Triangulation

Jean-François Remacle[1] and Christophe Geuzaine[2]

[1] Université catholique de Louvain (UCLouvain)

[2] Université de Liège (ULiege)

http://www.gmsh.info

October 7, 2024

Triangulations

A *simplex* is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions.

A triangulation $T(S)$ of the n points $S = \{p_1, \ldots, p_n\} \in \mathbb{R}^d$ is a set of non overlapping simplices that covers exactly the convex hull $\Omega(S)$ of the point set, and leaves no point p_i isolated.

Points p_j are *in general position* when they do not fall on subvarieties of lower degree than necessary; in the plane two points should not be coincident, three points should not fall on a line, four points should not fall on a circle.

Triangulation

There exist a finite but combinatorial number of triangulations (Catalan numbers) for a given set of points. In dimension 2, the number of triangles is constant for every triangulation of the same set of points. This is not true in 3D and in higher dimensions.

The Delaunay triangulation is a special triangulation that exist and is unique if points are in general position.

There exist algorithms to generate the Delaunay triangulation in $(\mathcal{O})n\log(n)$ complexity! Yet, the constant grows rapidely with d.

Delaunay triangulation

The Delaunay triangulation DT(S) of a point set S has the fundamental geometrical property that the circumsphere of any tetrahedron is empty.

If the empty empty sphere condition is verified for all tetrahedra, the triangulation $T(S)$ is said to be a Delaunay triangulation.

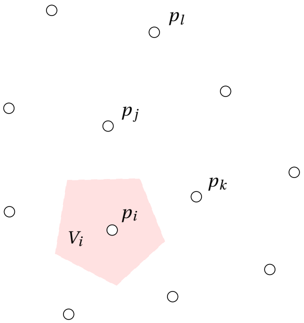
In dimension 2, DT(S) has interresting properties.

The Voronoï Diagram

Definition: Consider a finite set $S = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ of n distinct points in the plane. The *Voronoi cell* V_i of $p_i \in S$ is the set of points x that are closer to p_i than to any other points of the set:

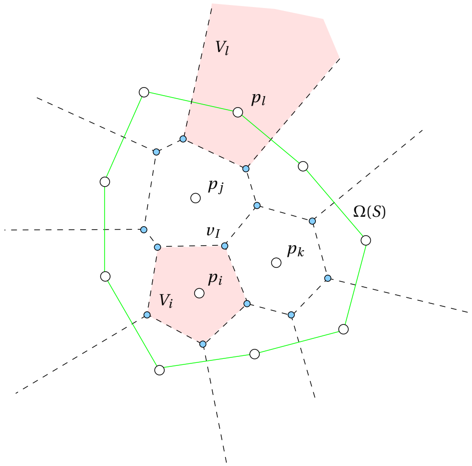
$$V_i = \left\{x \in \mathbb{R}^2 \mid \|x - p_i\| < \|x - p_j\|,\ \forall 1 \leq i \leq n, i \neq j\right\}$$

where $\|x - y\|$ is the euclidian distance between x and y.



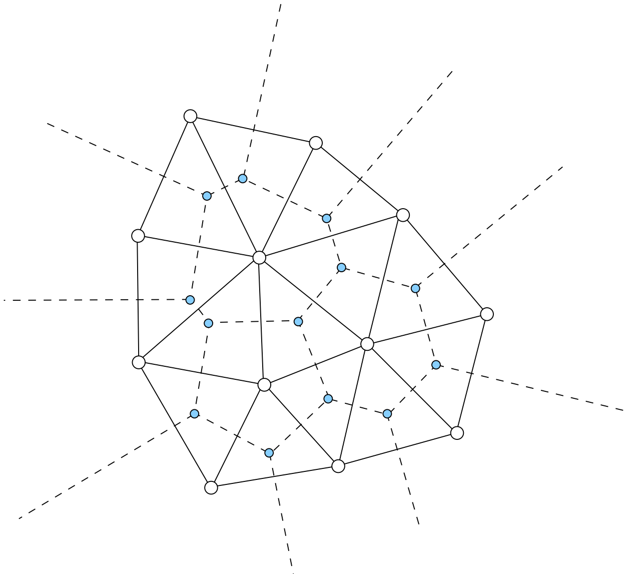
The Voronoï Diagram

The Voronoi diagram $V(S)$ is the unique subdivision of the plane into n cells. Its is the union of all Voronoi cells V_p:



The Delaunay triangulation

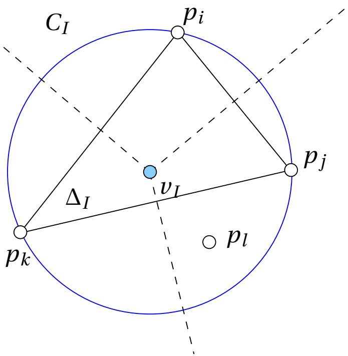
The Delaunay triangulation $\mathrm{DT}(S)$ is the geometric dual of the Voronoï diagram



