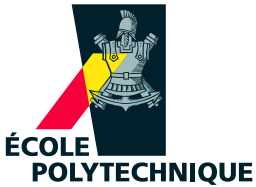
Lecture 4: Proximity search

INF562, 28 janvier 2014

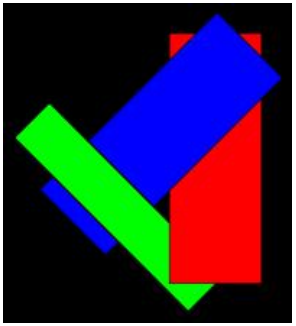
Luca Castelli Aleardi



Motivation

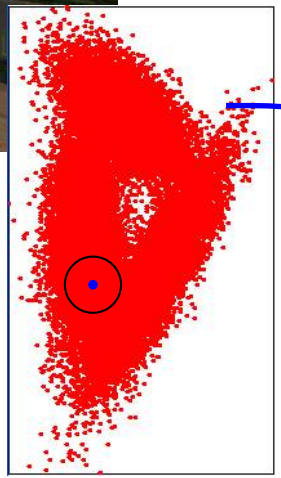
Geometric proximity search: applications

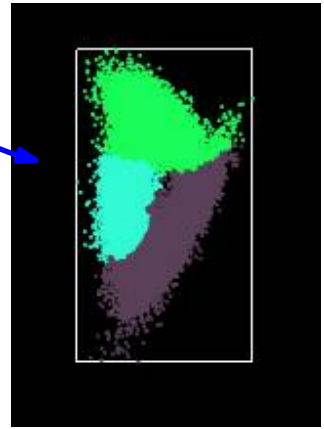
Painter's algorithm

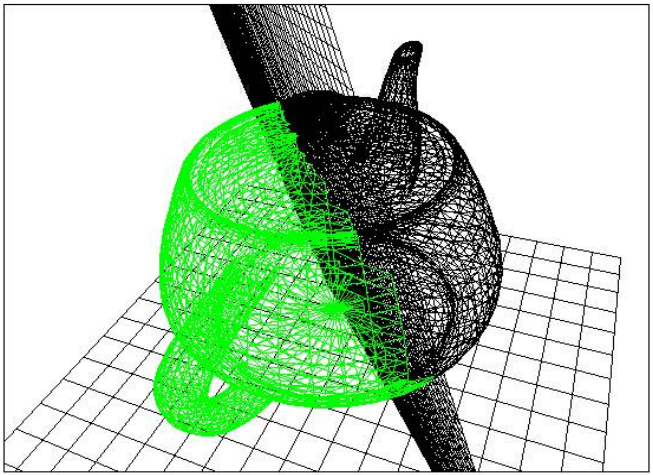


BSP for 3D rendering (Doom)

Clustering

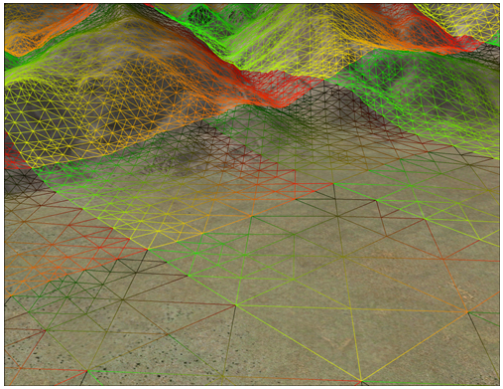




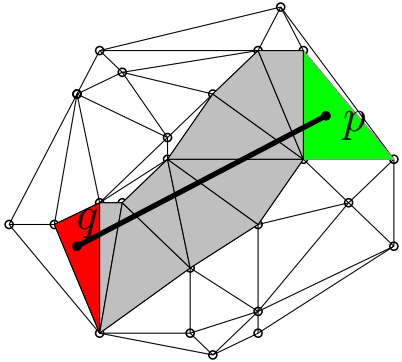


Collision detection

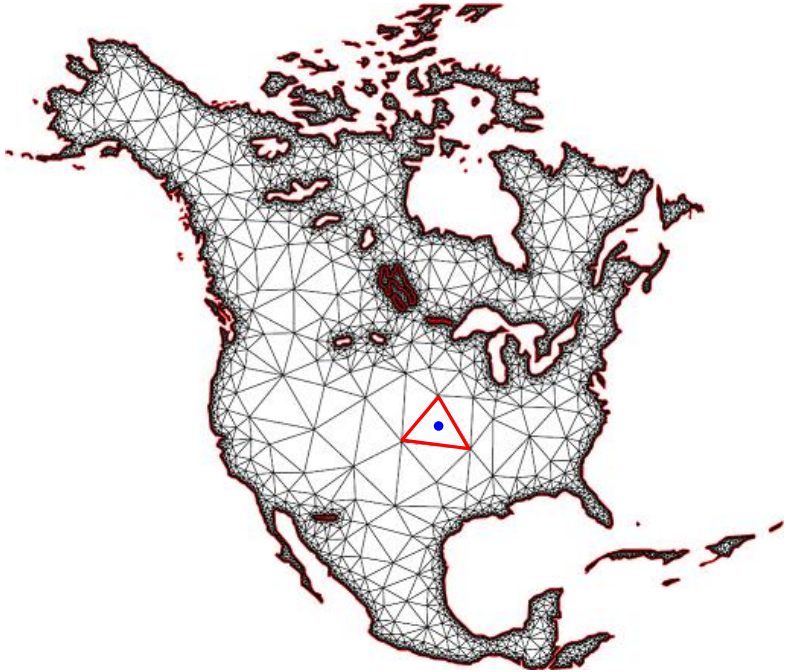
Planar point location: geographic localisation



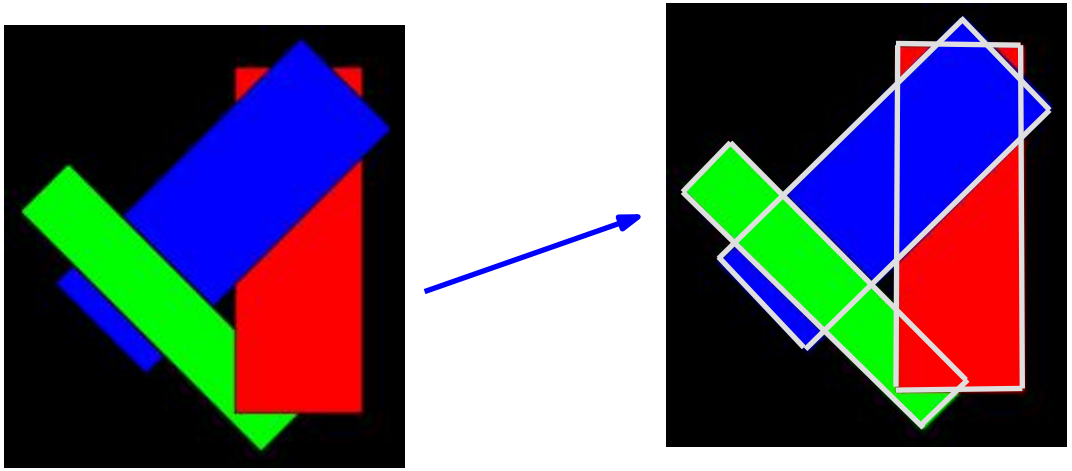
MS flight simulator

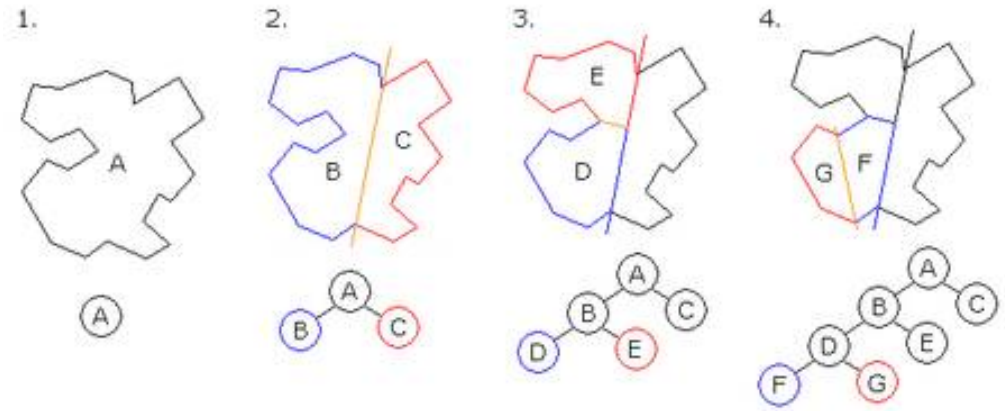


Maillage de L. Rineau, M. Yvinec

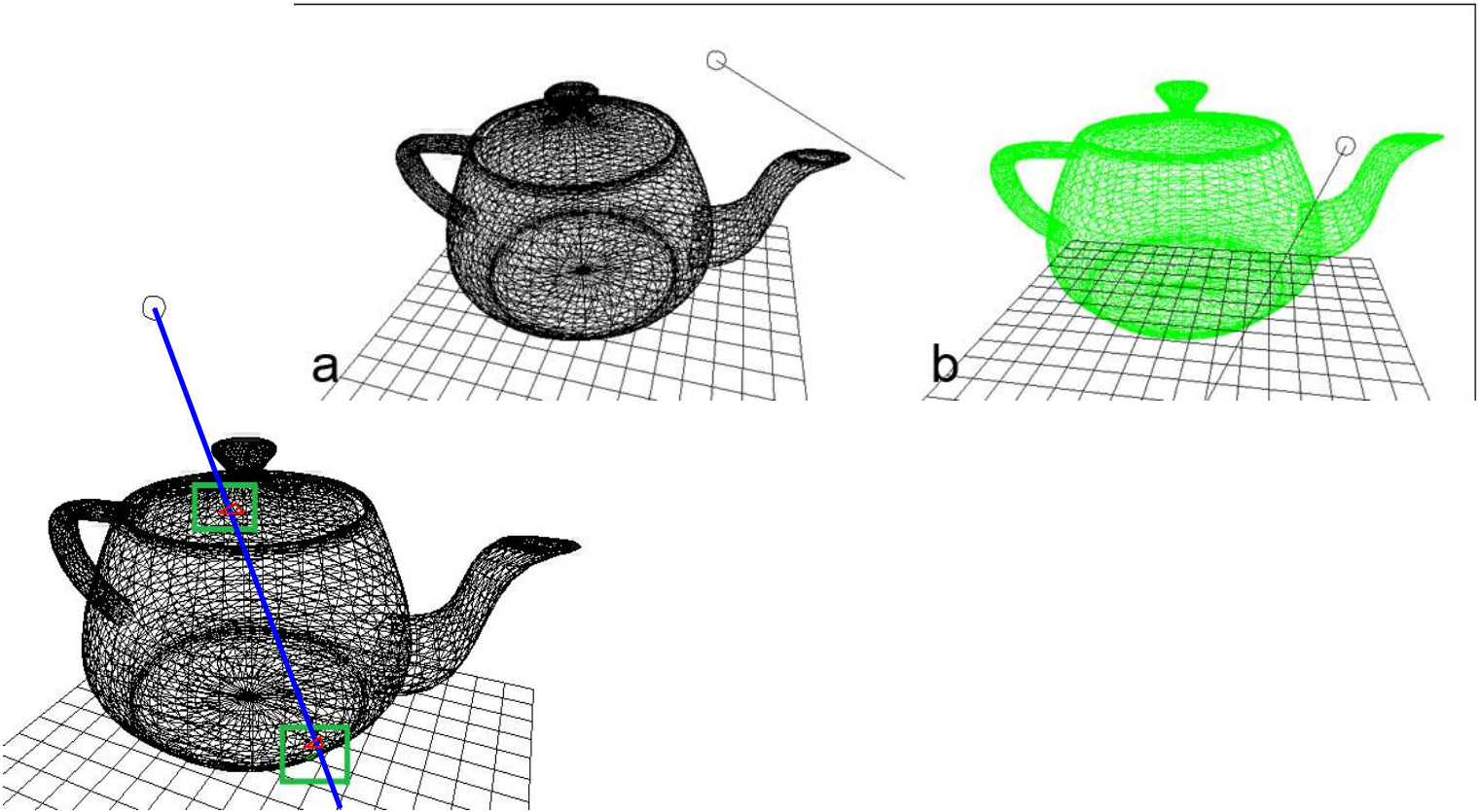


Binary space partitions: painter's algorithm

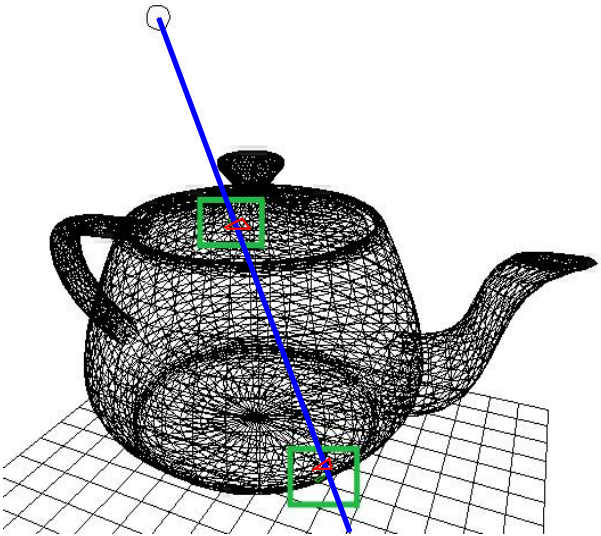




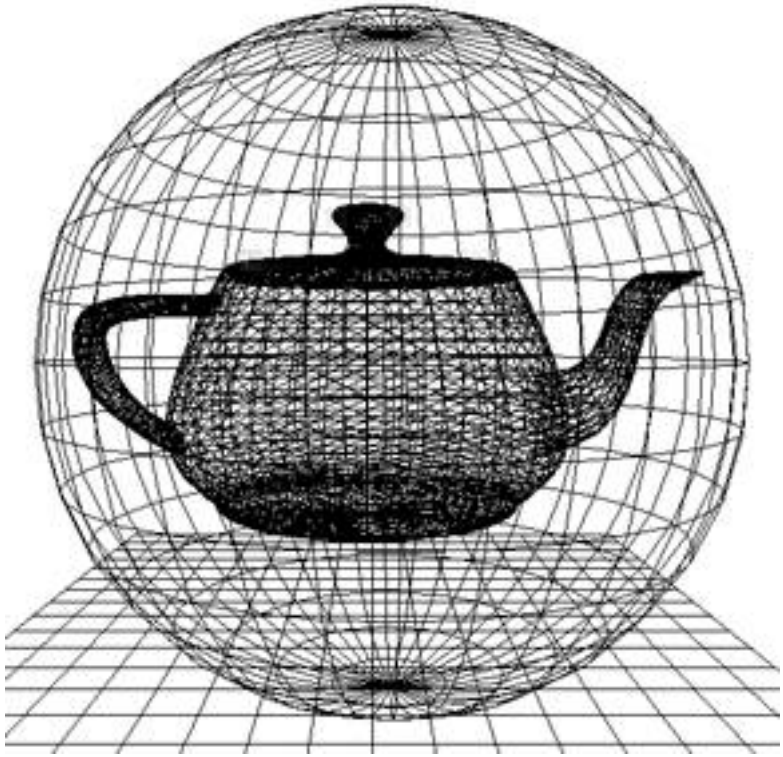
Binary space partitions: ray tracing, collision detection



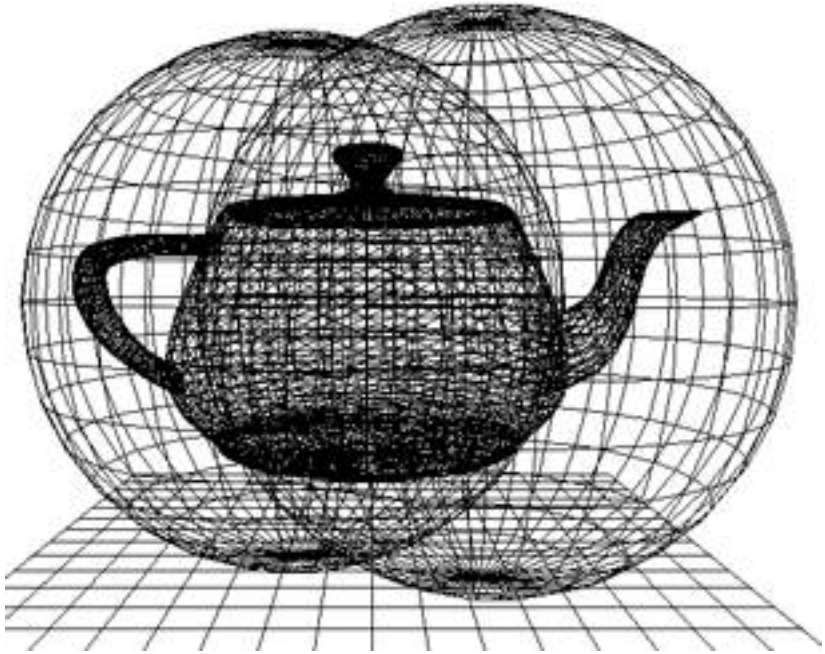
Binary space partitions: ray tracing, collision detection



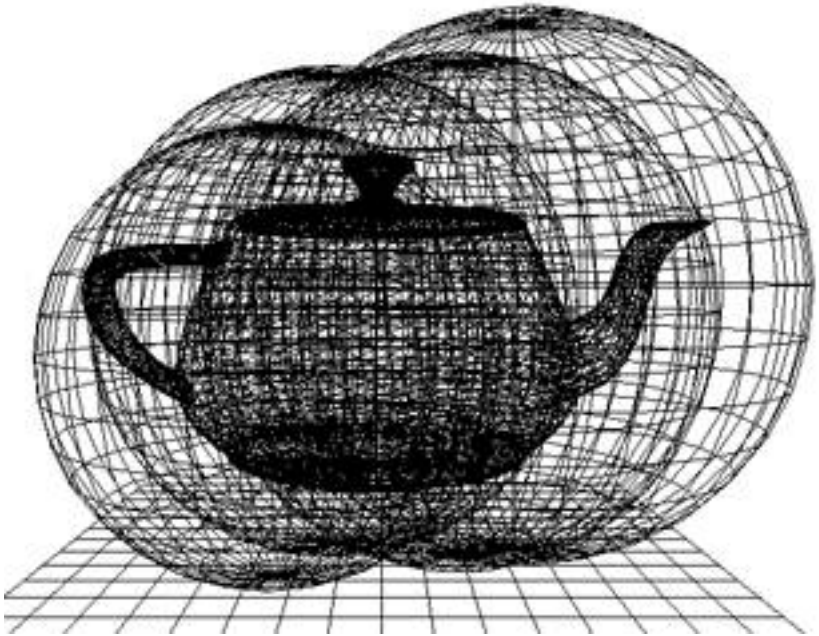
Binary space partitions: ray tracing, collision detection



Binary space partitions: ray tracing, collision detection

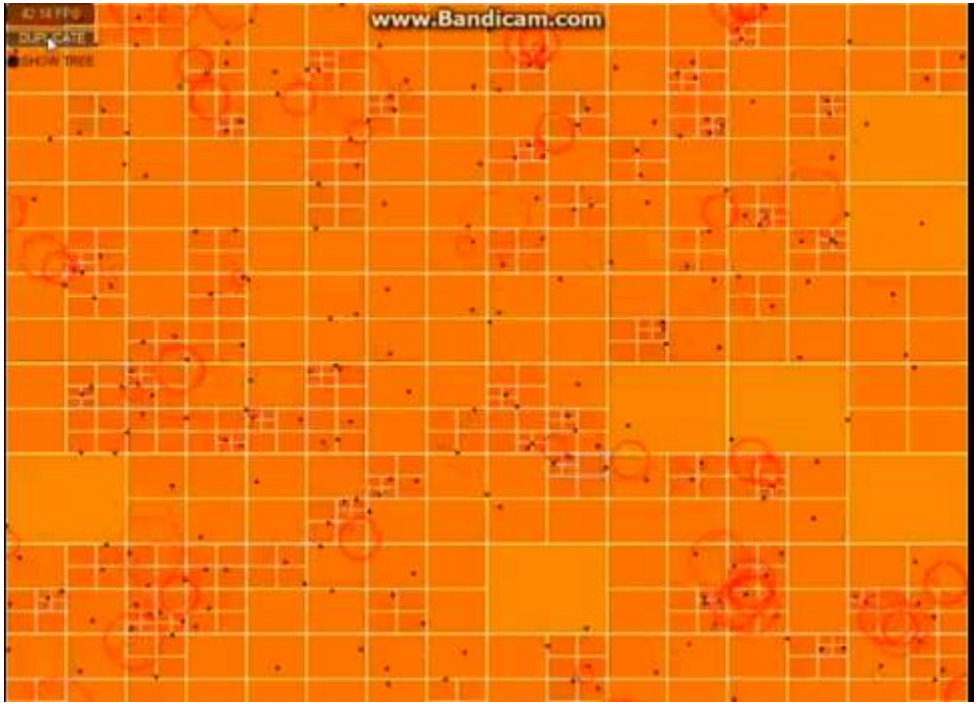


Binary space partitions: ray tracing, collision detection



Collision detection in 2D

Collision detection in 2D

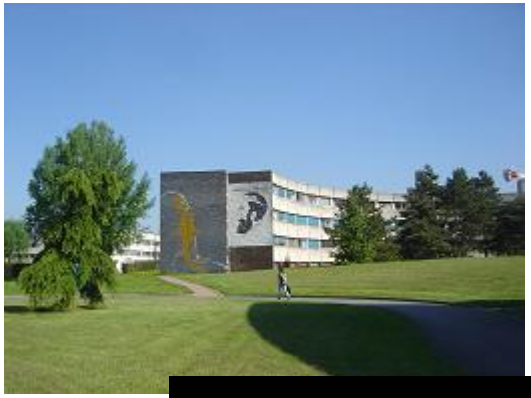


Nearest neighbor search (Kd-Trees)

Clustering et segmentation d'images

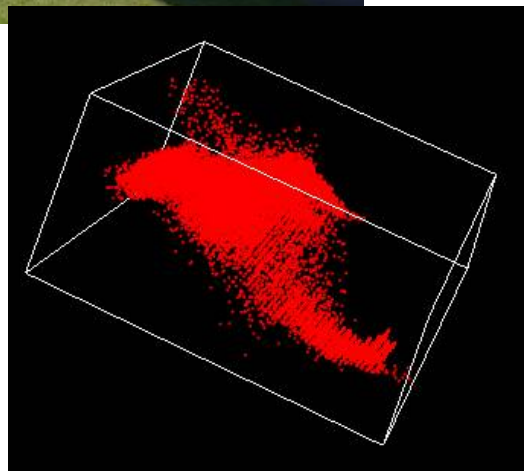
segmentation

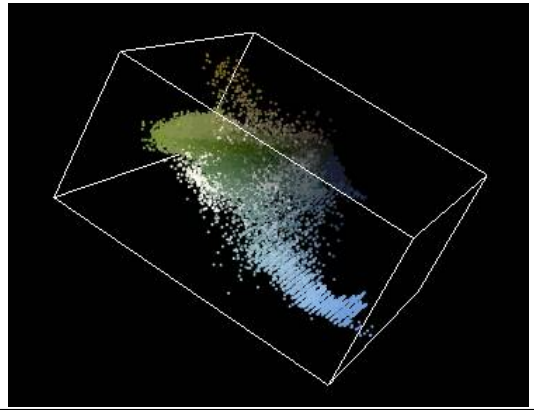
Nearest neighbor search (Kd-Trees)
Clustering et segmentation d'images



segmentation

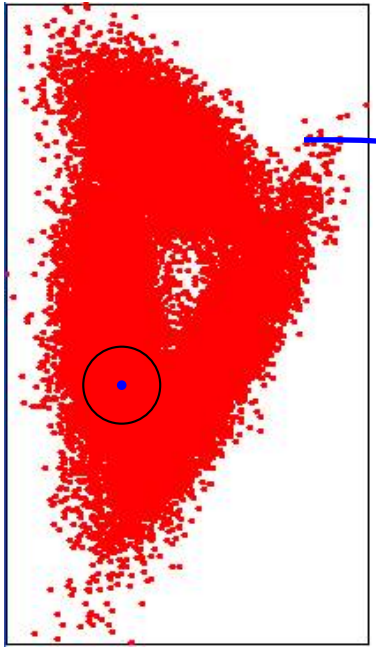
Luv

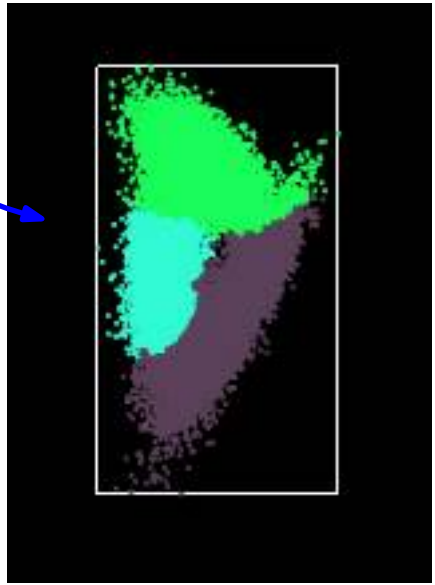


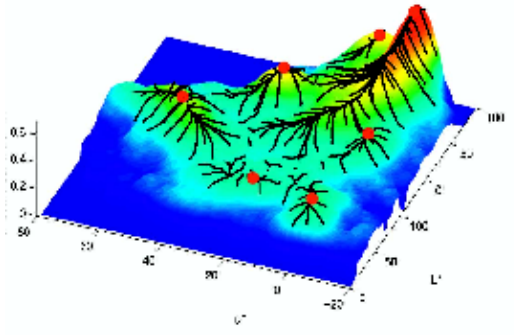


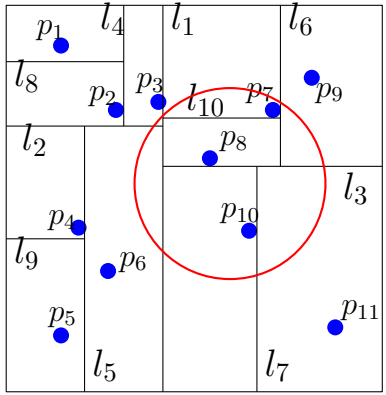
Nearest neighbor search (Kd-Trees)

Clustering et segmentation d'images



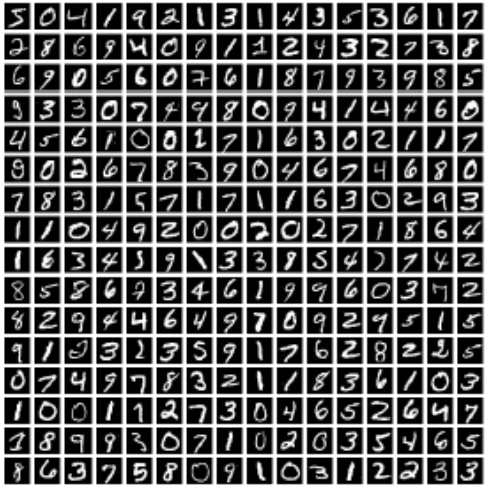


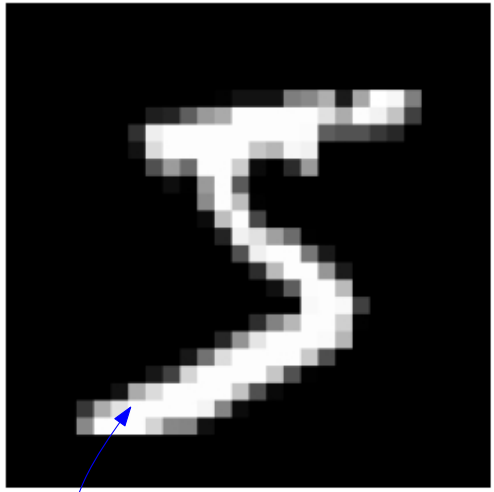




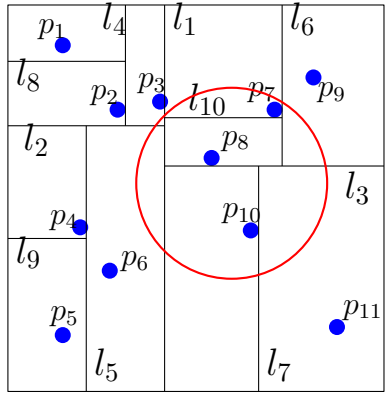
K-Nearest neighbors (Kd-Trees)

