A Randomized Approach to Point Set Triangulation

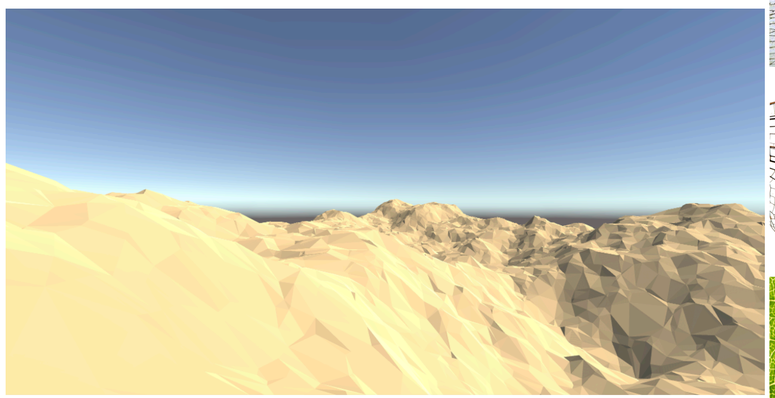
CS262

Pooja Shyamsundar

Motivation

- Robotics - Collision checking
- Graphics - Polygons are the basic building blocks in most geometric applications
- Terrain mapping - Decompose complex structures into simpler ones
- Generating Cryptographic Keys from images

[1] https://www.pngwing.com/en/search?q=delaunay+Triangulation



https://straypixels.net/delaunay-triangulation-terrain/

https://fbellelli.com/posts/2021-07-08-the-fascinating-world-of-voronoi-diagrams/

https://spectrum.ieee.org/custom-processor-speeds-up-robot-motion-

Introduction

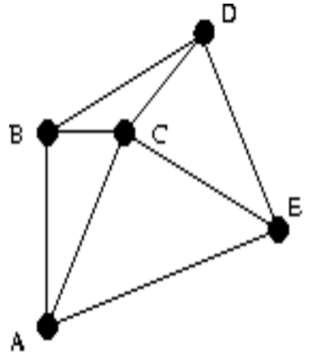
- In robotic motion planning the most time consuming part is triangulation for collision checking, especially in high dimensional spaces.
- In problems that deal with processing point clouds (from LiDAR or stereo camera data) to build meshes, triangulation is used in each update cycle.
- When building 3D meshes - used in CAD.
- Runtime is important, especially in safety critical applications.

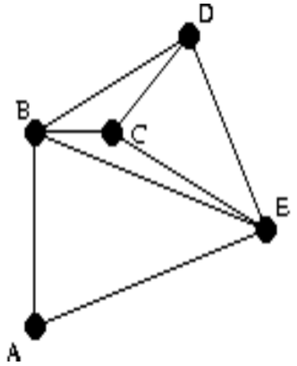
Introduction

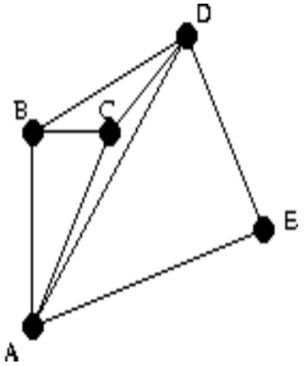
- The general problem of point set triangulation is NP-complete
- Various approaches for fast triangulation have been attempted.
- The naive approach to polygon triangulation is O(n^4). A long series of papers and algorithms to improve this bound.
- Chazelle [5] proposed a linear time algorithm for polygon triangulation but it is complicated and theoretical. No known practical implementations.

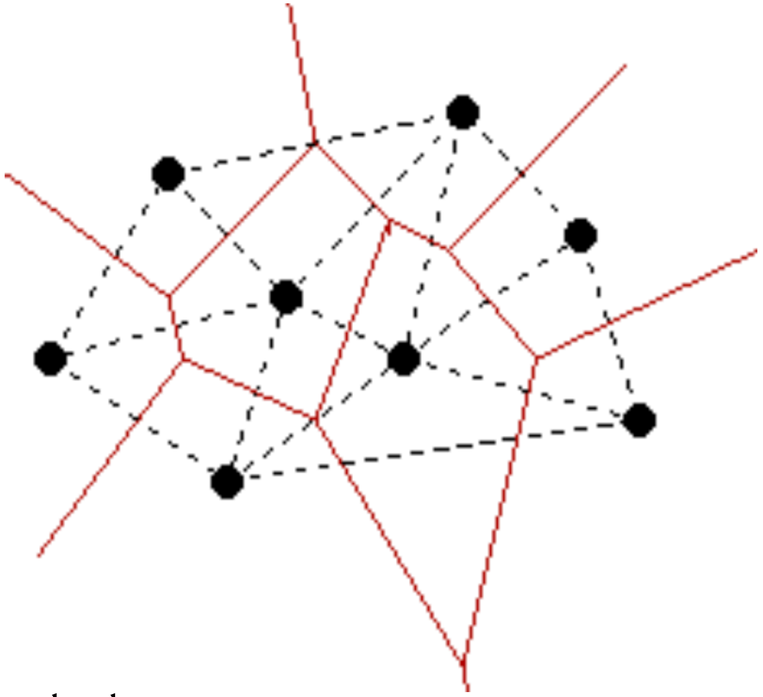
Problem Statement

- Speed up triangulation.
- Given a boundary, triangulate a randomly generated point set within the boundary in linear time.
- The original problem is general point set triangulation.
- Polygon triangulation, Delaunay triangulation and Voronoi diagrams are subproblems.









http://www.geom.uiuc.edu/~samuelp/del_project.html

Related Work

- [3] talks about how Delaunay triangulations in theory can have optimal bounds that rarely hold in practical cases. They provide a method for fast triangulation using randomized incremental construction that holds a O(nlogn) bound in real time applications.
- [4] This paper gives a randomized incremental algorithm for Voronoi and Delaunay triangulations in O(nlogn). Most of the processing is done ‘on-line’ and it also optimizes space usage.

