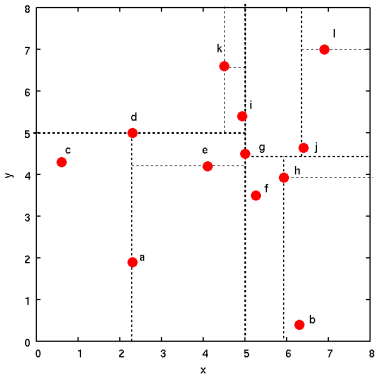
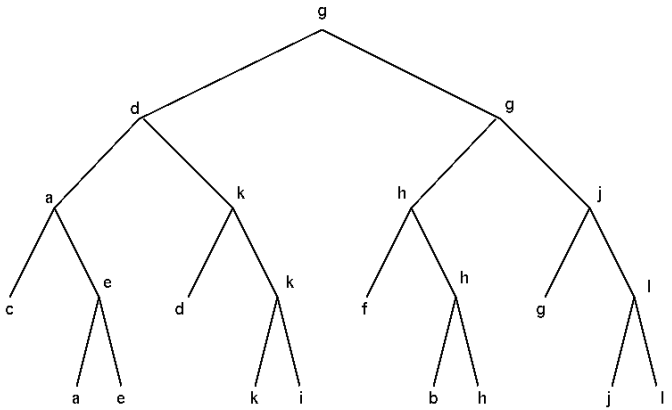
Algorithmique géométrique



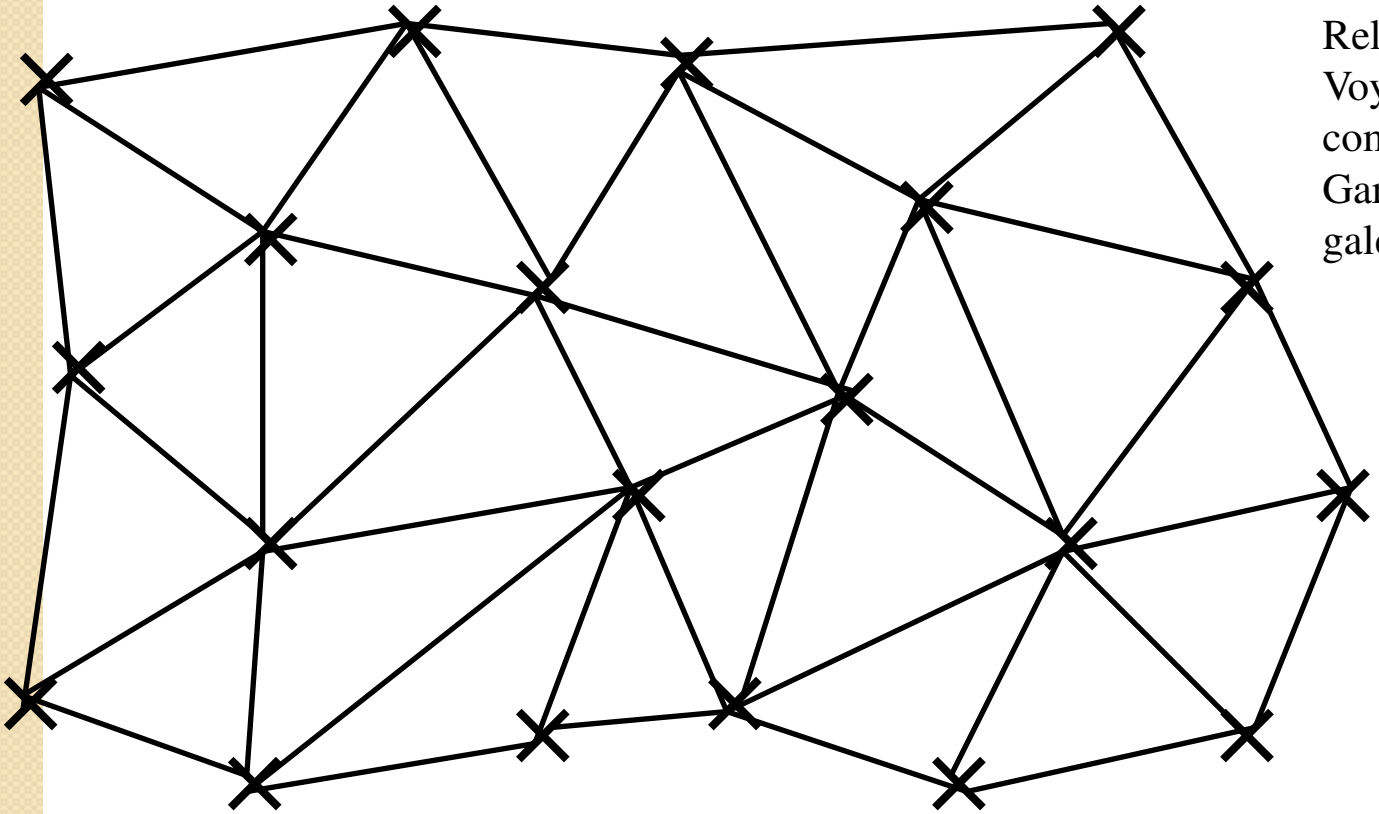


Algorithmique géométrique

1. Introduction
2. Localisation & structures de recherche
3. Triangulations de Delaunay
4. Graphes de Voronoï
5. Généralisation des Graphes de Voronoï

Recherche opérationnelle

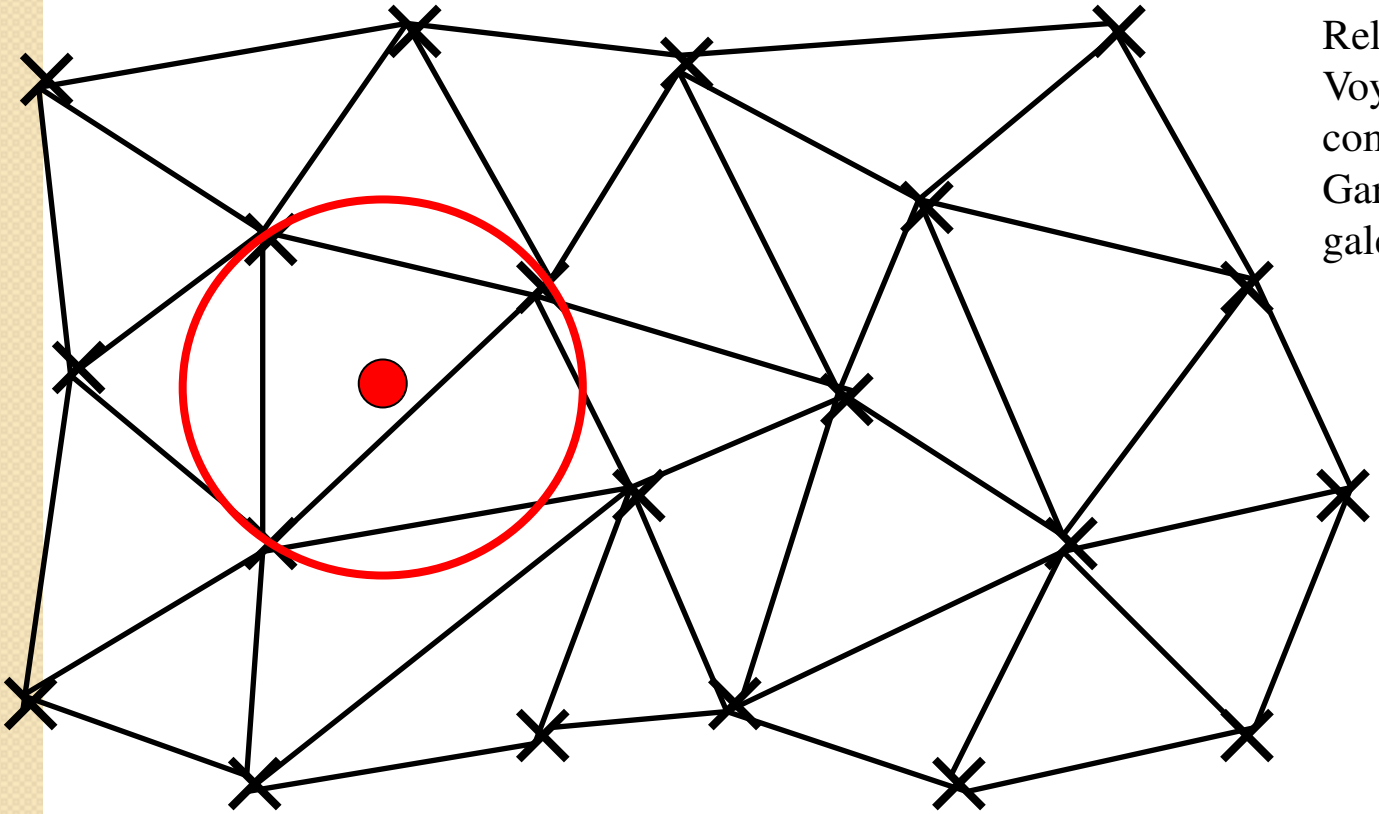
Graphes euclidien (planaire, 3D)



Problèmes types :
Usine polluante,
Relier des stations,
Voyageur de commerce,
Gardien de la galerie d'art

Recherche opérationnelle

Graphes euclidien (planaire, 3D)

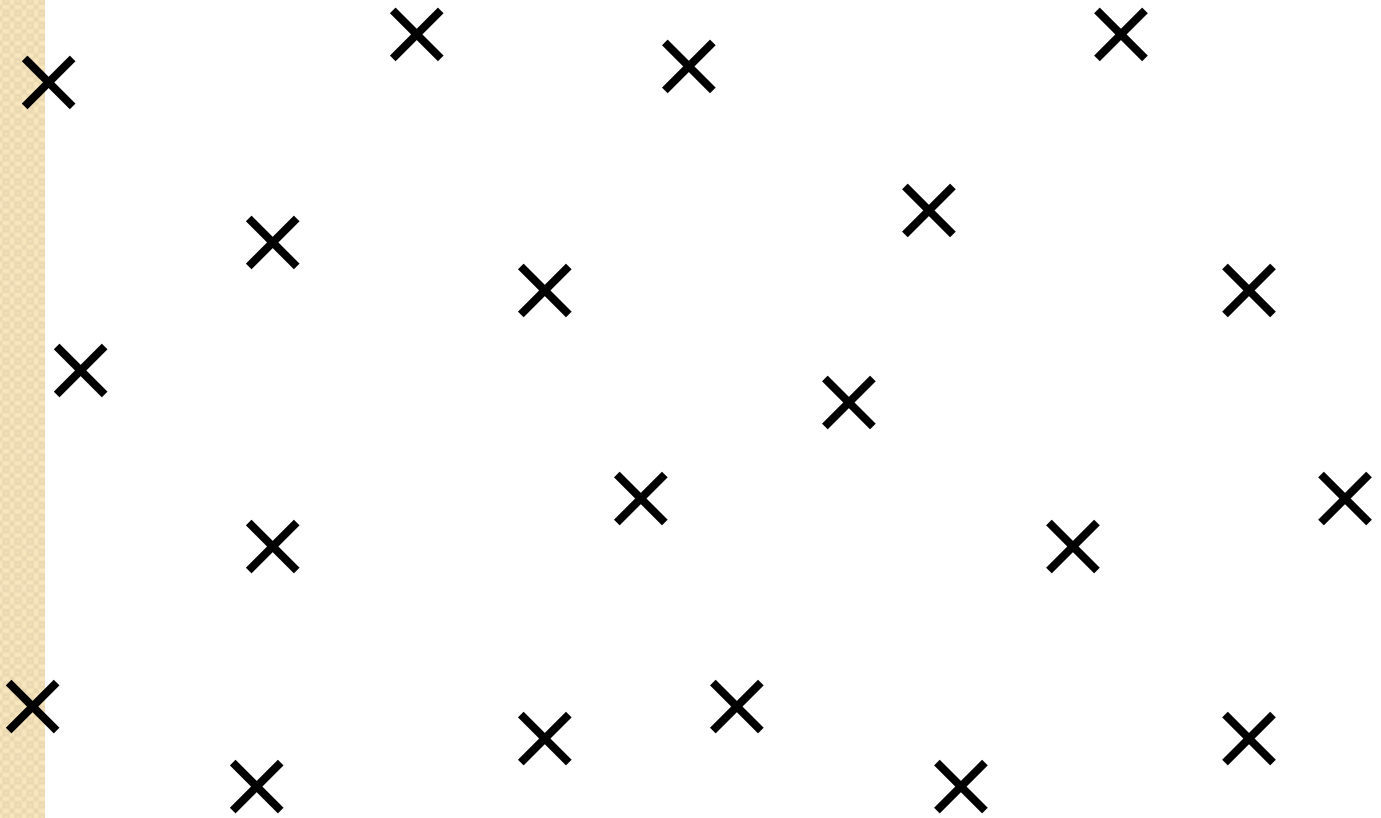


Problèmes types :
Usine polluante,
Relier des stations,
Voyageur de commerce,
Gardien de la galerie d'art

Les problèmes de base

Localisation et recherche 2D, 3D…:

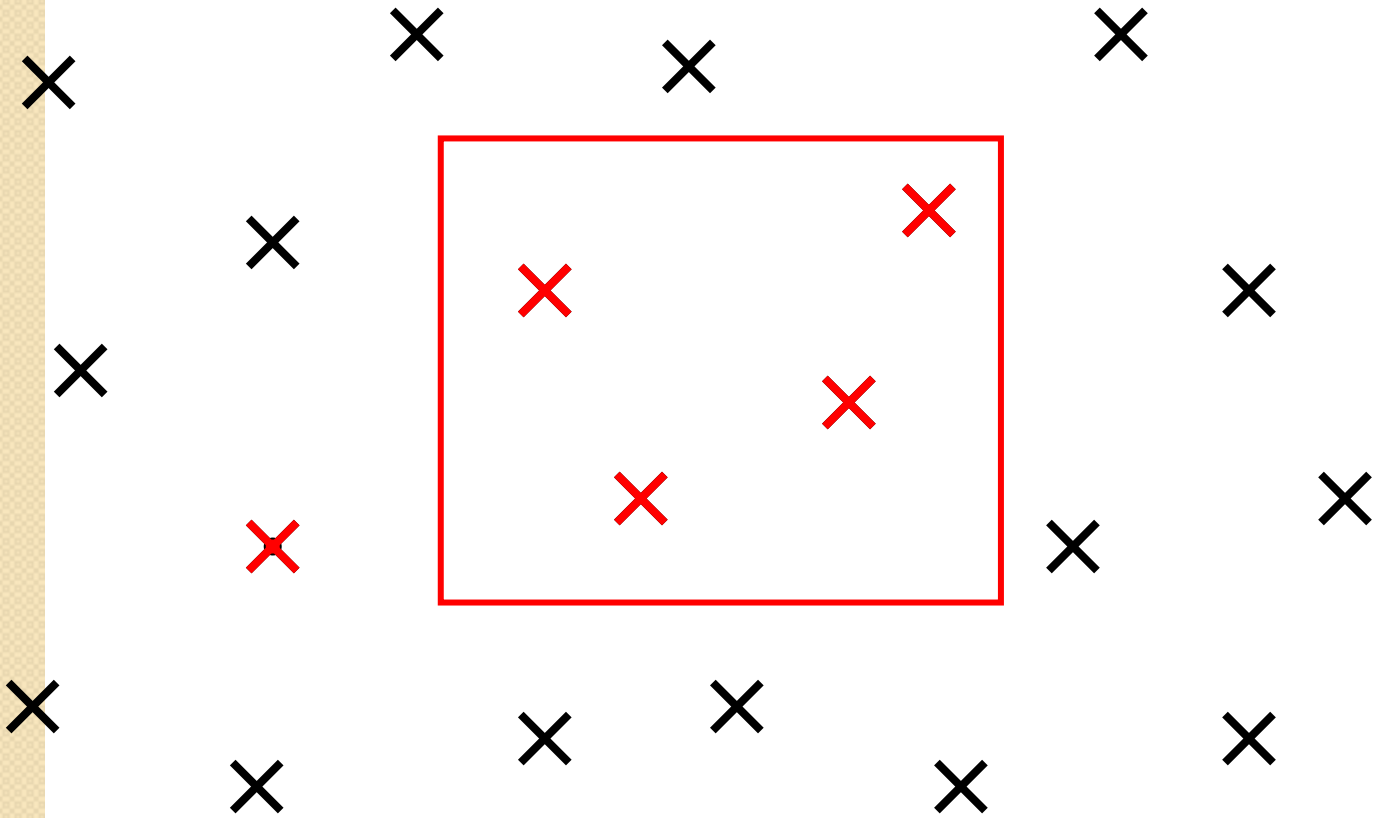
- Localisation point / polygone…
- Requêtes boite, requête exacte



Les problèmes de base

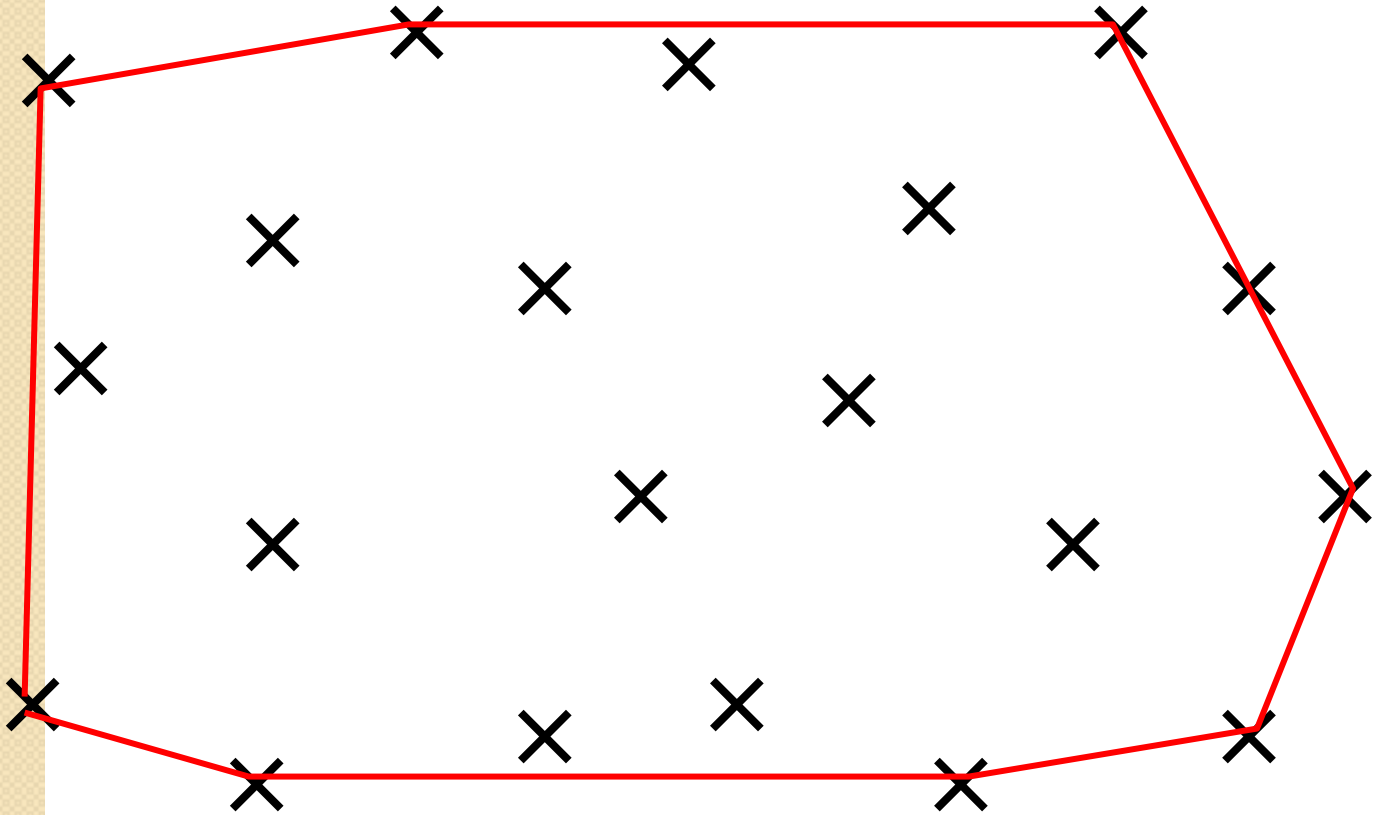
Localisation et recherche 2D, 3D…:

- Localisation point / polygone…
- Requêtes boite, requête exacte



Les problèmes de base

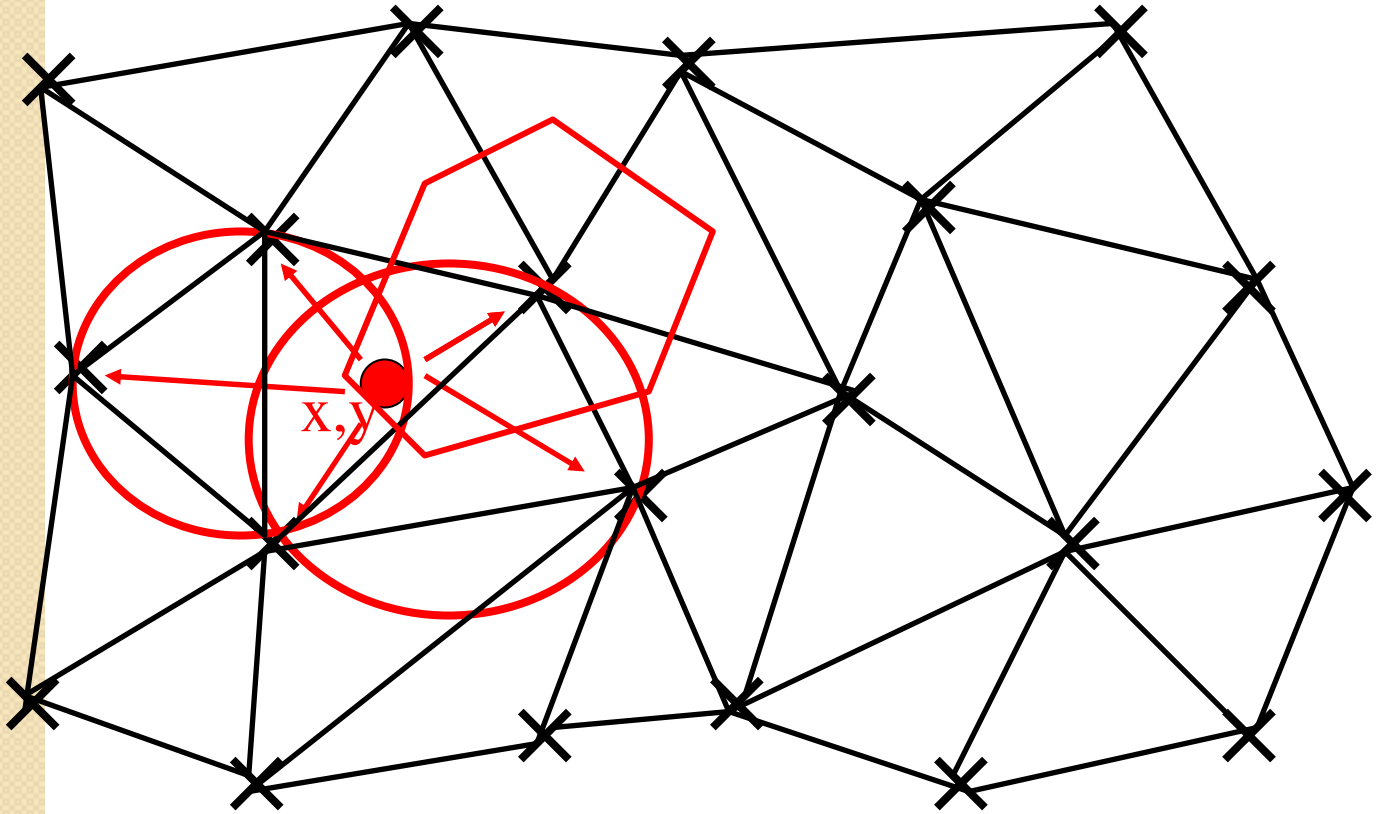
Enveloppe convexes 2D, 3D… :



Les problèmes de base

Calcul de proximité 2D, 3D… :

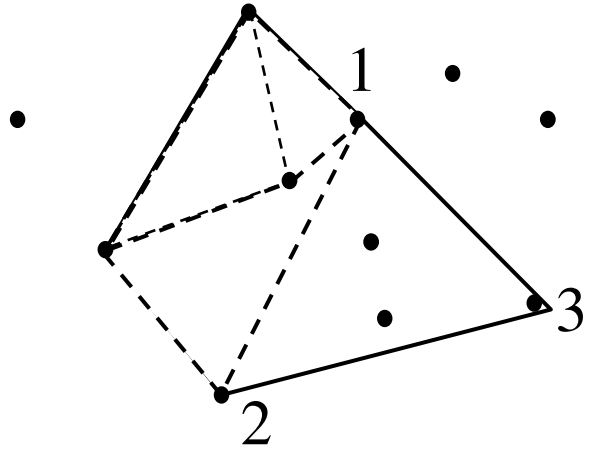
•Les K Plus Proches Voisins



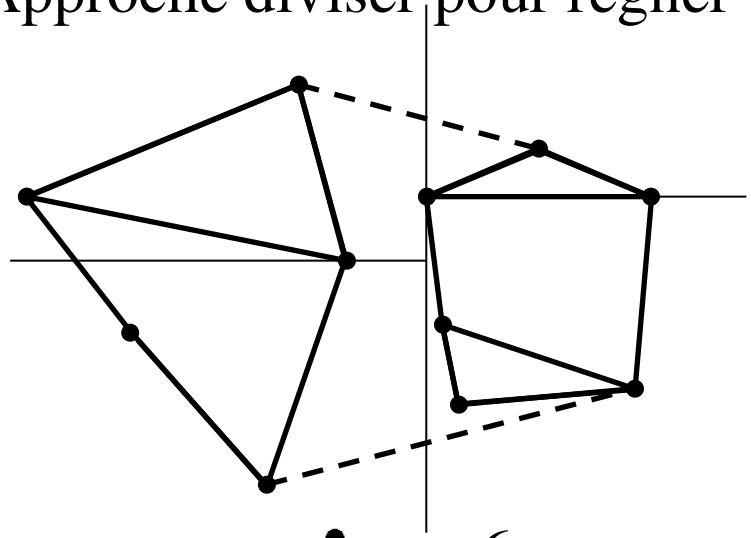
Les 3 approches déterministes

Sur l ’exemple de l’Enveloppe Convexe

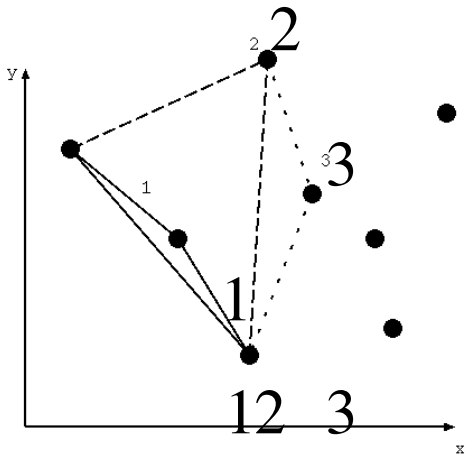
1) Approche incrémentale

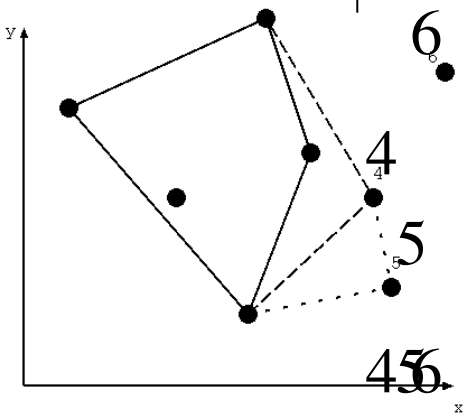


2) Approche diviser pour régner

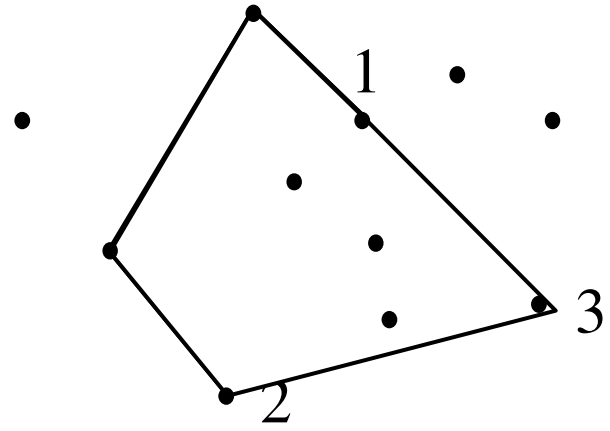


3) Approche par balayage



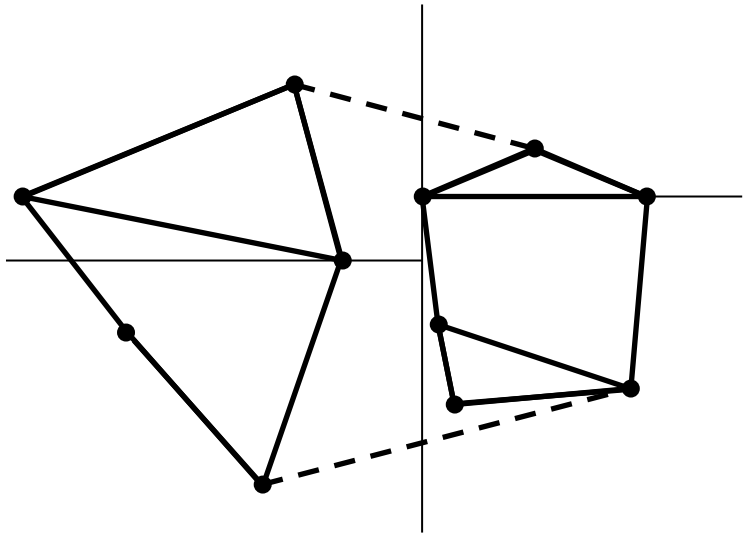


EC : approche incrémentale

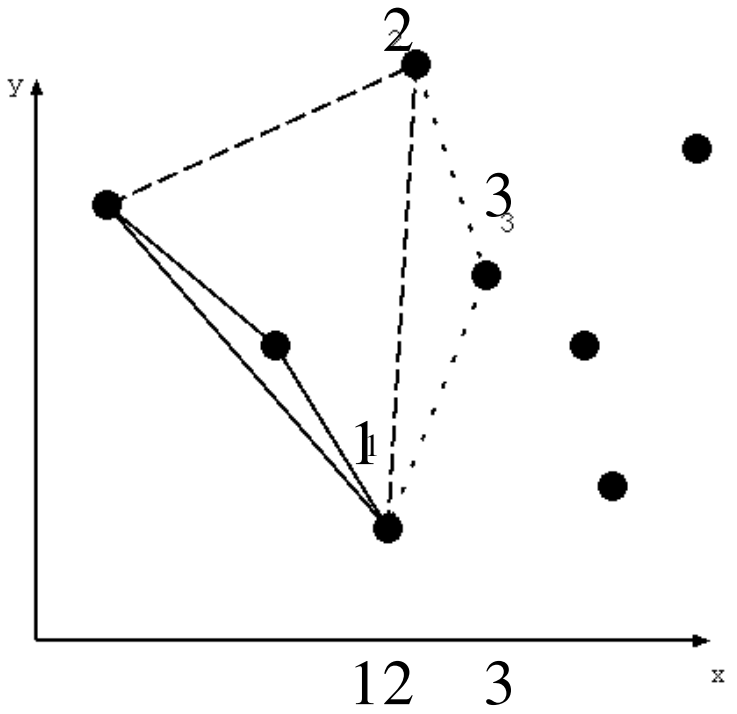


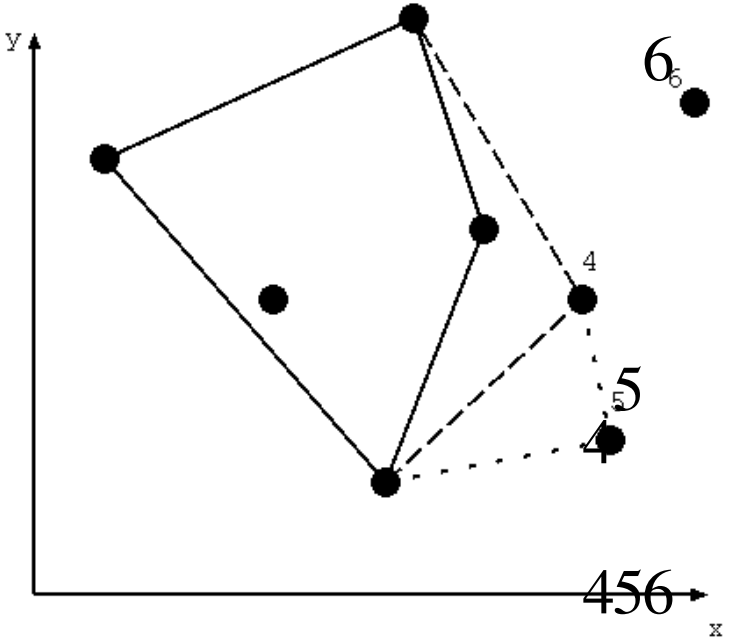
EC : diviser pour régner

1. Construire KDtree
2. Calculer l'EC élémentaire
3. Fusionner les EC 2 à 2



EC par balayage





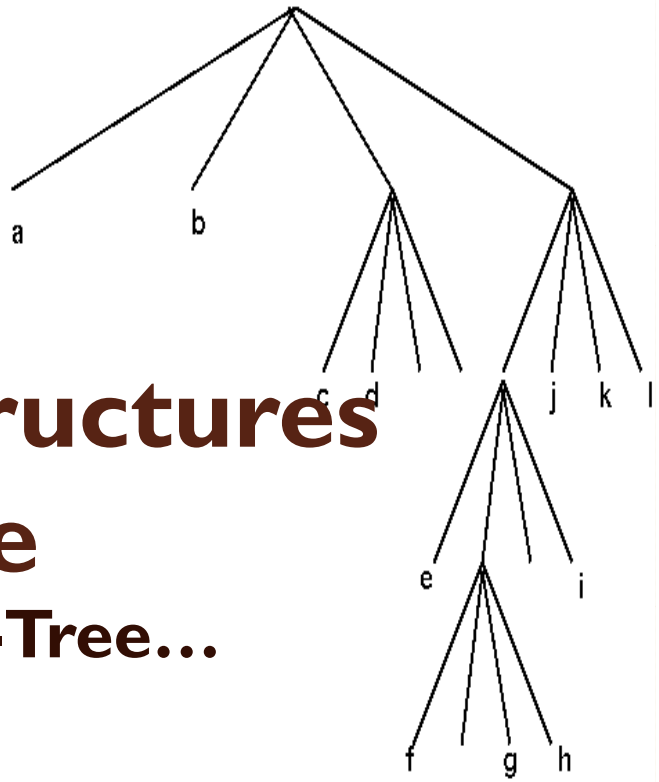
Algorithmique géométrique

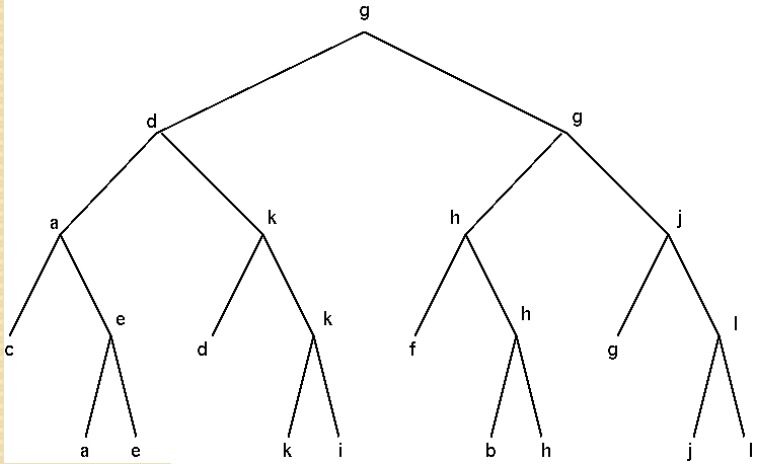
L'algorithmique géométrique est l'étude de **problèmes géométriques** et des **algorithmes** permettant de les traiter du point de vue de la **complexité** et de la robustesse.

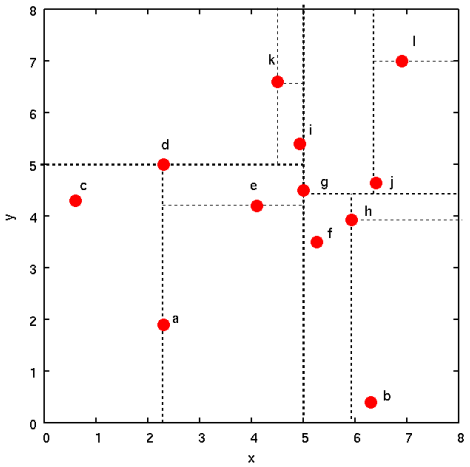
Problèmes de base	**Approches déterministes**	**Complexité**
Localisation et recherche	Incrémentale	Temps/Mémoire
Enveloppes Convexes	Diviser pour régner	Moyenne/Pire cas
Calcul de proximité	Balayage	Algo./Problème

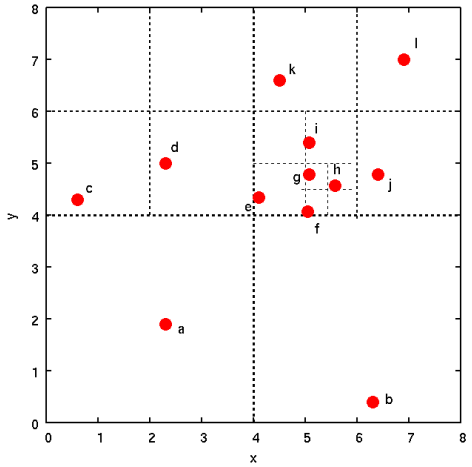
2.Localisation et Structures de Recherche

QuadTree, Octree, KD-Tree…

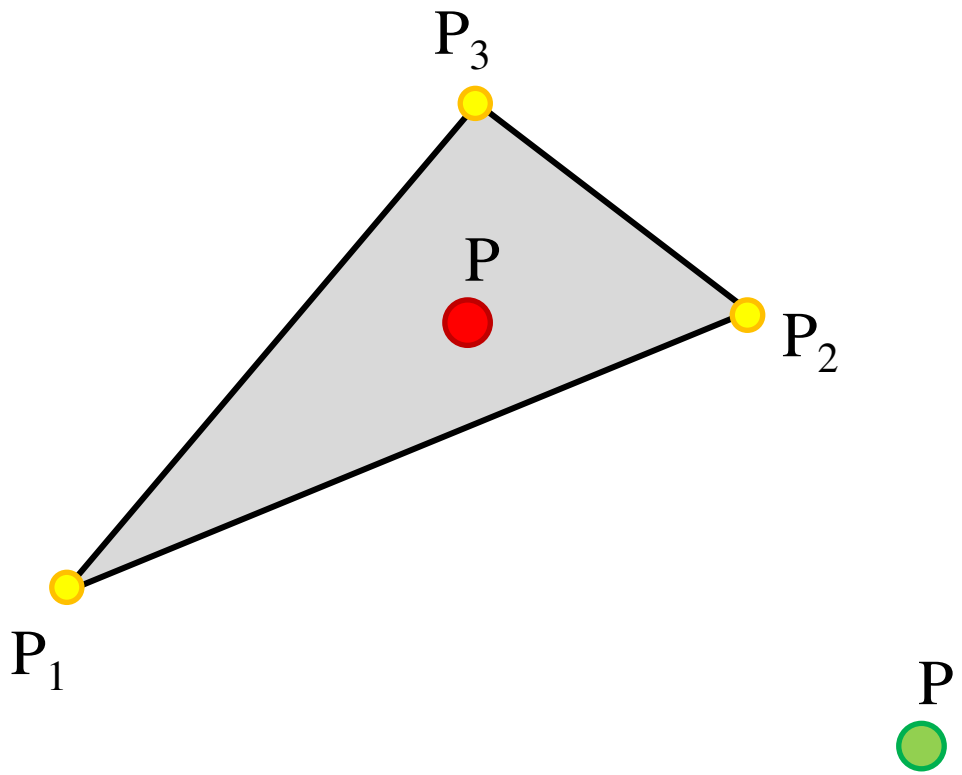






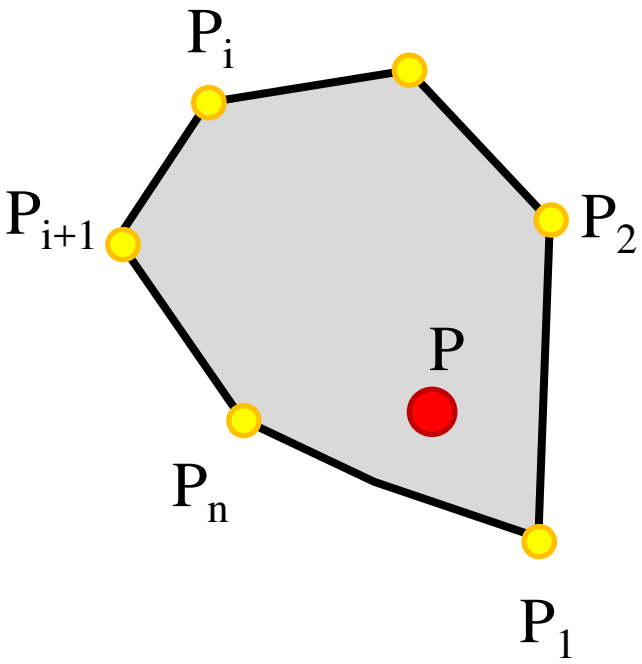


Localisation d'un Point :
est-il dans ou hors du triangle ?



