Algorithme et Structures de Données

Yousra ADERKAOUI - Hossam ELOUATI

April 16, 2020

1 Introduction

Etant donné un ensemble de polygones, le principe de détection d'inclusions de polygones repose essentiellement sur le choix de l'algorithme testant si un point donné est à l'intérieur d'un polygone ou à l'extérieur, vu que tous les polygones ne présentent aucune intersection entre eux. Savoir si un point d'un polygone X appartient ou pas à l'intérieur d'un polygone Y suffit de conclure le statut de X par rapport à Y.

2 Présentation des algorithmes

Pour résoudre le problème du test de l'appartenance d'un point à un polygone, on propose trois algorithmes différents: ***crossing number***, ***quadrant product*** *et* ***cutting triangle***.

2.1 Algorithme *Crossing Number*

Cette méthode se base sur l'intersection d'une demi droite, arrivant de l'infini vers le point, avec les segments du polygone. En choisissant une demi droite horizontale, on calcule le nombre d'intersection de cette demi droite et en raisonnant sur la parité de ce nombre, on déduit le statut du point: si ce nombre est **pair**, le point est à l'**extérieur**, sinon il est à l'**intérieur**. (*voir FIG.1*)

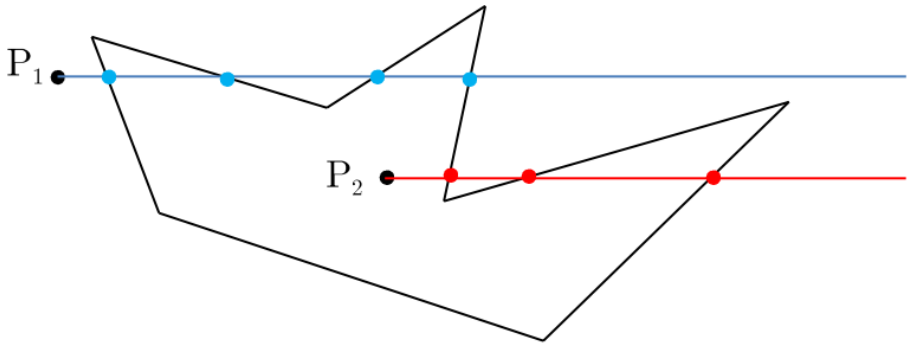

FIG.1: *Crossing number: pour le point P_1, le nombre d'intersection est pair donc il est à l'extérieur. Pour P_2, il est impair donc il est à l'intérieur.*

2.2 Algorithme *Quadrant Product*

Cet algorithme repose sur le principe fameux ***Diviser pour régner***. Tout d'abord, avant toute division, on trace le quadrant (bounding quadrant) du polygone. Si le point est à l'extérieur du quadrant, le point est systématiquement à l'extérieur du polygone. Sinon,

on oriente le polygone selon le sens des aiguilles de la montre et on procède par une division selon les quadrants des côtés du polygone.

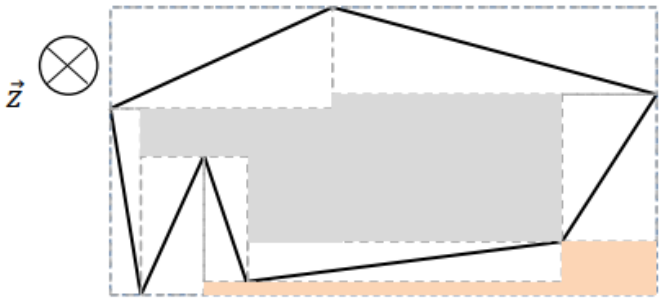

FIG.2: *Division du polygone en quadrants.*

Après cette division, si le point se trouve dans un des quadrants, on procède comme ceci:
- Le vecteur $\vec{z}$ entrant définit le sens d'orientation du polygone;
- On trace le vecteur dirigeant le segment selon le sens d'orientation que l'on nomme $\vec{a}$;
- Avec le vecteur $\vec{p}$ allant de l'origine du vecteur $\vec{a}$ vers le point, on effectue les tests suivants: si $(\vec{a} \wedge \vec{p}).\vec{z} > 0$ alors le point est intérieur, sinon il est extérieur. *($\wedge$ et . signifie respectivement le produit vectoriel et le produit scalaire de deux vecteurs.)*