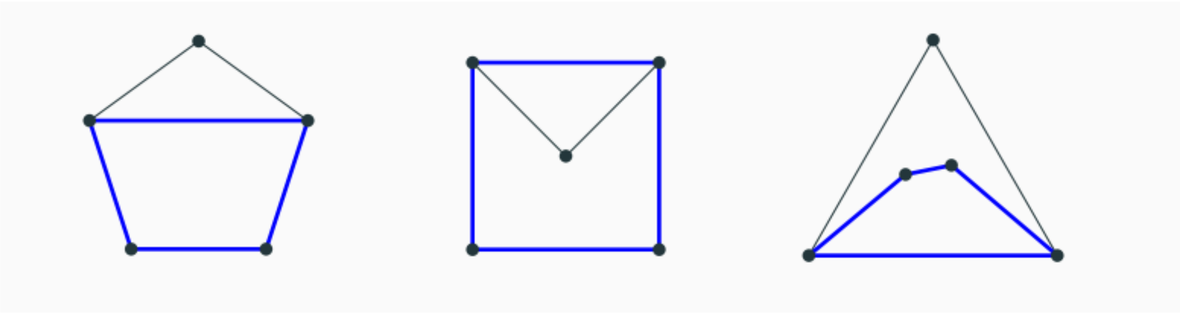
Benchmarking

State Of The Art

Algorithm	Runtime
Incremental Algorithm with sorting	O(n^2)
Incremental Algorithm without sorting	O(nlogn)
Incremental with hierarchical structure	O(n^2)
Incremental randomized	O(nlogn) expected
Incremental with Auxiliary points	O(n^2)
Grahams Algorithm	O(nlogn)
Divide and Conquer	O(nlogn)
Bower Watson Based	O(n^2)

Background

- Erdos-Szekeres convex polygon problem
- For any integer n ≥ 3, what is the smallest positive integer ES(n) such that any set of at least ES(n) points in the plane in contains n points in convex position?
- How many points do I need **at least**, until I am **guaranteed** to find a convex n-gon?
- For example, we can come up with the following cases for ES(4) so ES(4) <= 5



Background

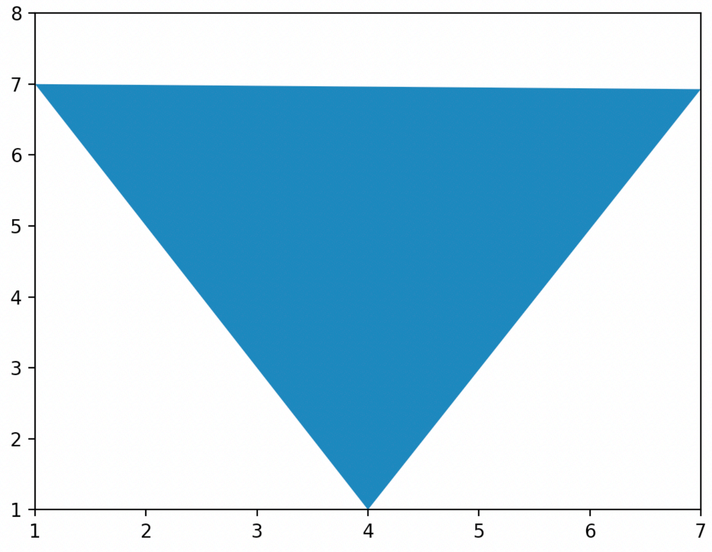
Erdos-Szekeres convex polygon problem - proof of existence of n points in convex position out of n + k points, for a minimum value of k.

