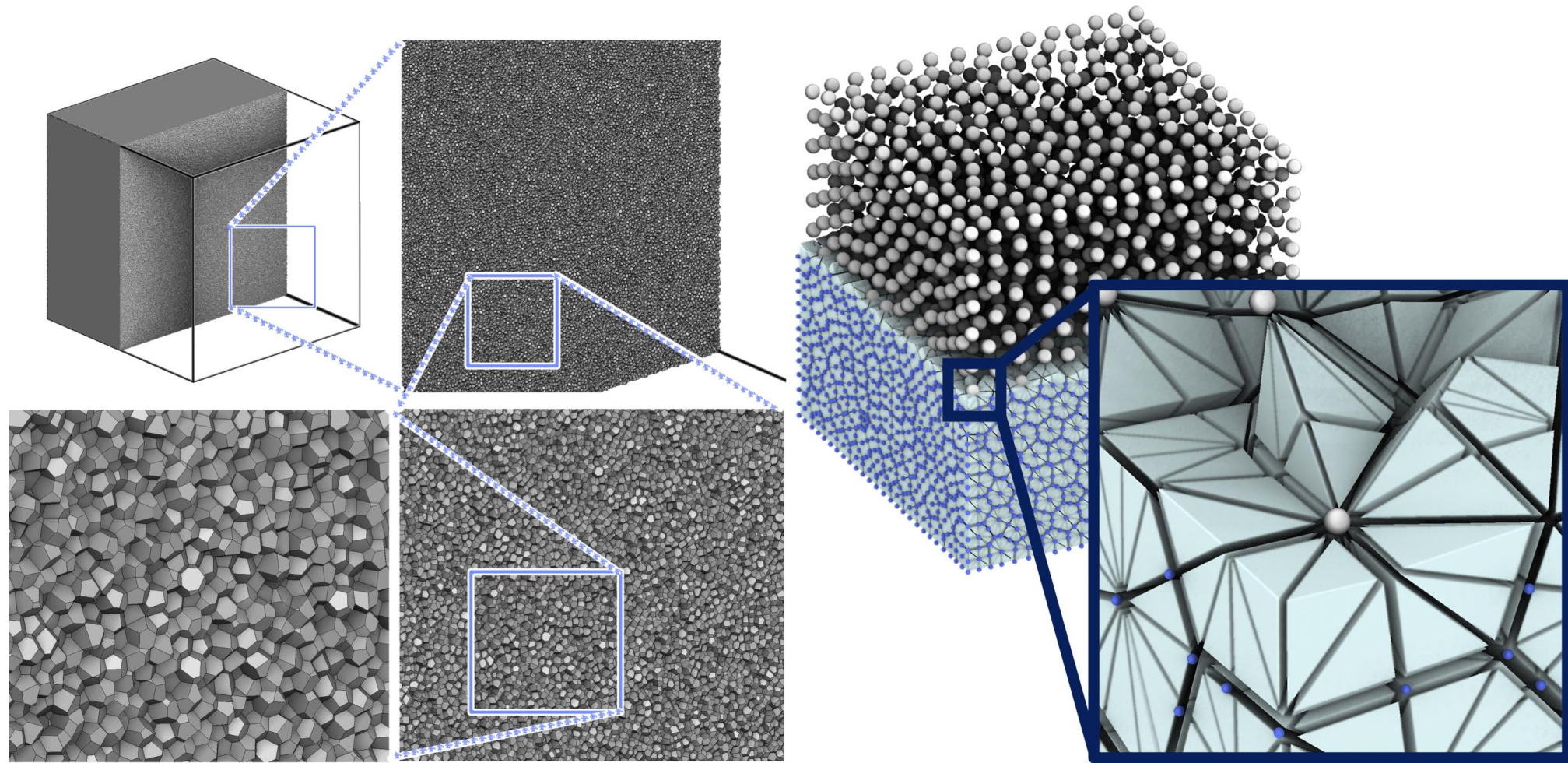




Diagrammes de Voronoï sur GPU

Groupe de lecture

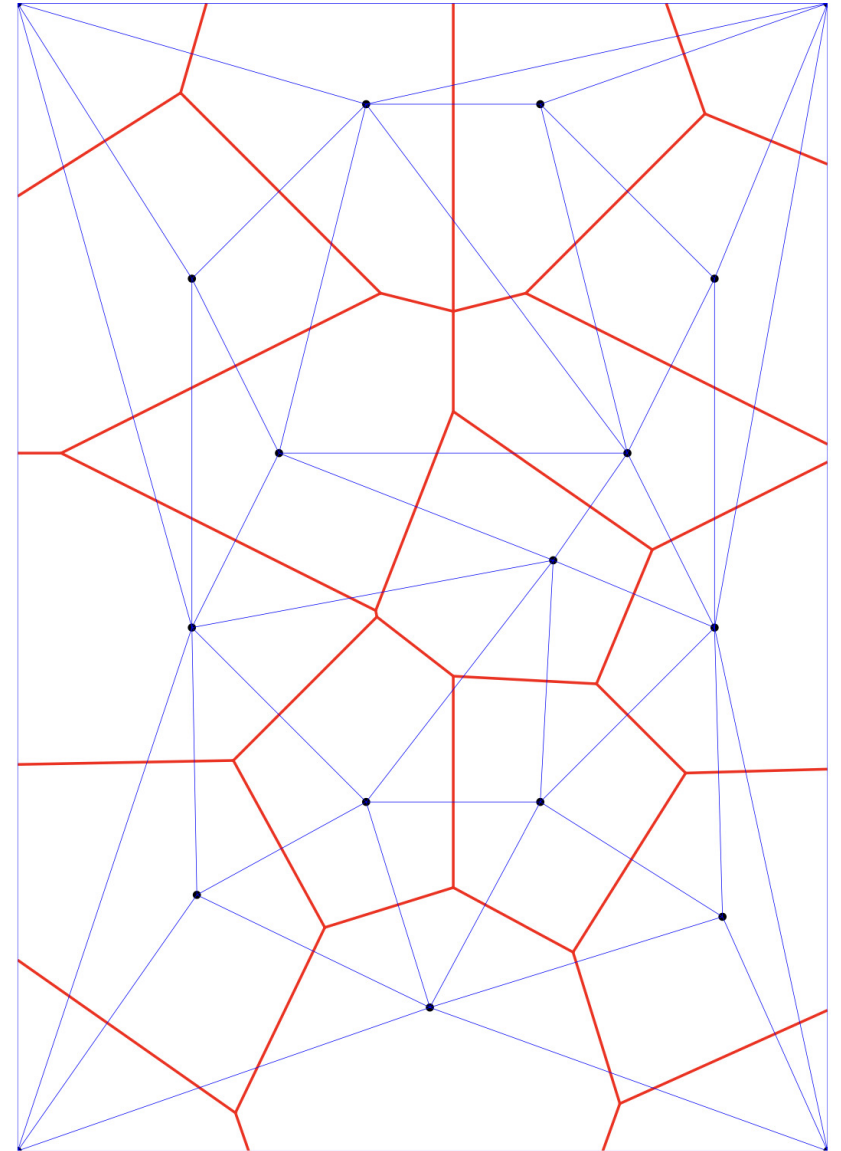
Diagrammes de Voronoï et de puissance

Diagramme de Voronoï

- $p_i \in \mathbb{R}^d$
- $d(x, p_i) = \|x - p_i\|$
- Partition de l'espace
- Chaque sommet décrit une sphère vide tangente à 3 sites

Triangulation de Delaunay

- Dual du diagramme de Voronoï
- Chaque simplex est associé à une sphère tangente vide