The empty circle property

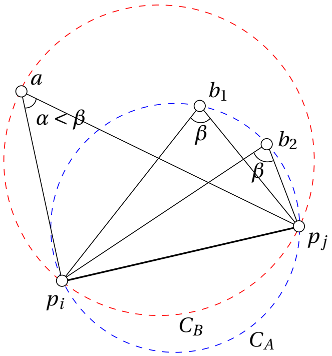
The circumcircle of any triangle in the Delaunay triangulation is empty i.e. it contains no point of S.

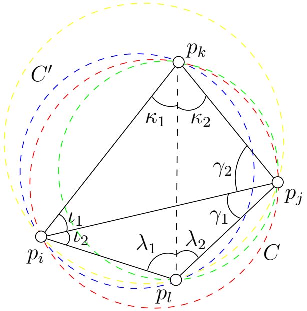
- Consider the Delaunay triangle $\Delta_I = p_i p_j p_k$. Assume now that point $p_l \in C_I$ where C_I is the circumcircle of Δ_I.
- By definition, the triple point v_I is at equal distance to p_i, p_j and p_k and no other points of S are closer to v_I than those three points.
- Then, if a point like p_l exist in S, v_I is not a triple point and triangle Δ_I cannot be a Delaunay triangle.



The MaxMin property

The Delaunay triangulation DT(S) is angle-optimal: it maximizes the minimum angle among all possible triangulations.



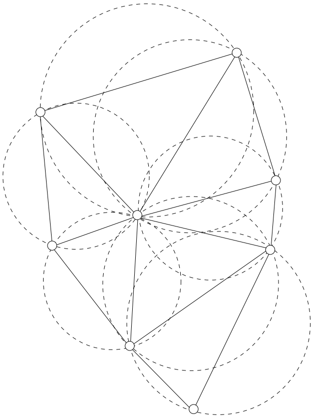


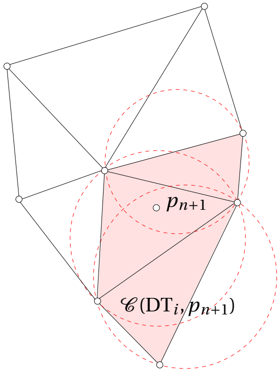
Thales theorem (left) and MaxMin property illustrated (right)

Bowyer-Watson Algorithm

Let DT_n be the Delaunay triangulation of a point set $S_n = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ that are in general position. We describe an incremental process allowing the insertion of a given point $p_{n+1} \in \Omega(S_n)$ into DT_n and to build the Delaunay triangulation DT_{n+1} of $S_{n+1} = \{p_1, \ldots, p_n, p_{n+1}\}$.

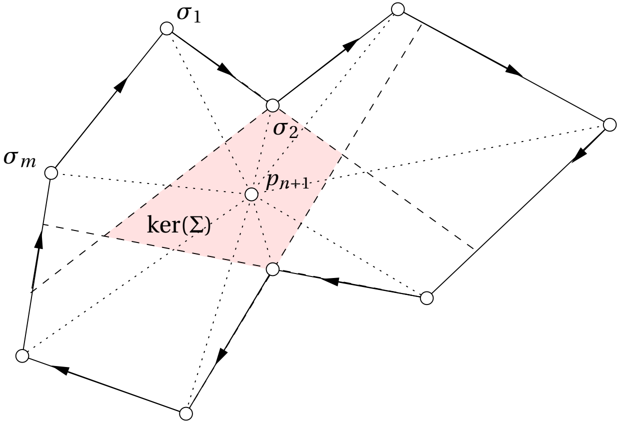
$$\mathsf{DT}_{n+1} = \mathsf{DT}_n - \mathcal{C}(\mathsf{DT}_n, p_{n+1}) + \mathcal{B}(\mathsf{DT}_n, p_{n+1}). \qquad (1)$$





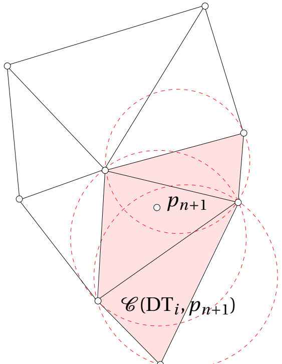
Bowyer-Watson Algorithm

Consider a polygon Σ with m corners $\sigma_1, \ldots, \sigma_m$ that is bounded by m edges $\sigma_i, \sigma_{(i+1)\%m}$, $1 \leq i \leq m$.
The kernel $\ker(\Sigma)$ is the set of point $x \in \mathbb{R}^2$ that are visible to every σ_j i.e. the line segment $x\sigma_j$ them do not intersect any edges of the polygon.
The kernel $\ker(\Sigma)$ can be computed by intersection of the halfplanes that correspond to all oriented edges of the polygon (see Figure).