$$N(4) = 5$$
$$N(5) = 9$$
$$N(6) = 17$$
$$\vdots$$
$$N(n) = 2^{(n-2)} + 1$$

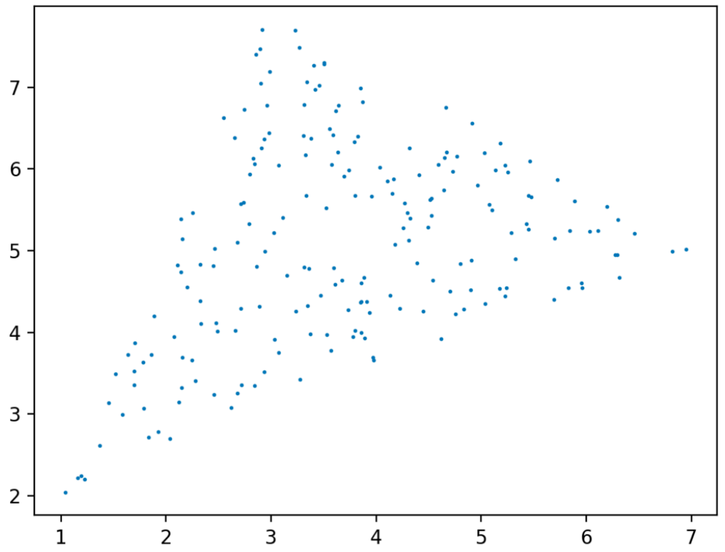
Methodology

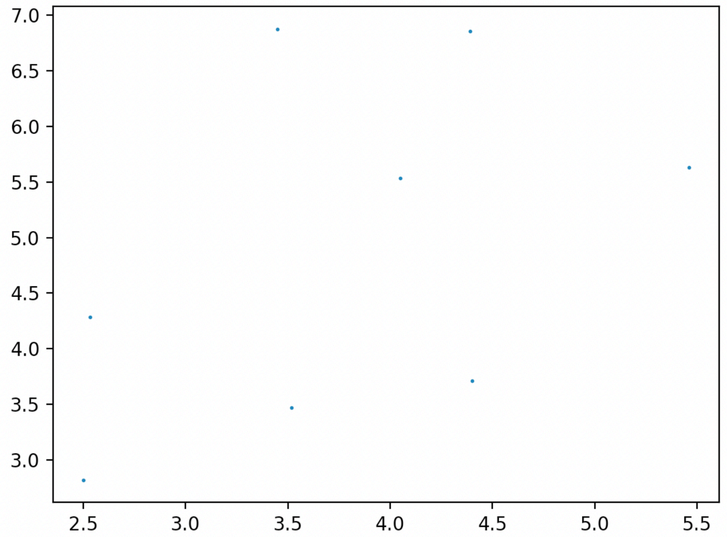
1. Form a triangle within the given boundary.
2. Generate n + k uniformly random points within the triangle (O(n)). Where n+k is the Erdos-Szekeres number.

Generate triangular area within bounds



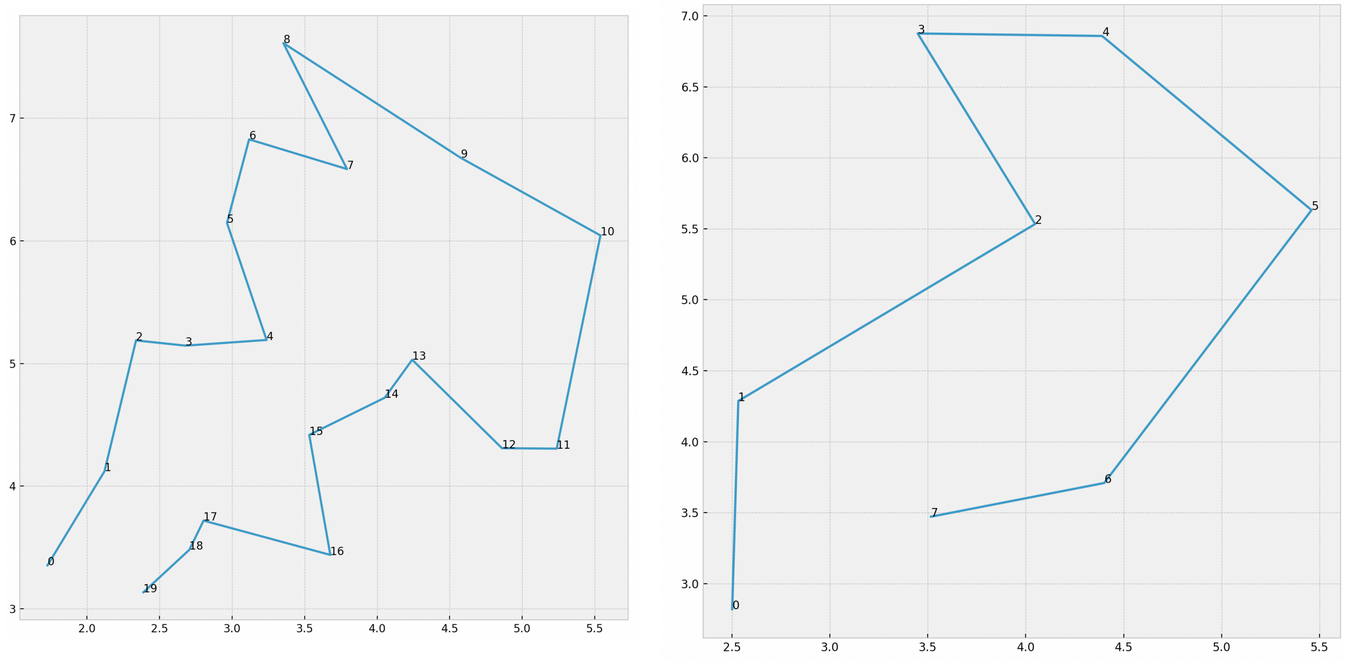
Generate Random Points within the area



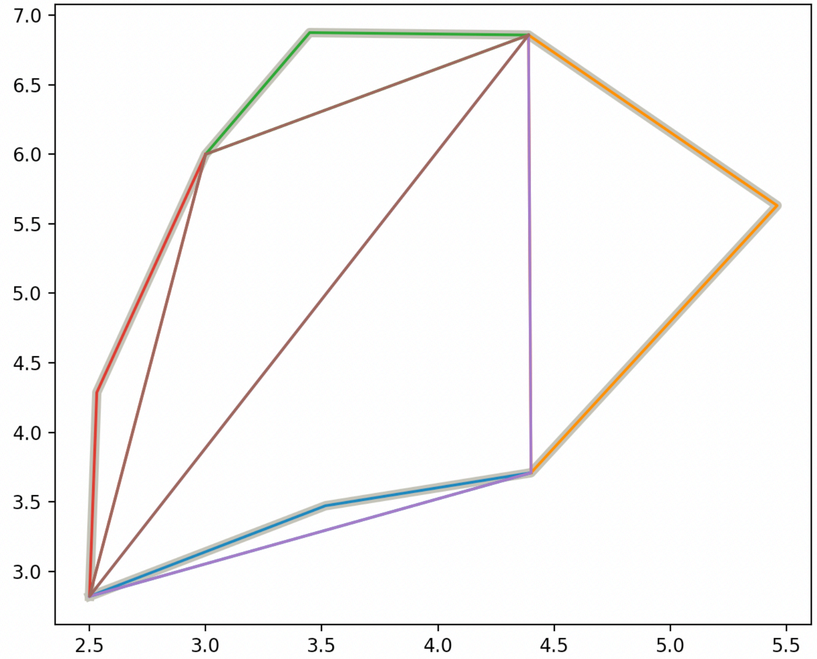


Methodology

3. Pick n of these n + k points and check if they are convex.
4. If convex, proceed to step 5. Else FAIL.
5. triangulate in O(n) - Simple iteration over edges and clip-off triangles as they are formed.

