Géométrie Algorithmique

Cours réalisé par Stéphane Bessy

Pour le second semestre du M1 Informatique

Version 1.2

Relecteurs
- Erik Martin-Dorel (M1 Maths-Info)
- Boris Massaré (M1 Maths-Info)
- Pierre Noyaret (M1 Info)

Année universitaire 2007-2008

Table des matières

Table des figures

Chapitre 1

Intersection dans un ensemble de segments

1.1 Position du problème

Soit $(s_1, \ldots, s_n)$ un ensemble de n segments dans le plan. Chaque segment est donné par ses deux extrémités :

$$s_i = [p_i, q_i] \quad \forall i, 1 \leqslant i \leqslant n \quad \text{avec par convention : } p_i \text{ est à gauche de } q_i.$$

On fait l'hypothèse qu'il n'existe aucun segment vertical.

Question : Existe-t-il deux segments s_i et s_j qui s'intersectent (avec $i \neq j$) ?

En T.D., nous avons pu déterminer si deux segments s'intersectent en $\Theta(1)$. Ainsi en testant tous les couples, on obtient un algorithme en $\Theta(n^2)$. Nous allons présenter un algorithme en $\Theta(n \log n)$ qui utilise une technique de balayage du plan avec un faisceau vertical, de la gauche vers la droite.

1.2 Tri de segments

À un instant t, nous comparons les segments coupés par le faisceau. Ce qui nous donne un ordre sur les segments :

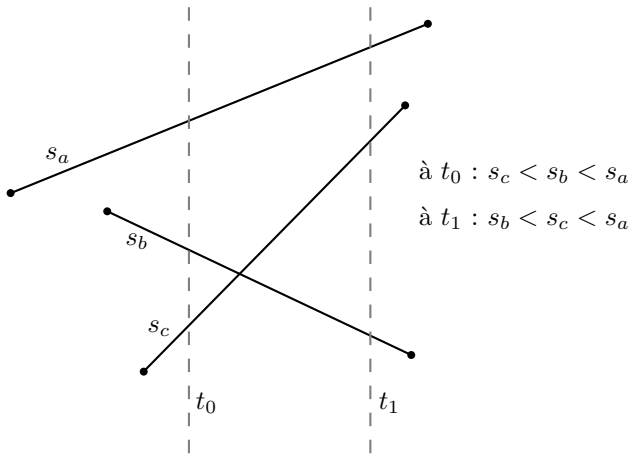

FIG. 1.1 – Ordre sur les segments.

Au temps t, $s < s'$ si le point d'intersection de s avec le faisceau a une ordonnée inférieure à celle du point d'intersection de s' avec le faisceau.

Remarque : Une intersection entre deux segments est repérée par une permutation de ces deux segments dans les ordres associés à deux dates différentes.

Nous allons gérer deux ensembles de données :
– un ordre total qui évolue ;
– un ensemble fini de dates par lesquelles nous nous intéresserons à l'ordre précédent (échéancier).

L'échéancier contiendra les $2n$ abscisses des segments triés par ordre croissant.

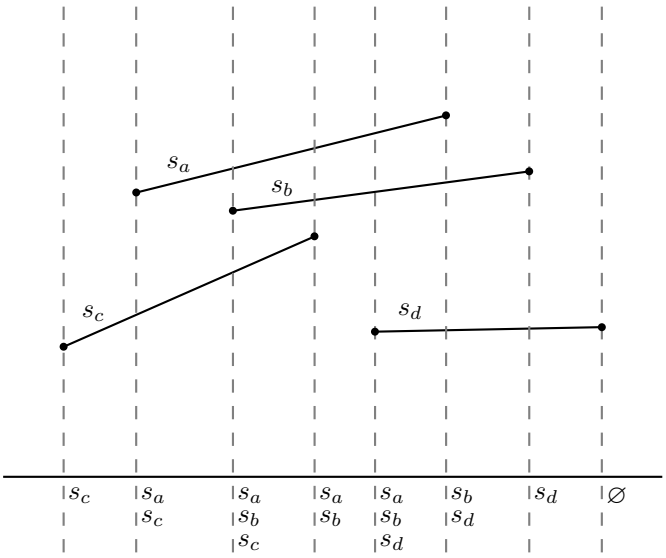

FIG. 1.2 – Exemple.

1.2.1 Primitives

(T désigne l'ordre sur les segments.)