Pattern recognition (OCR)





$$p_i \in 256^{28\times 28}$$

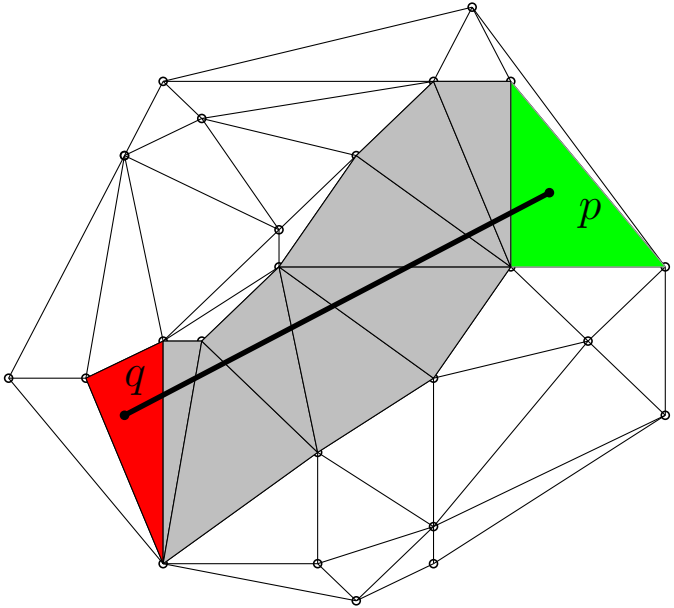


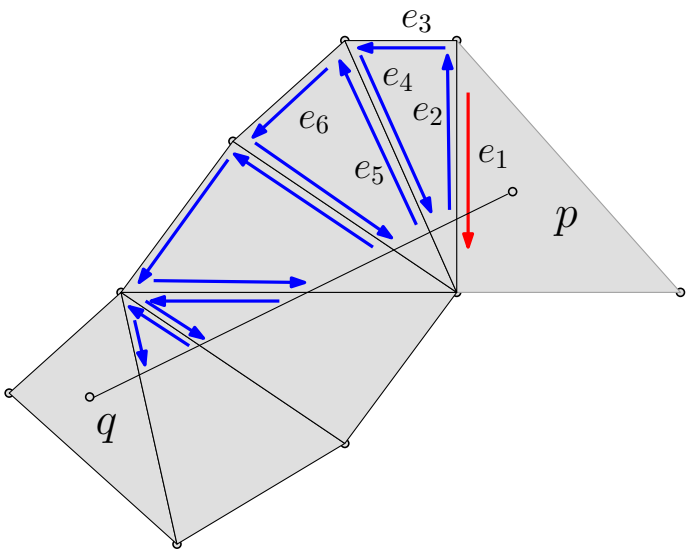
Part I: planar point location

Point location in planar subdivisions

(visibility walk)

complexité $\mathbf{O(\sqrt{n})}$

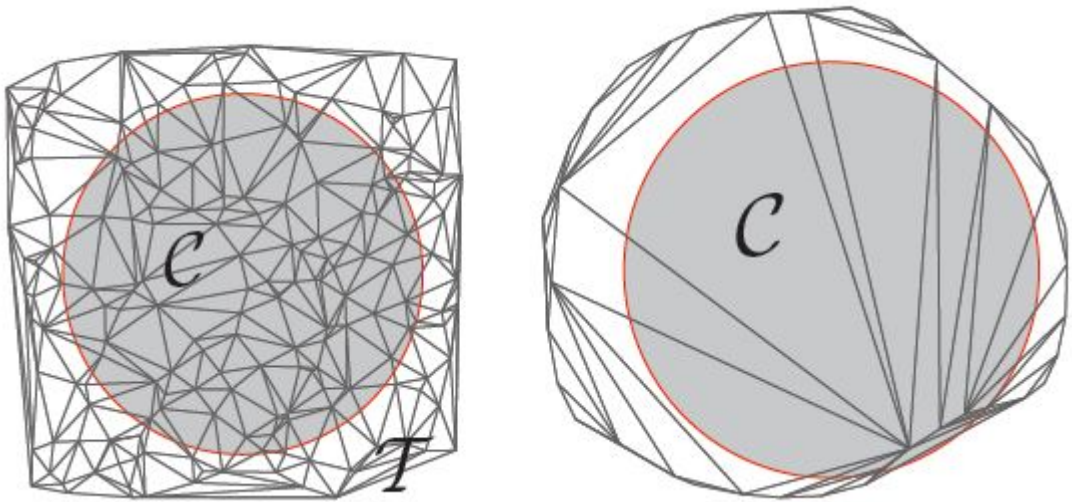




Point location in planar subdivisions

(visibility walk)

complexité $\mathbf{O(\sqrt{n})}$



Point location in planar subdivisions

(Jump and Walk)

$\mathbf{k = n^{\frac{1}{d+1}}}$ complexity $O(n^{1/3})$

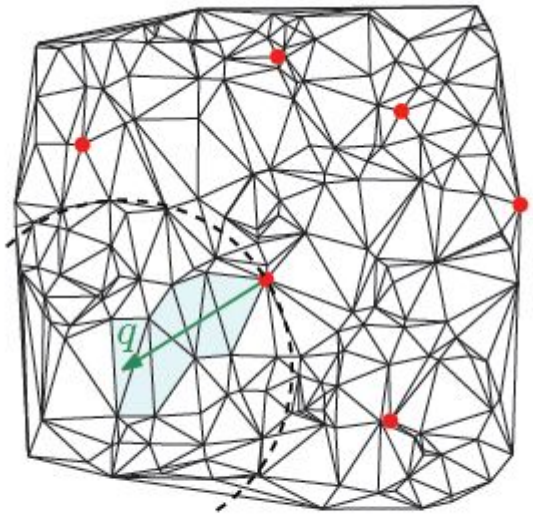


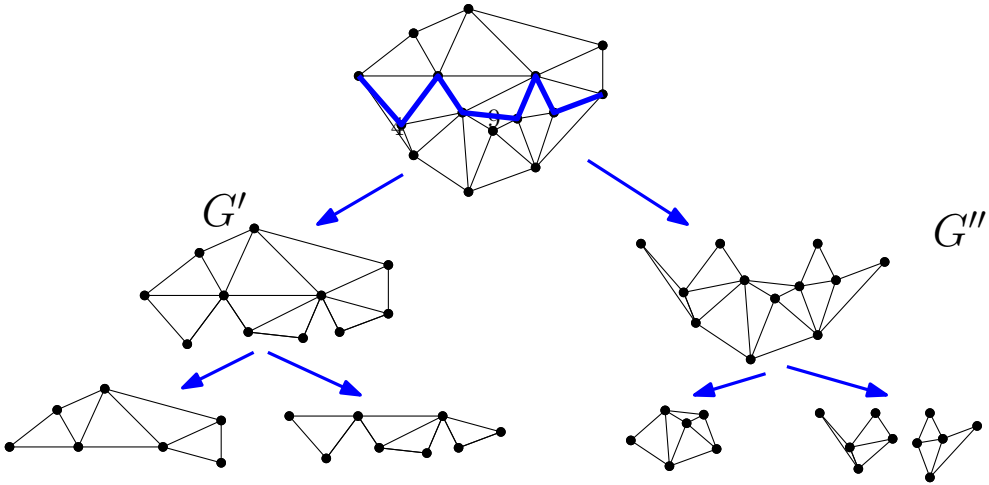
Figure 5: **Jump & Walk.** *The walk starts from the nearest landmark (represented by dots above) with respect to the query. Then it ends at the cell containing the query.*

Point location in planar subdivisions

$\mathbf{O}(\log^{\mathbf{2}} \mathbf{n})$

Thm (Edelsbrunner, Guibas, Stolfi)

Etant donnée une triangulation planaire de taille n, on peut effectuer la localisation d'un point en temps $\mathbf{O}(\log^{\mathbf{2}} \mathbf{n})$, la construction de la structure de donnees se faisant en espace $O(n^2)$.



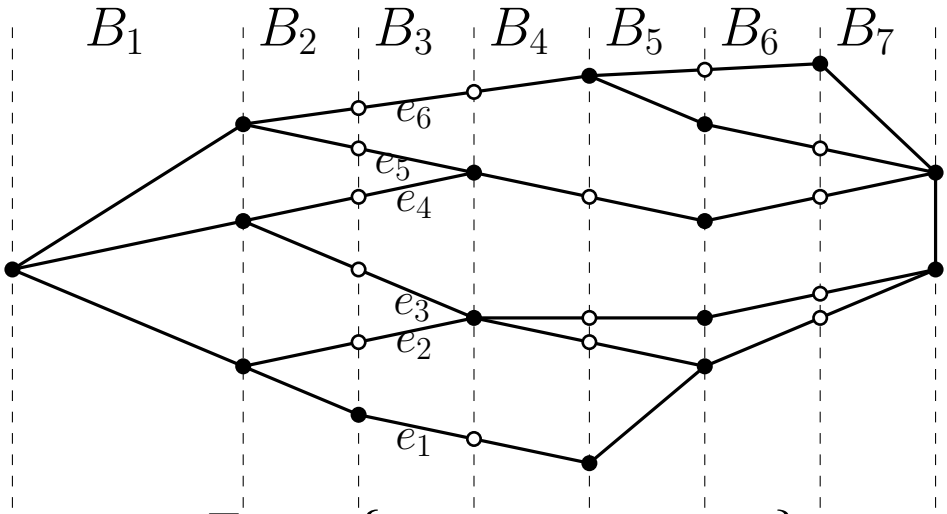
Methode du *Slab*

Methode du *Slab*

$\mathbf{O}(\log \mathbf{n})$

Thm (Dobkin, Lipton '76)

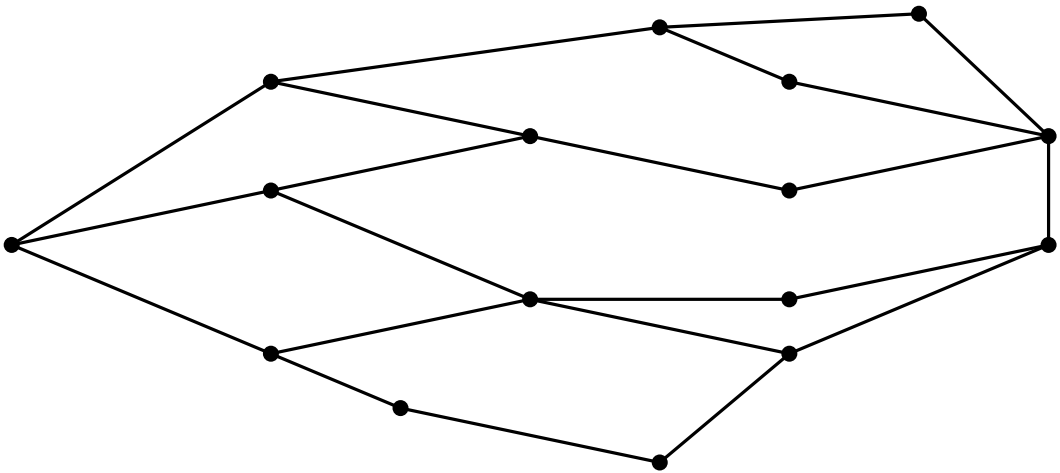
Etant donnée une subdivision planaire de taille n, on peut effectuer la localisation d'un point en temps $\mathbf{O}(\log \mathbf{n})$, la construction de la structure de donnees se faisant en temps $O(n^2 \log n)$. La taille de la structure est $O(n^2)$.



$$E_{B_3} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$$

Methode du *Slab*

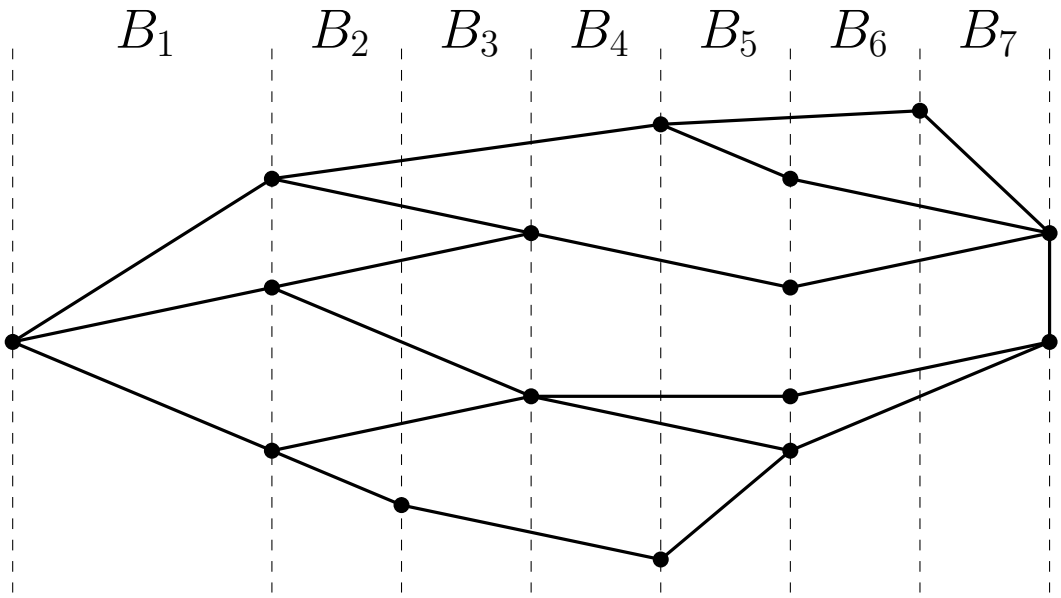
O(n) sommets, faces et aretes



Methode du *Slab*

O(**n**) bandes verticales

O(**n**) sommets, faces et aretes

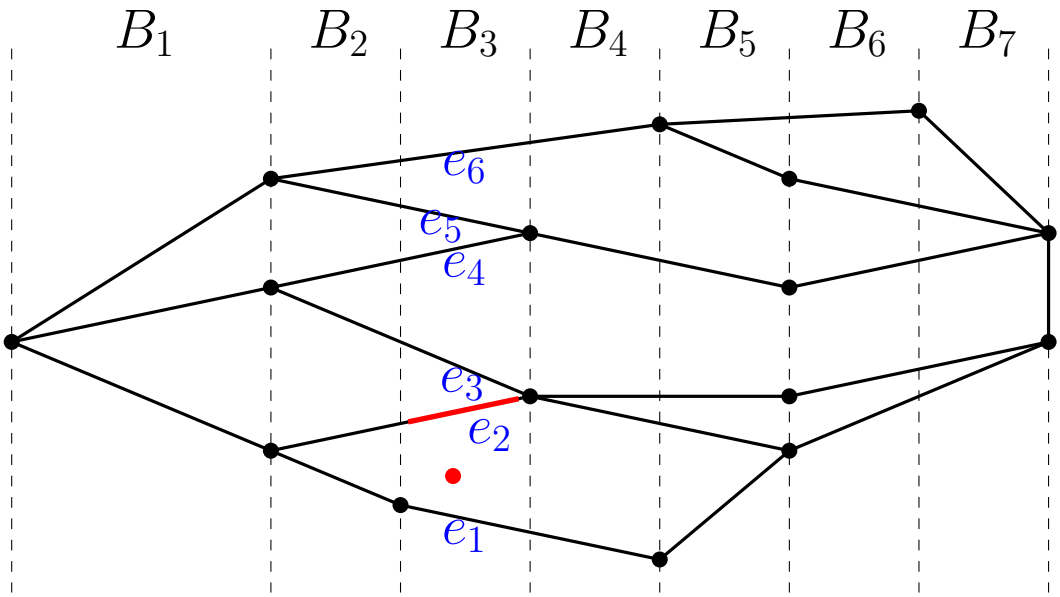


Methode du *Slab*

$O(n)$ aretes et faces dans chaque bande

O(n) bandes verticales

O(n) sommets, faces et aretes



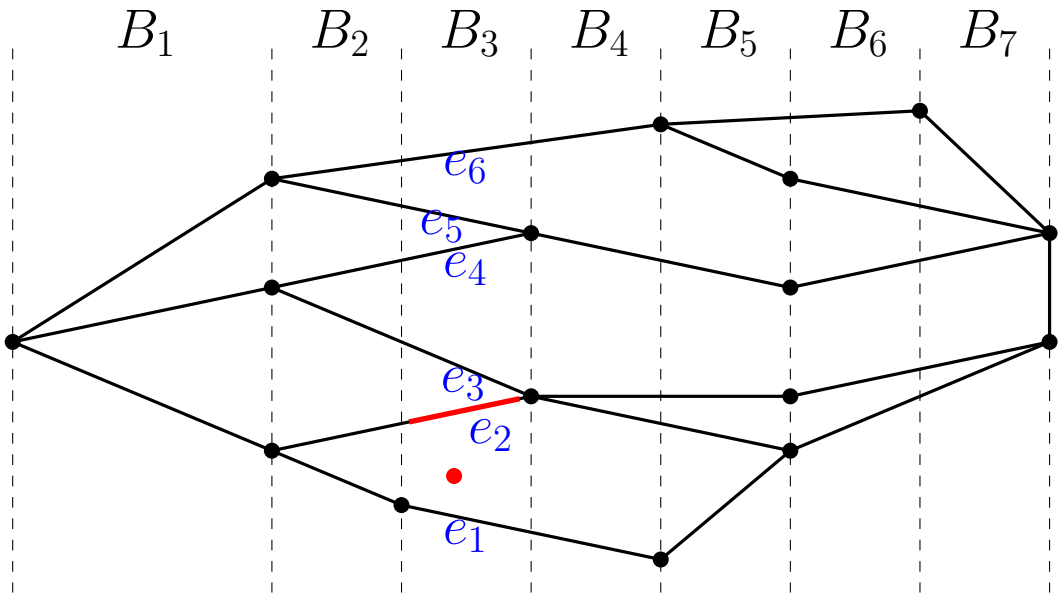
$E_{B_3} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

Methode du *Slab*

$O(n)$ aretes et faces dans chaque bande

O(n) bandes verticales

O(n) sommets, faces et aretes



$E_{B_3} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

temps $O(\log n)$ pour localiser B_3

temps $O(\log n)$ pour localiser e_2 dans E_{B_3}

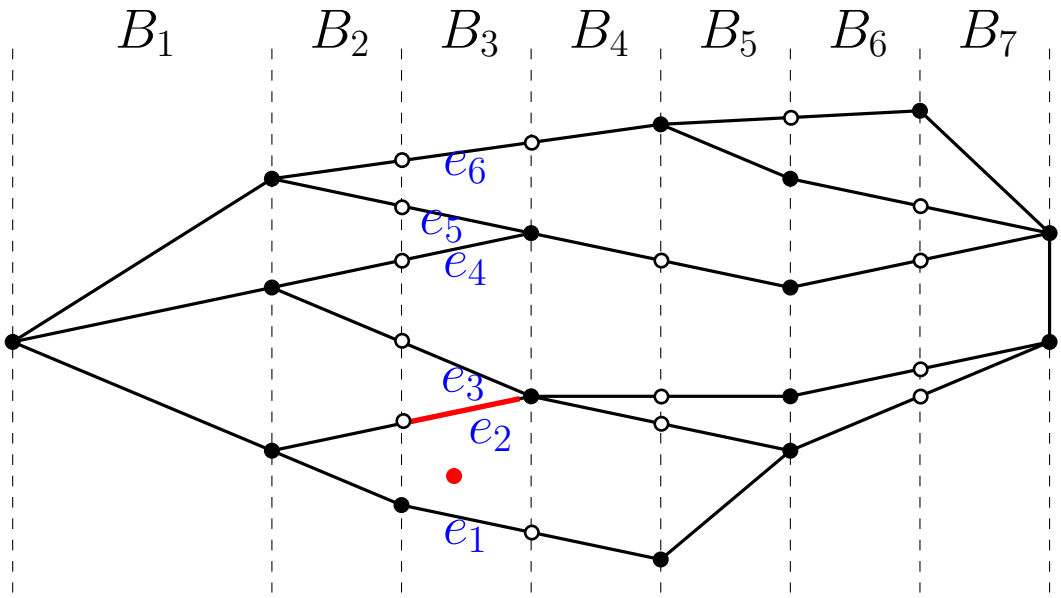
Methode du *Slab*

quadratic space in worst case

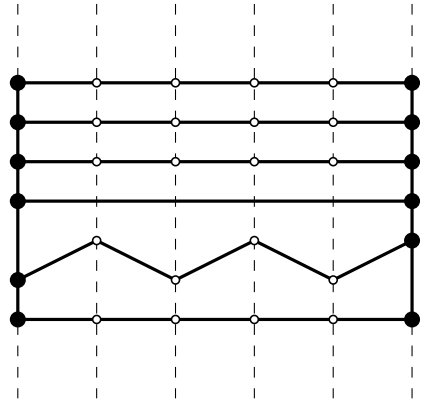
$O(n)$ aretes et faces dans chaque bande

$\mathbf{O(n)}$ bandes verticales

$\mathbf{O(n)}$ sommets, faces et aretes



$O(\frac{n}{4})$ faces



$E_{B_3} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

la subdivision S' est de taille $O(n^2)$

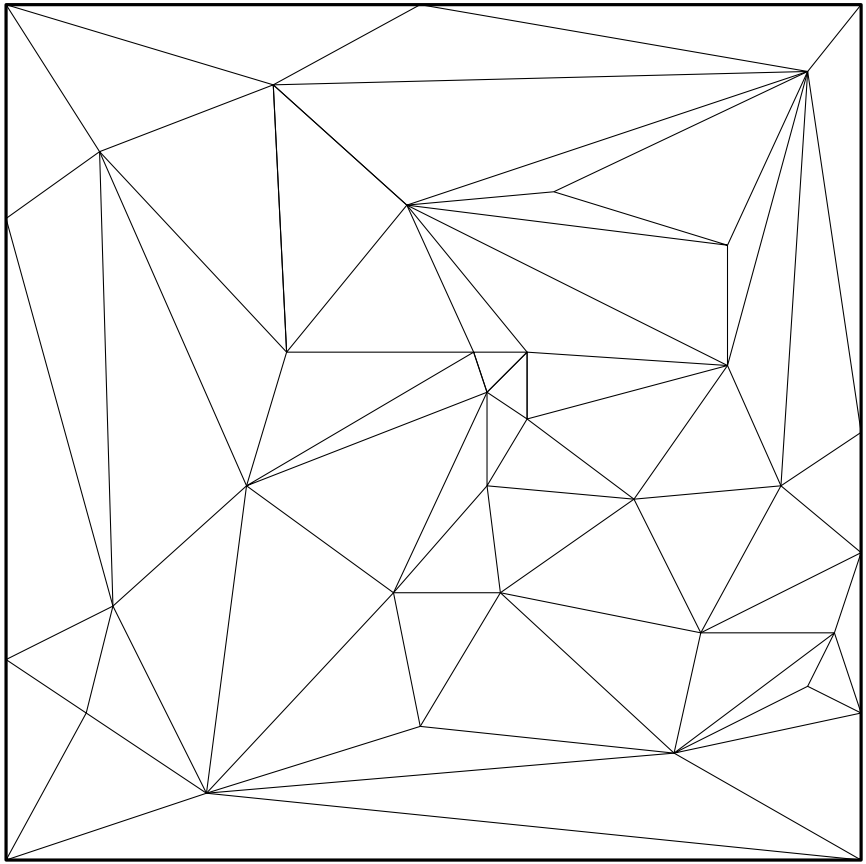
$\frac{n}{4}$ bandes verticales

Quad trees and planar point location

Quad trees and planar point location

(some images are by S. Oudot)

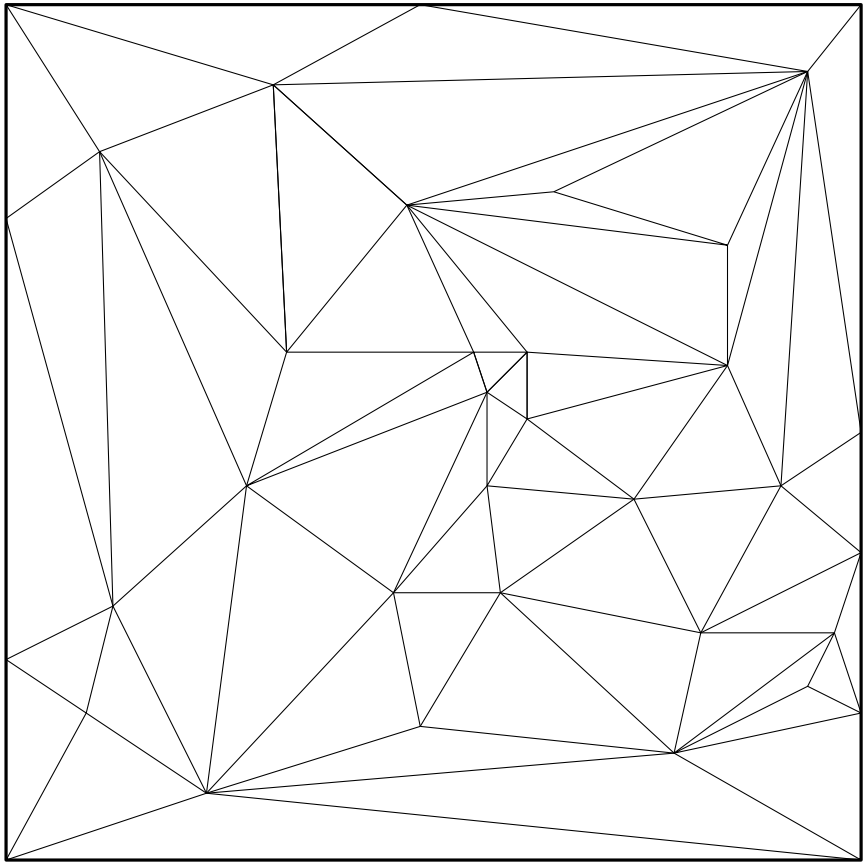
(following Har-Peled's notes)



Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)