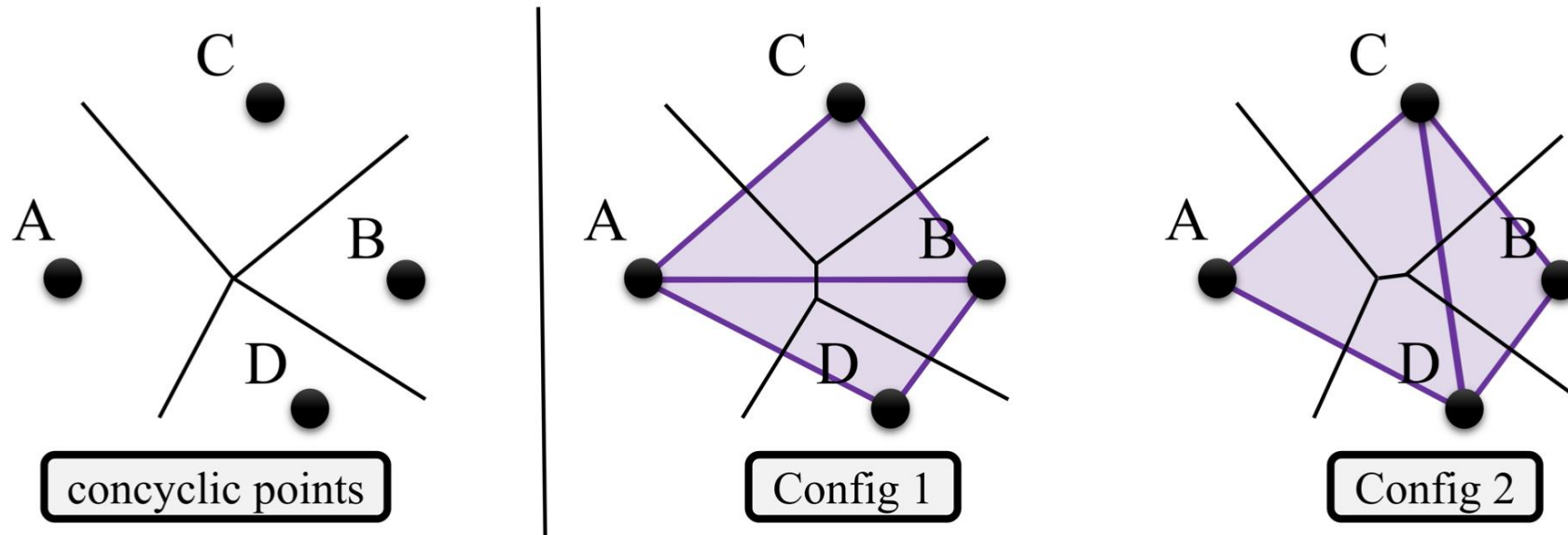


[Buchin2019]

Challenges de la construction d'un diagramme de Voronoï

Structure combinatoire & triangulation de Delaunay

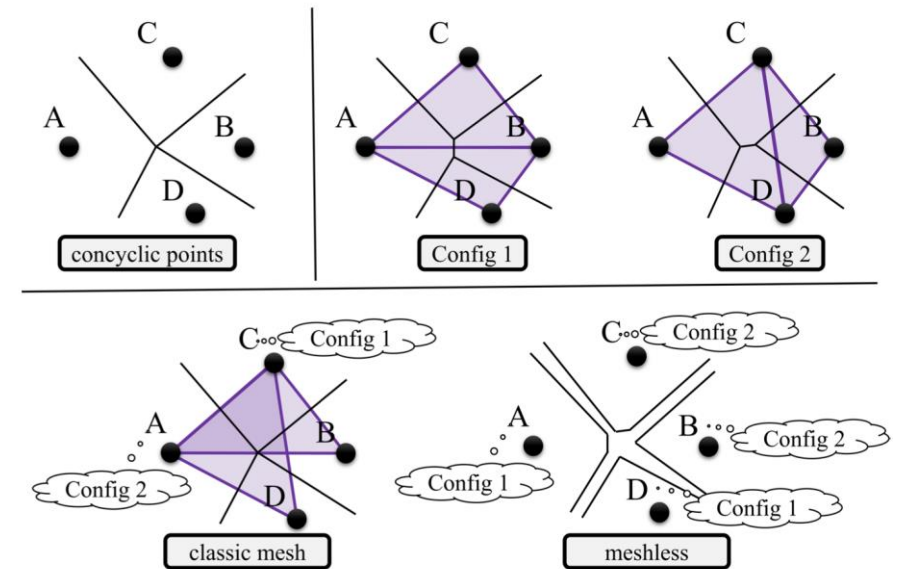
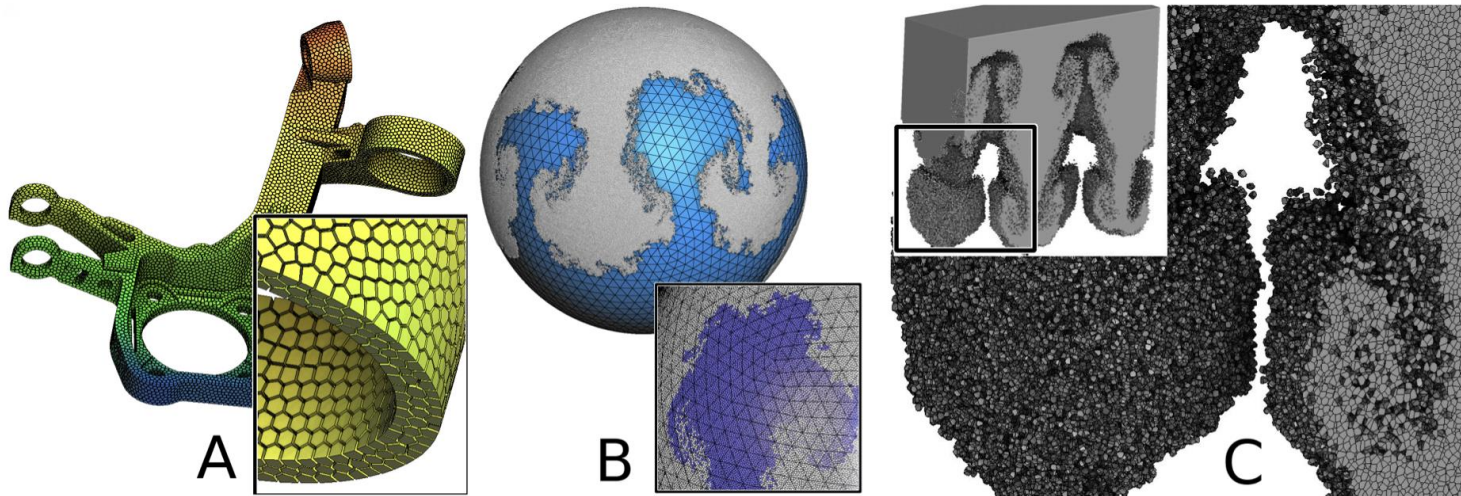
- Nécessaire pour les applications liées aux maillages
- Algorithme séquentiel assez rapide, mais difficile à paralléliser
- Requiert des prédicats exacts (Pas de triangles plats, ...)