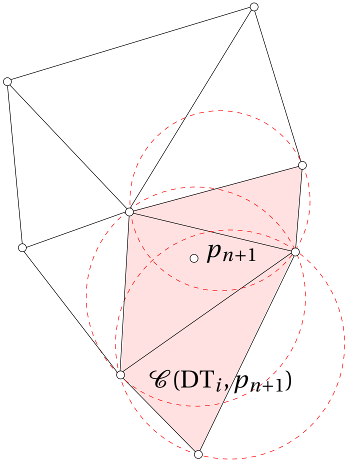


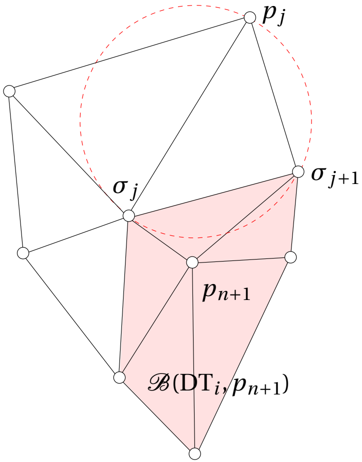
Bowyer-Watson Algorithm

The Delaunay cavity $\mathcal{C}(T_n, p_{n+1})$ is the set of m triangles $\Delta_1, \ldots, \Delta_m \in \mathrm{DT}_n$ for which their circumcircle contains p_{n+1}.
The Delaunay cavity contains the set of triangles that cannot belong to T_{n+1}. The region covered by those invalid triangles should be emptied and re-triangulated in a Delaunay fashion. The Delaunay cavity has some interresting properties.
Theorem: The Delaunay cavity $\mathcal{C}(T_n, p_{n+1})$ is a non empty connected set of triangles which the union form a star shaped polygon with p_{n+1} in its kernel.



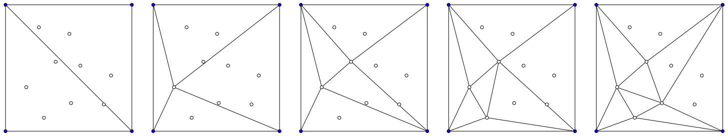
Bowyer-Watson Algorithm

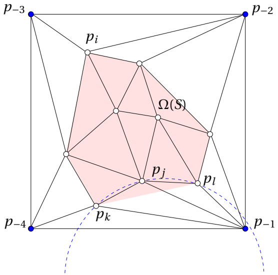




Bowyer-Watson Algorithm

Super triangles :

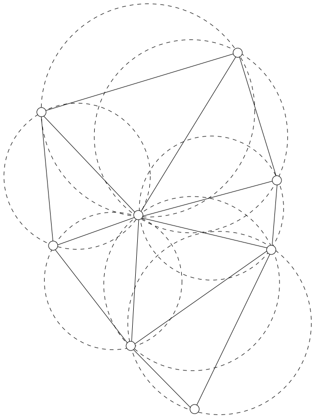


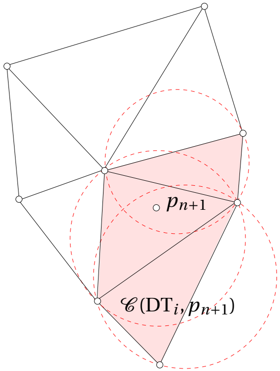


Bowyer-Watson Algorithm

Let DT_n be the Delaunay triangulation of a point set $S_n = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ that are in general position. We describe an incremental process allowing the insertion of a given point $p_{n+1} \in \Omega(S_n)$ into DT_n and to build the Delaunay triangulation DT_{n+1} of $S_{n+1} = \{p_1, \ldots, p_n, p_{n+1}\}$.