Et si on ne connait pas l'orientation du triangle ?

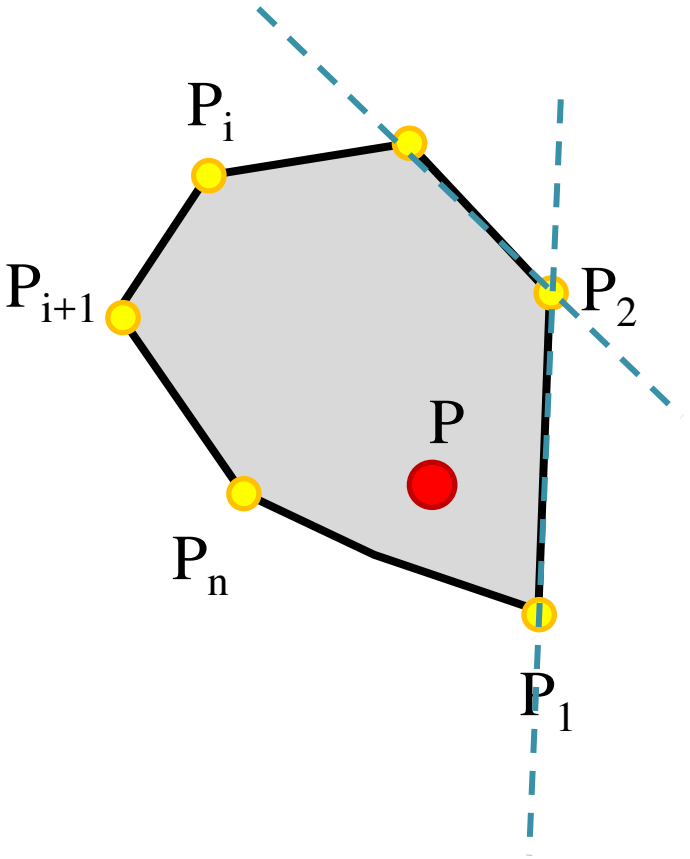
Point dans convexe ?



Le polygone = $\{P_i (x_i,y_i)\}$

Complexité de l'algorithme ?

Point dans convexe ?



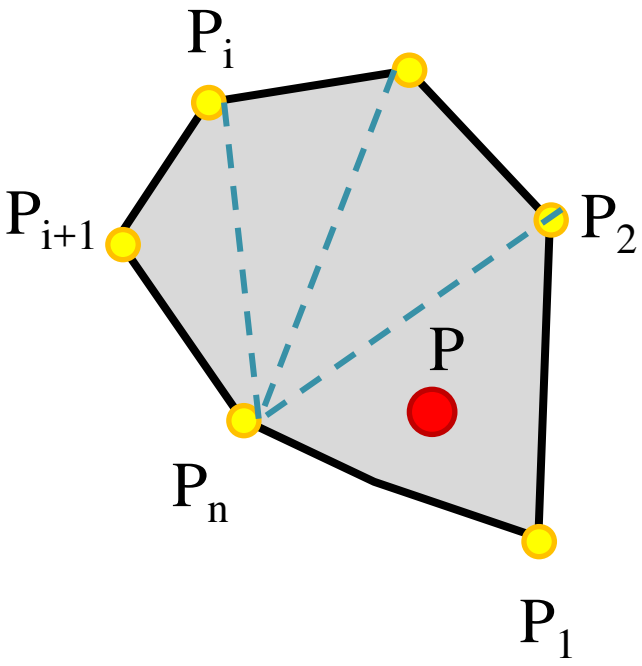
Le polygone = $\{P_i (x_i,y_i)\}$

➔ Parcours des arêtes

P

Complexité de l'algorithme ?

Point dans convexe ?



Le polygone = $\{P_i (x_i,y_i)\}$

➔ Triangulation

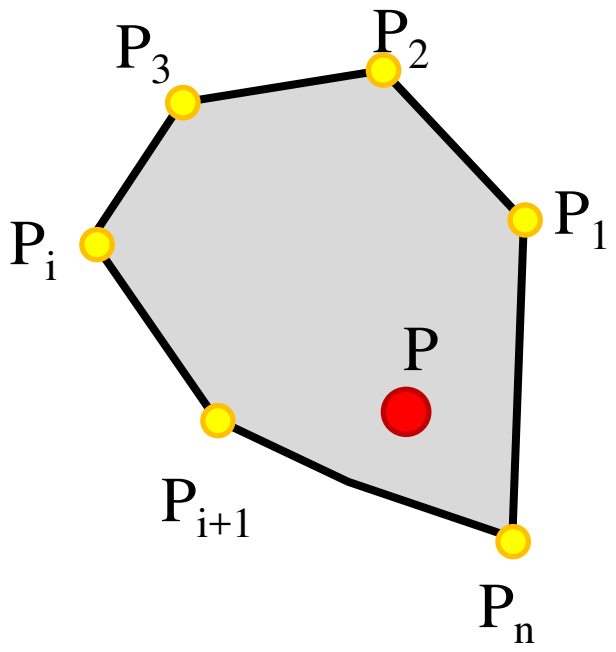
P

EXERCICE : proposez un algorithme en $O(n + k \log(n))$

Complexité de l'algorithme ?

EXERCICE: Point dans convexe ?

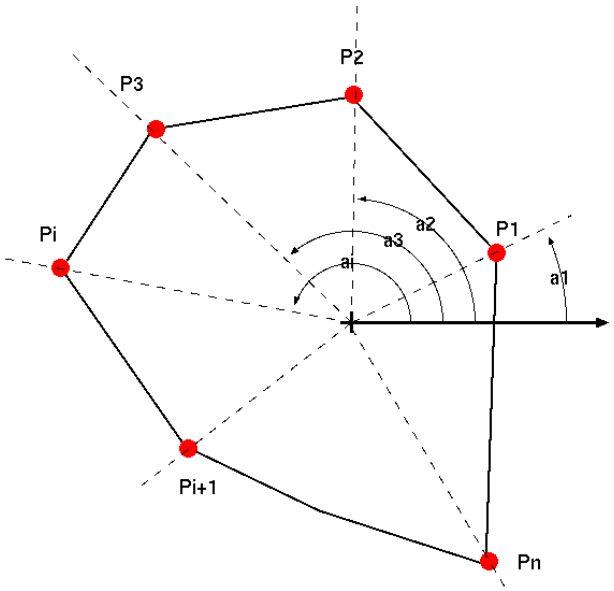
Algorithme efficace pour traiter K points (K grand)

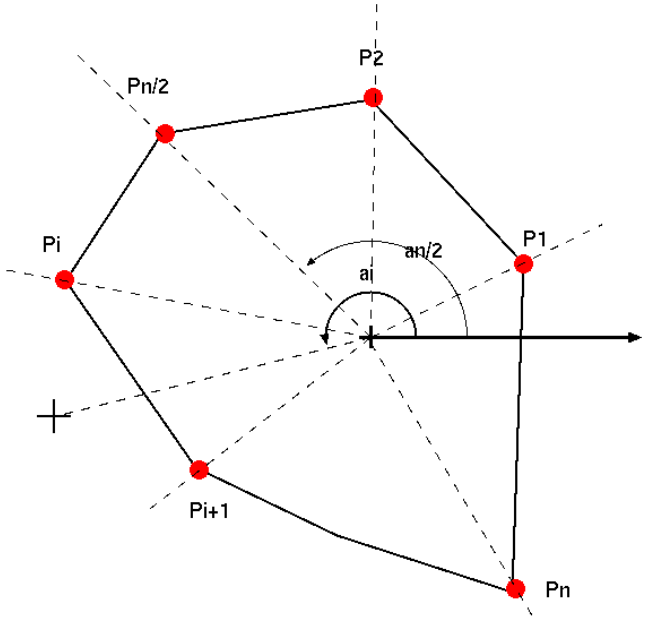


EXERCICE : proposez un algorithme en $O(n + k \log(n))$

EXERCICE: Point dans convexe ?

Algorithme efficace pour traiter K points (K grand)





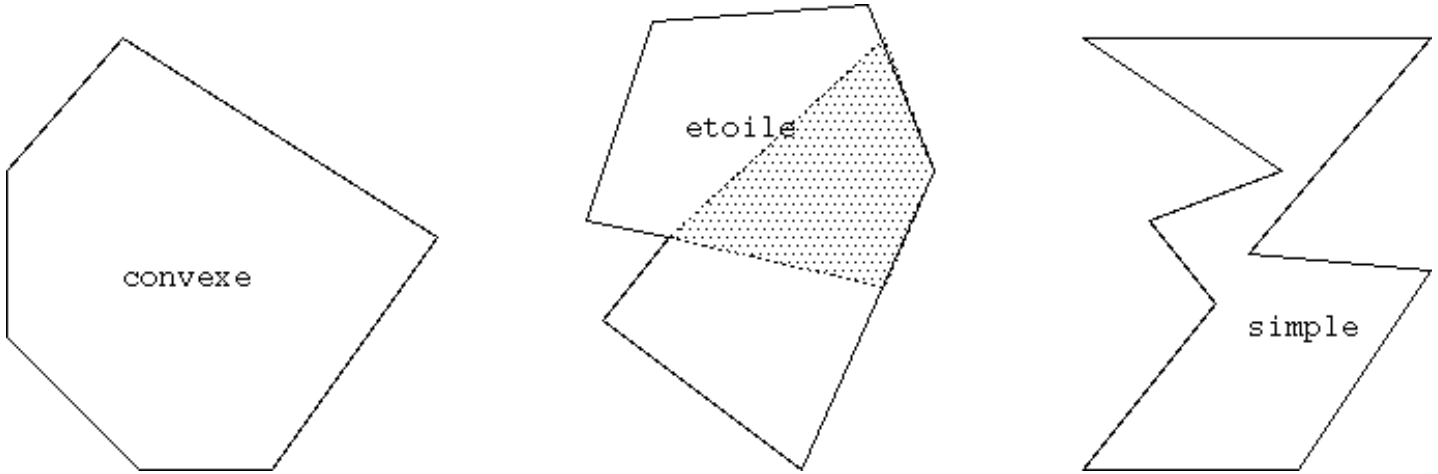
Préparation :
« construction » des secteurs

O(n)

Requête :
accès par dichotomie

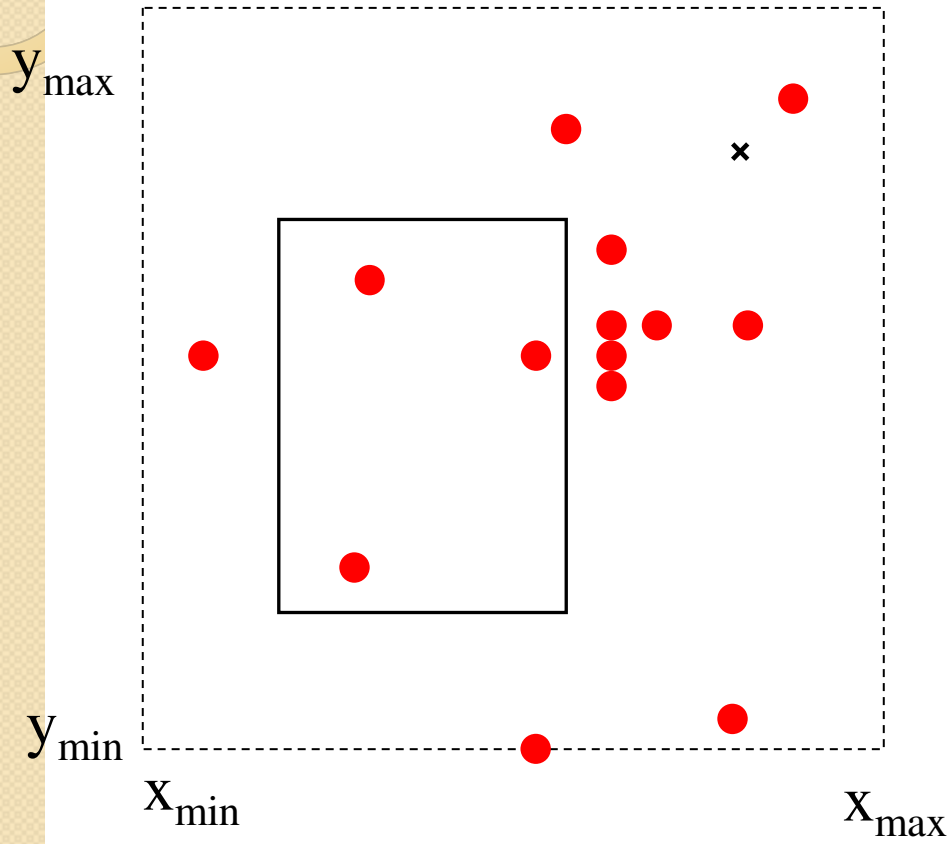
O(log(n))

Point dans polygone ?



Polygone	Temps	Mémoire	Pré-traitement
Simple	O(n)	O(n)	-
Etoilé	O(log(n))	O(n)	O(n)
Convexe	O(log(n))	O(n)	O(n)

Arbres de recherche



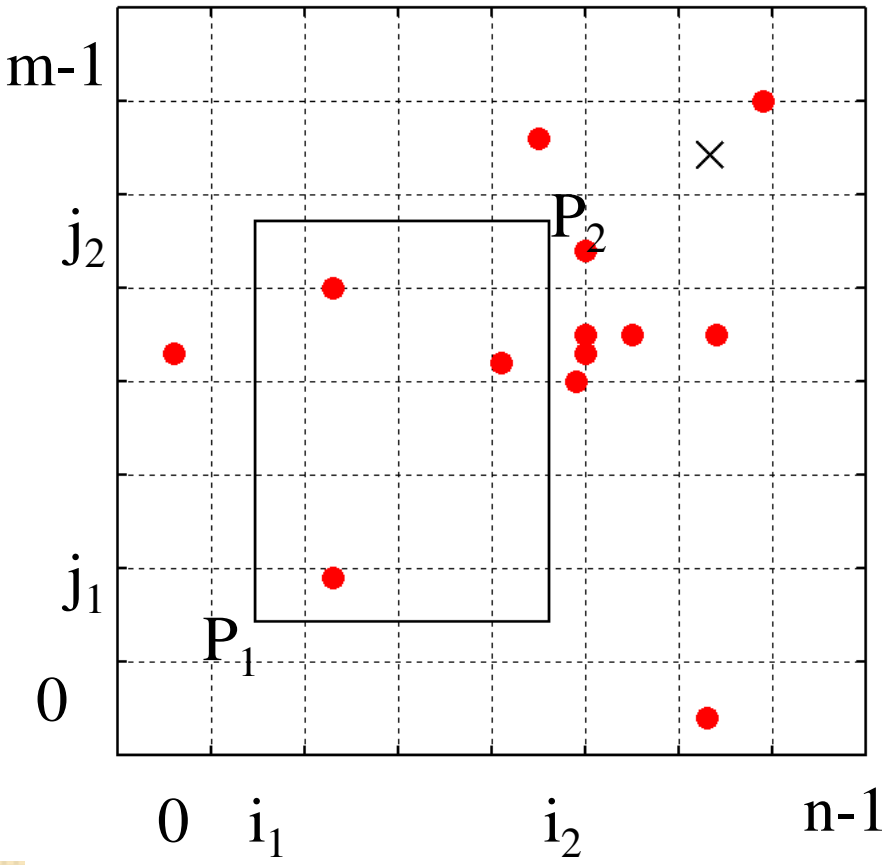
OBJECTIF

Optimiser les requêtes

PRINCIPE

1. Limitation de l'espace
2. Structure de découpage

La grille



Requête boite $B(P_1,P_2)$
$P_1=(x_1,y_1)$, $P_2=(x_2,y_2)$

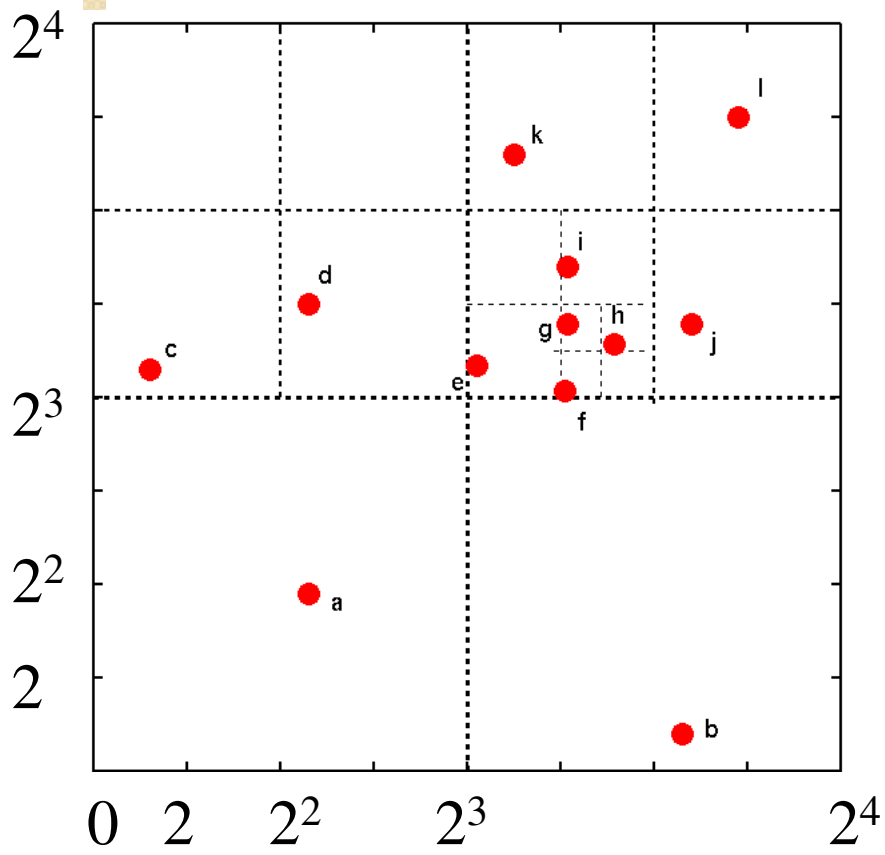
$i_1 = \text{floor}(n*(x_1-x_{min})/(x_{max}-x_{min}))$
$j_1 = \text{floor}(m*(y_1-y_{min})/(y_{max}-y_{min}))$

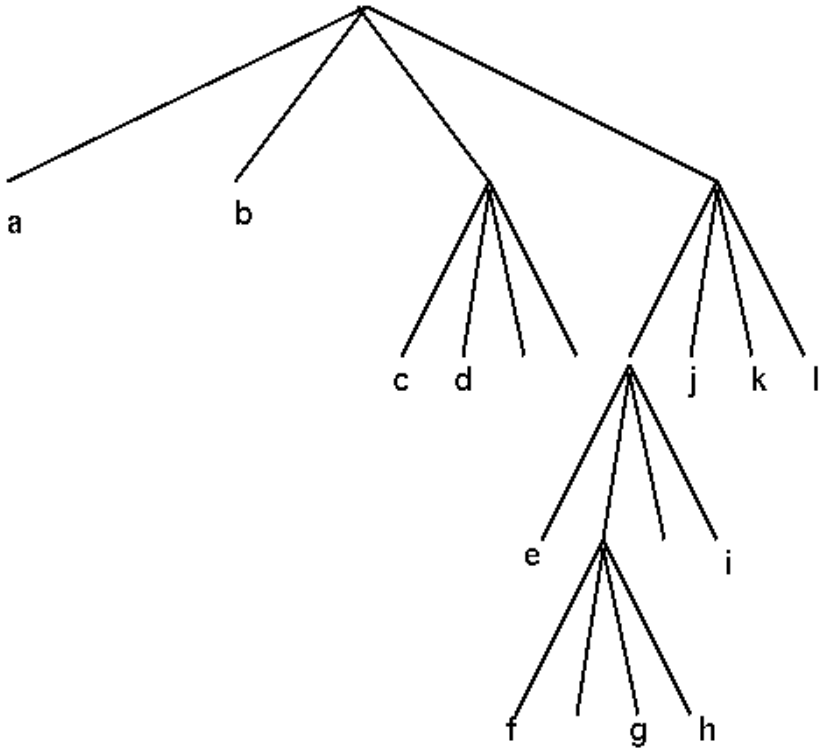
$i_2 = \text{floor}(n*(x_2-x_{min})/(x_{max}-x_{min}))$
$j_2 = \text{floor}(m*(y_2-y_{min})/(y_{max}-y_{min}))$

Cellules de i_1 à i_2 et de j_1 à j_2

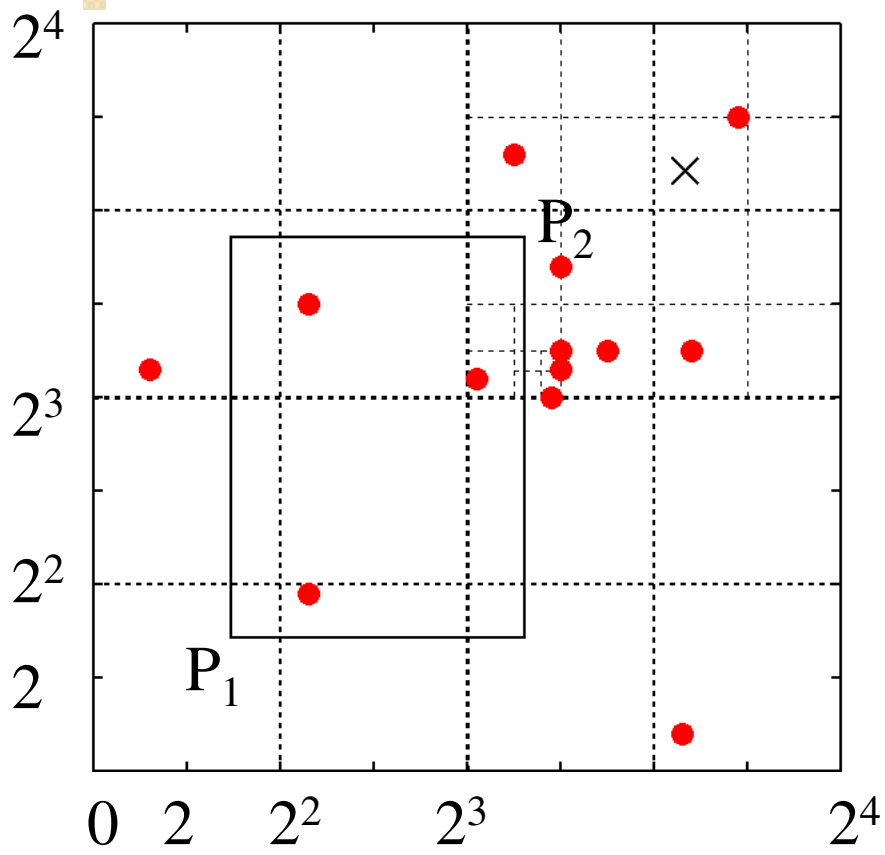
floor() renvoi la partie entière

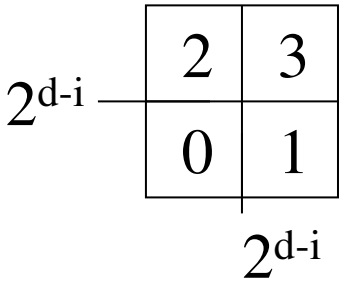
Le Quadtree





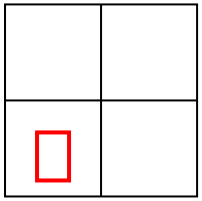
Requêtes sur un quadtree





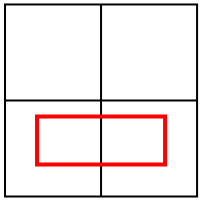
C=Classement(P_1,P_2,d,i)

C=(0,0)



C=(0,1)

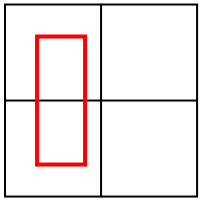
C=(2,3)



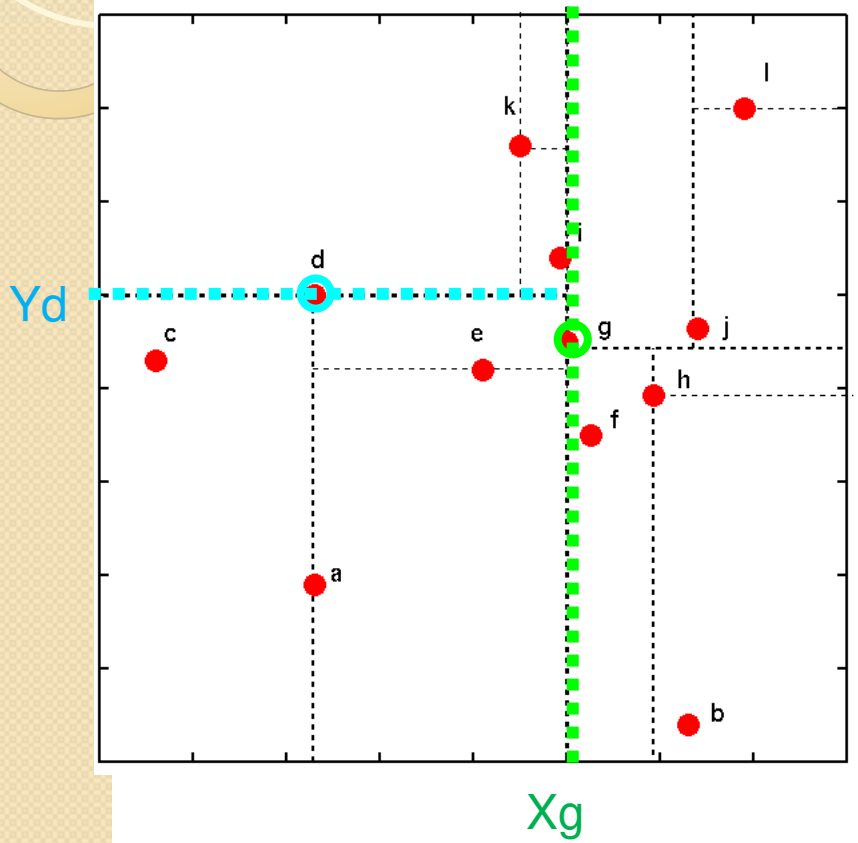
C=(0,2)

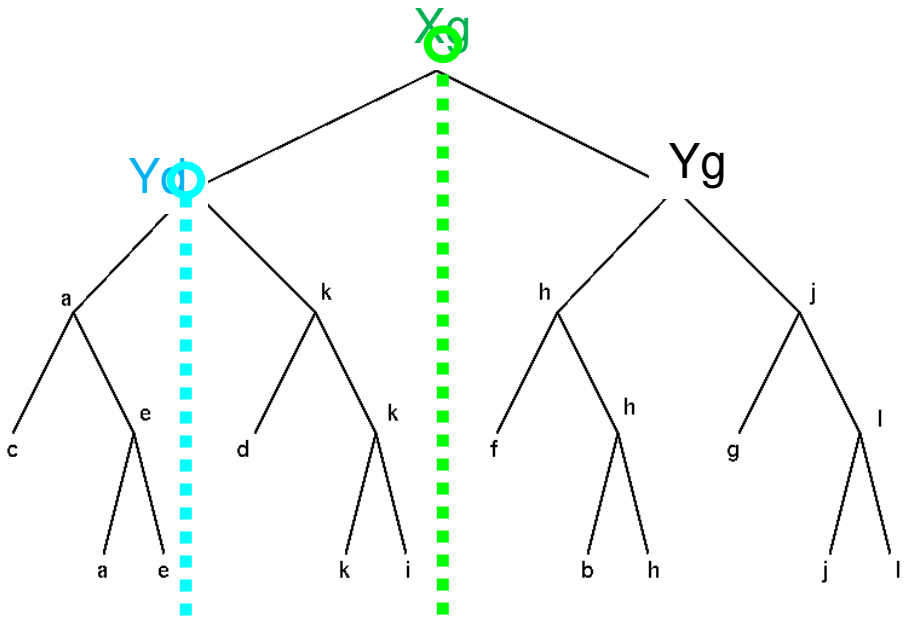
C=(1,3)

…



Le KDtree





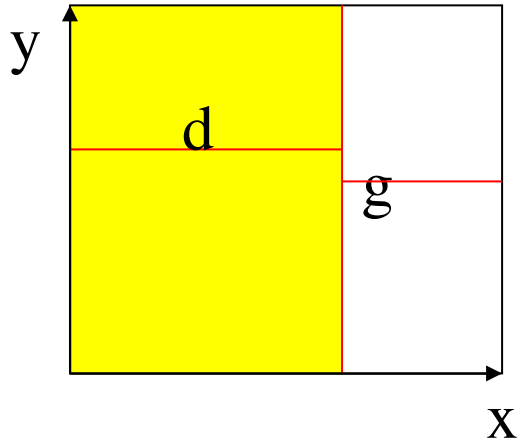
Construction du KDTree

tabX

c	a	d	k	e	i	g	f	h	b	j	l

tabY

b	a	f	e	c	h	g	j	d	i	k	l



tabY1G

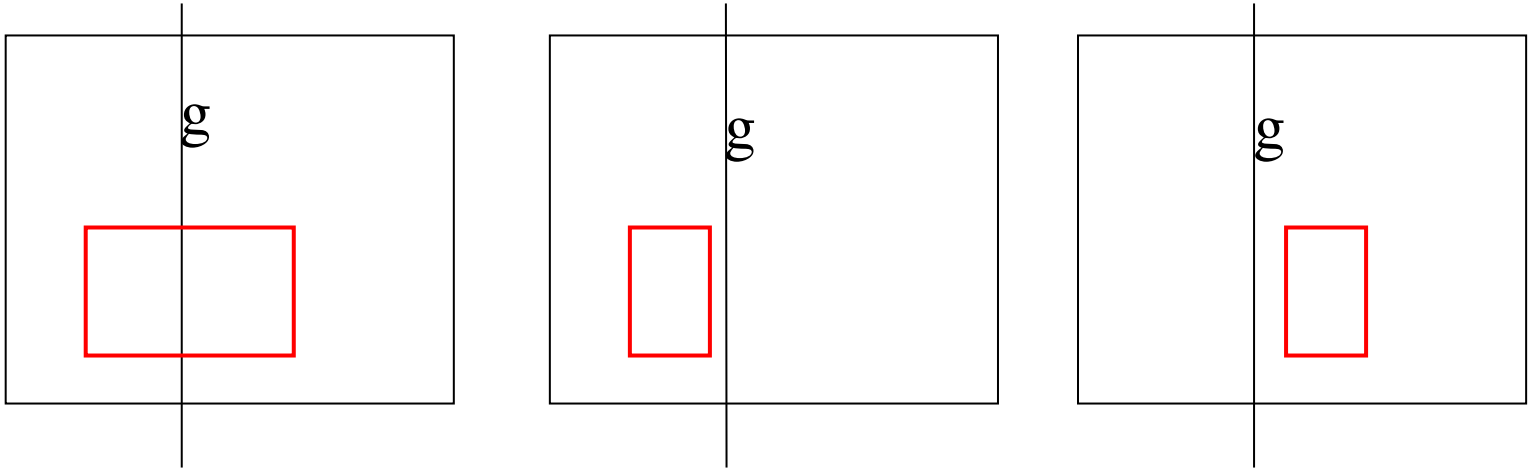
a	e	c	d	i	k

tabY1D

b	f	h	g	j	l

Requêtes sur un KDtree

3 cas de récursion

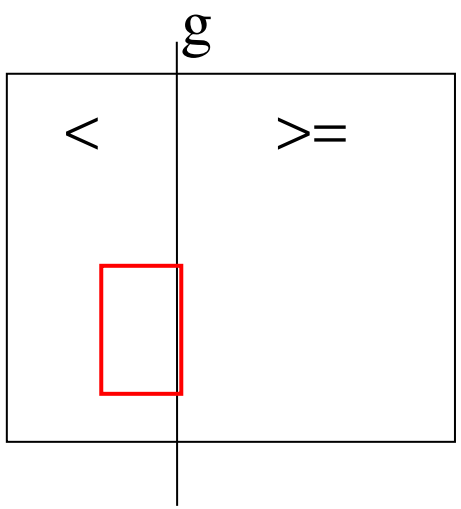


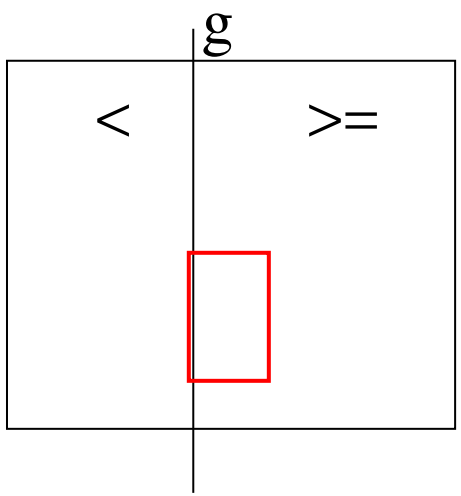
g est la valeur du pivot

Question :

1) Faut-il visiter le SAD
si max(Xbox) = g ?