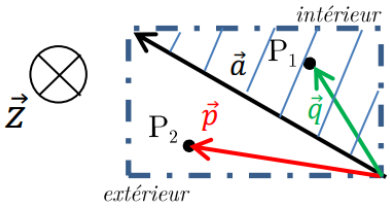

FIG.3: *Méthode de détection si un point est à l'intérieur ou à l'extérieur d'un polygone avec des produits vectoriels. Avec cette méthode, P_1 est à l'intérieur et P_2 est à l'extérieur. La partie hachurée montre la zone intérieure du polygone.*

Si le point ne se trouve dans aucun quadrant, alors il est soit à l'intérieur soit à l'extérieur. Pour cela, on cherche le plus proche sommet du polygone par rapport au point, puis on sélectionne le segment d'origine le sommet proche et l'autre segment allant vers ce sommet proche. On effectue les tests suivants :
- on note $\vec{a}$ le vecteur dirigeant le premier segment, $\vec{b}$ le vecteur dirigeant le second segment, $\vec{p}$ le vecteur du début du premier segment vers le point et $\vec{q}$ le vecteur du début du second segment vers le point.
- si $(\vec{a} \wedge \vec{b}).\vec{z} > 0$:
si de plus $(\vec{a} \wedge \vec{p}).\vec{z} > 0$ et $(\vec{b} \wedge \vec{q}).\vec{z} > 0$ alors le point est à l'**intérieur** sinon il est à l'**extérieur**.
- si $(\vec{a} \wedge \vec{b}).\vec{z} < 0$:
si de plus $(\vec{a} \wedge \vec{p}).\vec{z} > 0$ ou $(\vec{b} \wedge \vec{q}).\vec{z} > 0$ alors le point est à l'**intérieur** sinon il est à l'**extérieur**.

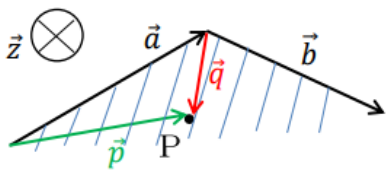

FIG.4: *Méthode de détection si un point est à l'intérieur ou à l'extérieur d'un polygone avec des produits vectoriels. Avec cette méthode, P est à l'intérieur du polygone.*

2.3 Algorithme *Cutting Triangle*

Cette méthode se base également sur le principe de **Diviser pour régner**. En revanche, la division se fait d'une façon plus ou moins différente. En effet, on divise le polygone en bandes selon les ordonnées des sommets de ce polygone. Après cette division, si le point n'appartient à aucune bande, alors le point est automatiquement à l'extérieur. Sinon, on sélectionne cette bande où le point est localisé.

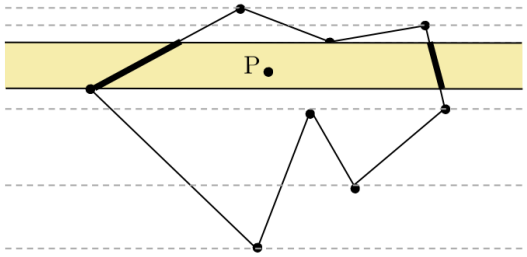

FIG.5: *Division du polygone en bandes selon les ordonnées des sommets.*

Après avoir sélectionné la bande, on détecte les parties des côtés qui sont délimitées par celle bande. Ces segments définissent des formes géométriques qui sont soit des trapèzes soit des triangles. Si une forme géométrique était un trapèze alors il faut la trianguler et ce juste en dessinant une diagonale.