Utilisation comme une partition de l'espace

La structure combinatoire n'est pas forcément nécessaire

- Simulation par éléments finis
- Relaxation de Lloyd / Kmeans
- Simulation particulière (fluide, cosmique, ...)



Construction de la géométrie par cellule

Contexte

- Distribution spatiale homogène (White noise, blue noise, ...)
 - Les k plus proches voisins sont $\approx$ équivalents aux voisins
- Calcul de la géométrie
 - On ne souhaite pas stocker la structure combinatoire

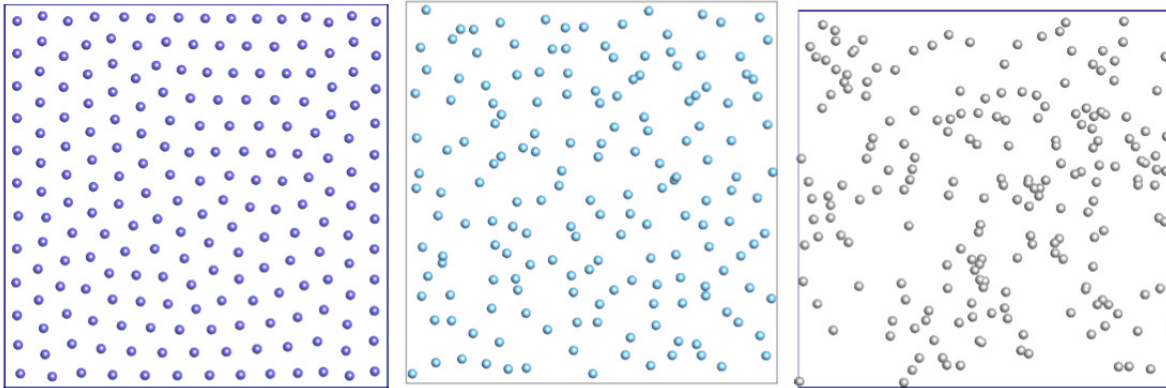


Fig. 5. The different classes of point distributions that we consider. From left to right: blue noise, perturbed grid and white noise.

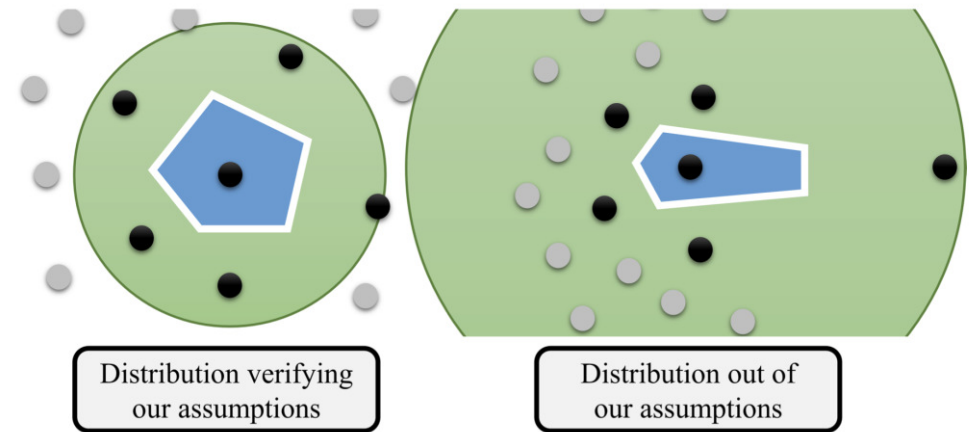


Fig. 4. Point-set distribution quality. To compute a Voronoi cell (blue), we need all the black points. We search them in the k nearest neighbors of the seed. For us, a good distribution will have a minimal number of points in the smallest disc (green) centered on the seed, and that contains all black points.

