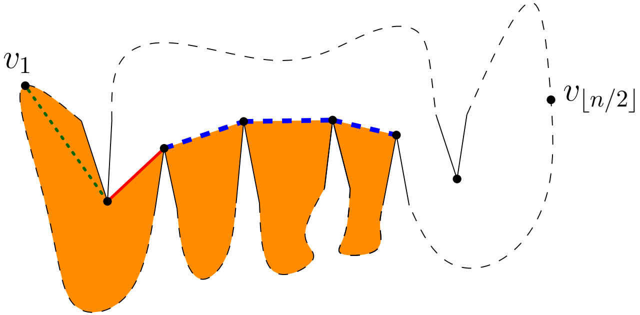
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



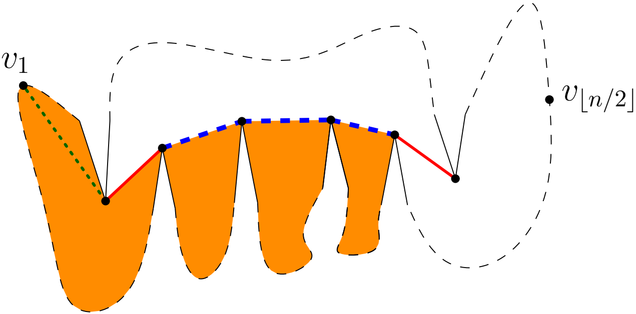
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



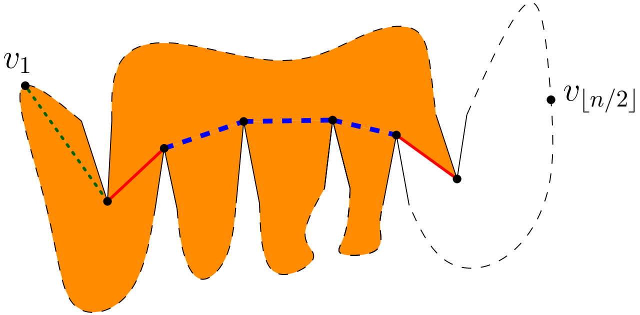
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



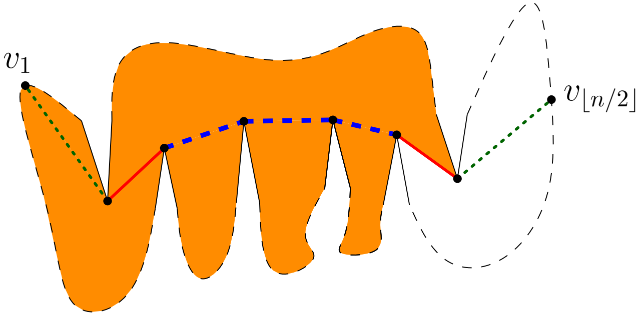
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



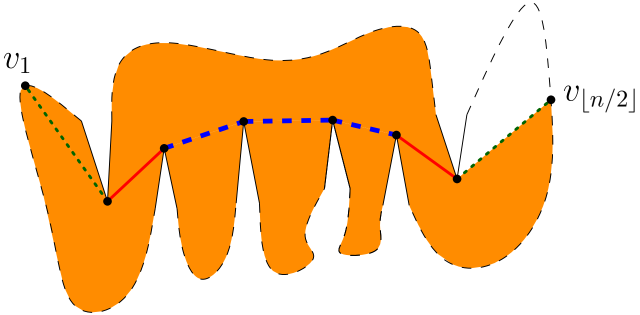
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



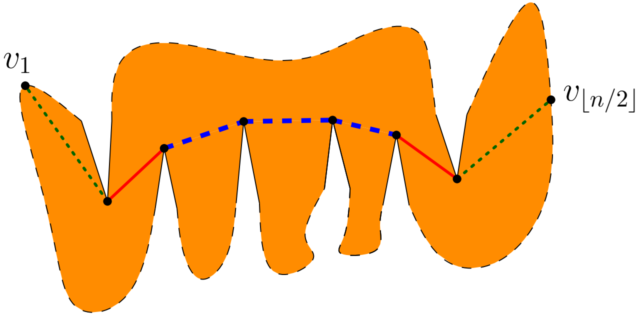
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