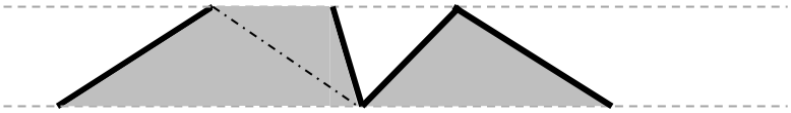

FIG.6: *Triangulation des formes géométriques délimitées dans une bande donnée.*

Après la triangulation, on cherche si le point est inclus dans un des triangles. Pour détecter si un point est inclus dans un triangle donné, on effectue le test suivant:
Soit ABC un triangle et P un point.
Si $(\overrightarrow{AB} \wedge \overrightarrow{AP}).\vec{z} \geq 0$ et $(\overrightarrow{BC} \wedge \overrightarrow{BP}).\vec{z} \geq 0$ et $(\overrightarrow{CA} \wedge \overrightarrow{CP}).\vec{z} \geq 0$ alors P est à l'intérieur de ABC, sinon il est à l'extérieur.

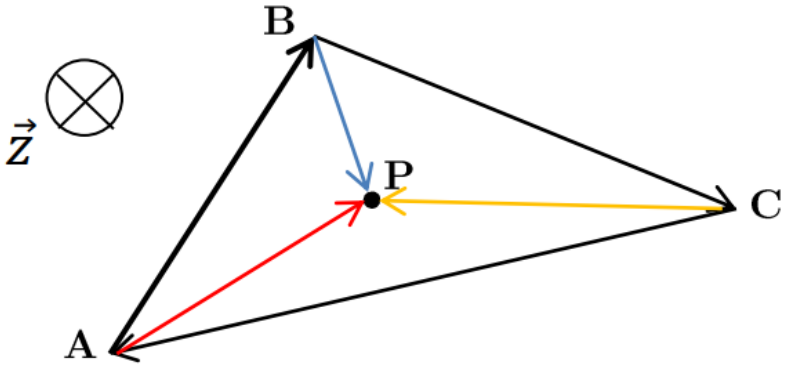

FIG.7: *Inclusion d'un point dans un triangle.*

3 Etude théorique: Calcul des coûts

On note **n**: le nombre de sommets d'un polygone.

3.1 *Crossing Number*

Pour cet algorithme, tous les côtés du polygone sont parcourus et pour chaque côté des tests arithmétiques et booléens qui se font en un temps $O(1)$, donc on peut conclure que

dans tous les cas, l'algorithme a un coût $O(n)$.

3.2 *Quadrant Product*

On note C le coût moyen de l'algorithme.
On a: $C = C_{orient} + C_{diviser} + C_{boucle} + C_{proche}$
- C_{orient} : est le coût pour orienter le polygone. Ses côtés sont tous parcourus lors de ce processus, donc ce coût est **linéaire**.
- $C_{diviser}$: est le coût de la division du polygone en quadrants. Pour cela, chaque côté de ce polygone est parcouru et le traçage du quadrant encadrant se fait en un temps $O(1)$. Donc la division a un coût $O(n)$.
- C_{boucle} : c'est le coût de la recherche du quadrant qui contient le point.
On a $C_{boucle} = \sum_{i=1}^{n} C_i$ avec C_i est le coût sachant que le point se trouve dans le quadrant du segment d'indice i.

$C_i = E[X]$ avec X: le nombre de quadrants parcourus pour arriver au quadrant i.

X est une variable géométrique de paramètre p où $p = \frac{i}{n}$ coorespondant au tirage de i quadrants parmi n.
Donc: $C_i = \frac{1}{p} = \frac{n}{i}$.

Et par suite $C_{boucle} = n.\sum_{i=1}^{n} \frac{1}{i} = n.H_n \simeq n.log(n)$
- $C_{proche} = C_{pointproche} + C_{segment1} + C_{segment2} = O(n) + O(n) + O(n) = O(n)$
En effet la recherche du sommet le plus proche du point en question et la sélection des deux segments liés par le sommet le plus proche se font chacun en un temps **linéaire**.
En meilleur cas, l'algorithme peut se faire en un temps linéaire si le point se trouvait dans dans le premier quadrant. En pire cas, l'algorithme pourra parcourir tous les segments sans aucun résultat.
Conclusion: L'algorithme *Quadrant Product* a un coût de $O(nlog(n))$