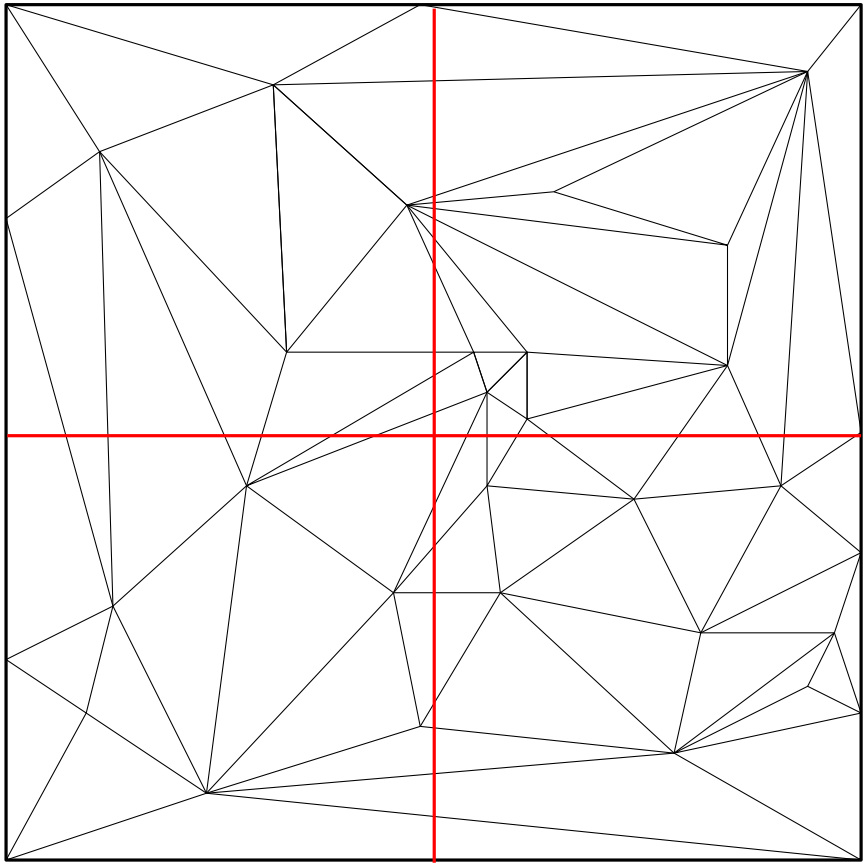


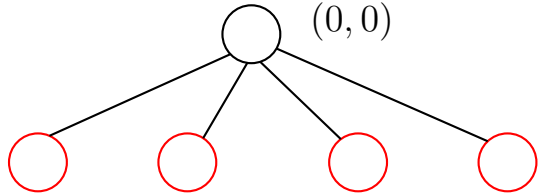
$(0, 0)$

Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)

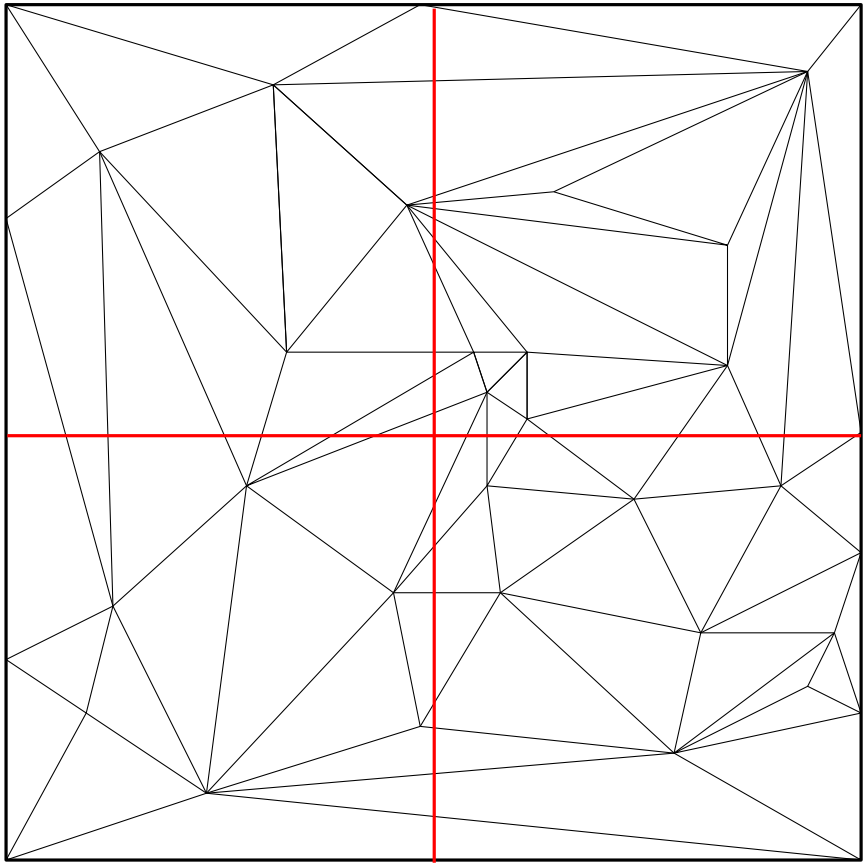


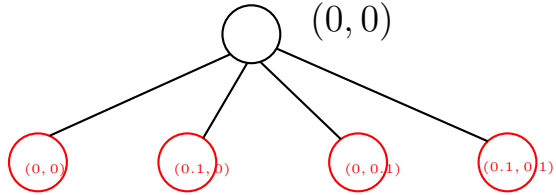


Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)

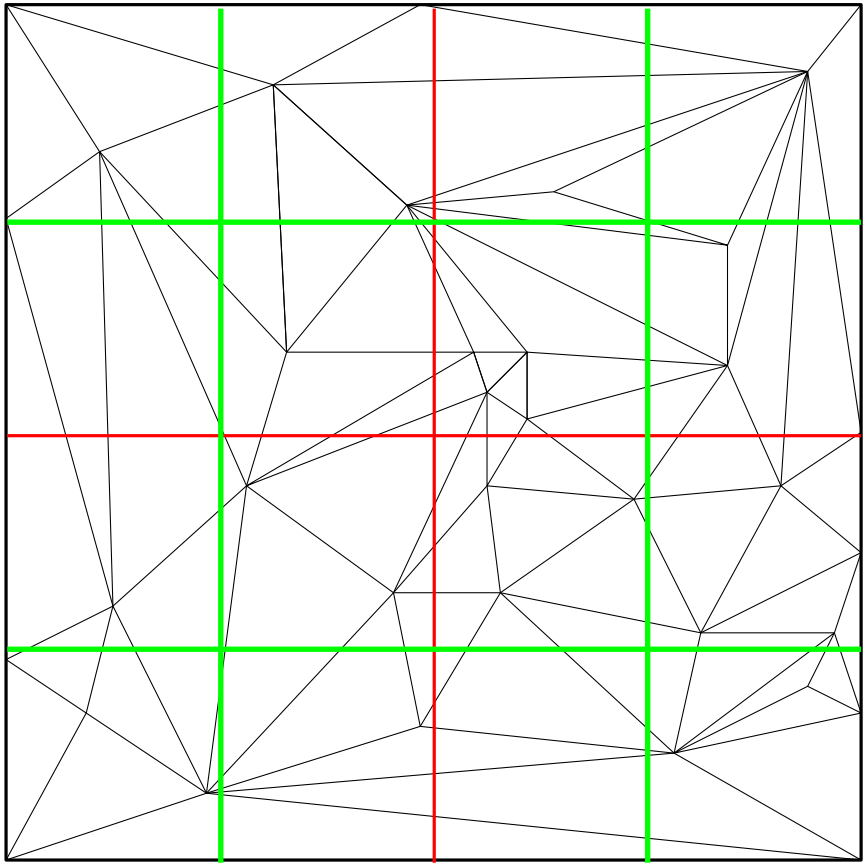


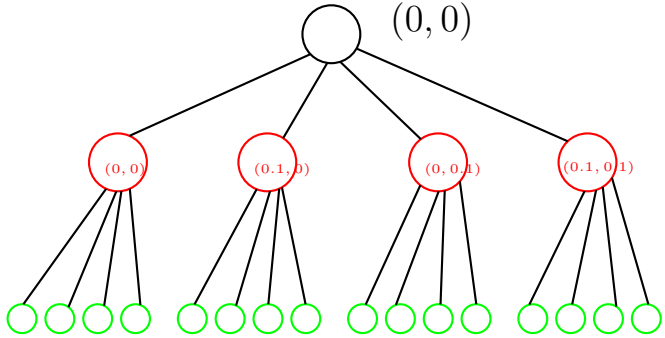


Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)

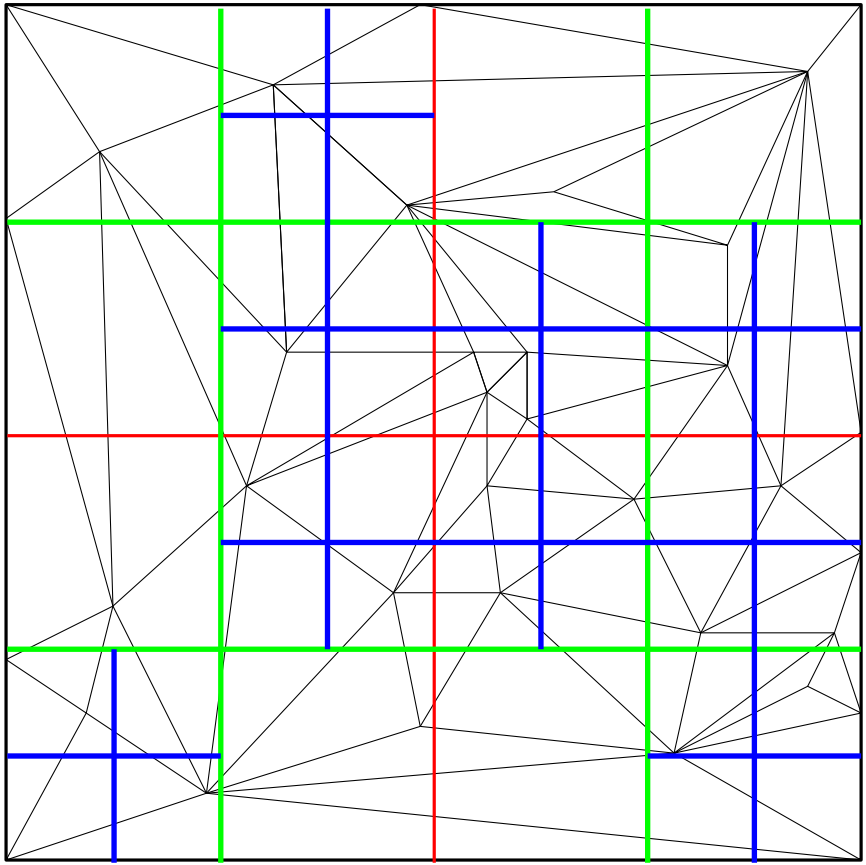


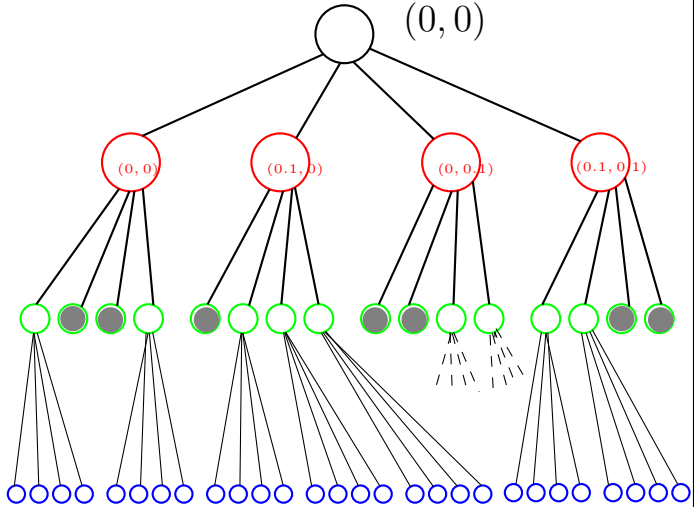


Quad trees and planar point location

(some images are by S. Oudot)

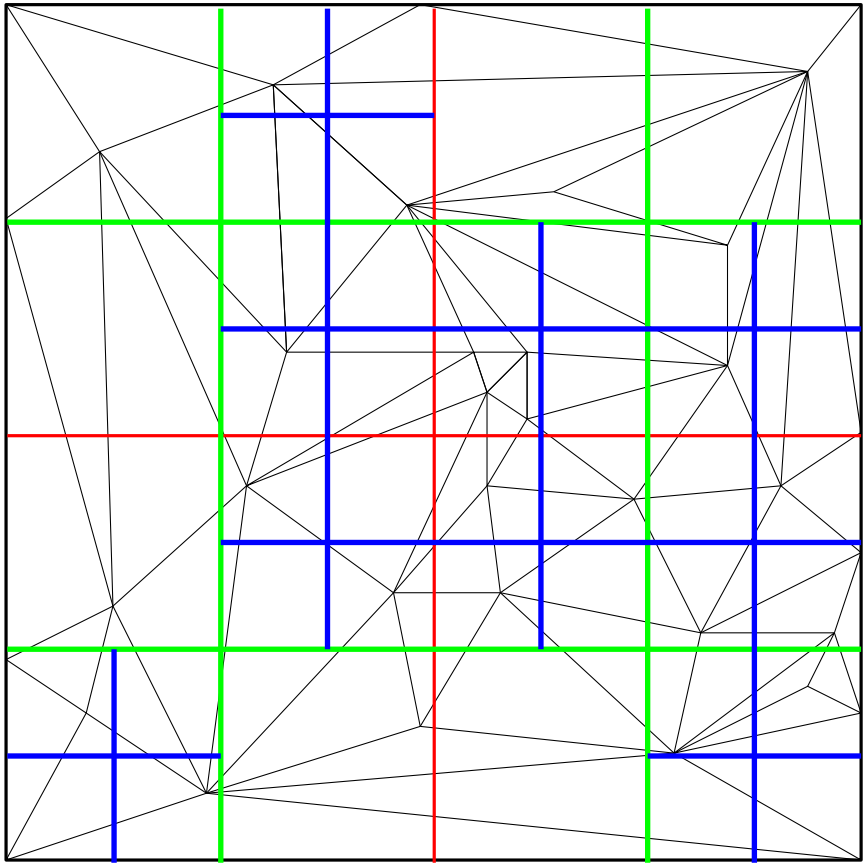
(following Har-Peled's notes)

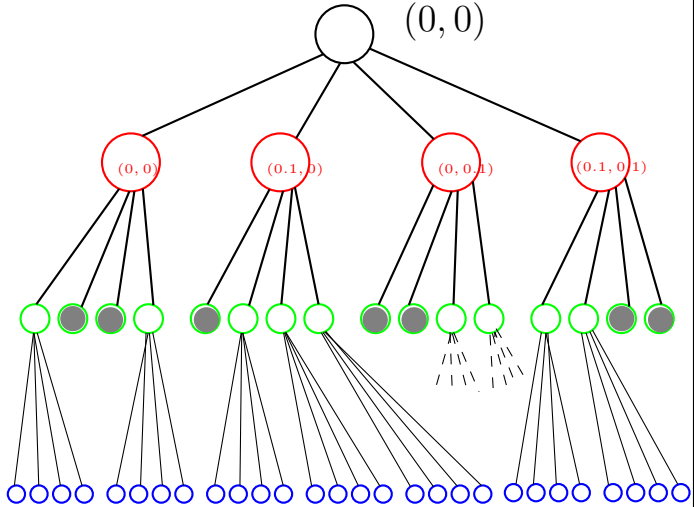




Quad trees and planar point location

(some images are by S. Oudot)
(following Har-Peled's notes)



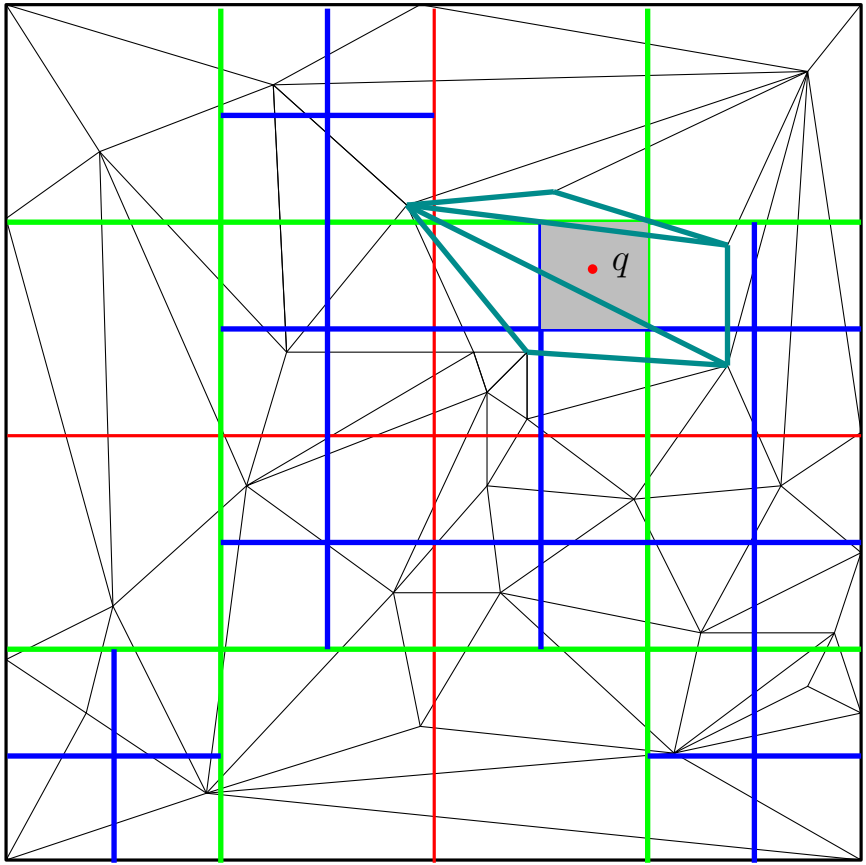


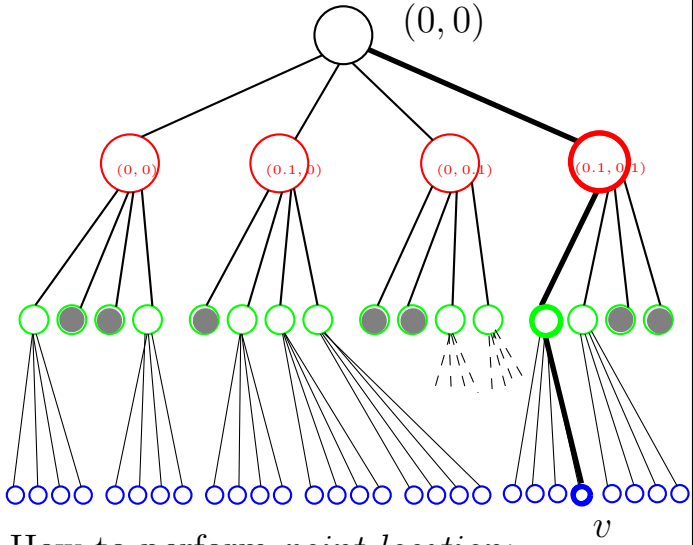
Idea: *construct a quaternary tree where leaves intersect at most C triangles*

Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)





How to perform *point location*:

Search in the quadtree node v (a leaf)

search among triangles intesercted by $\square_v$

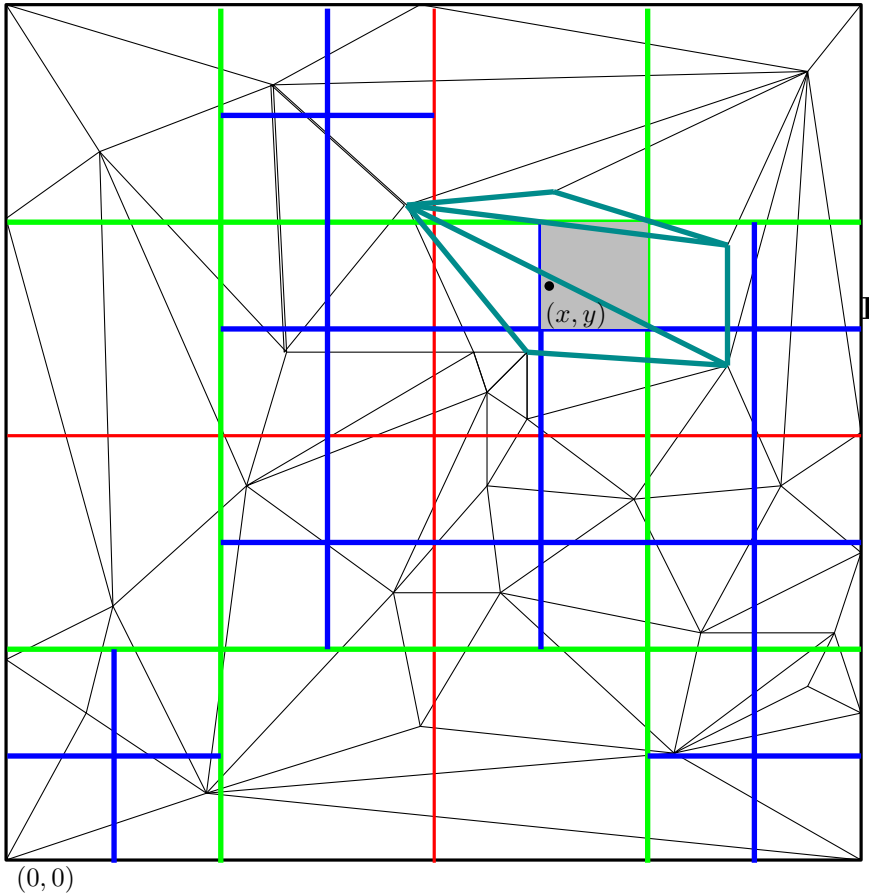
requires $O(C + h)$ time

h is the *depth of the tree*

Quad trees and planar point location

(some images are by S. Oudot)

(following Har-Peled's notes)



Use *geometric hashing* to speed-up the search

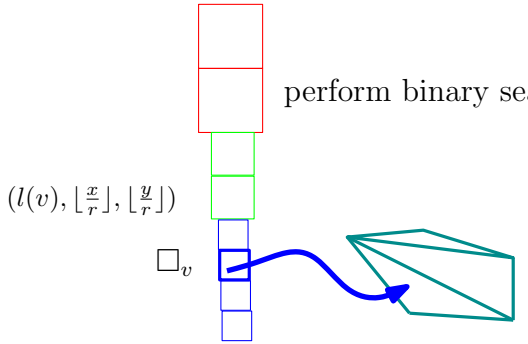
Idea: store every node in an *hash table*

$\square_v$ is a *canonical square* of side 2^{-i}

$l(v) = -i$ is the *level* of node v

Remark: every node can be uniquely identified by

$$id(v) = (l(v), \lfloor \tfrac{x}{r} \rfloor, \lfloor \tfrac{y}{r} \rfloor)$$



requires $O(\log h)$ time

$O(\log \log n)$ time , when $h = O(\log n)$

Hierarchical triangulations

planar point location in optimal time and space

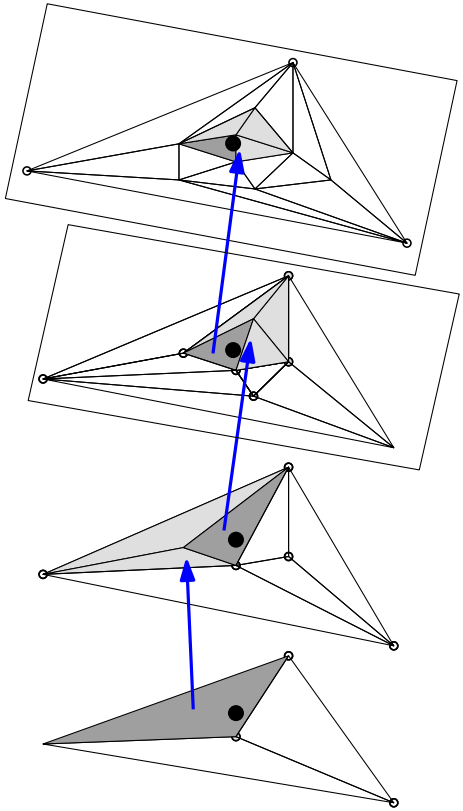
Hierarchical triangulations

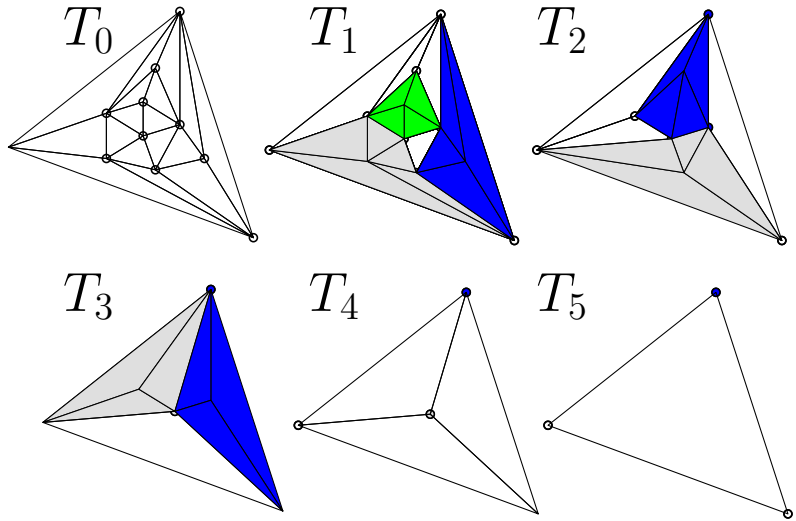
Hierarchical triangulations

Thm (Kirkpatrick, '83

Etant donnée une triangulation planaire de taille n, la localisation d'un point se fait:

en temps $O(\log n)$, ***espace*** $O(n)$





Hierarchical triangulations

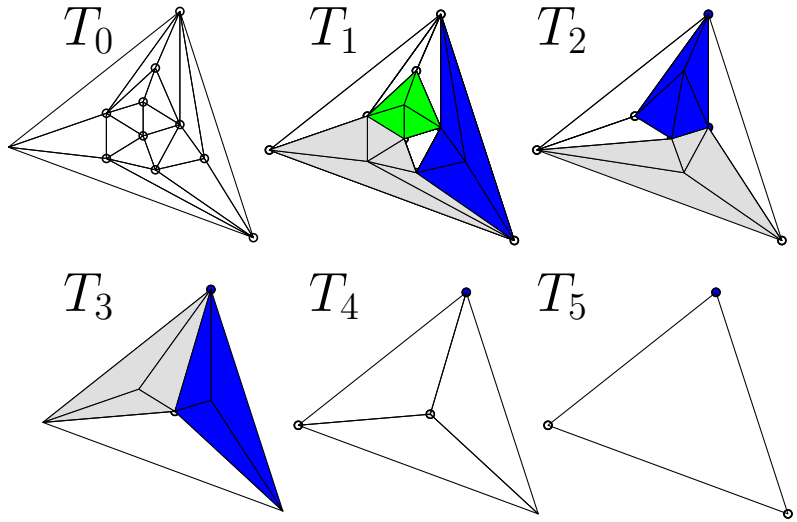
Thm (Kirkpatrick, '83

Etant donnée une triangulation planaire de taille n, la localisation d'un point se fait:

en temps $O(\log n)$, ***espace*** $O(n)$

Idea: *construct a sequence of triangulations* $T_0, T_1 \ldots, T_k$

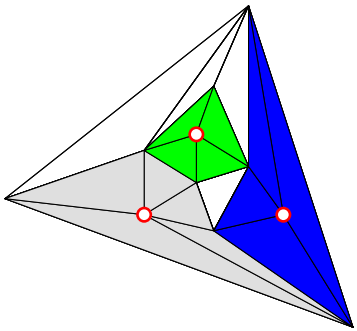
- $T_0 = \mathcal{T}$
- $k = O(\log n)$
- $|T_k| = 1$
- $T_{i+1} = T_i \setminus E^i_{ind}$



Hierarchical triangulations

Lemme 1(ensemble independants)

Il existe un ensemble independant de sommets de taille au moins $\frac{n}{18}$*, chaque sommet ayant degré au plus* 8.