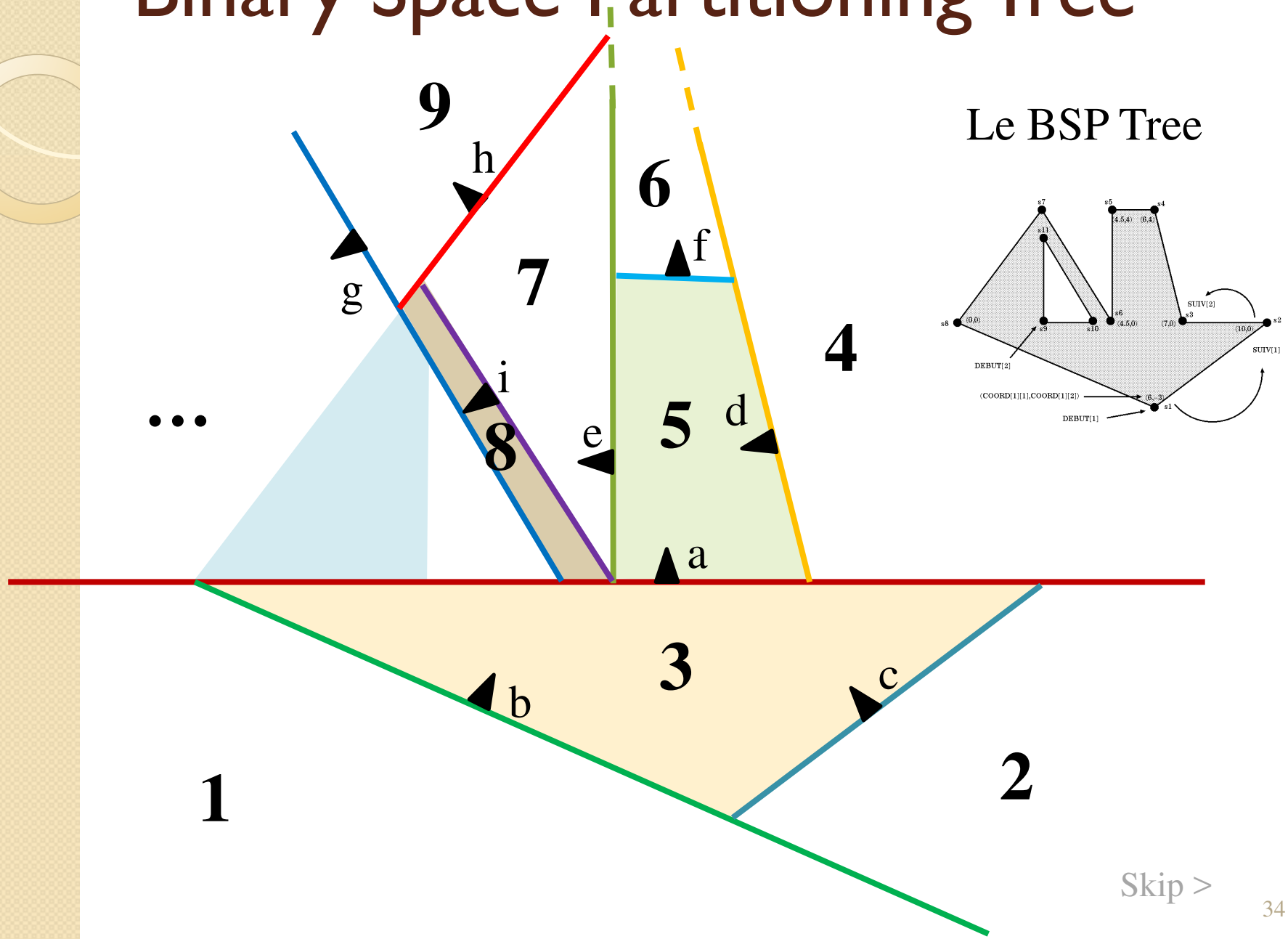
2) Faut-il visiter le SAG
si min(Xbox) = g ?





g est la valeur du pivot

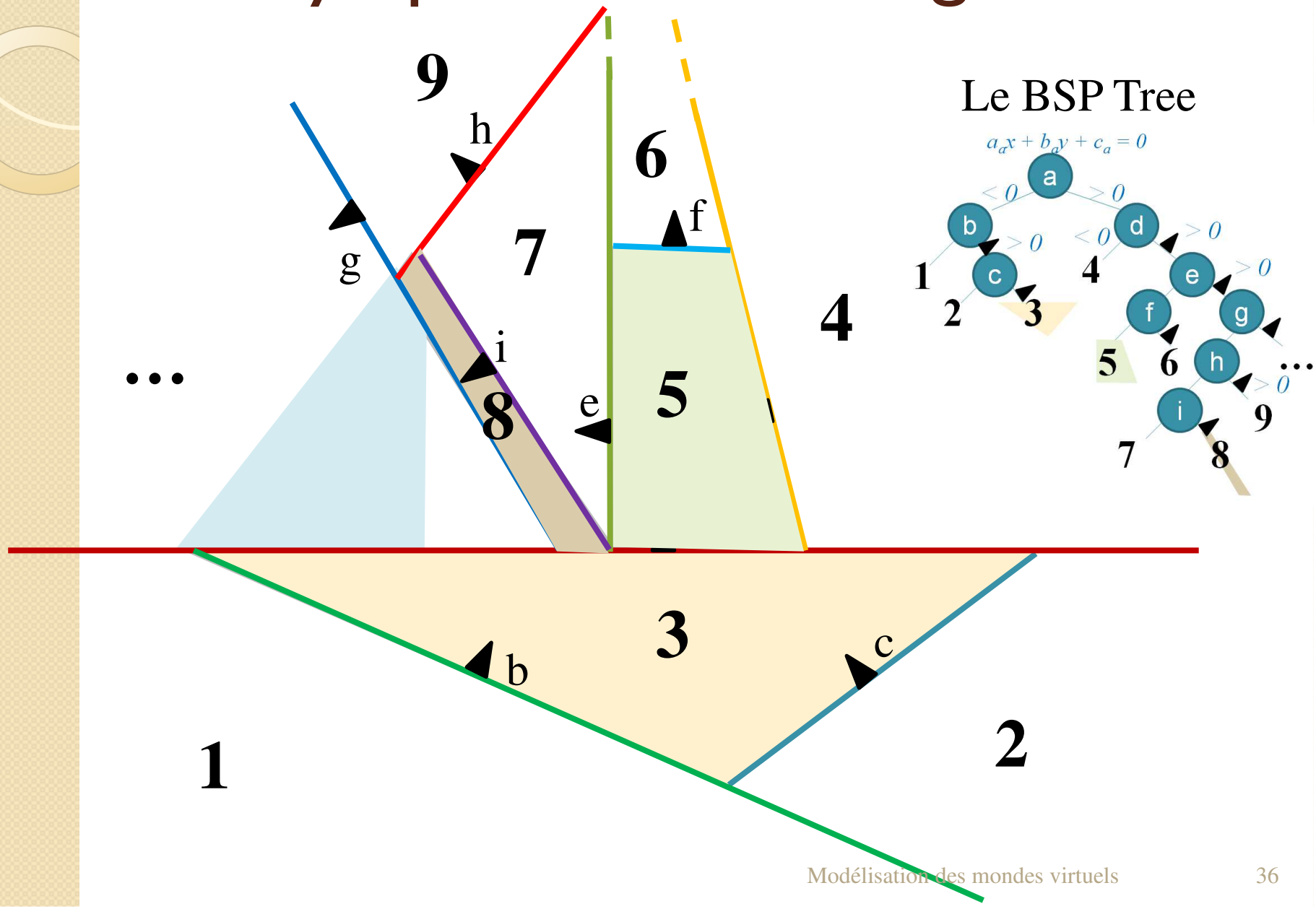
Binary Space Partitioning Tree



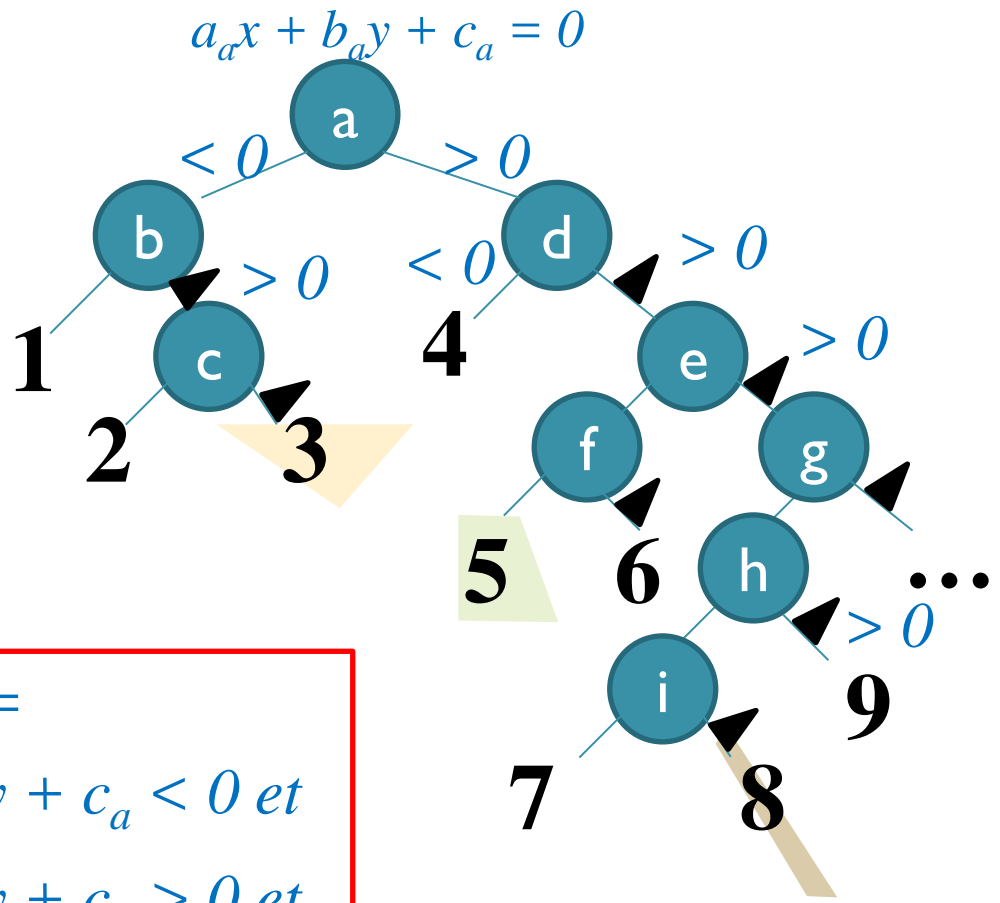
Le BSP Tree

Skip >

Binary Space Partitioning Tree



Binary Space Partitioning Tree



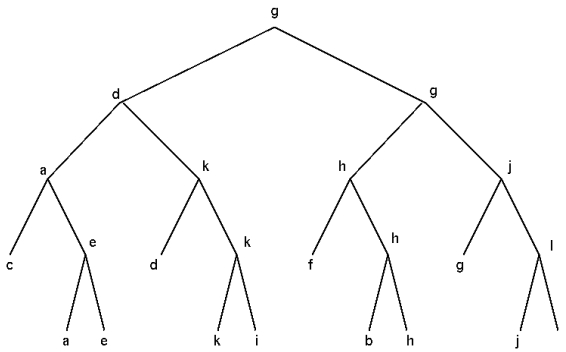
3 =

$a_a x + b_a y + c_a < 0$ et

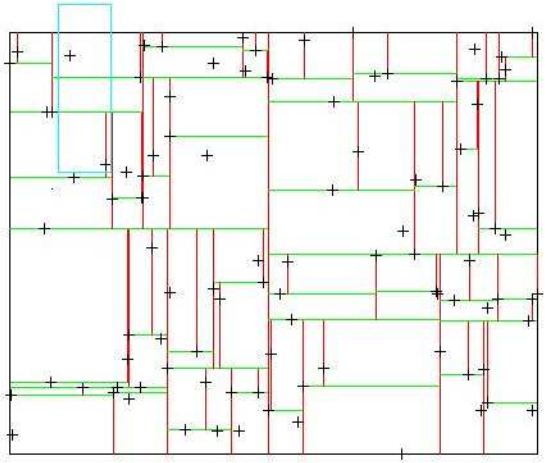
$a_b x + b_b y + c_b > 0$ et

$a_c x + b_c y + c_c > 0$

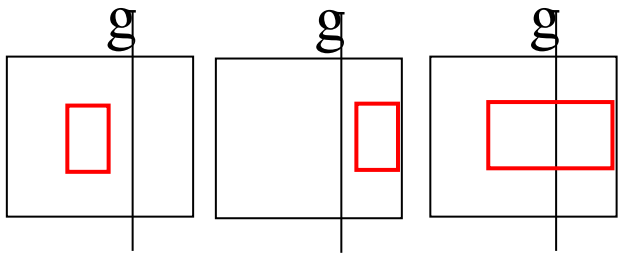
TP : KDTree

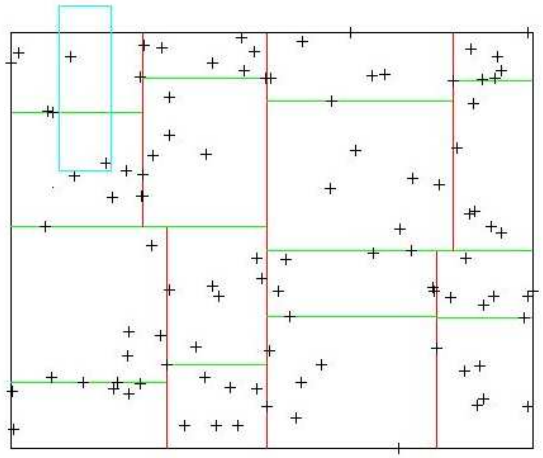


1. Manipulation de KDTree ; utilisation de : *[KDT]=buildKDTree(XY);*

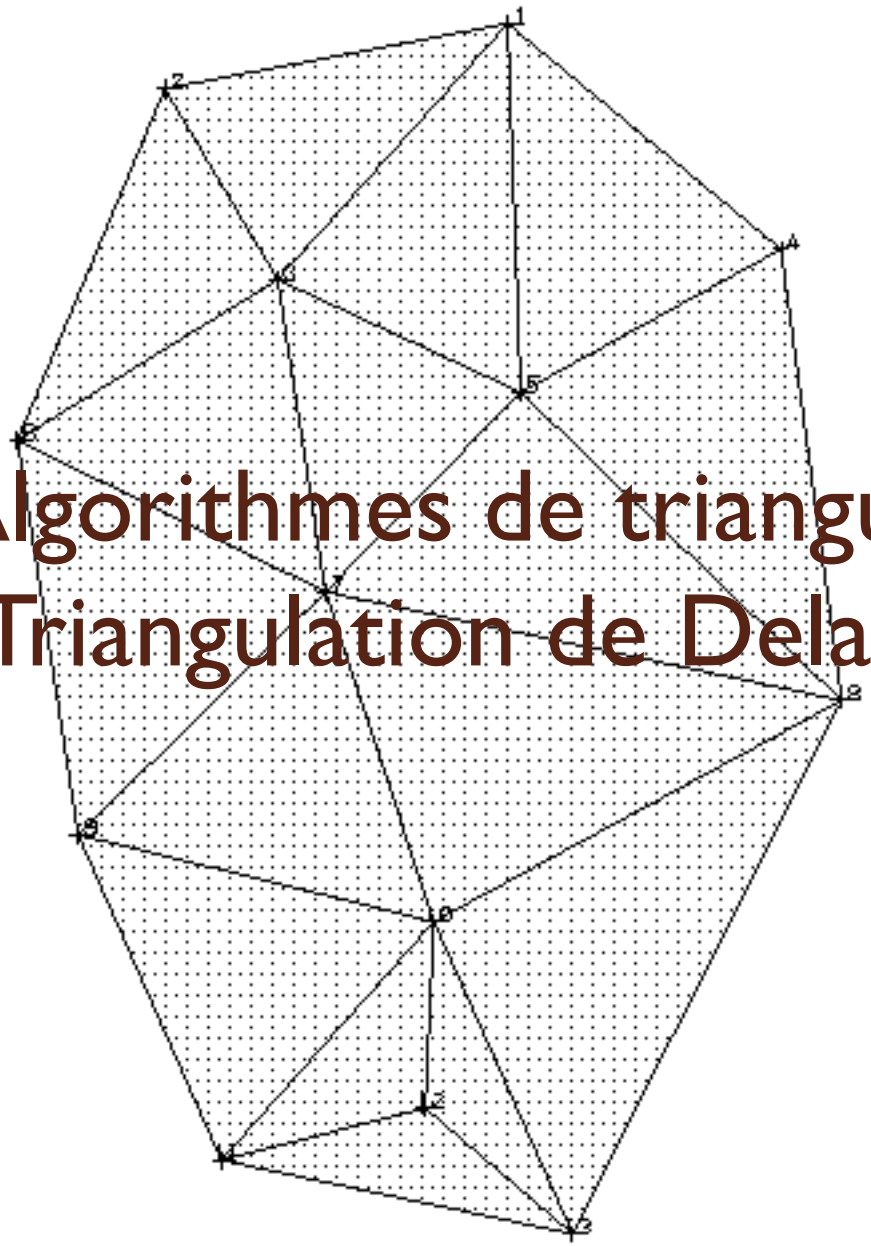


2. Programmation de la fonction *[]=findInBox(KDT,Box)*





3. Manipulation de KDTree avec des clusters > 1



3. Algorithmes de triangulations
Triangulation de Delaunay

Cardinaux (cas 2D)

Relation d'Euler adaptée : $t - a + s = 1$

Triangulation : $a' + 2a'' = 3t$

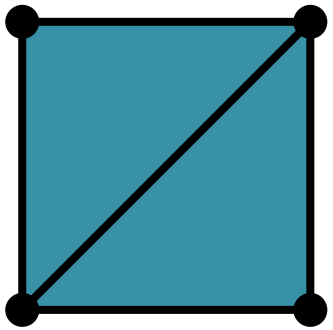
$$t = 2s - a' - 2$$

a' le nombre d'arêtes de frontière

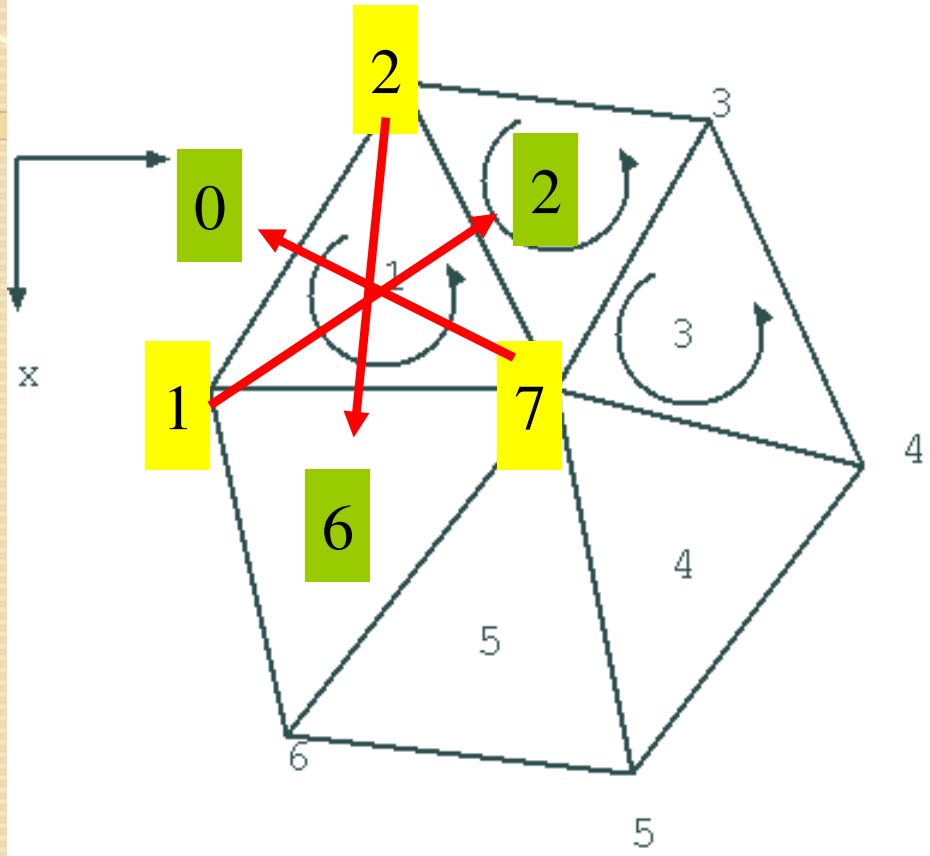
2-5+4=1

4+2=3*2

2=8-4-2



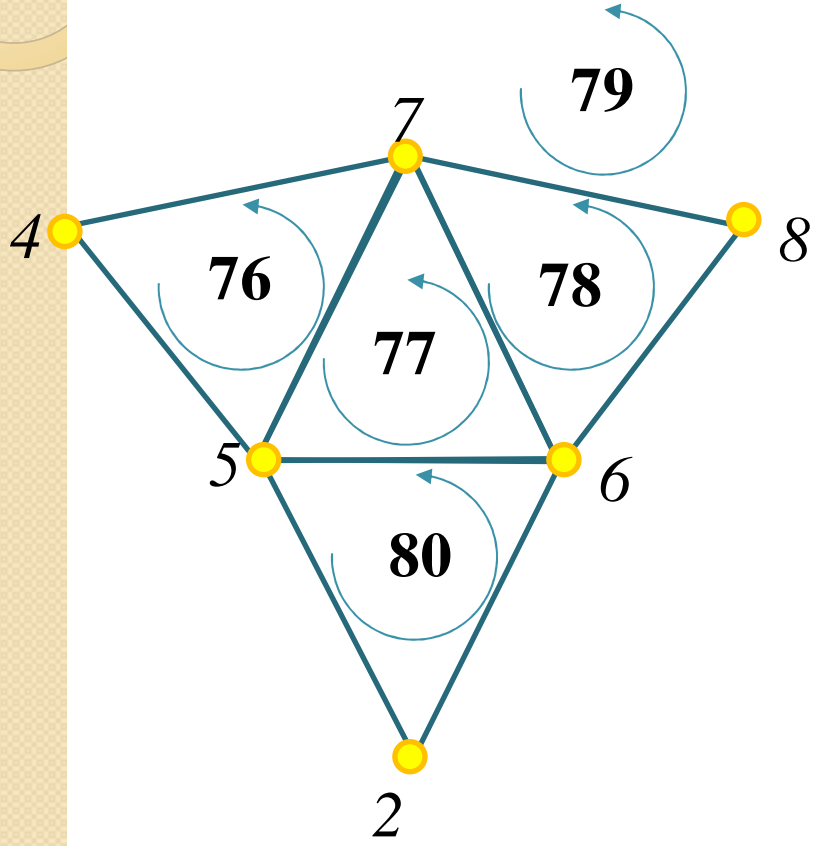
Structure



	ITRNOE	ITRTRI
1	1,7,2	2,0,6
2	3,2,7	1,3,0
3	7,4,3	0,2,4
4	7,5,4	0,3,5
	...	...

	COORD	NOETRI
1	1.5,1.2	6
2	-.5,2.5	2
3	-.2,4.5	2
	...	

Structure



NOETRI

7	77

ITRNOE

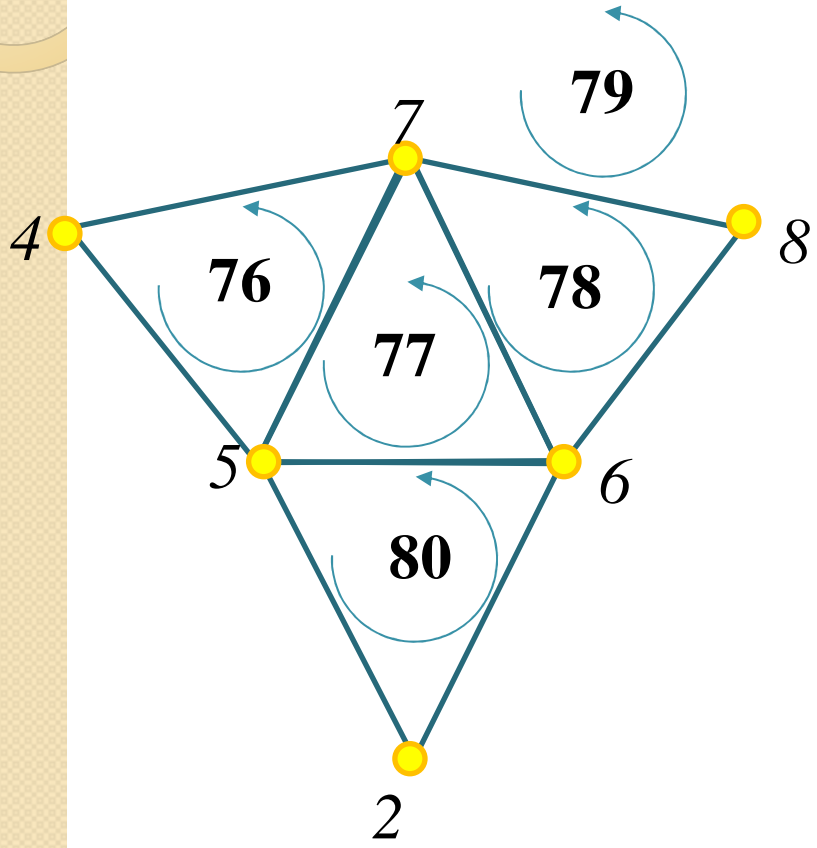
77			
78			
79			
80			

ITRTRI

Algo de parcours

```
iel0=NOETRI(S); iel= iel0;
do {
…
…
…
…
} while (iel != iel0)
```

Structure



NOETRI

7	77

ITRNOE

77	*6*	*7*	*5*
78	*6*	*8*	*7*
79	*7*	*8*	*?*
80	*2*	*6*	*5*

ITRTRI

76	**80**	**78**
79	**77**	**?**
?	**?**	**77**
77	**?**	**?**

Algo de parcours

```
iel0=NOETRI(S); iel= iel0;
do {
…
…
…
…
} while (iel != iel0)
```

Structure

NOETRI

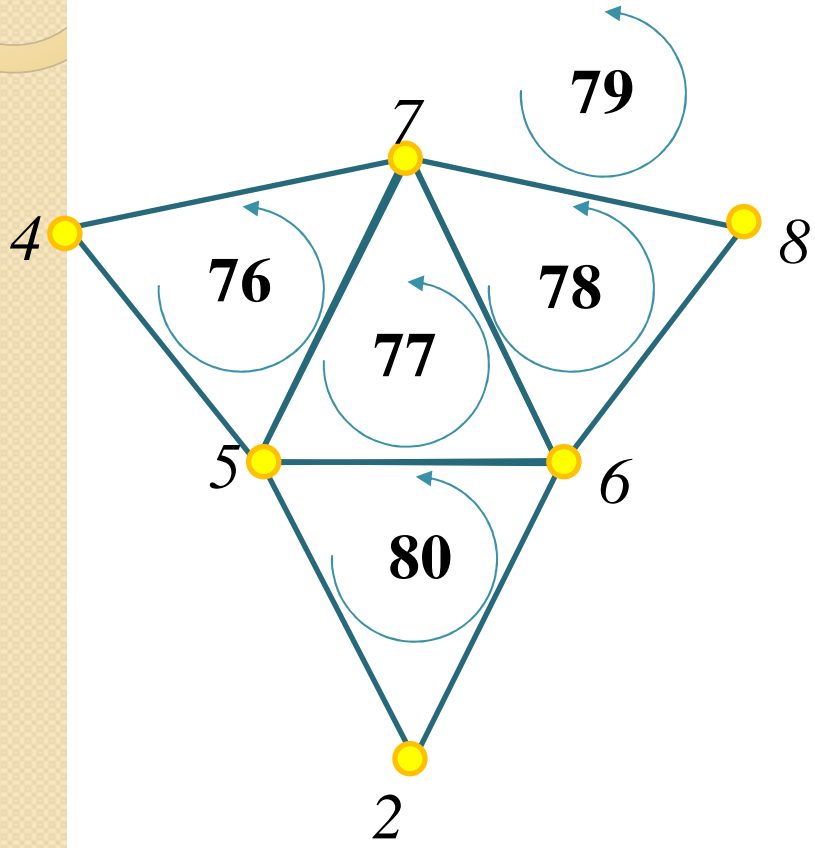
7	77

ITRNOE

77	*6*	*7*	*5*
78	*6*	*8*	*7*
79	*7*	*8*	*?*
80	*2*	*6*	*5*

ITRTRI

76	**80**	**78**
79	**77**	**?**
?	**?**	**77**
77	**?**	**?**

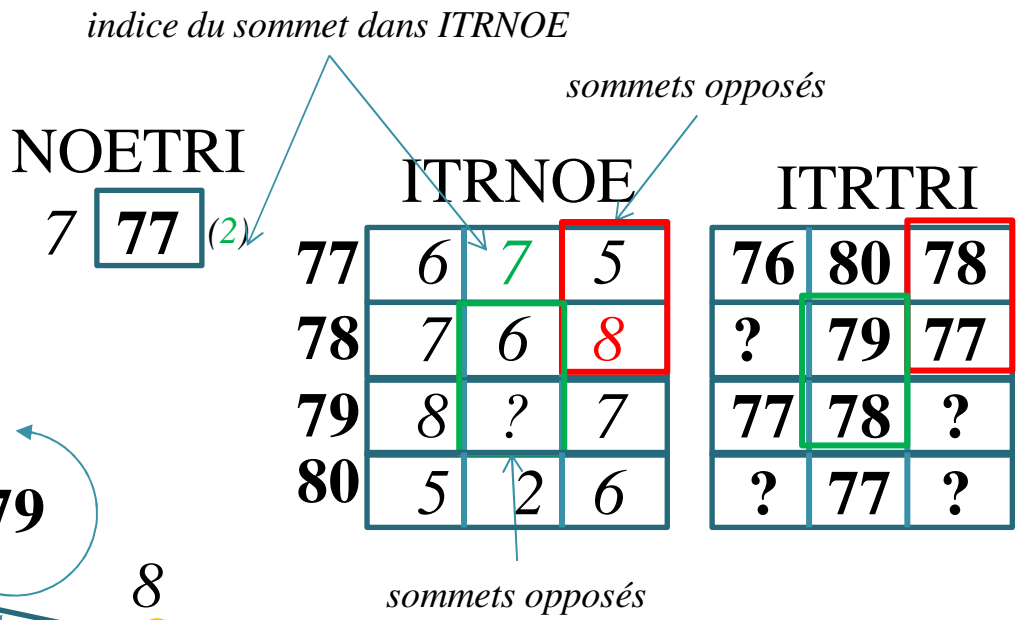


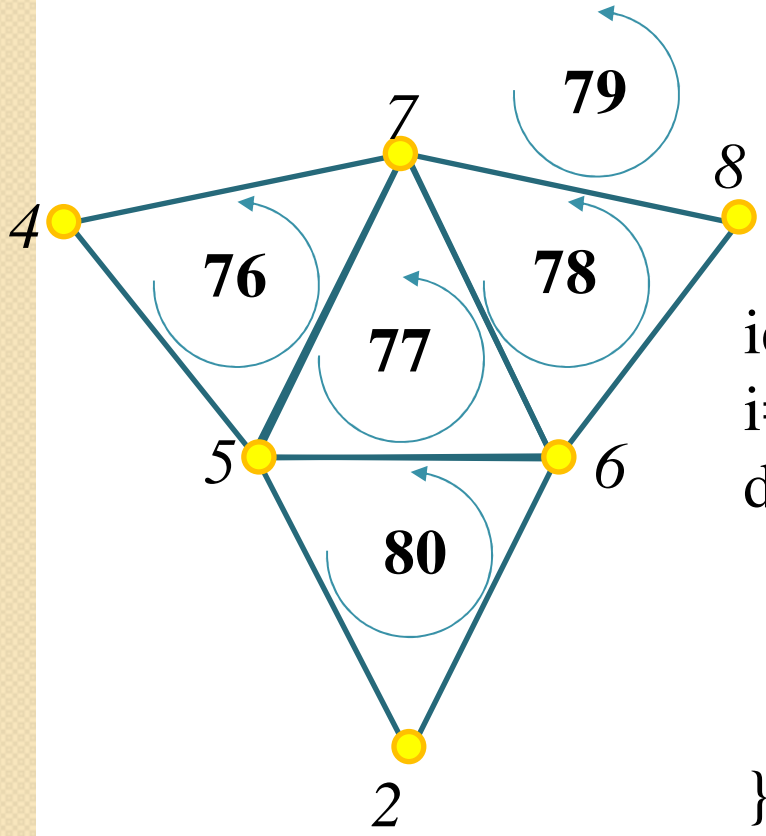
Algo de parcours

faire tourner pour S=7 :

```
iel0=NOETRI(S); iel= iel0;
do {
  i=1;
  while(ITRNOE(iel,i)!=S) i=i+1;
  i=modulo(i,3)+1
  iel=ITRTRI(iel,i)
} while (iel != iel0)
```

Structure Optimisée

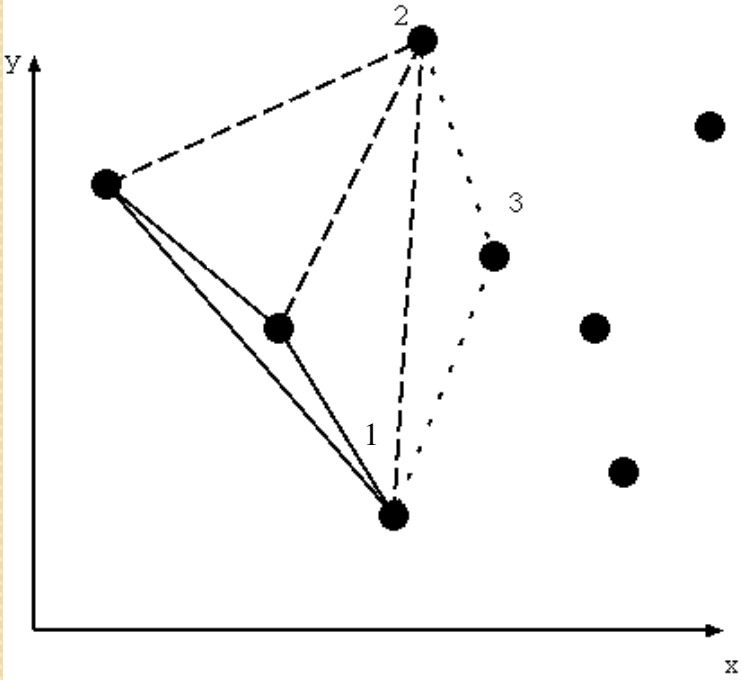




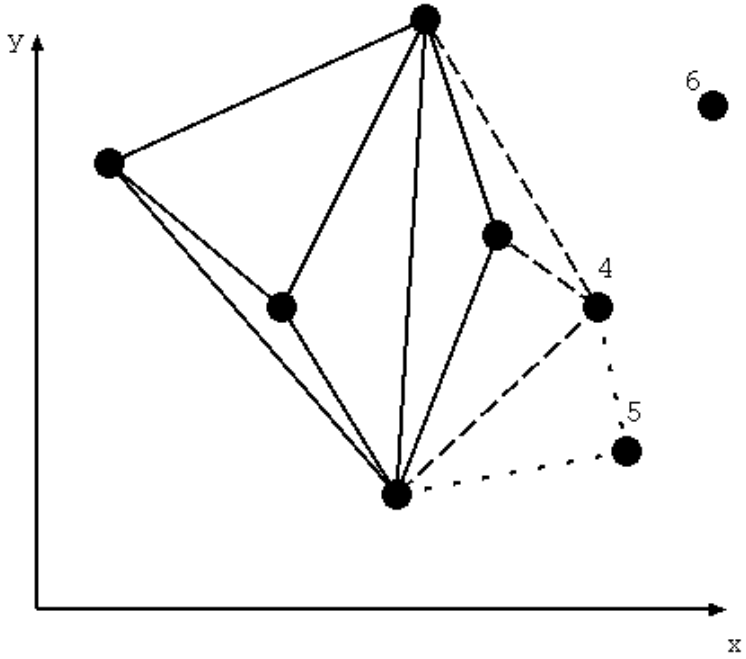
Algo de parcours

```
iel0=NOETRI(S);          iel=77,
i=I(S);iel= iel0; i=(2)  i=2
do {
    i=modulo(i,3)+1      i=3      i=2
    [iel]=ITRTRI(iel,i)  iel=78   iel=79
    i=modulo(i,3)+1      i=1      i=3
} while (iel != iel0)
```

Triangulation par balayage

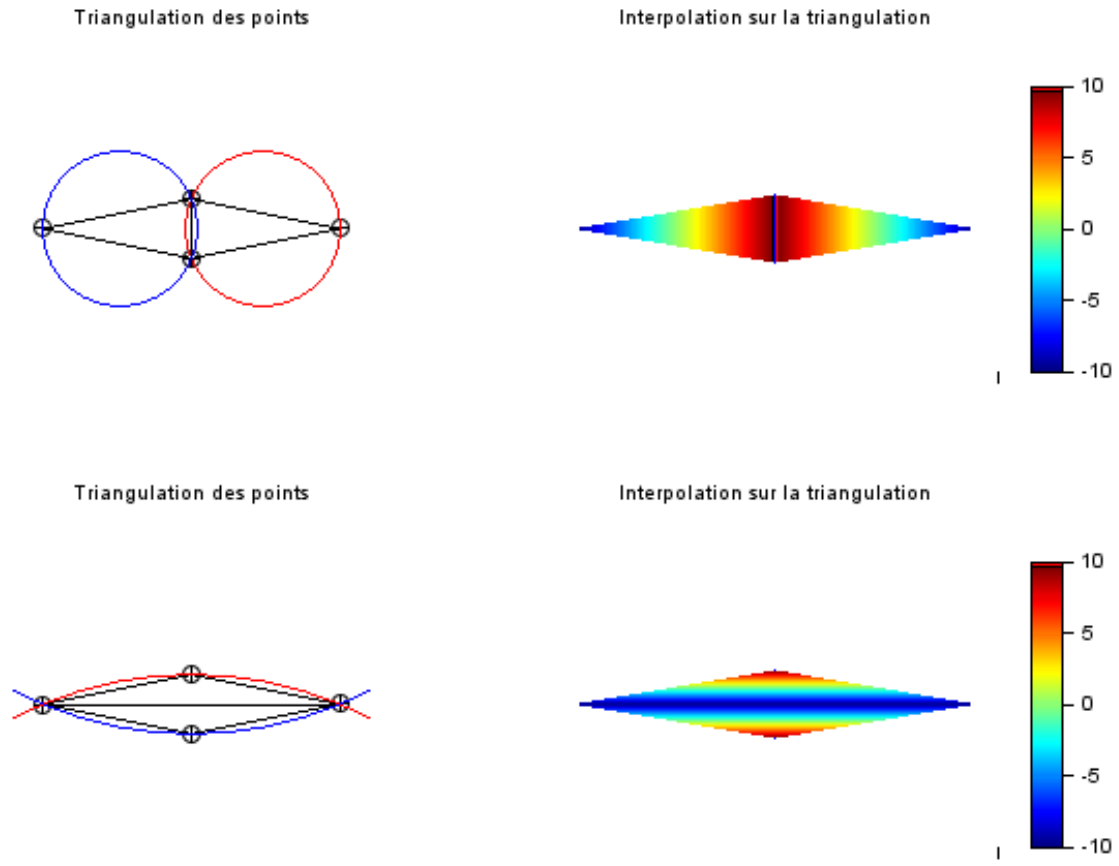


Tri en X :
$O(n*\log_2(n))$

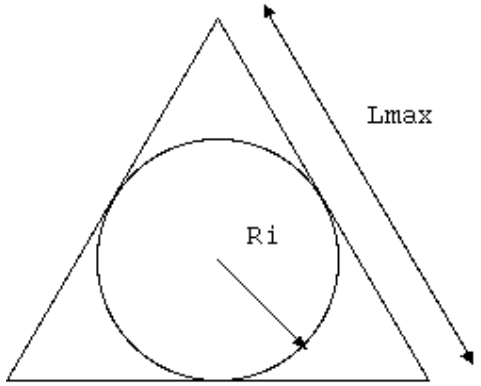


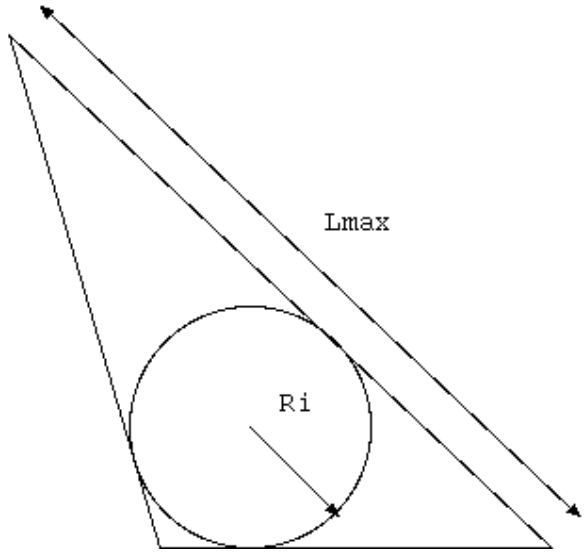
Balayage en X :
$O(n)$

Qualité des triangulations



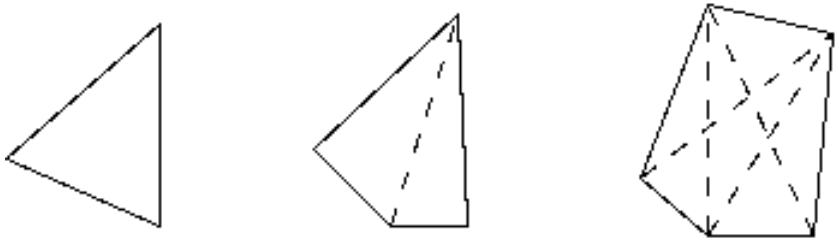
Qualité des triangulations

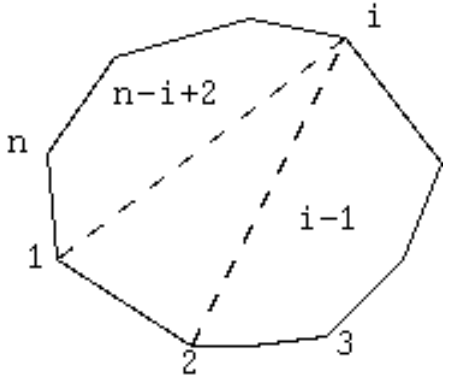




Coefficients de qualité : $c_1 R_i / L_{max}$, l_{min} / L_{max}, $c_3 L_{max} / R_c$

Nombre de triangulations



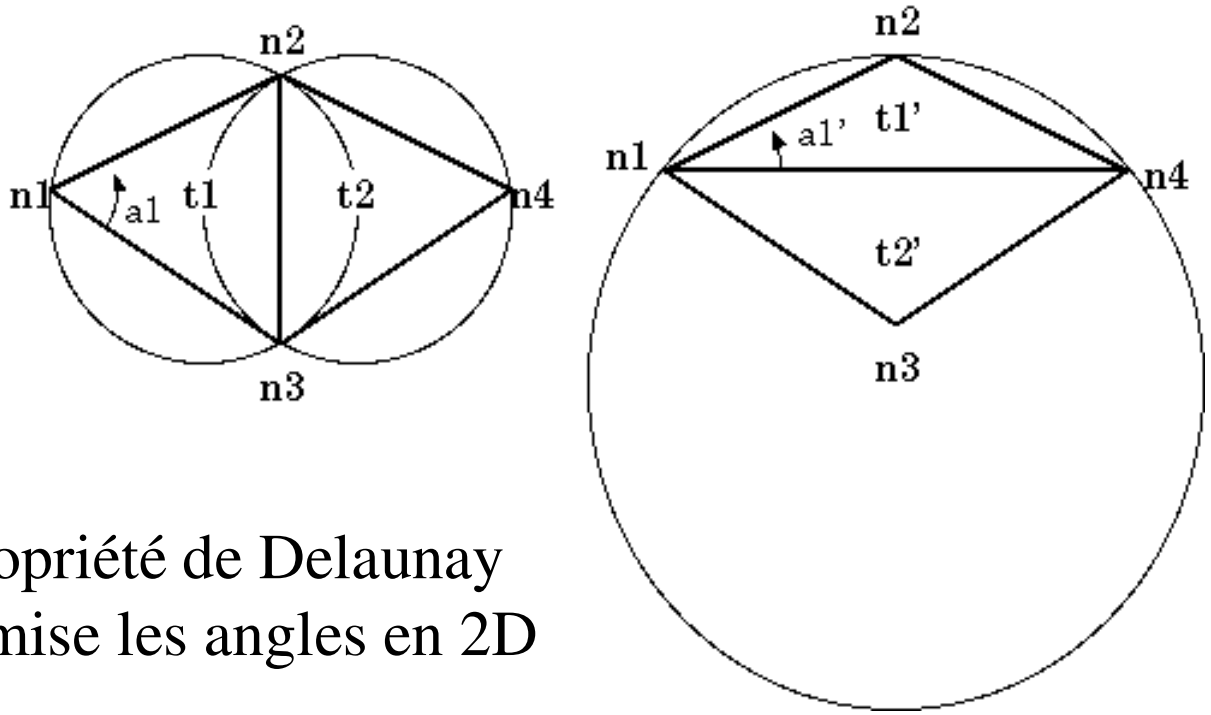


$$C_n = \sum_{i=3}^{n} C_{i-1} * C_{n-i+2}$$

$$C_{14} = 208012$$

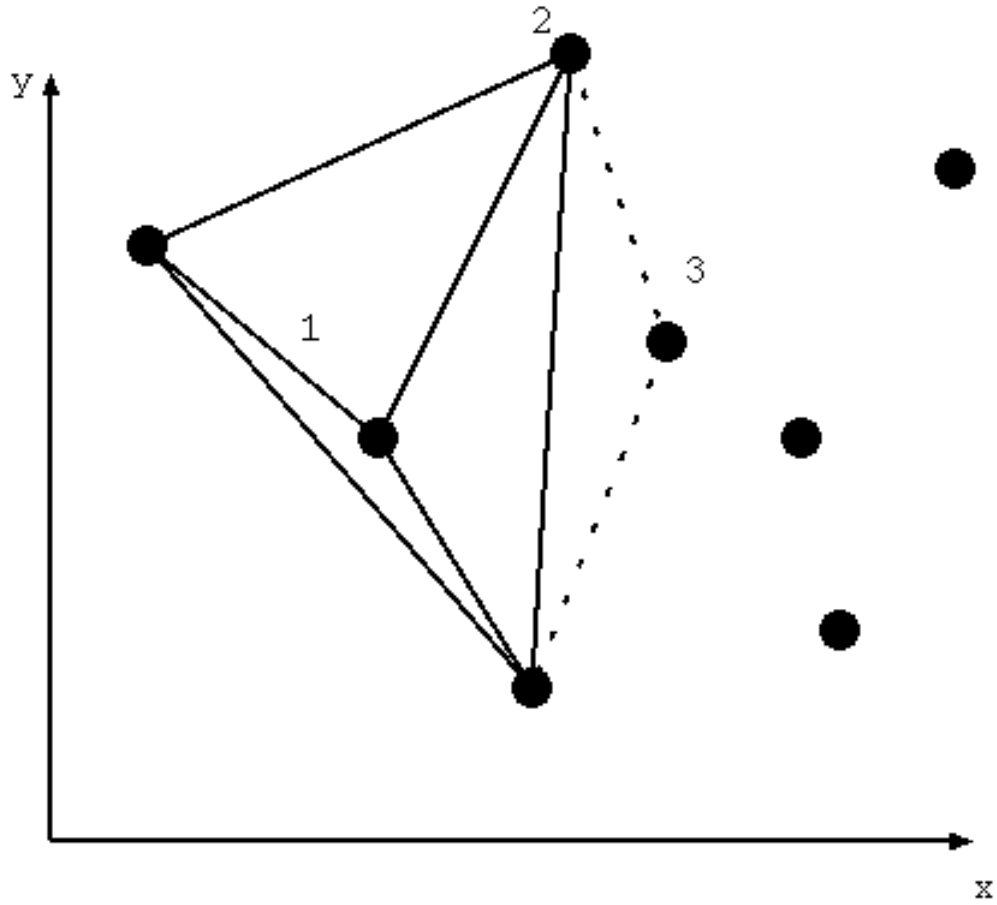
Propriété de Delaunay

Soit T une triangulation s'appuyant sur les points de N.
T est de Delaunay ssi : $\forall t \in T, \forall n \in N, n \notin B(t)$