3.3 *Cutting Triangle*

On note également C le coût moyen de l'algorithme.
On a: $C = C_{diviser} + C_{localiser} + C_{partition} + C_{formes} + C_{triang} + C_{tests}$
- $C_{diviser}$: La division, dans ce cas, se fait en un temps **linéaire** puisque tous les sommets du polygone sont parcourus et la construction de bandes se fait en prenant en compte les ordonnées de ces sommets.
- $C_{localiser}$: On note m le nombre de bandes produites par l'opération de division.
On a: $2 \leq m \leq n$.
Le coût de la recherche de la bande qui contient le point est:
$C_{localiser} = \sum_{i=1}^{m} C_i$ avec C_i est le coût sachant que le point se trouve dans la bande d'indice i.

$C_i = E[X]$ avec X: le nombre de bandes parcourues pour arriver à la bande i.

X est encore une variable géométrique de paramètre p où $p = \frac{i}{m}$ coorespondant au tirage de i bandes parmi m.
Donc: $C_i = \frac{1}{p} = \frac{m}{i}$.

Et par suite $C_{boucle} = m.\sum_{i=1}^{m} \frac{1}{i} = m.H_m \simeq m.log(m)$
- $C_{partition}$: Tous les segments sont parcourus dans tous les cas et on ne sélectionne que les parties délimitées par la bande. L'opération de troncage d'un segment se fait en un temps **constant** donc la partition des côtés du polygone a un coût $O(n)$.

On note α: le nombre de segments délimités dans la bande. On a: $2 \leq \alpha$ et $\alpha \leq n-1$ si n est impair, sinon $\alpha \leq n-2$.
- C_{formes}: Les formes géométriques délimitées dans la bande sont définies par les segments. Donc, $C_{formes} = O(\alpha)$.
- C_{triang}: Les formes géométriques sont soit des trapèzes soit des triangles. On note β le nombre de formes géométriques, on a $\beta = \frac{\alpha}{2}$. Ceci se fait alors en temps $O(\frac{\alpha}{2})$.
On note: s le nombre de triangles après la triangulation. On a: $\frac{\alpha}{2} \leq s \leq \alpha$. La borne inférieure signifie que toutes les formes sont des triangles, par contre la borne supérieure signifie que toutes les formes sont des trapèzes.
- C_{tests}: Tous les triangles sont parcourus et on essaie de chercher si le point appartient à un certain. Le coût est donc: $O(s)$
Conclusion: Le coût moyen de l'alogrithme est donc: $C = O(n) + O(m.log(m)) + O(n) + O(\alpha)$. Au meilleur cas, le point ne se trouve dans aucune bande, le coût est donc $O(n)$. Au pire cas, c'est à dire $m = n$, on le coût serait $O(n.log(n))$.

4 Etude expérimentale: Mesure de performances

4.1 Protocole expérimental

Pour comparer les trois algorithmes en titre de performances, des polygones, de nombre de sommets différents, seront générés et pour chacun, on effectuera un test d'appartenance à l'aide d'un point donné.
La génération de polygones se fait en deux étapes:
- Génération aléatoire d'un nombre $n \in [5, 10000]$ de points
- Construire l'enveloppe convexe qui lie les points générés
On mesure pour chaque polygone généré le temps effectué pour chaque test en utilisant les trois algorithmes. On note les prélévations et par simple mesure de simplicité, on multiplie les temps d'exécution par 10^5. Grâce à ces prélévements, on trace les trois courbes des trois algorithmes représentant le temps d'éxécution en fonction du nombre de sommets.