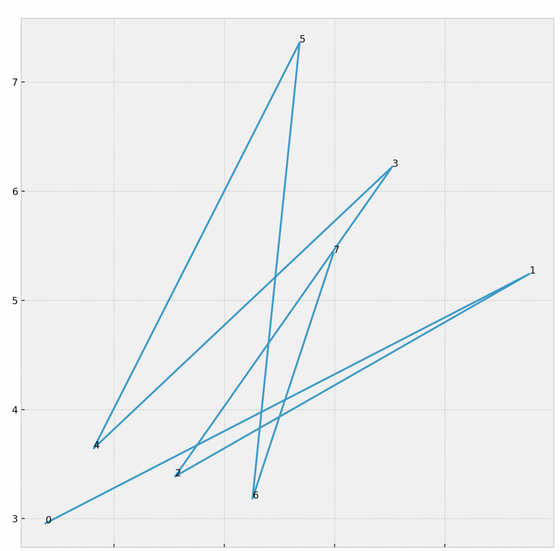


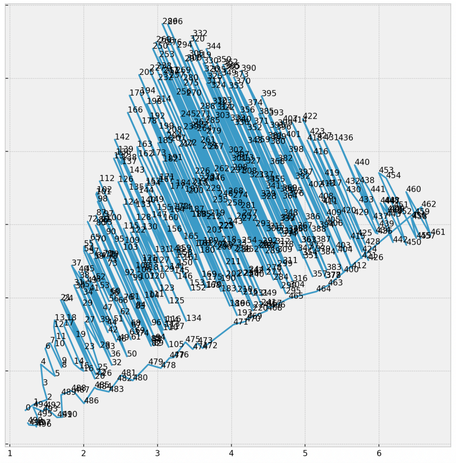
Pick k out of n+k points and form a convex polygon

