CSE520: Computational Geometry
Lecture 17
Randomized Incremental Construction of the Delaunay Triangulation

Antoine Vigneron

Ulsan National Institute of Science and Technology

June 15, 2020

1. Introduction

2. Algorithm

3. Analysis

4. Conclusion

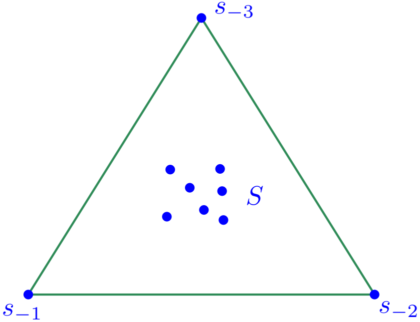
Outline

- Today, I will present an $O(n \log n)$ expected time algorithm for computing the Delaunay triangulation.
- It is a randomized incremental algorithm.
- Reference: Textbook Chapter 9.

Randomized Incremental Construction

Preliminary:

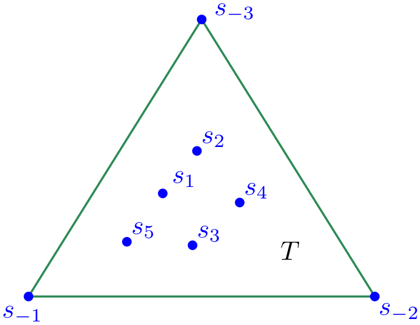
- Let $(s_1, s_2, s_3 \dots s_n)$ be a random permutation of S.
- Let $s_{-3}s_{-2}s_{-1}$ be a large triangle containing S.



- For each i, we denote $S_i = \{s_{-3}, s_{-2}, s_{-1}, s_1, s_2, \dots s_i\}$.

Randomized Incremental Construction

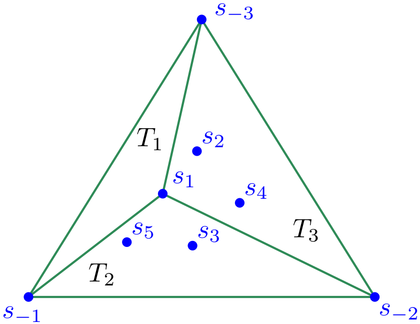
First step:



$$\mathcal{L}(T) = \{s_1, s_2, s_3, s_4, s_5\}$$

Randomized Incremental Construction

First step:



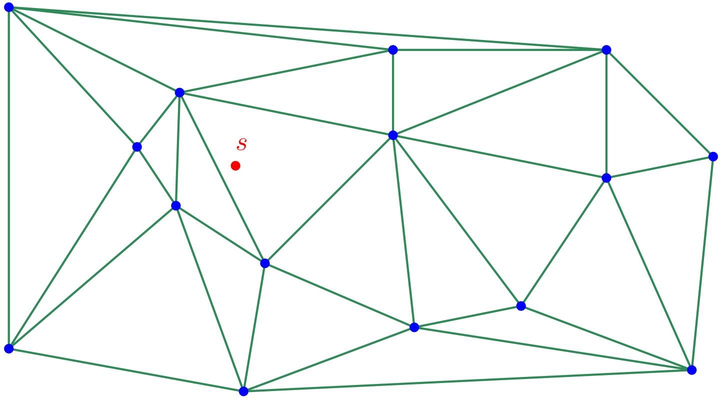
$\mathcal{L}(T_1) = \emptyset \qquad \mathcal{L}(T_2) = \{s_3, s_5\} \qquad \mathcal{L}(T_3) = \{s_2, s_4\}$

Randomized Incremental Construction

Idea:

- Insert s_1, then s_2 ... and finally s_n.
- Suppose we have computed $\mathcal{DT}(S_{i-1})$.
- The insertion of s_i splits a triangle into three
  - Find this triangle using conflict lists.
  - Each non inserted point has a pointer to the triangle in $\mathcal{DT}(S_{i-1})$ that contains it.
  - Each triangle T in $\mathcal{DT}(S_{i-1})$ is associated with the list $\mathcal{L}(T)$ of all the non-inserted points that it contains.
- Perform edge flips until no illegal edge remains.
  - We only need to perform flips around s_i.
  - On average, this step takes constant time.
- We have just computed $\mathcal{DT}(S_i)$.
- Repeat the process until $i = n$.

