Une application géométrique de la méthode diviser pour régner (d'après un problème de l'X).

I Problème : recherche de l'enveloppe convexe d'un nuage de points

I.1 Introduction

Notre objectif est de calculer des enveloppes convexes de nuages de points dans le plan affine, un grand classique en géométrie algorithmique. On rappelle qu'un ensemble $C \subseteq \mathbb{R}^2$ est convexe si et seulement si pour toute paire de points $p, q \in C$, le segment de droite $[p, q]$ est inclus dans C. L'enveloppe convexe d'un ensemble $E \subseteq \mathbb{R}^2$, notée Conv(E), est le plus petit convexe contenant E . Dans le cas où E est un ensemble fini (appelé *nuage de points*), le bord de Conv(E) est un polygône convexe dont les sommets appartiennent à E (résultat admis), comme illustré dans la figure 1.

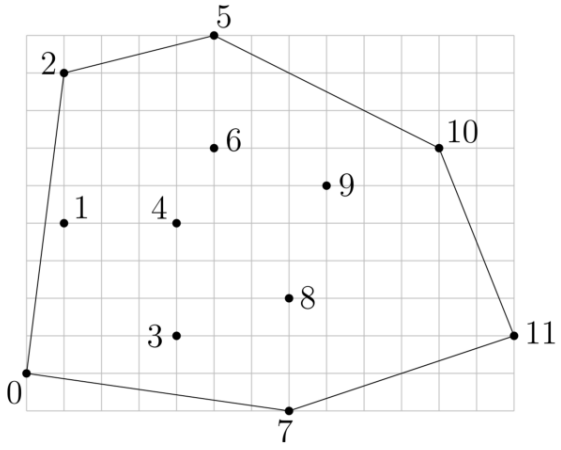

FIGURE 1 – Un nuage de points étiquettés de 0 à 11, et le bord de son enveloppe convexe.

Le calcul de l'enveloppe convexe d'un nuage de points est un problème fondamental en informatique, qui trouve des applications dans de nombreux domaines comme :

- la robotique, par exemple pour l'accélération de la détection de collisions dans le cadre de la planification de trajectoire,
- le traitement d'images et la vision, par exemple pour la détection d'objets convexes (comme des plaques minéralogiques de voiture) dans des scènes 2d,
- l'informatique graphique, par exemple pour l'accélération du rendu de scènes 3d par lancer de rayons,
- la théorie des jeux, par exemple pour déterminer l'existence d'équilibres de Nash,
- la vérification formelle, par exemple pour déterminer si une variable risque de dépasser sa capacité de stockage ou d'atteindre un ensemble de valeurs interdites lors de l'exécution d'une boucle dans un programme, et bien d'autres encore.

Nous allons écrire un algorithme de calcul du bord de l'enveloppe convexe d'un nuage de points E dans le plan affine, appelé *QuickHull*, qui s'appuie sur une approche du type diviser pour régner.

Les **points** seront représentés en CAML par des **couples de flottants** (float * float). Précisons enfin que les coordonnées sont données dans une base orthonormée du plan $(\overrightarrow{u_x}, \overrightarrow{u_y}, \overrightarrow{u_z})$, orientée dans le sens direct, $\overrightarrow{u_z}$ étant orthogonal au plan contenant le nuage.

Dans toute la suite on supposera que le nuage de points E est de taille $n \geqslant 3$ et en position générale, c'est-à-dire qu'il ne contient pas 3 points distincts alignés

Ces hypothèses vont permettre de simplifier les calculs en ignorant les cas pathologiques, comme par exemple la présence de 3 points alignés sur le bord de l'enveloppe convexe

I.2 Préliminaires

Rappel : Soit trois points du plan, p_1, p_2 et p_3 à partir desquels on construit les deux vecteurs $\overrightarrow{p_1p_2}$ et $\overrightarrow{p_1p_3}$. Le produit vectoriel $\overrightarrow{p_1p_2} \wedge \overrightarrow{p_1p_3}$ est un vecteur orthogonal au plan qui permet de définir l'aire $\mathcal{A}$ du triangle (p_1, p_2, p_3), comme illustré sur la figure 2. Cette aire $\mathcal{A}$ est algébrique (i.e. positive ou négative). On dit encore qu'elle est **signée**. Ainsi on écrira le produit vectoriel

$$\overrightarrow{p_1p_2} \wedge \overrightarrow{p_1p_3} = 2\mathcal{A}\,\overrightarrow{u}_z.$$