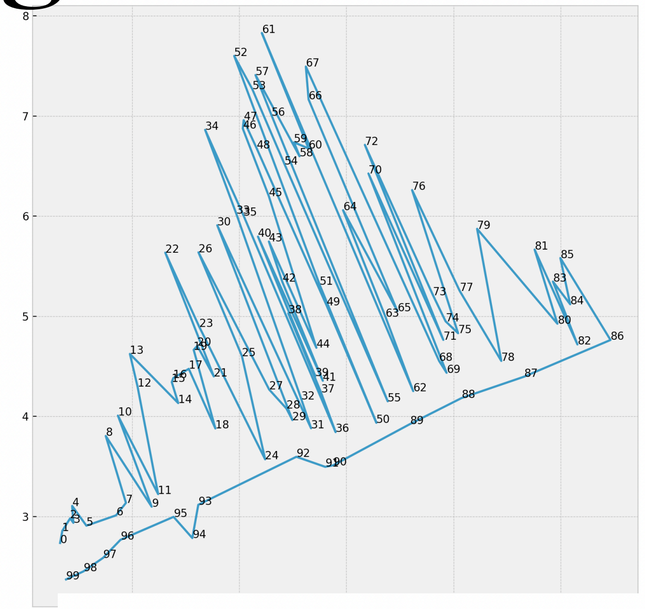


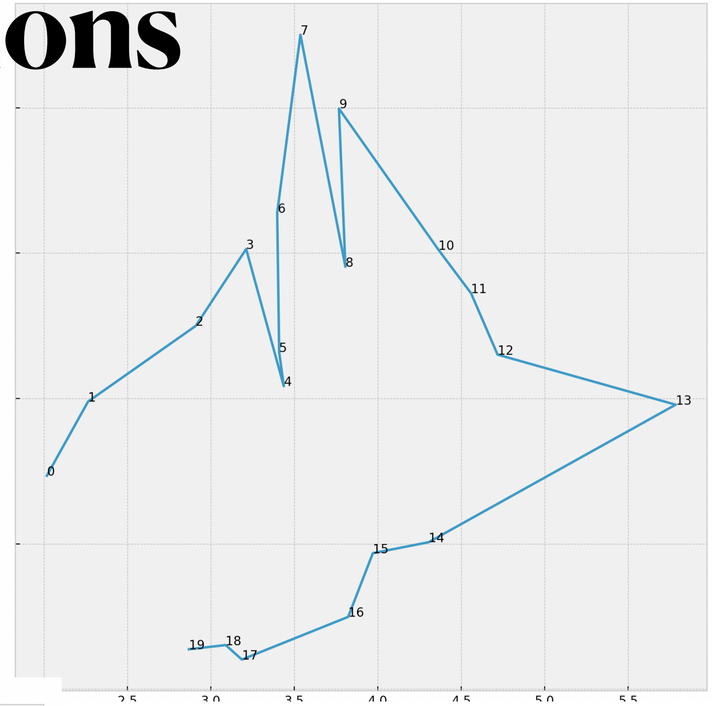
Triangulate in Linear Time

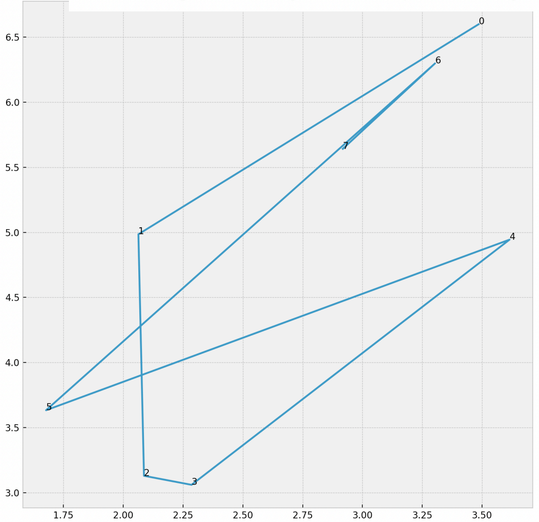
Convex Polygon Generations

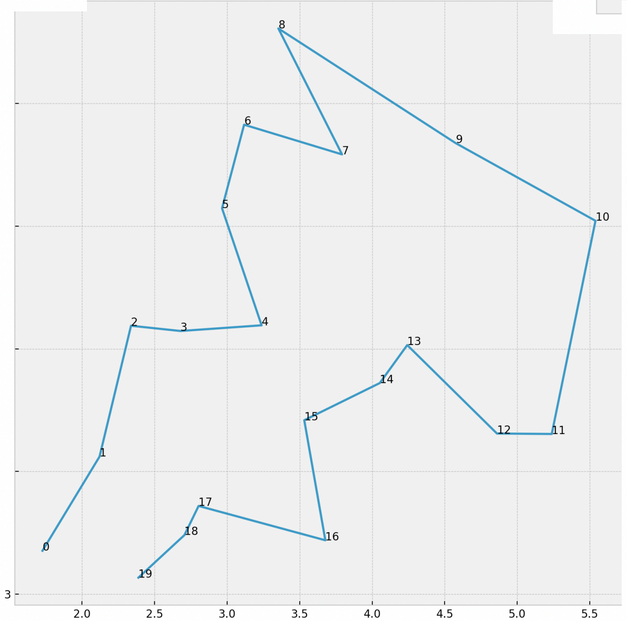


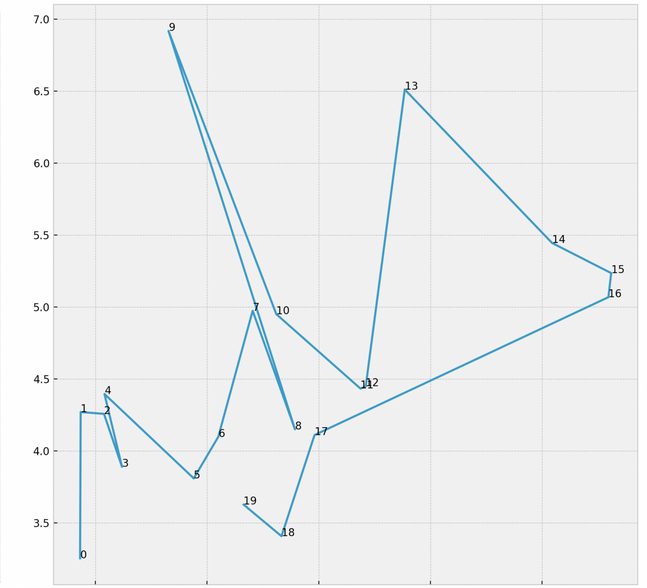


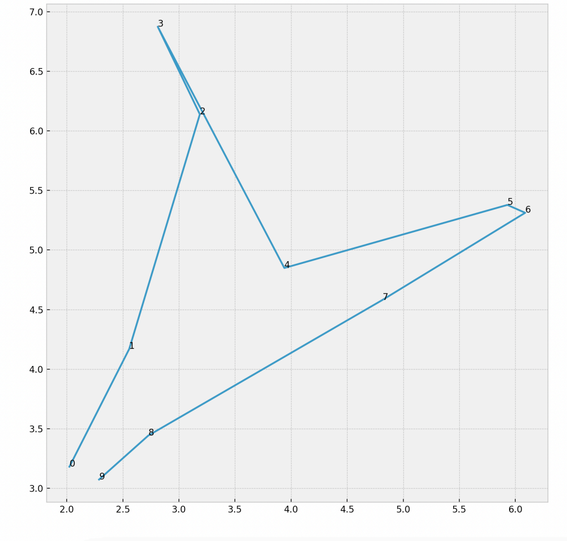












Proofs

Let X be an Indicator random variable defined the following way:

$$X_i = \begin{cases} 1 \; if \; n \; random \; points \; in \; a \; triangle \; are \; in \; convex \; position \\ 0 \; otherwise \end{cases}$$

$$P(X = 1) = \frac{2^n(3n - 3)!}{(n-1)!^3(2n)!}$$ (From the general proof for convex points in an n-gon)

$$\text{And, } P(X = 0) = 1 - \frac{2^n(3n - 3)!}{(n-1)!^3(2n)!}$$

The expectation of X will then be:

$$E[X_i] = 1.P[X_i = 1] + 0.P[X_i = 0]$$

$$= 1.\frac{2^n(3n - 3)!}{(n-1)!^3(2n)!} + 0.[1 - \frac{2^n(3n - 3)!}{(n-1)!^3(2n)!}]$$

$$= \frac{2^n(3n - 3)!}{(n-1)!^3(2n)!}$$

Proofs

$$E[X] = \sum_{i=1}^{m} E[X_i]$$

$$= m[\frac{2^n(3n\ -\ 3)!}{(n-1)!^3(2n)!}]$$

Applying Markov's inequality,

$$P(X \geqslant a) \leqslant E[X]/a$$

$$\leqslant \frac{2^n(3n\ -\ 3)!\ m}{(n-1)!^3(2n)!}/a$$

Proofs

Applying Chebyshev's inequality for a tighter bound,

$$\text{Var}[X_i] = E[X_i^2] - E[X_i]^2$$

$$= \left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right) - \left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right)^2$$

$$\text{Var}[X] = \sum_{i=1}^{m} \text{Var}[X_i] = E[X_i^2] - E[X_i]^2$$

$$= m\left[\left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right) - \left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right)^2\right]$$

$$P(X \geq a) = P(|X - E[X]| \geq a) \leq \text{Var}[X]/a^2$$

$$= m\left[\left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right) - \left(\frac{2^n(3n-3)!}{(n-1)!^3(2n)!}\right)^2\right] / a^2$$