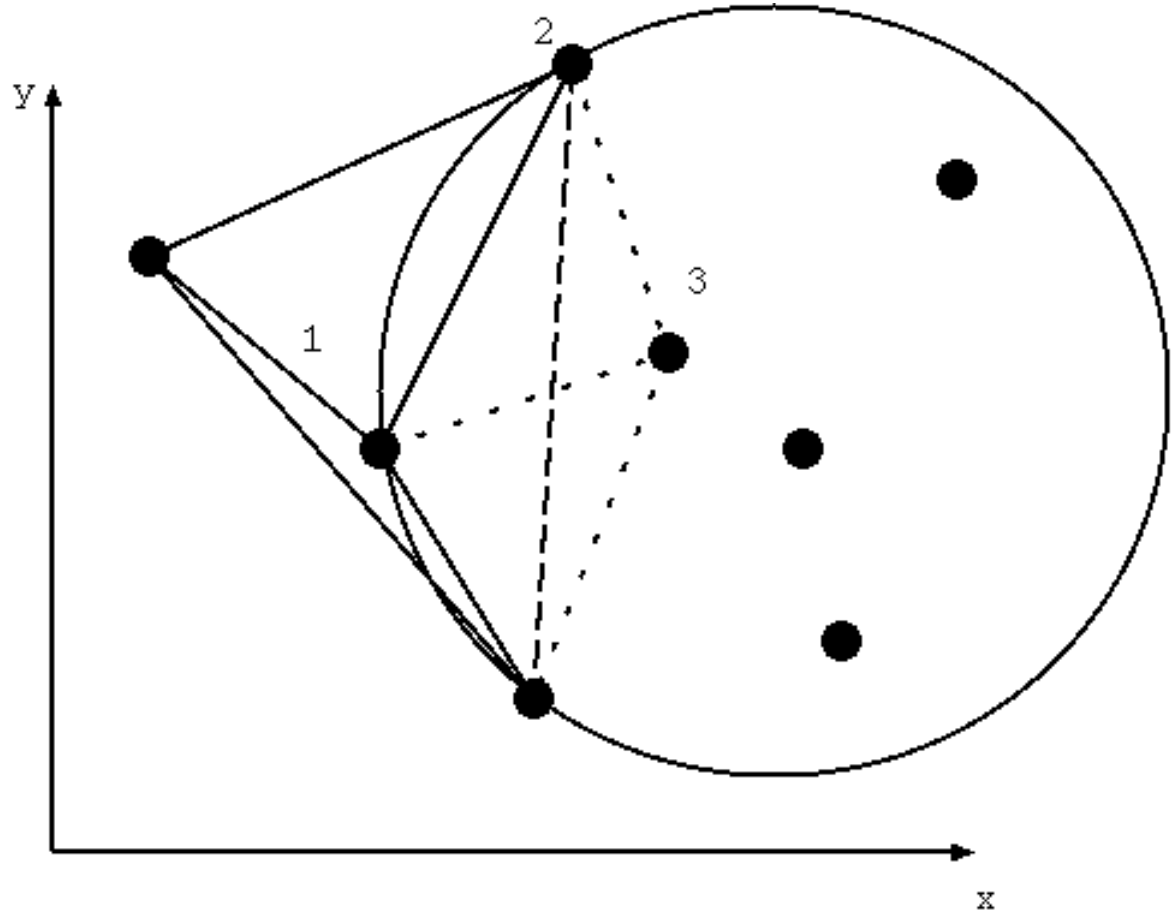


La propriété de Delaunay maximise les angles en 2D

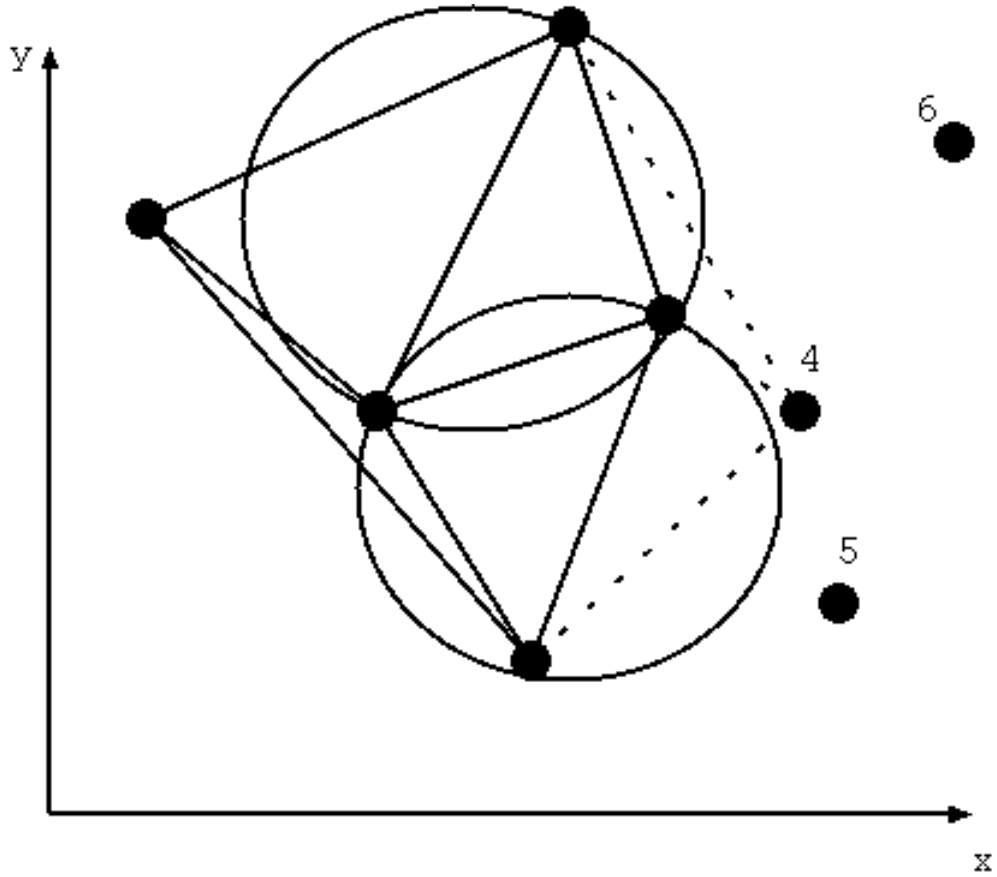
Delaunay par balayage



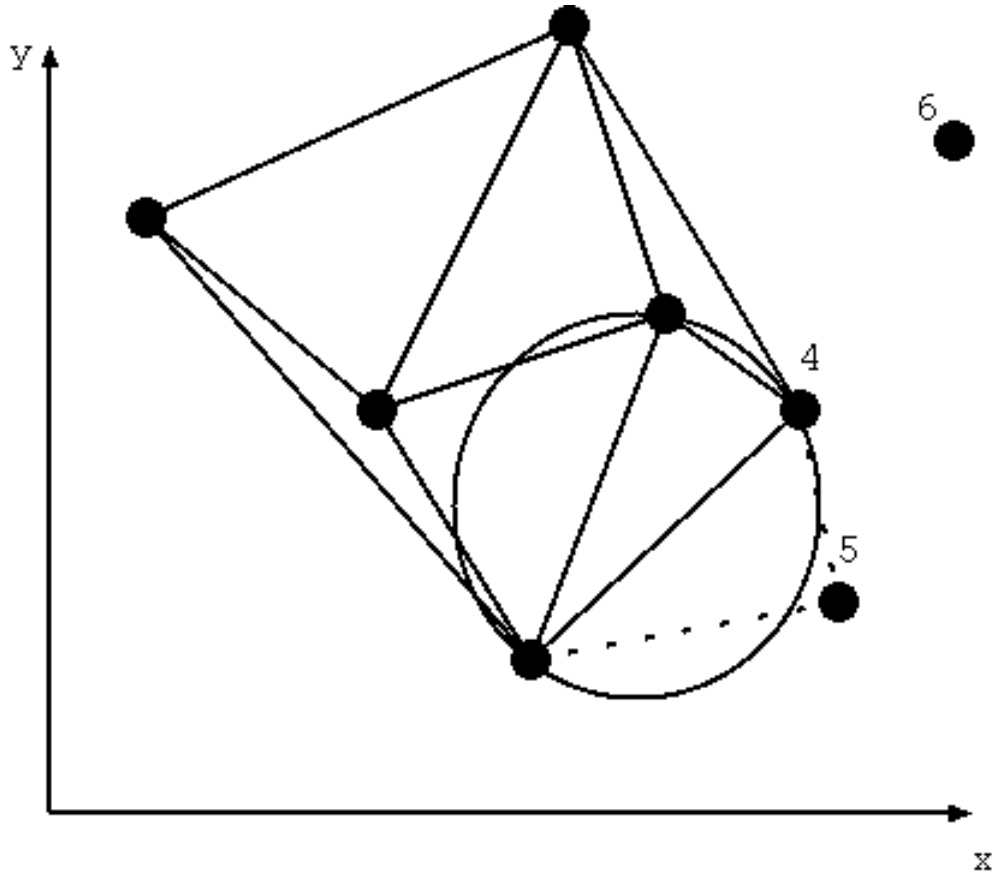
Delaunay par balayage



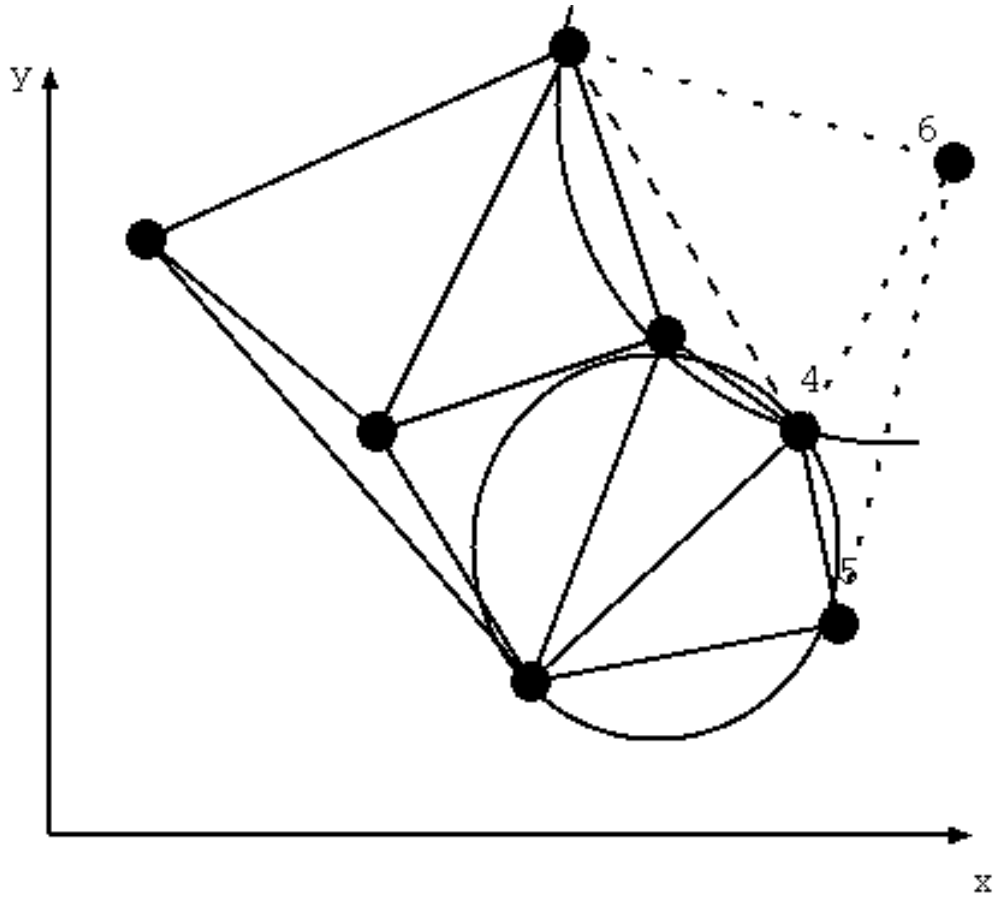
Delaunay par balayage



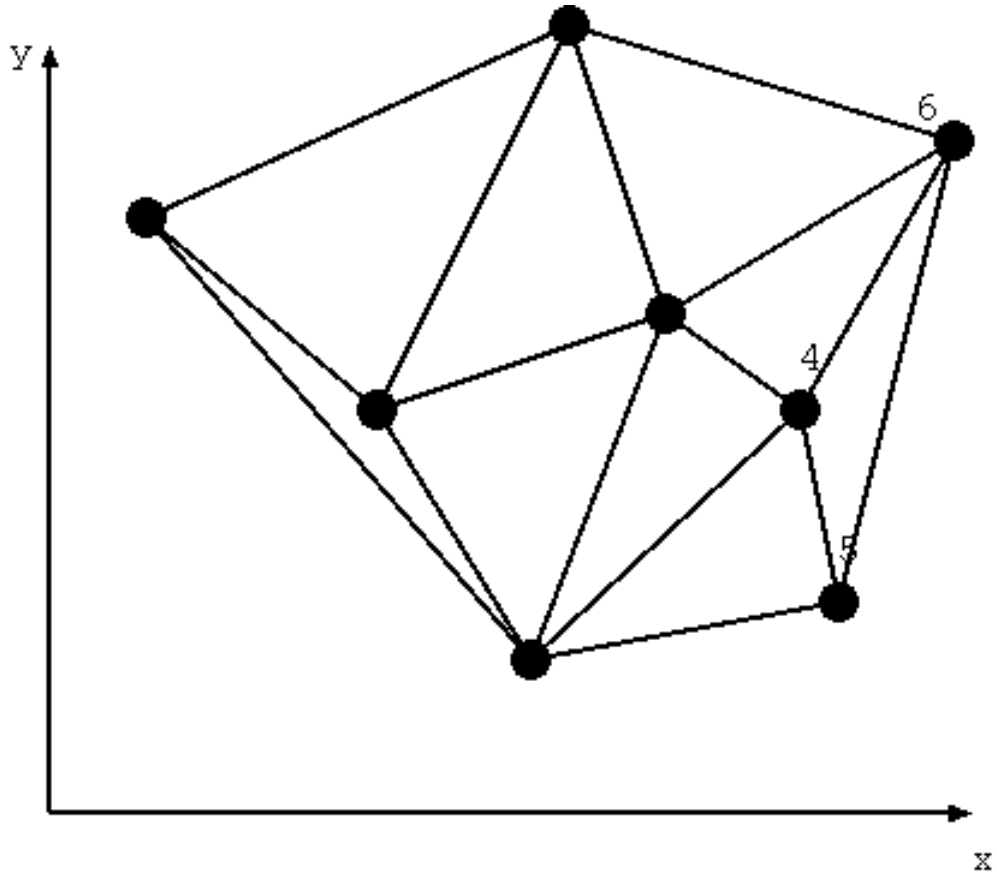
Delaunay par balayage



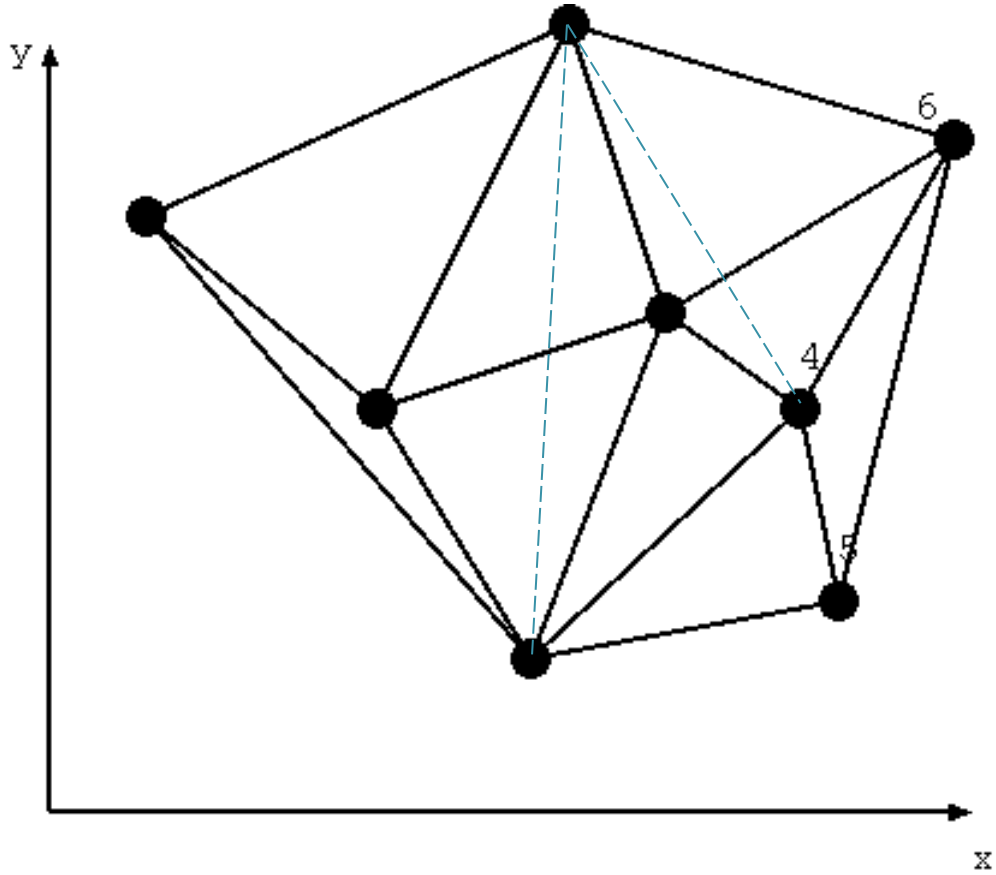
Delaunay par balayage



Delaunay par balayage

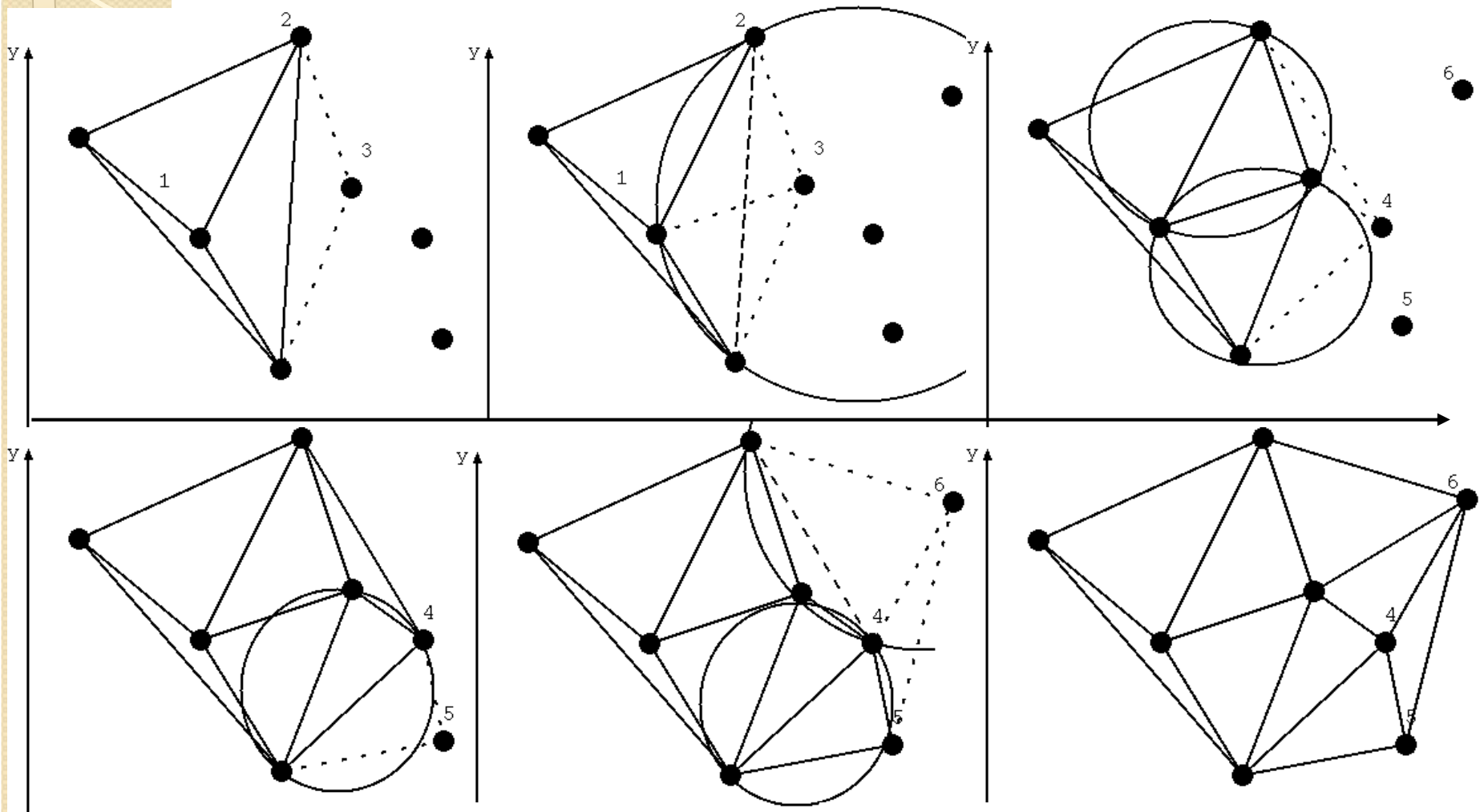


Delaunay par balayage



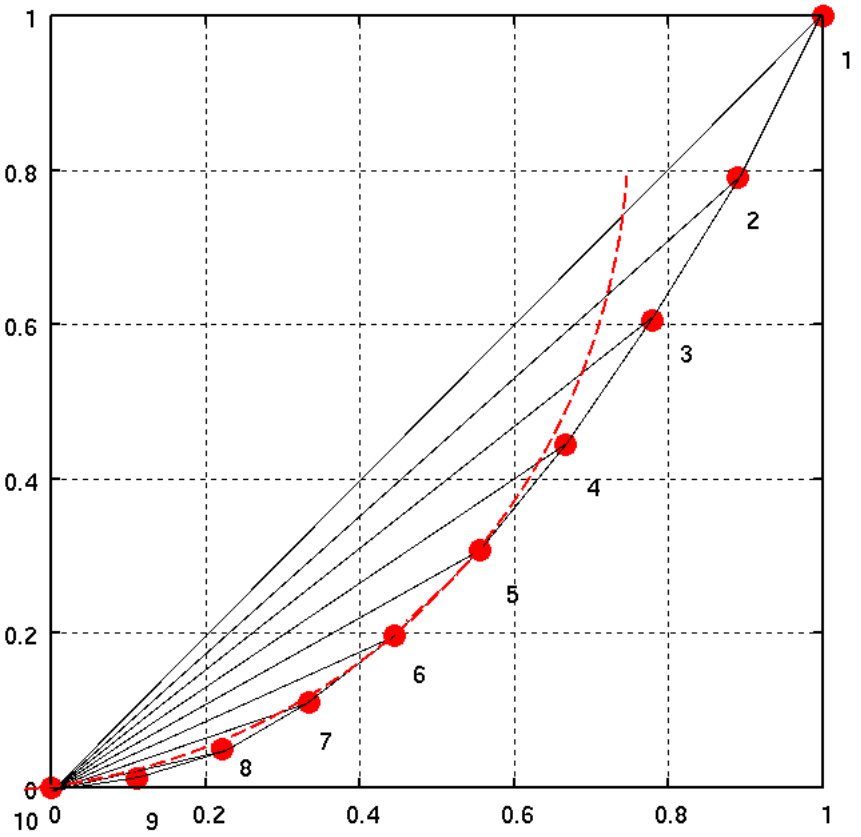
Delaunay par balayage

Complexité pire cas ?



1) Tri en X : $O(n*\log_2(n))$ — 2) Balayage en X : $O(n^2)$

Complexité pire-cas



Les points d'une parabole constituent un pire cas pour un balayage en x décroissant:

$O(n^2)$

C'est en revanche un « meilleur cas » pour un balayage en x croissant:

$O(n)$

Algorithme incrémental

Ajouter_nœud(n,T)

Pour tout t deT

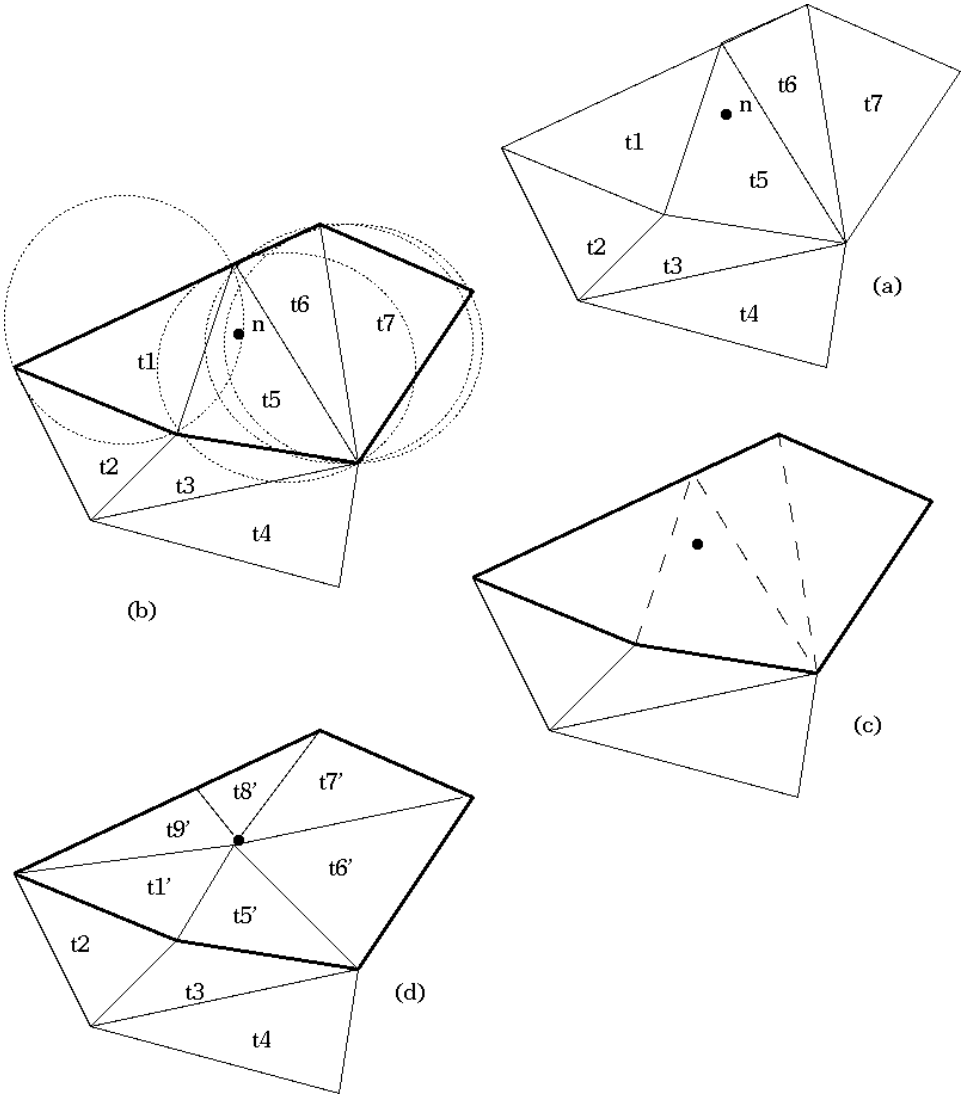
    Si n dans B(t), TD=TD+{t}

A=Frontière de TD

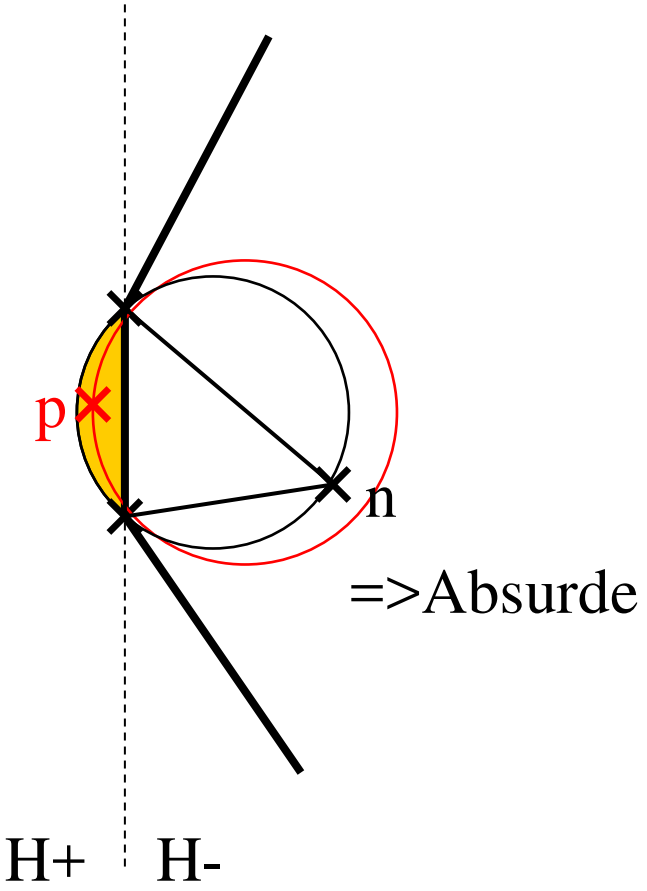
T=T-TD

Pour toute a de A

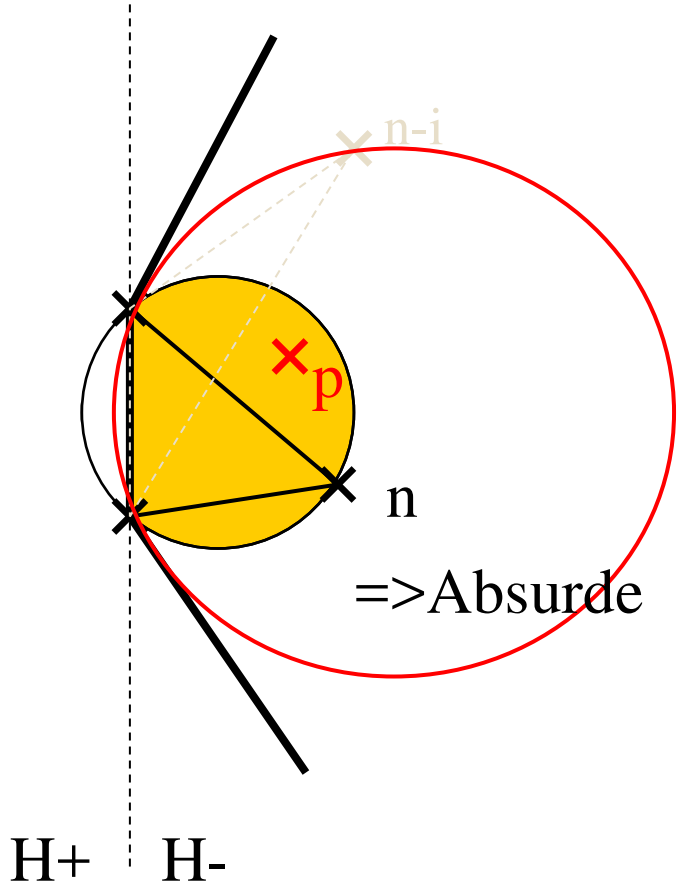
    T=T+triangle(a,n)



Démonstration (1)

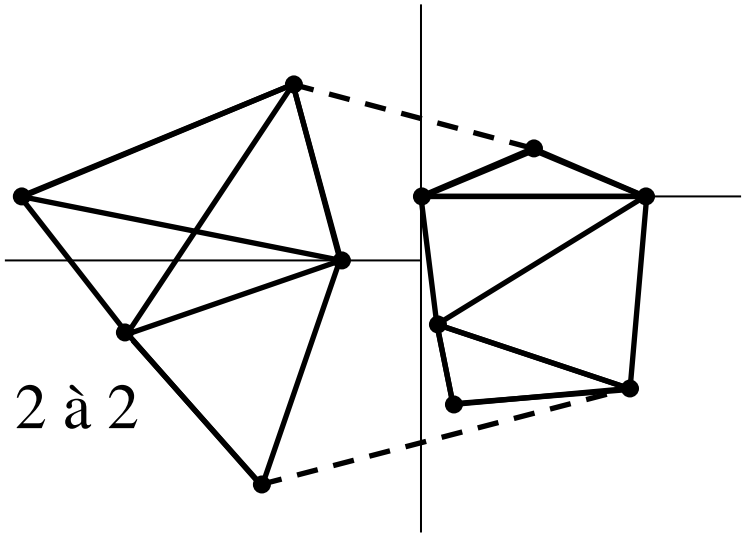


Démonstration (2)

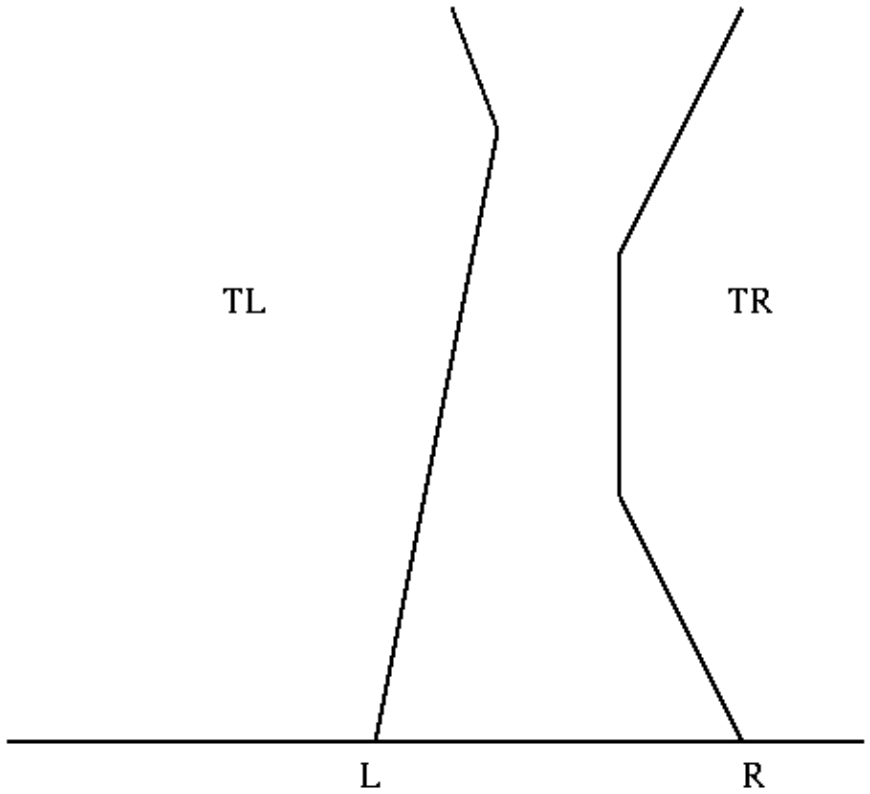


Delaunay : diviser pour régner

1. Construire KDtree
2. Calculer triangles
3. Fusionner les triangulations 2 à 2

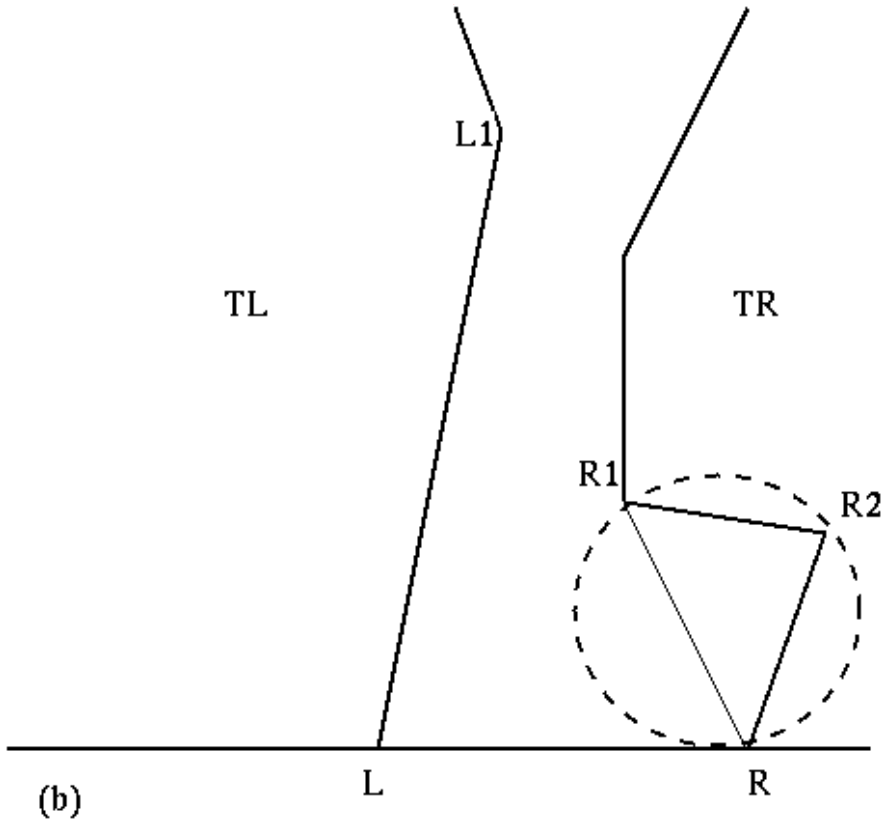


Fusion des Triangulations de Delaunay



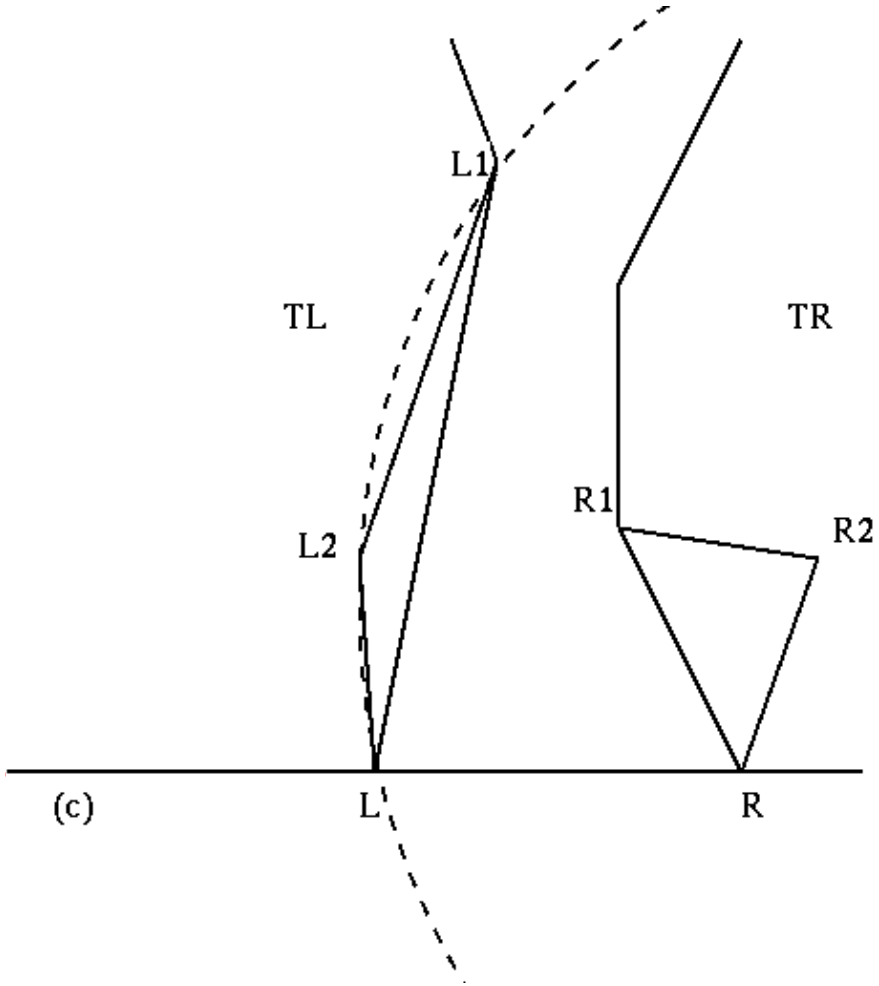
A) On joint les triangulations droite TR et gauche TL avec la tangente basse LR