4.2 Résultats expérimentaux

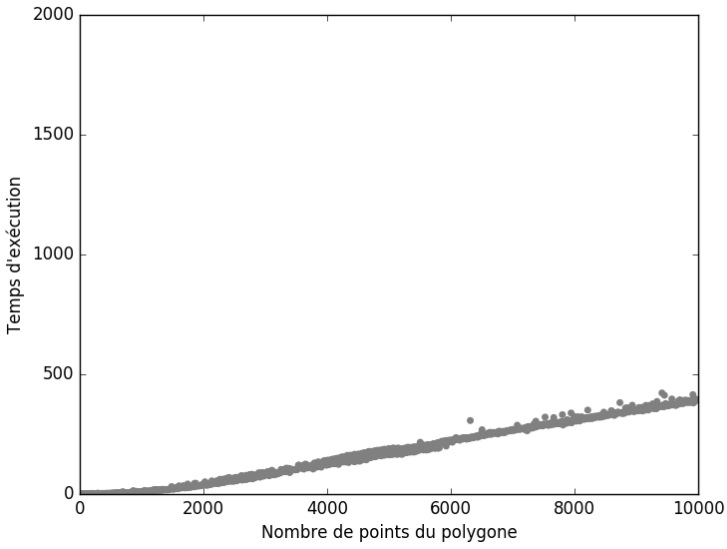

FIG.8: *Courbe de mesure de l'algorithme Crossing Number.*

La courbe de la **FIG.8** représente le résultat des tests effectués grâce à l'algorithme *Crossing Number*. Le nuage de points est condensé autour d'une droite. Ce qui renforce parfaitement ce qui a été évoqué dans la partie théorique. Le coût de cet algorithme est bel et bien **linéaire**.

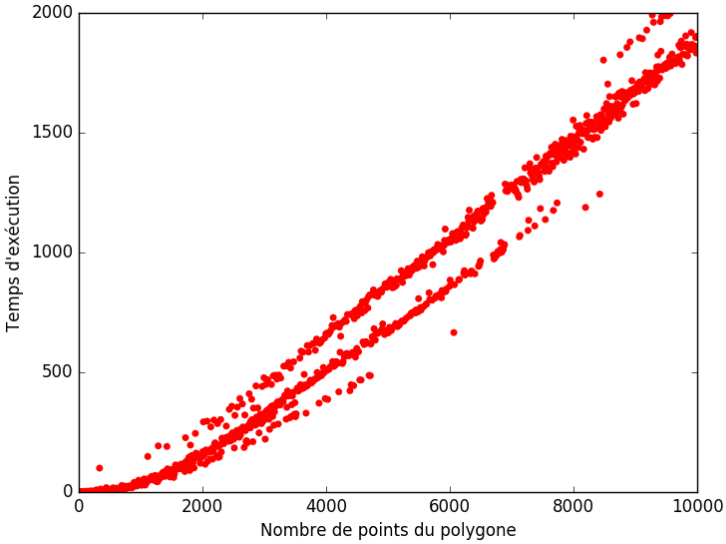

FIG.9: *Courbe de mesure de l'algorithme Quadrant Product.*

La courbe de la **FIG.9** représente le résultat des tests effectués grâce à l'algorithme *Quadrant Product*. Le nuage de points est plus ou moins dispersés mais on remarque deux familles de points qui se condensent autour de deux courbes qui ont une allure de $n.log(n)$ (La courbure qui se présente au voisinage de 0 montre qu'il s'agit d'une courbe $n.log(n)$ et non n.