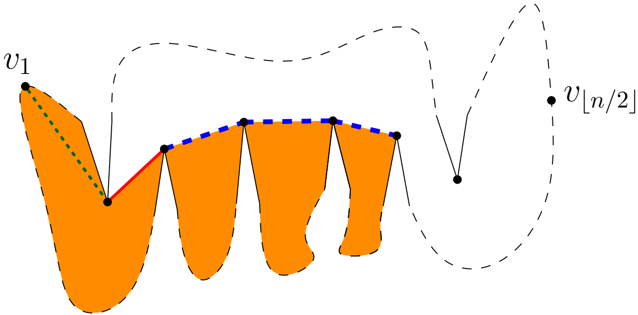
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

More Details



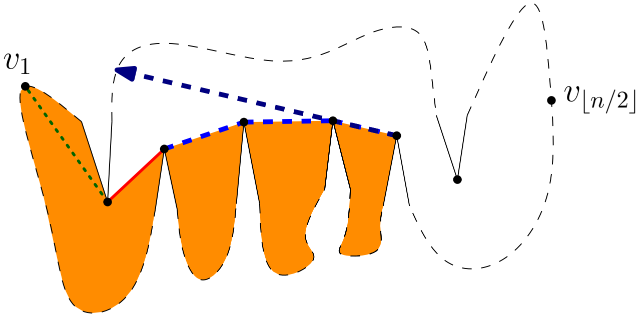
- Store:
  - Last alternating diagonal a_c
  - Edges of π since a_c
- Maintain invariant:
  - Subpolygon induced by a_c containing v_1 is triangulated

Second Challenge



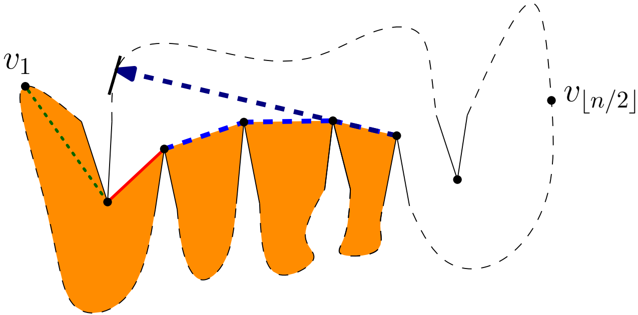
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory

Second Challenge



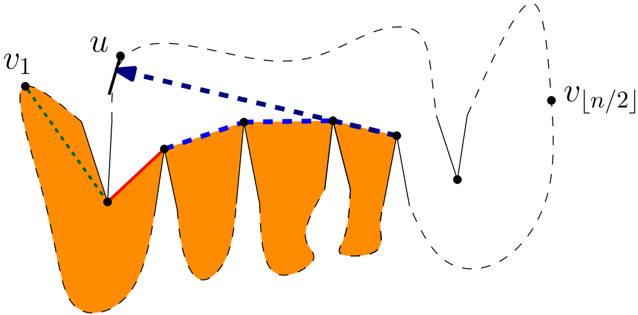
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



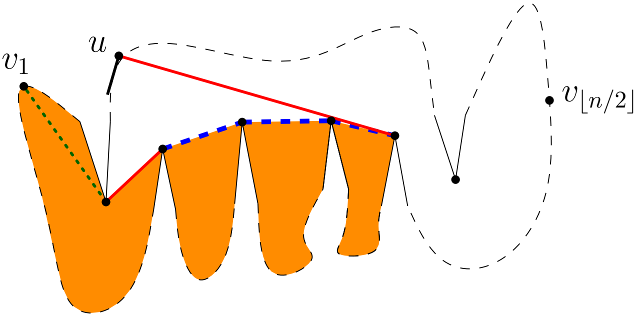
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



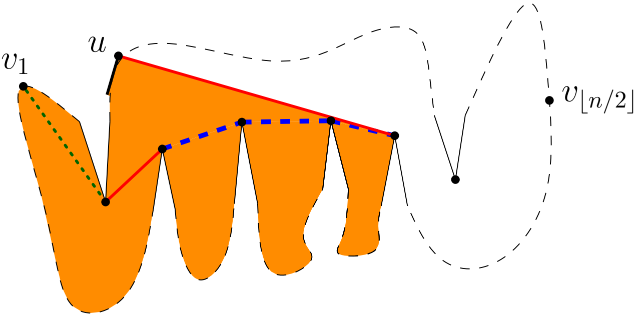
- Problem:
    - There may be more edges between alternating diagonals than can fit in memory
- Solution:
    - Find an alternating diagonal (not on π)
    - $O(n)$ time, $O(1)$ extra space