Fusion des Triangulations de Delaunay



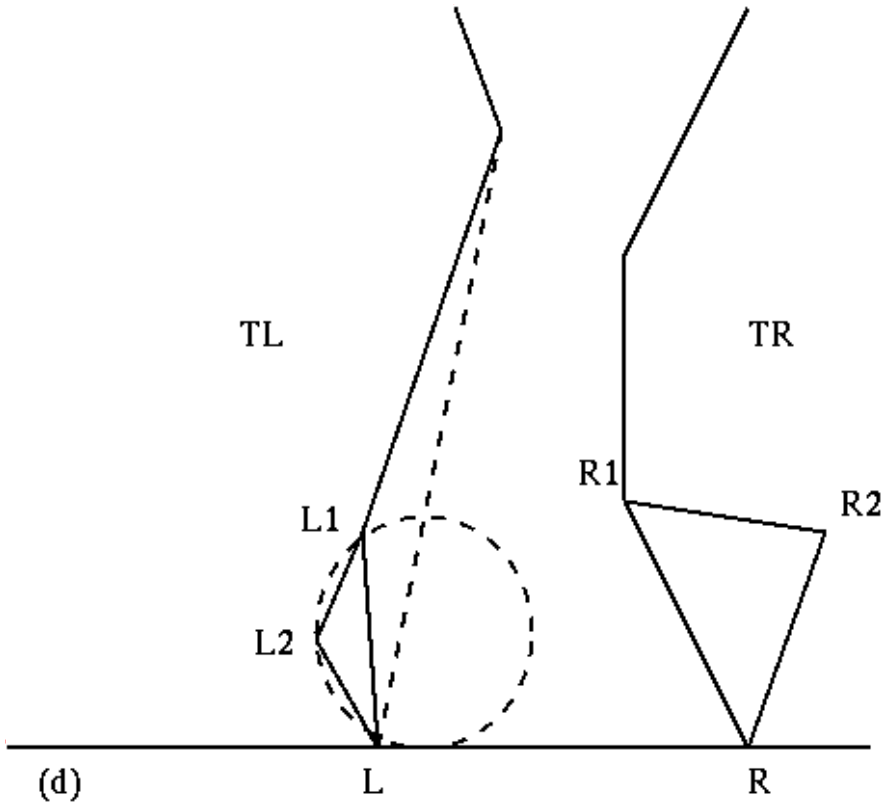
B) Les triangles de TR incidents à R qui ne satisfont pas Delaunay sont détruits

Fusion des Triangulations de Delaunay



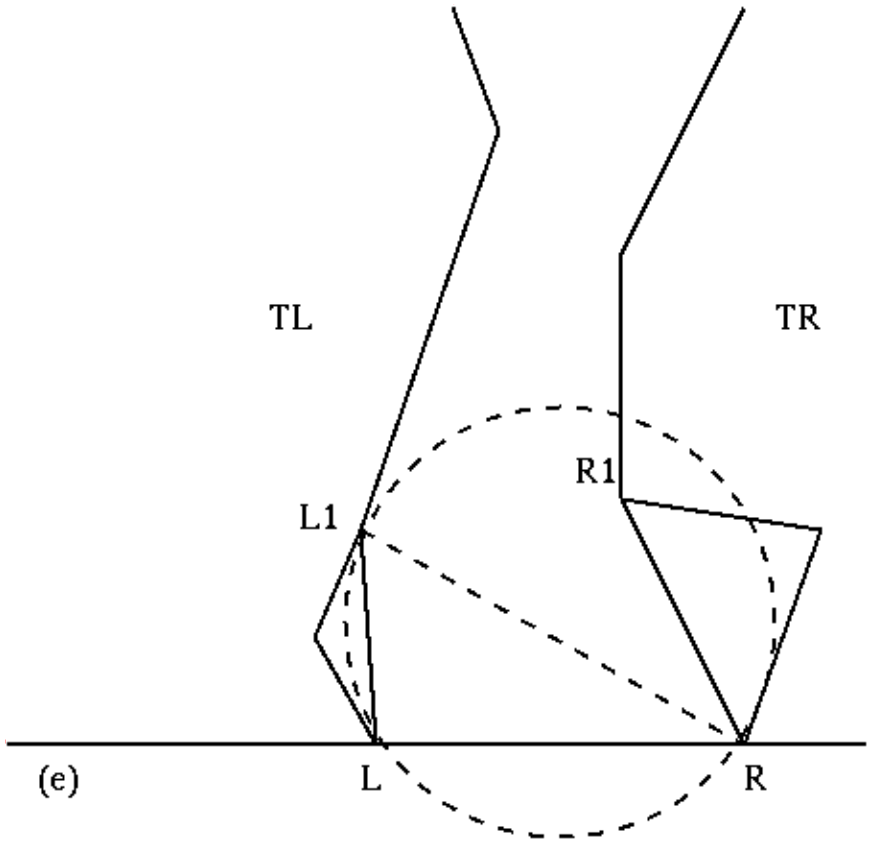
C) Les triangles de TL incidents à L qui ne satisfont pas Delaunay sont détruits

Fusion des Triangulations de Delaunay



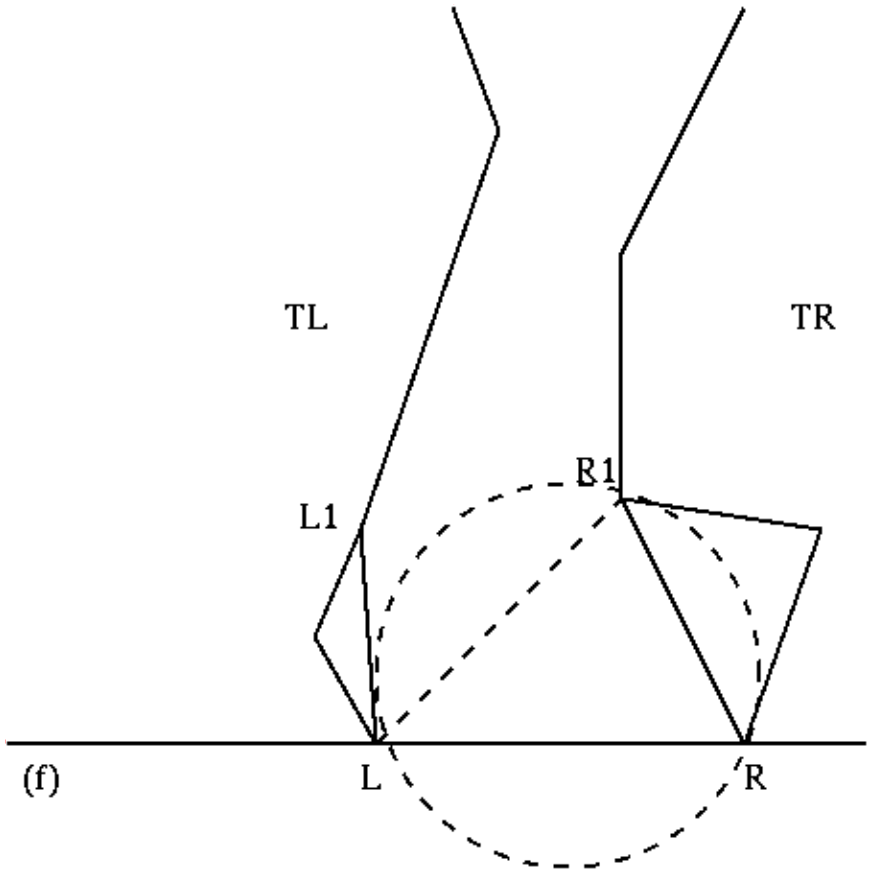
D) Les triangles de TL incidents à L qui ne satisfont pas Delaunay sont détruits

Fusion des Triangulations de Delaunay



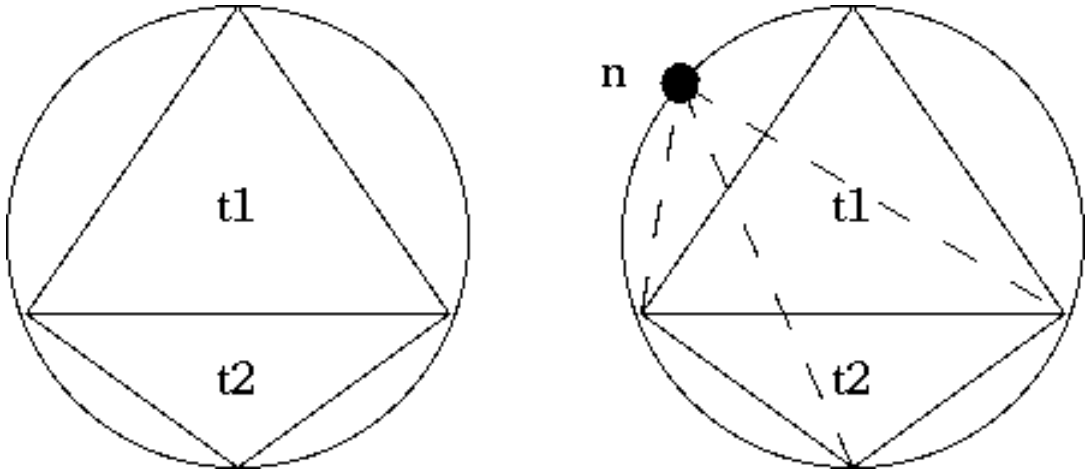
E) Construire le triangle de Delaunay s'appuyant sur l'arête LR

Fusion des Triangulations de Delaunay

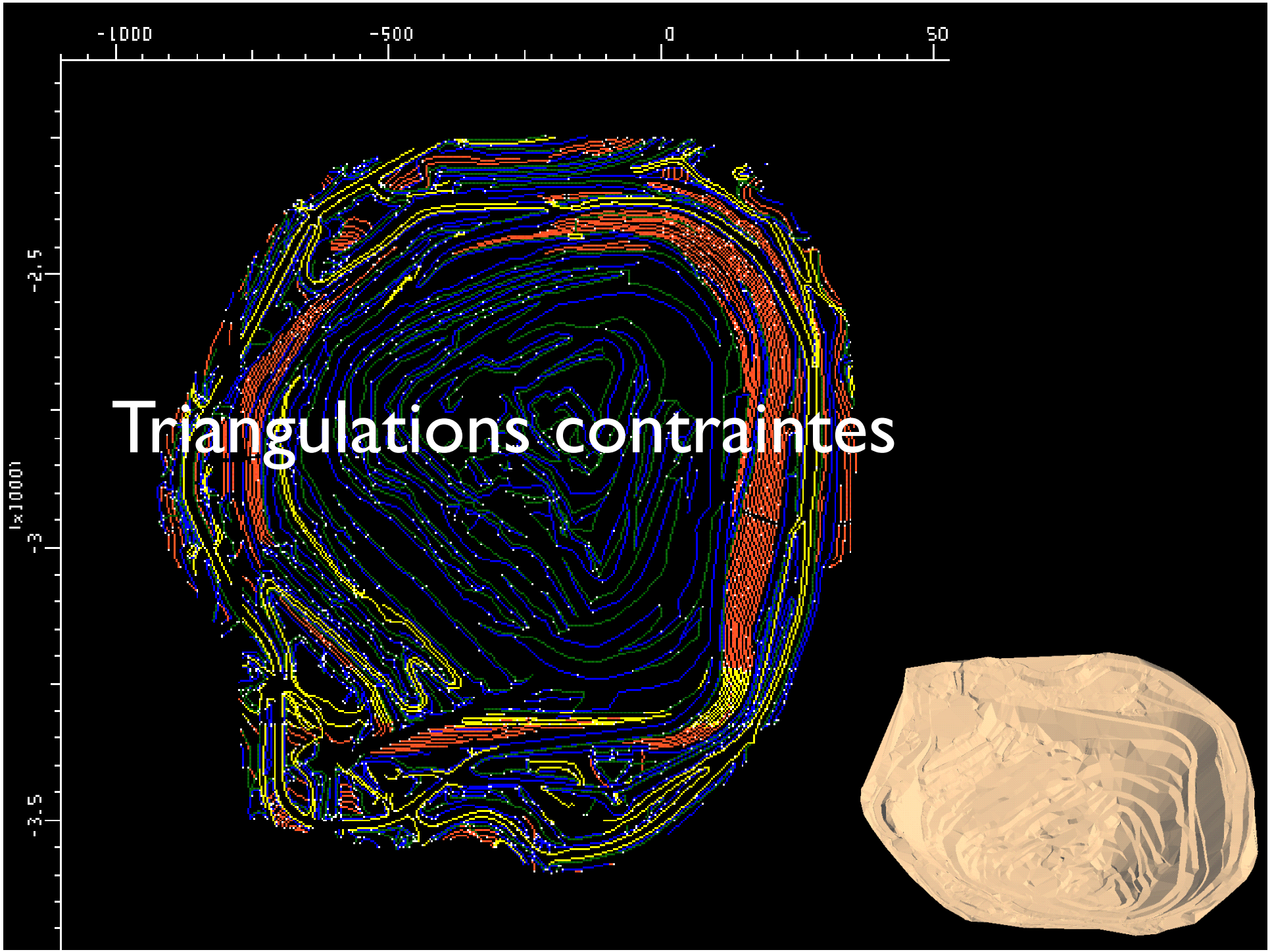


F) Construire le triangle de Delaunay s'appuyant sur l'arête LR

Erreur numérique

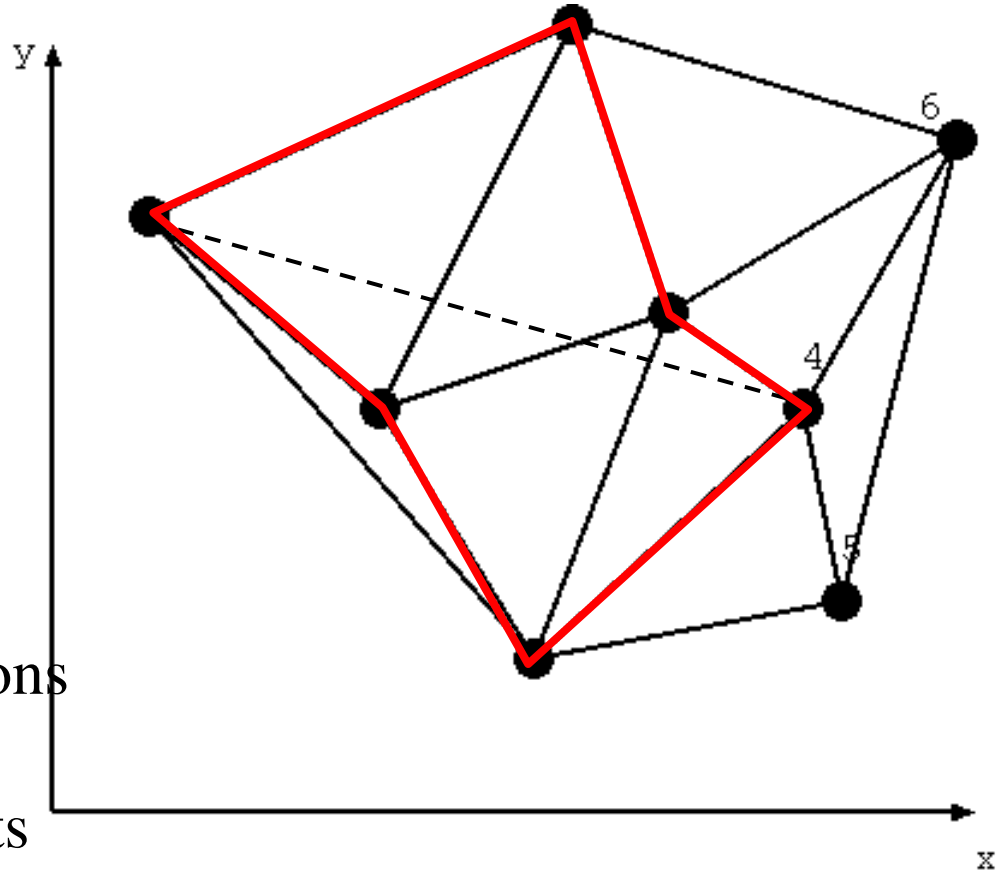


Triangulations contraintes



Triangulation contraintes

Incompatibilité entre :
le critère de Delaunay
le respect des arêtes

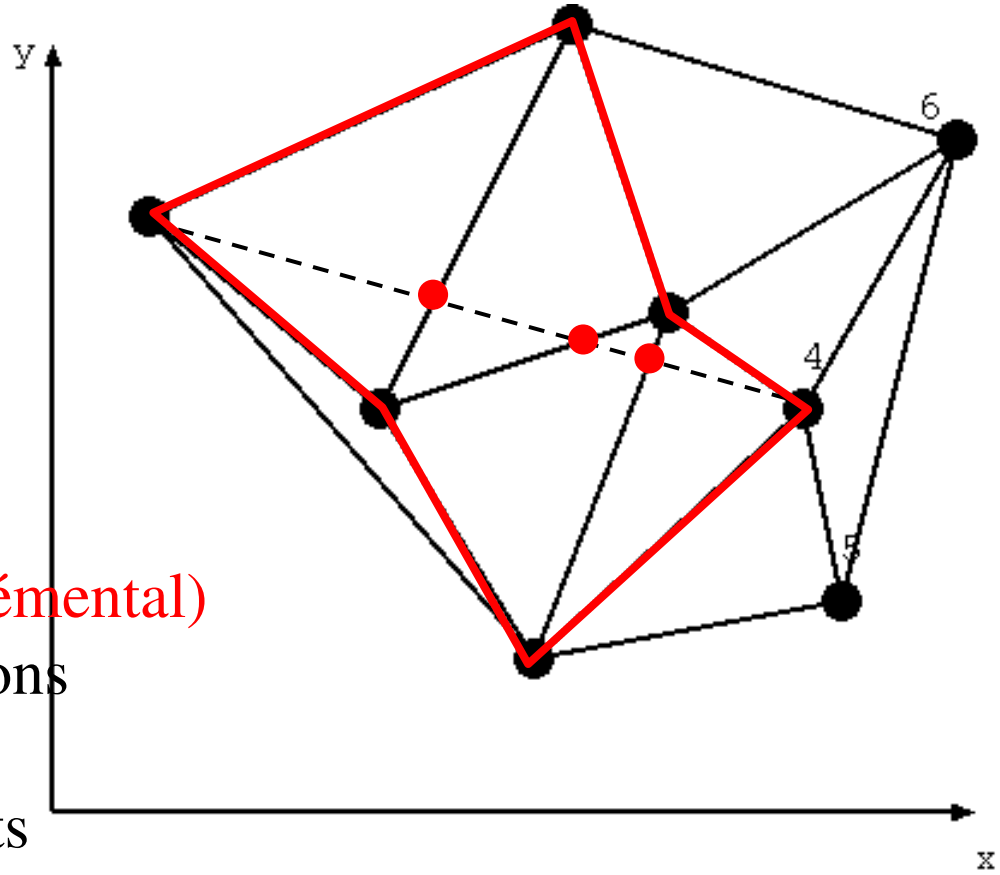


Traitement a posteriori
- Ajout de points
- Modification des connexions

Traitement a priori
- Détection et ajout de points

Triangulation contraintes

Incompatibilité entre :
le critère de Delaunay
le respect des arêtes

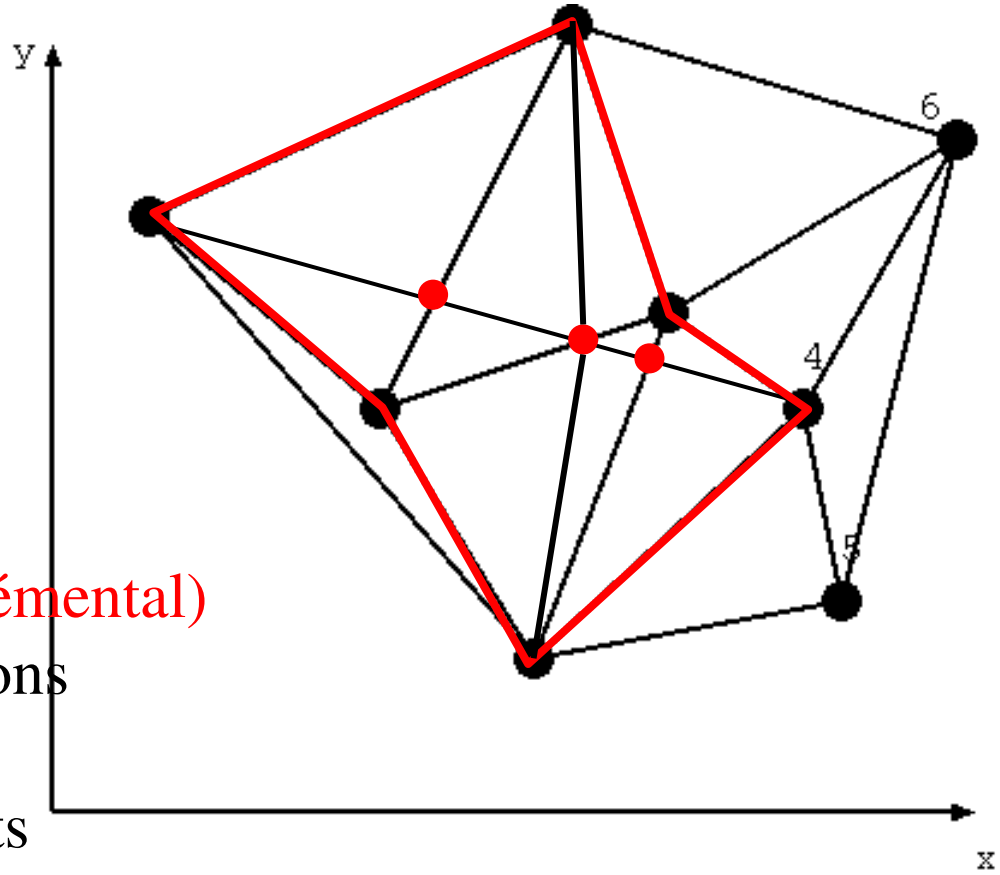


Traitement a posteriori
- Ajout de points (algo incrémental)
- Modification des connexions

Traitement a priori
- Détection et ajout de points

Triangulation contraintes

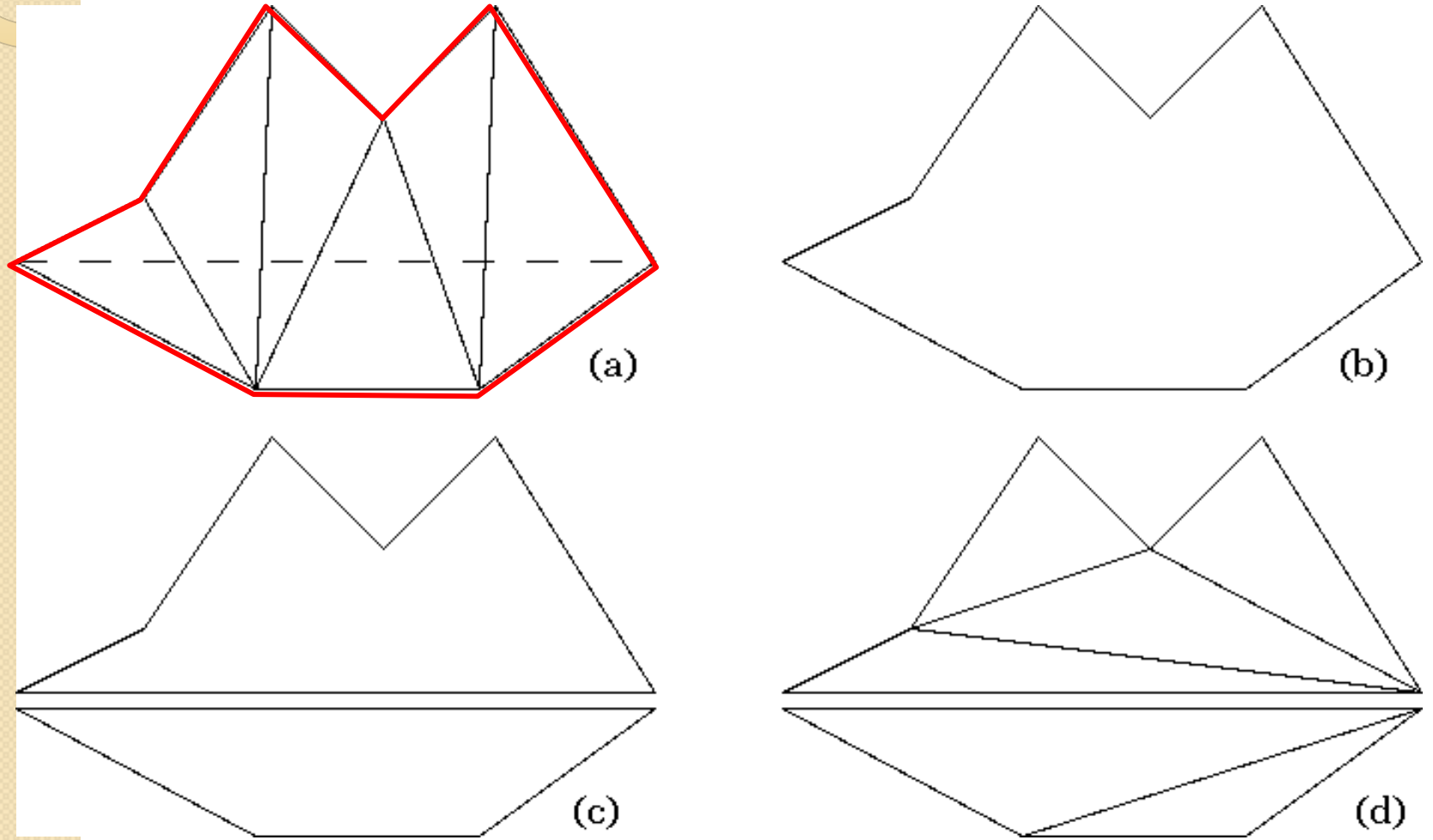
Incompatibilité entre :
le critère de Delaunay
le respect des arêtes



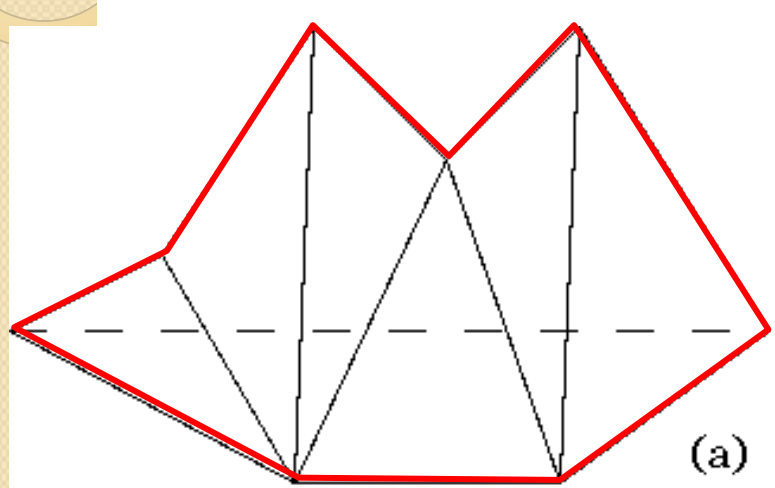
Traitement a posteriori
- Ajout de points (algo incrémental)
- Modification des connexions

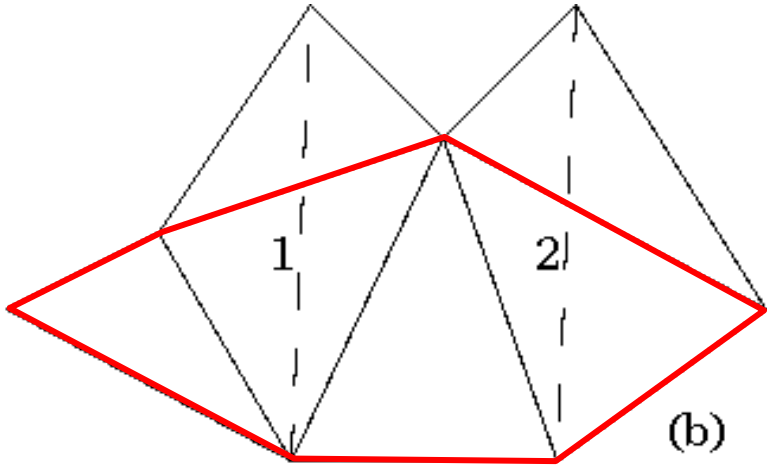
Traitement a priori
- Détection et ajout de points

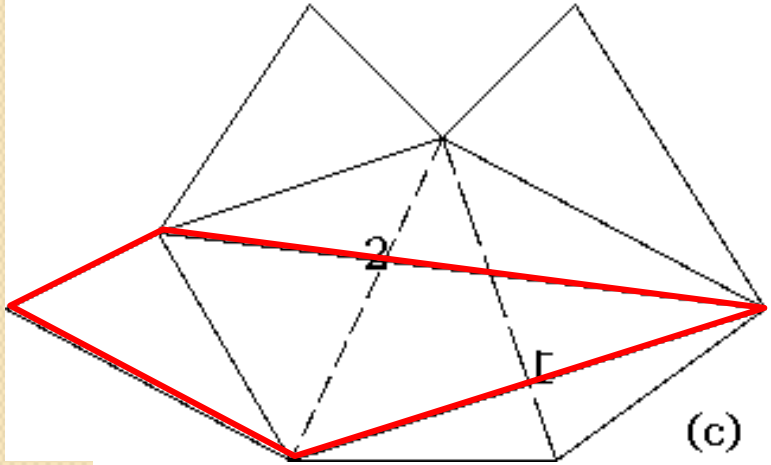
Remaillage

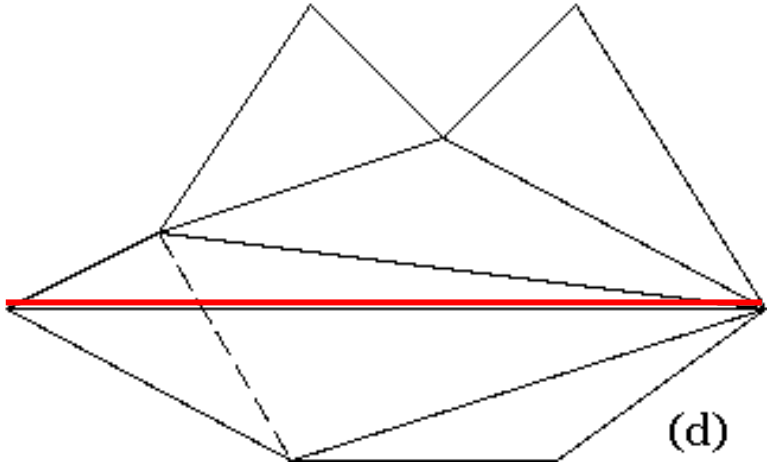


Forçage d'arêtes par inversion de diagonales

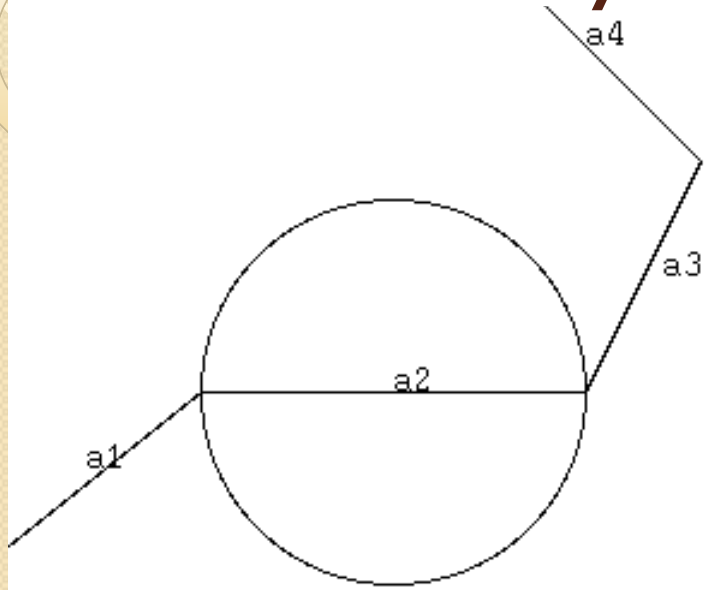


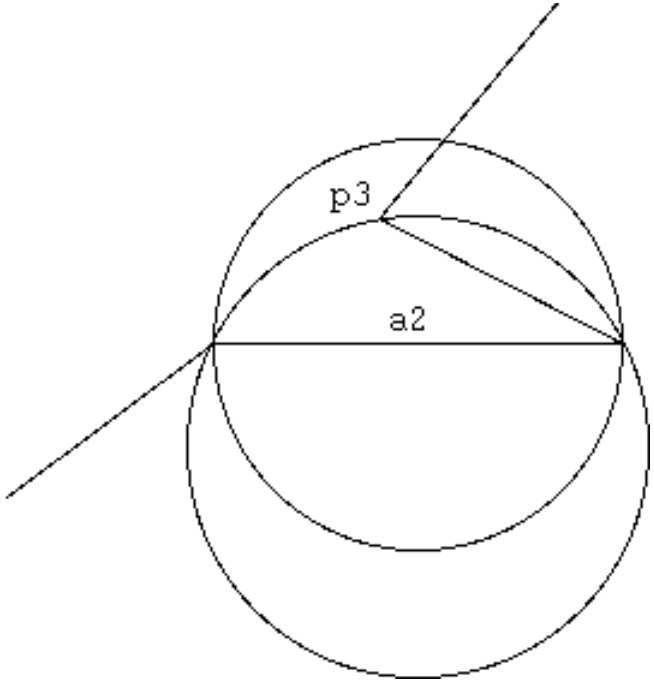






Détection des arêtes « non Delaunay-admissibles »





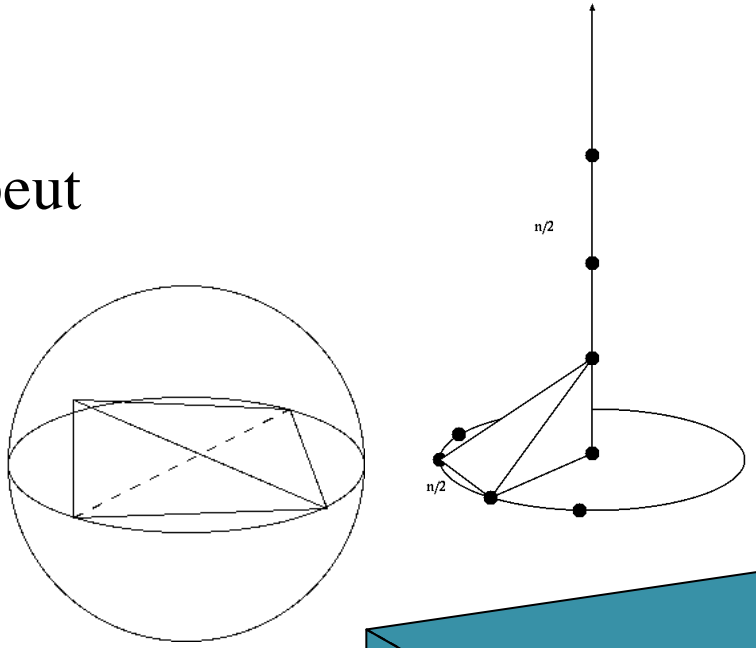
Si $B_\Phi(a_2)$ est vide alors :
a_2 est Delaunay admissible

Soit p_3 dans $B_\Phi(a_2)$ tel que $B_\Phi(a_2)$ soit vide du côté de a_2.
Si $B_\Phi(a_2)$ vide de l'autre côté alors :
a_2 est Delaunay admissible

Triangulations : Le cas 3D

- Cardinaux

Le nombre de tétraèdres peut être quadratique

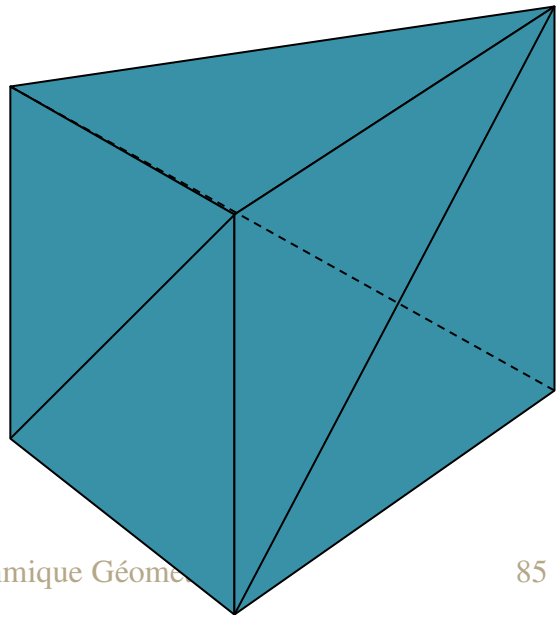


- Delaunay

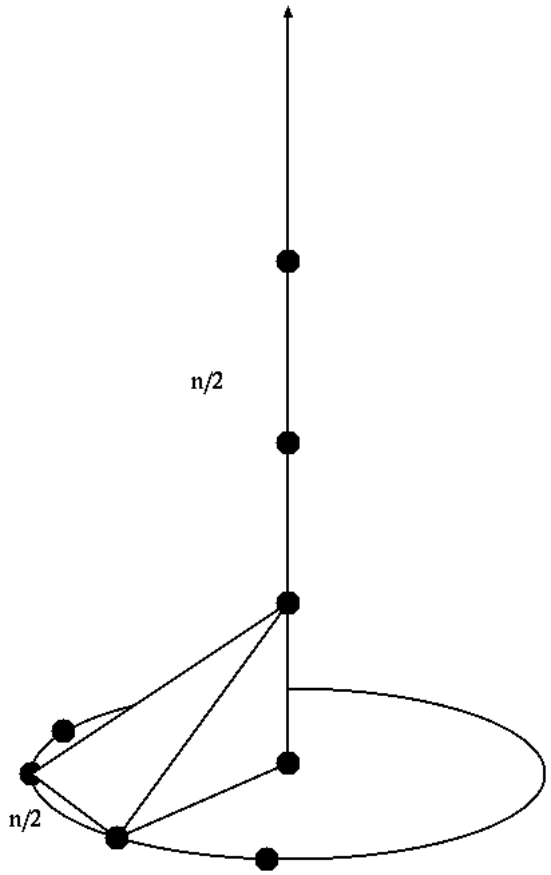
Le critère de Delaunay ne maximise pas les angles solides

- Forçage de frontière

Le Polyèdre de Schönhart ne peut être tétraédrisé sans ajout de point.

