Insertions and Deletions in Delaunay Triangulations using Guided Point Location
Construct Delaunay triangulation of convex polygon in linear expected time \[\text{[Chew 90]}\]
Guide - Example

Construct Delaunay triangulation of convex polygon in linear expected time [Chew 90]

Can we use guides for the Delaunay triangulation of a point set?
Can we use guides for deleting a point in a 3D DT?
Incremental Construction using Guides
  • con BRIO
  • Brio with dependent choices

Deletion in 3d Delaunay triangulations using Guides
  • guided randomized reinsertions
  • the Star Delaunay triangulation
Delaunay triangulations

- Point set $S$
- Delaunay triangulation $DT(S)$
- Empty-sphere property
Incremental construction of DTs

- Insertion cost
  - Point location: $O(\log n)$ using search structure
  - Structural update: $O(\log n)$ for walking but $O(\log n)$ using search structure
Incremental construction of DTs

- Insertion cost
  - Point location: $O(\log n)$ using search structure
  - Structural update: $O(C(p))$ 2D: $O(1)$ expected for random point

guides to avoid the $O(\log n)$ overhead?
• random order
  – prevents too many triangles/simplices
  – does not allow for guides

• local insertion strategies?

• *Biased Randomized Insertion Orders*: allows local strategies, enough randomness
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
- in every round
  - determine short tour through points: space-filling curve tour
• assign points to random rounds of halving size
• in every round
  – determine short tour through points: space-filling curve tour
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
- in every round
  - determine short tour through points: space-filling curve tour
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
- in every round
  - determine short tour through points: space-filling curve tour
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
- in every round
  - determine short tour through points: space-filling curve tour
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

• assign points to random rounds of halving size
• in every round
  – determine short tour through points: space-filling curve tour
Incremental Constructions con Brio [Amenta, Choi, Rote 03]

- assign points to random rounds of halving size
- in every round
  - determine short tour through points: space-filling curve tour
  - traverse DT of previous round along tour & insert
con Brio: known properties

- not worse than randomized incremental construction when used with point location data structure (no guides!) [ACR 03]

- expected linear time (after bucketing) for points uniform in cube ($\mathbb{R}^d$), and for normal distribution (space-filling curves as guides) [B 07]

- $O(n \log n)$ for points with integer coordinates in range $\{1, \ldots, n^s\}$ for constant $s$ [B 09]

- Is this optimal for integer coordinates?
Outline

Incremental Construction using Guides
  • con BRIO
  • Brio with dependent choices

Deletion in 3d Delaunay triangulations using Guides
  • guided randomized reinsertions
  • the Star Delaunay triangulation
Given \( n \) points in the plane, it takes \( \Omega(n \log n) \) time to compute their Delaunay triangulation.
Real RAM and word RAM

BUT: The lower bound is proved by a reduction from sorting and applies to the real RAM.

The real RAM allows operations on arbitrary real numbers.

Actual computers can handle only finite precision, but they allow bit-manipulation in constant time. This is captured by the word RAM.
The word RAM

On a word RAM data is represented as a sequence of $w$-bit words ($w > \log n$).

Input consists of integers in $[0, 2^w - 1]$.

We can perform operations such as $+, -, *, /$, bitwise and, or, xor in constant time.

Constant time table look-up is available.
Previous work – integer sorting

There is a long history of sorting algorithms for the word RAM (integer sorting).

Highlights include radix sort, van Emde Boas trees [vEB77], fusion trees [FW90], signature sort [AHNR95], …

The current champions are a deterministic algorithm by Han [H02] ($O(n \log \log n)$) and a randomized algorithm by Han and Thorup [HT02] ($O(n (\log \log n)^{1/2})$).
Previous work – computational geometry on a word RAM

There are many transdichotomous results for orthogonal problems, for example...

...rectangle intersection [KO88]...

...orthogonal range searching [W92]...

...point location in orthogonal subdivisions [dBvKS95]...

...$l_1$-Voronoi Diagrams [CF97]...

