Computational Geometry

Masters in Computer Science

Le service...

Eric Béchet (it's me !)

- Studied engineering in Nancy (Fr.)
- Ph.D. in Montréal (Can.)
- Academic career back in Europe in Nantes, Metz (Fr.) and then Liège ...

Website

http://www.cgeo.ulg.ac.be/CG

Contact : **eric.bechet@ulg.ac.be**

Déroulement du cours

- 6-7 classes (theory) – 2 to 4 hours
- 30 hours of individual practical work – 5 or 6 subjects to solve + small reports
- Project – subject TBD with E. Bechet
- Assessment : Project 80 %, reports 20 %
- Availability hours : Friday mornings

  Office B52 bldg, room +2/438

Déroulement du cours

- Lectures take place on Wednesday, 2PM room -1/433, bldg B52
- Starts on Sept. 20th then :
  - Oct. 4th and 18th , Nov. 8th , 15th and 29th , Dec 13th (always indicated on the website of the course)
- Prerequisites : bases in C or C++ programming languages; elementary geometry.
- Recommended book : M. de Berg, O. Cheong , M. van Keveld, M. Overmars , Computal Geometry, 3rd ed. 2008, Springer-Verlag

  Available in electronic format at the library !

Introduction

- Exemple : soif urgente

  On veut se rendre au bar le plus proche.

  Il faut une carte qui subdivise le quartier en régions et indique pour chaque région, le bar le plus proche

  - Quelle est la forme des régions ?
  - Comment construire la carte ?
  - Comment savoir dans quelle région on se trouve ?
  - ...

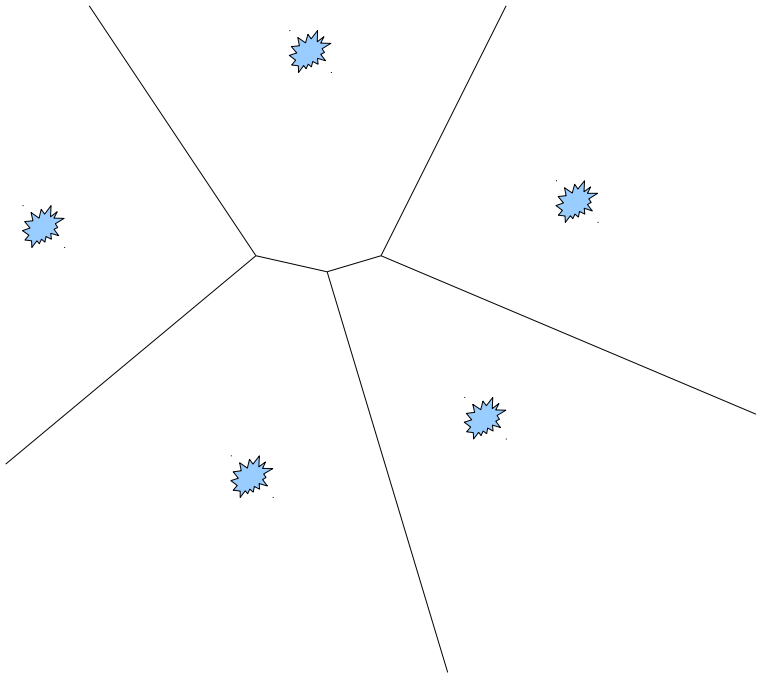
Introduction

Problème : Soif urgente

Solution : Diagramme de Voronoï.

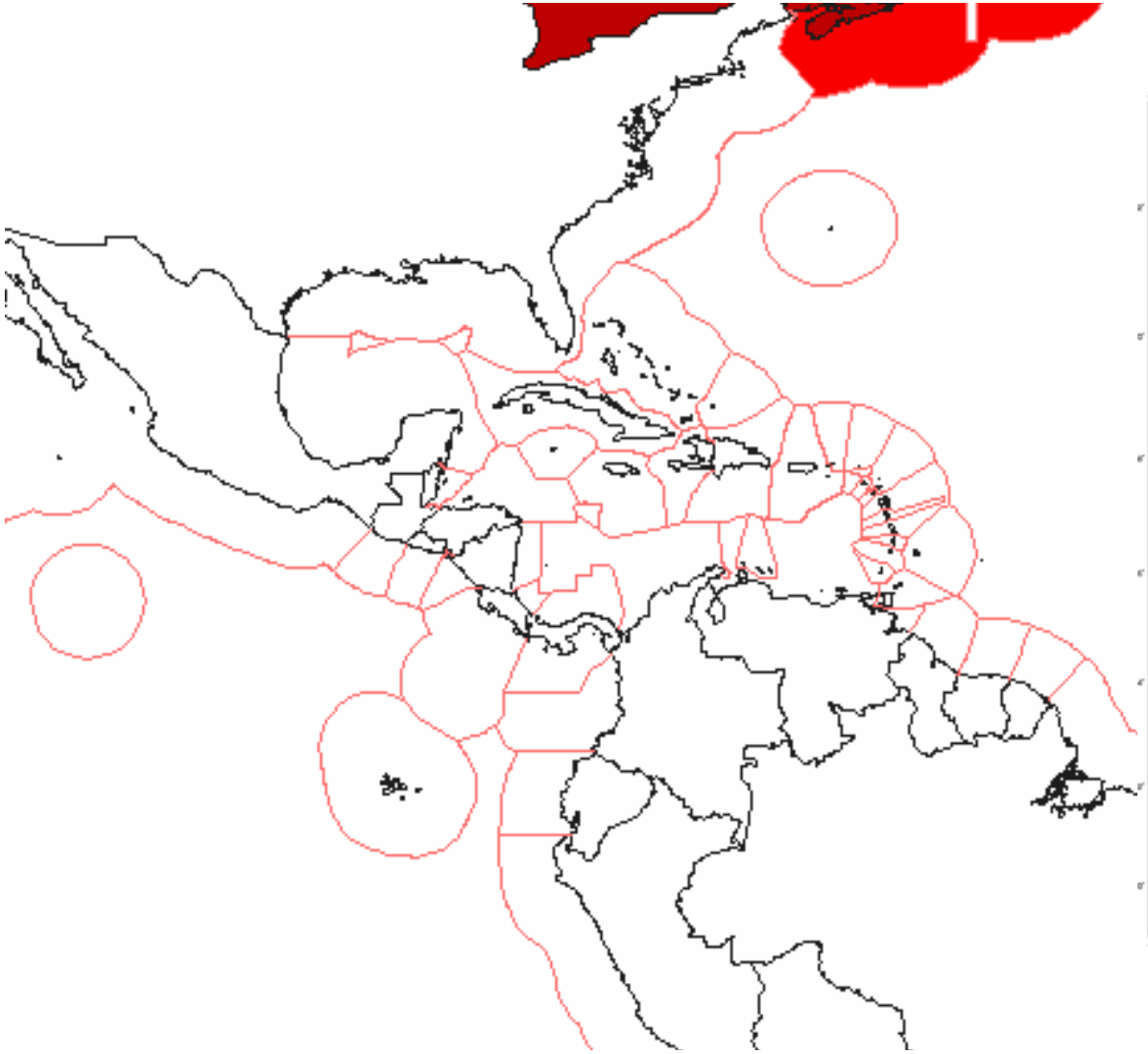
Utile dans beaucoup d'autres applications

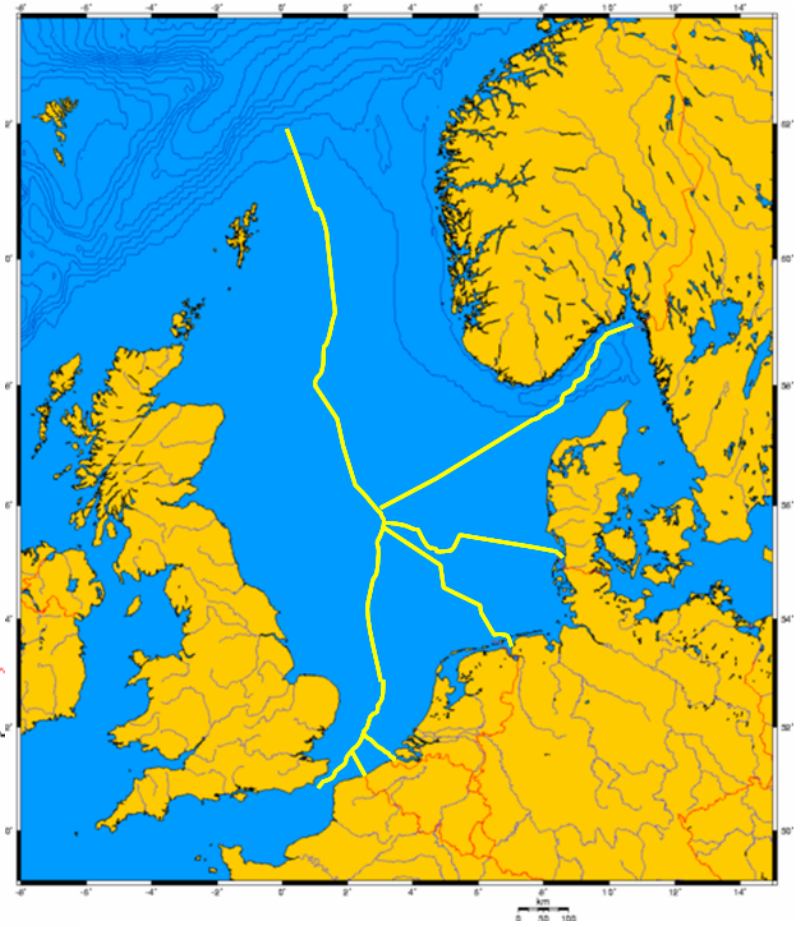
- Détermination (en partie) des zones économiques exclusives
- Guidage de robots
- Croissance des cristaux
- etc...



Introduction

- Zones économiques exclusives...





Introduction

- Exemple 2
  On dispose de deux cartes
  - Description des bâtiments, y compris les bars
  - Description des routes et chemins

    Pour prévoir un déplacement, il faut combiner ces deux cartes. Cela suppose de localiser les entités d'une carte sur l'autre, des calculs d'intersections etc...