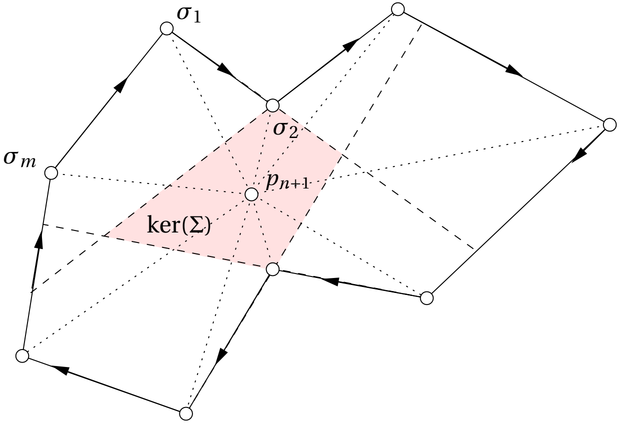
$$\mathsf{DT}_{n+1} = \mathsf{DT}_n - \mathcal{C}(\mathsf{DT}_n, p_{n+1}) + \mathcal{B}(\mathsf{DT}_n, p_{n+1}). \tag{2}$$





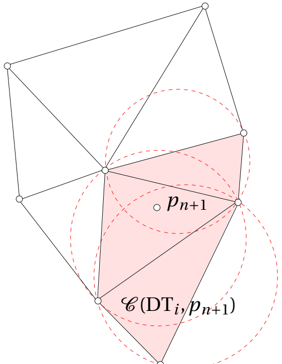
Bowyer-Watson Algorithm

Consider a polygon Σ with m corners $\sigma_1, \ldots, \sigma_m$ that is bounded by m edges $\sigma_i, \sigma_{(i+1)\%m}$, $1 \leq i \leq m$.
The kernel $\ker(\Sigma)$ is the set of point $x \in \mathbb{R}^2$ that are visible to every σ_j i.e. the line segment $x\sigma_j$ them do not intersect any edges of the polygon.
The kernel $\ker(\Sigma)$ can be computed by intersection of the halfplanes that correspond to all oriented edges of the polygon (see Figure).

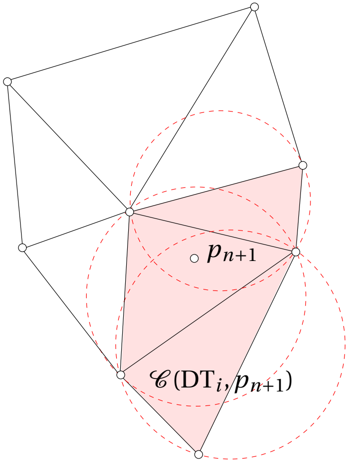


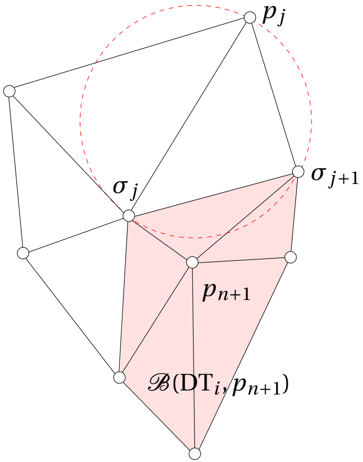
Bowyer-Watson Algorithm

The Delaunay cavity $\mathcal{C}(T_n, p_{n+1})$ is the set of m triangles $\Delta_1, \ldots, \Delta_m \in \mathsf{DT}_n$ for which their circumcircle contains p_{n+1}.
The Delaunay cavity contains the set of triangles that cannot belong to T_{n+1}. The region covered by those invalid triangles should be emptied and re-triangulated in a Delaunay fashion. The Delaunay cavity has some interresting properties.
Theorem: The Delaunay cavity $\mathcal{C}(T_n, p_{n+1})$ is a non empty connected set of triangles which the union form a star shaped polygon with p_{n+1} in its kernel.



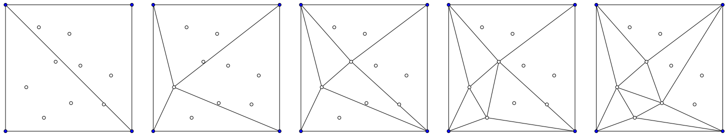
Bowyer-Watson Algorithm

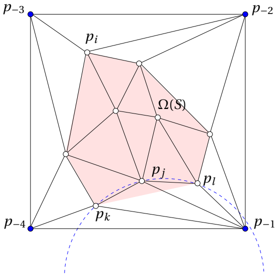




Bowyer-Watson Algorithm

Super triangles :

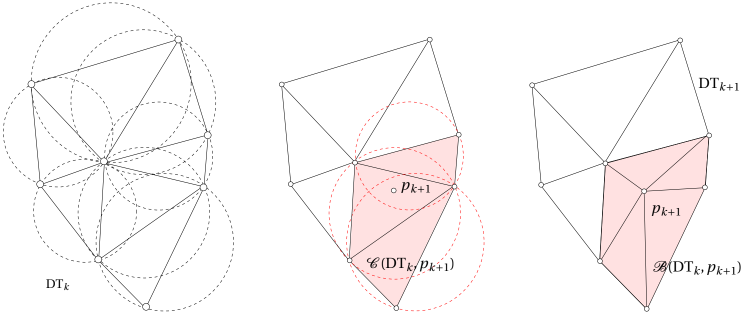




DT of n points in $n\log(n)$ complexity

- Use Bowyer-Watson algorithm (not the best choice in 2D)

$$\mathsf{DT}_{k+1} = \mathsf{DT}_k - \mathcal{C}(\mathsf{DT}_k, p_{k+1}) + \mathcal{B}(\mathsf{DT}_k, p_{k+1})$$