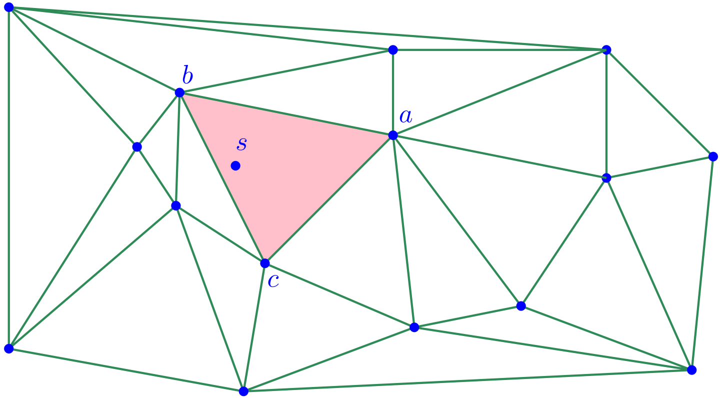
Example



inserting $s = s_i$

(We did not draw the outer triangle $s_{-1}s_{-2}s_{-3}$.)

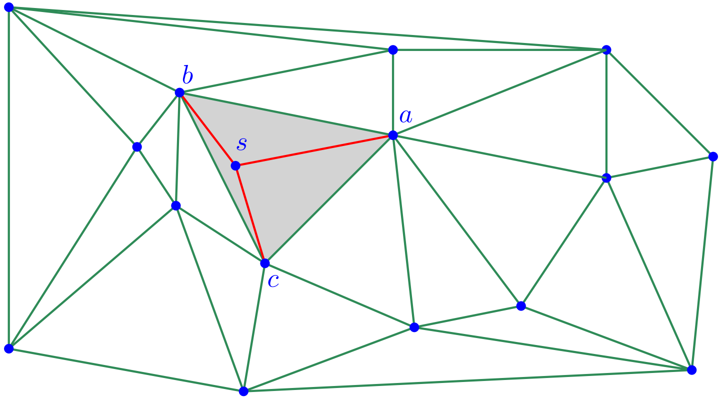
Example



find the triangle abc containing s

using the pointer stored in our data structure.

