Parallel d-D Delaunay Triangulations in Shared and Distributed Memory

Daniel Funke and Peter Sanders

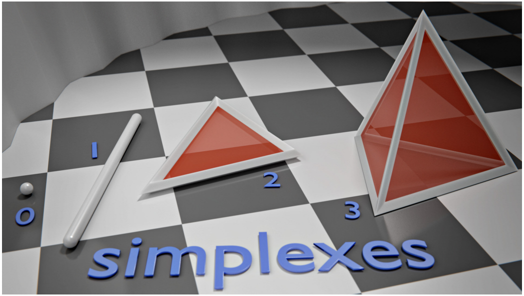
Jared Di Carlo

March 14, 2019

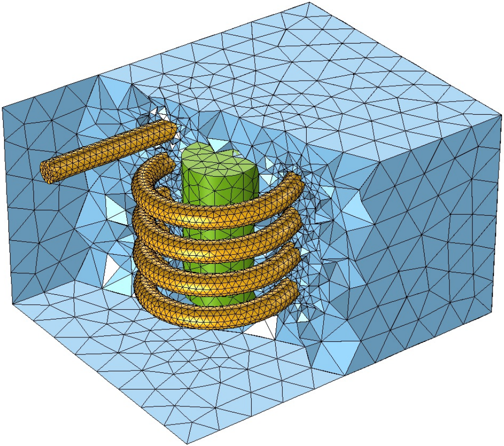
Triangulation in Arbitrary Dimensions

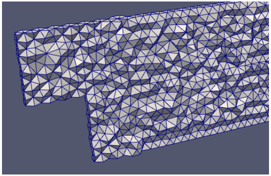
Definition

A *k-simplex* is a k-dimensional generalization of a triangle.



3D Triangulation

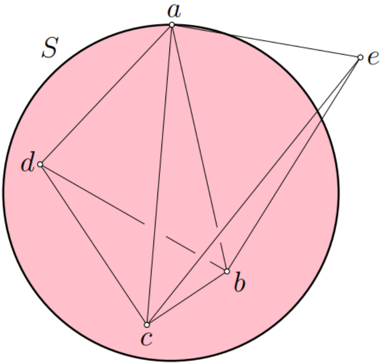


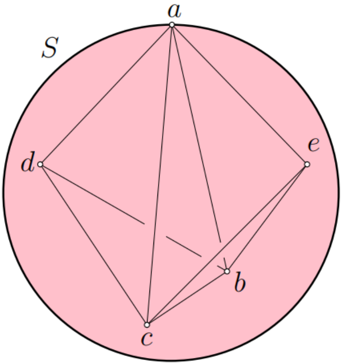


Delaunay Triangulation in Arbitrary Dimensions

Definition

In $\mathbb{R}^n$, given a set S of points, the *Delaunay Triangulation* of S is a triangulation of S such that the circumsphere of any simplex in $\mathcal{D}(S)$ does not contain any other point in S.





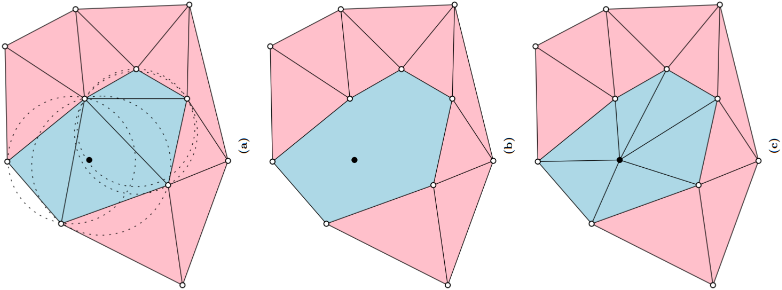
Delaunay Triangulation in Arbitrary Dimensions