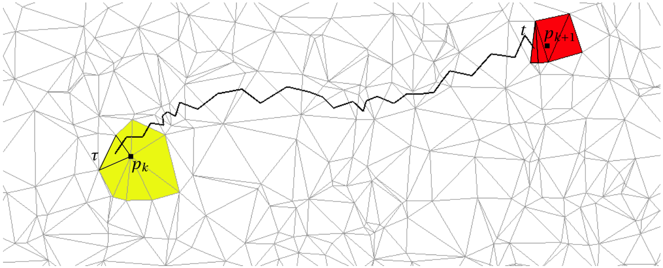


DT_k
p_{k+1}
$\mathscr{C}(\mathrm{DT}_k, p_{k+1})$
DT_{k+1}
p_{k+1}
$\mathscr{B}(\mathrm{DT}_k, p_{k+1})$

DT of n points in $n\log(n)$ complexity

- Use Bowyer-Watson algorithm (not the best choice in 2D)
- **Sort the points**, N. Amenta, S. Choi, and G. Rote. *Incremental constructions con brio.*, 2003.

Without sort: $\mathcal{O}(n^{1/d})$ "walking" steps per insertion → overall (best) complexity of $\mathcal{O}(n^{1+\frac{1}{d}})$



DT of n points in $n\log(n)$ complexity

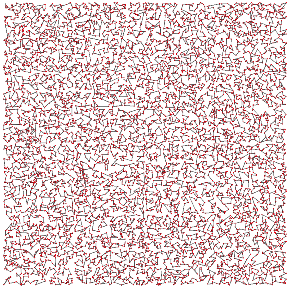
- Use Bowyer-Watson algorithm (not the best choice in 2D)
- **Sort the points**, N. Amenta, S. Choi, and G. Rote. *Incremental constructions con brio.*, 2003.
- **Robust predicates with static filters**, H. Si. *Tetgen, a delaunay-based quality tetrahedral mesh generator.*, 2015.

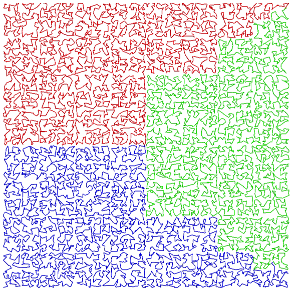
n	10^3	10^4	10^5	10^6	10^3	10^4	10^5	10^6
	2D				3D			
N_{walk}	23	73	230	727	17	38	85	186
t(sec)	$3.6\ 10^{-3}$	$9.1\ 10^{-2}$	3.98	187	$1.2\ 10^{-2}$	$1.8\ 10^{-1}$	3.42	73
	2D (BRIO)				3D (BRIO)			
N_{walk}	2.3	2.4	2.5	2.5	2.9	3.0	3.1	3.1
t(sec)	$2\ 10^{-3}$	$1.5\ 10^{-2}$	$1.5\ 10^{-1}$	1.47	$9.0\ 10^{-3}$	$7.5\ 10^{-2}$	$7.8\ 10^{-1}$	7.81

DT of n points in $n\log(n)$ complexity

- Use Bowyer-Watson algorithm (not the best choice in 2D)
- **Sort the points**, N. Amenta, S. Choi, and G. Rote. *Incremental constructions con brio.*, 2003.
- **Robust predicates with static filters**, H. Si. *Tetgen, a delaunay-based quality tetrahedral mesh generator.*, 2015.
- **Multitreading**: distribute the Hilbert curve in M threads.

$$\mathsf{DT}_{k+1} = \mathsf{DT}_k + \sum_{i=0}^{M-1} \left[-\mathcal{C}(\mathsf{DT}_k, p_{k+i\frac{n}{M}}) + \mathcal{B}(\mathsf{DT}_k, p_{k+i\frac{n}{M}}) \right].$$





Hilbert curves

A curve $x(t)$ is defined as the mapping

$$x(t)\ ,\ t \in [0,1] \rightarrow x \in \mathbb{R}^3.$$

Curves are perceived as one dimensional objects. Yet, it can be shown that a continuous curve can pass through every point of a unit square. The Hilbert space filling $\mathcal{H}(t)$ curve is a one dimensional curve which visits every point within a two dimensional space. It may be thought of as the limit

$$\mathcal{H}(t) = \lim_{k\rightarrow\infty} \mathcal{H}_k(t)$$

of a sequence of curves $\mathcal{H}_k$ (see Figure 1).