$e = 3n - 6$ $\qquad \sum_v deg(v) = (2e) = 6n - 12$

(from Euler formula)

Claim $\boxed{|\{sommets\, degre\ \leq 8\}| \geq \frac{n}{2}}$

$\sum_{v\in V} degre(v) = \sum_{v\in V_9} 9 + \sum_{v\in V\setminus v_9} degre(v) > 9\frac{n}{2} + 3\frac{n}{2} = 6n > 6n - 12$

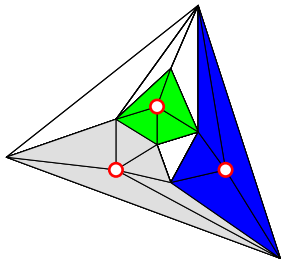
(contradiction)

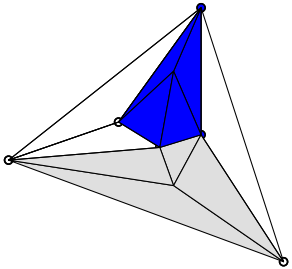
E_{ind} := independent set $\qquad \boxed{|E_{ind}| \geq \frac{n}{2}/9 = \frac{n}{18}}$

Hierarchical triangulations

Lemme 1(complexité de la structure)

Le nombre de niveaux est $k = O(\log n)$*, et le cout de la structure est* $O(n)$*.*





$$T_{i+1} = T_i \setminus E^i_{ind} \qquad |E^i| = \Omega(n)$$

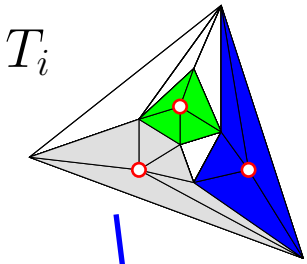
$$|T_i| \leq C|T_{i+1}| \longrightarrow |T_0| \leq C^k$$

$$k = O(\log n)$$

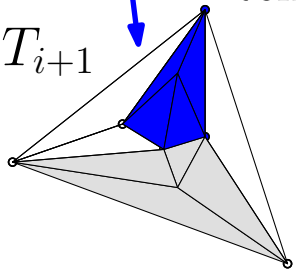
Hierarchical triangulations

Lemme 1(complexité de la structure)

Le nombre de niveaux est $k = O(\log n)$*, et le cout de la structure est* $O(n)$*.*



temps $O(|T_i|)$



$$T_{i+1} = T_i \setminus E^i_{ind} \qquad |E^i| = \Omega(n)$$

$$|T_i| \leq C|T_{i+1}| \longrightarrow |T_0| \leq C^k$$

$$k = O(\log n)$$

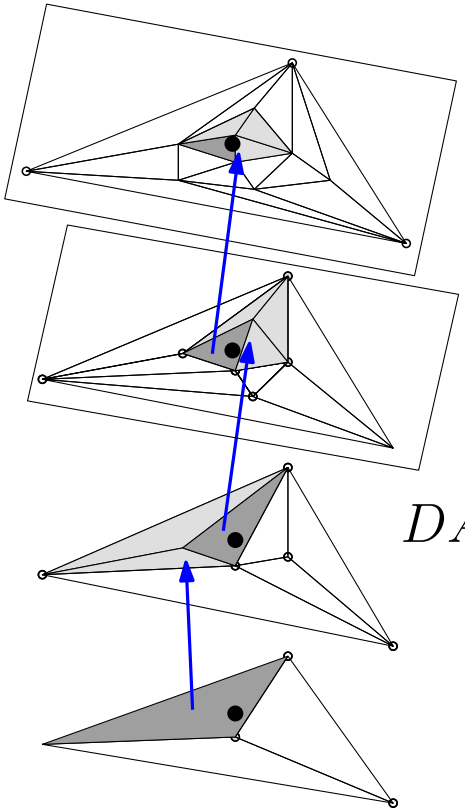
cout total:

$$\sum_{i \leq k} O(|T_i|) \leq \sum_{i=O(\log n)} Cn(\frac{17}{18})^i = O(n)$$

Hierarchical triangulations

Lemme 3(complexité de la localisation)

La localisation se fait en temps $O(\log n)$.



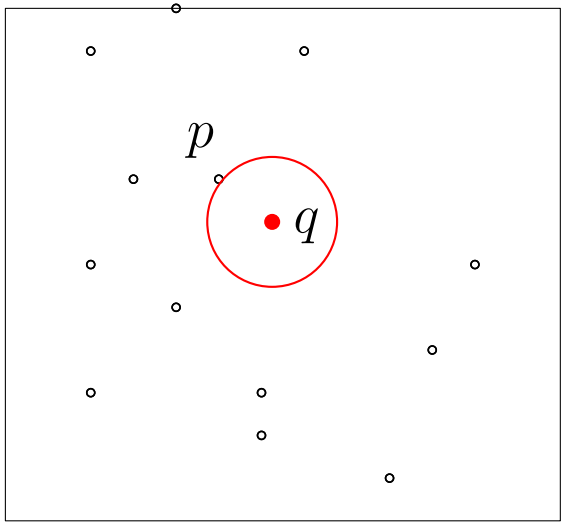
DAG

$k = O(\log n)$

les sommets dans E^i_{ind} ont degré borné

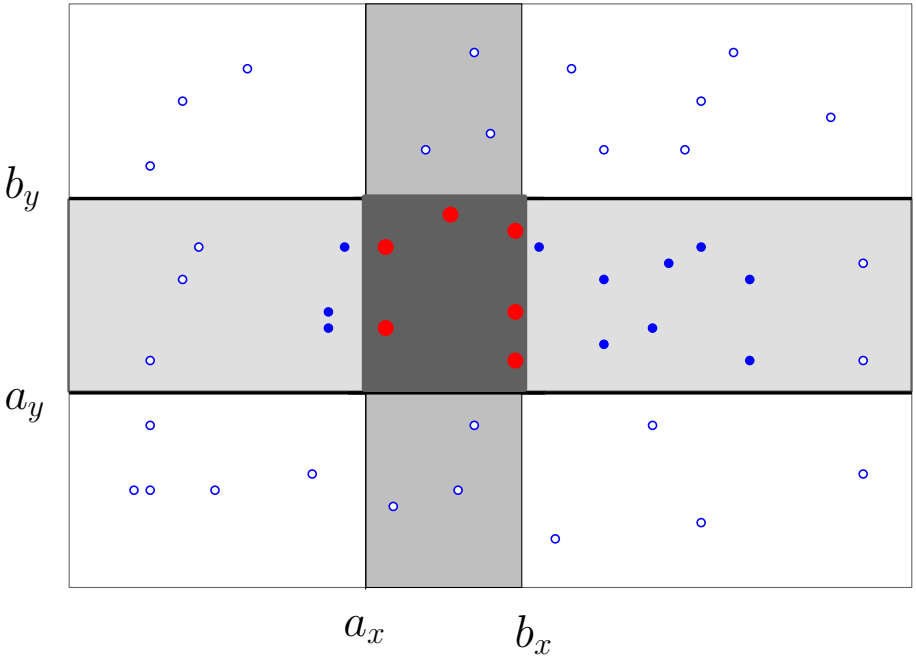
Part II
Proximity queries on points

Nearest Neighbor



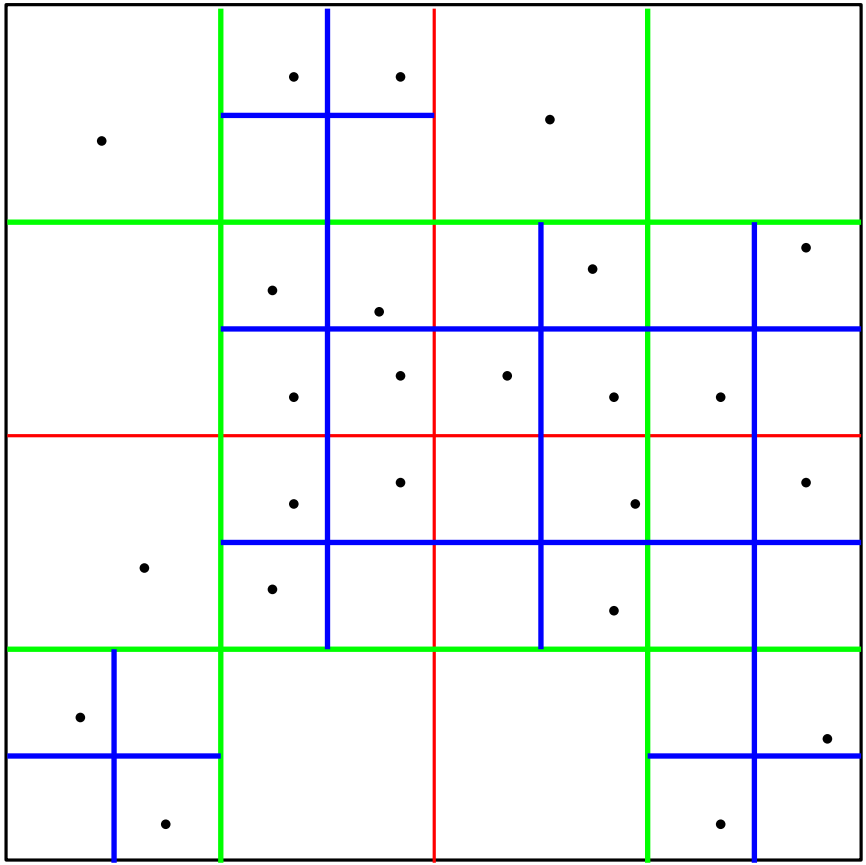
Range Queries

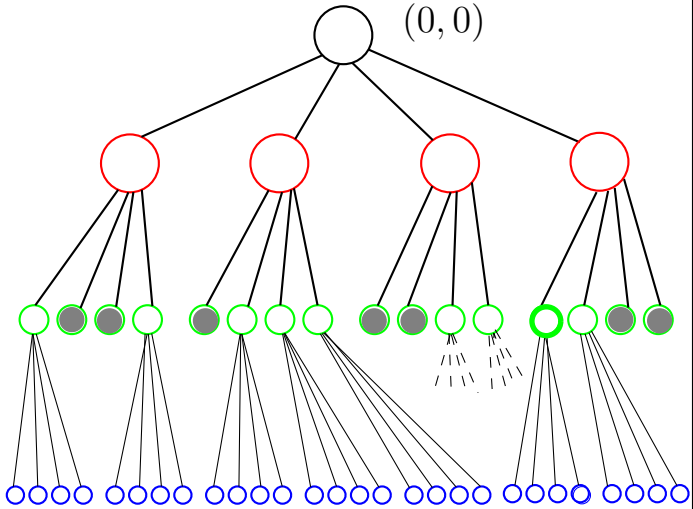
$$R = [a_x, b_x] \times [a_y, b_y]$$



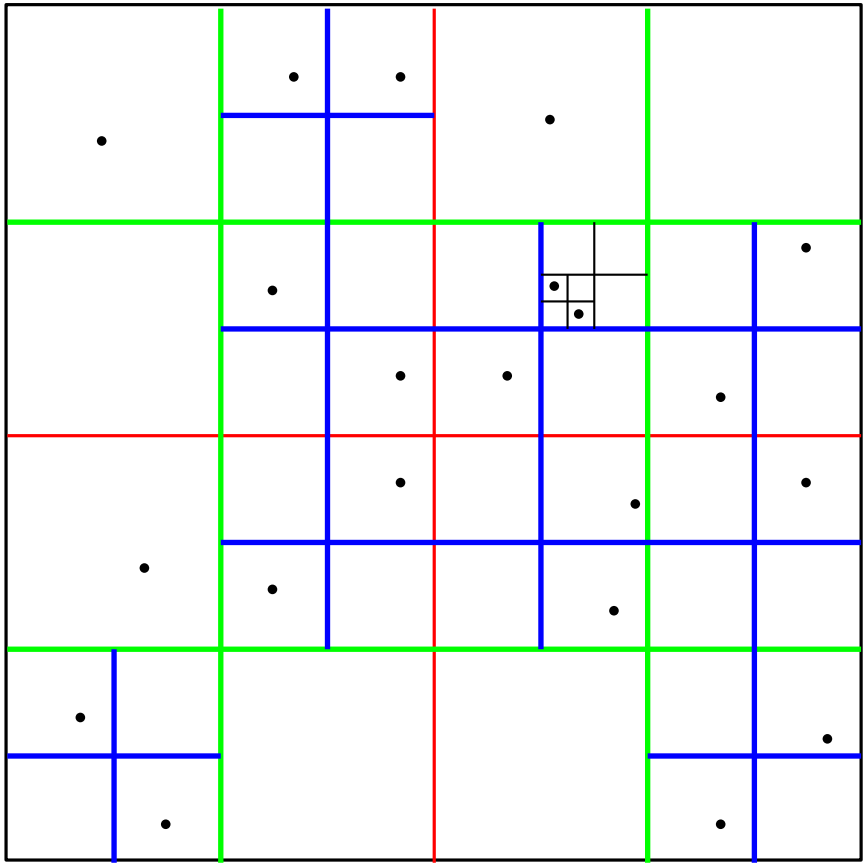
Quad-trees and point sets

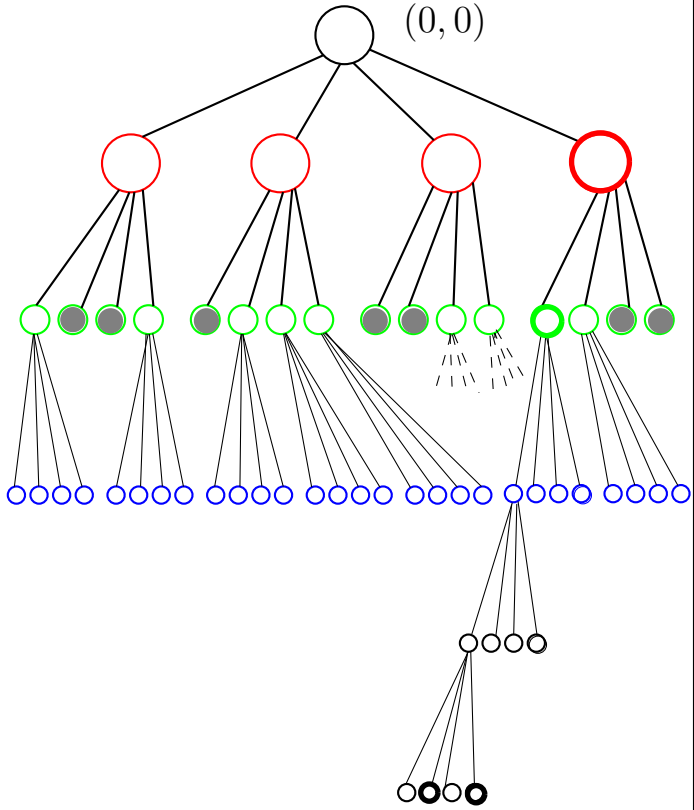
Quad-trees and point sets



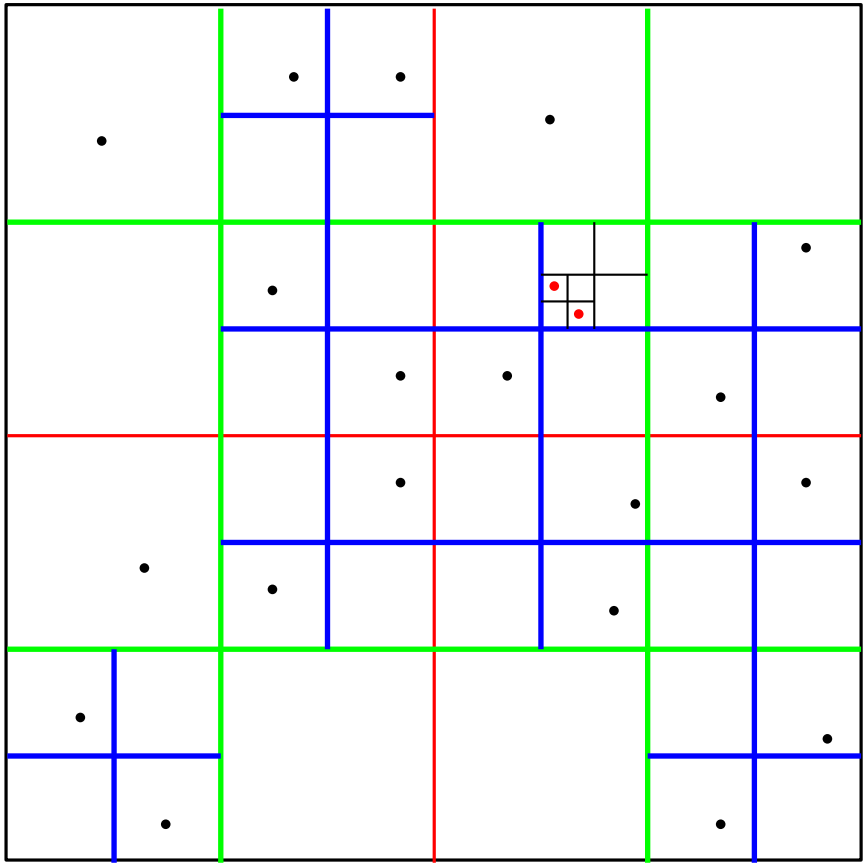


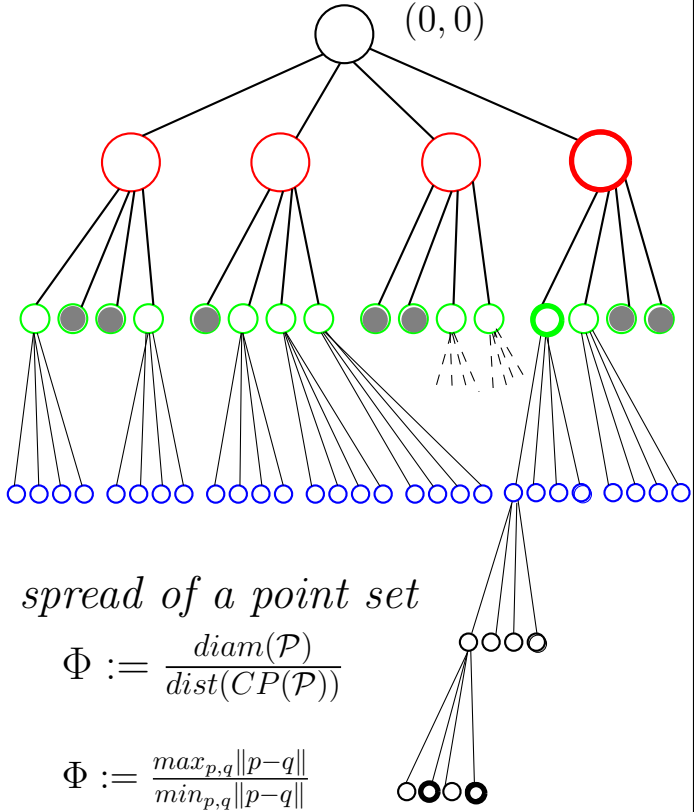
Quad-trees and point sets





Quad-trees and point sets





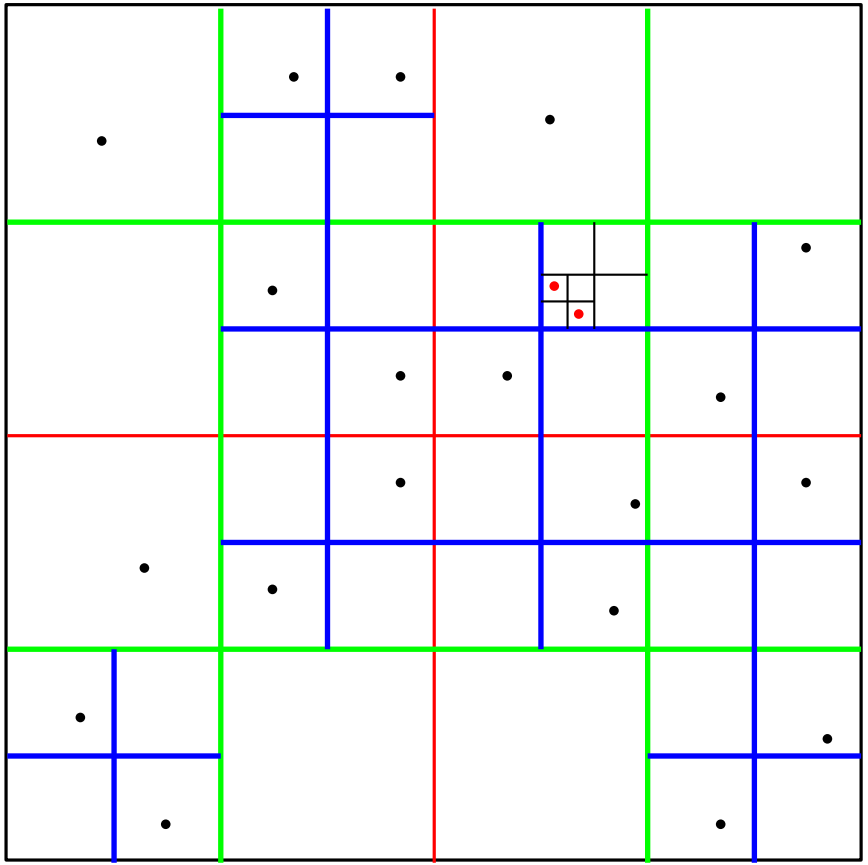
spread of a point set

$$\Phi := \frac{diam(\mathcal{P})}{dist(CP(\mathcal{P}))}$$

$$\Phi := \frac{max_{p,q}\|p-q\|}{min_{p,q}\|p-q\|}$$

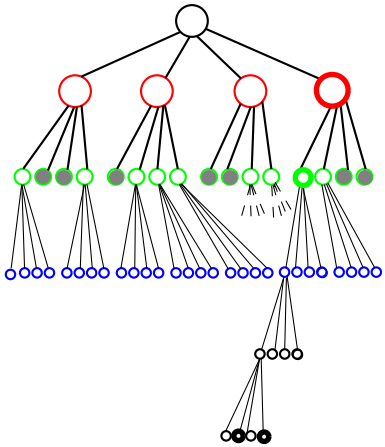
$CP(\mathcal{P})$:= closest pair of points in $\mathcal{P}$

Quad-trees and point sets



Lemme::Soit $\mathcal{P}$ tel que $diam(\mathcal{P}) \geq \frac{1}{2}$. Alors, la profondeur d'un quad-tree (chaque feuille avec un seul point) est au plus $O(\log \Phi)$.

$h = O(\log \Phi)$

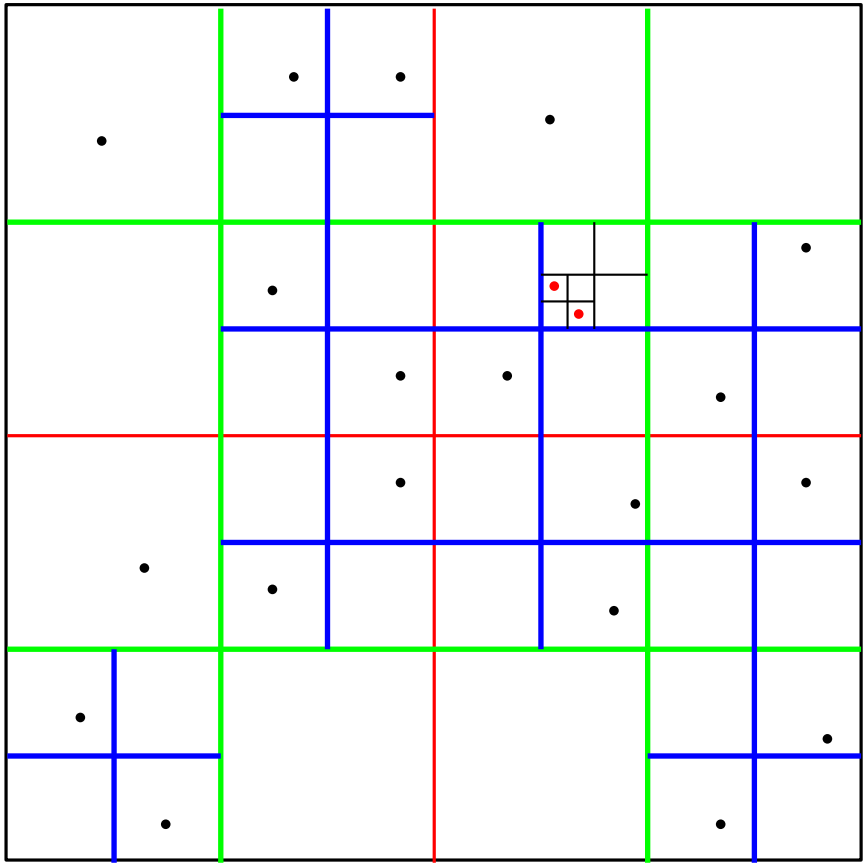


spread of a point set

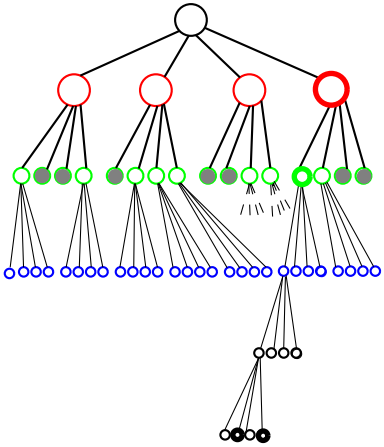
$$\Phi := \frac{diam(\mathcal{P})}{dist(CP(\mathcal{P}))}$$

$$\Phi := \frac{max_{p,q}\|p-q\|}{min_{p,q}\|p-q\|}$$

Quad-trees and point sets



Lemme::Soit $\mathcal{P}$ tel que $diam(\mathcal{P}) \geq \frac{1}{2}$. Alors, la profondeur d'un quad-tree (chaque feuille avec un seul point) est au plus $O(\log \Phi)$.