...and more [IL00],[AEIS01].
Only recently did Chan and Pătrașcu obtain results for nonorthogonal problems [C06], [P06], [CP07].

By giving transdichotomous algorithms for the slab-problem, they obtain fast algorithms for many geometric problems, such as:

- planar point location in time $O\left(\frac{\log n}{\log \log n}\right)$
- planar Voronoi diagrams and 3d convex hulls in time $n^{2O\left(\frac{\log \log n}{\sqrt{\log \log n}}\right)}$.

Can the running times be improved?
Is the approach through point location necessary?
Planar Delaunay triangulations can be computed in time $O(\text{sort}(n))$ on a word RAM that supports the shuffle operation in constant time.

Analogous results hold in higher dimensions, for well-behaved point sets.

BrioDC has many other consequences:

- Sorting helps for Delaunay triangulations (somewhat).
- We can preprocess a given point set [set of regions] so that Delaunay triangulations of subsets [a point per region] can be computed faster.
- We can preprocess a given convex point set in 3d so that convex hulls of subsets can be computed faster.
BrioDC: Overview

- point set
- compressed quadtree
- nearest-neighbor graph
- Delaunay triangulation
- $O(sort(n))$
- $O(n)$
- shuffle sorting [C08, BET99]
- WSPD [CK95]
- BrioDC
- nearest-neighbor graph
- BrioDC
- Delaunay triangulation
The shuffle operation

We assume our word RAM supports the shuffle operation in constant time.

Performing the shuffle operation on a point set and sorting it gives the Z-order or Morton-order of the set [M66].
The shuffle order and quadtrees

The shuffle order is closely related to quadtrees.

**Theorem [BET99][C08]:** If a point set is sorted according to the shuffle order, we can find a compressed quadtree for it in $O(n)$ time on a word RAM.
Quadtrees and nearest-neighbor graphs

Theorem [CK95]: Given a compressed quadtree for a point set $P$, we can find its nearest-neighbor graph in time $O(n)$ using the well-separated pair decomposition for $P$. 
**Theorem**: If the nearest-neighbor graph for any planar point set $Q$ can be found in time $f(|Q|)$, such that $f(|Q|)/|Q|$ is nondecreasing, the Delaunay triangulation of a planar point set $P$ can be found in expected time $O(f(n)+n)$. 
Nearest-neighbor graphs and Delaunay triangulations

**Theorem:** If the nearest-neighbor graph for any planar point set $Q$ can be found in time $f(|Q|)$, such that $f(|Q|)/|Q|$ is nondecreasing, the Delaunay triangulation of a planar point set $P$ can be found in expected time $O(f(n)+n)$.

**Proof:** We use a randomized incremental construction with biased insertion order and dependent sampling.

Given $P$.

Find $\text{NNG}(P)$.

Pick an edge in each component.

Sample one point from each edge, sample the rest independently.
**Theorem:** If the nearest-neighbor graph for any planar point set $Q$ can be found in time $f(|Q|)$, such that $f(|Q|)/|Q|$ is nondecreasing, the Delaunay triangulation of a planar point set $P$ can be found in expected time $O(f(n)+n)$.

**Proof:** We use a randomized incremental construction with biased insertion order and dependent sampling.

Recurse on the sample.

Insert the remaining points: walk along the edges of $\text{NNG}(P)$ and insert the points along the way.
Theorem: If the nearest-neighbor graph for any planar point set \( Q \) can be found in time \( f(|Q|) \), such that \( f(|Q|)/|Q| \) is nondecreasing, the Delaunay triangulation of a planar point set \( P \) can be found in expected time \( O(f(n)+n) \).

Proof: We use a randomized incremental construction with biased insertion order and dependent sampling.

Recurse on the sample.

Insert the remaining points: walk along the edges of \( \text{NNG}(P) \) and insert the points along the way.
Extensions and variants

The reduction does not need any bit-manipulation and also works on a real RAM.

If $P$ is sorted in $x$- and $y$-direction, we can find a quadtree for it with an algebraic computation tree of depth $O(n)$, so $DT(P)$ can be found by an ACT of depth $O(n)$.