    Ce sont des problèmes courants dans le cadre des « systèmes d'information géographiques » (ex. Google maps, openstreetmaps)

Plan du cours (tentative)

- Introduction
  - Some notions of algorithmic complexity
  - One example : convex hull of a set of points in 2D
  - Basic data structures
- Line intersections
- Polygon triangulation
- Geometric search
- Voronoï diagrams
- Delaunay Triangulations
- Binary Space Partition Trees

Introduction

Complexité algorithmique

- Notion de complexité algorithmique

  Définition suivante donnée par D. Knuth (1976)

  - Notation grand O :
    $g(n) \in O(f(n))$ ssi $\exists C > 0$ tel que $g(n) \leq Cf(n) \forall n \geq n_0$
    Fonctions au plus aussi grandes que $Cf(n)$: borne supérieure de la complexité
  - Notation grand Oméga :
    $g(n) \in \Omega(f(n))$ ssi $\exists C > 0$ tel que $g(n) \geq Cf(n) \forall n \geq n_0$
    Fonctions au moins aussi grandes que $Cf(n)$: borne inférieure de la complexité

Complexité algorithmique

- Notation petit thêta :
  $g(n) \in \theta(f(n))$ ssi $\exists C_1, C_2 > 0$ tels que
  $C_1 f(n) \leq g(n) \leq C_2 f(n) \forall n \geq n_0$
  Fonctions du même ordre de grandeur

  C'est de ce concept dont on a besoin pour qualifier un algorithme d'optimal (par rapport à un résultat théorique)
- Notation petit o :
  $g(n) \in o(f(n))$ ssi $\forall C > 0$, $g(n) \leq Cf(n) \forall n \geq n_0$
  Fonctions au plus aussi grandes que $Cf(n)$ (sans inclure $\theta(f(n))$), pour tout C... (y compris $C \to 0$)

Complexité algorithmique

La complexité peut porter sur :

- Le temps d'exécution T
- La taille mémoire maximale occupée pour compléter l'opération, M

Il existe deux « types » de mesures de la complexité

- Sur le « pire cas » (limite = performance théorique)

  On obtient une borne supérieure de la complexité

- Sur un « cas moyen » (performance pratique)

  Problème : définition de ce qu'est un cas typique d'utilisation de l'algorithme.

Complexité algorithmique

- Sur le « pire cas »

  C'est généralement un résultat que l'on peut établir théoriquement

- Sur un « cas moyen »

  Résultats théoriques plutôt rares :

  - Problème de la définition du cas moyen

  - Difficultés mathématiques considérables même si la distribution considérée (cas moyen) est simple

  Malheureusement (ou heureusement), il arrive que la mesure de la complexité sur le cas moyen soit plus avantageuse que la complexité sur le cas le pire.

  - Un algorithme de complexité théorique mauvaise peut se révéler bon à l'usage , en fonction des données fournies en entrée...
  - Il reste indispensable de *tester*... d'autre part, la complexité théorique ne dit rien sur le coefficient de proportionnalité ...

Complexité algorithmique

- On peut aussi calculer la complexité par rapport à la taille de la solution du problème

  Parfois, la borne est dépendante de cette taille mais meilleure que par rapport à la taille du problème initial

  On dit alors que l'on parle d'algorithmes de complexité « sensible à la sortie »

Complexité algorithmique

Classification des algorithmes géométriques

- Algorithmes déterministes

  Pour une entrée donnée, ils s'exécutent toujours de la même façon. Le résultat est bien entendu le même.

- Algorithmes « randomisés »

  Pour les mêmes données d'entrée, le même algorithme peut s'exécuter différemment, mais le résultat ne dépend pas du chemin choisi...

  - Intérêt : profiter d'une complexité pour un cas moyen même en cas de données en entrée correspondant à un cas le pire.
  - Attention avec les calculs en virgule flottante...

Complexité algorithmique

- Algorithmes incrémentaux

  Ils permettent la construction « en ligne » de la solution, par ajout des primitives une à une dans un ordre imposé par l'utilisateur.

- Algorithmes dynamiques

  Ils permettent l'ajout ou le retrait (délétion) de primitives en gardant une solution valide à tout moment.

- Les algorithmes qui ne rentrent pas dans ces catégories prennent l'ensemble des données « en une fois ». En cas de modification, il faut recommencer l'ensemble de la construction...

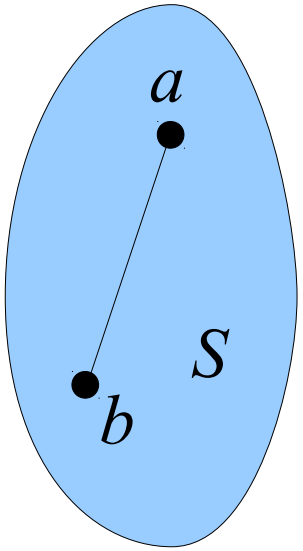
Complexité algorithmique

- Algorithmes efficaces en géométrie
  - Souvent conçus en suivant des modèles classiques
    - Approches de type « diviser pour régner »
    - Récursion
    - ...
    - Cf livres classiques en algorithmique (Knuth par exemple)
  - Utilisation du caractère géométrique
    - Algorithmes de balayage
      - Dans le plan, on simule le passage d'une droite...

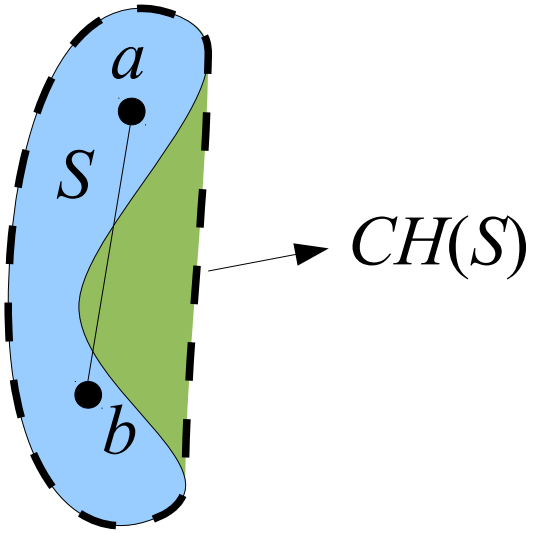
Enveloppe convexe en 2D

Convexité...

- Un sous ensemble S est convexe ssi pour toute paire de points $a, b \in S$, le segment de droite $\overline{ab}$ est entièrement compris dans S.
- L'enveloppe convexe (*convex hull*) d'un ensemble S est le plus petit ensemble $CH(S)$ convexe, contenant S. C'est aussi l'intersection de tous les ensembles convexes contenant S.