- In $\mathbb{R}^2$, Delaunay maximizes the minimum angle
- In $\mathbb{R}^3$, it doesn't
- In $\mathbb{R}^n$, it minimizes the maximum *containment sphere*
- Number of tetrahedra in 3D Delaunay tetrahedralization
  - $O(n)$: uniform volume
  - $O(n \log n)$: uniform surface
  - $O(n^2)$: worst case (two lines)

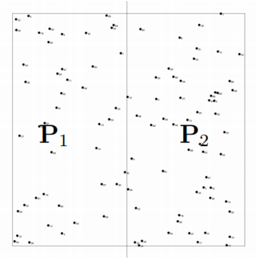
Sequential 3D Delaunay

- Randomized insertion
- Flipping
- Watson Method

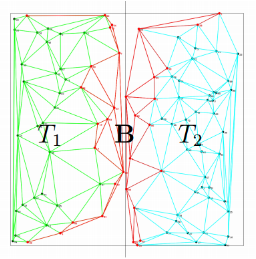
Watson Method



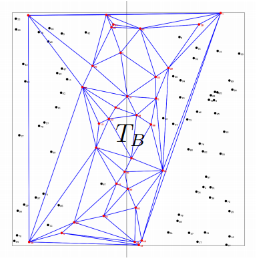
Strategy



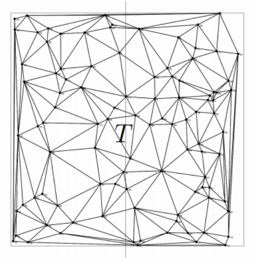
(a) partitioning



(b) partial DTs



(c) border DT



(d) final DT

Part 1: Split

$$
\begin{array}{l}
\textbf{if } n < N \vee r = \log P \textbf{ then} \\
\quad \textbf{return } \textit{sequentialDelaunay}(\mathbf{P}) \\
k \leftarrow \text{splittingDimension}(\mathbf{P}) \\
\begin{pmatrix} \mathbf{P}_1 & \mathbf{P}_2 \end{pmatrix} = \begin{pmatrix} p_1 & \cdots & p_s & | & p_{s+1} & \cdots & p_n \end{pmatrix} \leftarrow \text{divide}(\mathbf{P}, k) \\
\mathbf{T} = \begin{pmatrix} T_1 & T_2 \end{pmatrix} \leftarrow \begin{pmatrix} \text{Delaunay}(\mathbf{P}_1, r+1) & \text{Delaunay}(\mathbf{P}_2, r+1) \end{pmatrix}
\end{array}
$$

- Constant
- Alternating
- Largest

Part 2: Finding Border Triangles

$\mathbf{B} \leftarrow \varnothing$; $\mathbf{Q} \leftarrow \mathrm{convexHull}(T_1) \cup \mathrm{convexHull}(T_2)$ ▷ initialize

parfor $s_{i,x} \in \mathbf{Q}$ **do** ▷ si

  $\mathrm{mark}(s_{i,x})$

  if $\mathrm{circumsphere}(s_{i,x}) \cap \mathrm{boundingBox}(T_j) \neq \varnothing$, with $i \neq j$ **then**

    $\mathbf{B} \mathrel{\cup=} \{s_{i,x}\}$ ▷ circumsphere intersects oth

    for $s_{i,y} \in \mathrm{neighbors}(s_{i,x}) \wedge \neg\,\mathrm{marked}(s_{i,y})$ **do**

      $\mathbf{Q} \mathrel{\cup=} s_{i,y}$; $\mathrm{mark}(s_{i,y})$

$T_B \leftarrow \mathrm{Delaunay}(\mathrm{vertices}(\mathbf{B}), r+1)$

Part 3: Merge

$T \leftarrow (T_1 \cup T_2) \setminus \mathbf{B}; \quad \mathbf{Q} \leftarrow \varnothing$

parfor $s_b \in T_B$ **do**

  if $\mathrm{vertices}(s_b) \not\subset \mathbf{P}_1 \wedge \mathrm{vertices}(s_b) \not\subset \mathbf{P}_2$ **then**