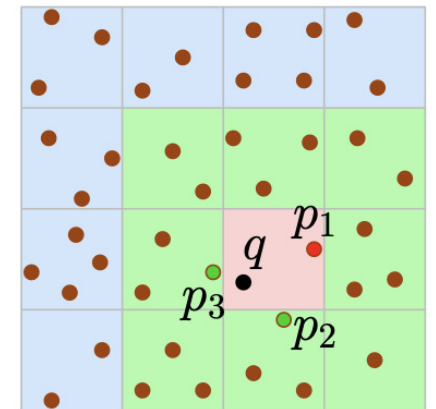
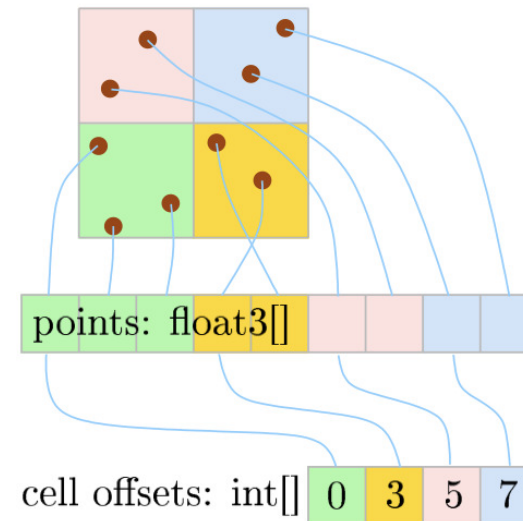
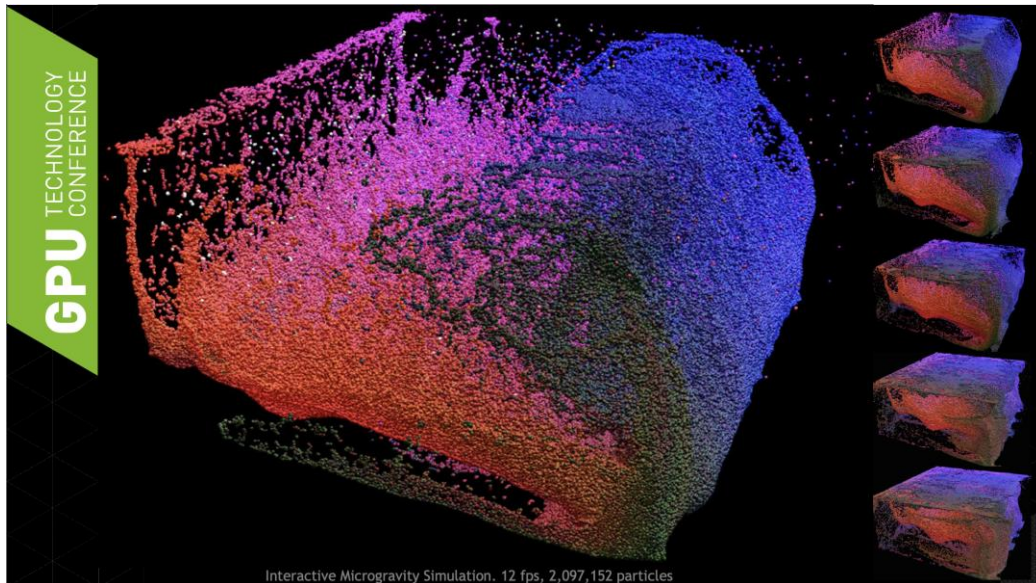
Overview

1. Recherche des k plus proches voisins

- Grille accélératrice 3D
- Visite en cercle concentrique
- Tri par tas (in-place) permettant de facilement stocker la valeur maximale

2. Calcul des cellules

- Itération à travers le tableau de voisin
- Mise à jour de la structure de données
- Calcul d'un critère d'arrêt



Construction par intersection de semi-espace

Bisecteur : $\Pi_{ij} := \{x \mid d(x, p_i) = d(x, p_j) \leq d(x, p_k) \forall k \in P \setminus \{i, j\}\}$

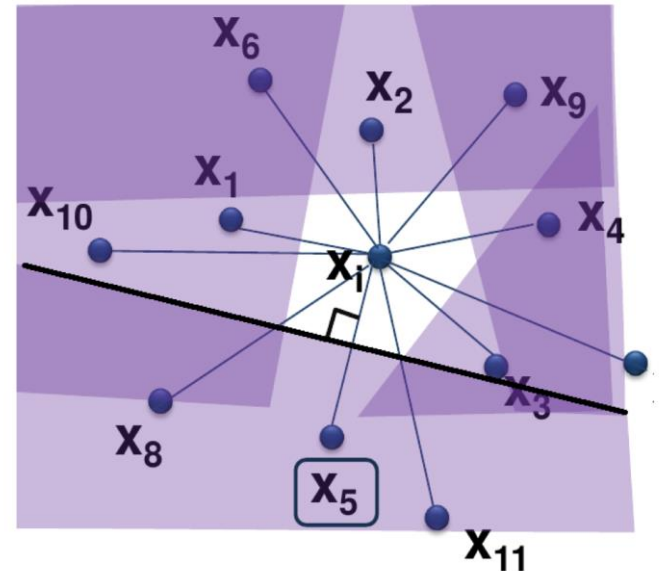
Semi-espace : $\Pi_{ij}^+ := \{x \mid d(x, p_i) < d(x, p_j)\}$, $\Pi_{ij}^- := \{x \mid d(x, p_j) < d(x, p_i)\}$