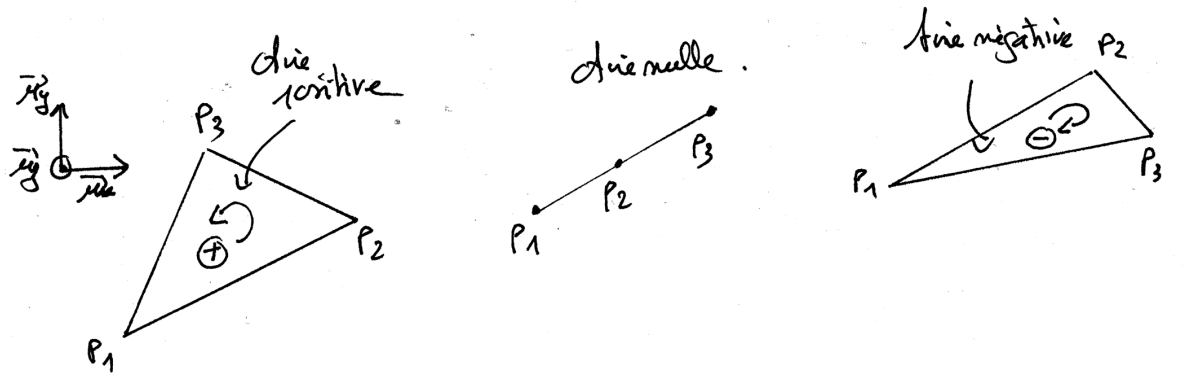

FIGURE 2 – Aire signée d'un triplet

1. En s'appuyant sur le calcul explicite des coordonnées d'un vecteur obtenu par un calcul de produit vectoriel, écrire une fonction `aire` qui prend comme arguments trois points du plan telle que `aire p1 p2 p3` calcule l'aire signée $\mathcal{A}$ définie précédemment.

   On rappelle que l'expression dans la base $\overrightarrow{u}_x, \overrightarrow{u_y}, \overrightarrow{u_z}$ du produit vectoriel $\overrightarrow{v}_1 \wedge \overrightarrow{v}_2$, des vecteurs $\overrightarrow{v}_1 = x_1\overrightarrow{u}_x + y_1\overrightarrow{u}_y + z_1\overrightarrow{u}_z$ et $\overrightarrow{v}_2 = x_2\overrightarrow{u}_x + y_2\overrightarrow{u}_y + z_2\overrightarrow{u}_z$ est

   $$\overrightarrow{v}_1 \wedge \overrightarrow{v}_2 = (y_1 \times z_2 - y_2 \times z_1)\,\overrightarrow{u}_x + (z_1 \times x_2 - z_2 \times x_1)\,\overrightarrow{u}_y + (x_1 \times y_2 - x_2 \times y_1)\,\overrightarrow{u}_z.$$

   Il nous faut également un test d'orientation que l'on définit ainsi :

   Définition 1 Étant donnés trois points p_1, p_2, p_3, distincts ou non, le test d'orientation renvoie $+1$ si l'aire signée du triangle p_1, p_2, p_3 est positive, -1 si elle est négative, et 0 si les trois points sont alignés.
2. Écrire une fonction `orient` qui prend comme arguments trois points du plan telle que `orient p1 p2 p3` calcule l'entier ($-1, 0$ ou $+1$) du test d'orientation.

I.3 Approche diviser pour régner : QuickHull

Un nuage de points est représenté par une **liste** de points ((float * float) list).

Commençons par définir l'ensemble des points situés à gauche d'une droite orientée.

Définitions 2 Soit $(\overrightarrow{ab})$ une droite orientée dans la direction donnée par le vecteur $\overrightarrow{ab}$ et E un ensemble de points.

On dira que le point m est à gauche de $(\overrightarrow{ab})$ si le test d'orientation pour le triangle (a, b, m) vaut $+1$ (cf. figure 4).

On note $E_{g,(\overrightarrow{ab})}$, l'ensemble contenant a, b et tous les points de E situés à gauche de $(\overrightarrow{ab})$.

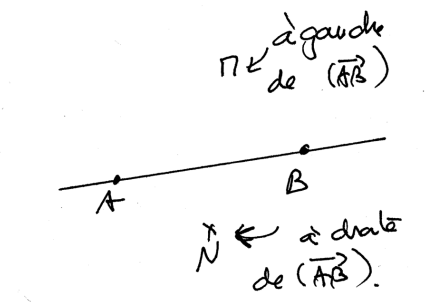

FIGURE 3 – Points à gauche ou à droite d'une droite orientée

On représente l'ensemble $E_{g,(\overrightarrow{ab})}$ des points situés à gauche de la droite orientée $(\overrightarrow{ab})$, par **une liste**, dont les deux premiers éléments sont **dans cet ordre** les points a et b, i.e. de la forme

```
a :: b :: queue .
```

Dans la suite vous utiliserez exclusivement la programmation fonctionnelle (i.e. récursive). La programmation impérative est interdite (pas de `ref` , pas de `while` , pas de `for`). L'utilisation de `if` est autorisée

3. Écrire une fonction `nuage_gauche` qui prend en paramètre une liste `nuage` décrivant un nuage de points ainsi que deux points p et q et telle que `nuage_gauche nuage p q` calcule la liste décrivant $E_{g,(\overrightarrow{pq})}$, i.e. l'ensemble des points de E situés à gauche de $(\overrightarrow{pq})$ union $\{p, q\}$, p et q étant dans cet ordre les deux premiers éléments de cette liste.