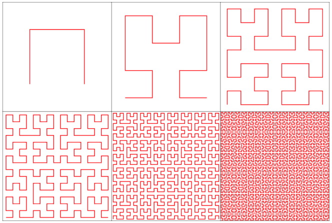


Figure: Sequense of Hilbert curves $\mathcal{H}_k$.

Hilbert curves

Curves $\mathcal{H}_1$ and $\mathcal{H}_2$ are depicted on Figure 2.

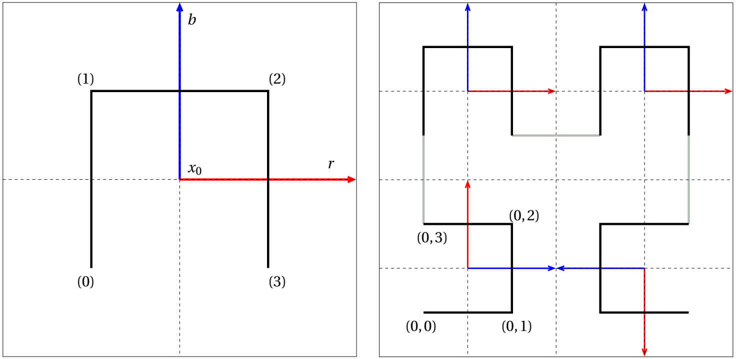


Figure: Curves $\mathcal{H}_1$ and $\mathcal{H}_2$.

Look at `hilbert2d.cpp`.

Hilbert curves

Hilbert curves provide an ordering for points on a plane. Forget about how to connect adjacent sub-curves, and instead focus on how we can recursively enumerate the quadrants.
A local frame is associated to each quadrant: it consist in its center x_0 two orthogonal vectors b and r (see Figure 2). At the root level, enumerating the points is simple: proceed around the four quadrants, numbering them

$$(0) = x_0 - \frac{b+r}{2} \quad (1) = x_0 + \frac{b-r}{2} \quad (2) = x_0 + \frac{b+r}{2} \quad (3) = x_0 - \frac{b-r}{2}.$$

We want to determine the order we visit the sub-quadrants while maintaining the overall adjacency property. Examination reveals that each of the sub-quadrants curves is a simple transformation of the original pattern. Figure 2 illustrate the first level of that recursion.

Hilbert curves

Quadrant (0) is itself divided into four quadrants $(0,0)$, $(0,1)$, $(0,2)$ and $(0,3)$. Its center is simply set to (0) and two vectors b and r are changed as

$$b \leftarrow r/2 \text{ and } r \leftarrow b/2.$$

For quadrant $(0,1)$ and $(0,2)$ we have

$$b \leftarrow b/2 \text{ and } r \leftarrow r/2.$$

and finally for quadrant $(0,3)$:

$$b \leftarrow -r/2 \text{ and } r \leftarrow -b/2.$$

creates 4 sub quadrants. If we consider a maximal recursion depth of d, each of the final subquadrants will be assigned to a set of d "coordinates" i.e. $(k_0, k_1, \dots, k_d)$, k_j being 0,1,2 or 3.
Algorithm in Listings **??** compute the Hilbert coordinates of a given point x, y, starting from an initial quadrant define by its center x_0, y_0 and two orthogonal directions.

Hilbert curves

Each point x of $\mathbb{R}^2$ has its coordinates on the Hilbert curve. Sorting a point set with respect to Hilbert coordinates allow to ensure that two successive points of the set are close to each other. In the context of the Bowyer-Watson algorithm, this kind of data locality could potentially decrease the number of local searches N_{search} that were required to find the next invalid triangle.

Sets of 1000 and 10000 sorted points are presented on Figure 3. On the Figure, two successive points in the sorted list are linked with a line.

The main cost of sorting points is on the sorting algorithm itself and not on the computation of the Hilbert curve coordinates: sorting over a million points takes less than a second on a standard laptop. Table 1 present timings and statistics for the same point sets as in table **??**, but while having sorted the points S using the Hilbert curve.

n	10^3	10^4	10^5	10^6
N_{search}	2.34	2.46	2.50	2.50
N_{cavity}	4.06	4.13	4.16	4.17
t(sec)	0.0097	0.090	0.92	9.2

Table: Results of the `delaunayTrgl` algorithm applied to random points. Points were initially sorted through using a Hilbert sort.