Une cellule comme l'intersection des semi-espaces

$$Vor(p_i) := \{x \mid d(x, p_i) < d(x, p_j), \forall j \in P \setminus \{i\}\}$$

$$\Leftrightarrow x \in \bigcap_j \{x \mid d(x, p_i) < d(x, p_j)\}$$

$$\Leftrightarrow x \in \bigcap_j \Pi_{ij}^+(i, j)$$



Rayon de sécurité

La contribution d'un voisin potentiel est bornée

- Les voisins sont obtenus par ordre croissant
- On peut trouver la sphère englobante de la cellule au temps t
- Un voisin ne peut pas contribuer s'il est 2x plus loin que son rayon

→ **La cellule est totalement connue et donc valide**

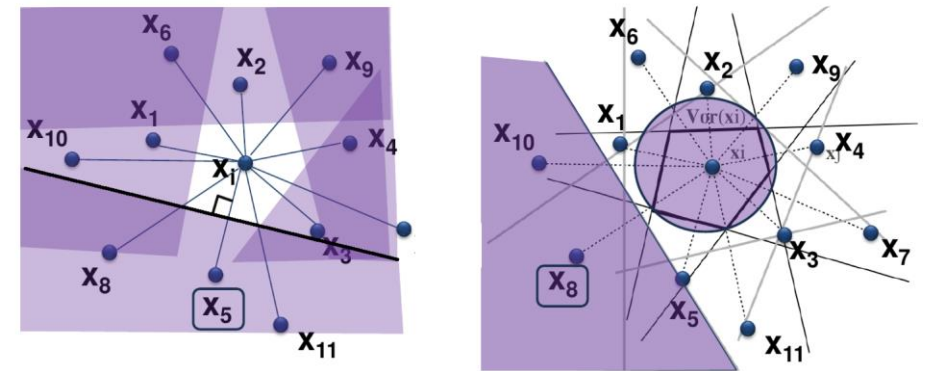
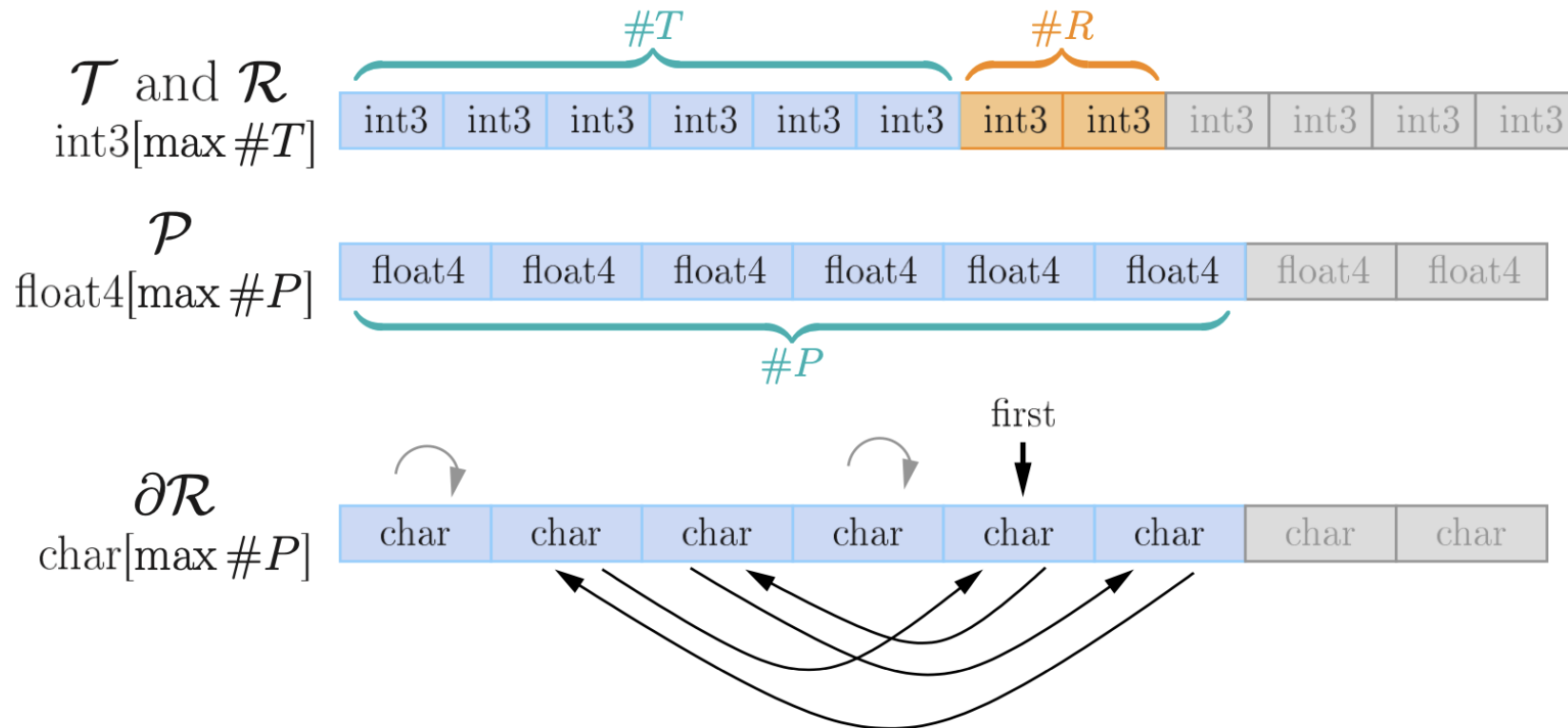