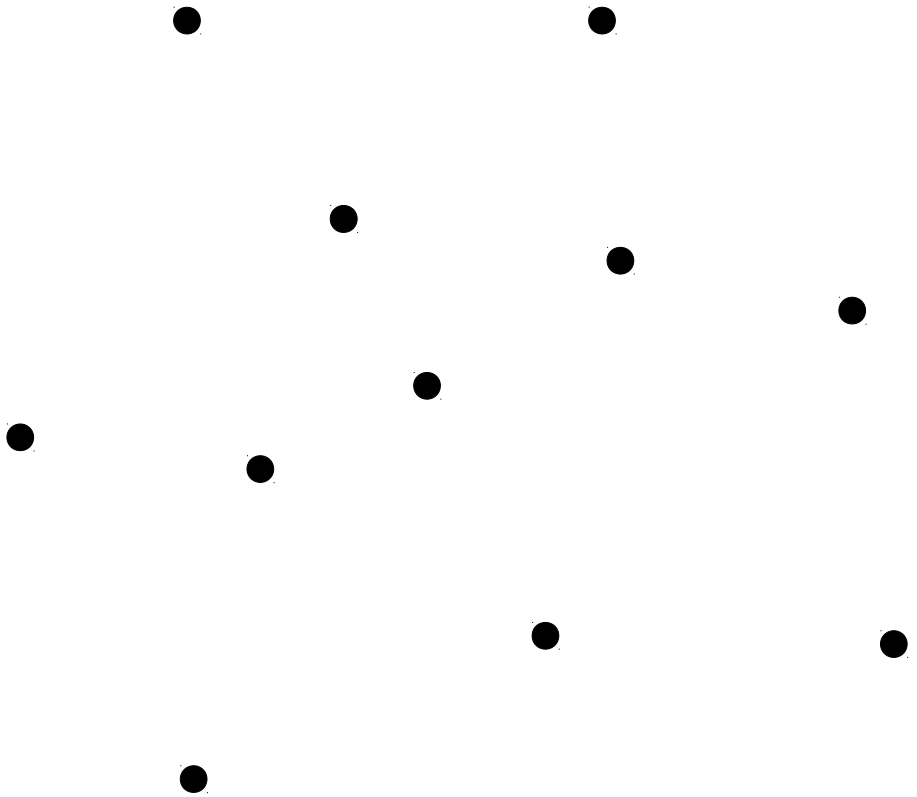
S est convexe



S est non convexe

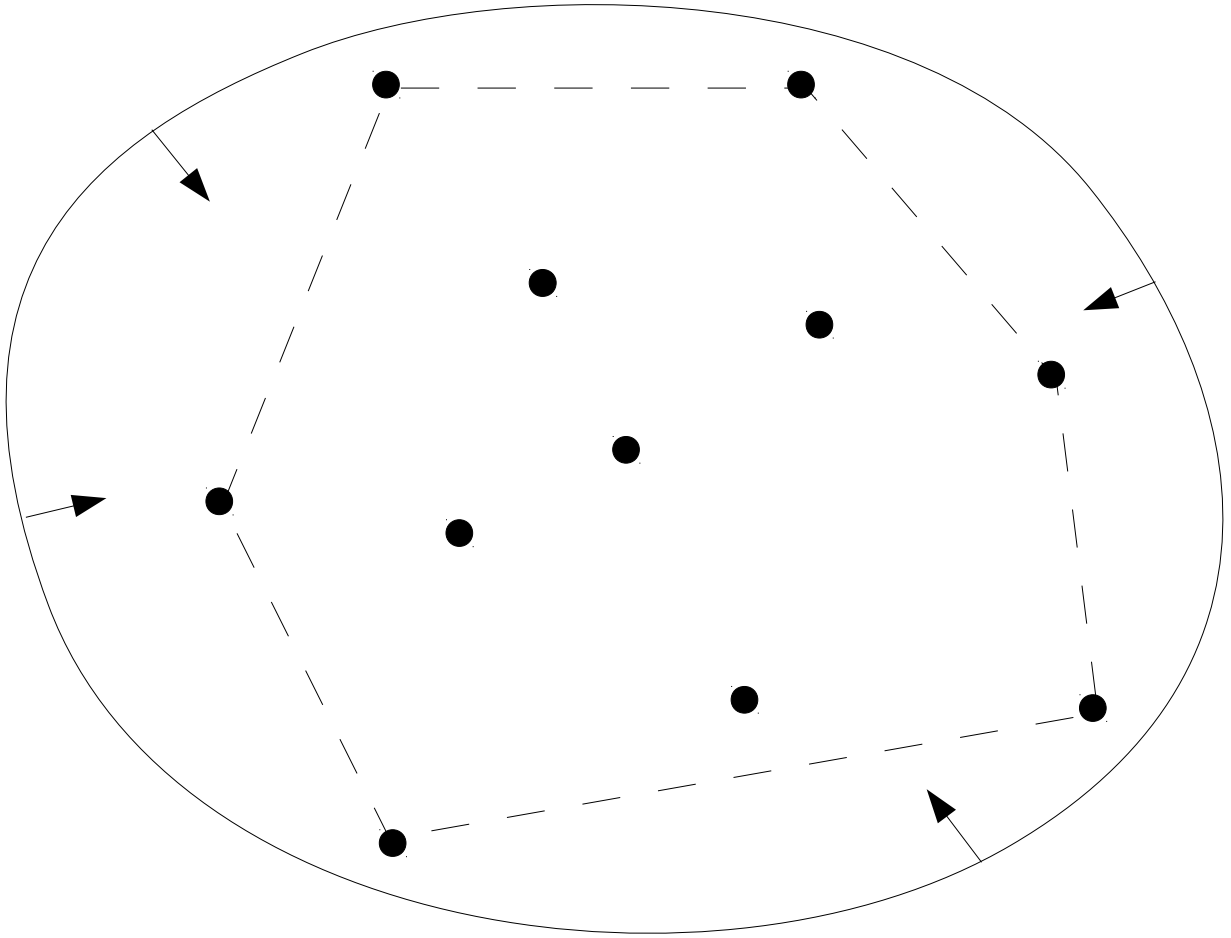
Enveloppe convexe d'un ensemble de points

- Soit P un ensemble de n points du plan.
  On souhaite calculer l'enveloppe convexe de cet ensemble.



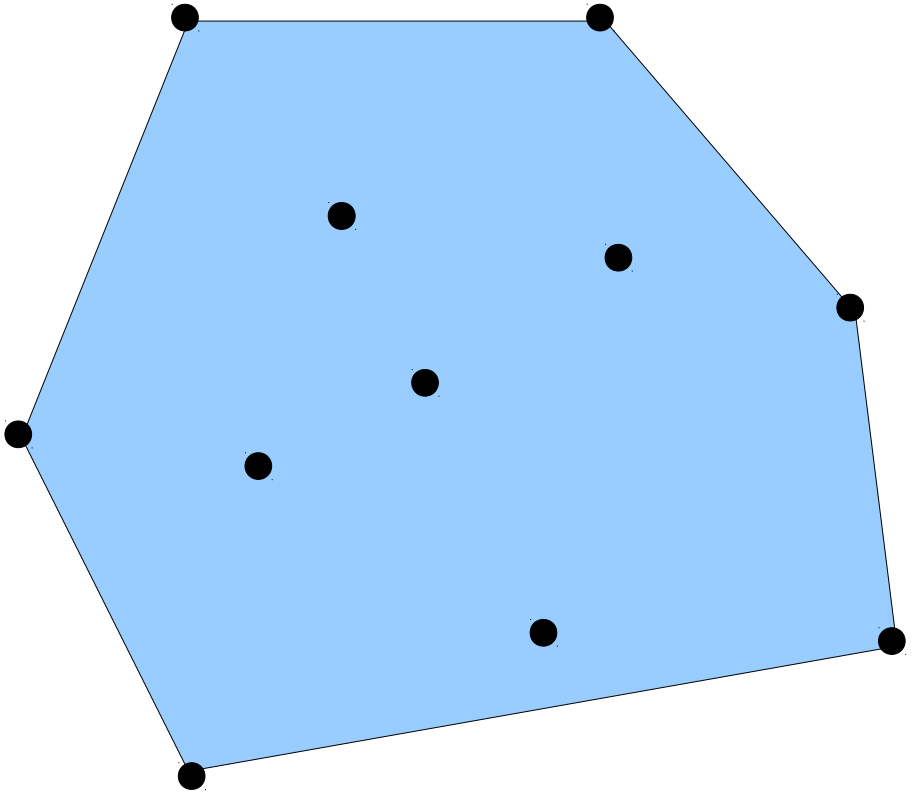
Enveloppe convexe d'un ensemble de points

C'est le polygone (fermé) dont les sommets appartiennent à P et qui contient tous les points de P.



Enveloppe convexe d'un ensemble de points

- La définition précédente est équivalente à la définition originale (il faudrait le prouver ...)

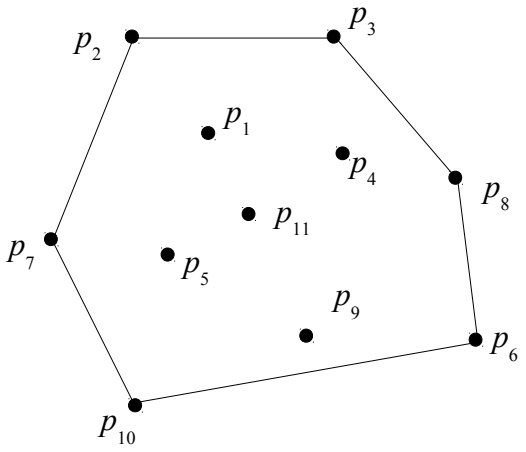


Enveloppe convexe d'un ensemble de points

- Comment *calculer* l'enveloppe convexe ?

  Que veut dire calculer ?
  Définir les données d'entrée et la sortie

  P.ex. on peut lister les sommets du polygone dans le sens horaire.



Entrée = ensemble de points P

$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}$

Sortie = enveloppe convexe $CH(P)$

$p_7, p_2, p_3, p_8, p_6, p_{10}$

Enveloppe convexe d'un ensemble de points

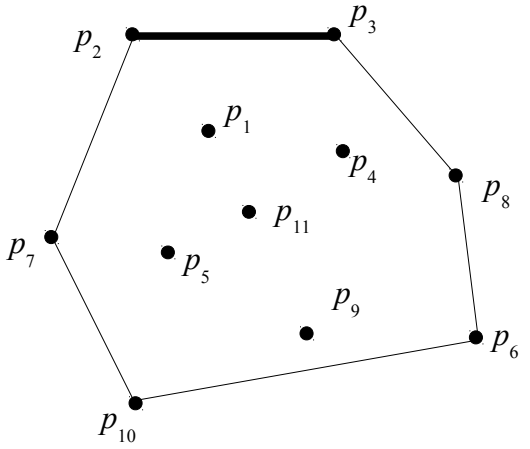
- Construction de l'algorithme

  La définition générale de l'enveloppe convexe est peu utile - intersection de toutes les ensembles convexes contenant P.

  - Il y en a une infinité

  Utilisons plutôt l'observation que l'enveloppe convexe dans ***ce cas ci*** est un polygone dont les sommets appartiennent à P.

- En particulier, prenons un coté du polygone

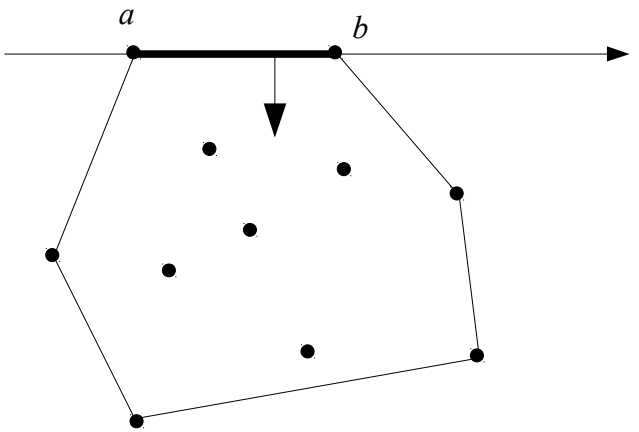


Enveloppe convexe d'un ensemble de points

Pour un coté du polygone:

- Les extrémités appartiennent à P.
- Si l'on trace une droite allant de a à b de telle façon que $CH(P)$ soit à « droite », alors tous les autres points de P sont aussi à droite.
- Si l'on prend n'importe quelle droite orientée ab ET que tous les autres points de P sont à droite, alors ab fait partie de l'enveloppe convexe $CH(P)$.

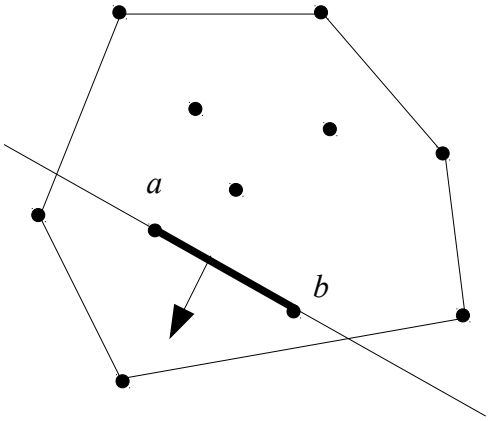
  On peut alors écrite un *algorithme*.