Second Challenge



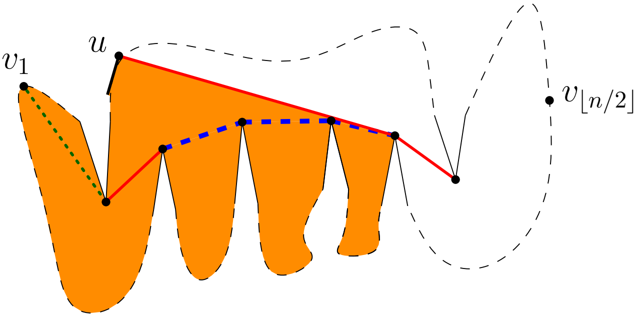
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



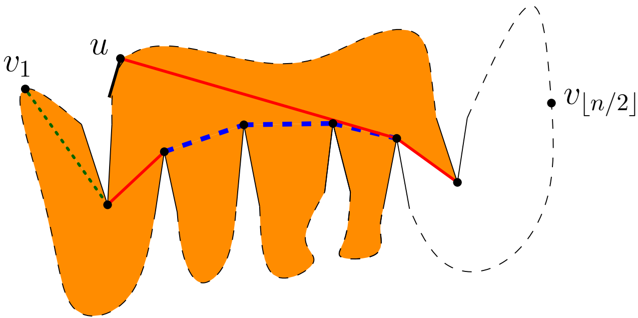
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



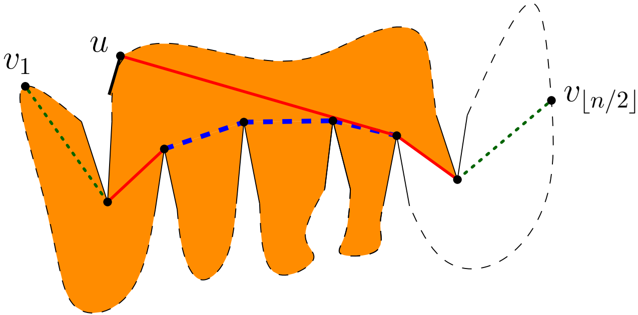
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



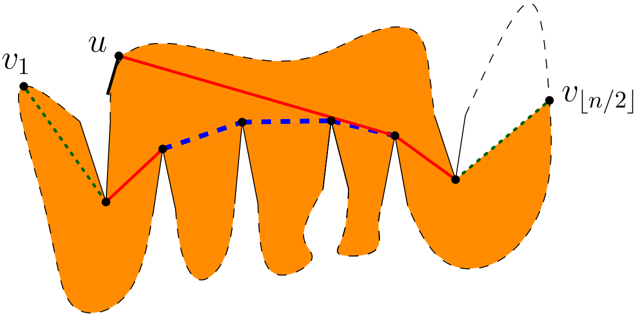
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



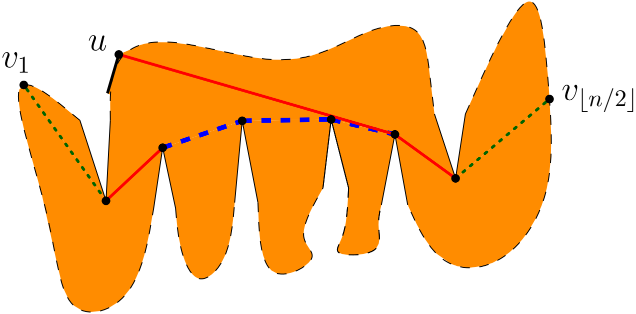
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