Fig. 7. Left: a Voronoi cell corresponds to the intersection of half-spaces; Right: the “Radius of Security” criterion selects the points that contribute to a Voronoi cell.

Structure de données

Peu de mémoire disponible, légèreté visée

- Équation des semi-espaces ($P_j := \mathbf{a}x_1 + \mathbf{b}x_2 + \mathbf{c}x_3 + \mathbf{d} = 0$): 4 flottants
- Structure combinatoire des sommets (P_j, P_k, P_l): 3 indices
- Arêtes à la frontière d'une zone invalidée: 1 indice



Mise à jour de la structure de données

- I. Identification des sommets invalides
- II. Ajout du nouveau semi-espace
- III. Calcul des bordures de la zone invalide
- IV. Pour chaque arête de la bordure: ajout d'un nouveau sommet

→ Stockage implicite des arêtes

→ Mise à jour simple et rapide

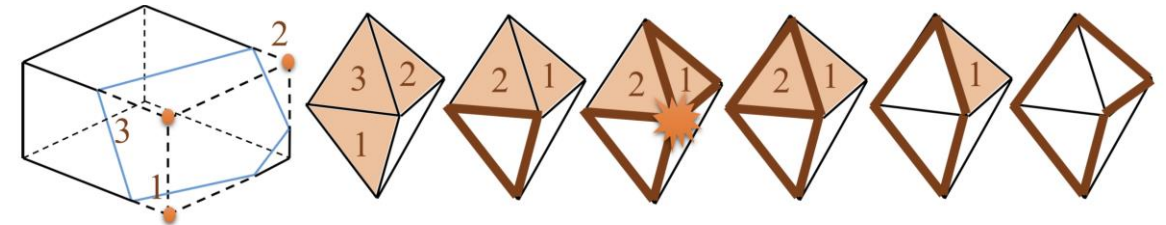
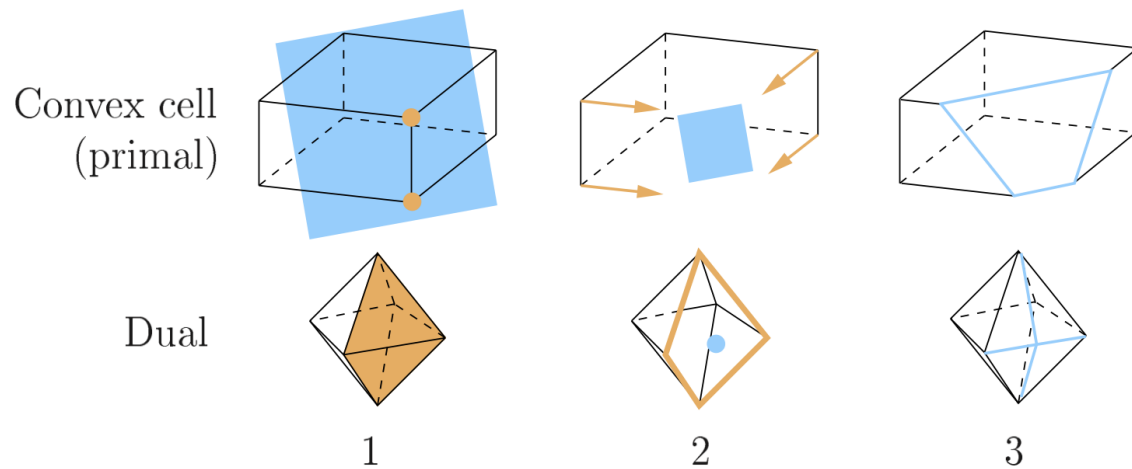


Fig. 11. Finding the boundary of $\mathcal{R}$: We initialize with the boundary of 1, and remove it from $\mathcal{R}$. We try to add the boundary of the new 1, but it would fail to produce a simple cycle. We try to add the boundary of 2 and it works, so we remove 2 from $\mathcal{R}$. Now we finish by adding the boundary of 1 and emptying $\mathcal{R}$.