Enveloppe convexe d'un ensemble de points

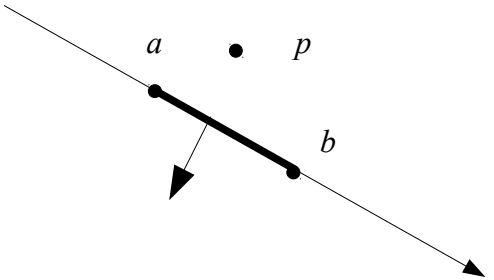
Algorithme:

EnveloppeConvexeLent(P, L)
Entrée : un ensemble de points P dans le plan
Sortie : une liste ordonnée L des points de CH(P) dans le sens horaire
{
 $E = \emptyset$
 Pour toutes les paires $(a, b) \in P \times P$ avec $a \neq b$
 {
  valide=vrai
  Pour tous les points $p \in P$, $p \neq a$ et $p \neq b$
  {
   Si p est à gauche de ab alors *valide=faux*
  }
  Si (*valide=vrai*) alors ajoute (a,b) à E.
 }
 Construire la liste ordonnée L à partir de E.
}



Enveloppe convexe d'un ensemble de points

- Explications
  - L'opération



...

Si p est à gauche de ab alors *valide=faux*

...

est un ***prédicat*** : Il s'agit ici d'une opération de base nécessaire pour le déroulement de l'algorithme. Dans la suite du cours, on suppose de telles opération accessibles. Ici il est évident que l'opération peut être effectuée en ***temps constant*** (indépendant de n)

- Signifie que le comportement asymptotique (lorsque n est grand) n'est pas affecté.

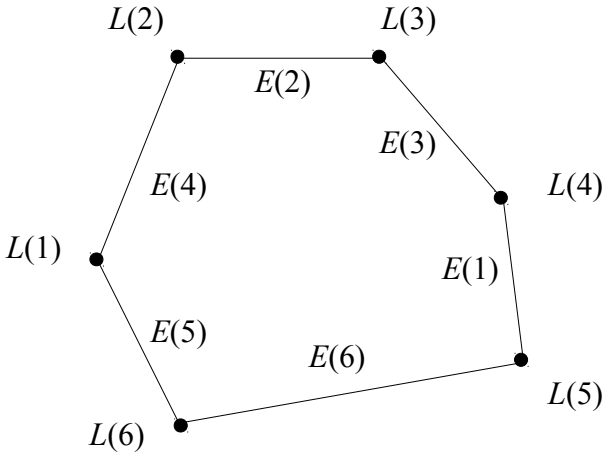
Enveloppe convexe d'un ensemble de points

- Explications

...

Construire la liste L à partir de E.

…



est une opération non triviale.
Voici un exemple d'implémentation :

ConstruitListeOrdonnée(E, L)
Entrée : liste désordonnée d'arêtes E
Sortie : une liste ordonnée L des points de $CH(P)$ dans le sens horaire
{
Prend le premier élément de E : $E(1)$
Ajoute le départ de E(1) , puis l'arrivée dans L ; Retire $E(1)$ de la liste E.
Tant que E contient plus d'un élément
{
Trouve l'élément $E(i)$ dont le départ est égal au dernier élément de L.
Ajoute l'arrivée de E(i) dans L ; retire E(i) de la liste E ;
}
}

Le cout est proportionnel à n^2 , mais on peut aisément améliorer cela en $n \log n$ par un classement adéquat.

Enveloppe convexe d'un ensemble de points

Analyse de la complexité de l'algorithme

C'est assez aisé ...

On on teste $n^2 - n$ paires de points

- Pour chaque paire, on vérifie le prédicat avec les $n - 2$ autres points.

En tout, il y a donc $(n^2 - n)(n - 2) = n^3 - 3n^2 + 2n$ tests.

Lorsque n est grand, le terme en n^3 domine. On dit que l'algorithme prend un temps en $\mathrm{O}(n^3)$ pour s'exécuter.

Nous n'avons pas tenu compte du tri final. Dans une implémentation la plus stupide, celui ci prend un temps en $\mathrm{O}(n^2)$ pour s'exécuter.

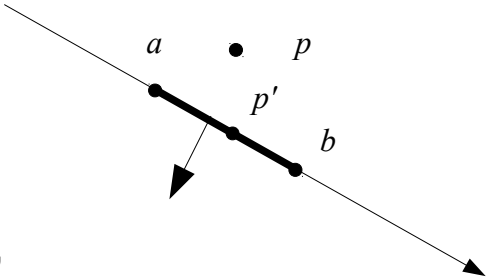
- Cela n'affecte pas la complexité totale qui est toujours en $\mathrm{O}(n^3)$.

Un algorithme en $\mathrm{O}(n^3)$ est généralement beaucoup trop gourmand pour être d'une quelconque utilité pratique si n est grand.

Enveloppe convexe d'un ensemble de points

Analyse plus approfondie de l'algorithme

- Revenons sur le prédicat utilisé :
  Un point n'est pas toujours *à droite* ou *à gauche* de la ligne. Il peut être *sur* la ligne.
  - Il s'agit d'un cas ***dégénéré*** – autrement dit, les points ne sont pas en position générale, certains sont colinéaires.

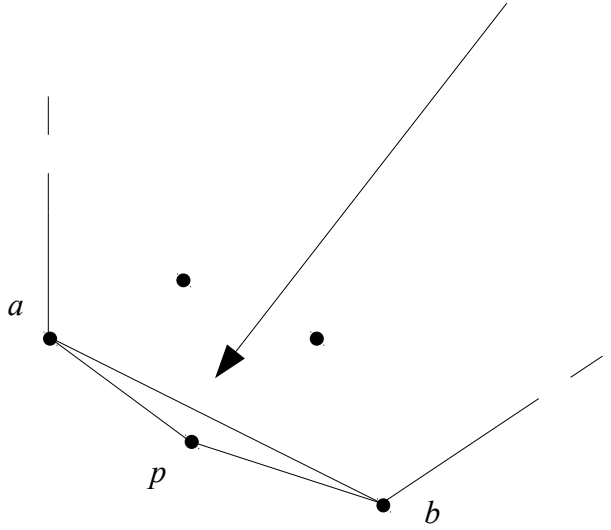


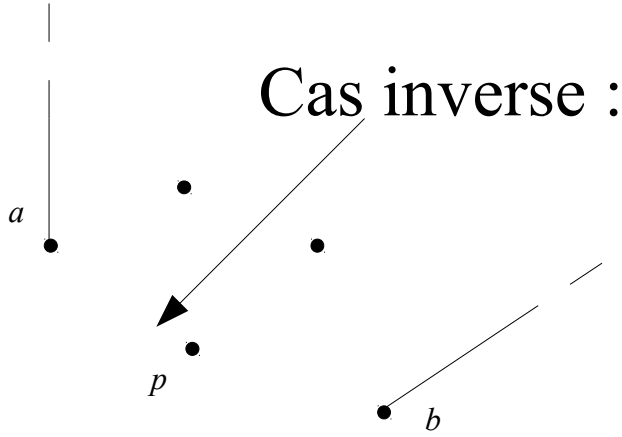
Reformulons le test : un segment orienté ab est un segment de CH(P) ssi tous les autres points de P sont strictement à droite de ab, ou si ils sont sur le segment ouvert ab.

Enveloppe convexe d'un ensemble de points

Analyse plus approfondie de l'algorithme

- On a supposé que le test pouvait être fait de façon exacte.
  Avec des coordonnées en virgule flottante, ce n'est pas le cas.
  Prenons le cas de trois points a,b et p presque colinéaires.
  L'algorithme va tester entre autres (a,b), (a,p) et (p,b).
  Il est possible que des erreurs d'arrondi impliquent que le test indique **à la fois** que p soit à droite de (a,b) ET que b soit à droite de (a,p) ET que a soit à droite de (p,b).
  - C'est bien entendu géométriquement impossible...
  - Cela donne un résultat incohérent





Enveloppe convexe d'un ensemble de points

- Premier algorithme
  - Calcule l'enveloppe convexe correctement
    - En supposent que l'on peut évaluer le prédicat de façon **exacte** !!
  - Est lent - $O(n^3)$
  - Traite les cas dégénérés de façon maladroite
  - Est non robuste

  On peut faire bien mieux.

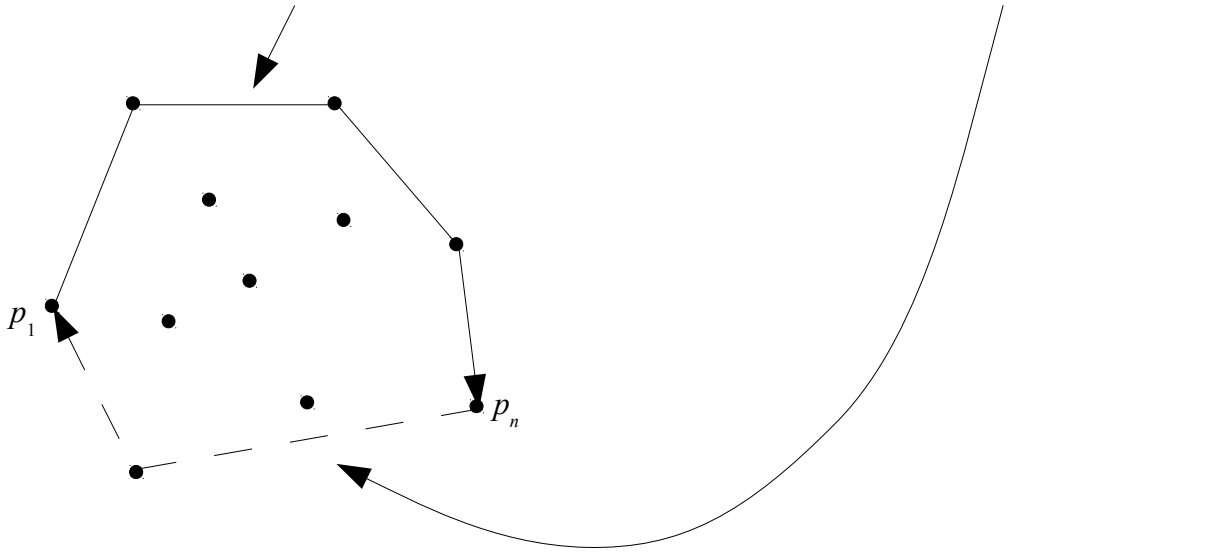
Enveloppe convexe d'un ensemble de points