– `Insérer(`s`,`T`)` : insère s dans T (*) ;
– `Supprimer(`s`,`T`)` ;
– `AuDessus(`s`,`T`)` : retourne le successeur de s dans l'ordre T ;
– `AuDessous(`s`,`T`)` : retourne le prédécesseur de s dans l'ordre T.

(*) pour connaître la position d'un segment $s = [p, q]$ par rapport à $s' = [p', q']$ précédemment inséré :
si $\det\left(\overrightarrow{p'q'}, \overrightarrow{p'p}\right) > 0$
alors s est au-dessus de s',
sinon s est au-dessous de s'.

1.2.2 Structures de données

	tableau	liste doublement chaînée	arbre binaire (**)
`Insérer`	$\Theta(n)$	$\Theta(n)$	Θ (hauteur)
`Supprimer`	$\Theta(n)$	$\Theta(1)$	Θ (hauteur)
`AuDessus`	$\Theta(1)$	$\Theta(1)$	Θ (hauteur)
`AuDessous`	$\Theta(1)$	$\Theta(1)$	Θ (hauteur)

(**) arbre binaire : valeur(fils gauche) $\leqslant$ valeur(père) $\leqslant$ valeur(fils droit).

– arbre Rouge et Noir : hauteur en $2\log n$;
– arbre AVL : hauteur en $\log n$.

1.3 L'algorithme

1.3.1 Principe

À l'insertion, nous testons s'il y a une intersection avec celui du dessus et celui du dessous :

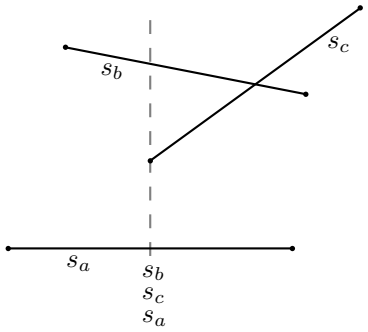

FIG. 1.3 – Insertion de s_c.

À la suppression, nous testons si les segments successeur et prédécesseur s'intersectent :

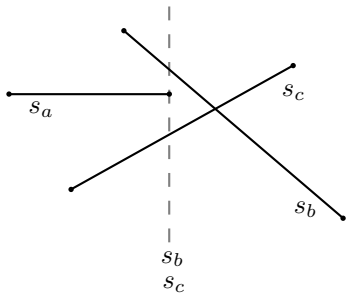

FIG. 1.4 – Suppression de s_a.

1.3.2 L'algorithme proprement dit

Algorithme 1 : Intersection de segments.

Données : Un ensemble de segments.
Sorties : VRAI si deux segments s'intersectent, sinon FAUX.

début
  $T \leftarrow \varnothing$
  Trier les abscisses des extrémités des segments (échéancier)
  pour chaque point r de l'échéancier **faire**
    si r est extrémité gauche d'un segment s **alors**
      `Insérer(`s,T`)`
      si `AuDessus(`s,T`)` et s s'intersectent **ou** `AuDessous(`s,T`)` et s s'intersectent **alors**
        retourner VRAI
    si r est extrémité droite d'un segment s **alors**
      si `AuDessus(`s,T`)` et `AuDessous(`s,T`)` s'intersectent **alors**
        retourner VRAI
      `Supprimer(`s,T`)`
  retourner FAUX
fin

1.3.3 Preuve de l'algorithme

Si l'algorithme renvoie VRAI : il existe bien deux segments qui s'intersectent !

Si l'algorithme renvoie FAUX : supposons qu'il existe deux segments qui s'intersectent, et notons I l'intersection la plus à gauche de l'ensemble des segments. S'il y a plus de deux segments qui contiennent I, on en choisit deux consécutifs dans l'ordre circulaire :

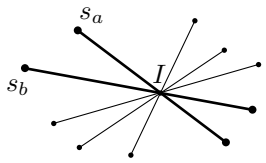

FIG. 1.5 – Intersection multiple.

Si à l'insertion de s_a au temps t, l'algorithme n'a pas détecté I, alors s_b n'est pas prédécesseur de s_a. Aucun des segments inclus entre s_a et s_b dans l'ordre (au temps t) n'intersecte s_a ou s_b car I est le point d'intersection le plus à gauche. Notons s_c celui de ces segments dont l'extrémité droite est la plus à droite :

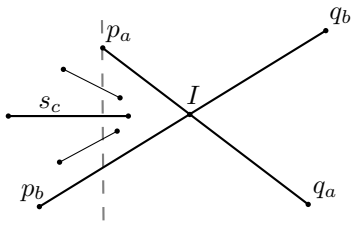

FIG. 1.6 – Schéma de la situation.