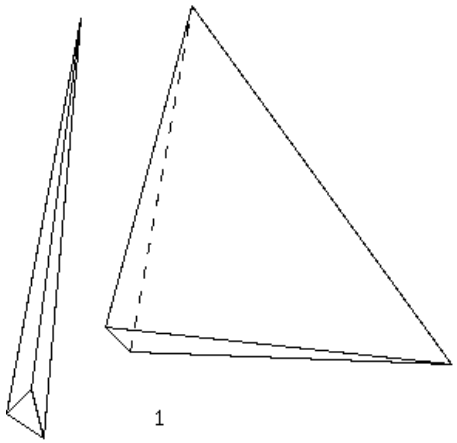


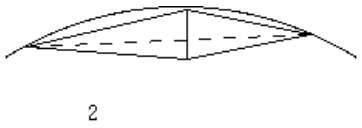
Cardinaux (cas 3D)

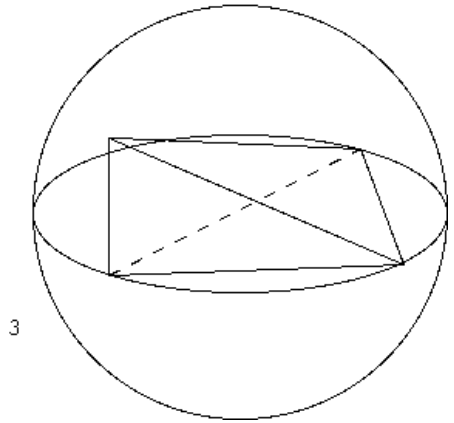


Qualité des triangulations (3D)

$c_1 R_i / L_{max} , < \tau_{min}$



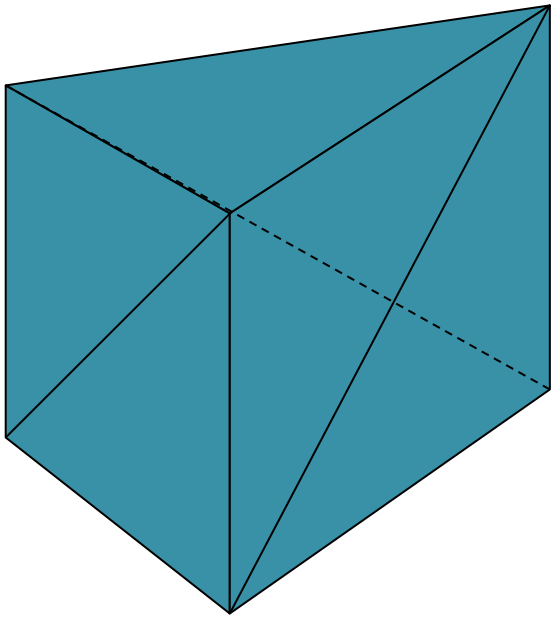




3 catégories :

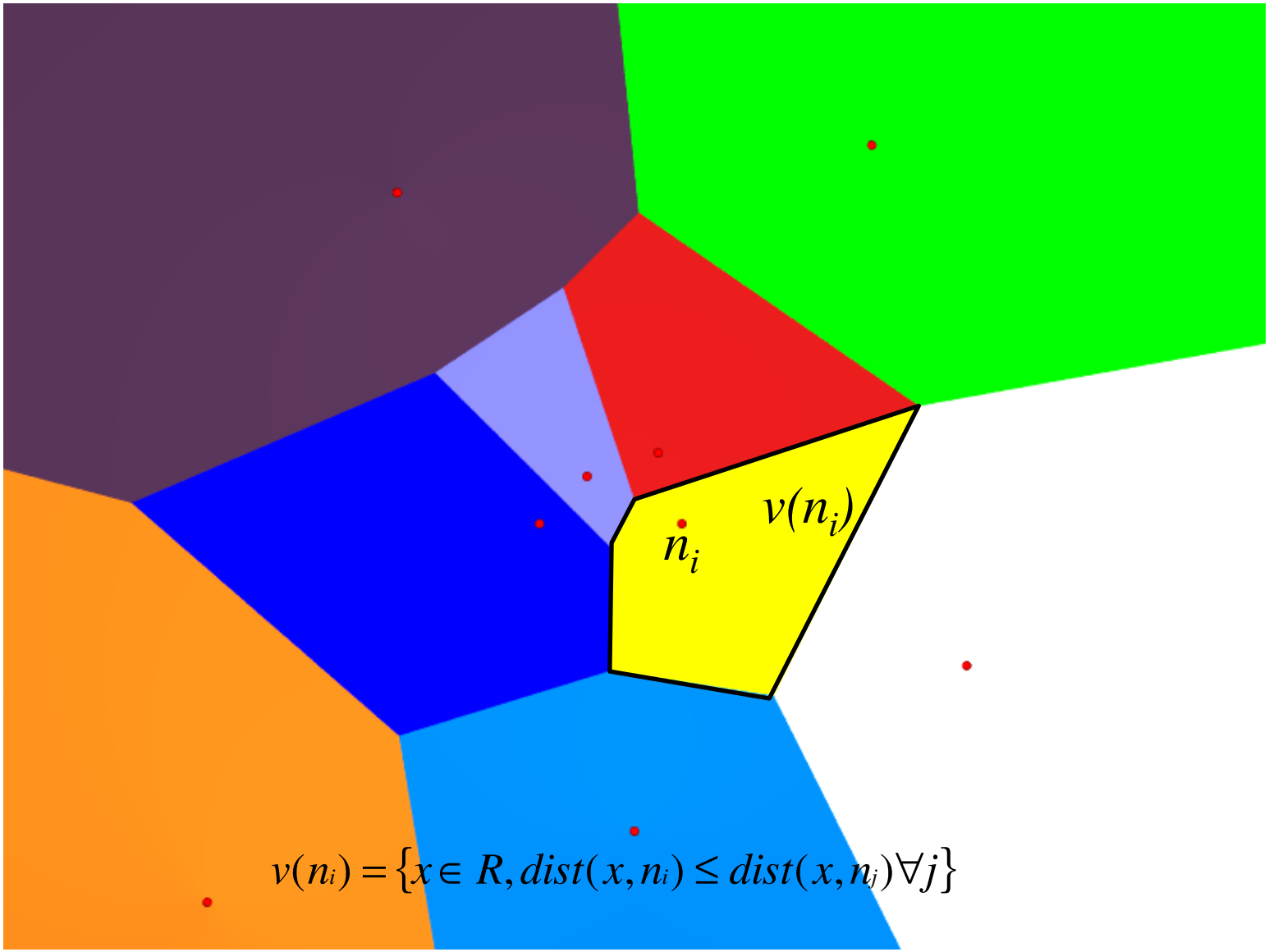
1) $l_{min}/L_{max} < \kappa_{min}$ et $c_3 L_{max}/R_c > \omega_{min}$
2) $l_{min}/L_{max} > \kappa_{min}$ et $c_3 L_{max}/R_c < \omega_{min}$
3) $l_{min}/L_{max} > \kappa_{min}$ et $c_3 L_{max}/R_c > \omega_{min}$

Triangulation contrainte en 3D

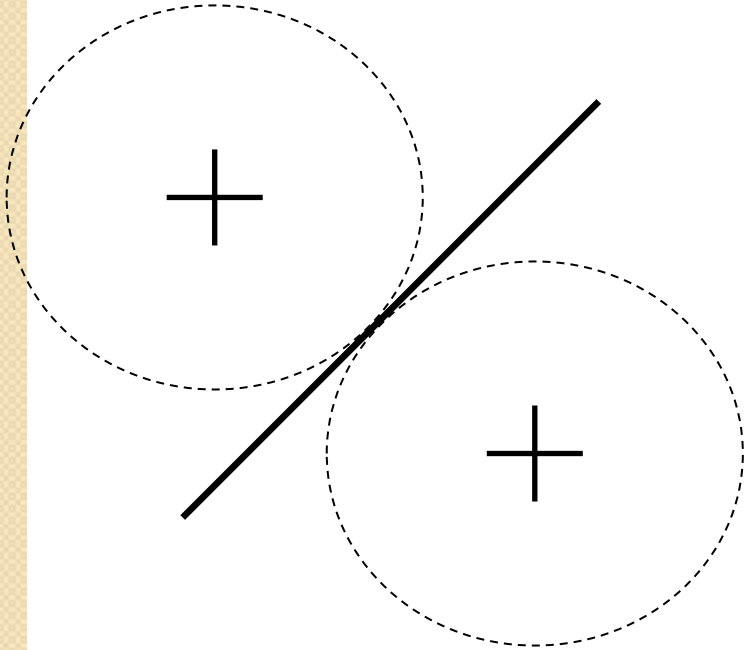


Le Polyèdre de Schönhart ne peut être tétraédrisé sans ajout de point.

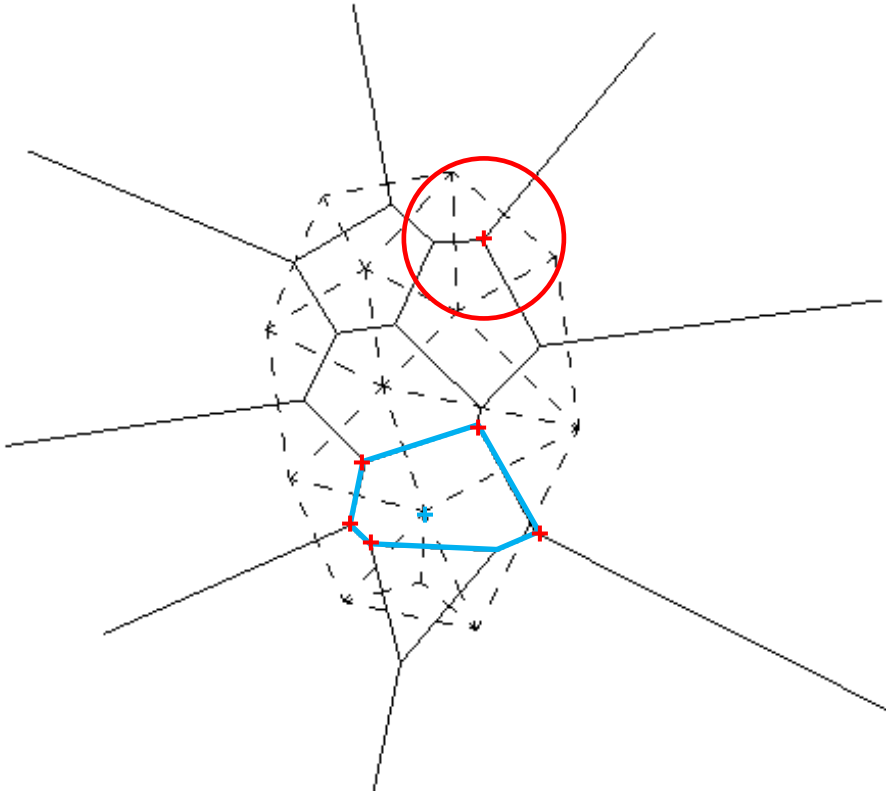
4. Graphe de Voronoï-Dirichlet



Voronoï métrique euclidienne



Dualité Delaunay/Voronoï

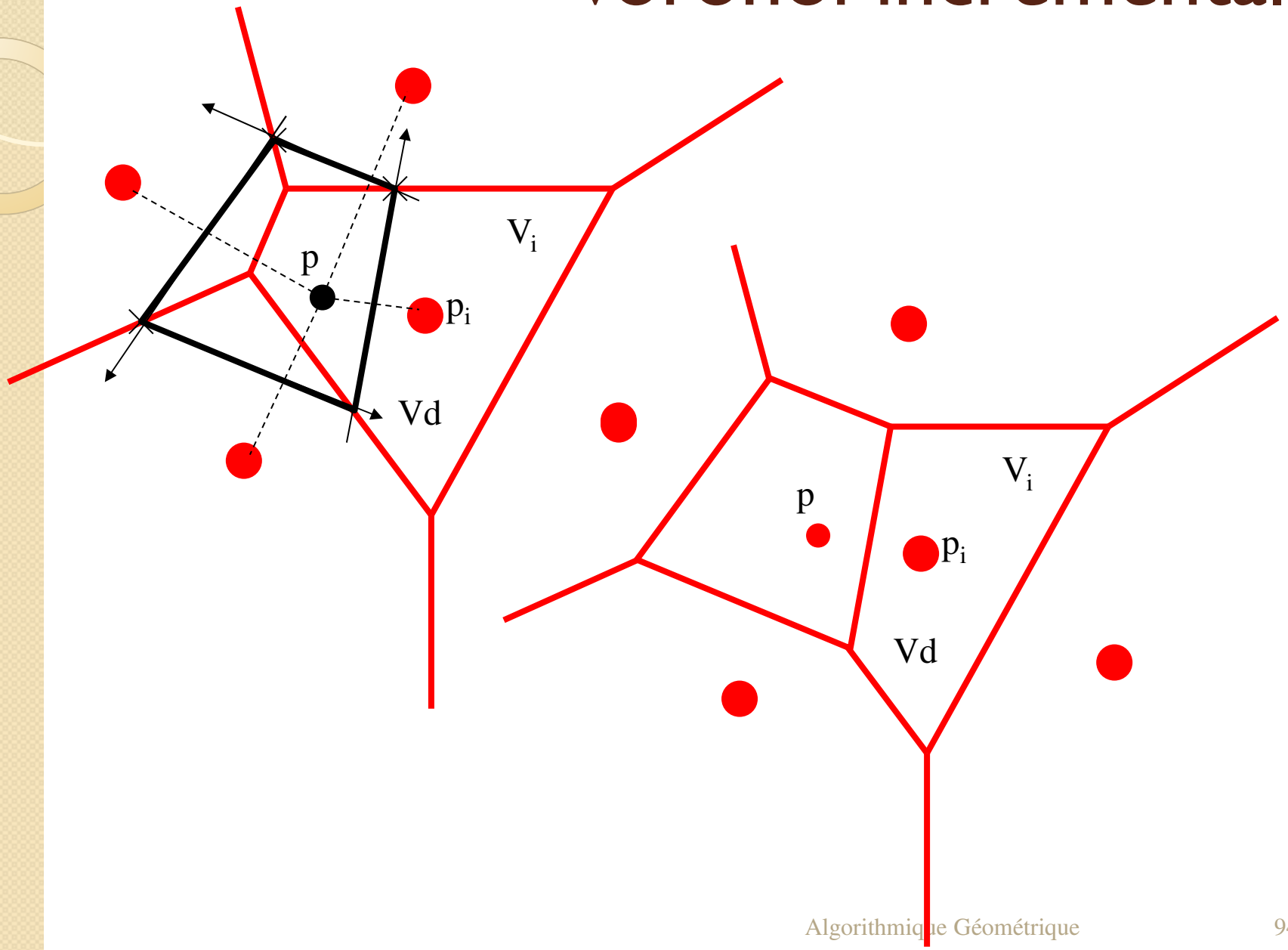


Principe de conversion : Delaunay➔Voronoï

Vor(S)=Polygone ={ le centre des cercles des triangles incidents à S}

Algorithme incrémental

Voronoï incrémental



Algorithme incrémental

Ajouter_nœud(p,G)

V_i,p_i = PPV(p,G)

Vd=V_i

Faire

m_i = médiatrice(p,p_i)

a_i= 1ère arête de V_i intersecté par m_i

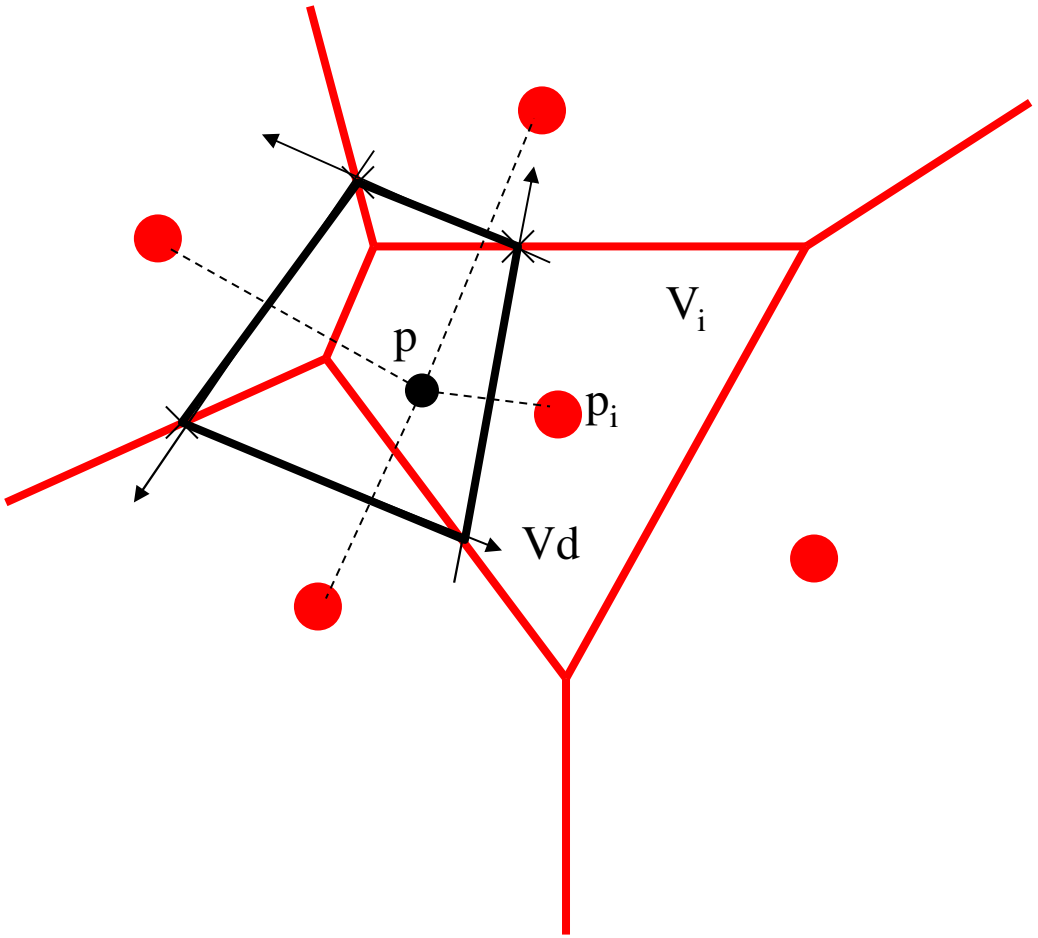
pv = point d'intersection(a_i,m_i)

V=V u {pv}

V_i,p_i = région adjacente à V_i sur a_i

Tant que (V_i != Vd)

créer_région(V,p)



Algorithme incrémental

Ajouter_nœud(p,G)

V_i,p_i = PPV(p,G)

Vd=V_i

Faire

m_i = médiatrice(p,p_i)

a_i= 1ère arête de V_i intersecté par m_i

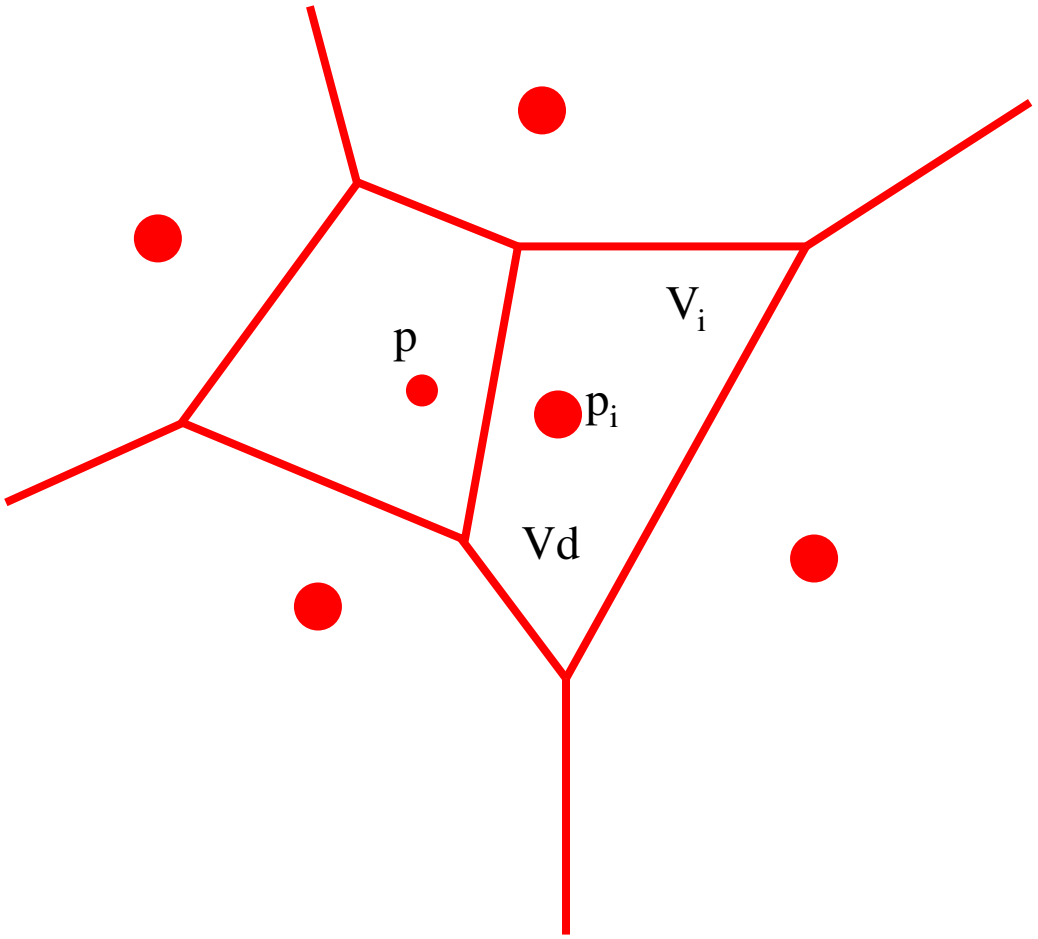
pv = point d'intersection(a_i,m_i)

V=V u {pv}

V_i,p_i = région adjacente à V_i sur a_i

Tant que (V_i != Vd)

créer_région(V,p)



Voronoï D&C

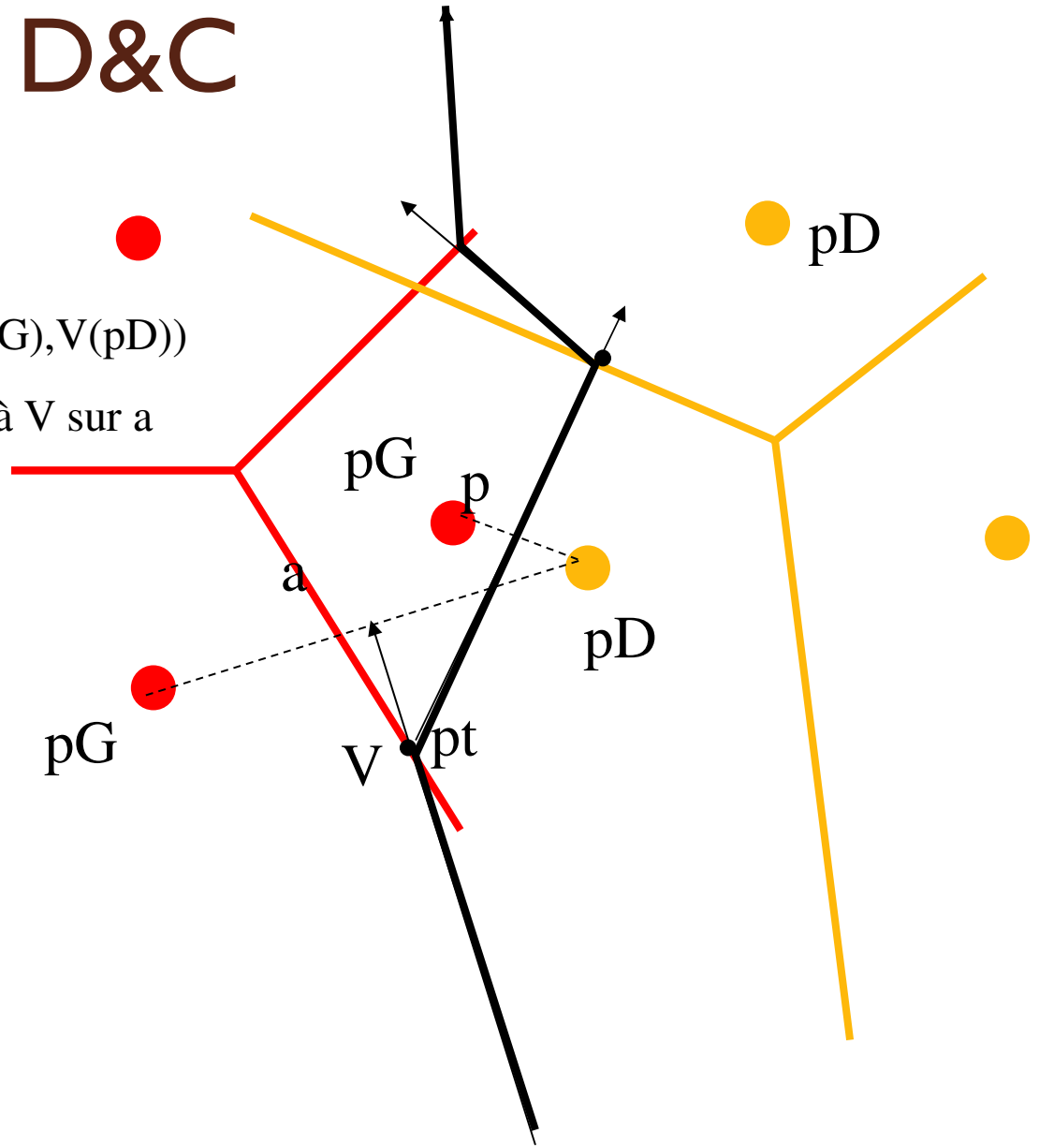
m=médiatrice(pG,pD)

V,a,pt = 1ere intersection(m,V(pG),V(pD))

p = point de la région adjacente à V sur a

Si V==V(pG) pG=p sinon pD=p

couper(a,pt,m)



Algorithme diviser pour régner

FusionGraphe(VG,VD)

qG',qD' = TanSup(EC(G),EC(D));

qG,qD = TanInf(EC(G),EC(D));

pG=qG' ; pD=qD'

Faire

  m=médiatrice(pG,pD)

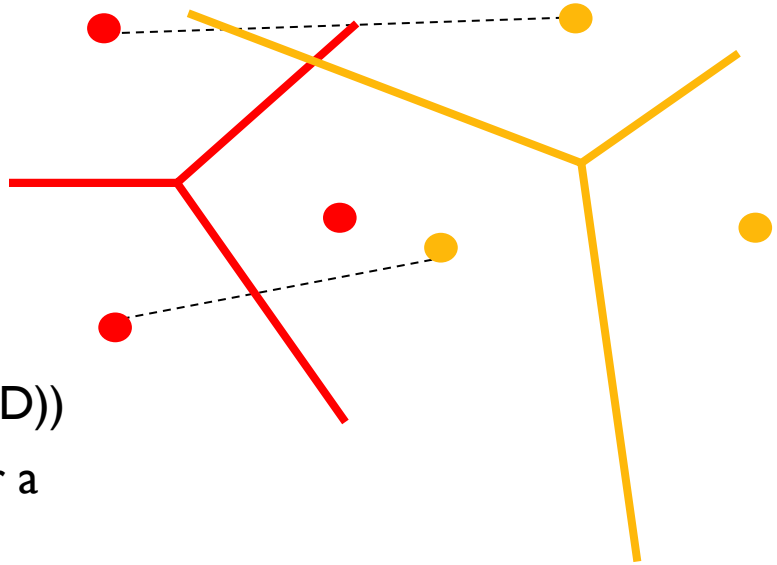
  V,a,pt = 1ere intersection(m,V(pG),V(pD))

  p = point de la région adjacente à V sur a

  Si V==V(pG) pG=p sinon pD=p

  couper(a,pt,m)

Tant que (pG != pG' et pD!=pD')



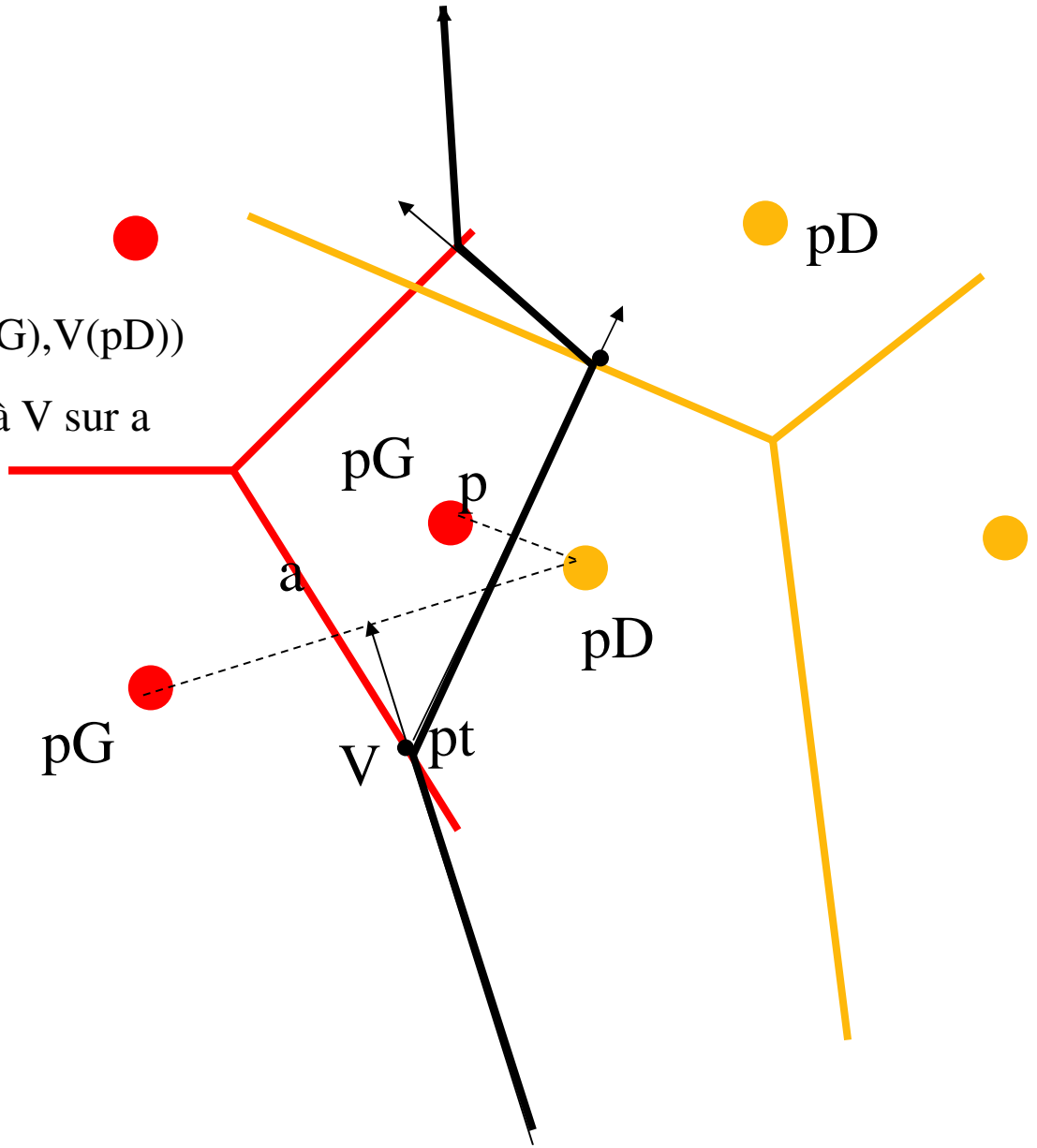
m=médiatrice(pG,pD)

V,a,pt = 1ere intersection(m,V(pG),V(pD))

p = point de la région adjacente à V sur a

Si V==V(pG) pG=p sinon pD=p

couper(a,pt,m)



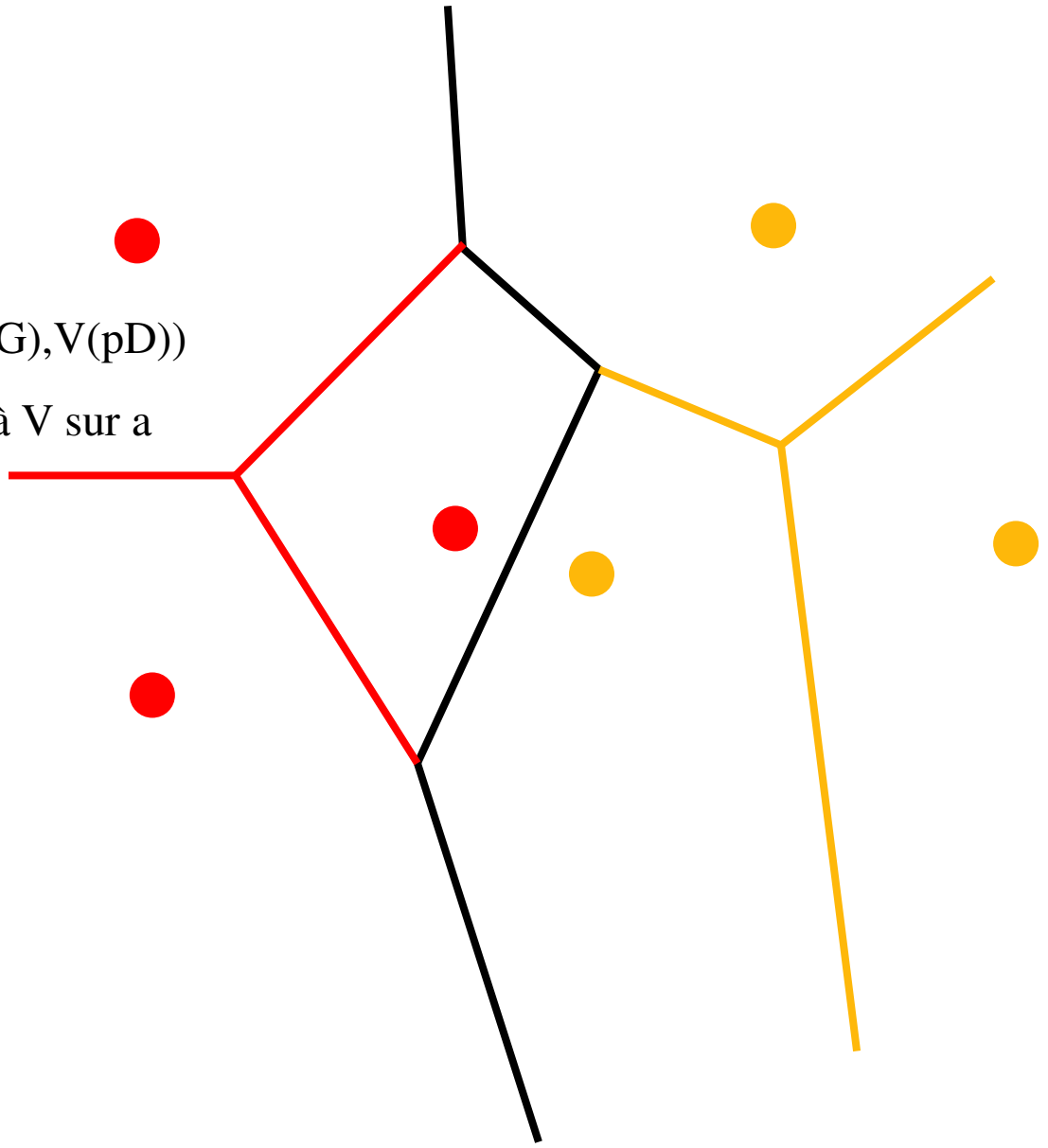
m=médiatrice(pG,pD)

V,a,pt = 1ere intersection(m,V(pG),V(pD))

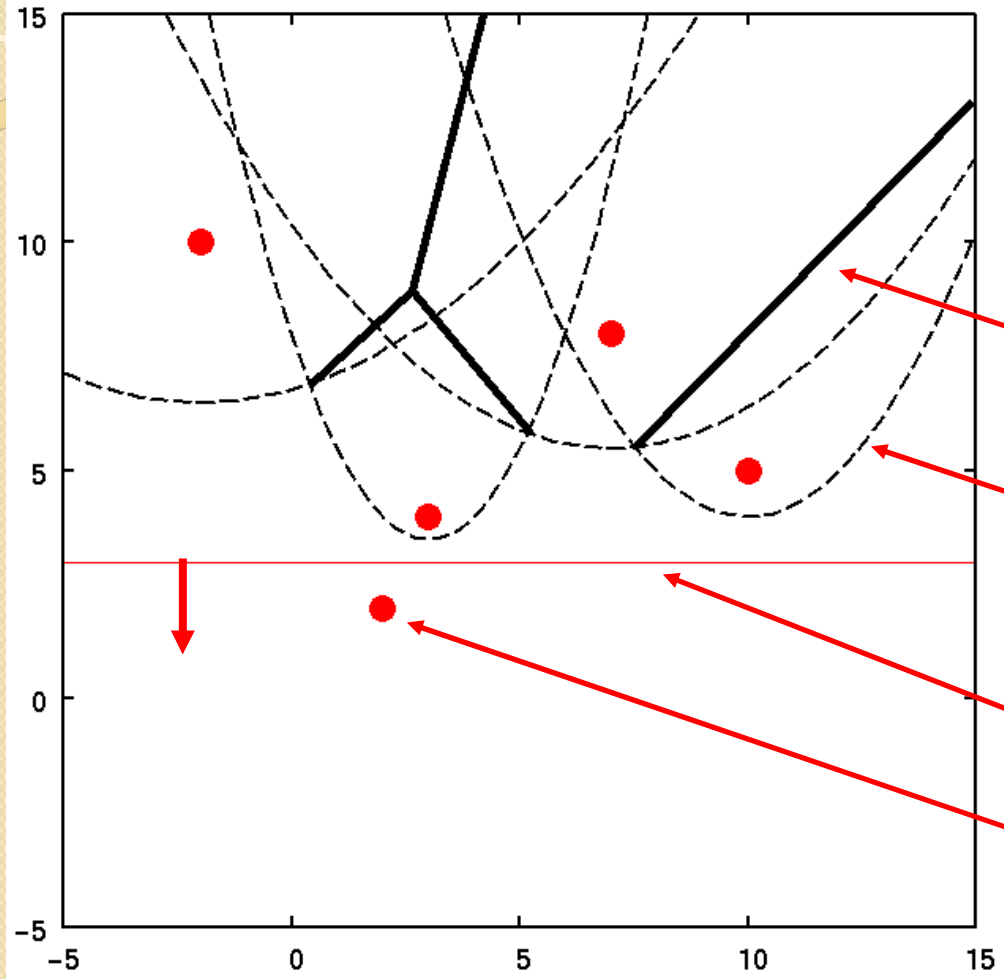
p = point de la région adjacente à V sur a

Si V==V(pG) pG=p sinon pD=p

couper(a,pt,m)