- Algorithme de Graham

Graham, R.L. (1972). An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters* 1, 132-133

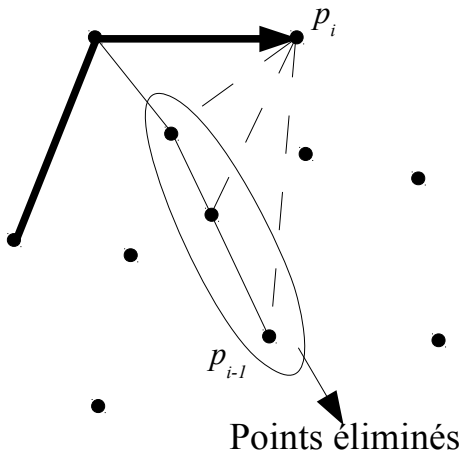
Enveloppe convexe d'un ensemble de points

- Algorithme
  - On classe les points de droite à gauche
  - Ajout des points de P un par un, tout en maintenant la solution à jour après chaque addition
  - On calcule l'enveloppe convexe en deux passes
    - Partie supérieure L_{sup} puis partie inférieure L_{inf}



Enveloppe convexe d'un ensemble de points

- Le point délicat est la mise à jour de l'enveloppe convexe après chaque ajout de point p_i.
  - Partant de $p_1, \ldots, p_{i-1}$ on veut obtenir $p_1, \ldots, p_i$
  - En parcourant un polygone convexe dans le sens horaire, on effectue toujours un « virage à droite » en passant par chaque sommet.
    - p_i est forcément le dernier point de L_{sup}, donc on l'insère
    - On teste les 3 derniers points de L_{sup},
      - si le virage est « à droite », fin
      - sinon on efface l'avant dernier point, et on recommence si l'on a au moins trois points dans Lsup
    - On fait la même chose pour la partie inférieure L_{inf}



Enveloppe convexe d'un ensemble de points

Algorithme :

EnveloppeConvexe(P, L)
Entrée : un ensemble de points P dans le plan
Sortie : une liste ordonnée L des points de CH(P) dans le sens horaire
{
Classer les point P en fonction de x croissant
Inserer les points p_1 et p_2 dans cet ordre dans la liste L_{sup}
Pour i=3 à n
{
Ajouter pi à Lsup
Tant que
Lsup contient plus de 2 points
ET que les 3 derniers points ne forment pas un virage à droite
{
Effacer l'avant dernier point de Lsup
}
}

>>>suite

Enveloppe convexe d'un ensemble de points

Algorithme (suite):

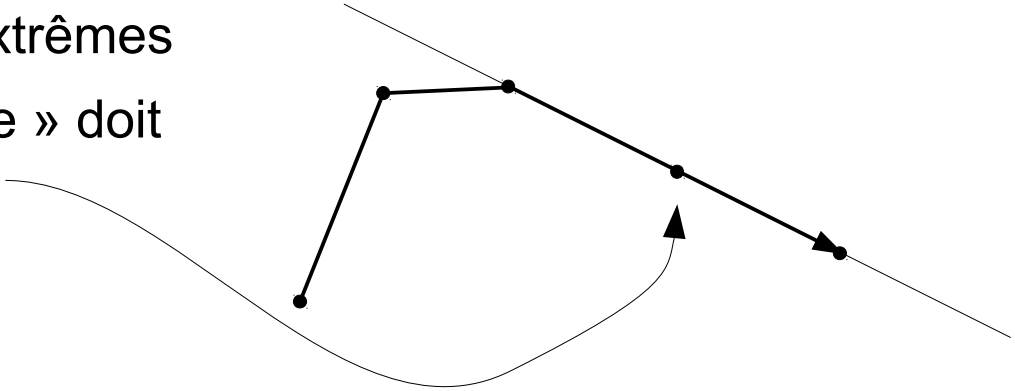
Inserer les points p_n et p_{n-1} dans cet ordre dans la liste L_{inf}
Pour i=n–2 à 1
{
  Ajouter p_i à L_{inf}
  Tant que
        L_{inf} contient plus de 2 points
        ET que les 3 derniers points ne forment pas un virage à droite
  {
    Effacer l'avant dernier point de L_{inf}
  }
}
Effacer le permier et le dernier point de L_{inf}
Concaténer L_{sup} puis L_{inf} dans cet ordre dans L.
}

Enveloppe convexe d'un ensemble de points

- Que se passe-t-il si deux points ont les mêmes coordonnées x ?
  - Il faut trier ceux qui ont la même coordonnée x par rapport à y !

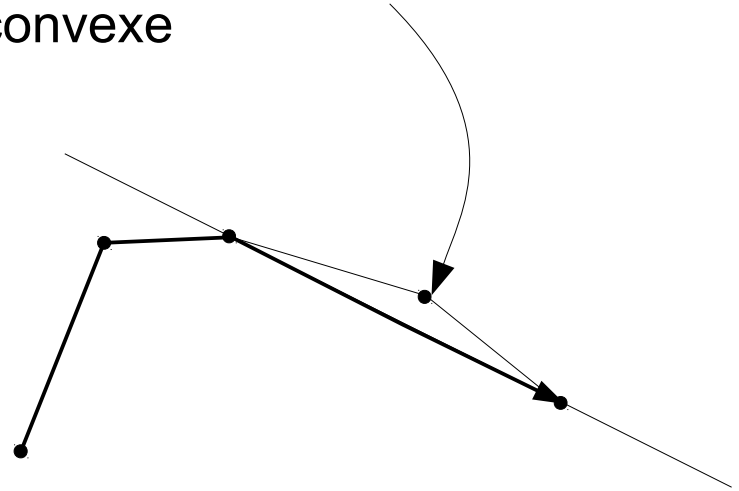
    C'est un tri *lexicographique.*
- Que se passe-t-il si trois points sont colinéaire ?
  - On ne garde que les points extrêmes

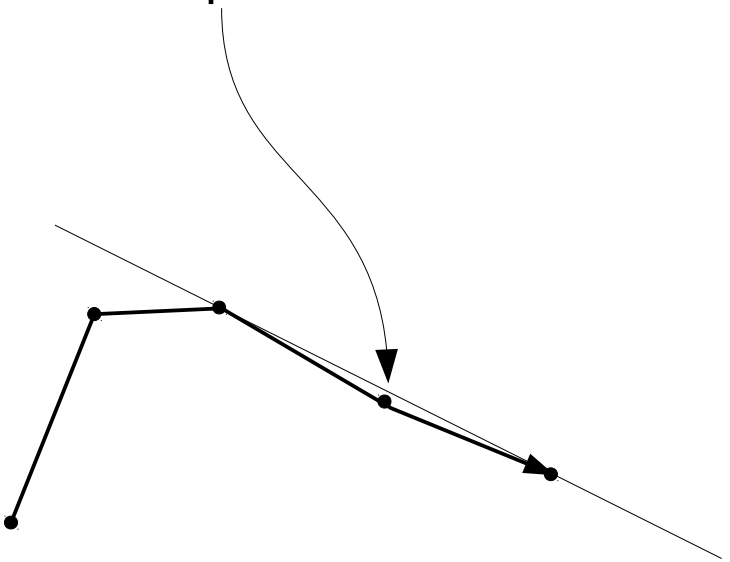
    Le test « est un virage à droite » doit être considéré comme **faux**.
  - Moyennant ces modifications, l'algorithme est correct.



Enveloppe convexe d'un ensemble de points

- Que se passe-t-il si des erreurs d'arrondi altèrent le test « est un virage à droite » ?
  - On risque de ne pas effacer un sommet qui est un peu à l'intérieur de l'enveloppe convexe
  - Ou d'effacer un sommet qui devrait se trouver sur l'enveloppe convexe



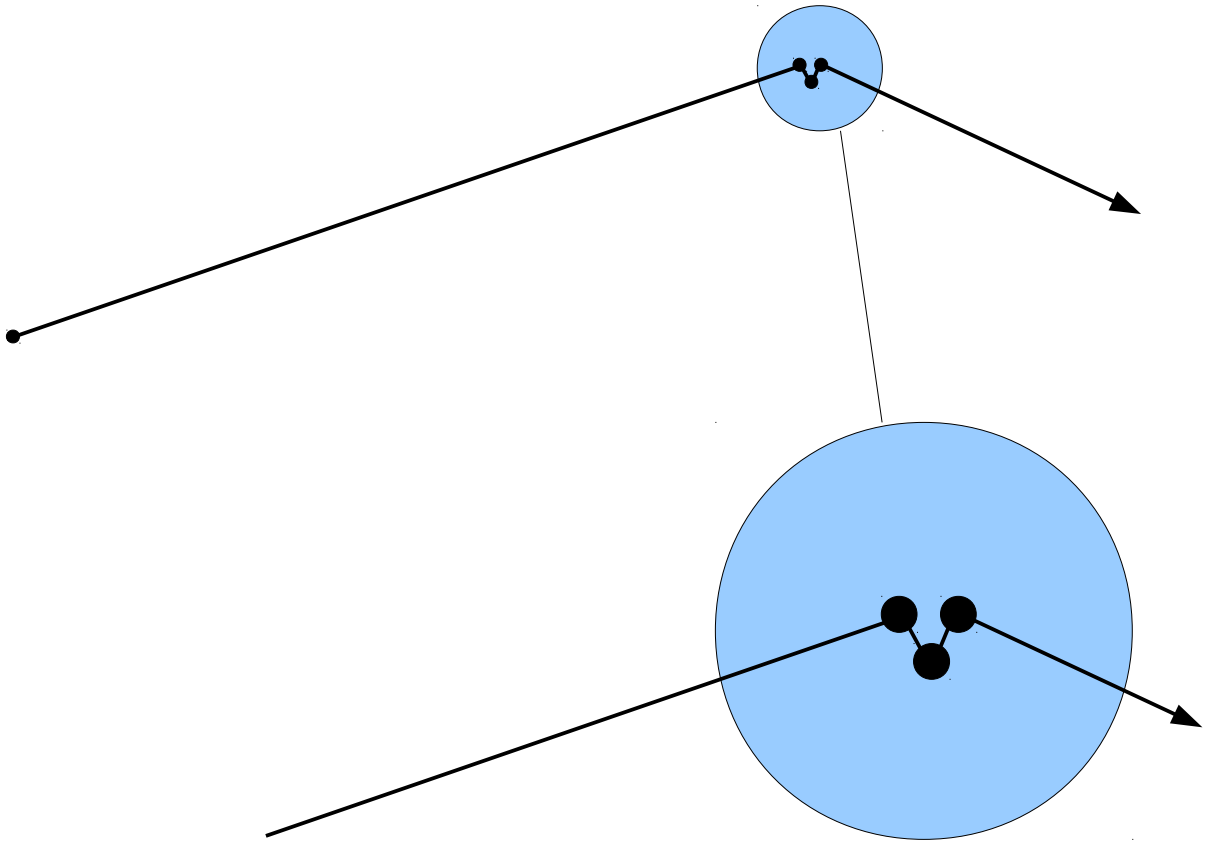


  - Dans tous les cas, l'enveloppe convexe calculée reste cohérente : un polygone fermé, décrit dans le sens horaire, pour lequel les « virages » sont toujours à droite (du point de vue de l'ordinateur)