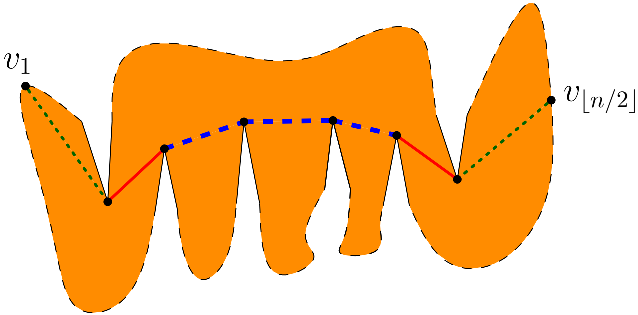
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Second Challenge



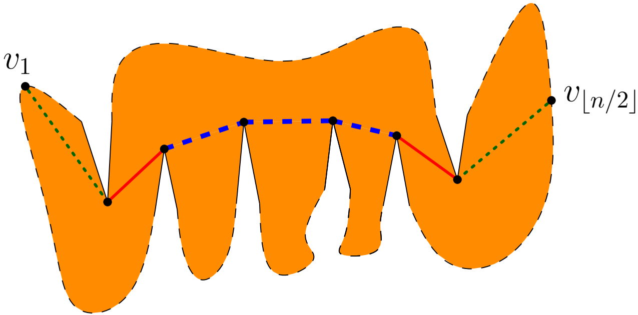
- Problem:
  - There may be more edges between alternating diagonals than can fit in memory
- Solution:
  - Find an alternating diagonal (not on π)
  - $O(n)$ time, $O(1)$ extra space

Efficiency: Overview



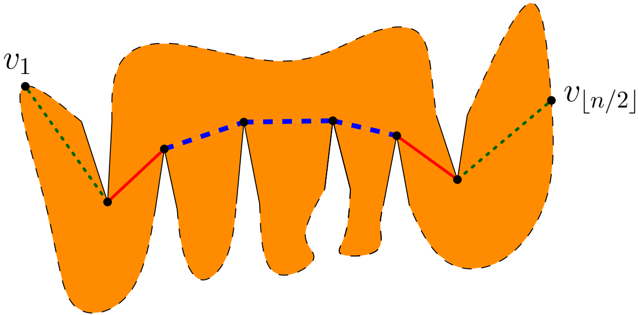
- SPACE: $O(s)$

Efficiency: Overview



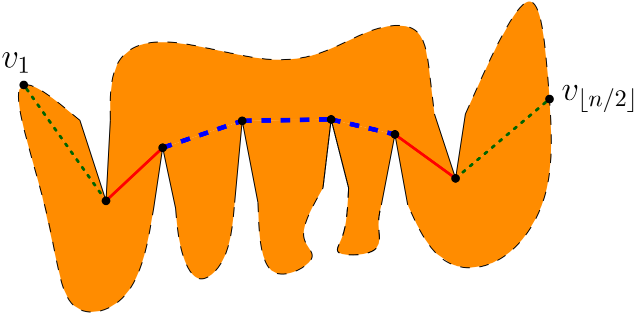
- SPACE: $O(s)$
  - Use a parameter τ to keep track of available space

Efficiency: Overview



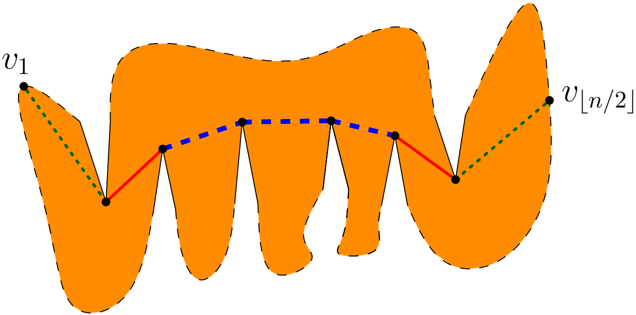
- SPACE: $O(s)$
  - Use a parameter τ to keep track of available space
  - Initially, $\tau = s$

Efficiency: Overview



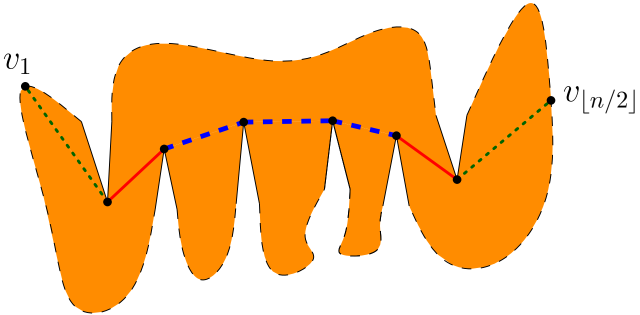
- Space: $O(s)$
  - Use a parameter τ to keep track of available space
  - Initially, $\tau = s$
  - τ shrinks by a constant factor at each level of recursion