Proofs

An illustration:

For 5 points, substituting n=5, we get:

$$\frac{2^5(3.5 - 3)!}{(5-1)!^3(2.5)!} \approx 0.306$$

Applying Markov's Inequality,

$$P(X \geqslant \frac{3m}{5}) \leqslant 0.51$$

Applying Chebyshev's inequality, we get

$$P(X - 0.306 \geqslant 0.494m) \leqslant Var[X] / (0.494m)^2$$
$$\leqslant 0.8702 /m$$

Benchmarking

- Bower-Watson algorithm implementation
- https://github.com/jmespadero/pyDelaunay2D
- Computes triangulations of less than 1000 points in a reasonable amount of time.
- Bower-Watson algorithm is an incremental algorithm that adds points one at a time to a valid DT of a subset of the points. It works in O(NlogN) average case and O(N^2) worst case.

Data

Experimental Setup

- 100 point sets with 10 points in each set
- 100 point sets with 50 points in each set
- 100 point sets with 100 points in each set
- 100 point sets with 500 points in each set

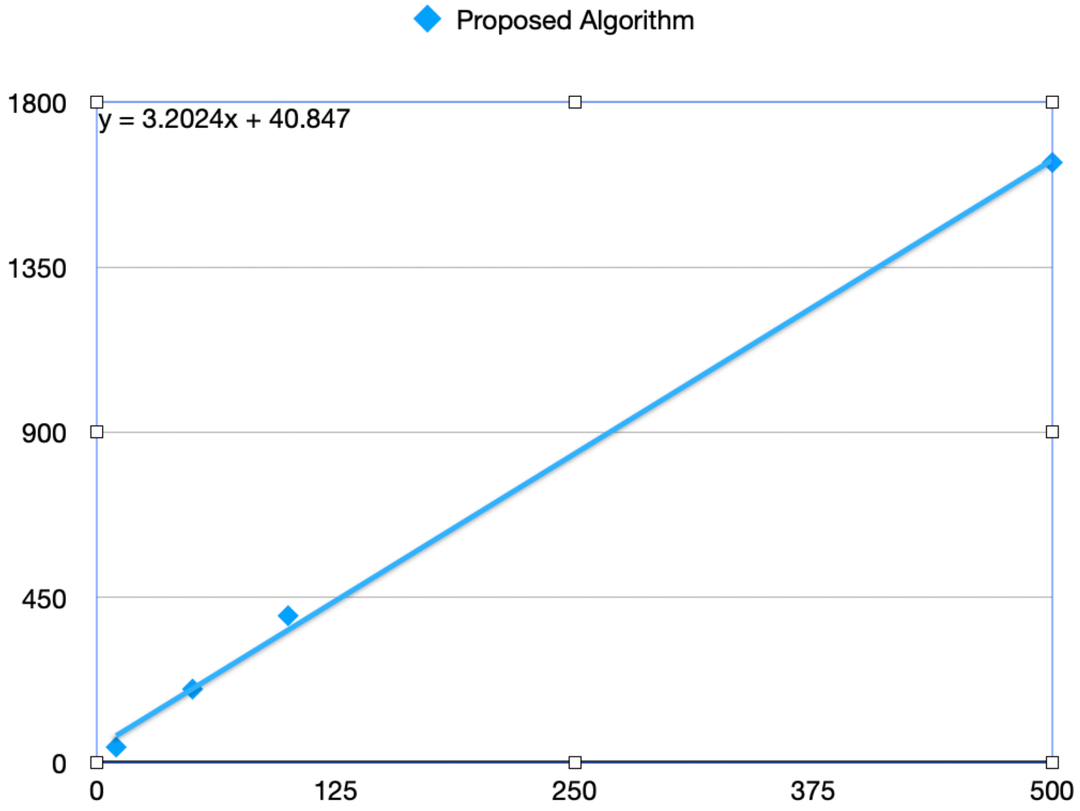
Data

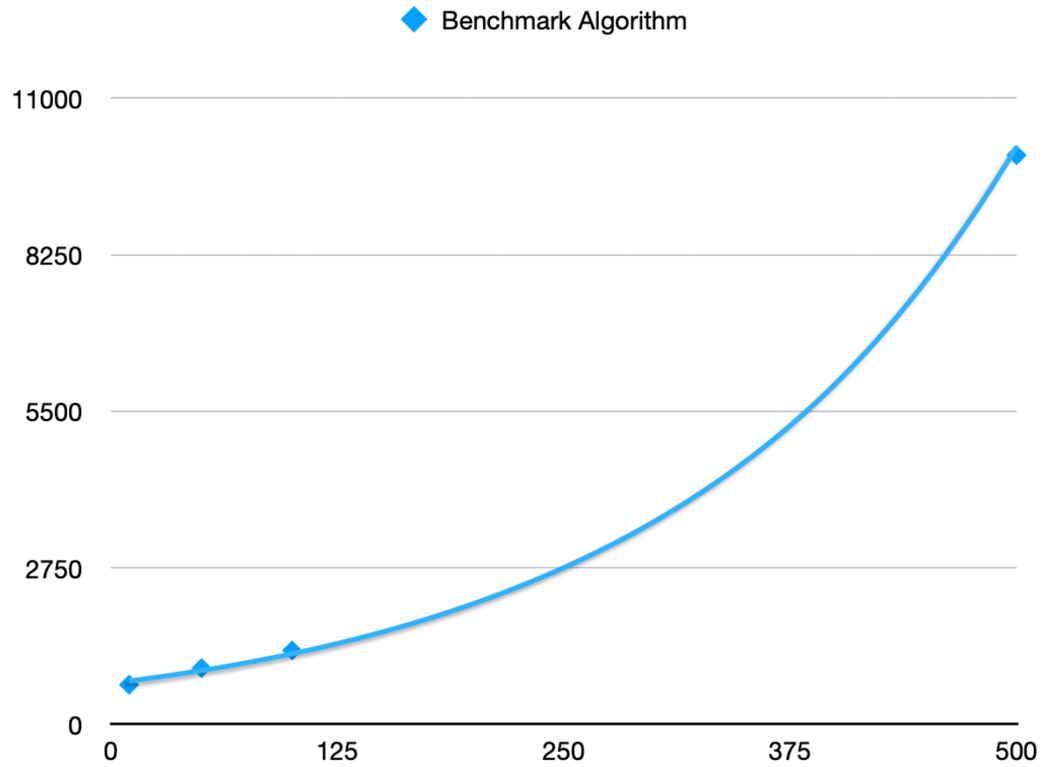
Experimental Setup

- Many real-time applications like collision avoidance, work with <500 data points.
- The benchmark algorithm also slows down significantly for > 1000 points. So comparing time taken for fewer than 1000 points.
- Changing number of points [10, 50, 100, 500] for better visualization and insight.
- Randomly generating the point sets in each run.
- Macbook, Big Sur (2.4 GHz 8-Core Intel Core i9, 32 GB). X point sets with Y points. Compared the algorithms w.r.t time, space and failure rate.
- Used probabilistic bounds to improve runtime.

Results

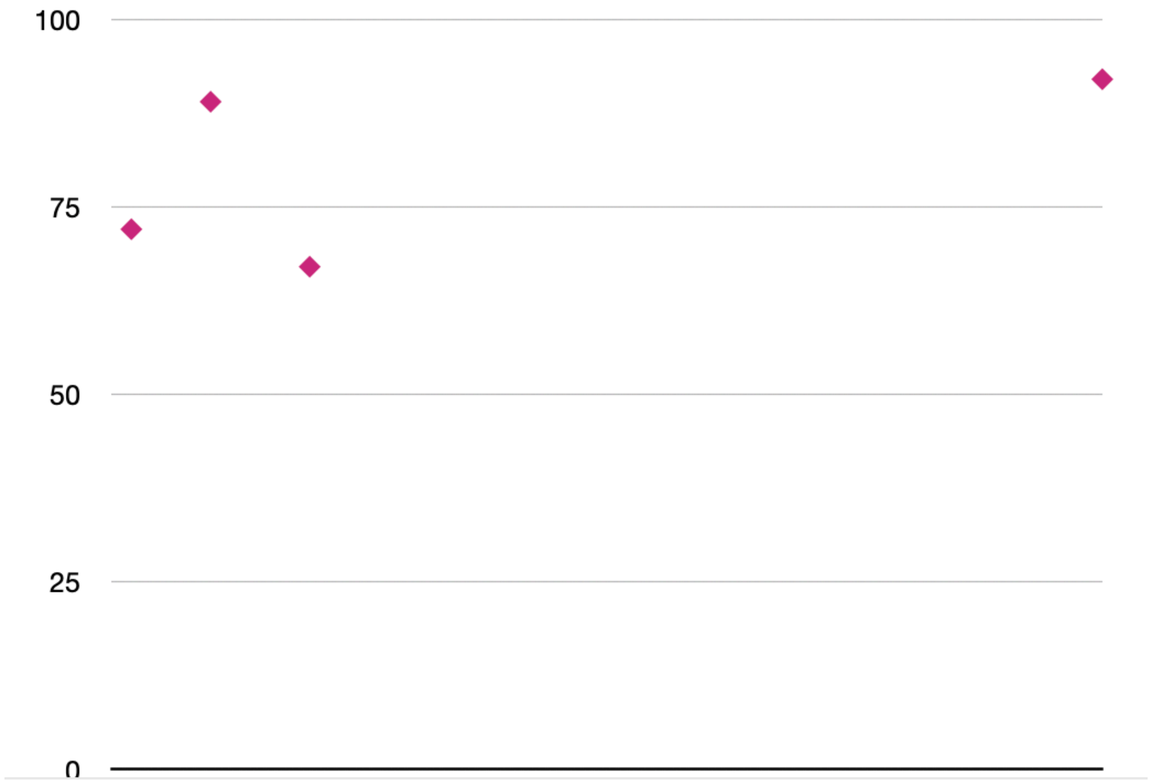
Runtime (y) vs Number of points (x)





Results

Number of Points (x) vs. Number of Failures (y)