$h = O(\log \Phi)$

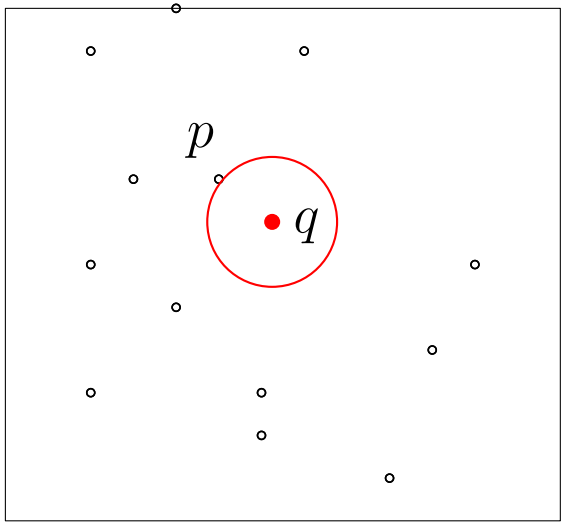
$size(\square_v) = \frac{s}{2^i}$ $s :=$ side length of the initial square

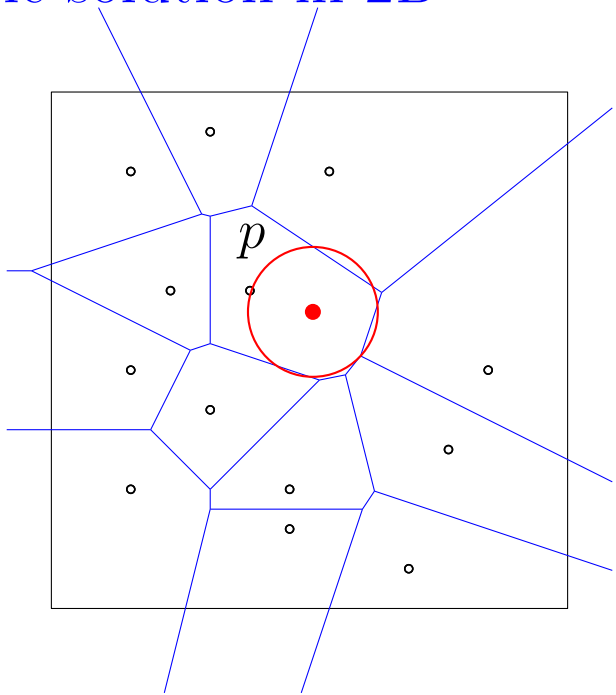
$\max dist(p, q) \leq s\frac{\sqrt{2}}{2^i}$

max distance between points in $\square_v$

$s\frac{\sqrt{2}}{2^i} \geq d_{min}$ $\quad i \leq \log \frac{s\sqrt{2}}{d_{min}} = \log \Phi + \frac{1}{2}$

Nearest Neighbor: simple solution in 2D





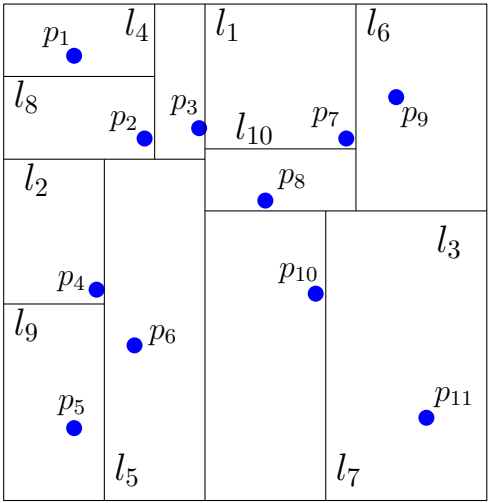
Solution:

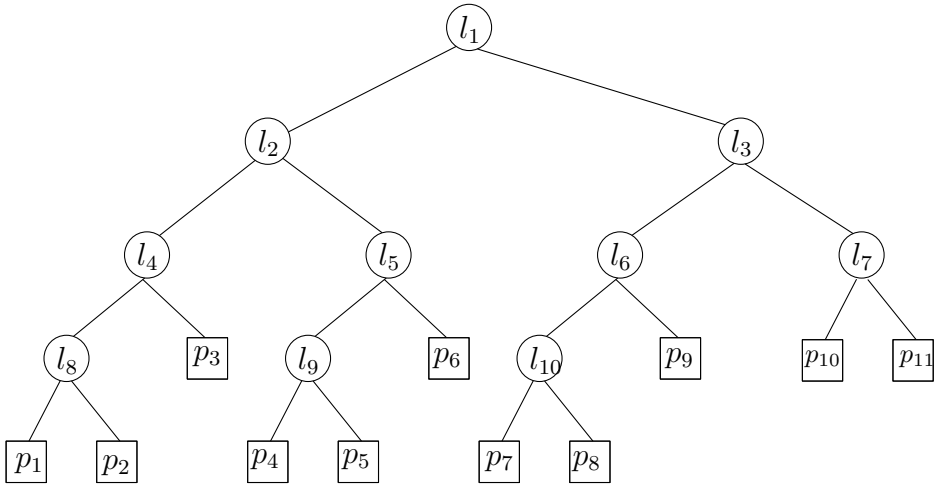
temps de requete: $O(\log n)$

pretraitement: $O(n \log n)$

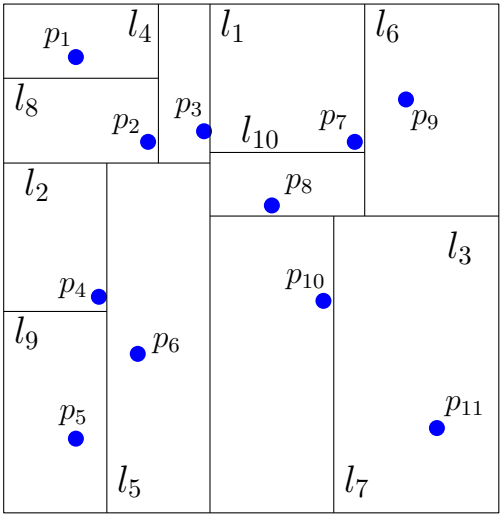
Kd-Trees

Kd-Trees et Nearest Neighbor Problem

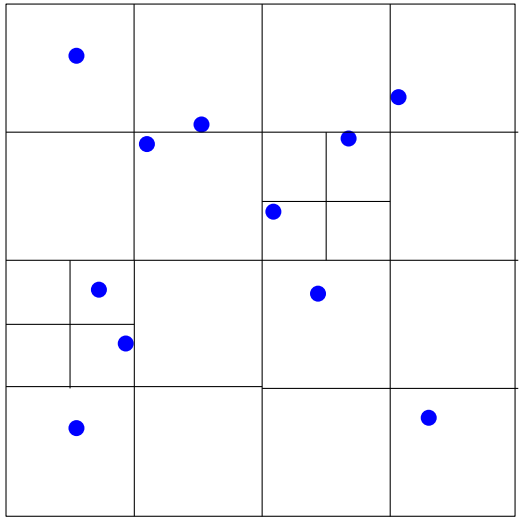




Binary space partitions

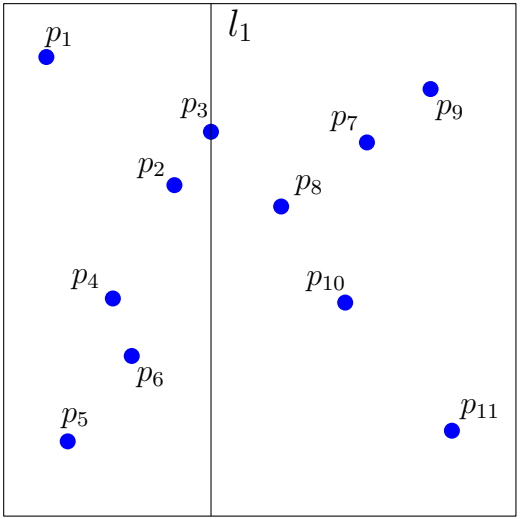


Kd-Tree



QuadTree

Calcul de la mediane

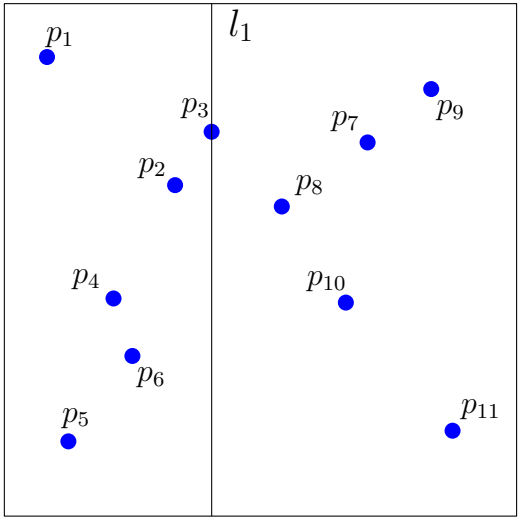


2	3	5	6	8	10	13	16	17	20	21
0	1				$\frac{n}{2}$					$n-1$

Si les nombres sont triés

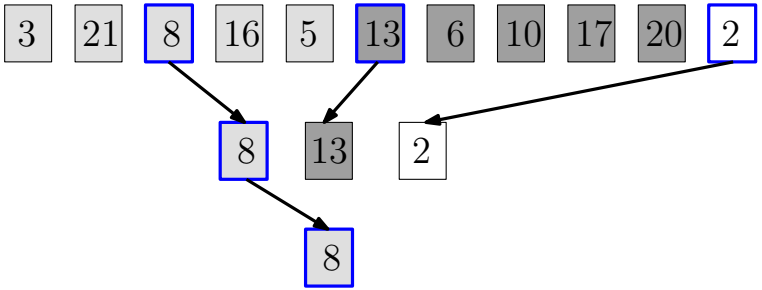
Calcul de la mediane en temps linéaire

(si les nombres ne sont pas triés)



fonction $median(Tableau\ A)$

- Partitionner le tableau en $\lfloor \frac{n}{5} \rfloor$ groupes de taille 5
  plus eventuellement un groupe de taille $n\, mod(5)$
- Calculer la mediane de chacun des $\lceil \frac{n}{5} \rceil$ groupes
  avec une methode quelconque (exemple: tri par insertion)
- Appeler recursivement $median(L_0)$
  pour calculer la mediane des $\lceil \frac{n}{5} \rceil$ medianes

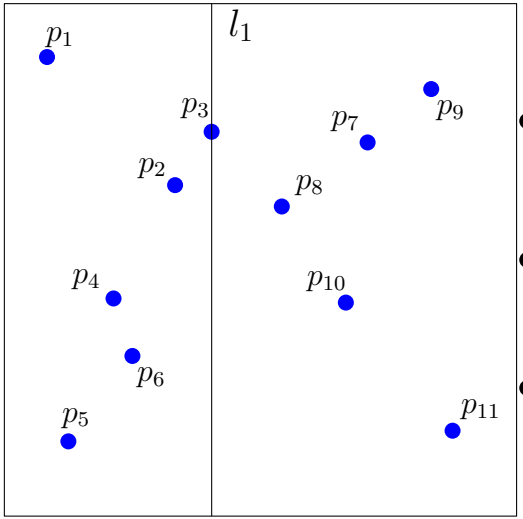


$A_0 = A$

L_0: liste des medianes

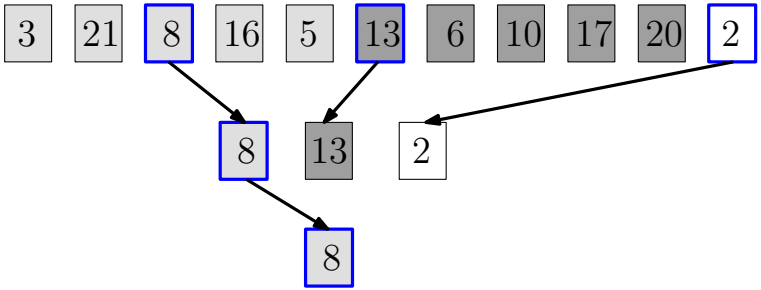
$A_1 = median(L_0)$

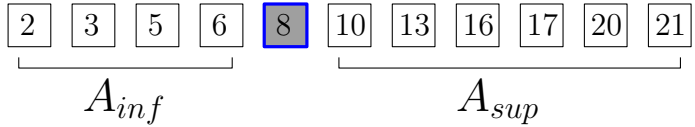
Calcul de la mediane en temps linéaire



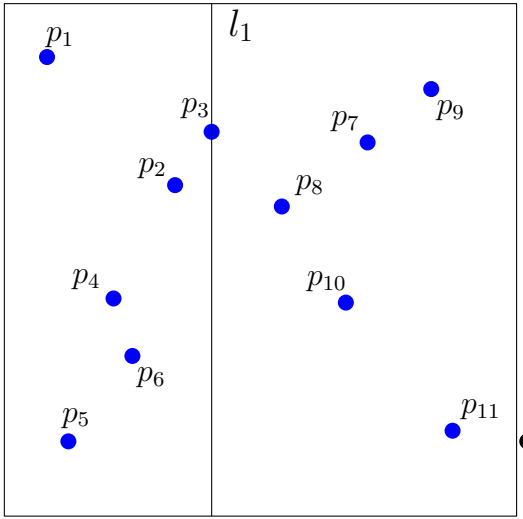
fonction $median(Tableau\ A)$

- Partitionner le tableau en $\lfloor \frac{n}{5} \rfloor$ groupes de taille 5
  plus eventuellement un groupe de taille $n\ mod(5)$
- Calculer la mediane de chacun des $\lceil \frac{n}{5} \rceil$ groupes
  avec une methode quelconque (exemple: tri par insertion)
- Appeler recursivement $median(L_0)$
  pour calculer la mediane des $\lceil \frac{n}{5} \rceil$ medianes



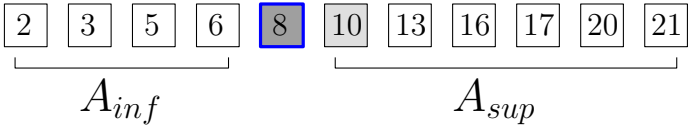


Calcul de la mediane en temps linéaire

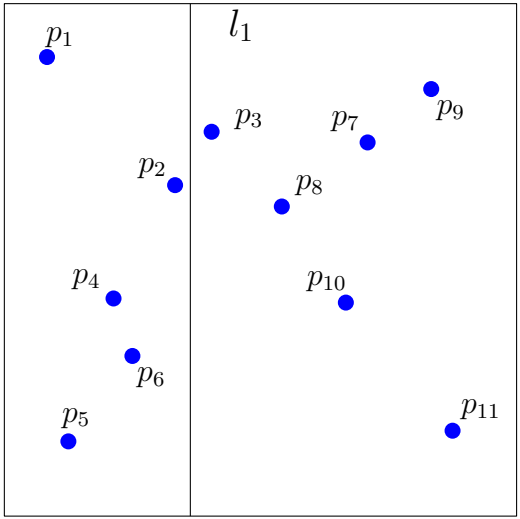


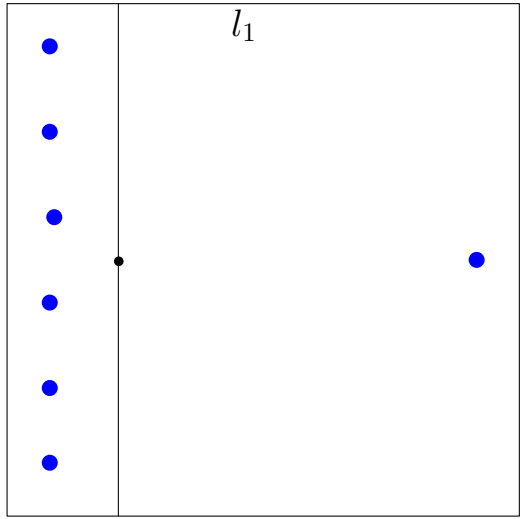
fonction $median(Tableau\ A)$

- Partitionner le tableau en $\lfloor \frac{n}{5} \rfloor$ groupes de taille 5
  plus eventuellement un groupe de taille $n\, mod(5)$
- Calculer la mediane de chacun des $\lceil \frac{n}{5} \rceil$ groupes
  avec une methode quelconque (exemple: tri par insertion)
- Appeler recursivement $median(L_0)$
  pour calculer la mediane des $\lceil \frac{n}{5} \rceil$ medianes
- partitionner A autour de $median(L_0)$
  en deux sous-tableaux, de taille k et $n-k$
- appeler recursivement $median()$ sur A_{inf} (ou A_{sup})
  selon que $\frac{n}{2} \leq k$ (ou $\frac{n}{2} > k$)

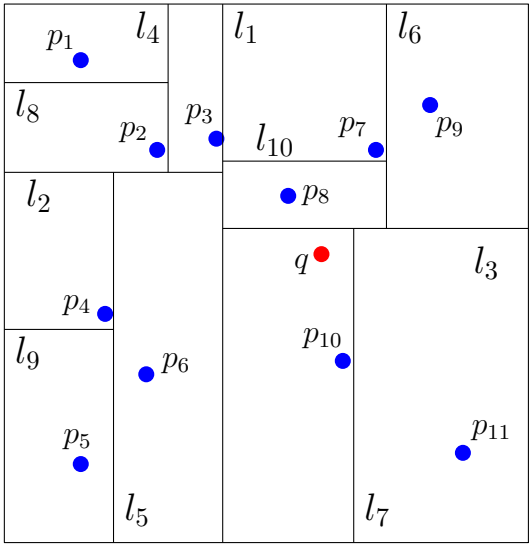


Calcul du plan séparateur avec barycentre

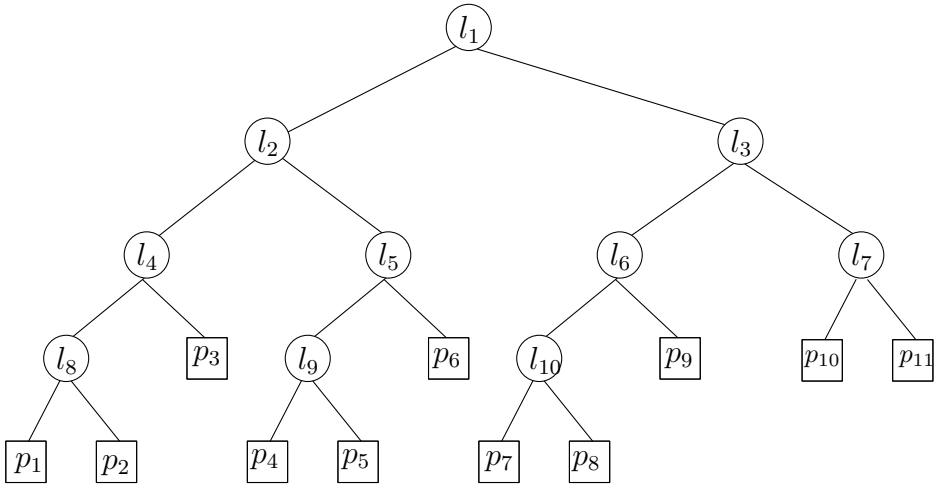




Nearest Neighbor Search

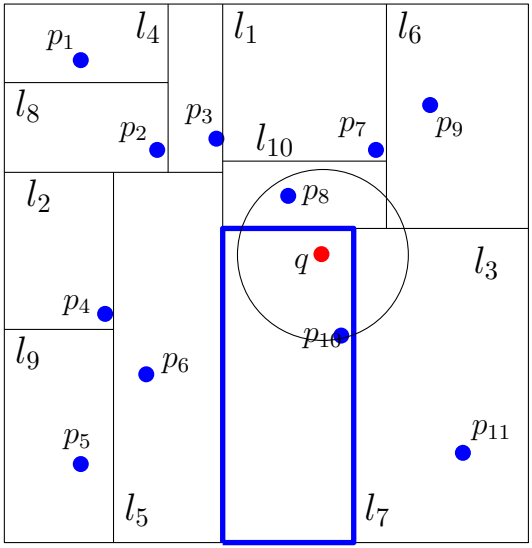


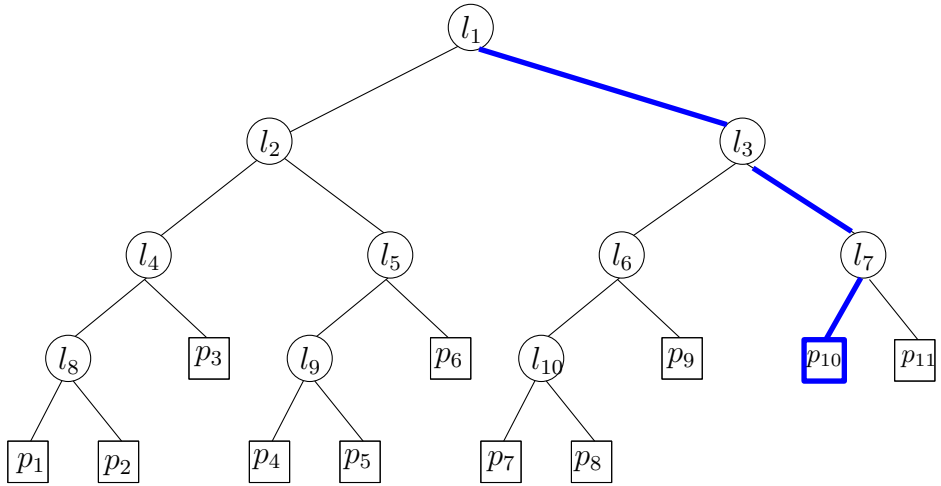
On commence une visite recursive de la racine



Nearest Neighbor Search

$NN(q) = p_{10}$

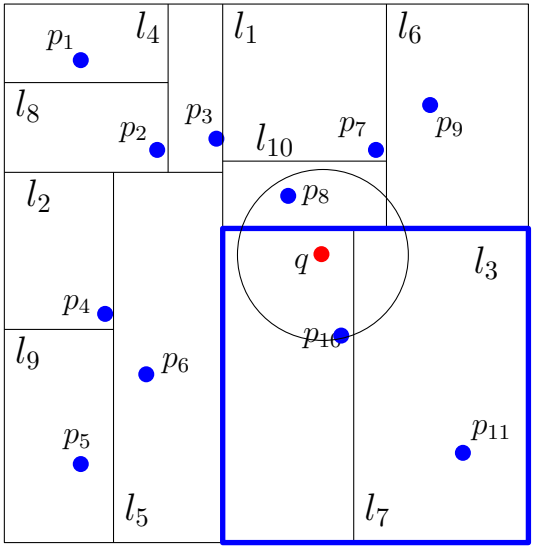


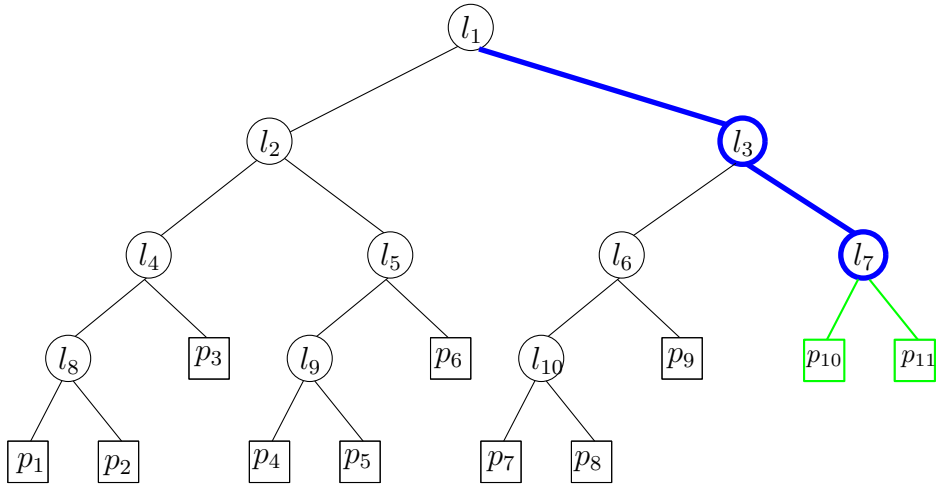


We perform a recursive visit in order to find the leaf containing q

Nearest Neighbor Search

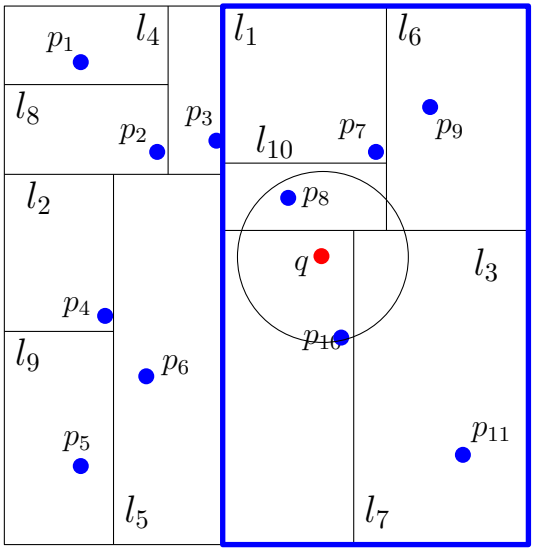
$NN(q) = p_{10}$



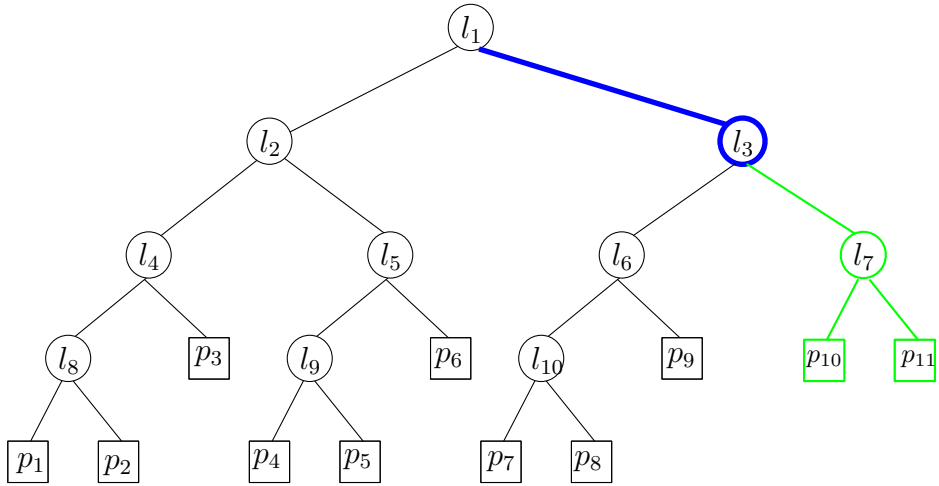


After the recursive call, we search in the neighboring cells for closer neighbors

Nearest Neighbor Search

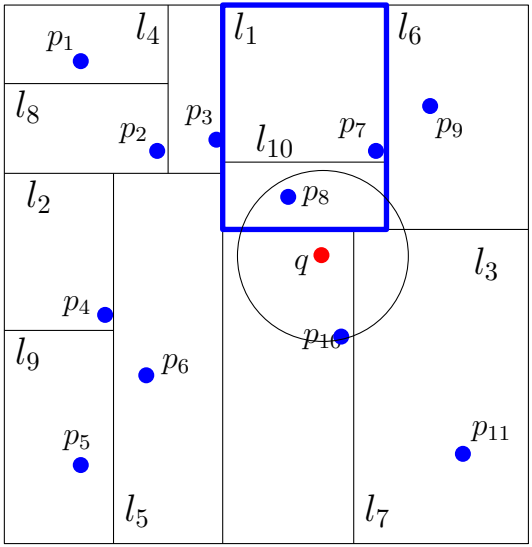


$NN(q) = p_{10}$

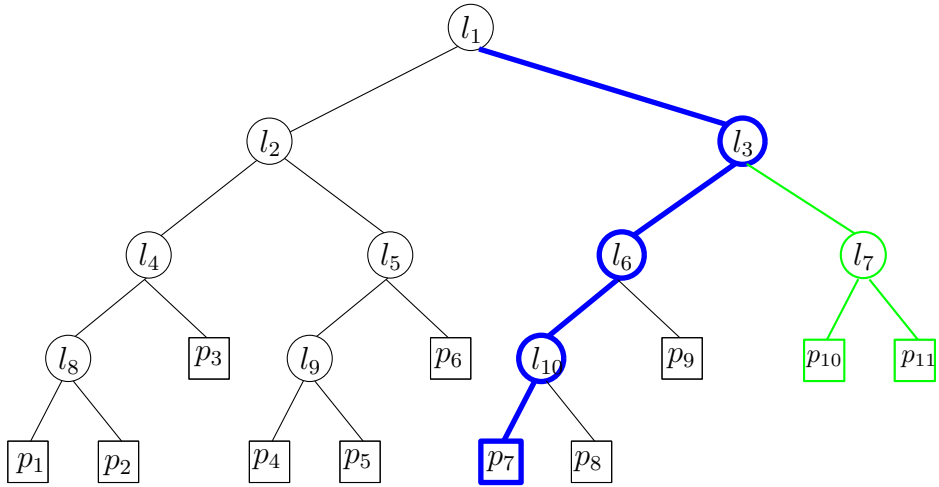


After the recursive call, we search in the neighboring cells for closer neighbors

Nearest Neighbor Search

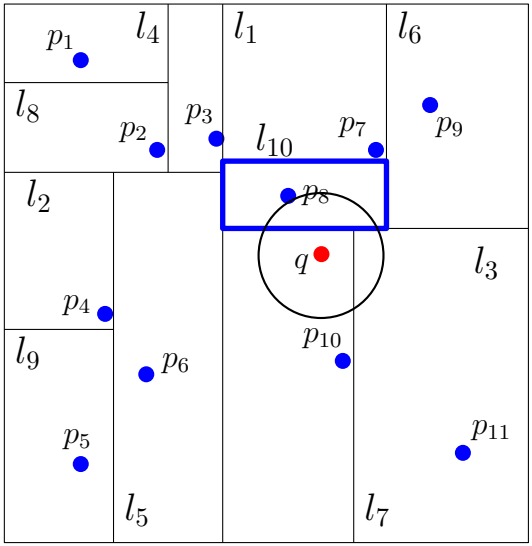


$NN(q) = p_{10}$

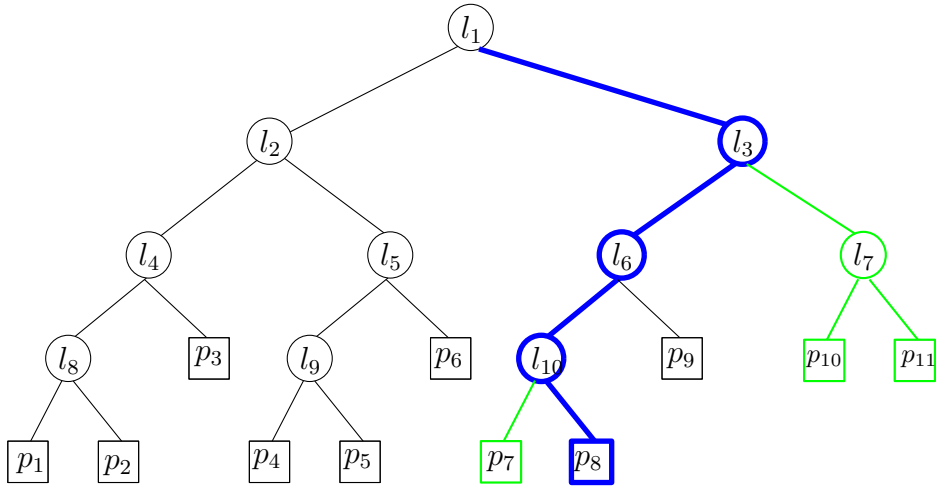


After the recursive call, we search in the neighboring cells for closer neighbors