Fun fact: If $P$ is sorted only in $x$-direction, there is a $\Omega(n \log n)$ lower bound [DL95]. Similarly for 3d convex hulls [S84] (when sorted in any $O(1)$ directions).
Potential of BrioDC?

Can we find a faster algorithm for 3d convex hulls?

Are there other applications of BrioDC?
Running Time (per point)

The graph shows the running time in microseconds for different input sizes. The lines represent different algorithms:

- **BrioDC (linear)**
- **BrioDC (n log n)**
- **Triangle Incr.**
- **Triangle D&C**

The x-axis represents the input size n, while the y-axis shows the time in microseconds. The graph illustrates how each algorithm's performance changes with increasing input size.
More space-filling curve type orders
Delaunay triangulation - Deletions

- Insert point $q$
  - Find simplices in conflict
  - Connect boundary to $q$

- Delete point $q$
  - Find incident simplices
  - Retriangulate cavity
Outline

Incremental Construction using Guides
  • con BRIO
  • Brio with dependent choices

Deletion in 3d Delaunay triangulations using Guides
  • guided randomized reinsertions
  • the Star Delaunay triangulation

[Schrijvers 12]
[Schrijvers, B, Devillers, Mulzer, Shewchuk]
## Related work

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (2D)</th>
<th>2D?</th>
<th>3D?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary completion</td>
<td>$O(d^2)$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Flipping</td>
<td>$O(d^2)$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ear queue</td>
<td>$O(d \log d)$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Low degree optimization</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Triangulate and sew</td>
<td>$O(d \log d)$</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Guided randomized reinsertion</td>
<td>$O(d)$</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

$d = \text{degree of deleted vertex}$
Triangulate and Sew

We improve this part

1. Use boundary info
2. Only create interior

Incident simplices
Incident vertices
“Outer” simplices
“Inner” simplices
Analysis of Triangulate and Sew

Running time
$O(d \log d + C(P))$

Point location
$O(d \log d)$

Use boundary for point location
$O(d)$

Structural cost
$C(P)$

Only create interior
$C^*(P)$

$d = \text{degree of deleted vertex}$
Approach

- Aim at 3D implementation
  - 2D implementation is already fast (insert ~5.8 ms, delete ~2.5 ms)
  - 3D implementation slow (insert ~25 ms, delete ~106 ms)

- Combine 3D “triangulate and sew” with “guided reinsertion”

- **CGAL 3D deletions**
  1. Delete cells incident to q
  2. Create the complete DT of incident vertices from scratch
  3. Match edges and sew triangles

- Use the current incidence information to jumpstart point location

- Build only the triangles on the inside of the cavity
Algorithm Overview

- **Goal**: Use the current incidence information to jumpstart point location

- Deconstruct cavity triangulation point-per-point
  - Store incidence information

- Small enough → create simplex

- Reconstruct using incidence information
Algorithm Overview – Example

Delete $p_5$
Store $p_1$ as guide

Delete $p_4$
Store $p_2$ as guide

Create simplex

Insert $p_5$
Use $p_1$ as guide

Insert $p_4$
Use $p_2$ as guide
“So, to delete 1 vertex, you delete all incident vertices first. How can this be fast(er)?”

Answer:
Use lower-dimensional deletions.
Surface Delaunay triangulation DT-(Q_i)

- Every simplex has q as vertex
- Incident vertices form lower-dimensional Delaunay-like triangulation

- Perform Delaunay deletion algorithm on the surface only
Surface Delaunay triangulation $DT^{-}(Q_i)$
Surface Delaunay triangulation $\text{DT}^{-}(Q_i)$
In 3D, the surface is a 2-dimensional triangulation.

In 3D:

**Theorem.** For a vertex $p_i$ that is random or has constant expected degree, \texttt{STARDTDELETE} runs in $O(1)$ time.