Voronoï par balayage



PRINCIPE

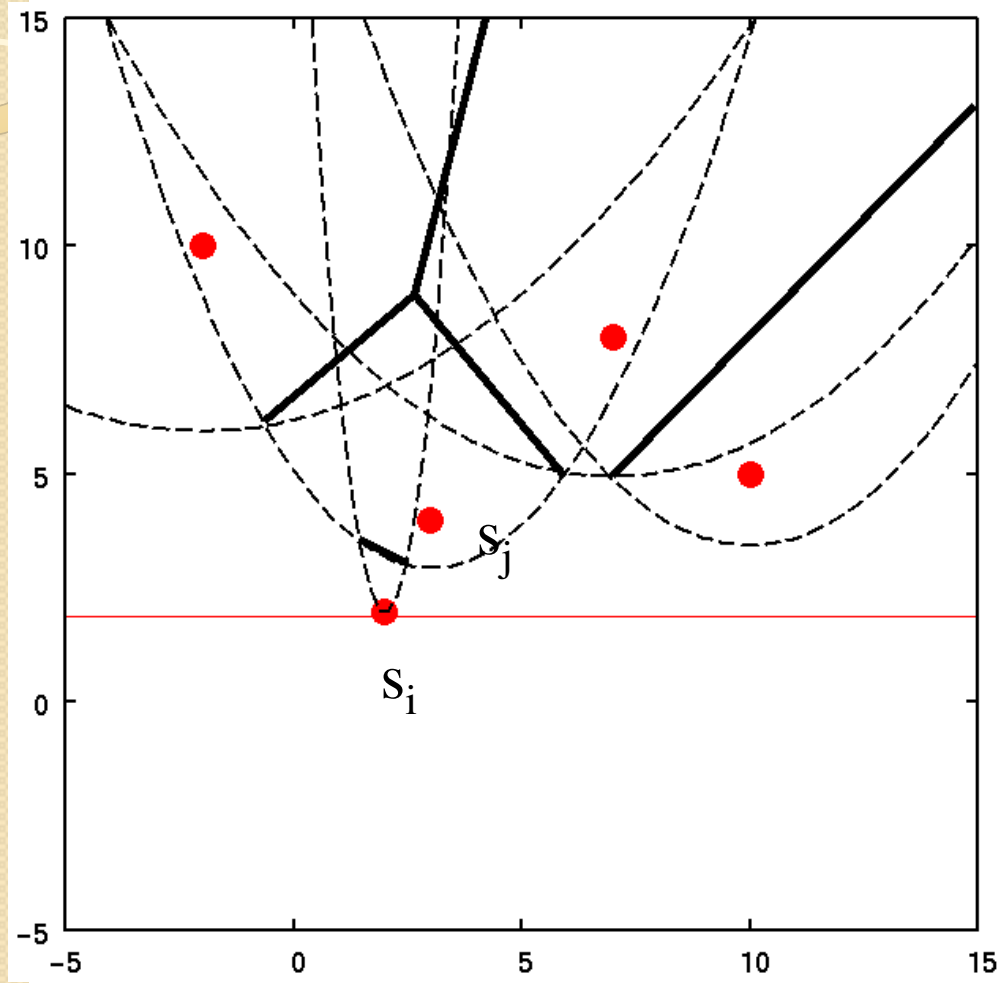
Voronoï en construction

Ligne de plage

Droite de balayage

Evènements

Voronoï par balayage

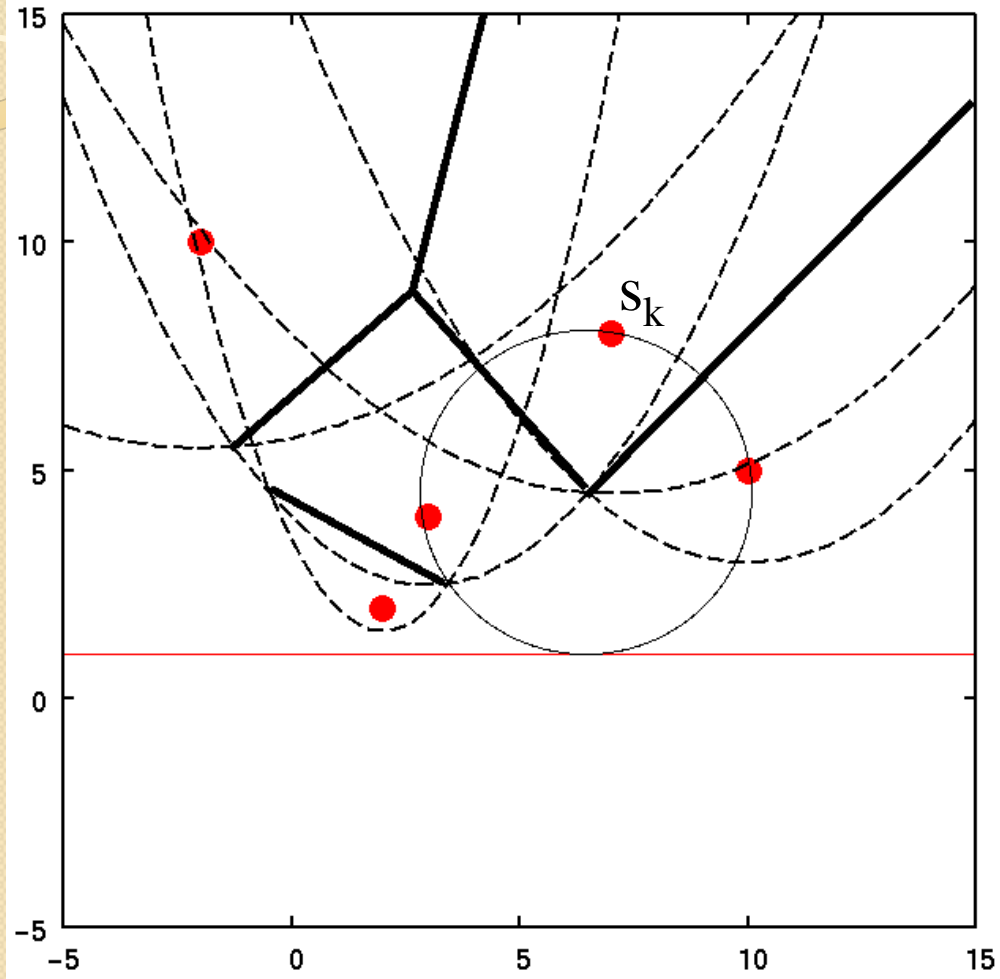


Apparition d'un arc (de s_i)

Mise à jour de la ligne de plage

Insertion de l'arête entre s_i et s_j

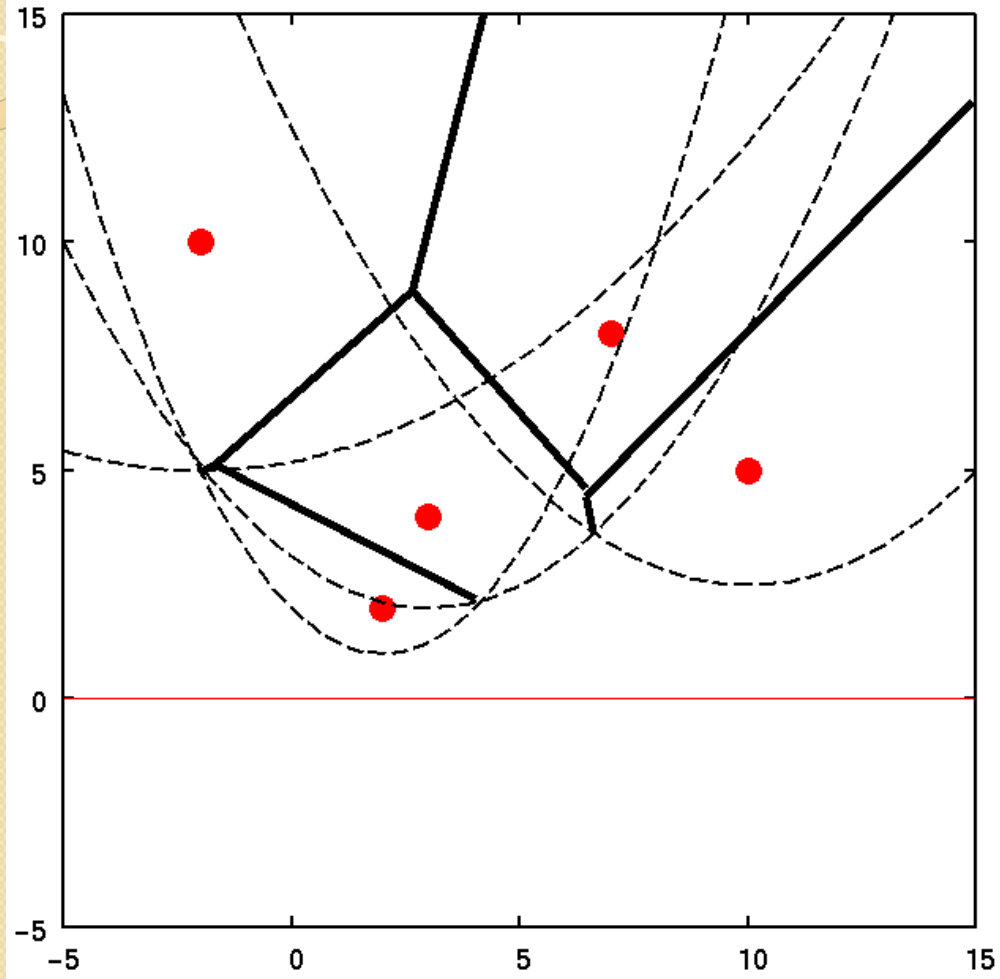
Voronoï par balayage



Disparition d'un arc (de s_k)

Pour la détection de ces évènements la programme maintient la liste des cercles circonscrits à 3 sites successifs sur la ligne de plage

Voronoï par balayage

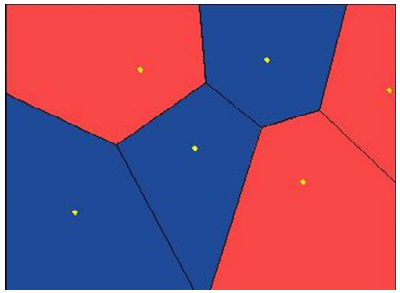


Jusqu'à traiter tous les évènements…

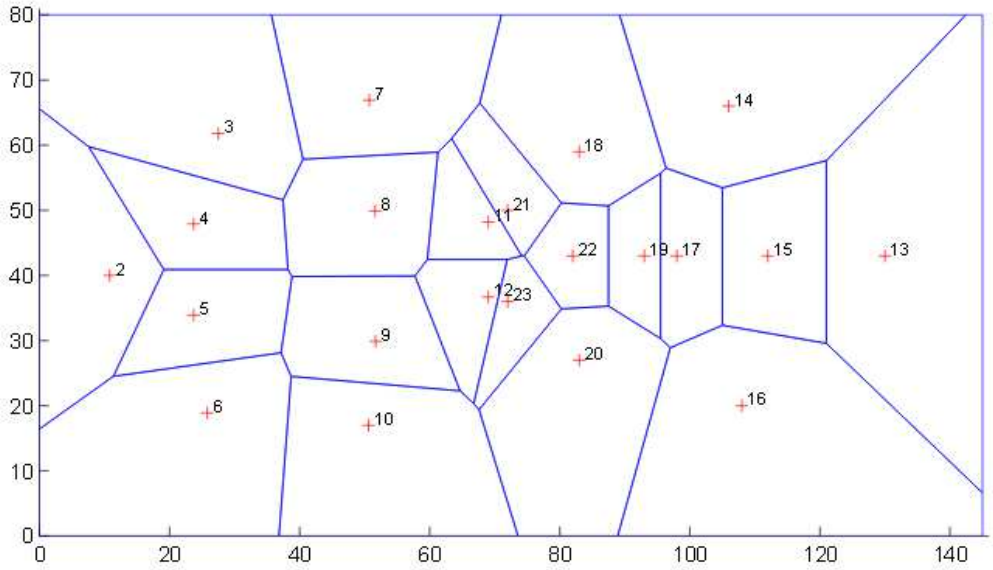
Algorithme de Fortune
Optimal : *O(nlog(n))*

Application dans les jeux

- Un jeu en lui-même
  http://www.voronoigame.com/



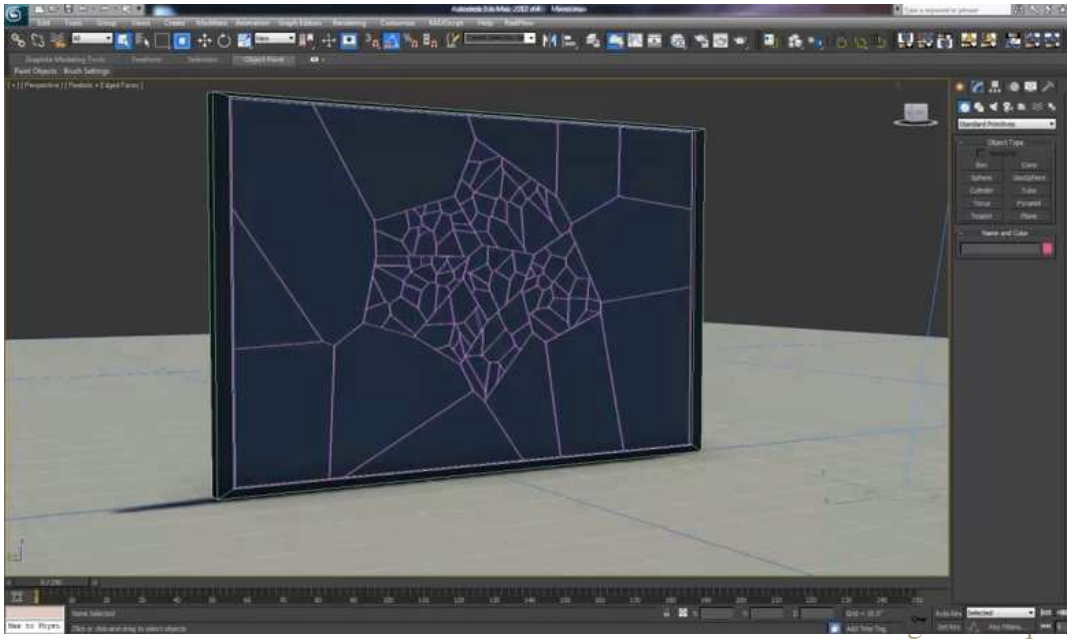
- Stratégie / influence / Estimation d'algorithmes de distribution



Voronoï appliqué à la fracturation

Blender et Bullet Physics

3DSmax



Maya script :
voronoï
texture
fracture

5. Voronoï généralisé

$$v(N_i) = \left\{ X \in R^d, \forall M_{ik} \in N_i, \forall M_{jk} \in N_j, dist(X, M_{ik}) \leq dist(X, M_{jk}) \forall j \right\}$$

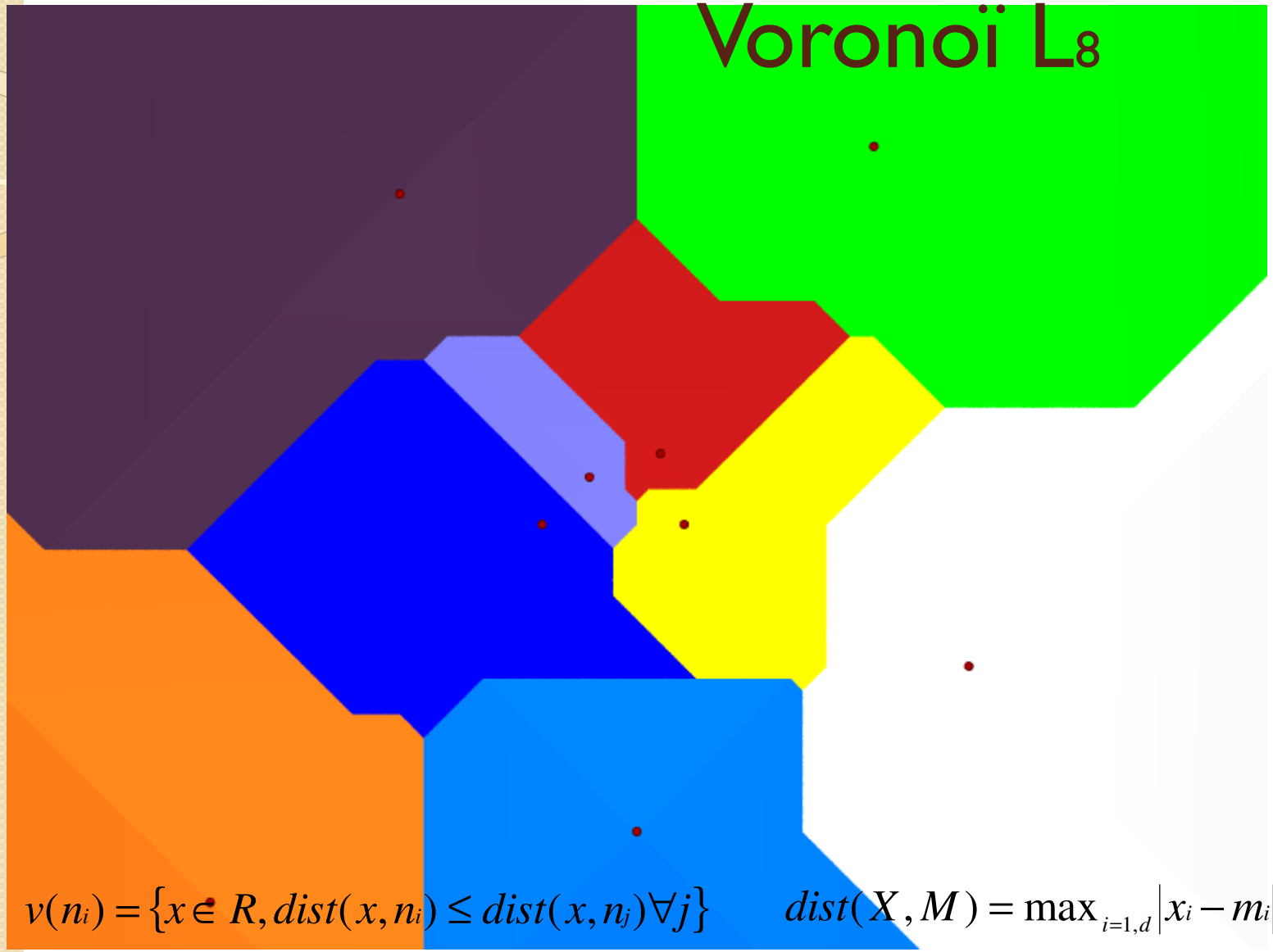
- En dimension : d=3…
- Avec pondération
- Métrique : $dist(X, M) = \sum_{i=1}^{d} |x_i - m_i|$

  $dist(X, M) = \max_{i=1,d} |x_i - m_i|$

- Ordre supérieur : *Card(N_i) =2, 3 K* sites

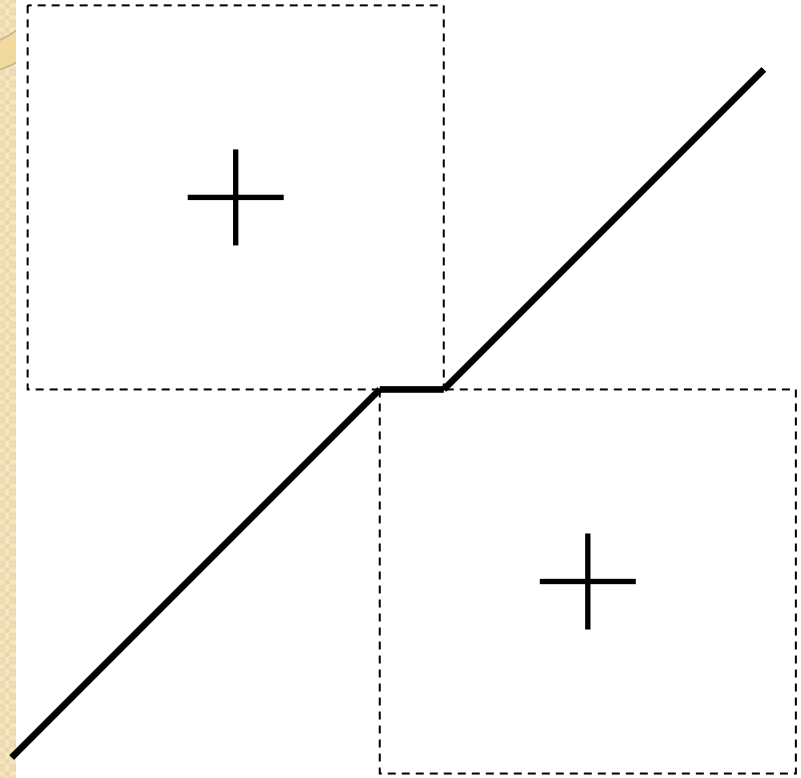
- Variété d'ordre supérieur : arêtes

Voronoï L_8



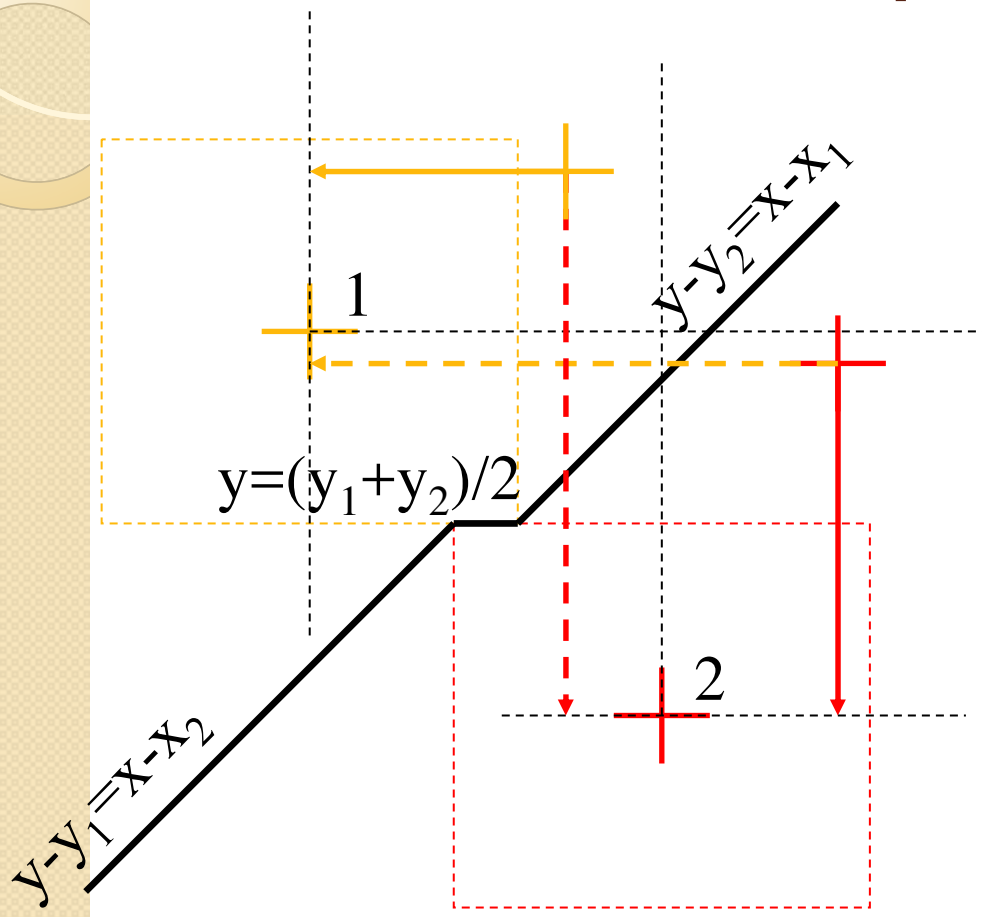
$$v(n_i) = \{x \in R, dist(x, n_i) \leq dist(x, n_j) \forall j\} \qquad dist(X, M) = \max_{i=1,d} |x_i - m_i|$$

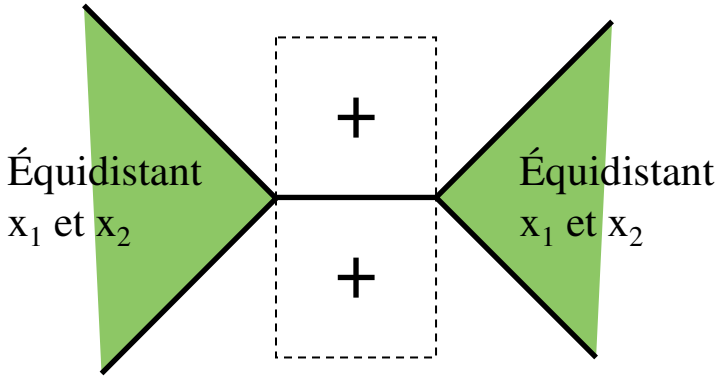
Voronoï métrique L_8



$$dist(X, M) = \max_{i=1,d} |x_i - m_i|$$

Voronoï métrique L_∞

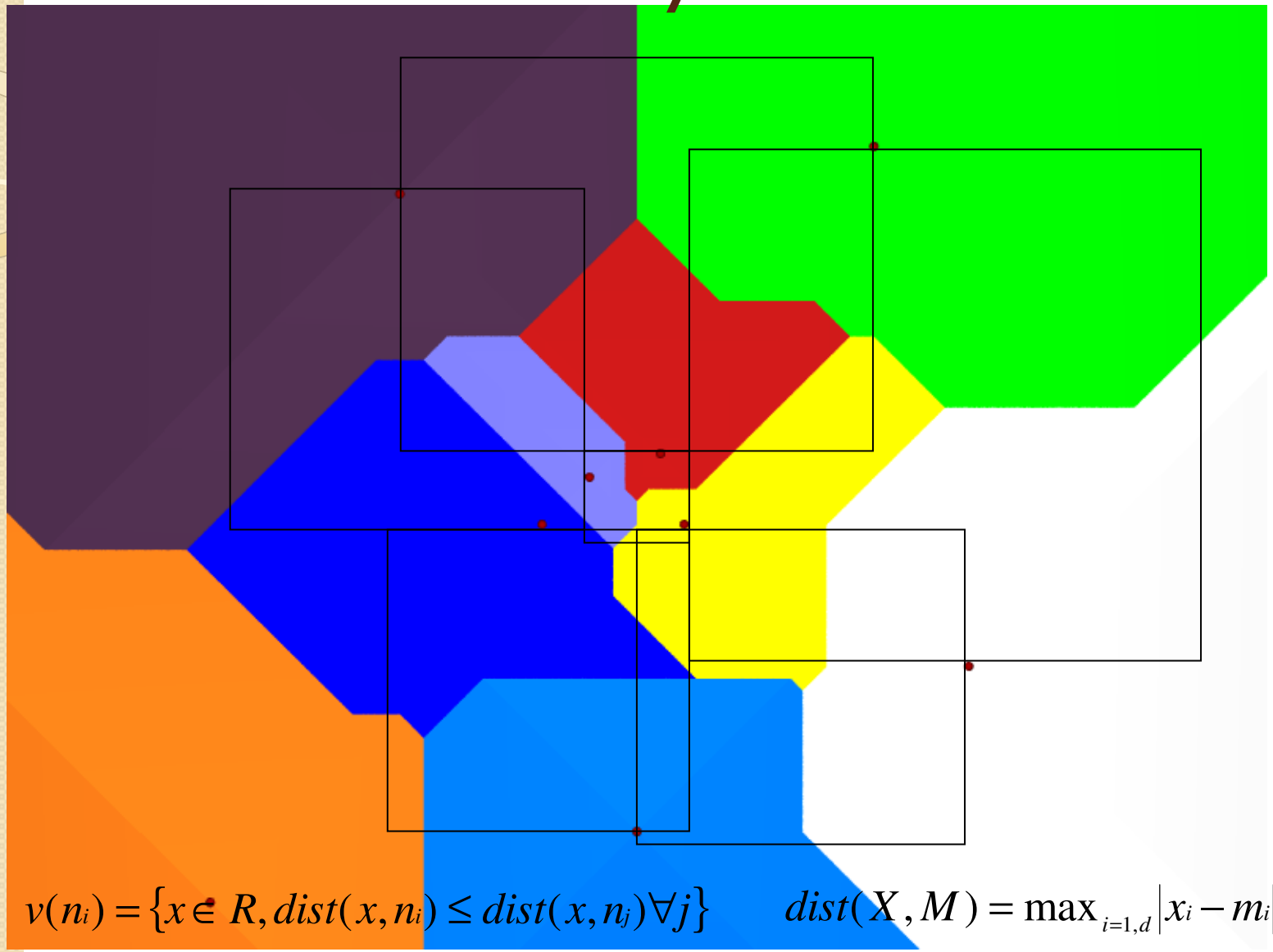




Ambiguïté

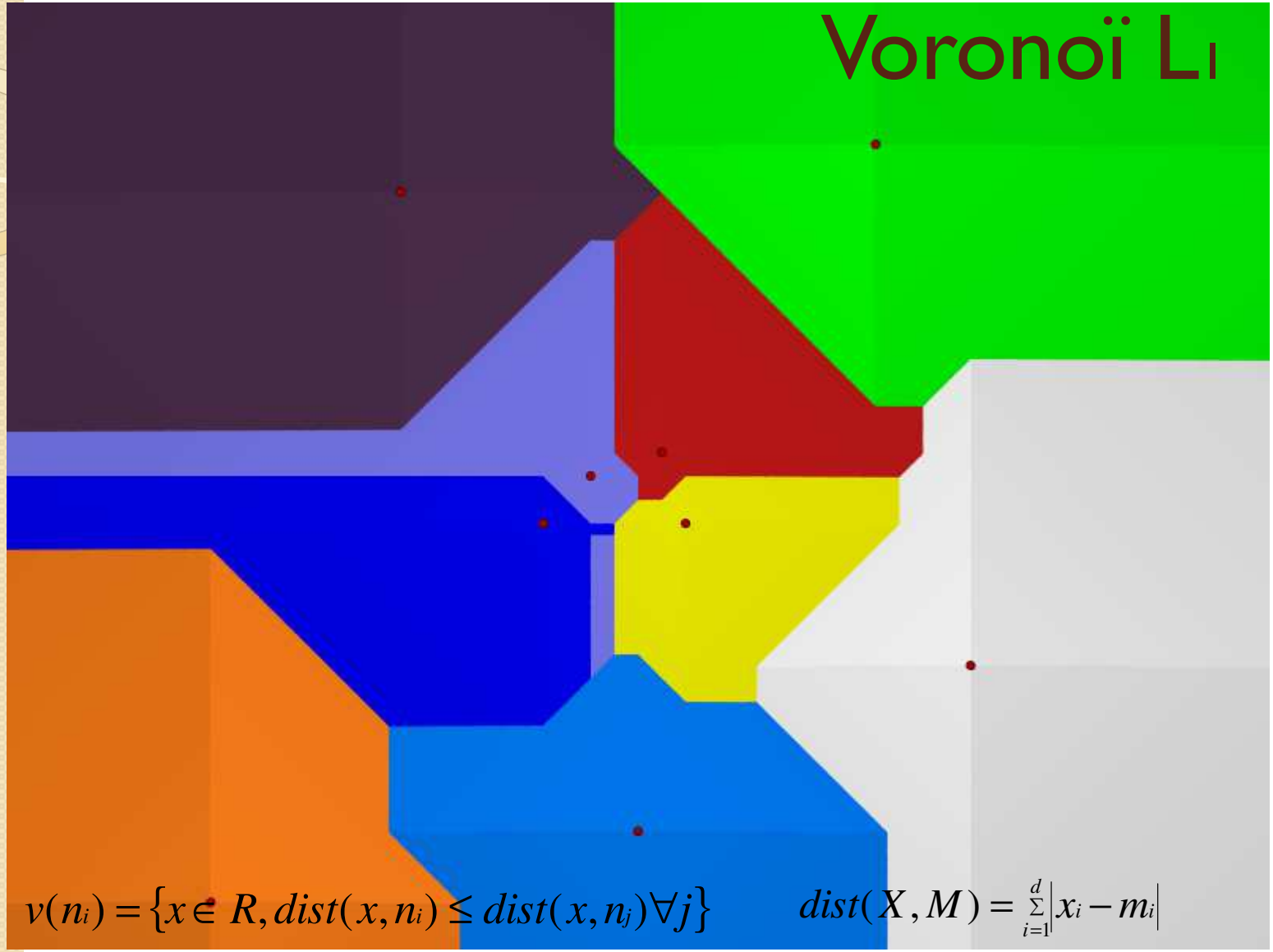
$$dist(X, M) = \max_{i=1,d} |x_i - m_i|$$

Dualité Delaunay/Voronoï en L_8



$$v(n_i) = \{x \in R, dist(x, n_i) \leq dist(x, n_j) \forall j\} \qquad dist(X, M) = \max_{i=1,d} |x_i - m_i|$$

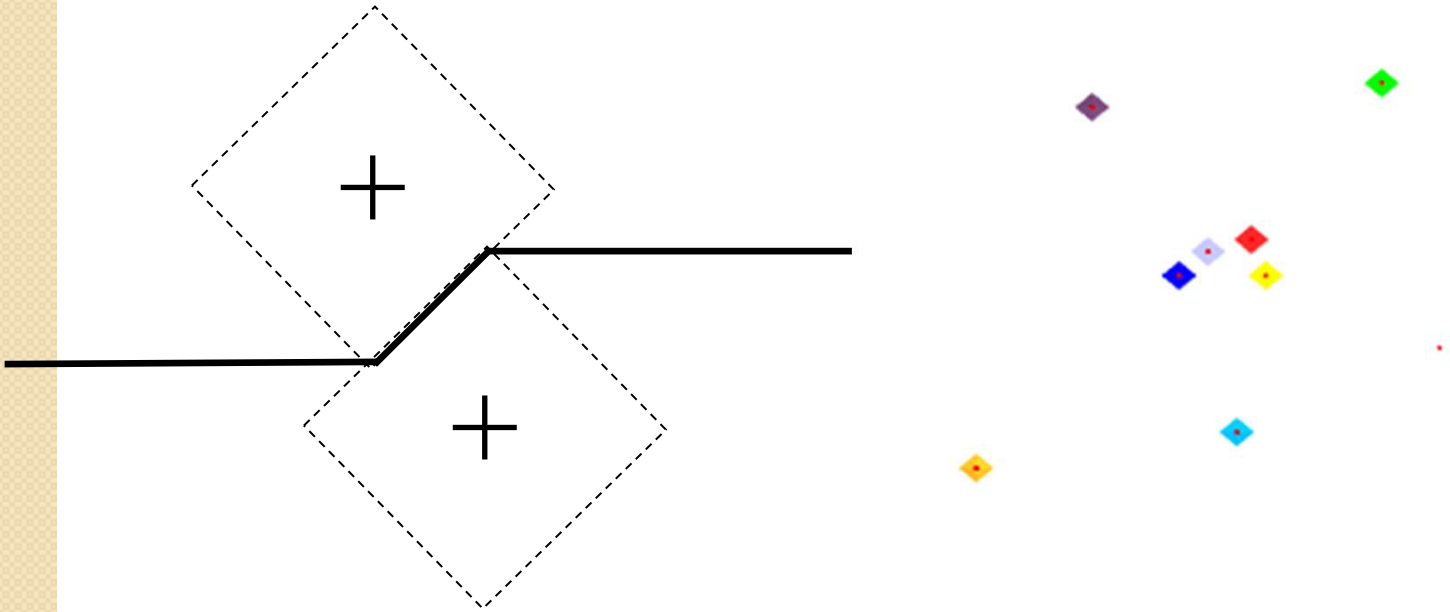
Voronoï L1



$$v(n_i) = \{x \in R, dist(x, n_i) \leq dist(x, n_j) \forall j\}$$

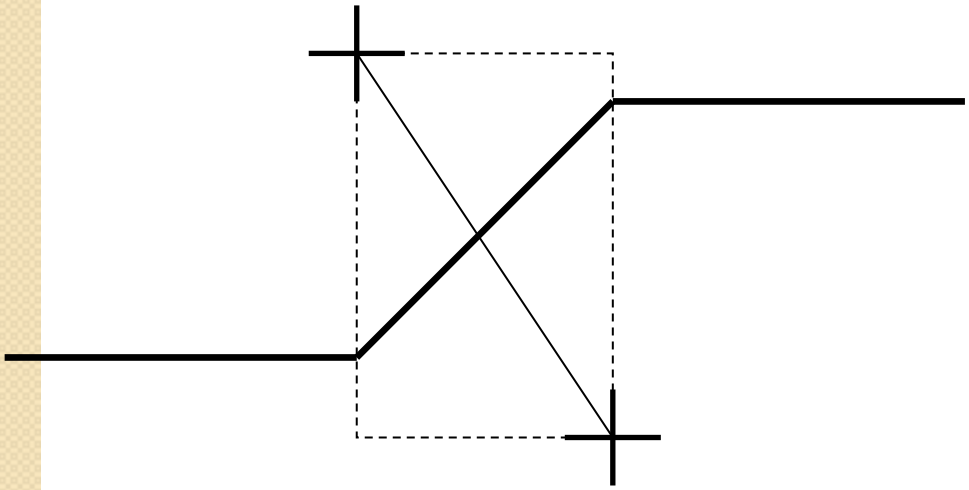
$$dist(X, M) = \sum_{i=1}^{d} |x_i - m_i|$$

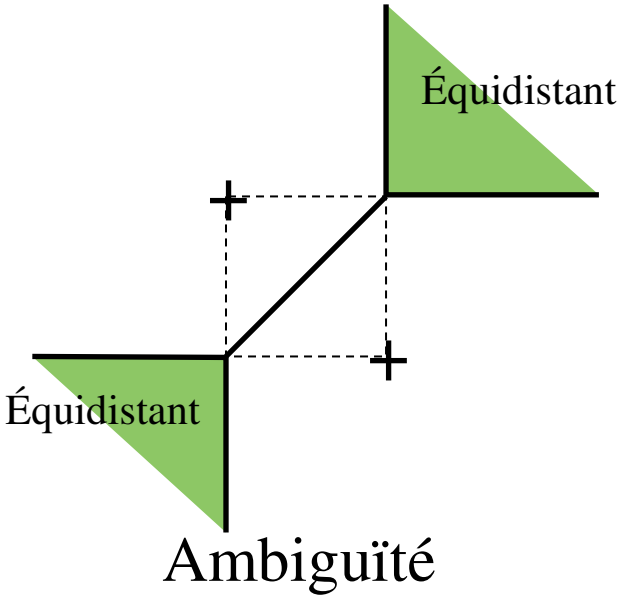
Voronoï métrique L_1



$$dist(X, M) = \sum_{i=1}^{d} |x_i - m_i|$$

Métrique L_1

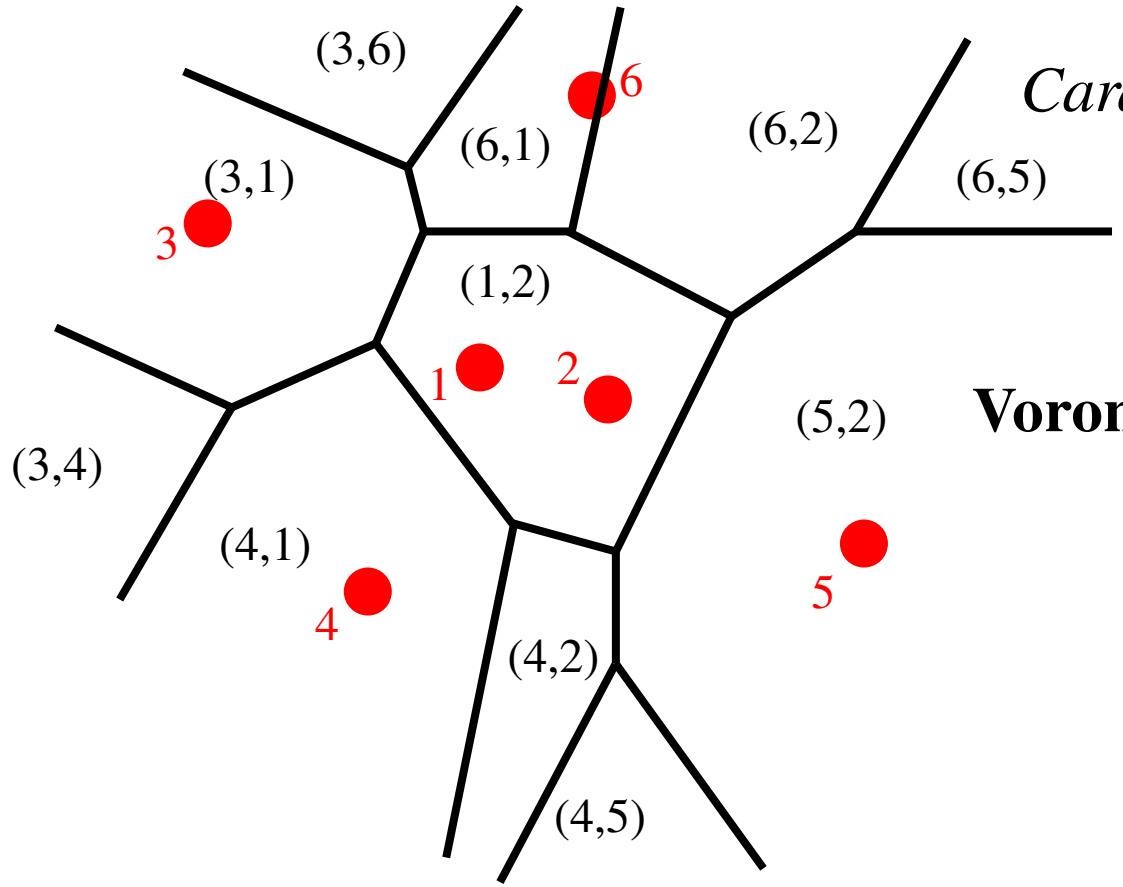




$$dist(X, M) = \sum_{i=1}^{d} |x_i - m_i|$$

Voronoi du 2ème ordre

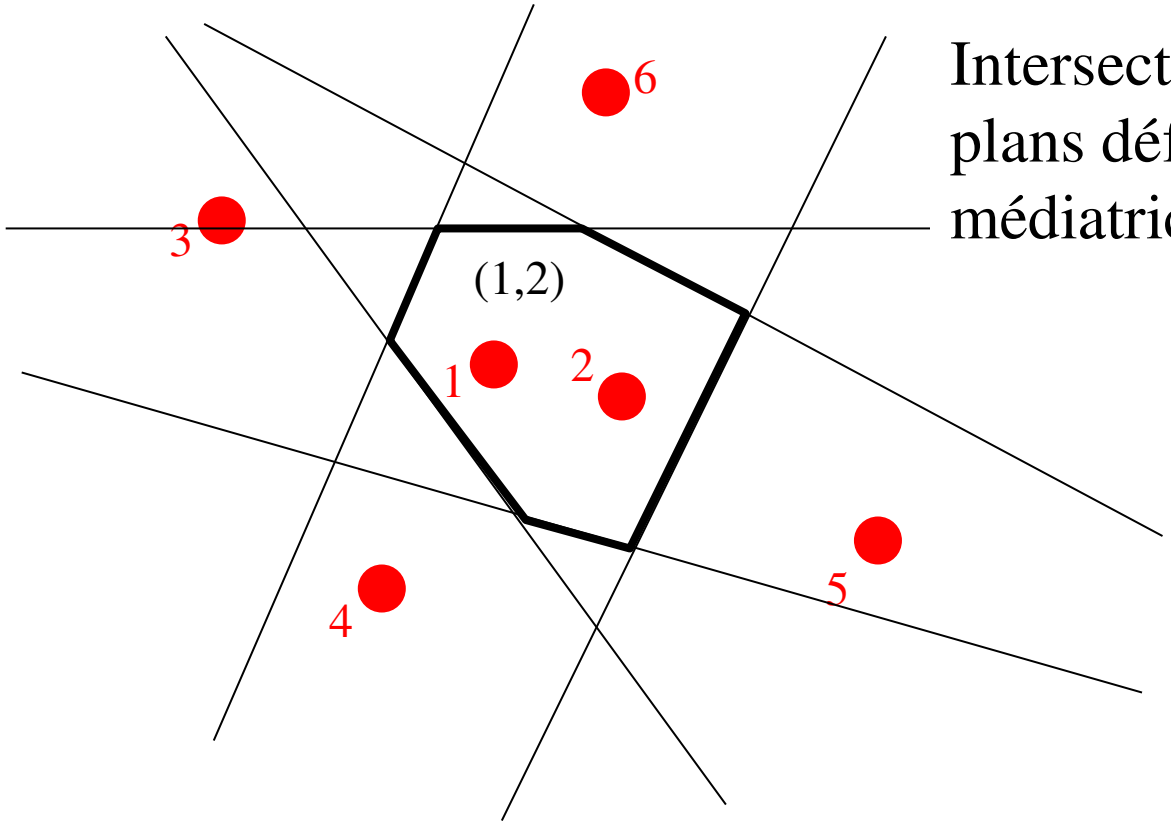
$$v(N_i) = \left\{ X \in R^d, \forall M_{ik} \in N_i, \forall M_{jk} \in N_j, dist(X, M_{ik}) \leq dist(X, M_{jk}) \forall j \right\}$$