Efficiency: Overview



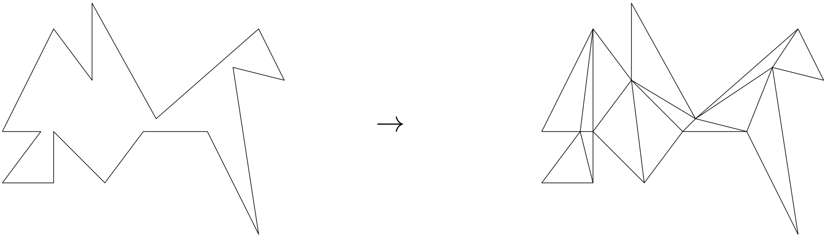
- SPACE: $O(s)$
  - Use a parameter τ to keep track of available space
  - Initially, $\tau = s$
  - τ shrinks by a constant factor at each level of recursion
- TIME: $O\big(n^2/s + n \log s \log^5(n/s)\big)$ expected

Efficiency: Overview



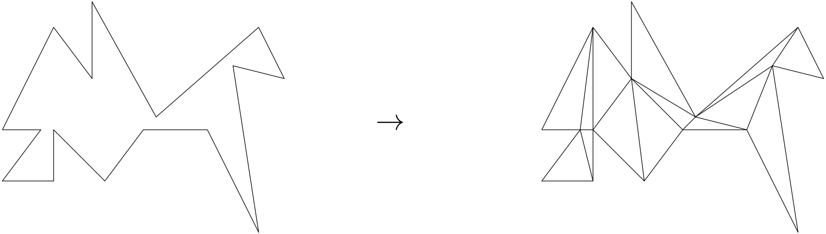
- SPACE: $O(s)$
  - Use a parameter τ to keep track of available space
  - Initially, $\tau = s$
  - τ shrinks by a constant factor at each level of recursion
- TIME: $O\big(n^2/s + n \log s \log^5(n/s)\big)$ expected
  - Most work done in computing the geodesic

Conclusion



Using Chazelle's, Asano's, and Har-Peled's algorithms as building blocks.

Conclusion

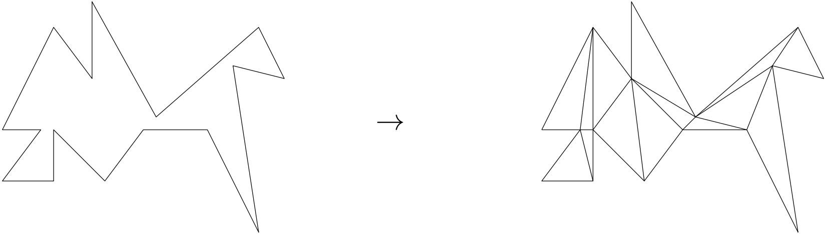


Using Chazelle's, Asano's, and Har-Peled's algorithms as building blocks.

Triangulate a polygon in:

- $O(s)$ space
- $O\left(n^2/s + n \log s \log^5(n/s)\right)$ expected time, for $s \in O(n)$

Conclusion



Using Chazelle's, Asano's, and Har-Peled's algorithms as building blocks.

Triangulate a polygon in:

- $O(s)$ space
- $O\left(n^2/s + n \log s \log^5(n/s)\right)$ expected time, for $s \in O(n)$
- $O\left(n^2/s\right)$ expected time, for "reasonable" $s \in O\left(\frac{n}{\log n \log^5 \log n}\right)$

References

1. Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, André Schulz. *Memory-constrained algorithms for simple polygons.* Journal of Computational Geometry, Volume 46, Issue 8, 2013, Pages 959–969.
2. Bernard Chazelle. *Triangulating a simple polygon in linear time.* Journal of Discrete & Computational Geometry, Number 6, Issue 3, 2007, Pages 485–524.
3. Sariel Har-Peled. "Shortest path in a polygon using sublinear space." Journal of Computational Geometry, Volume 7, Number 2, 2016, Pages 19–45.