Hilbert curves

The number of serarches is not increasing anymore with the size of the set. This is important: the complexity of the Delaunay triangulation algorithm now is linear in time. Of course, sorting points has a $n \log n$ complexity so that the overall process is in $n \log n$ as well. Yet, the relative cost of sorting the points is negligible with respect to the cost of the triangulation itself.

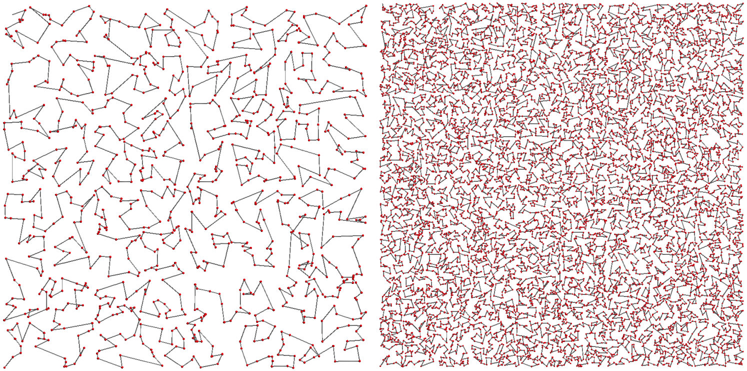
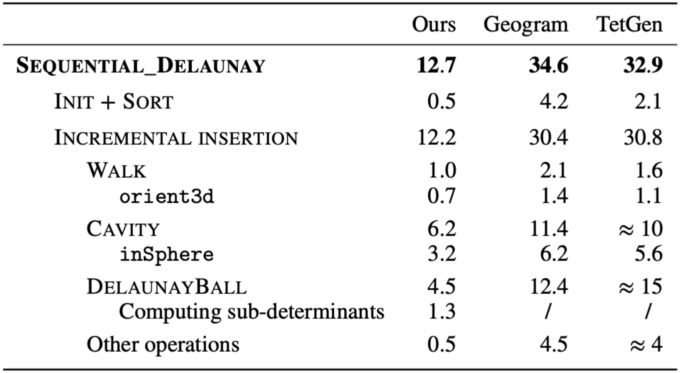


Figure: Hilbert sort of sets of 1000 and 10000 random points.

Results for 5×10^6 vertices



	Ours	Geogram	TetGen
SEQUENTIAL_DELAUNAY	**12.7**	**34.6**	**32.9**
INIT + SORT	0.5	4.2	2.1
INCREMENTAL INSERTION	12.2	30.4	30.8
WALK	1.0	2.1	1.6
`orient3d`	0.7	1.4	1.1
CAVITY	6.2	11.4	≈ 10
`inSphere`	3.2	6.2	5.6
DELAUNAYBALL	4.5	12.4	≈ 15
Computing sub-determinants	1.3	/	/
Other operations	0.5	4.5	≈ 4

Sorting using HXTSort

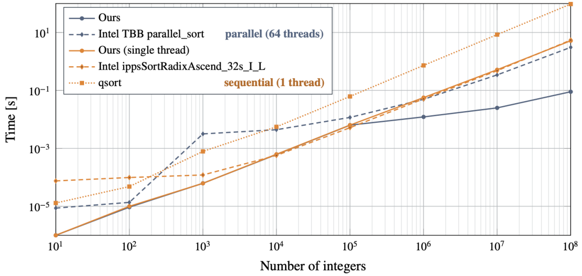
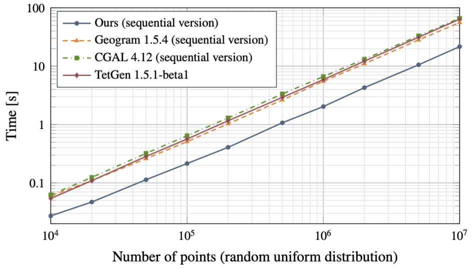


FIGURE 3 Performances of `HXTSort` for sorting 31-bit integers produced by `rand()` on an Intel® Xeon Phi™ 7210 CPU and comparison with widely used implementations. Each integer is both the key and the value.

Comparison (sequential)



# vertices	10^4	10^5	10^6	10^7
Ours	0.027	0.21	2.03	21.66
Geogram	0.060	0.51	5.53	56.02
CGAL	0.062	0.64	6.65	66.24
TetGen	0.054	0.56	5.89	63.99

(a) Intel® Core™ i7-6700HQ CPU, maximum core frequency of 3.5Ghz

Multithreading

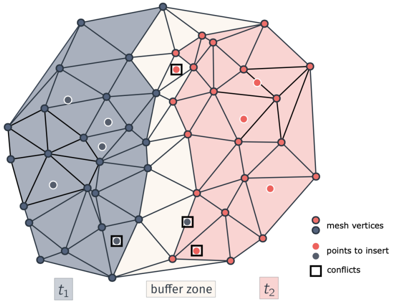
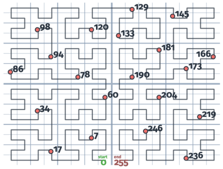
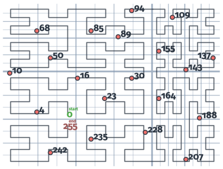


FIGURE 7 Vertices are partitionned such that each vertex belongs to a single thread. A triangle can only be modified by a thread that owns all of its three vertices. Triangles that cannot be modified by any thread form a buffer zone.