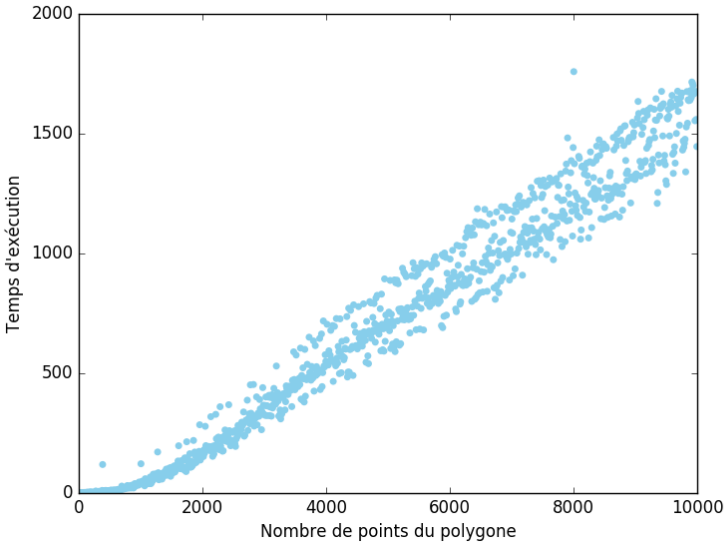

FIG.10: *Courbe de mesure de l'algorithme Cutting Triangle.*

La courbe de la **FIG.10** représente quant à elle les tests effectués grâce à l'algorithme *Cutting Triangle*. Le nuage des points, au voisinage de l'infini est considérablement dispersé et ce est dû au fait qu'il y a une marge assez importante entre le meilleur cas et le pire cas. Mais en tout cas, le nuage oscille autour d'une courbe de $n.log(n)$.

4.3 Conclusion

Les hypothèses faites dans la partie théorique sont bel et bien confirmées par les mesures de performances. On conclut que l'algorithme *Crossing Number* est l'algorithme le plus performant parmi les trois. Il sera donc utilisé lors de l'affichage des inclusions de l'ensemble des polygones.

5 Affichage des inclusions

Après avoir traité le problème de l'inclusion d'un point dans un polygone, il est à présent temps d'afficher les inclusions des polygones.
Etant donné la liste de polygones, on la trie suivant le critère de la surface des polygones. On parcourt la liste triée élément par élément. On choisit un point de ce polygone. Pour ce point, on parcourt une nouvelle fois la liste triée. Si le point est à l'extérieur du quadrant incluant le polygone, alors on ne change rien. Sinon, on effectue le test d'inclusion suivant l'un des algorithmes.
Le coût de l'algorithme de détection d'inclusions se fait en un temps $O(n^2)$ avec (n est le nombre de polygones).

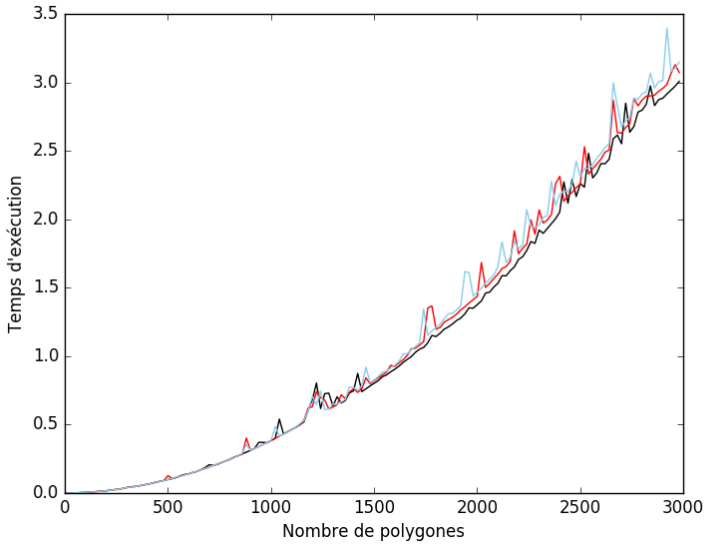

FIG.11: *Courbe de mesure de l'algorithme de détection d'inclusions.* ***Bleu****:Cutting Triangle,* ***Rouge****:Quadrant Product,* ***Noir****:Crossing Number.*

Ces résultats sont obtenus en choisissant un nombre de polygones compris entre 2 et 3000. L'allure des trois courbes sont bel et bien celle de la fonction n^2, ce qui renforce ce qui a été conclut auparavant.
On remarque que les courbes de l'algorithme *Quadrant Product*(en rouge) et *Cuuting Triangle*(en bleu) sont presque identique. Par contre, la courbe de *Crossing Number* est légèrement au-dessous des deux autres. On conclut donc que *Crossing Number* est l'algorithme le plus performant. L'indifférence des courbes montre que le choix d'un algorithme de test du point dans un polygone a une influence plus ou moins importante sur l'algorithme de détection d'inclusions.