Enveloppe convexe d'un ensemble de points

- Il est aussi possible de confondre un virage serré à gauche avec un virage à droite si trois points sont très proches

  Solution : éviter cela et fusionner les points trop proches (par arrondi)



Enveloppe convexe d'un ensemble de points

- Analyse de l'algorithme et complexité
  - Pour L_{sup} : (arguments pour L_{inf} identiques)
    - La boucle principale s'exécute n-3 fois
    - Le test (prédicat) est effectué un nombre inconnu de fois à chaque itération de la boucle principale
    - Toutefois, le nombre TOTAL de noeuds retirés de L_{sup} pour toutes les itérations ne peut excéder n !
    - En conséquence, cette partie de l'algorithme s'exécute avec un temps en $O(n)$.
  - La concaténation des deux liste s'effectue aussi en $O(n)$
  - Le tri en début d'algorithme se fait habituellement en $O(n \log n)$
  - En conséquence, la complexité est $O(n \log n)$ à cause du tri prépondérant.

Enveloppe convexe d'un ensemble de points

- Construction d'un algorithme géométrique
  - D'abord comprendre la géométrie du problème en ignorant tout ce qui peut rendre cette étape plus complexe (cas particuliers, dégénérés, etc...)
  - Dans une seconde étape , intégrer les cas dégénérés. Il est utile des les intégrer directement dans l'algorithme plutôt que de faire des cas particuliers.
  - Enfin, vient l'implémentation. Il faut coder ou disposer des prédicats robustes, et éventuellement modifier l'algorithme afin qu'il réponde correctement en cas de prédicats basés sur des opérations en virgule flottante.
    - Cf . Second algorithme de l'enveloppe convexe.

Enveloppe convexe d'un ensemble de points

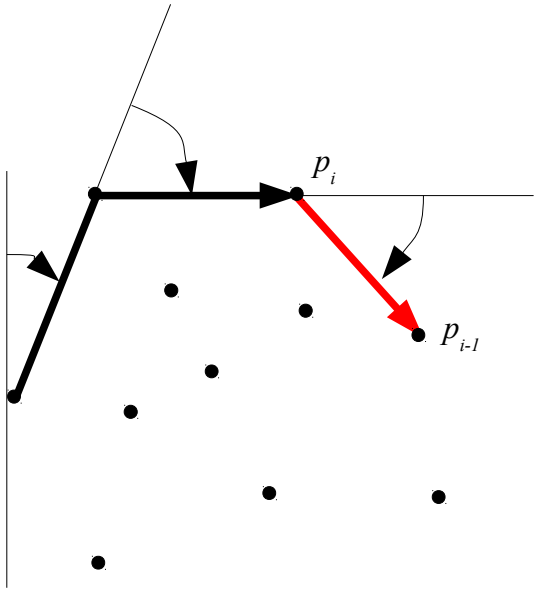
- Exercice : analyse de l'algorithme de Jarvis

  Jarvis, R. A. (1973). "On the identification of the convex hull of a finite set of points in the plane". *Information Processing Letters* 2: 18–21.

  - Complexité ?
  - Nature (incrémental ou pas, randomisé ...)
  - Robustesse ?

Enveloppe convexe d'un ensemble de points

- Algorithme : depuis Pi, parcourir les autres sommets et rechercher celui dont l'angle est le plus petit.

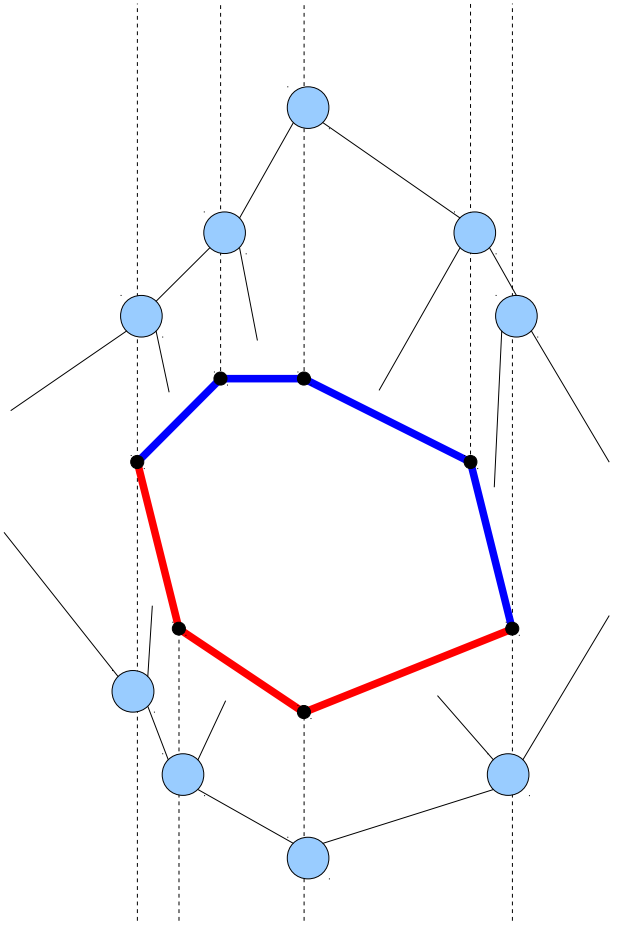


Enveloppe convexe d'un ensemble de points

- Algorithme *incrémental* ?
  - Cela signifie que l'on souhaite pouvoir insérer un nouveau point P_i , connaissant l'enveloppe convexe des i-1 points précédents.
    - Première étape vers les algorithme *dynamiques*, permettant également d'éliminer un point à tout moment...
  - Il existe une implémentation performante :
    - Idée : parvenir à localiser le point inséré rapidement, et modifier l'enveloppe convexe de l'étape i-1 le cas échéant.

Enveloppe convexe d'un ensemble de points

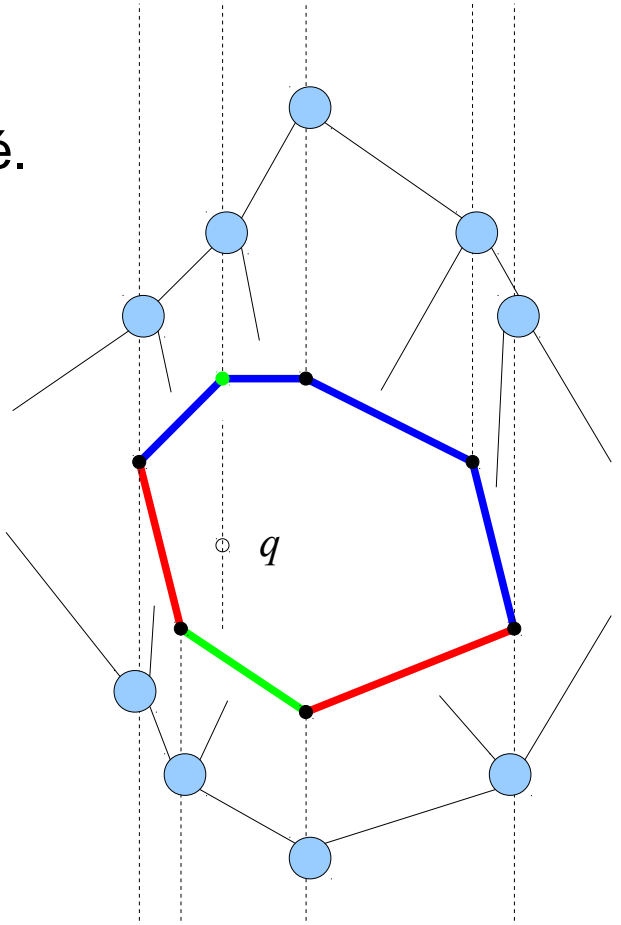
- On stocke les points sur les parties supérieure et inférieure de l'enveloppe convexe dans deux arbres binaires balancés...



Enveloppe convexe d'un ensemble de points

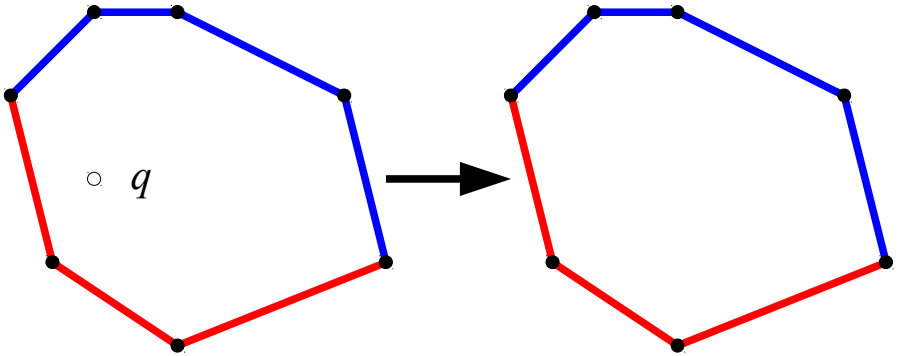
- On stocke les points sur les parties supérieure et inférieure de l'enveloppe convexe dans deux arbres binaires balancés...
  - Il est facile de trouver le segment/point immédiatement inf et sup au point q inséré.