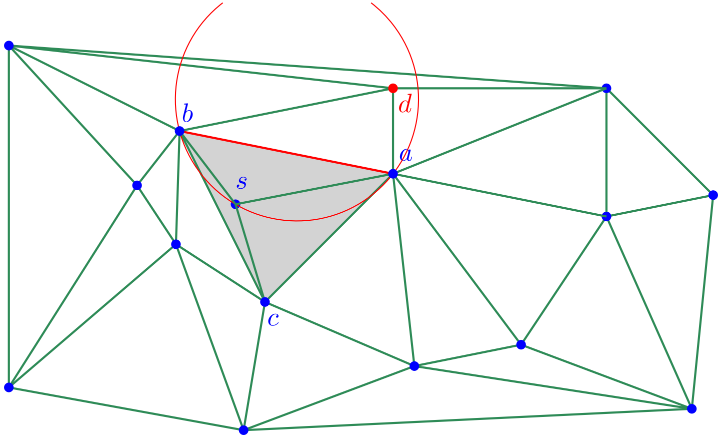
Example



insert edges sa, sb, sc

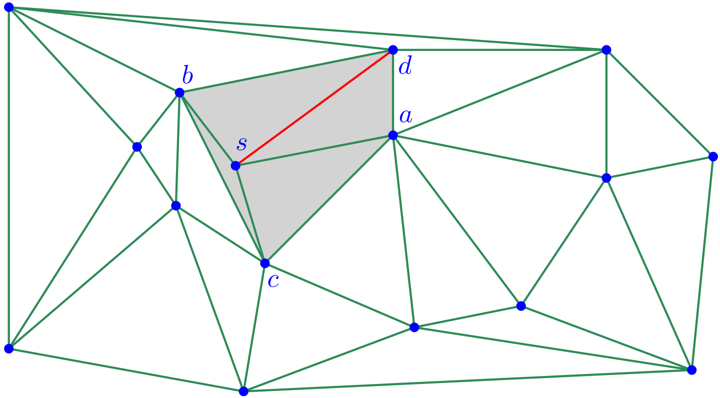
update the conflict lists

Example



edge ab is illegal

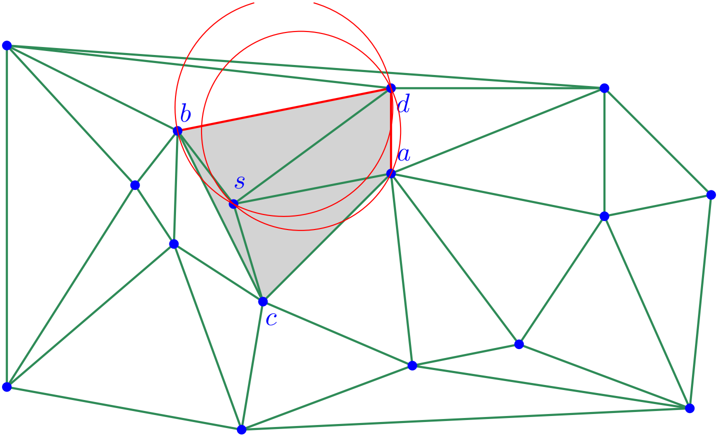
Example



flip the edge ab

update the conflict lists

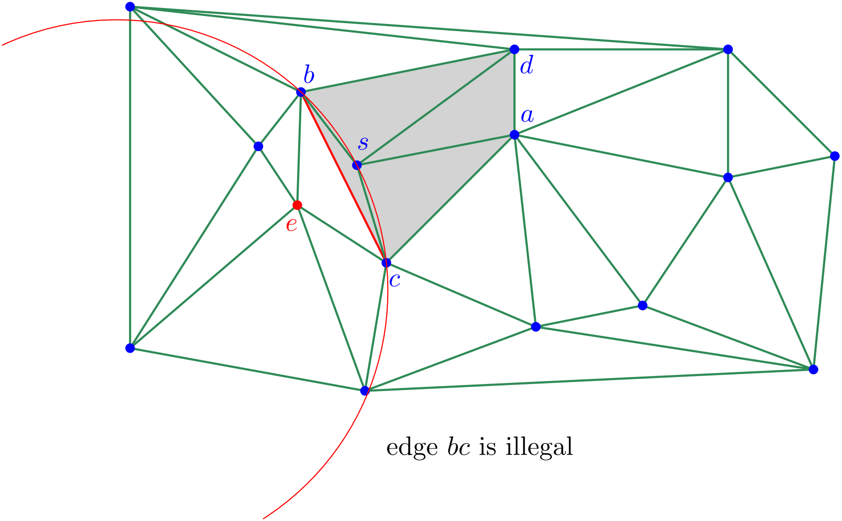
Example



Edges ad and bd are locally Delaunay.

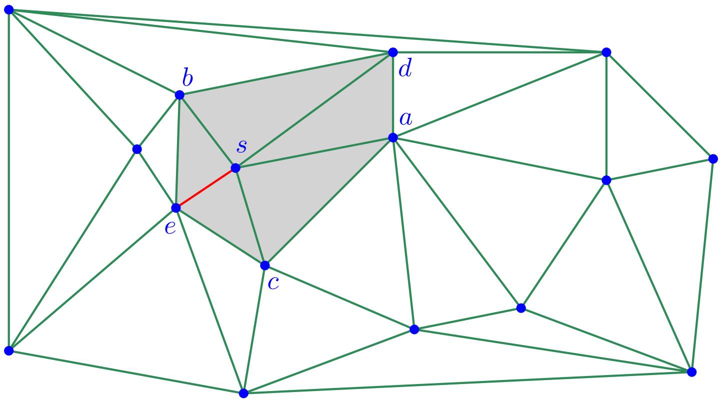
We keep them.