Point location time is dependent on sampling
- We return to that later
Reminder: Approach

- **CGAL 3D deletions**
  1. Delete cells incident to q
  2. Create the *complete DT* of incident vertices *from scratch*
  3. Match edges and sew triangles

- Use the current adjacency information to jumpstart point location

- Build only the triangles on the inside of the cavity
Star Delaunay triangulation DT*(P_i)

Generate DT for incident vertices
Star Delaunay triangulation

- **Approach:**
  - Change local conflict definition
  - Run normal insertion algorithm

- **Difficulties:**
  - Handle when q is on the Convex Hull
  - Correctness proof requires large case distinction
Reminder: Approach

- **CGAL 3D deletions**
  1. Delete cells incident to q
  2. Create the *complete DT* of incident vertices *from scratch*
  3. Match edges and sew triangles

- Use the current adjacency information to jumpstart point location

- Build only the triangles on the inside of the cavity

- Design algorithm for sampling / selecting guiding points
An $O(C^*(P))$-time algorithm

- Sample point uniformly at random, triangle as guide + hashing

- **Delete**($P_i$, DT$^-$($P_i \cup q$), q)
  1. If $|P_i| = 4$, create DT$^-$($P_i$), triangle hashtable directly $O(1)$
  2. Sample random point $p_i$ $O(1)$
  3. Remove from DT$^-$($P_i \cup q$) to get DT$^-$($P_{i-1} \cup q$) $O(1)$
  4. Pick newly created triangle $t$ from DT$^-$($P_{i-1} \cup q$) $O(1)$
  5. DT$^*$($P_{i-1}$) $\leftarrow$ **Delete**($P_{i-1}$, DT$^-$($P_{i-1} \cup q$), q) $T(d-1)$
  6. Insert $p_i$ in DT$^*$($P_{i-1}$) using $t$ $O(C^*(p_i))$
  7. Store new boundary triangles in hash table $O(1)$

- So $T(d) = O(1 + C^*(p_i)) + T(d-1) = O(d + C^*(P)) = O(C^*(P))$. 
Recapitulation

- Strong theoretical result
  - Improved point location time
  - Improved structural complexity

- What about practice?
  - Experimentation
Implementations

- Guided Randomized Reinsertion
  - Sample random point (deg < 8), save neighborhood, take lowest degree neighbor
  - Sample random edge (deg < 16)
  - Sample random point (deg < 8), triangle as guide + hashing

- Brio
  - Bounded-degree spanning tree

- CGAL Triangulate and Sew
  - Points in order
  - Randomized

- Misc
  - Only edge as guide, no Star DT
  - Only Star DT, no guide
Experimental Setup

• Distributions
  - 3D instances of 2D synthetic distributions
  - Stanford surface data
  - Moment curve
  - Helix

• Measures
  - Running time
  - Geometric tests
    - 2D and 3D
  - Number of simplices
    - Created / destroyed
Running time

The graph shows the running time in milliseconds for different degrees of a system. Different algorithms are compared, including GRR - Edge, GRR - Triangle Hashing, GRR - Neighborhood, BRIO, Guide Only (Edge), Star DT Only, CGAL, and CGAL Randomized. The x-axis represents the degree, while the y-axis shows the running time in milliseconds. The graph indicates an increase in running time as the degree increases for all algorithms.
C(P) vs. C*(P) – low degree
C(P) vs. C*(P) – high degree
Guide Running Time

The graph illustrates the running time in microseconds for different degrees of surface models. The x-axis represents the degree range, and the y-axis shows the time in microseconds. The graph compares running times with and without the use of a guide for various tasks, including surface deletion, surface construction, and retriangulation. The data indicates a clear trend of increased running time as the degree increases, with distinct bars and lines representing different scenarios. The legend clarifies the color codes used for each type of task and with or without the guide.
Conclusion

- BRIO is powerful and fast tool to allow guides

- BrioDC is even more powerful; faster implementation? more applications?

- Guides help also for 3d deletions (for high degree) analysis for low-degree sampling strategies?

Thanks to Frits van Bommel, Olivier Devillers, Wolfgang Mulzer, Jonathan Shewshuk and Okke Schrijvers
Thank You