À la suppression de s_c, nous devons tester l'intersection entre s_a et s_b : l'algorithme trouve I ! □

1.3.4 Complexité

Bilan :
- tri de $2n$ points ;
- n appels à `Insérer` et `Supprimer` ;
- $2n$ appels à `AuDessous` et `AuDessus`.

D'où une complexité en :
$\Theta\left(n^2\right)$ avec des tableaux.
$\Theta\left(n \log n\right)$ avec des arbres de recherche de hauteur $\log n$. □

1.3.5 Remarque

Un tel algorithme peut être utilisé pour le recouvrement d'objets :

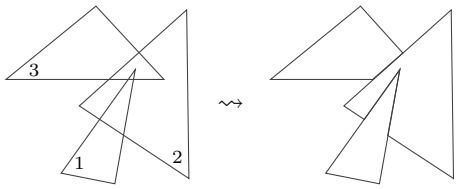

FIG. 1.7 – Application au recouvrement d'objets graphiques.

Chapitre 2

Calcul d'enveloppe convexe dans le plan

2.1 Définitions et problématique

But : Trouver la plus petite surface (resp. volume) convexe qui contienne un ensemble de points de $\mathbb{R}^2$ (resp. de $\mathbb{R}^3$).

Exemple 1. Dans $\mathbb{R}^2$:

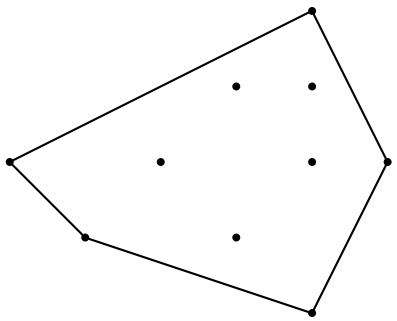

FIG. 2.1 – Enveloppe convexe de 10 points du plan.

Exemple 2. Dans $\mathbb{R}^3$:

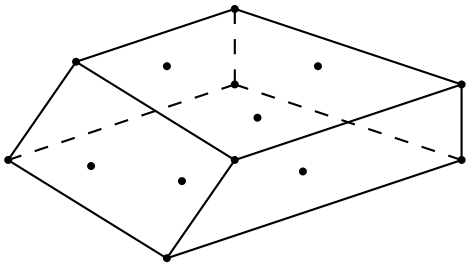

FIG. 2.2 – Enveloppe convexe de 14 points de l'espace.

Dans tout ce qui suit, on se placera systématiquement dans le plan euclidien $\mathbb{R}^2$.

Par commodité, on notera $[[a, b]]$ l'ensemble $\mathbb{Z} \cap [a, b]$ pour a et b entiers.

Définition 1. *Un* **polygone** *est une suite finie de segments $S_i = [p_i, q_i]$, $i \in [[1, n]]$ vérifiant :*

$$\begin{cases} q_i = p_{i+1} \text{ pour tout } i \in [[1, n-1]] \text{ ;} \\ q_n = p_1. \end{cases}$$

Quelquefois (par abus de langage)*, on appellera également* "polygone" *l'intérieur d'un tel polygone.*

Définition 2. *Un polygone est dit* **simple** *si pour tous les $i \neq j$ dans $[[1, n]]$ tels que S_i s'intersecte avec S_j, alors i et j sont consécutifs et les segments S_i et S_j s'intersectent uniquement en leurs extrémités.*

Exemple 3.

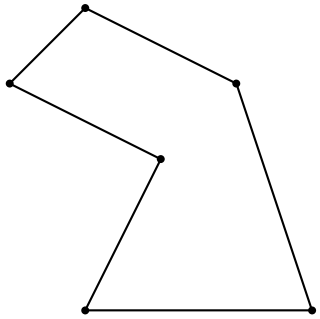

FIG. 2.3 – Un polygone simple.

Exemple 4.