$Card(N_i) = 2$ sites

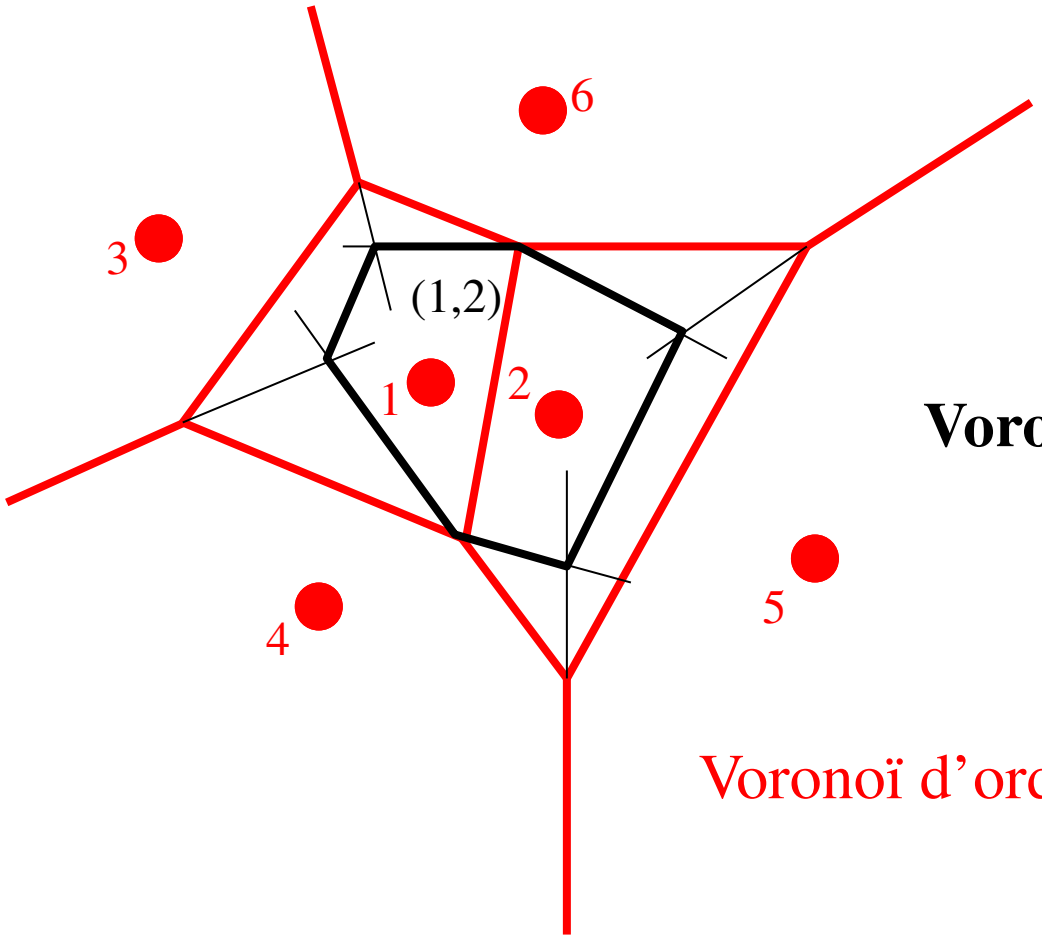
Voronoï d'ordre 2

Voronoi du 2ème ordre

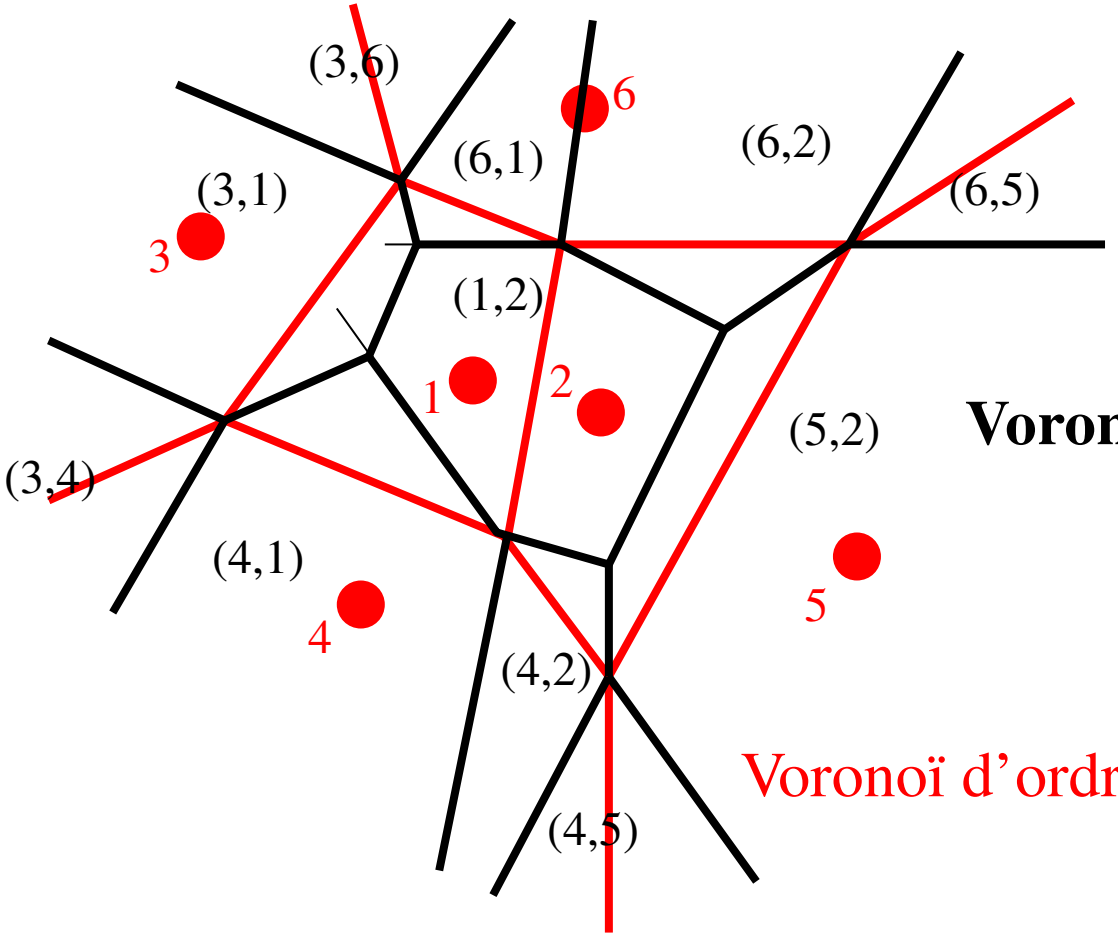


Intersection des ½ plans définit par les médiatrices.

Voronoi du 2ème ordre

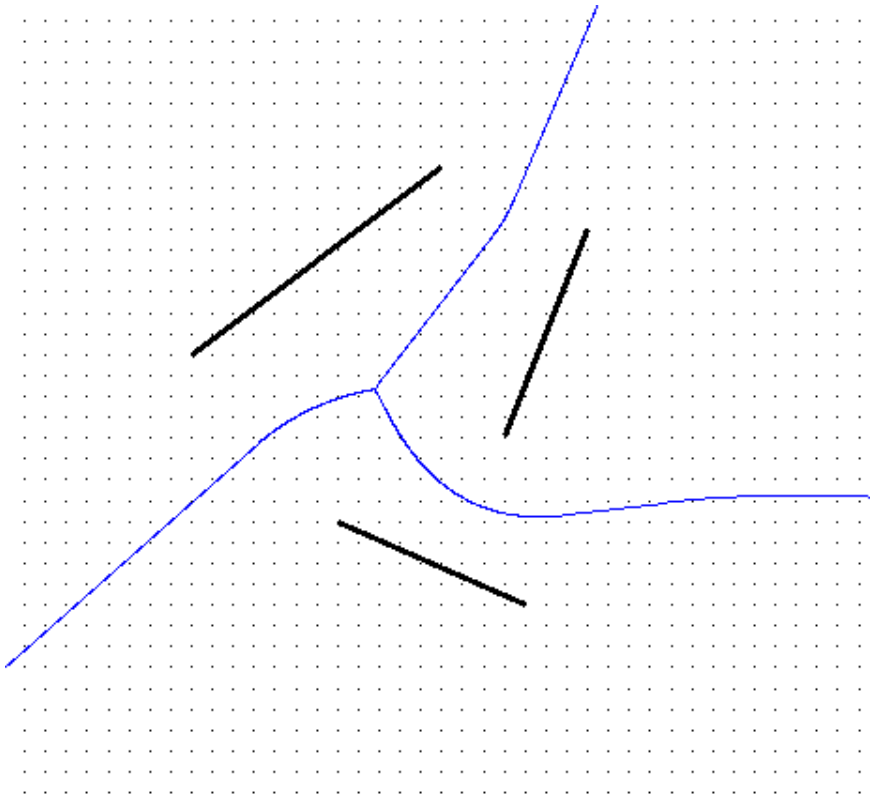


Voronoi du 2ème ordre

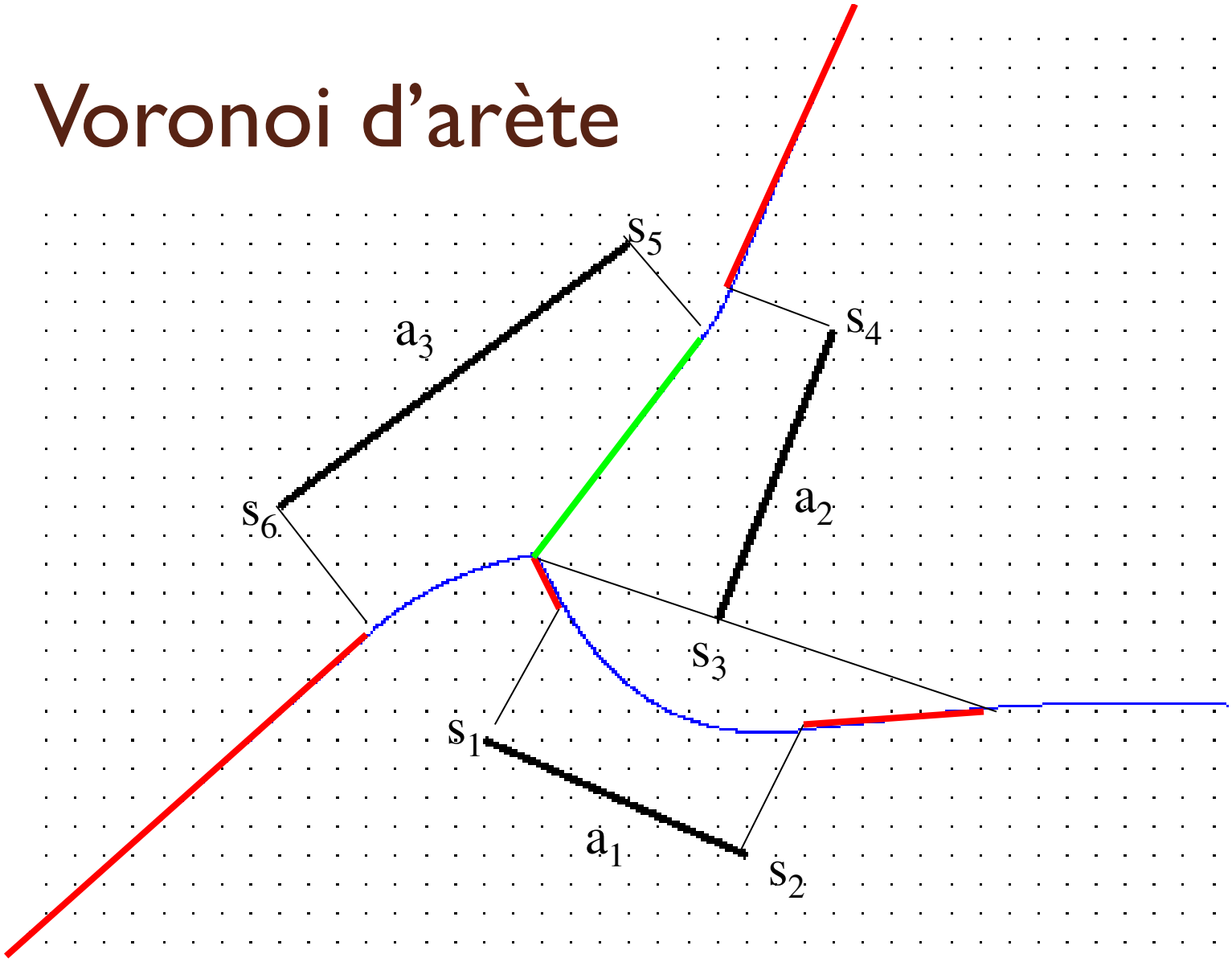


Voronoi d'arète

$$v(N_i) = \left\{ X \in R^d, \forall M_{ik} \in N_i, \forall M_{jk} \in N_j, dist(X, M_{ik}) \leq dist(X, M_{jk}) \forall j \right\}$$



Voronoi d'arète

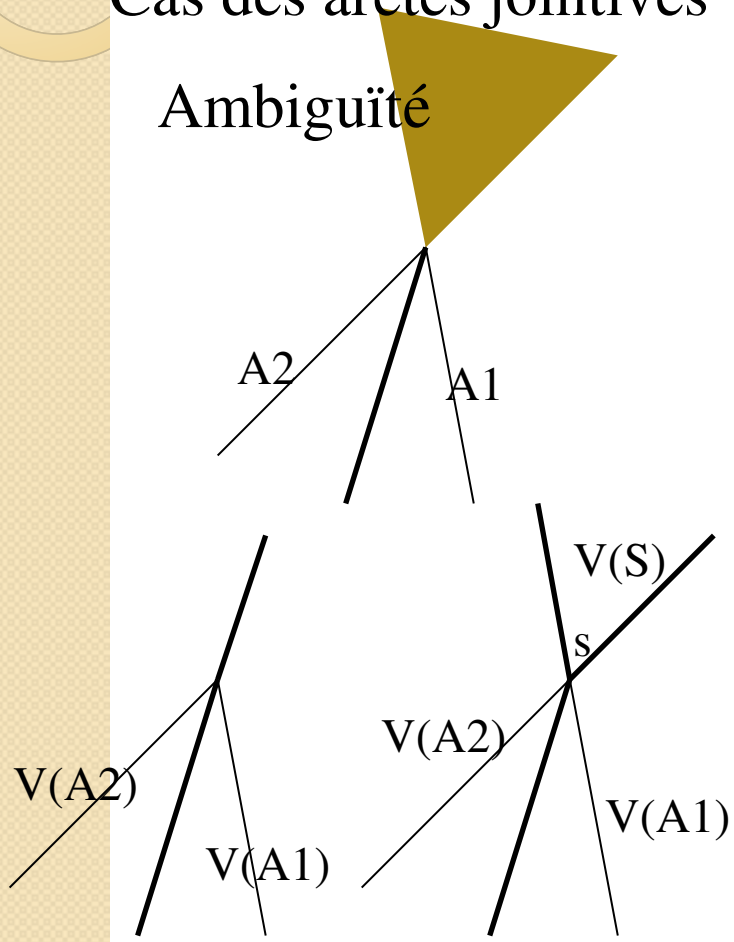


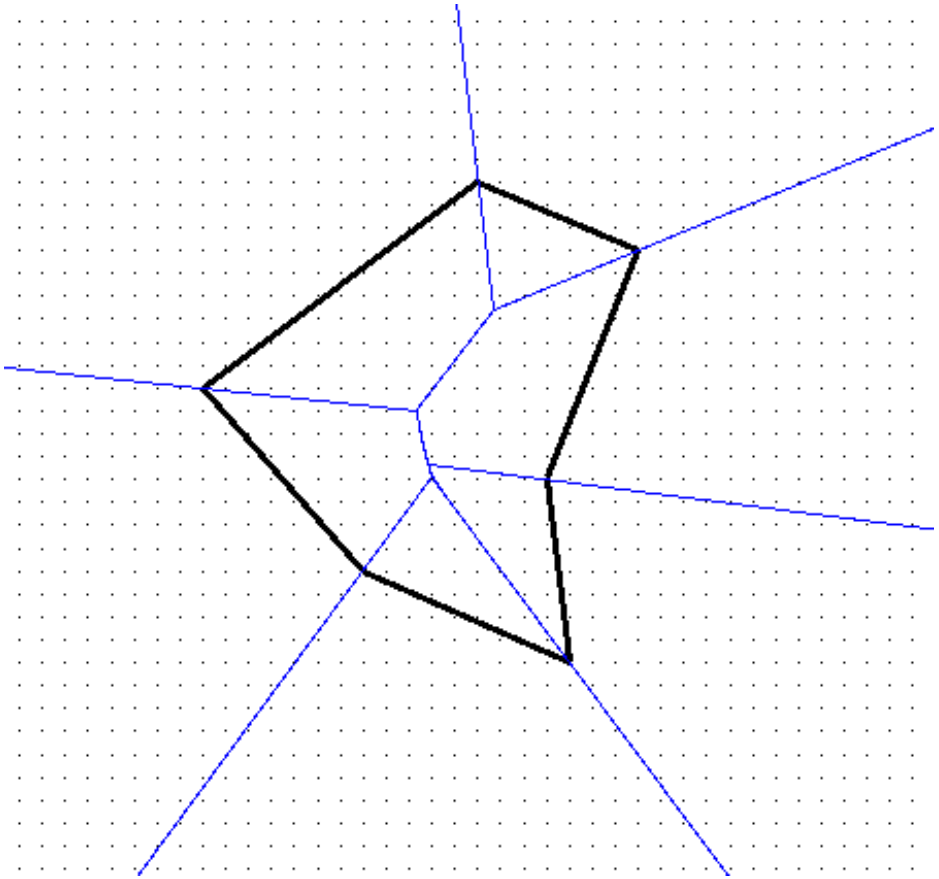
$$v(N_i) = \left\{ X \in R^d, \forall M_{ik} \in N_i, \forall M_{jk} \in N_j, dist(X, M_{ik}) \leq dist(X, M_{jk}) \forall j \right\}$$

Voronoi d'arètes

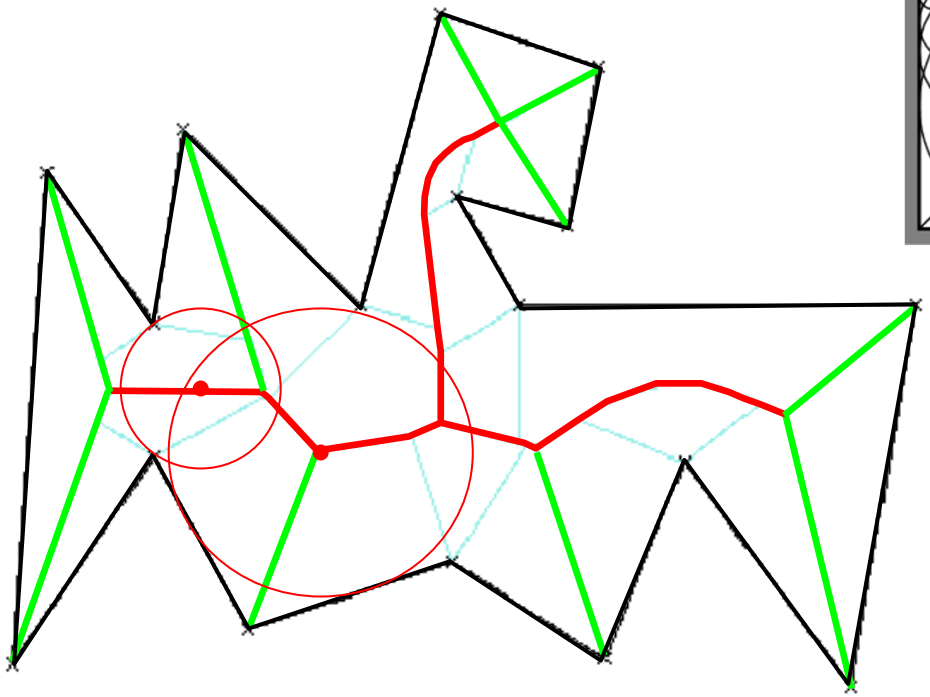
Cas des arètes jointives

Ambiguïté

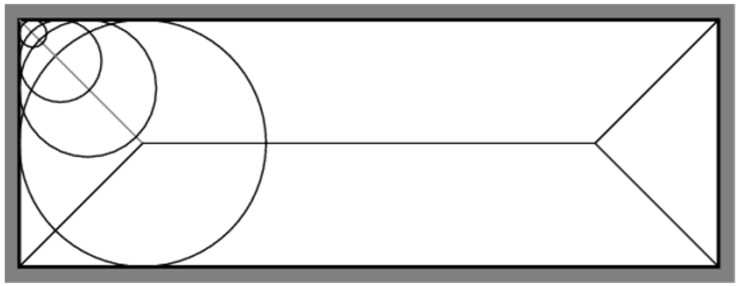




Axe médian/Voronoï

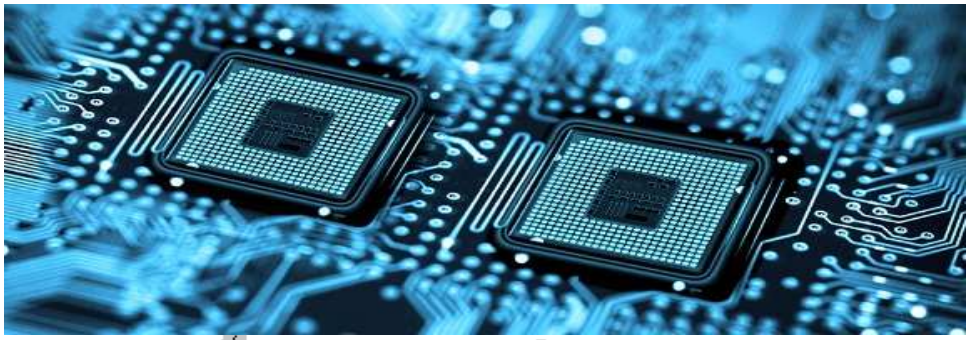


Le graphe de voronoï contient l'axe médian



L'axe médian est la trace du centre du cercle de rayon maximum, intérieur au polygone

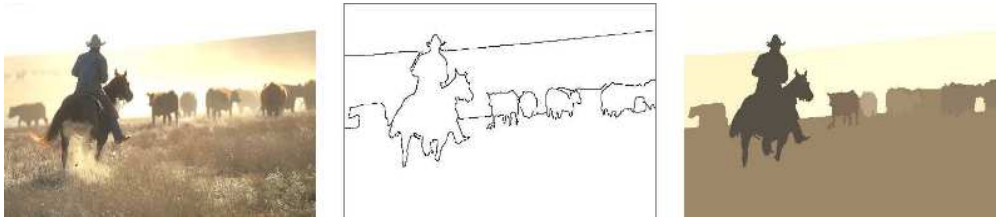
Applications de «Voronoï »



Analyse de zones critiques des circuits VLSI (IBM)

Reconnaissance des formes

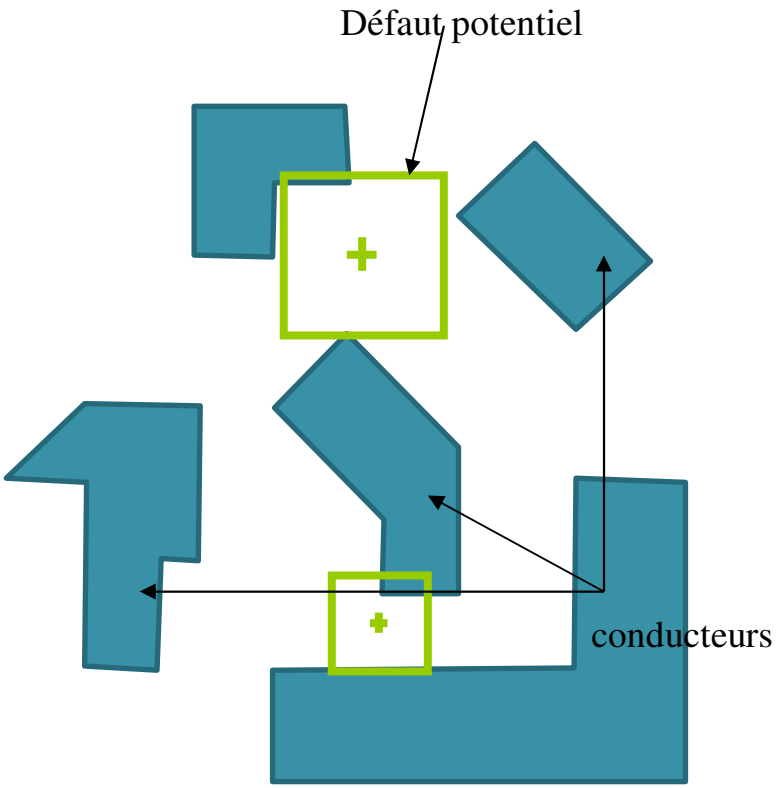
Analyse et synthèse de texture (de fracturation)



Segmentation et compression d'image

Analyse des zones critiques VLSI

Rayon des défauts (carrés) provoquant des court-circuits



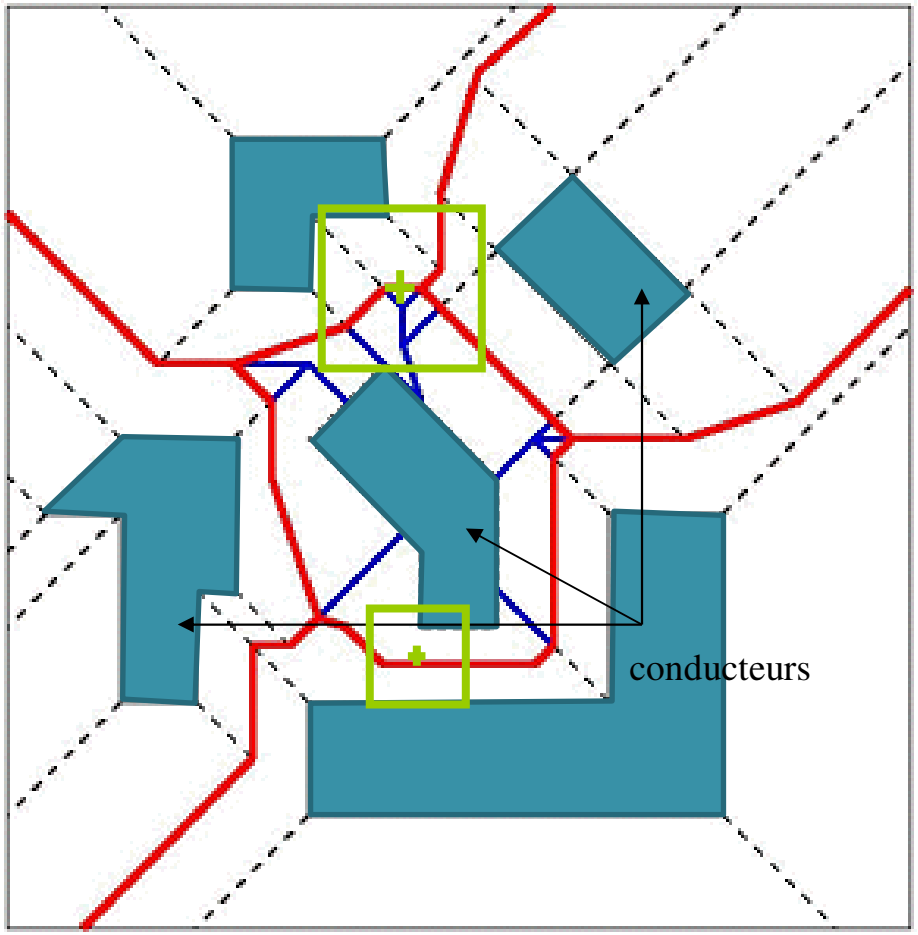
Comparer le problème à celui de « l'usine polluante ».

Que peut-on en déduire ?

(Source IBM)

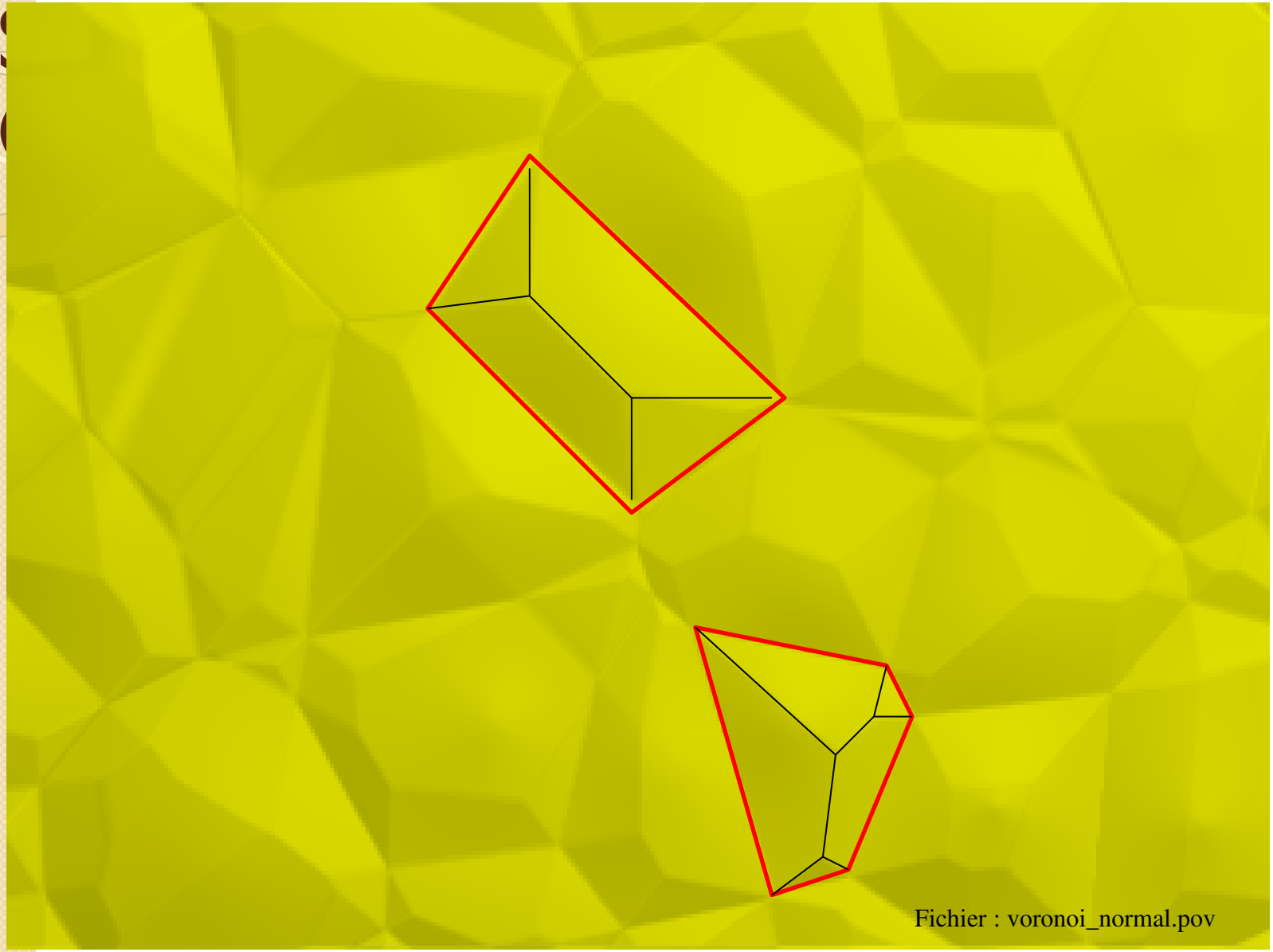
Analyse des zones critiques VLSI

Rayon des défauts (carrés) provoquant des court-circuits

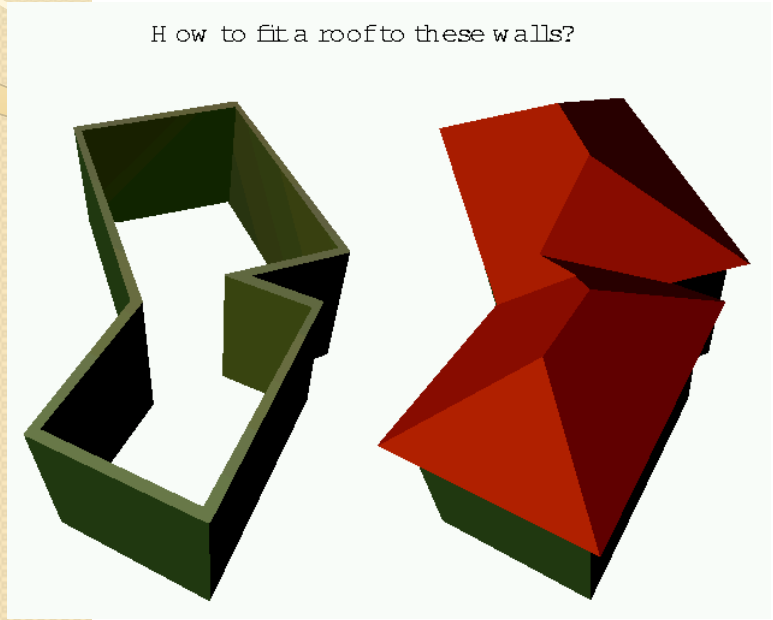


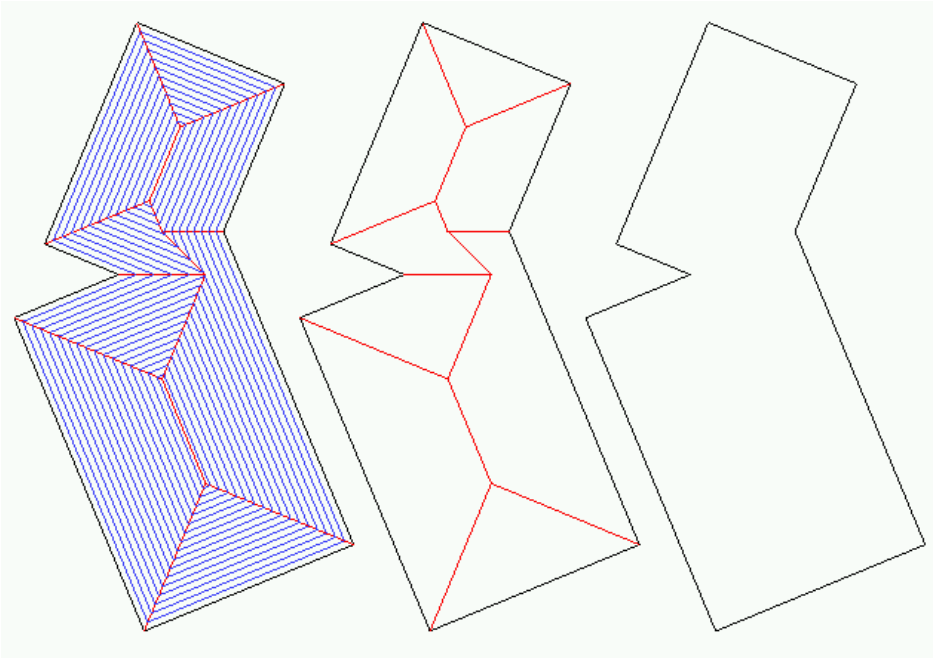
(Source IBM)

Pour répondre au problème il faut construire un Voronoï d'arêtes du second ordre en métrique L_8


Fichier : voronoi_normal.pov

Quelle toiture pour ma maison ?





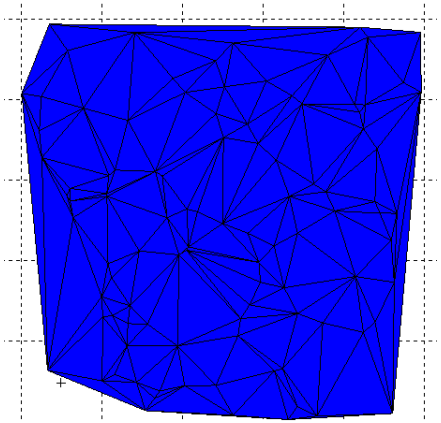
www.sable.mcgill.ca/~dbelan2/roofs/roofs.html.

Bibliographie

- Introduction à la modélisation et à l'algorithmique géométrique
  O.Stab *(people.mines-paristech.fr/olivier.stab/geomodeling.pdf)*
- Computational Geometry, an introduction
  M.I.Shamos, F.P. Preparata, *Springer-Verlag, 1998*
- Computational Geometry Algorithms Library
  CGAL, http://www.cgal.org
- Géométrie Algorithmique, J.D.Boissonnat et M.Yvinec,
  Ediscience international 1995
- Computational Geometry, Algorithms and Applications
  M.Berg et al. *Springer 1997*
- Computational Geometry in C, J.O.Rourke, *Cambridge Press 1998*

TP : Triangulations de Delaunay

1. Triangulation de Delaunay



2. Extraction de l'EC

3. Conversion Delaunay➔ Voronoï