Analyse

Performances

- 10x plus rapide que les méthodes CPU
- Nécessite une recherche des meilleurs paramètres
- Très sensible à la distribution des entrées

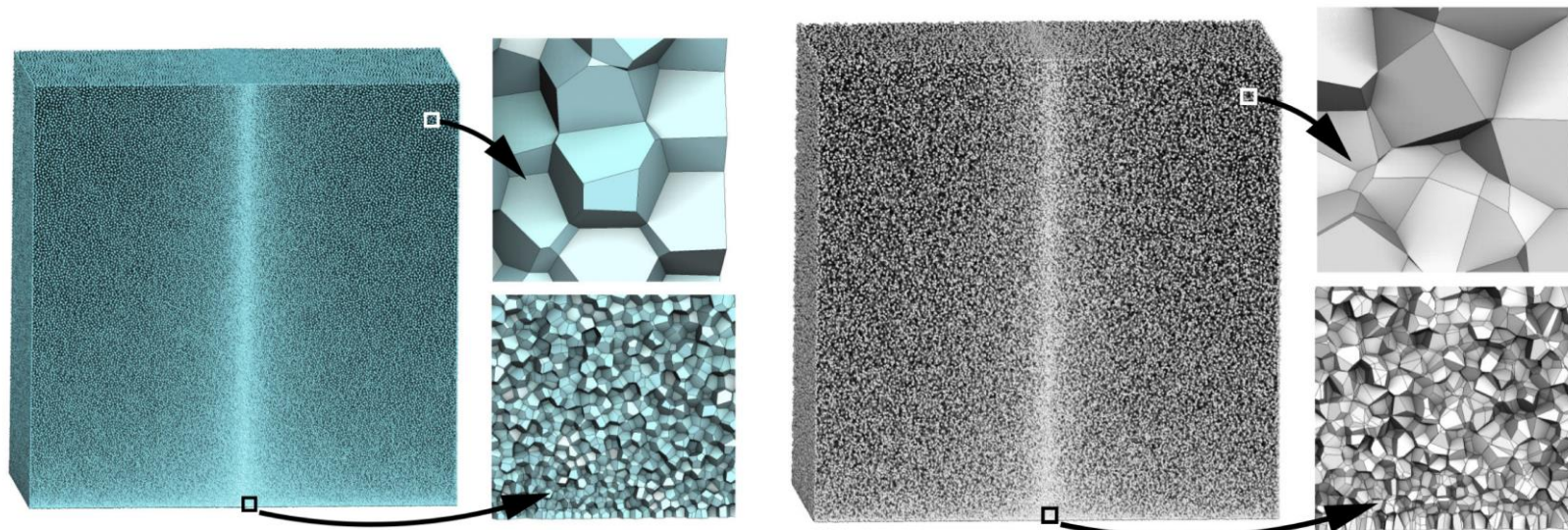


Table 2. Timings for different types of data-sets with tuned settings (seconds). white noise ($k = 190, \max \#T = 96, \max \#P = 64$) perturbed grid ($k = 90, \max \#T = 96, \max \#P = 50$) blue noise ($k = 35, \max \#T = 96, \max \#P = 32$)

	100K points		1M points		10M points	
	knn	clip	knn	clip	knn	clip
Tesla V100:						
white noise	0.041	0.037	0.360	0.333	3.387	3.334
perturbed grid	0.010	0.018	0.091	0.172	0.772	1.699
blue noise	0.002	0.004	0.016	0.046	0.158	0.417
GTX1080:						
white noise	0.134	0.089	1.275	0.919	n/a	n/a
perturbed grid	0.037	0.051	0.301	0.458	2.575	4.708
blue noise	0.005	0.011	0.052	0.117	0.520	1.135

Table 3. GEOGRAM vs CGAL vs VORO++, 12 cores Intel i7-6800K CPU @ 3.40GHz. Note that VORO++ uses a single core only.

	100K points	1M points	10M points
GEOGRAM 1.6.5:			
white noise	0.2	1.6	15.5
perturbed grid	0.2	1.6	15.5
blue noise	0.2	1.6	15.5
CGAL 4.11:			
white noise	0.15	1.3	12.5
perturbed grid	0.15	1.3	12.5
blue noise	0.15	1.3	12.6
VORO++ 0.4.6:			
white noise	1.6	17.2	173.8
perturbed grid	1.2	12.7	127.4
blue noise	0.9	10.0	98.8