Example



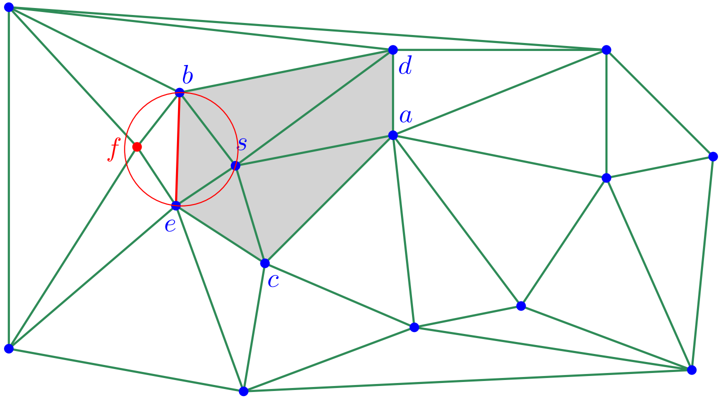
edge bc is illegal

Example



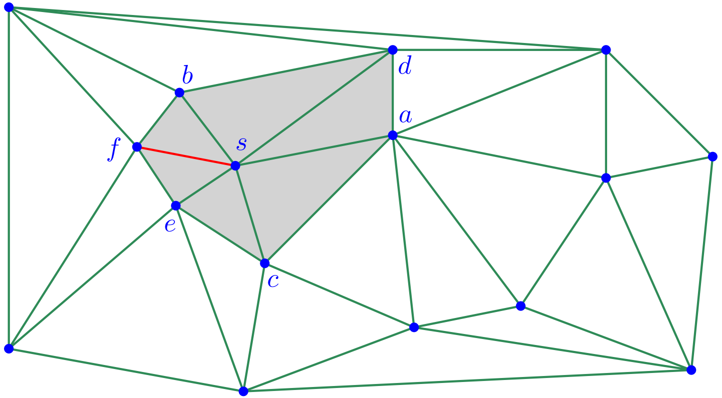
flip the edge bc

Example



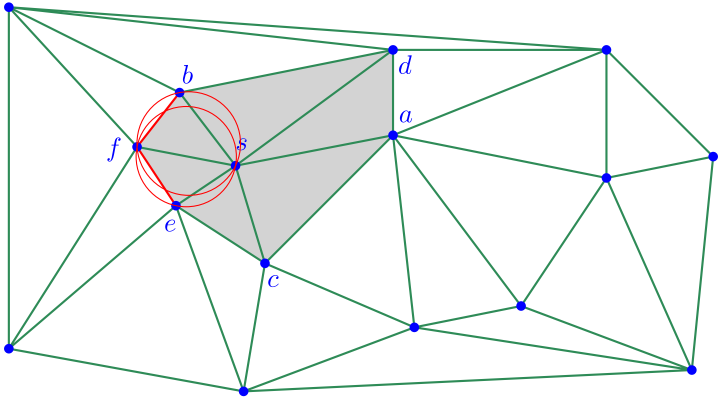
edge be is illegal

Example



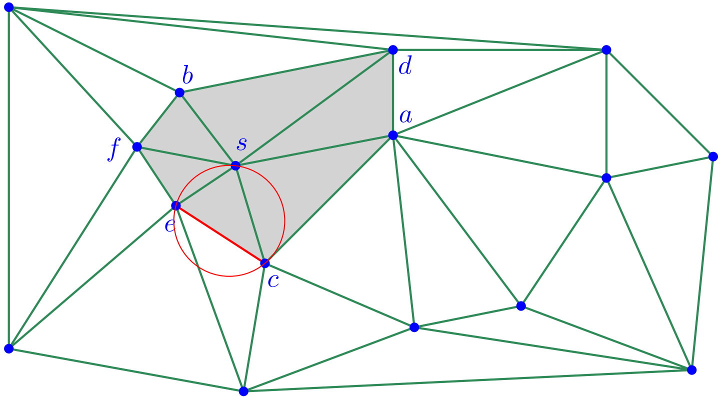
flip edge be

Example



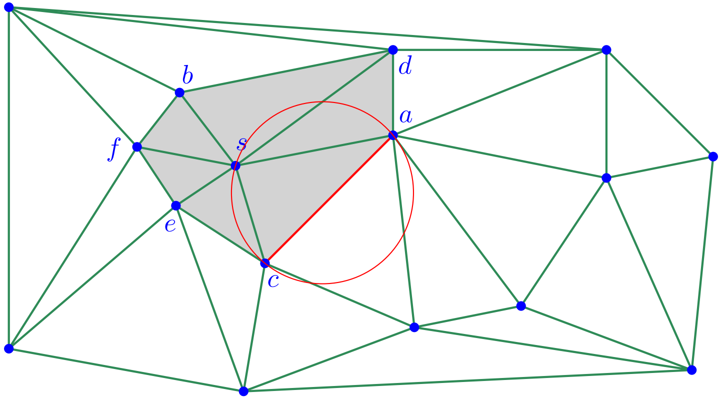
Edges be and bf are locally Delaunay.

Example



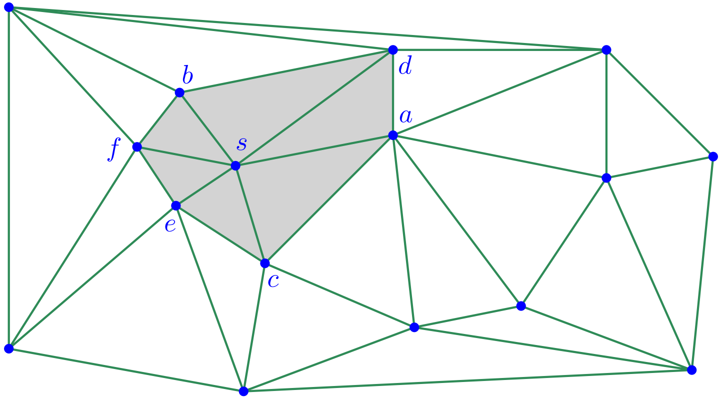
edge ec is locally Delaunay

Example



edge ac is locally Delaunay

Example



Delaunay triangulation after inserting s

The algorithm has only modified the shaded area.

Algorithm

- The algorithm checks all edges opposite to s in counterclockwise order. If an edge is illegal, it is flipped.
- We will first give the pseudocode, and then prove that this algorithm is correct.

Randomized Incremental Construction

Pseudocode

procedure $\textsc{Insert}(s)$
  find the triangle abc of $\mathcal{DT}(S)$ containing s
  ▷ use reverse pointers from conflict lists
  ▷ abc is chosen to be counterclockwise
  insert edges sa,sb and sc
  update conflict lists
  SwapTest(ab) ▷ see next slide
  SwapTest(bc)
  SwapTest(ca)

Randomized Incremental Construction

Pseudocode

```
procedure SwapTest(ab)
    if ab is an edge of the exterior face then
        return
    d ← the vertex on the other side of ab
    if inCircle(s, a, b, d) < 0 then
        flip edge ad for sd
        update the conflict lists
        SwapTest(ad)
        SwapTest(db)
```

Proof of Correctness

- We only flipped edges of triangles that contain s.
- Why is it sufficient?
- Remember the theorem: *locally* Delaunay implies (globally) Delaunay.
- Any edge between two triangles that do not contain s was locally Delaunay before insertion of s.
- So it is still locally Delaunay.
- Thus the triangulation we obtain is the Delaunay triangulation.