Nearest Neighbor Search



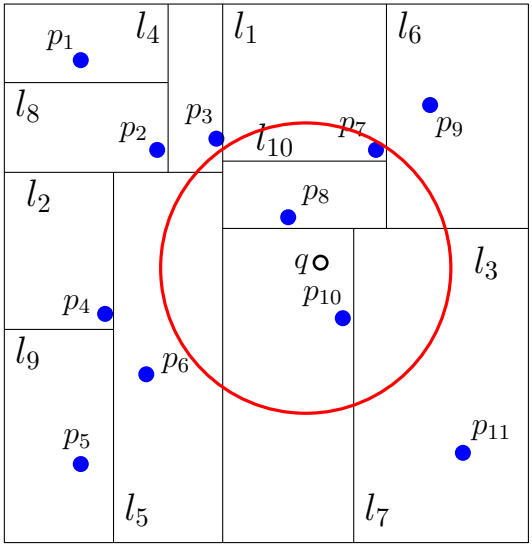
$NN(q) = p_8$



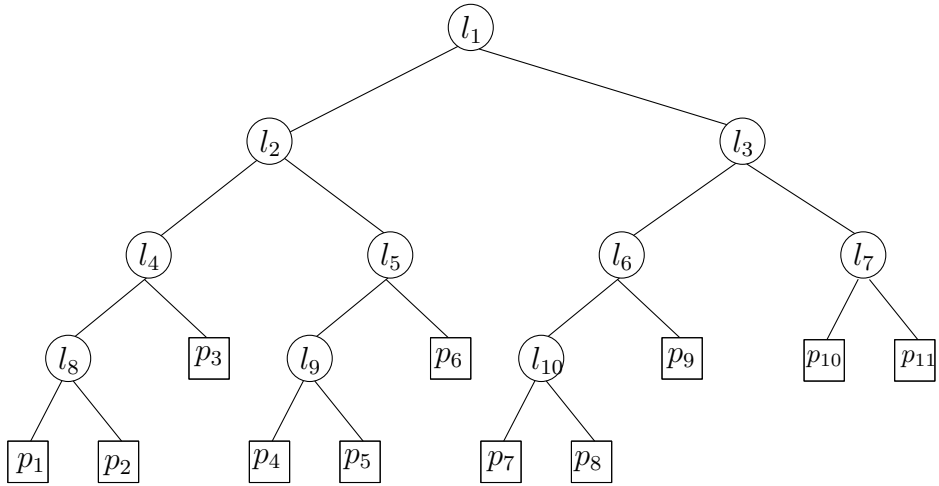
After the recursive call, we search in the neighboring cells for closer neighbors

Kd-Trees and range search

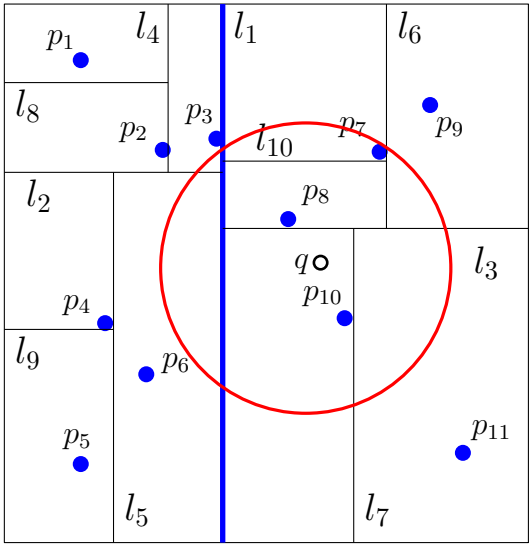
Kd-Trees and range search

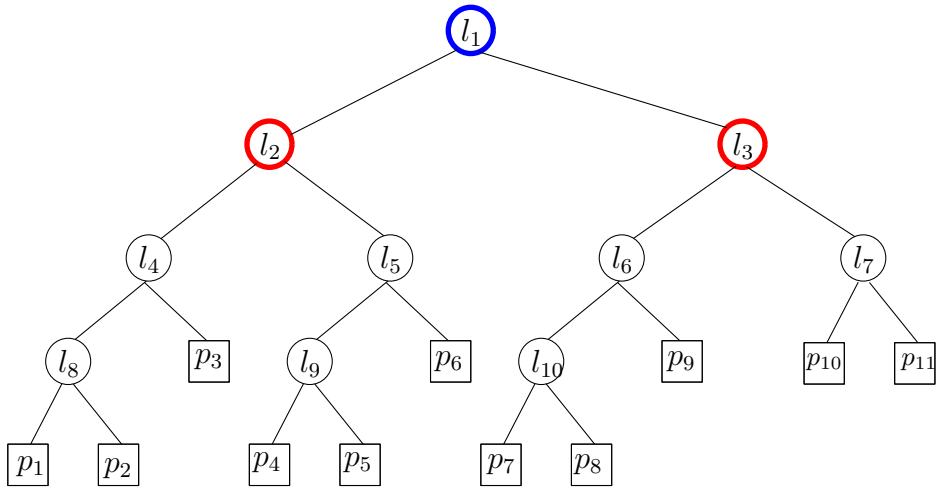


On commence une visite recursive de la racine

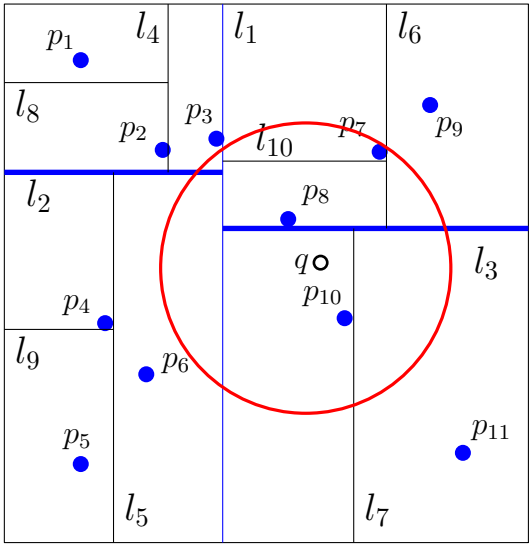


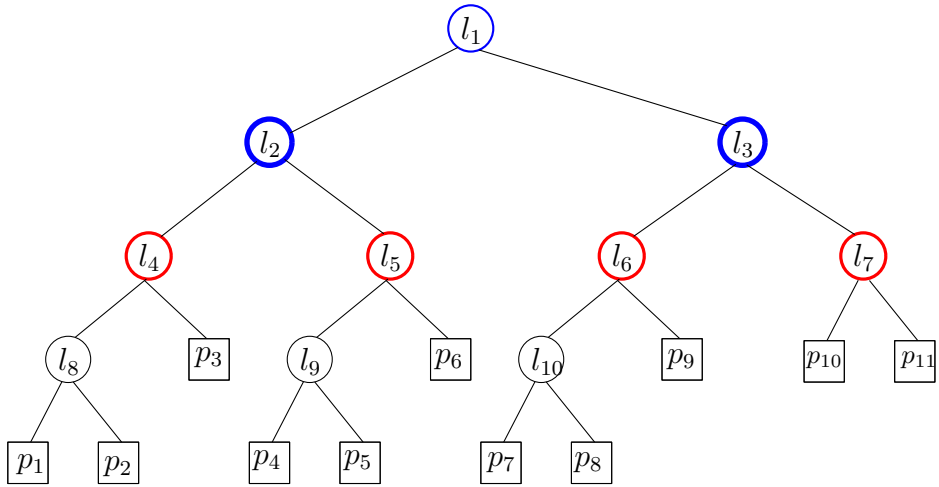
Kd-Trees and range search



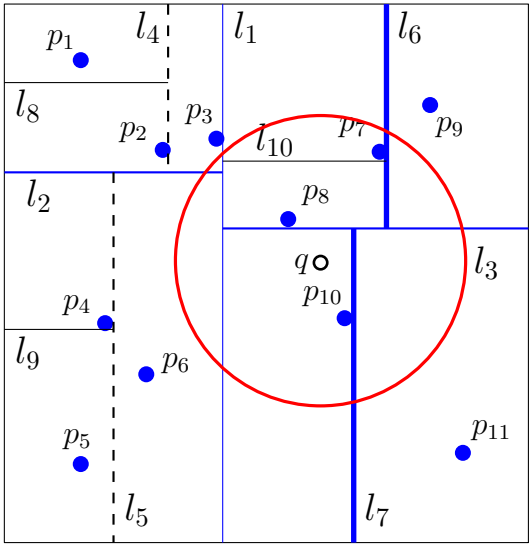


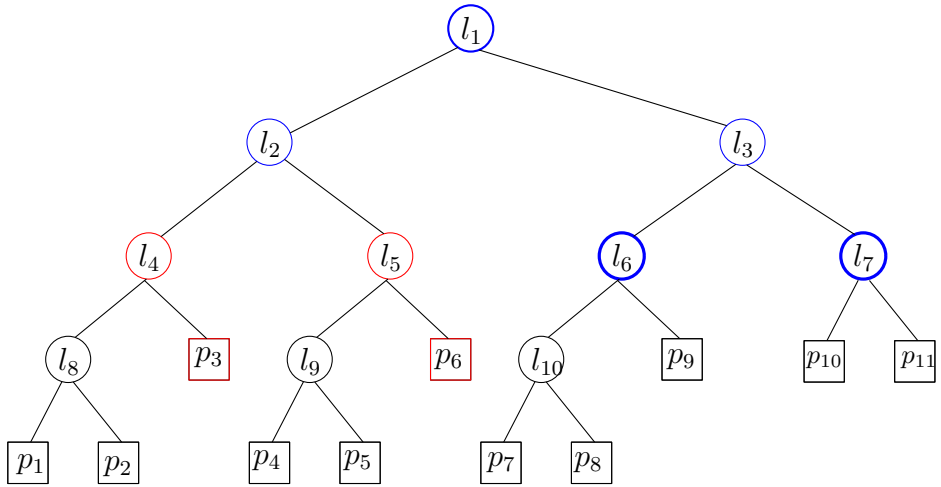
Kd-Trees and range search



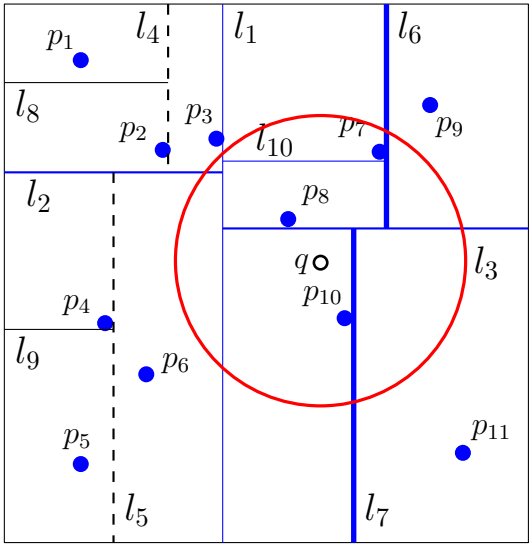


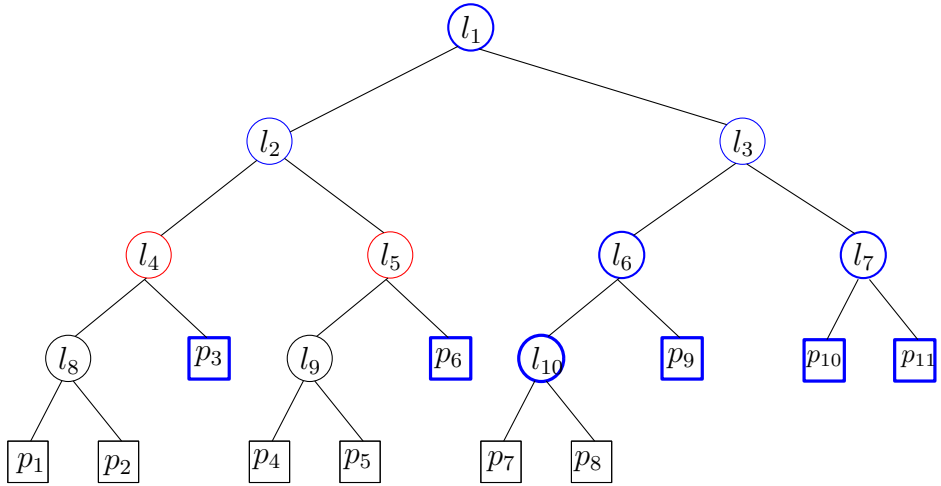
Kd-Trees and range search



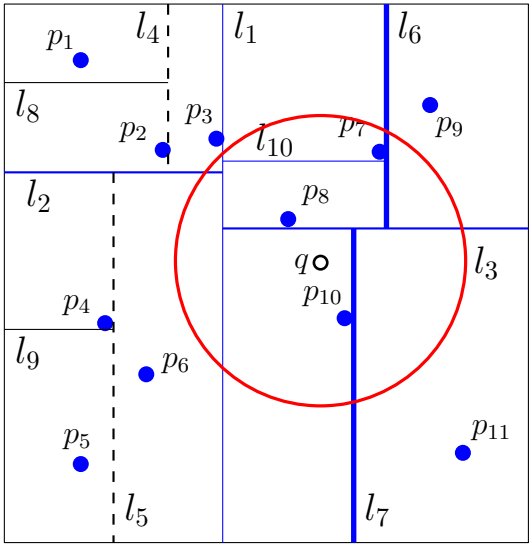


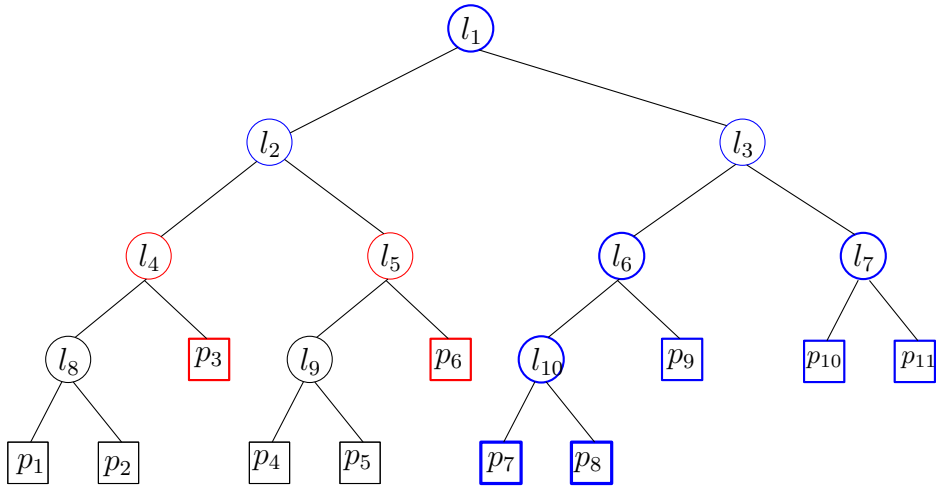
Kd-Trees and range search





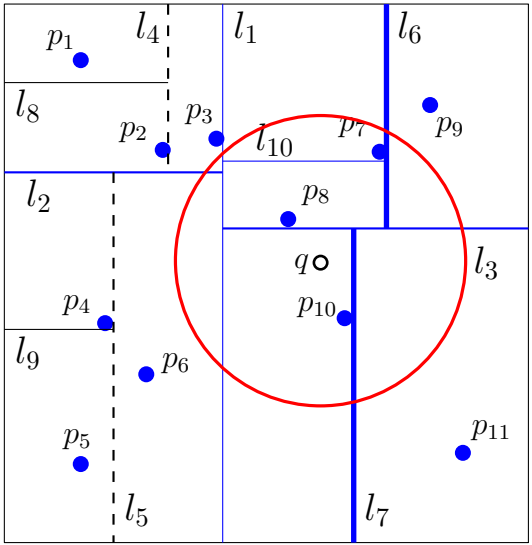
Kd-Trees and range search

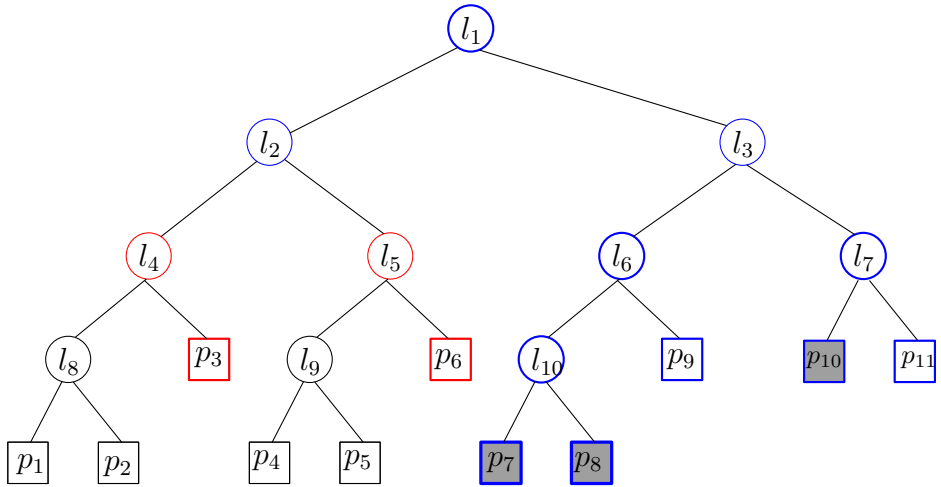




On a fini d'explorer tous les noeuds de l'arbre

Kd-Trees and range search

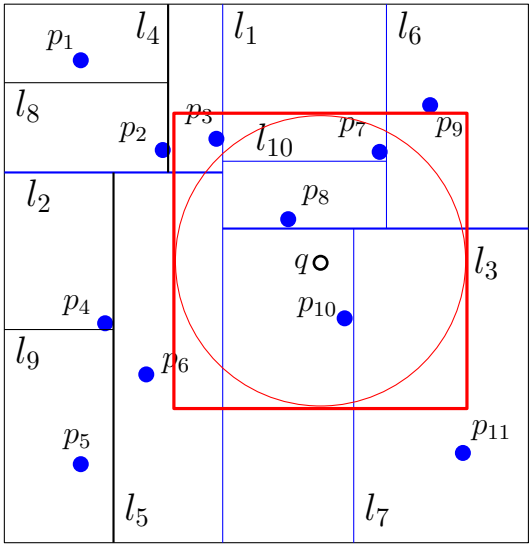


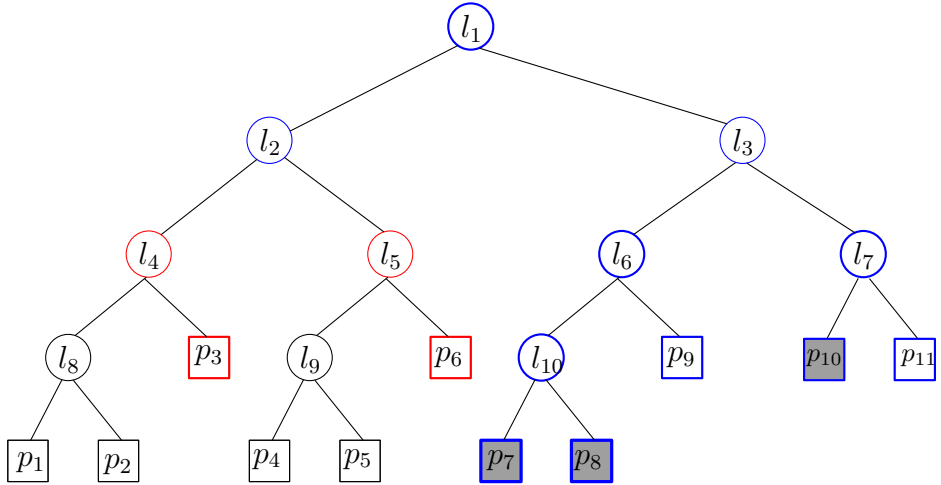


On recherche, parmi les noeuds explores, ceux qui contiennent les points proches de q

Kd-Trees et Range search: analyse de complexité

Lemme:: les Kd-Trees permettent de repondre aux *Range queries en 2D* en temps $Q(n) = O(\sqrt{n} + k)$, necessitant espace $O(n)$.

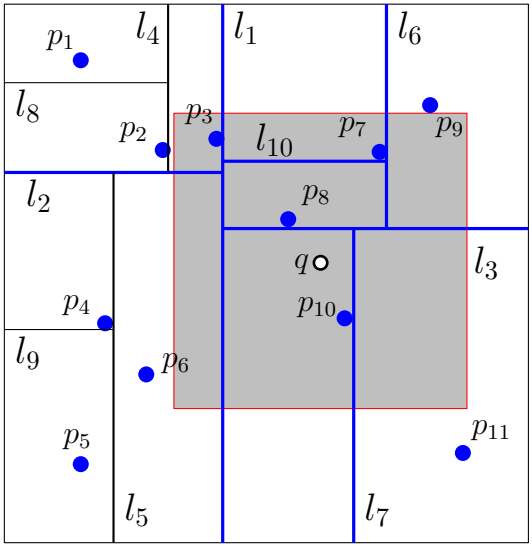


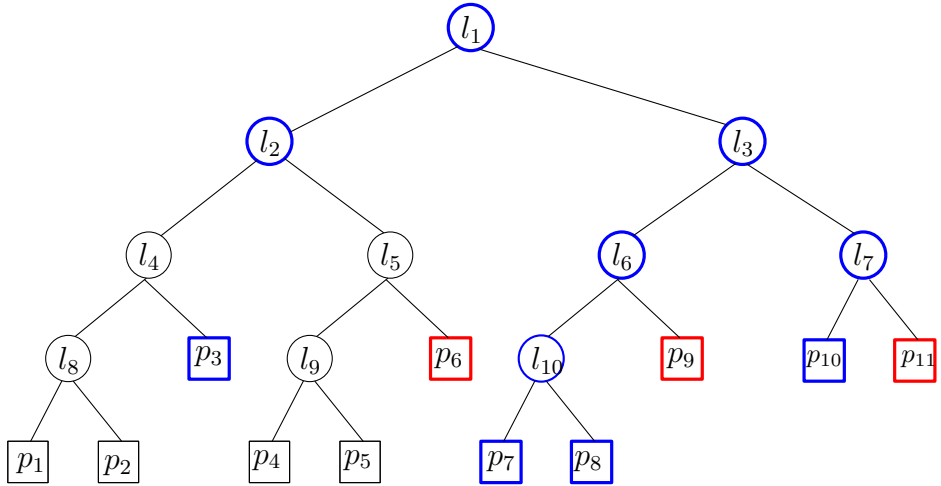


On recherche, parmi les noeuds explores, ceux qui contiennent les points proches de q

Kd-Trees et Range search: analyse de complexité

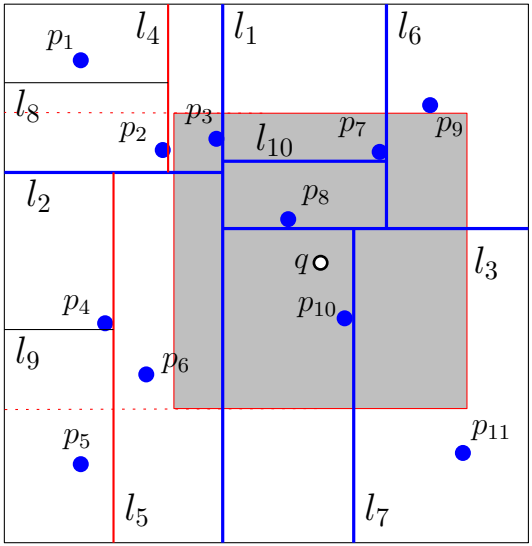
Idée: $Q(n)$ est proportionnelle au nombre de noeuds couverts par la région $R(q)$

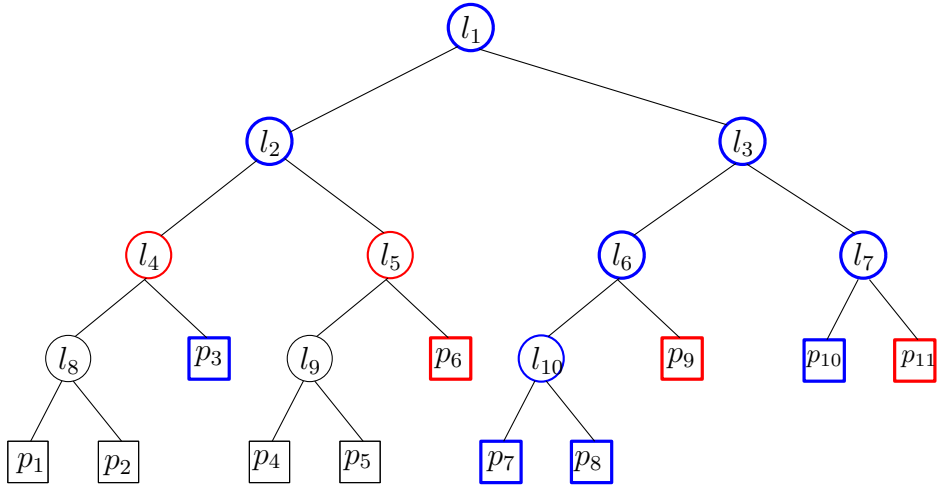




Kd-Trees et Range search: analyse de complexité

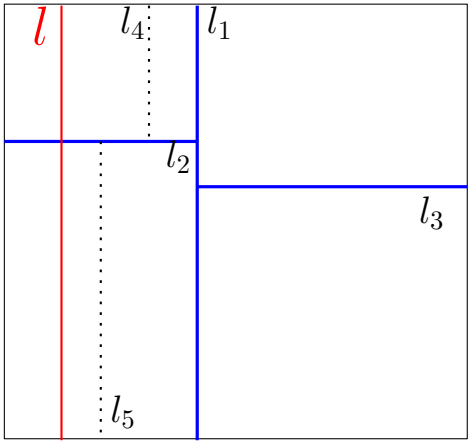
Lemme: Pour toute droite (verticale) l, il existe au plus $Int_l(n) = \sqrt{n}$ noeuds intersectés par l.

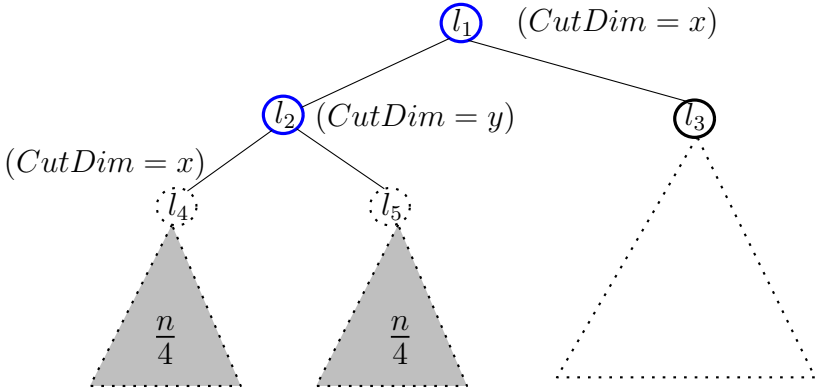




Kd-Trees et Range search: analyse de complexité

Preuve





Remarque 1: exactement un seul des deux sous-noeuds de n_1 intersècte l (appelons-le l_2).