    4 cas :
    - Pas existence de ces éléments, OUT
    - Si q est sur un de ces éléments, ON
    - Si q est au dessus de l'élément inf et en dessous de l'élément sup : IN
    - Sinon : OUT
  - En fonction de ces cas, on doit insérer ou non le point q dans l'enveloppe convexe.

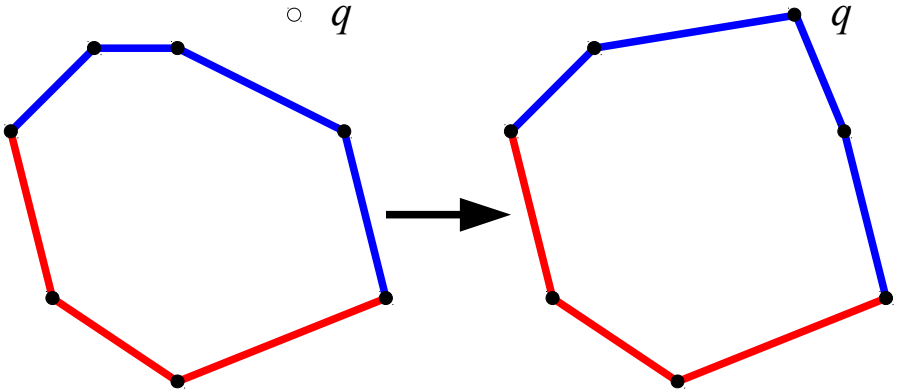


Enveloppe convexe d'un ensemble de points

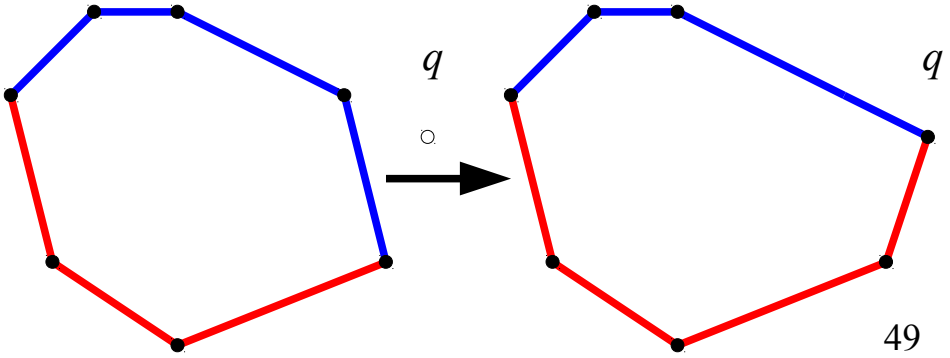
- En fonction de ces cas ...
  - Cas IN : point q écarté.



  - Cas OUT et au dessus : insertion dans la chaine supérieure
  - Cas OUT et dessous : insertion dans la chaine inférieure



  - Cas OUT et Gauche ou Droite : insertion dans les deux chaines.



Enveloppe convexe d'un ensemble de points

- Insertion dans la chaîne (ici la chaîne supérieure)
  - Trouver l'arête e (ou le sommet v) dont l'extension supérieure contient q

    w est le sommet à la gauche de e ou v

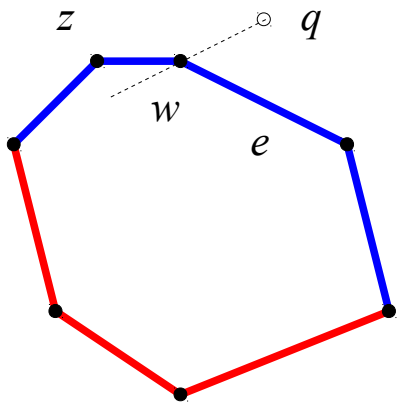
    z est le sommet à la gauche de w
  - Tant que $\text{orientation}(q,w,z)$ dans les sens des aiguilles d'une montre (ou colinéaire)

    Éliminer w

    $w=z$

    z=voisin à gauche de w
  - Faire la même chose de l'autre côté...
  - Ajouter q dans la chaîne.

    (NB. Pour la lisibilité, je ne parle pas des cas particuliers aux extrémités...)



Enveloppe convexe d'un ensemble de points

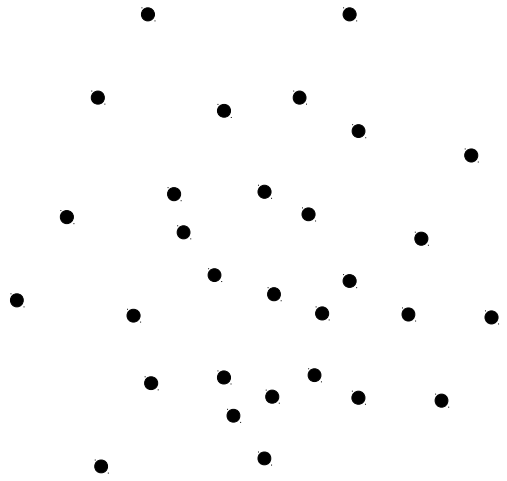
- Analyse de complexité (exercice)

Enveloppe convexe d'un ensemble de points

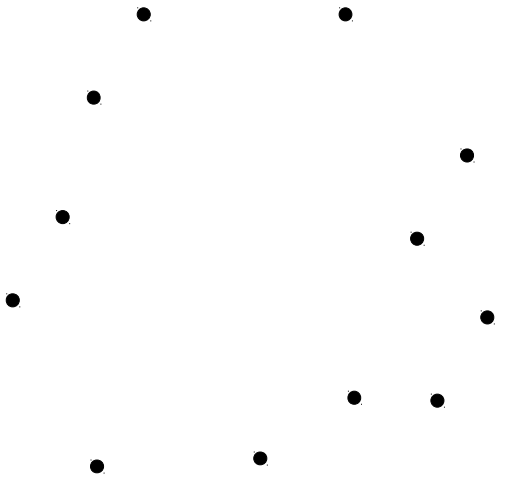
- Outils : arbre binaire balancé (p.ex. Red and Black tree)
- Implémentation disponible dans la STL en C++ …
- C'est une structure de données dynamique : l'insertion et la délétion sont en temps optimal (logarithmique...)

Enveloppe convexe d'un ensemble de points

- Que se passe-t-il si les données ont une structure particulière ?
  - Il se peut que l'un des algorithmes soit meilleur que les autres alors qu'il est moins bon dans le cas le pire
  - Supposons ici que les points en entrée sont régulièrement répartis dans une zone donnée...



On est certain d'avoir ce cas



On ne rencontre pas ce cas

Structures de données génériques

Structures de données

- Quelques structures génériques et algorithmes utiles (en C++)
  - Vecteurs (vector) → remplace les tableaux en "C"
    - Rangement contigu en mémoire
    - Accès aléatoire en $O(1)$
    - Insertion/délétion en $O(n)$ sauf à la fin : $O(1)$ si réserve mémoire.
  - Liste chainées / piles (list)
    - Insertion au début, à la fin, au milieu (pour les listes) en O(1)
    - Accès séquentiel en $O(1)$ / aléatoire en $O(n)$
  - Ensembles (set) / applications (map)
    - Accès séquentiel en $O(1)$ / Accès aléatoire en $O(\log n)$
    - Insertion / délétion en $O(\log n)$
    - Recherche en $O(\log n)$

Structures de données

- Librairie STL de conteneurs en C++
  - Générique
  - Efficace
  - Bug-free
  - Disponible
  - Standard

Vecteurs (vector)

```
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
  vector<int> myvector (3,100) ; // initialisation 3 entiers = 100
  vector<int>::iterator it ;    // iterateur == pointeur

  it = myvector.begin();
  it = myvector.insert ( it , 200 );
  myvector.insert (it,2,300);
  it = myvector.begin() ; // "it" invalide, obtiens un nouveau

  vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());

  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);

  cout << "myvector contains:";
  for (int i=0; i<myvector.size(); i++)
    cout << " " << myvector[i] ;
  cout << endl ;
//  for (it=myvector.begin(); it<myvector.end(); it++) // alternative
//    cout << " " << *it ;
  return 0;
}
```

→ myvector contains: 501 502 503 300 300 400 400 200 100 100 100

Listes (list)

```
// inserting into a list
#include <iostream>
#include <list>
#include <vector>
using namespace std;
int main ()
{
  list<int> mylist;
  list<int>::iterator it;
  // set some initial values:
  for (int i=1; i<=5; i++) mylist.push_back(i); // 1 2 3 4 5
  it = mylist.begin();
  ++it;       // it points now to number 2           ^
  mylist.insert (it,10);                        // 1 10 2 3 4 5
  // "it" still points to number 2                     ^
  mylist.insert (it,2,20);                      // 1 10 20 20 2 3 4 5
  --it;       // it points now to the second 20          ^
  vector<int> myvector (2,30);
  mylist.insert (it,myvector.begin(),myvector.end());
                                                // 1 10 20 30 30 20 2 3 4 5
                                                //               ^
  cout << "mylist contains:";
  for (it=mylist.begin(); it!=mylist.end(); it++)
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

→ `mylist contains: 1 10 20 30 30 20 2 3 4 5`

Ensembles (set)

```
// set::insert
#include <iostream>
#include <set>
using namespace std;
int main ()
{
  set<int> myset;
  set<int>::iterator it;
  pair<set<int>::iterator,bool> ret;
  // set some initial values:
  for (int i=1; i<=5; i++) myset.insert(i*10);    // set: 10 20 30 40 50
  ret = myset.insert(20);               // no new element inserted
  if (ret.second==false) it=ret.first;  // "it" now points to element 20
  myset.insert (it,25);                 // max efficiency inserting
  myset.insert (it,24);                 // max efficiency inserting
  myset.insert (it,26);                 // no max efficiency inserting
  int myints[]= {5,10,15};              // 10 already in set, not inserted
  myset.insert (myints,myints+3);
  cout << "myset contains:";
  for (it=myset.begin(); it!=myset.end(); it++)
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

```
myset contains: 5 10 15 20 24 25 26 30 40 50
```