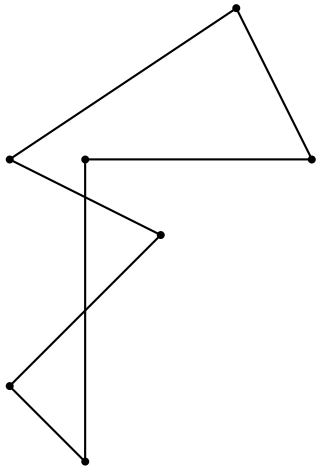

FIG. 2.4 – Un polygone pas simple !

Définition 3. *Un ensemble $E \subseteq \mathbb{R}^2$ est dit* **convexe** *si :*

$$\forall\ a, b \in E \text{ et } \forall\ c \in [a, b],\ \text{on a } c \in E.$$

Exemple 5.

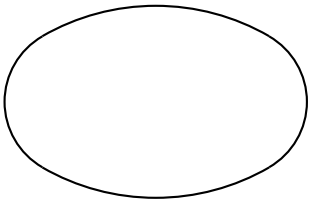

FIG. 2.5 – Un ensemble convexe.

Exemple 6.

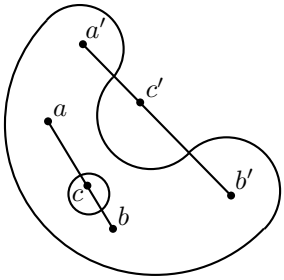

FIG. 2.6 – Un ensemble pas convexe.

Remarque 1. *Les polygones convexes sont des intersections de demi-plans... Plus précisément : un polygone convexe est exactement une intersection bornée non vide d'un nombre fini de demi-plans (résultat admis).*

Exemple 7.

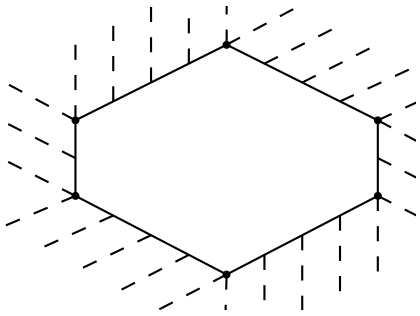

FIG. 2.7 – Intersection de demi-plans.

Définition 4.
*L'***enveloppe convexe** *d'un ensemble fini Q de points du plan est* le plus petit polygone convexe qui contient Q. *On la note : $EC(Q)$.*

Remarque 2. *Ici,* « le plus petit » *est au sens de l'inclusion.*

Problème : Comment calculer $EC(Q)$?

2.2 Algorithme de Jarvis : 1973

2.2.1 Principe

La technique utilisée est celle du « paquet cadeau ».

Exemple 8.

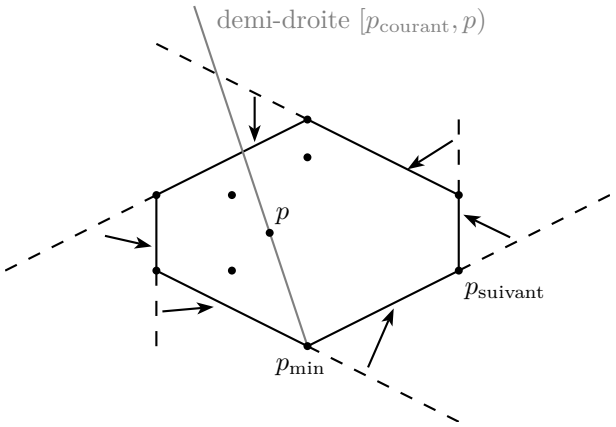

FIG. 2.8 – Jarvis : Illustration du principe.

2.2.2 L'algorithme

Algorithme 2 : Jarvis.

Données : Q = un ensemble fini de points du plan.
Sorties : L = la liste des sommets de $EC(Q)$ donnée dans le sens direct.

```
1  début
2      p_min ← le sommet le plus bas
3      p_courant ← p_min
4      Insérer(p_courant,L)
5      répéter
6          Choisir p ≠ p_courant
7          Prendre p_suivant le point le plus à droite de [p_courant, p)
8          p_courant ← p_suivant
9          Insérer(p_courant,L)
10     jusqu'à p_courant = p_min
11     retourner L
12 fin
```

2.2.3 Analyse de l'algorithme

Preuve :
On admet la validité de cet algorithme. □

Complexité :
La ligne 7 peut prendre un temps en $\Theta(n)$.
La boucle s'exécute autant de fois qu'il y a de sommets dans l'enveloppe convexe de Q.

Donc en tout, on est en $\Theta(n \cdot h)$ où $h := |EC(Q)|$.