Remarque 2: les sous arbres enracinés à l_4 et l_5 ont taille $\frac{n}{4}$.

Remarque 3:pour toute intersection de l avec un segment (horizontal) on a deux noeuds intersectés.

$$Int_l(n) = \begin{cases} O(1) & n = 1 \\ 2 + 2Int_l(\frac{n}{4}) & n > 1 \end{cases} \longrightarrow Int_l(n) = O(\sqrt{n})$$

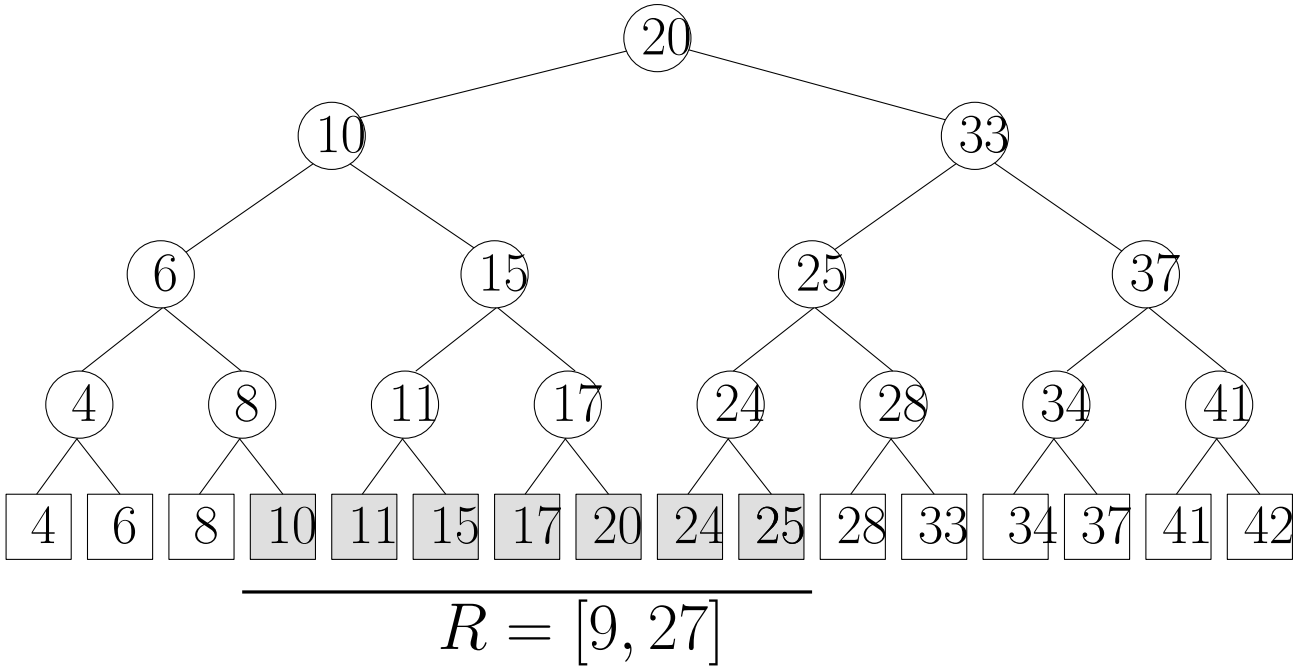
□

Range Trees

Range Queries en dim 1

Thm

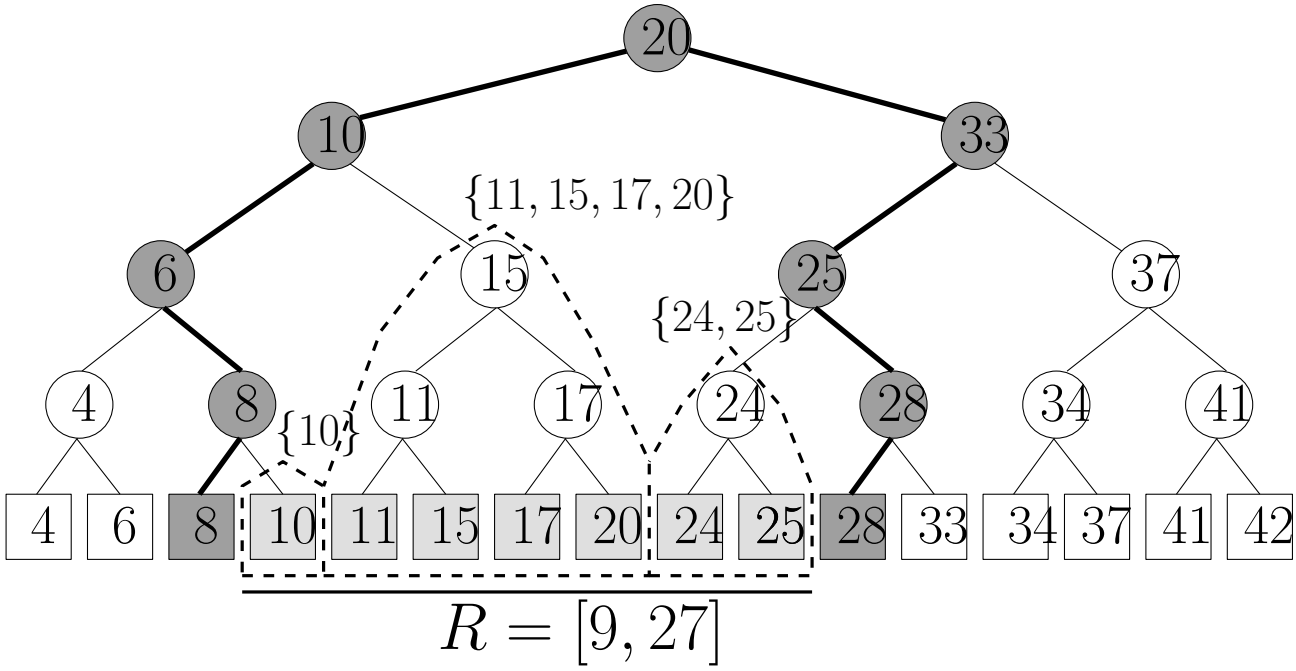
Les range trees permettent de resoudre le range probem orthogonal en dimension 1 en temps $O(k + \log n)$.



Range Queries en dim 1

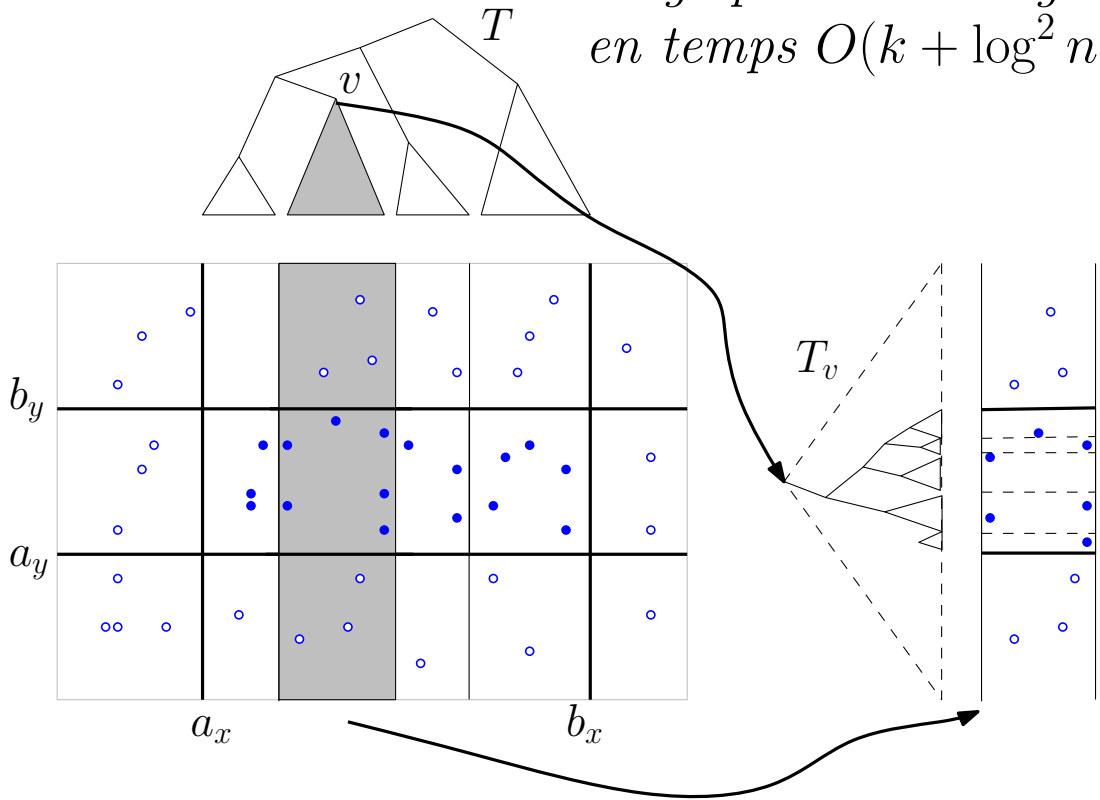
Thm

Les range trees permettent de resoudre le range probem orthogonal en dimension 1 en temps $O(k + \log n)$.



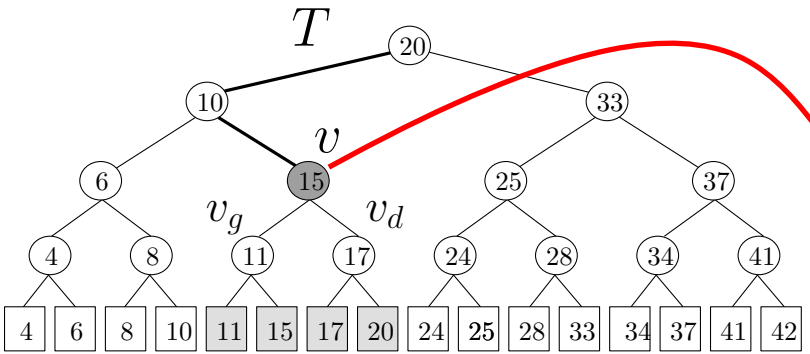
Range Queries en dim 2

Thm *Les range trees permettent de resoudre le range probem orthogonal en dimension 2 en temps* $O(k + \log^2 n)$.



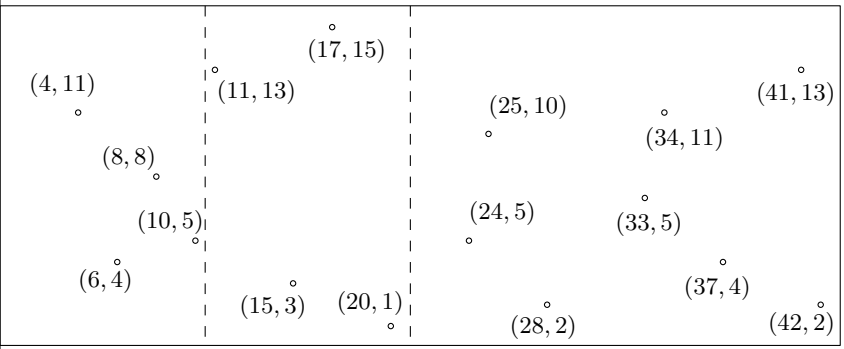
Fractional Cascading

Thm *On peut resoudre le range probem orthogonal en dimension 2 en temps* $O(k + \log n)$.



A_v

(20, 1)	(15, 3)	(11, 13)	(17, 15)
1	3	13	15



A_{v_g}

(15, 3)	(11, 13)
3	13

A_{v_d}

(20, 1)	(17, 15)
1	15

(11, 13)
13

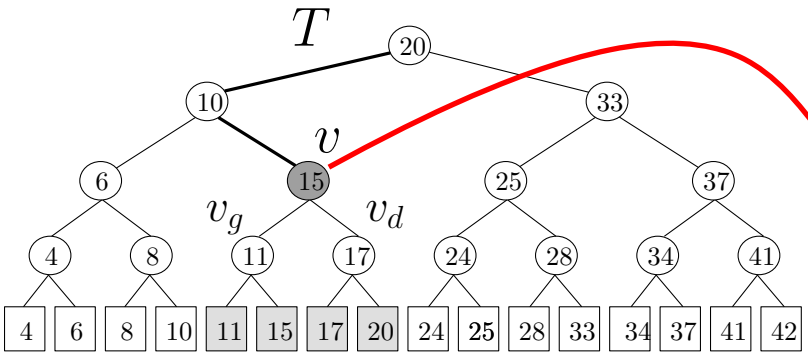
(15, 3)
3

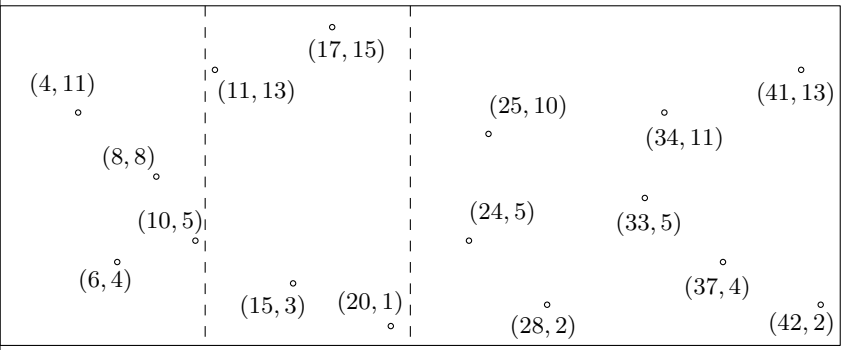
(17, 15)
15

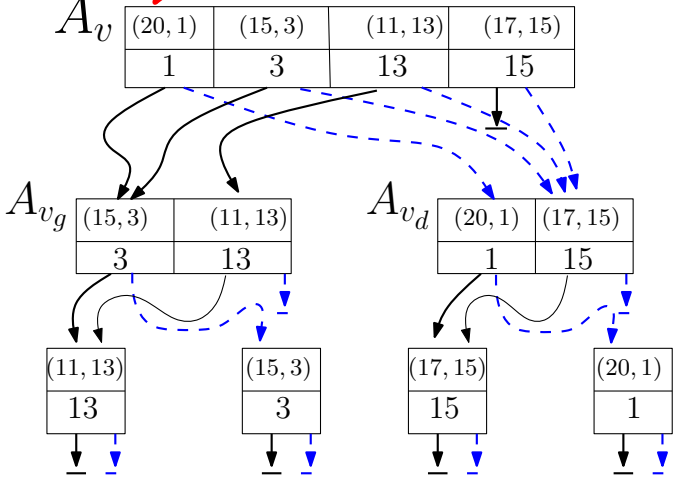
(20, 1)
1

Fractional Cascading

Thm *On peut resoudre le range probem orthogonal en dimension 2 en temps* $O(k + \log n)$.

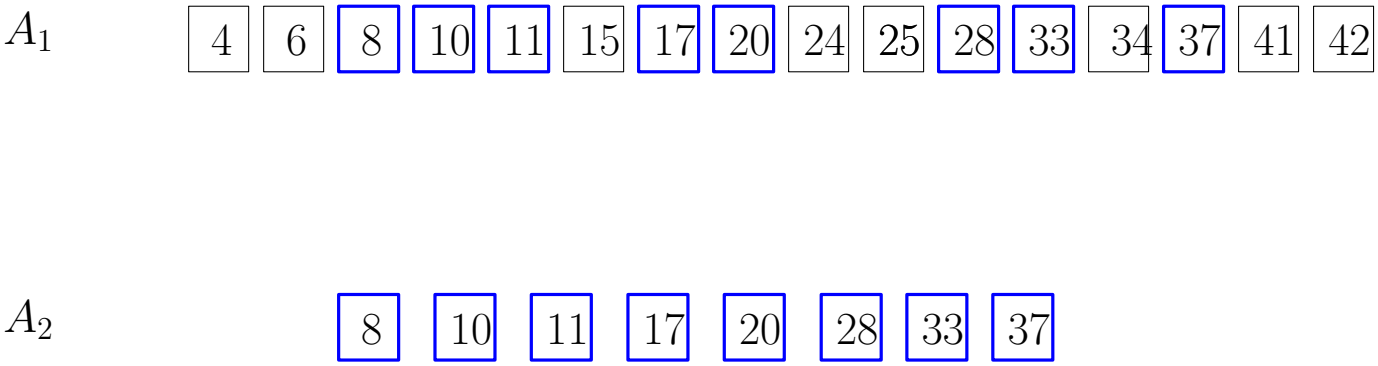






Fractional Cascading

searching in sub-lists $A_2 \subset A_1$

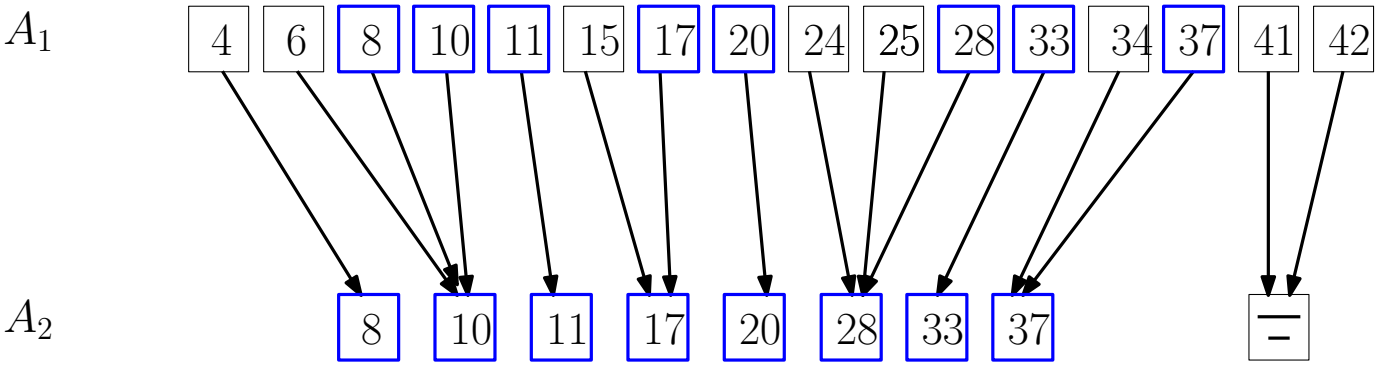


naive approach: $\log_2 |A_1| + \log_2 |A_2|$

Fractional Cascading

searching in sub-lists

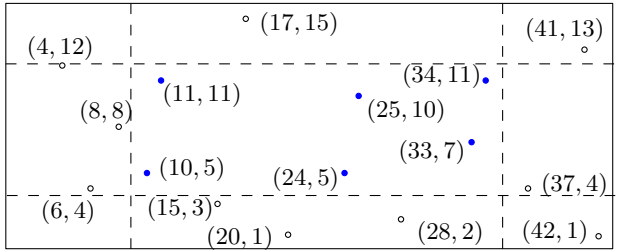
$A_2 \subset A_1$

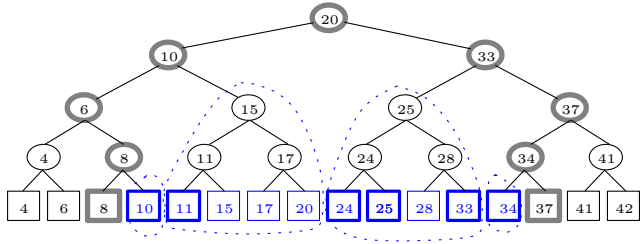


better approach: $\log_2 |A_1| + k$

solution: stores a pointer, from $A_1[i]$ to $A_2[j]$ such that j is the smallest index for which $A_1[i] \leq A_2[j]$

Fractional Cascading





A_{20}

(20, 1)	(42, 1)	(28, 2)	(15, 3)	(6, 4)	(37, 4)	(10, 5)	(24, 5)	(33, 7)	(8, 8)	(34, 11)	(11, 11)	(4, 12)	(25, 10)	(41, 13)	(17, 15)
1	1	2	3	4	4	5	5	7	8	11	11	12	10	13	15

A_{10}

(20, 1)	(15, 3)	(6, 4)	(10, 5)	(8, 8)	(11, 11)	(4, 12)	(17, 15)
1	3	4	5	8	11	12	15

A_{33}

(42, 1)	(28, 2)	(37, 4)	(24, 5)	(33, 7)	(25, 10)	(34, 11)	(41, 13)
1	2	4	5	7	10	11	13

A_6

(6, 4)	(10, 5)	(8, 8)	(4, 12)
4	5	8	12

A_{15}

(20, 1)	(15, 3)	(11, 11)	(17, 15)
1	3	11	15

A_{25}

(28, 2)	(24, 5)	(33, 7)	(25, 10)
2	5	7	10

A_{37}

(42, 1)	(37, 4)	(34, 11)	(41, 13)
1	4	11	13

A_4

(6, 4)	(4, 12)
4	12

A_8

(10, 5)	(8, 8)
5	8

A_{11}

(15, 3)	(11, 11)
3	11

A_{17}

(20, 1)	(17, 15)
1	15

A_{24}

(24, 5)	(25, 10)
5	10

A_{28}

(28, 2)	(33, 7)
2	7

A_{34}

(37, 4)	(34, 11)
4	11

A_{41}

(42, 1)	(41, 13)
1	13

(4, 12)	(6, 4)	(8, 8)	(10, 5)	(11, 11)	(15, 3)	(17, 15)	(20, 1)
12	4	8	5	11	3	15	1

Nearest neighbors and image segmentation

Mean-Shift clustering et segmentations d'images (application des Kd-Trees, voir TD)

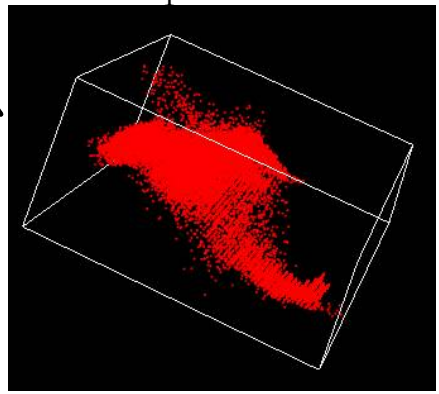
image originale (50700 pixels)

image segmentation

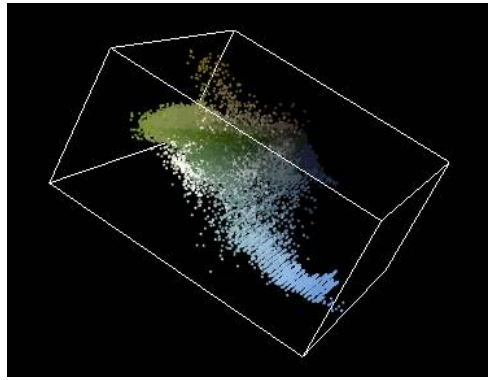
Mean-Shift clustering et segmentations d'images

image originale (50700 pixels)

50700 points en 3D
espace *Luv*

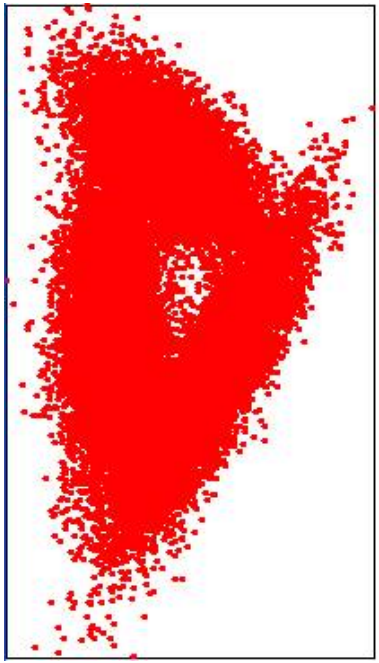


clustering



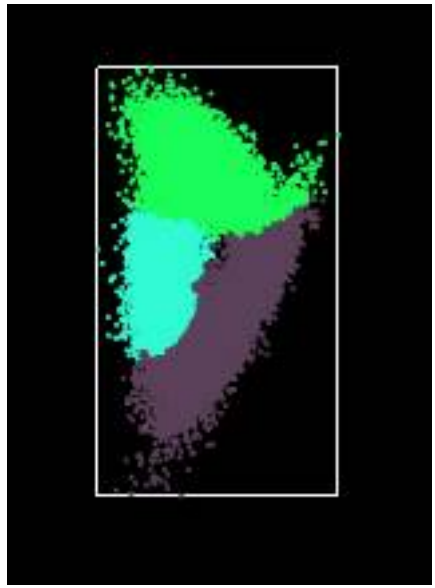
Clustering: en deux mots

n points (exemple en 2D)



clustering

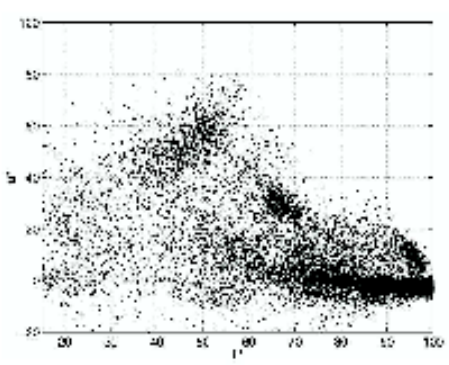
3 clusters

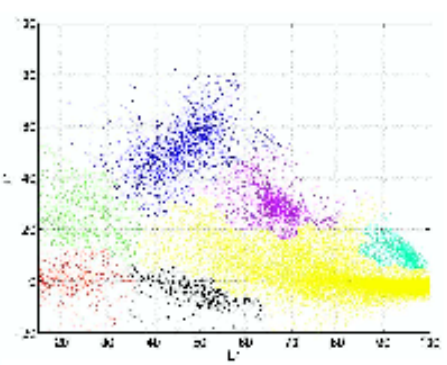


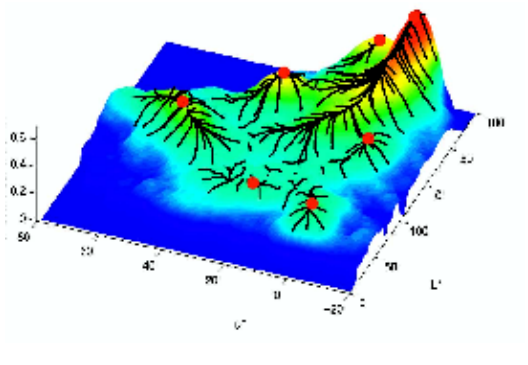
Mean-Shift clustering

n points

7 clusters







- Classification par les bassins d'attraction des maxima de l'estimateur de densité

En pratique, on ne connait pas la densite,

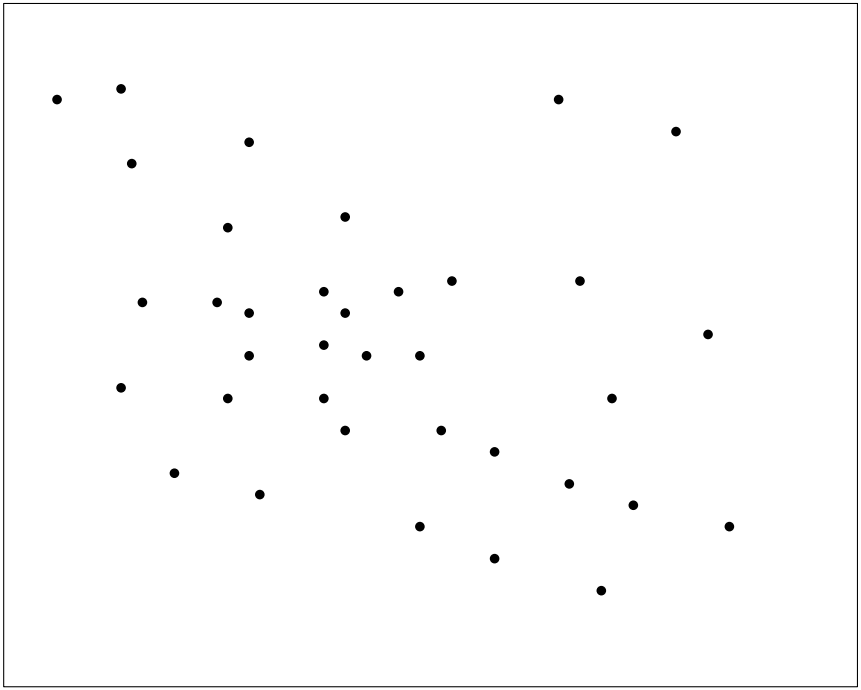
on l'estime par la methode des noyaux

imaginons qu'on connaisse la fonction densite. Alors,...