Application (map)

```
// map::insert
#include <iostream>
#include <map>
using namespace std;
int main ()
{
  map<char,int> mymap;
  map<char,int>::iterator it;
  pair<map<char,int>::iterator,bool> ret;
  // first insert function version (single parameter):
  mymap.insert ( pair<char,int>('a',100) );
  mymap.insert ( pair<char,int>('z',200) );
  ret=mymap.insert (pair<char,int>('z',500) );
  if (ret.second==false)
  {
    cout << "element 'z' already existed";
    cout << " with a value of " << ret.first->second << endl;
  }
  // second insert function version (with hint position):
  it=mymap.begin();
  mymap.insert (it, pair<char,int>('b',300));  // max efficiency inserting
  mymap.insert (it, pair<char,int>('c',400));  // no max efficiency inserting
  // third insert function version (range insertion):
  map<char,int> anothermap;
  anothermap.insert(mymap.begin(),mymap.find('c'));
  // showing contents:
  cout << "mymap contains:\n";
  for ( it=mymap.begin() ; it != mymap.end(); it++ )
    cout << (*it).first << " => " << (*it).second << endl;
  cout << "anothermap contains:\n";
  for ( it=anothermap.begin() ; it != anothermap.end(); it++ )
    cout << (*it).first << " => " << (*it).second << endl;
  return 0;
}
```

Algorithmes (tri)

```
// sort algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
bool myfunction (int i,int j) { return (i<j); }
struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;
int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33
  vector<int>::iterator it;
  // using default comparison (operator <):
  sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26 80 53 33
  // using function as comp
  sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)
  // using object as comp
  sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)
  // print out content:
  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

```
myvector contains: 12 26 32 33 45 53 71 80
```

Algorithmes (recherche)

```
// find example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main () {
  int myints[] = { 10, 20, 30 ,40 };
  int * p;
  // pointer to array element:
  p = find(myints,myints+4,30);
  ++p;
  cout << "The element following 30 is " << *p << endl;
  vector<int> myvector (myints,myints+4);
  vector<int>::iterator it;
  // iterator to vector element:
  it = find (myvector.begin(), myvector.end(), 30);
  ++it;
  cout << "The element following 30 is " << *it << endl;
  return 0;
}
```

→

```
The element following 30 is 40
The element following 30 is 40
```

Ici, recherche linéaire en O(n)

Algorithmes (recherche)

```
// binary_search example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
bool myfunction (int i,int j) { return (i<j); }
int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3 2 1
  // using default comparison:
  sort (v.begin(), v.end());
  cout << "looking for a 3... ";
  if (binary_search (v.begin(), v.end(), 3))
    cout << "found!\n"; else cout << "not found.\n";
  // using myfunction as comp:
  sort (v.begin(), v.end(), myfunction);
  cout << "looking for a 6... ";
  if (binary_search (v.begin(), v.end(), 6, myfunction))
    cout << "found!\n"; else cout << "not found.\n";
  return 0;
}
```

Ici, recherche dans un conteneur déjà trié = en O(log n) si accès aléatoire disponible (vector)

→

```
looking for a 3... found!
looking for a 6... not found.
```

Tri lexicographique

```
#include <iostream>
#include <set>
#include <cmath>
using namespace std;

struct point2
{
  double x;
  double y;
};

struct point_comparison
{
  bool operator ()(const point2 &i,const point2 &j) const
  {
   if (i.x<j.x) return true; if (i.x>j.x) return false;
   if (i.y<j.y) return true; return false;
  }
};

int main()
{
  set<point2,point_comparison> myset;
  set<point2,point_comparison>::iterator it,itlow,itup;
  for (int i=-3; i<=3; i++)
  {
    point2 pt ; pt.x=sin(i); pt.y=cos(i); myset.insert(pt);
    pt.y=-cos(i); myset.insert(pt);
  }
  cout << "myset contains:" << endl;
  for (it=myset.begin(); it!=myset.end(); it++)
    cout << "x=" << it->x <<  " y=" << it->y << endl;
  point2 pt;
  pt.x=0; pt.y=1;
  itlow=myset.lower_bound(pt);
  itup=myset.upper_bound(pt);
  cout << "point :" << endl;
  cout << "x=" << pt.x <<  " y=" << pt.y << endl;
  cout << "lower bound - first element that does not \ compare
strictly less (includes equal elements) : " << \ endl;
  if (itlow!=myset.end())
    cout <<"x="<< itlow->x <<" y="<< itlow->y << endl;
  else
    cout << "no such element" << endl;
  cout << "upper bound -  first element that does \ compare
strictly greather" << endl;
  if (itup!=myset.end())
    cout <<"x="<< itup->x <<" y="<< itup->y << endl;
  else
    cout << "no such element" << endl;
  return 0;
}
```

Tri lexicographique

```
bechet@cg-eb:cours1$ ./test_lexico
myset contains:
x=-0.909297 y=-0.416147
x=-0.909297 y=0.416147
x=-0.841471 y=-0.540302
x=-0.841471 y=0.540302
x=-0.14112 y=-0.989992
x=-0.14112 y=0.989992
x=0 y=-1
x=0 y=1
x=0.14112 y=-0.989992
x=0.14112 y=0.989992
x=0.841471 y=-0.540302
x=0.841471 y=0.540302
x=0.909297 y=-0.416147
x=0.909297 y=0.416147
point :
x=0 y=1
lower bound -  first element that does not compare strictly less (includes equal elements) :
x=0 y=1
upper bound -  first element that does compare strictly greather
x=0.14112 y=-0.989992
```

Computational geometry

Outil de visualisation

- Visualisateur élémentaire basé sur VTK (http://www.vtk.org/) ou FLTK, disponible sur le site web du cours

```
#include "nutil.h"
#include "vtkdisplay.h"
int main(void)
{
  data_container data;
  vtkdisplay display(color(0,0,0),(char*)"Test VTK");
  npoint p1(0,0,0),p2(1,0,0),p3(1,1,0),p4(0,1,0),
         p5(0,0,1),p6(1,0,1),p7(1,1,1),p8(0,1,1);
  properties ptr=data.getproppoints();
  ptr.c=color(100,200,60);ptr.pointsize=15;
  data.setproppoints(ptr);
  data.add_point(p1);data.add_point(p2);
  data.add_point(p3);data.add_point(p4);
  ptr.c=color(255,0,255);ptr.pointsize=7;
  data.setproppoints(ptr);
  data.add_point(p5);data.add_point(p6);
  data.add_point(p7);data.add_point(p8);
  data.add_point(npoint(0.5,0.5,0.5));
  ptr=data.getproptriangles();
  ptr.c=color(255,0,0);  ptr.edgeon=true;
  ptr.edgecolor=color(255,255,255);ptr.edgethickness=5;
  data.setproptriangles(ptr);
    triangle tr;
  tr.pts[0]= npoint3(0,0.5,0.5);
  tr.pts[1]= npoint3(0.5,0,0.5);
  tr.pts[2]= npoint3(0.5,0.5,0);
  data.add_triangle(tr);
  ptr.c=color(0,0,255); ptr.edgeon=false;
  data.setproptriangles(ptr);
  tr.pts[0]= npoint3(0,-0.5,-0.5);
  tr.pts[1]= npoint3(-0.5,0,-0.5);
  tr.pts[2]= npoint3(-0.5,-0.5,0);
  data.add_triangle(tr);

  line l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11,l12;
  l1.pts[0]=p1; l1.pts[1]=p2; l2.pts[0]=p2; l2.pts[1]=p3;
  l3.pts[0]=p3; l3.pts[1]=p4; l4.pts[0]=p4; l4.pts[1]=p1;
  l5.pts[0]=p5; l5.pts[1]=p6; l6.pts[0]=p6; l6.pts[1]=p7;
  l7.pts[0]=p7; l7.pts[1]=p8; l8.pts[0]=p8; l8.pts[1]=p5;
  l9.pts[0]=p1; l9.pts[1]=p5; l10.pts[0]=p2; l10.pts[1]=p6;
  l11.pts[0]=p3; l11.pts[1]=p7; l12.pts[0]=p4; l12.pts[1]=p8;
  ptr=data.getproplines();
  ptr.c=color(0,125,0); ptr.thickness=5;
  data.setproplines(ptr);
  data.add_line(l1); data.add_line(l2);
  data.add_line(l3); data.add_line(l4);
  ptr.c=color(255,0,0); data.setproplines(ptr);
  data.add_line(l5); data.add_line(l6);
  ptr.thickness=10; data.setproplines(ptr);
  data.add_line(l7); data.add_line(l8);
  ptr.thickness=15; data.setproplines(ptr);
  data.add_line(l9); data.add_line(l10);
  ptr.thickness=2; data.setproplines(ptr);
  data.add_line(l11); data.add_line(l12);
```

Outil de visualisation

- Visualisateur élémentaire basé sur VTK (http://www.vtk.org/) ou FLTK, disponible sur le site web du cours

```
  quad qu;
  qu.pts[0]=p1;  qu.pts[1]=p6;
  qu.pts[2]=p7;  qu.pts[3]=p4;
  ptr=data.getpropquads();
  ptr.c=color(0,255,0,128); // semi-transparent
  ptr.edgeon=false;
  data.setpropquads(ptr);
  data.add_quad(qu);
  qu.pts[0]=p1;  qu.pts[1]=p2;
  qu.pts[2]=p3;  qu.pts[3]=p4;
  ptr.c=color(100,100,100,255);
  data.setpropquads(ptr);
  data.add_quad(qu);

  npoint p10(2,2,2);
  point pp;
  pp.pts=p10;
  char mess[255];
  sprintf(mess,"Hello");
  pp.info=mess;
  color cc=color(0,255,255);
  std::pair<point,color> tmp(pp,cc);
  data.add_text(0,tmp);
  display.init_data(data);
  display.display(true);
  return 0;
}
```

Outil de visualisation