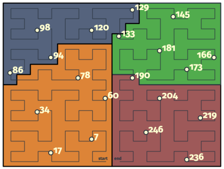
Multithreading

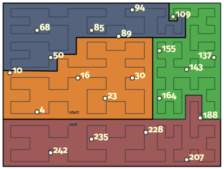


(a)



(b)





Multithreading

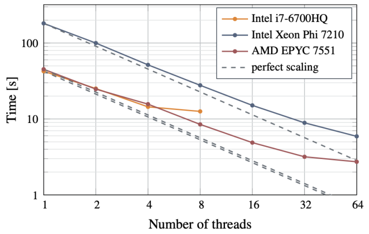


FIGURE 9 Scaling of our parallel Delaunay for a random uniform distribution of 15 million points, resulting in over 100 million tetrahedra on 3 machines: a quad-core laptop, an Intel Xeon Phi with 64 cores and a dual-socket AMD EPYC 2 × 32 cores.

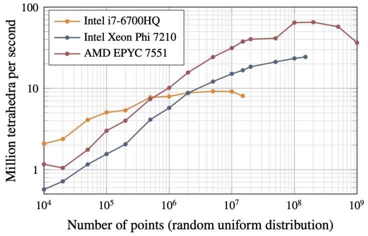
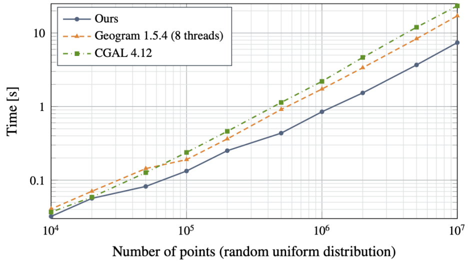


FIGURE 10 Number of tetrahedra created per second by our parallel implementation for different number of points. Tetrahedra are created more quickly when there is a lot of points because the proportion of conflicts is lower. A rate of 65 million tetrahedra created per second is obtained on the EPYC.

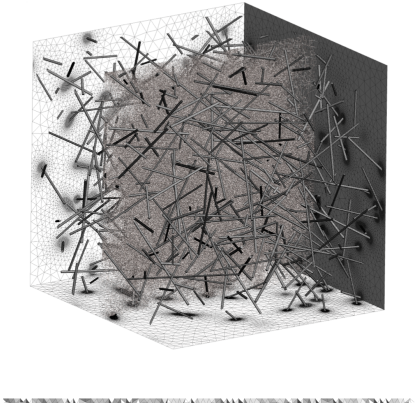
Multithreading



# vertices	10^4	10^5	10^6	10^7
Ours	0.032	0.13	0.85	7.40
Geogram	0.041	0.19	1.73	17.11
CGAL	0.037	0.24	2.20	23.37

(a) 4-core Intel® Core™ i7-6700HQ CPU.

Multithreading



100 thin fibers

# threads	# tetrahedra	t_{brec}	t_{ref}	t
1	325,611,841	3.1	492.1	497.2
2	325,786,170	2.9	329.7	334.3
4	325,691,796	2.8	229.5	233.9
8	325,211,989	2.7	154.6	158.7
16	324,897,471	2.8	96.8	100.9
32	325,221,244	2.7	71.7	75.8
64	324,701,883	2.8	55.8	60.1
127	324,190,447	2.9	47.6	52.0

500 thin fibers

# threads	# tetrahedra	t_{brec}	t_{ref}	t
1	723,208,595	18.9	1205.8	1234.4
2	723,098,577	16.0	780.3	804.8
4	722,664,991	86.6	567.1	659.8
8	722,329,174	15.8	349.1	370.1
16	723,093,143	15.6	216.2	236.5
32	722,013,476	15.6	149.7	169.8
64	721,572,235	15.9	119.7	140.4
127	721,591,846	15.9	114.2	135.2

Multithreading

Mechanical part

# threads	# tetrahedra	t_{brec}	t_{ref}	t
1	24,275,207	8.6	43.6	56.3
2	24,290,299	8.4	30.4	41.8
4	24,236,112	8.1	24.6	35.3
8	24,230,468	8.1	21.8	32.6