- Maxima de l'estimateur = points fixes du mean shift

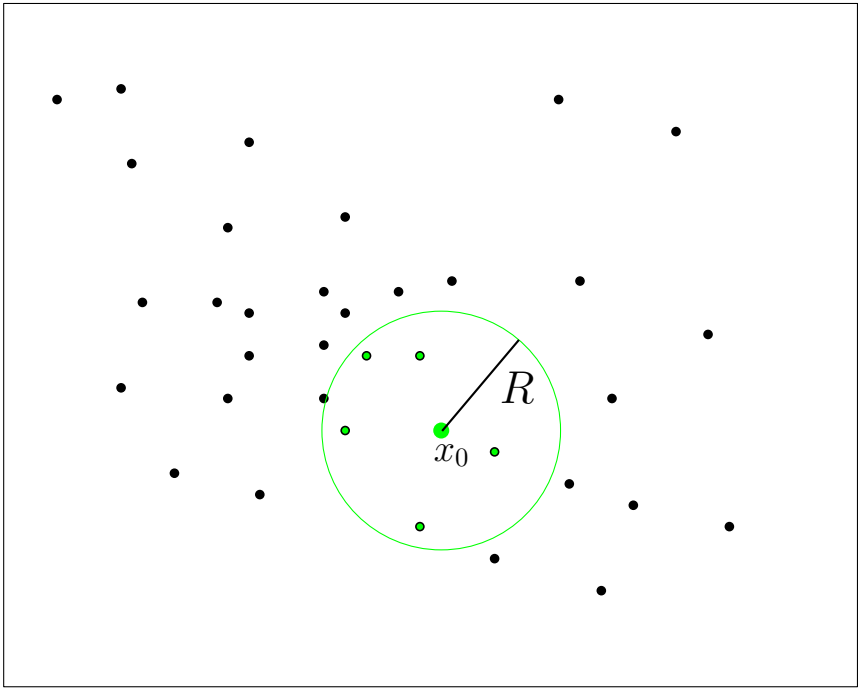
Mean-Shift clustering: detection d'un cluster

n points



Mean-Shift clustering: detection d'un cluster

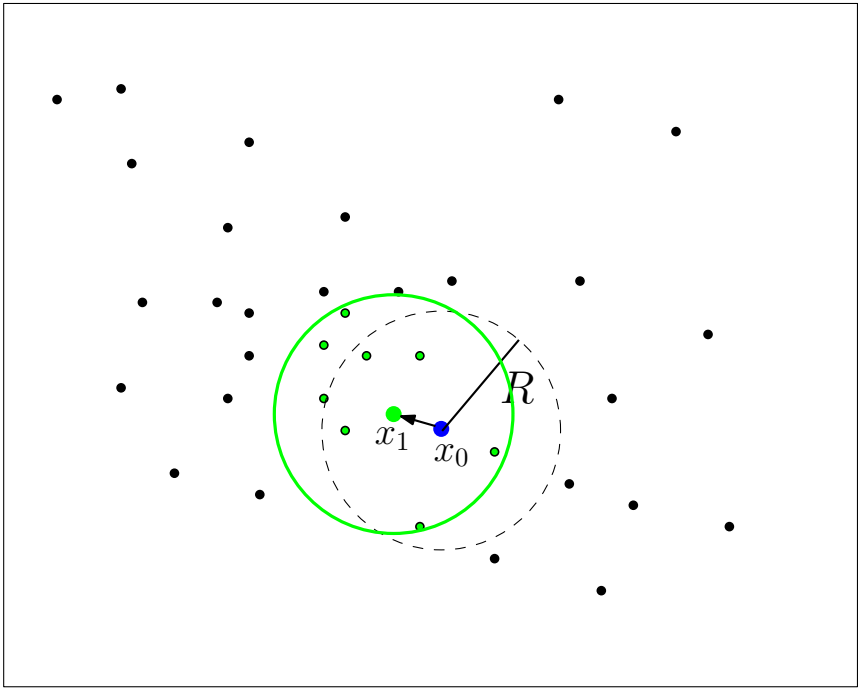
n points 1 seed



Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre

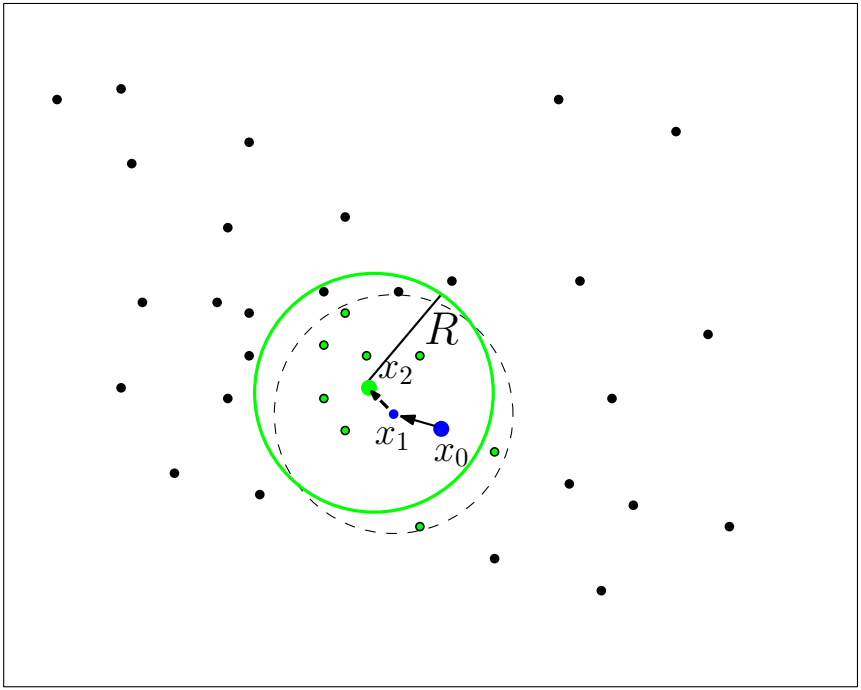
$$x_{i+1} := mean\{neighbors(x_i)\}$$



Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre

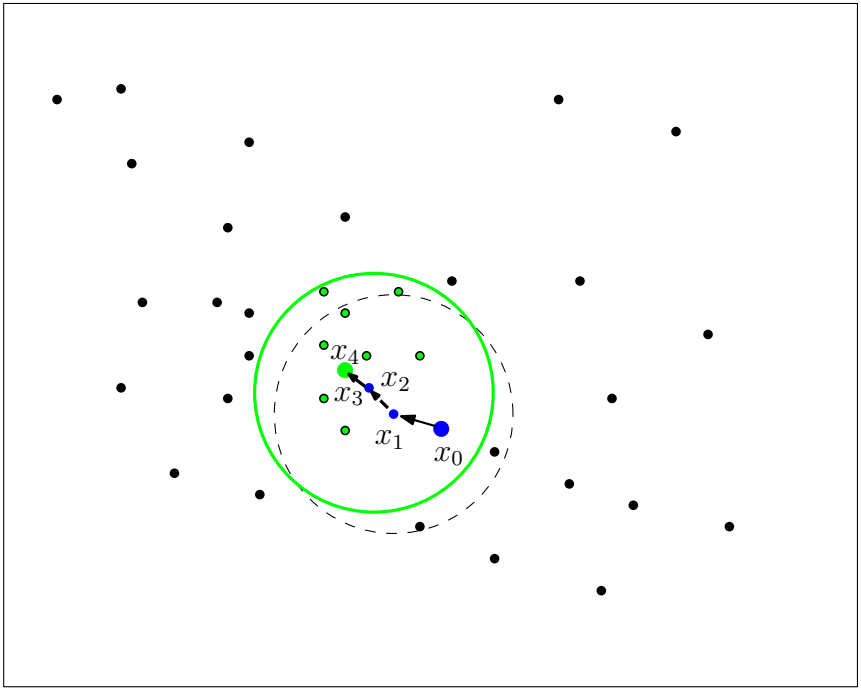
$$x_{i+1} := mean\{neighbors(x_i)\}$$



Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre

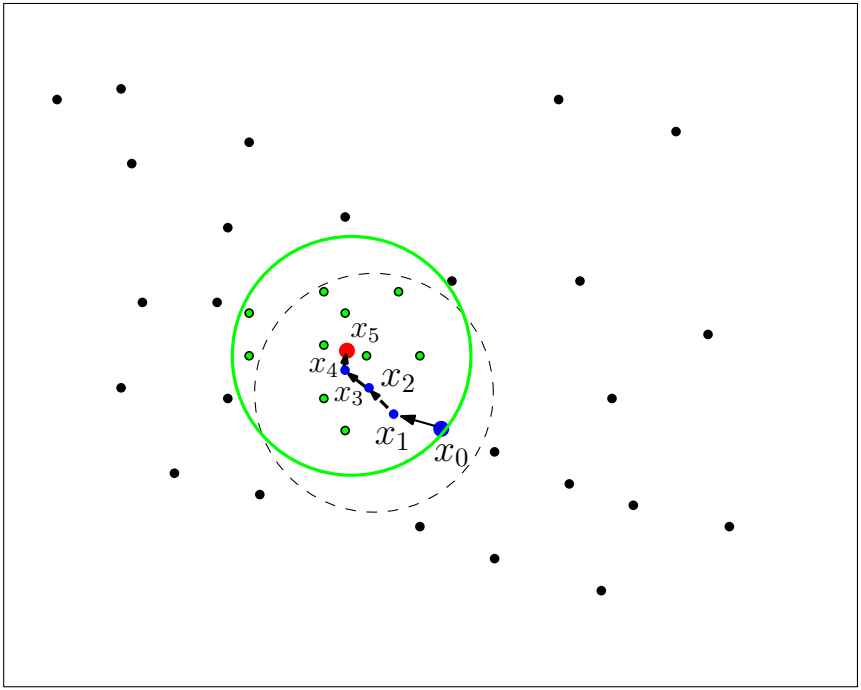
$$x_{i+1} := mean\{neighbors(x_i)\}$$



Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté

on a convergence lorsque

$$d(x_{i+1}, x_i) \leq Rc$$

x_5 point stationnaire du Mean-Shift

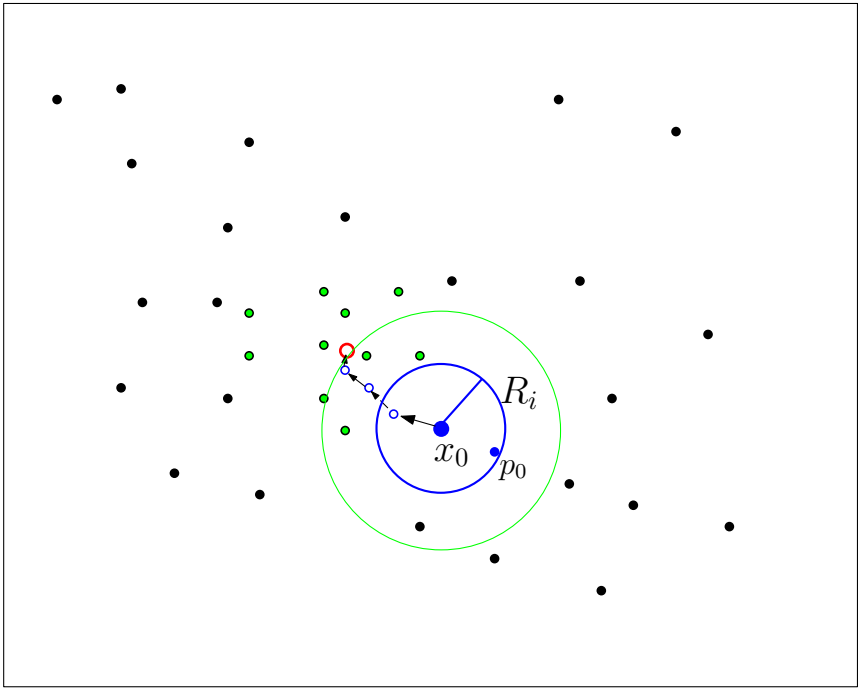
C_5 cluster associé à x_5

$$C_5 = \{x_0\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté

on a convergence lorsque

$$d(x_{i+1}, x_i) \leq Rc$$

x_5 point stationnaire du Mean-Shift

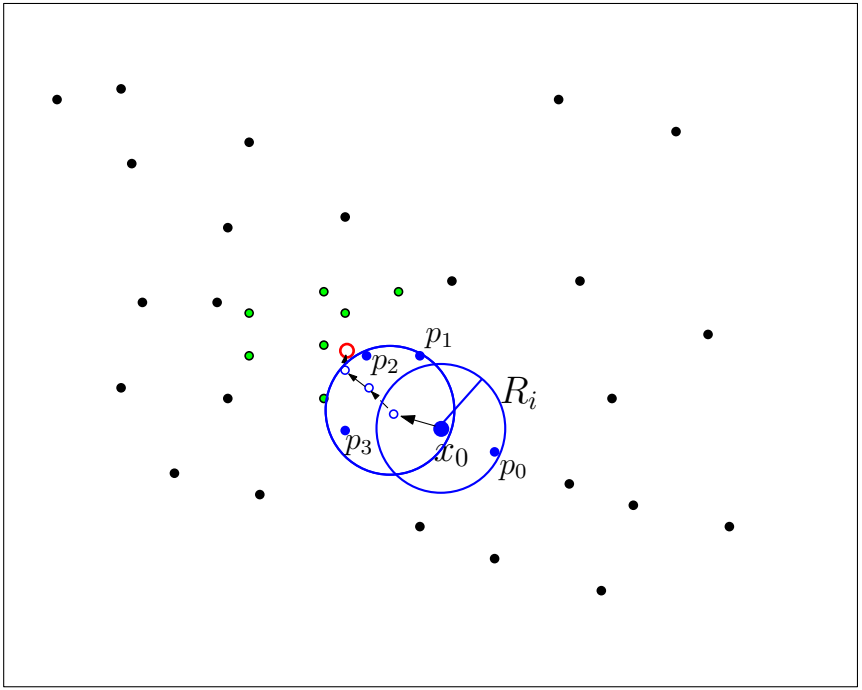
C_5 cluster associé à x_5

$$C_5 = \{x_0, p_0\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté

on a convergence lorsque

$$d(x_{i+1}, x_i) \leq Rc$$

x_5 point stationnaire du Mean-Shift

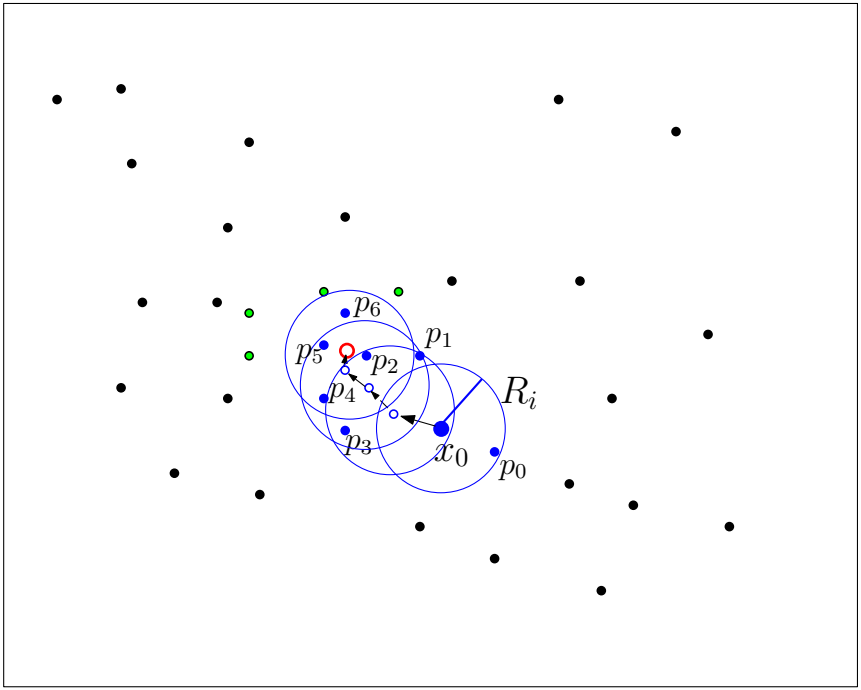
C_5 cluster associé à x_5

$$C_5 = \{x_0, p_0, p_1, p_2, p_3\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté

on a convergence lorsque

$$d(x_{i+1}, x_i) \leq Rc$$

x_5 point stationnaire du Mean-Shift

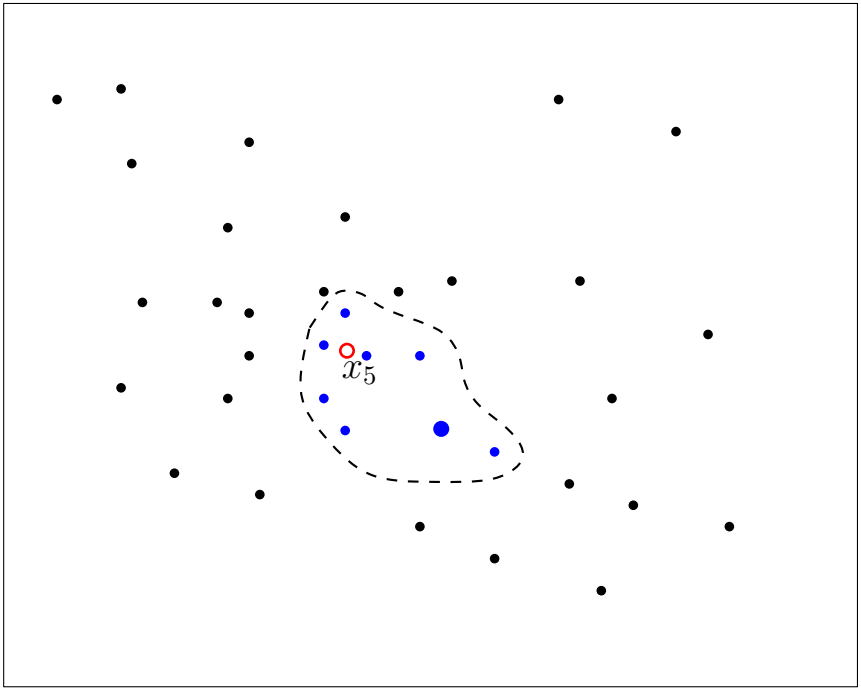
C_5 cluster associé à x_5

$$C_5 = \{x_0, p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté

on a convergence lorsque

$$d(x_{i+1}, x_i) \leq Rc$$

x_5 point stationnaire du Mean-Shift

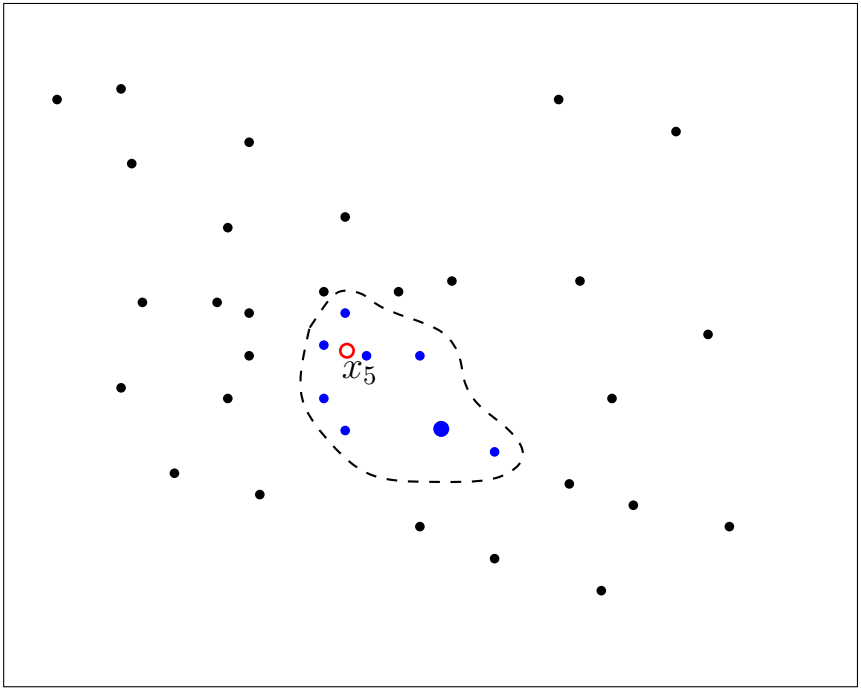
C_5 cluster associé à x_5

$$C_5 = \{x_0, p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté
on a convergence lorsque
$d(x_{i+1}, x_i) \leq Rc$

x_5 point stationnaire du Mean-Shift

C_5 cluster associé à x_5

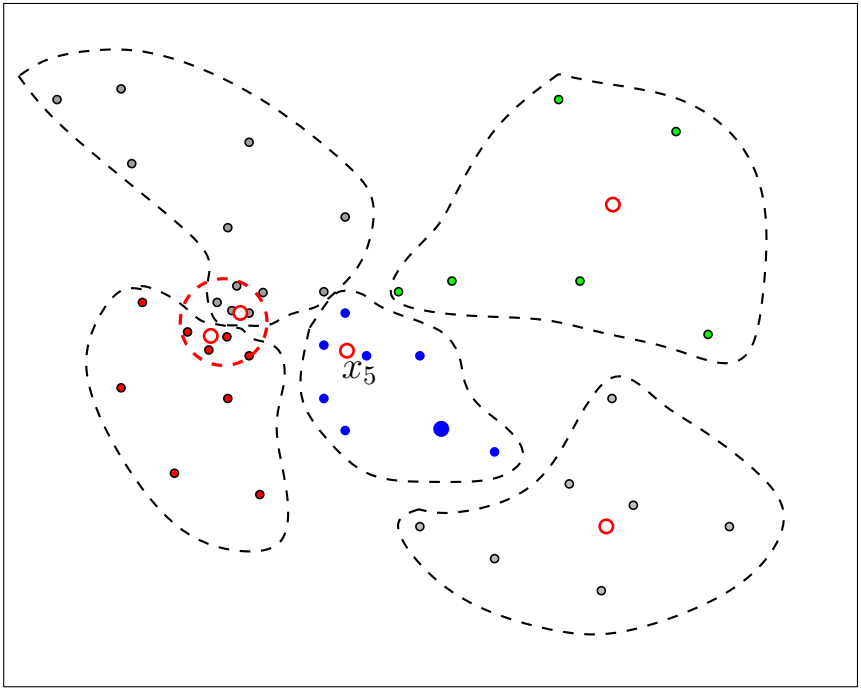
cluster center

$$C_5 = \{x_0, p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

$$x_{i+1} := mean\{neighbors(x_i)\}$$



cluster detecté
on a convergence lorsque
$d(x_{i+1}, x_i) \leq Rc$

x_5 point stationnaire du Mean-Shift

C_5 cluster associé à x_5

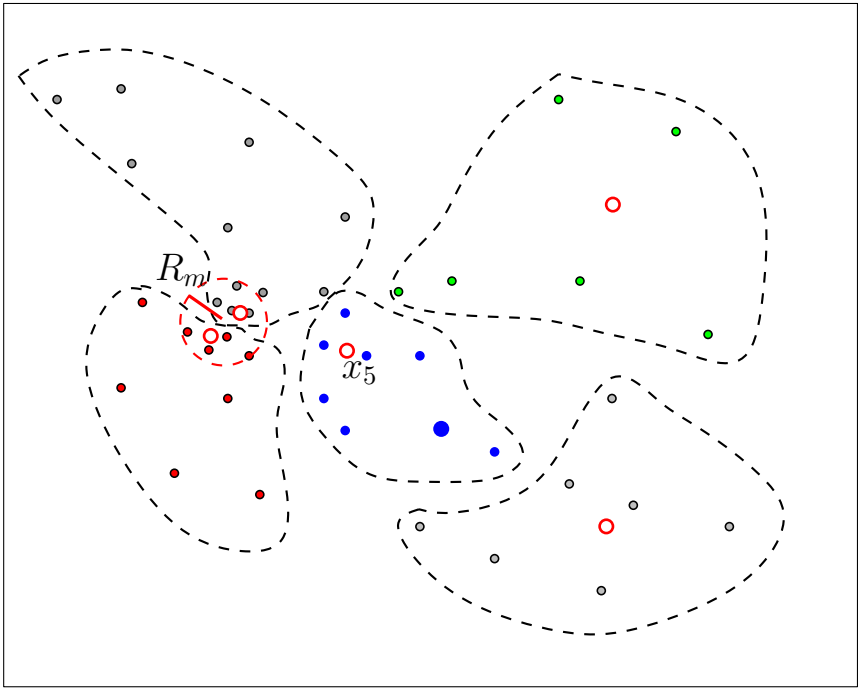
cluster center

$C_5 = \{x_0, p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$
$C_{10} = \{p_7, p_8, \ldots\}$
$C_{\ldots} = \{\ldots\}$

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

R_m rayon de fusion



fusion de clusters (proches)

x_p, x_q points stationnaires du Mean-Shift

x_p, x_q cluster centers

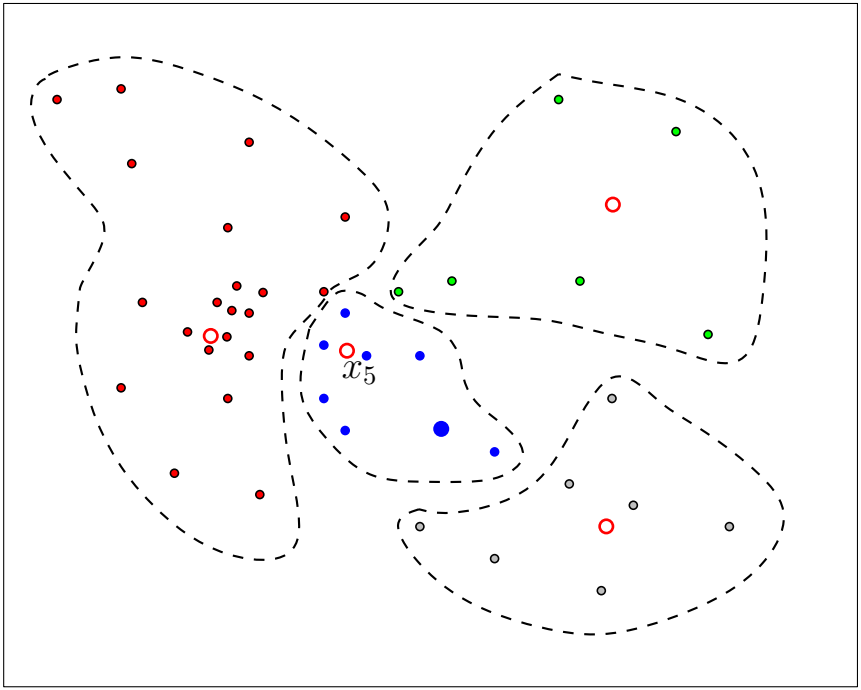
$d(x_p, x_q) \leq R_m$

fusionner C_p et C_q

Mean-Shift clustering: detection d'un cluster

n points 1 seed R rayon de la fenetre R_i rayon d'influence

R_m rayon de fusion



fusion de clusters (proches)

x_p, x_q points stationnaires du Mean-Shift

x_p, x_q cluster centers

$d(x_p, x_q) \leq R_m$

fusionner C_p et C_q