Analysis

- Consider the time t_i taken to update the current triangulation while inserting s_i.
- It does not account for conflict lists updates.
- Each new edge (obtained by splitting abc or a flip) is incident to s_i.
- So t_i is proportional to the degree of s_i in $\mathcal{DT}(S_i)$.

Analysis

- We use backward analysis: S_i is fixed, s_i is random.
- Each edge has two endpoints.
- So each edge of $\mathcal{DT}(S_i)$ that is not an edge of the outer triangle is incident to s_i with probability at most $\frac{2}{i}$.
    - It is $\frac{1}{i}$ when one endpoint is s_{-1}, s_{-2} or s_{-3}
- There are $3i$ edges in $\mathcal{DT}(S_i)$ that are not edges of the outer triangle. Proof: We create 3 more edge when we insert a point s_i.
- So by backward analysis, the expected degree of s_i is at most

$$\frac{2}{i} \cdot 3i = 6.$$

- It means that $E[t_i] = O(1)$.
- Thus the expected time for updating the triangulation is $O(n)$ over the whole construction of $\mathcal{DT}(S)$.

Analysis

- We now bound the time needed to update the conflict lists.
- We say that a site is rebucketted if its conflicting triangle changes, that is, if it lies inside a newly created face of $\mathcal{DT}(S_i)$.
- While inserting s_i, what is the probability that $s \in S \setminus S_i$ is rebucketted?
- Backward analysis:
- It is the probability that $s_i \in \{a, b, c\}$, when abc is the triangle of $\mathcal{DT}(P_i)$ that contains s.
- This probability is at most $3/i$ because s_i is chosen at random from $\{s_1, \dots, s_i\}$.
- So when we insert s_i, we rebucket at most $3n/i$ sites on average.

Analysis

- Problem: a site may be rebucketted several times at step i.
- Intuition: On average, we only perform a constant number of flips at each step, so this issue only accounts for a constant factor in the running time. (Detailed proof in textbook.)
- So overall, rebucketting takes expected time

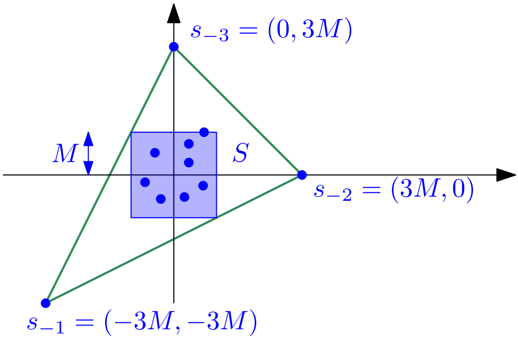
$$O\left(\sum_{i=1}^{n} \frac{n}{i}\right) = O(n \log n).$$

- Remark: It is the same as what we obtained for computing the trapezoidal map.

Theorem

The randomized incremental construction of the Delaunay triangulation takes $O(n \log n)$ time.

How to choose the points s_{-1}, s_{-2} and s_{-3}



- M: Maximum of any coordinate of any point in S.
- For incircle test, do as if these three points are outside any circle defined by three points in S.

Concluding Remarks

- The Delaunay triangulation of n points can be computed in expected time $O(n \log n)$.
- It holds for worst case input, the expectation is over the random choices made by the algorithm.
- It can also be done in $O(n \log n)$ deterministic time.
  - Not covered in this course.
  - Less practical; the RIC is used in practice.
- Knowing the Delaunay triangulation of S, we can find the Voronoi diagram of S in $O(n)$ time.
  - How?
- It is optimal, as we observe that there is an $\Omega(n \log n)$ lower bound for computing the Delaunay triangulation and the Voronoi diagram.

Concluding Remarks

- Combined with the point location data structure of Lecture 9, we can answer proximity queries in the plane (see Lecture 11) in
  - $O(\log n)$ expected query time,
  - $O(n \log n)$ expected preprocessing time, and
  - $O(n)$ expected space usage.
- Using this algorithm, we can also compute 2D closest pairs and EMST in $O(n \log n)$ expected time.