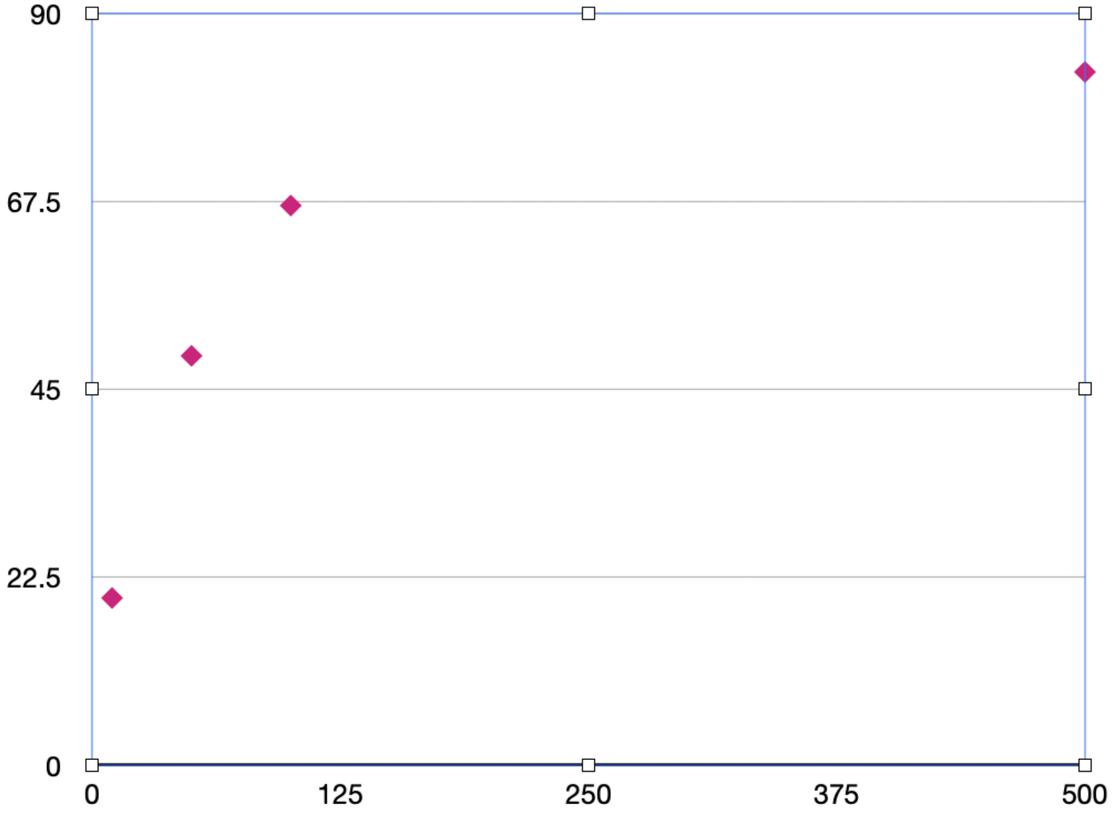
Number of points	Failures
10	72
50	89
100	67
500	92

Approaches to Improve Bound

- Run the experiment multiple times, the condition for convexity is hit at some point.
- Increase k in (n+k) sample points. That way, intuitively, you have a higher chance of finding a permutation that works.
- Pick unique n points out of n+k and check if convex until the condition is met.
- Find a range of values where the algorithm works the best.
- Run for points in clockwise order.

Improved Results

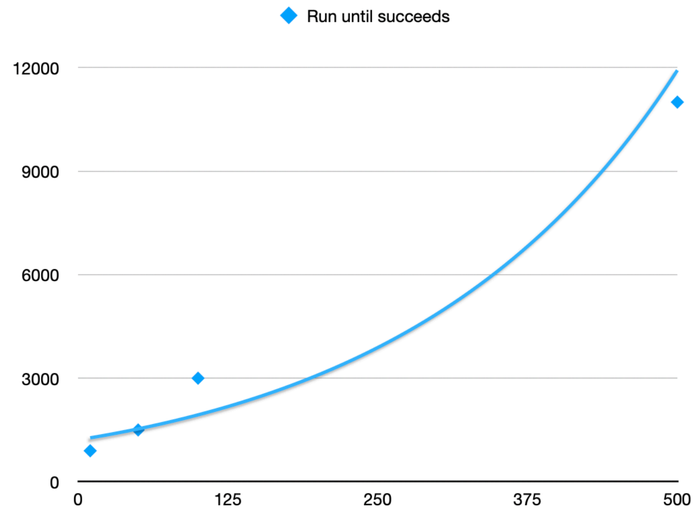
Number of Points vs Number of Failures (for 100 point sets) new



Number of points	Failures
10	20
50	49
100	67
500	83

Run Until Succeeds

- By running the algorithm until it succeeds (in the case of failure) and stopping when it succeeds.
- Recorded runtime and compared with benchmark.
- It still performs slightly better with 10, 50, and 500 points!!



Future Work

- So many more ways to tweak the algorithm to improve for increase in point size.
- Can be run in real time applications - simple code and no theoretical constructs.
- Handle edge cases - doesn’t work for some extreme cases.
- Extend linear time algorithm for a more general case of point sets.
- Try it on this robot.

References

- [1] https://upennig.weebly.com/uploads/7/4/0/3/74037187/erd%C3%B6s-szekeres.pdf
- [2] https://users.renyi.hu/~barany/cikkek/79.pdf
- [3] Boissonnat, JD., Devillers, O., Dutta, K. *et al.* Randomized Incremental Construction of Delaunay Triangulations of Nice Point Sets. *Discrete Comput Geom* **66,** 236–268 (2021). https://doi.org/10.1007/s00454-020-00235-7
- [4] Guibas, L.J., Knuth, D.E. & Sharir, M. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica* **7,** 381–413 (1992). https://doi.org/10.1007/BF01758770
- [5]https://www.cs.princeton.edu/~chazelle/pubs/polygon-triang.pdf

Thank You