Or $EC(Q) \subseteq Q \Longrightarrow h = |EC(Q)| \leqslant |Q| = n$.
Donc dans le pire des cas, on obtiendra une complexité en $\mathbf{O}(n^2)$. □

Remarque 3. *Tout ceci peut s'étendre à la dimension* 3.

2.3 Algorithme de Graham : 1972

2.3.1 Principe

Tout d'abord, on trie les sommets par angle polaire croissant.
Puis à chaque nouveau sommet :
- si on tourne à gauche, alors on continue à empiler le sommet ;
- si on tourne à droite, alors on revient d'un cran en arrière.

Exemple 9.

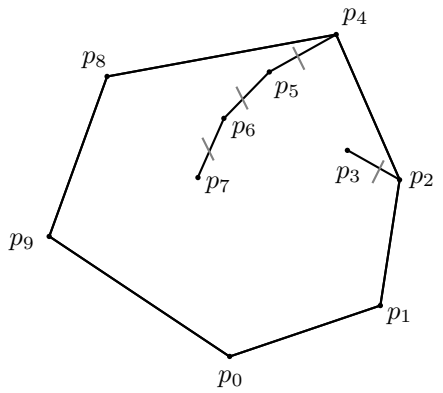

FIG. 2.9 – Graham : Illustration du principe.

2.3.2 Structure de données

Comme nous l'avons suggéré au paragraphe précédent, nous aurons besoin d'utiliser une structure de **pile**, notée S.

Les primitives associées à notre pile S seront :
- `Dernier(`S`)` : renvoie le sommet tout en haut de la pile S ;
- `AvantDernier(`S`)` : renvoie le sommet juste avant `Dernier(`S`)` ;
- `Empiler(`v`,`S`)` : empile le sommet v sur S ;
- `Dépiler(`S`)` : dépile le dernier élément de S.

2.3.3 L'algorithme

Algorithme 3 : Graham.

Données : Q = un ensemble fini de points du plan appelés sommets.
Sorties : S = l'enveloppe convexe de Q empilée dans le sens direct.

début
  `// ********** PHASE 1 : **********`
  $p_0 \leftarrow$ le sommet d'ordonnée minimale (en cas d'égalité, prendre celui qui est situé le plus à gauche).
  $p_1, \ldots, p_n \leftarrow$ les points restants de Q, **triés** par angle polaire croissant autour de p_0,
  à partir de la demi-droite horizontale issue de p_0 vers la droite.
  (En cas d'égalité entre plusieurs sommets, supprimer ces derniers sauf celui qui est le plus éloigné de p_0.)
  `// ********** PHASE 2 : **********`
  `Empiler(`p_0`,`S`)`
  `Empiler(`p_1`,`S`)`
  `Empiler(`p_2`,`S`)`
  pour chaque $i = 3$ jusqu'à n **faire**
    tant que p_i est à droite de [`AvantDernier(`S`)`,`Dernier(`S`)`] **faire**
      `Dépiler(`S`)`
    `Empiler(`p_i`,`S`)`
  retourner S
fin

2.3.4 Analyse de l'algorithme

Preuve :
On admet la validité de l'algorithme. □

Complexité :

Phase 1 → le tri est en $\Theta(n \log n)$;
Phase 2 → naïvement en $\mathbf{O}(n^2)$,
mais en fait, chaque point est empilé (et éventuellement dépilé) au plus une fois,
donc en $\Theta(n)$.

D'où une complexité en $\Theta(n \log n)$. □

Remarque 4. *Nous verrons en T.D. que $\Omega(n \log n)$ est une borne inférieure pour le calcul d'enveloppe convexe. Autrement dit, l'algorithme de Graham est* "optimal".

2.4 Rappels mathématiques

- $f(n) = \Omega(n \log n)$
  $\iff \exists\ \alpha$ telle que $f(n) \geqslant \alpha \cdot n \log n$ à partir d'un certain rang ;
- $f(n) = \mathbf{O}(n \log n)$
  $\iff \exists\ \beta$ telle que $f(n) \leqslant \beta \cdot n \log n$ à partir d'un certain rang ;
- $f = \Theta(n \log n)$
  $\iff \exists\ \alpha, \beta$ telles que $\alpha \cdot n \log n \leqslant f(n) \leqslant \beta \cdot n \log n$ à partir d'un certain rang.