    $T \cup= \{s_b\}; \quad Q \cup= \{s_b\}$

  else

    if $\exists s \in \mathbf{B} : \mathrm{vertices}(s) = \mathrm{vertices}(s_b)$ **then**

      $T \cup= \{s_b\}; \quad Q \cup= \{s_b\}$

Part 4: Update Data Structures

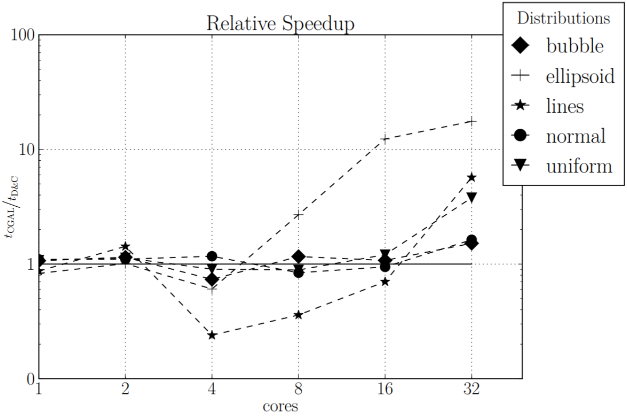
Neighborhood update:

parfor $s_x \in Q$ **do**
    for $d \in \{1, \ldots, D+1\}$ **do**
        if $\text{neighbors}_d(s_x) \notin T$ **then**
            $C \leftarrow \{s_c : f_d(s_x) = f_d(s_c)\}$
            for $s_c \in C$ **do**
                if $|\text{vertices}(s_x) \cap \text{vertices}(s_c)| = D$ **then**
                    $\text{neighbors}_d(s_x) \leftarrow s_c; \quad Q \cup= s_c$
return T

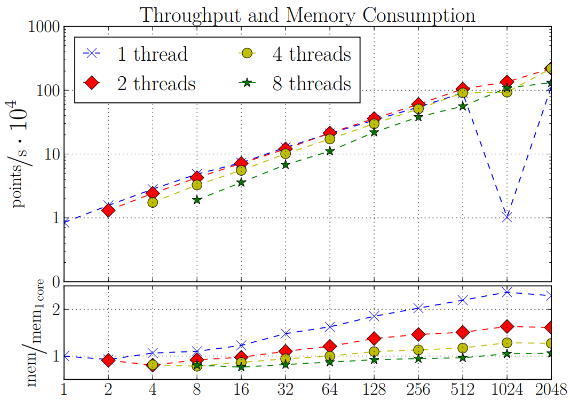
Data Structures

- Check for duplicates
  - Hash table of simplices
- Update Neighbors
  - Face hash table
  - Build during Part 2
- Merge Large Blocks
  - Binary Search Tree
  - $O(\log k)$ access after k merges

Results


Relative Speedup
100
10
1
0
$t_{CGAL}/t_{D\&C}$
1
2
4
8
16
32
cores
Distributions
bubble
ellipsoid
lines
normal
uniform

Results


Throughput and Memory Consumption
1 thread
2 threads
4 threads
8 threads
points/s · 10^4
mem/mem_1 core
1000
100
10
1
0
2
1
1
2
4
8
16
32
64
128
256
512
1024
2048

Comparison Of Results

- Tested on 1 million uniformly distributed points (same dataset)
- Different output size depending on # of threads?
- Volume is wrong
- Intel E5-2699V3 (2014, 2.30 GHz, 18 cores), 32 GB RAM (nobody used this much)
- Very hard to find program (author has deleted published GitHub page)

	1 Core	16 Core
This Paper	233 s	42 s
CGAL	8 s	n/a
Tetgen	5 s	n/a

Questions

- Results?
  - Paper: 50,000,000 points in 64 seconds on 32 cores (24k pts/(sec cpu))
  - My result: 1,000,000 points in 42 seconds on 16 cores (1.4k pts/(sec cpu))
- Do we actually need Delaunay?
  - Delaunay algorithms are $O(n \log n)$
  - Can get a "good" tetredralization in $O(n)$ time...