L'algorithme *QuikHull* repose sur l'algorithme récursif *SemiHull* suivant qui calcule l'enveloppe convexe d'un nuage de points de la forme $E_{g,(\overrightarrow{pq})}$. On commence par rechercher le point h de cet ensemble le plus éloigné de la droite $(\overrightarrow{pq})$. On considère alors les ensembles (**Attention à l'ordre des points !**) $E_{g,(\overrightarrow{ph})}$ et $E_{g,(\overrightarrow{hq})}$. On appelle récursivement *SemiHull* sur ces deux ensembles pour obtenir les listes des points constituant les deux bords des enveloppes convexes correspondantes. Il ne reste qu'à concaténer ces deux listes, en prenant garde à ne pas insérer le point h deux fois, pour obtenir la liste décrivant le bord de l'enveloppe convexe de $E_{g,(\overrightarrow{pq})}$.

Si $E_{g,(\overrightarrow{pq})}$ est réduit aux deux points p et q, *SemiHull* calcule la liste `[p; q]` (toujours faire attention à l'ordre...)

4. Justifier que la recherche du point h peut se résumer à celle du point h pour lequel `aire p q h` est maximale.
5. Écrire une fonction récursive `plus_loin` prenant en paramètre une liste `nuage_g` décrivant l'ensemble $E_{g,(\overrightarrow{pq})}$ telle que `plus_loin nuage_g` calcule le point de $E_{g,(\overrightarrow{pq})}$ le plus éloigné de la droite $(\overrightarrow{pq})$.

   Si la liste `nuage_g` a deux éléments ou moins on lèvera une exception `"nuage trop petit"` avec `failwith` .

6. Justifier par un schéma qu'une fois que le point h a été déterminé on ne s'intéresse qu'aux ensembles $E_{g,(\overrightarrow{ph})}$ et $E_{g,(\overrightarrow{hq})}$, et pas $E_{g,(\overrightarrow{hp})}$ ni $E_{g,(\overrightarrow{qh})}$.
7. Écrire une fonction récursive `semihull` prenant en paramètre une liste `nuage_g` décrivant l'ensemble $E_{g,(\overrightarrow{pq})}$ telle que `semihull nuage_g` calcule la liste des points constituant le bord de l'enveloppe convexe de $E_{g,(\overrightarrow{pq})}$.
   Si la liste `nuage_g` a un élément ou moins on lèvera une exception `"nuage trop petit"` avec `failwith` .

On peut maintenant revenir au problème initial et écrire l'algorithme *QuickHull*. L'idée est de déterminer les deux points p le plus à gauche et q le plus à droite de l'ensemble initial. On peut alors utiliser l'algorithme *SemiHull* sur $E_{g,(\overrightarrow{pq})}$ et $E_{g,(\overrightarrow{qp})}$ et combiner les résultats pour avoir l'enveloppe convexe recherchée, en faisant attention à ne pas insérer deux fois le même point.

8. Écrire la fonction `points_extremes` prenant en paramètre une liste `nuage` décrivant un nuage de points et qui calcule le couple de points `(gauche, droite)` tel que `gauche` soit un (il n'y a pas nécessairement unicité) point du nuage de plus petite abscisse et que `droite` soit un (il n'y a pas nécessairement unicité) point du nuage de plus grande abscisse.
   Le nuage initial étant supposé avoir au moins trois points, on ne gèrera pas d'exception (mais le compilateur risque de râler en levant un warning...).
9. Écrire enfin la fonction `quickhull` prenant en paramètre une liste `nuage` décrivant un nuage de points telle que `quickhull nuage` calcule la liste des points constituant le bord de l'enveloppe convexe de ce nuage. Même remarque que pour la fonction `points_extremes` à propos de l'exception...
10. On suppose qu'en moyenne dans *SemiHull* après la découverte du point h le découpage du nuage de n points amène à étudier deux nuages de tailles à peu près égales à $n/2$. Écrire la relation permettant d'exprimer le coût $C(n)$ de l'algorithme. En déduire la complexité de *QuickHull* en moyenne.
11. Décrire un pire des cas et estimer la complexité correspondante.