

COMPLEXITÉ ALGÈBRE DES ALGORITHMES GÉOMÉTRIQUES — LE PROBLÈME D'INTERSECTIONS D'UN ENSEMBLE DE SEGMENTS

RADU-LUCIAN LUPȘA

ABSTRACT. Il est connu qu'un problème difficile dans l'implantation des algorithmes géométriques est le calcul exact des points d'intersection des droites ou des courbes, ainsi que l'évaluation des prédicats géométriques. D'un part, il s'agit du fait que, si les calculs sont inexacts, il est possible qu'un algorithme se comporte complètement imprévisible. D'autre part, beaucoup d'algorithmes ne traitent pas explicitement les cas particuliers dégénérés.

On étudie dans cet article le cas de l'algorithme de Balaban pour trouver les intersections d'un ensemble de segments, du point de vue de l'évaluation des prédicats géométriques.

1. INTRODUCTION

L'article étudie les problèmes liés à l'évaluation des prédicats et au traitement des cas dégénérés. Le cas étudié est celui de l'algorithme de Balaban pour trouver les intersections d'un ensemble de segments.

Le reste de l'introduction présente l'algorithme de Balaban [1], y compris les définitions dont on a besoin à la suite. Puis, on présente l'étude de l'auteur sur les solutions au problème des cas dégénérés, au problème de l'évaluation des prédicats et finalement une comparaison des performances.

Quelques notations et définitions:

Si les ordonnées des extrémités d'un segment ne nous intéressent pas, le segment sera noté (l, r) , où l et r sont les abscisses des extrémités gauche, respectivement droite.

On appelle verticale une droite verticale passant par une extrémité de segment. La verticale v est donc la droite $x = v$.

On appelle bande une partie du plan délimitée par deux verticales.

On dit qu'un segment (l, r) :

- est intérieur pour la bande (b, e) si $l > b$ et $r < e$;
- traverse la bande (b, e) si $l \leq b$ et $r \geq e$.

Received by the editors: December 10, 2003.

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.3.5: [**Computer Graphics**]: Computational Geometry and Object Modeling – *Geometric algorithms, languages, and systems.*

On note $\text{Int}_{b,e}(S_1, S_2)$, où S_1 et S_2 sont des ensembles quelconques de segments l'ensemble des paires $(s_1, s_2) \in S_1 \times S_2$ de segments qui s'intersectent dans la bande (b, e) .

Une liste de segments qui intersectent une verticale v est triée par rapport à cette verticale si elle est triée dans l'ordre croissant des ordonnées des points d'intersection entre les segments de la liste et la verticale v .

Enfin, on appelle un escalier un triplet (b, e, Q) où (b, e) est une bande et Q est un ensemble de segments qui traversent la bande (b, e) et ne s'intersectent pas à l'intérieur de la bande (l'ensemble Q doit être trié d'après la règle ci-dessus par rapport aux verticales b et e (l'ordre est la même par rapport aux deux)).

Pour trouver les intersections à l'intérieur d'une bande élémentaire (sans extrémités de segments à l'intérieur), on utilise l'algorithme suivant:

```

procedure ChercherDansBande( $b, e, L, R$ )
  Diviser( $b, e, L, Q, L'$ )
  if  $Q = \emptyset$ 
    then  $R := L$ 
    else
      trouver et écrire  $\text{Int}_{b,e}(Q, L')$ 
      ChercherDansBande( $b, e, L', R'$ )
      Fusioner( $b, e, R', Q, R$ )
    endif
  end

```

Ici:

- $\text{Diviser}(b, e, L, Q, L')$ divise la liste L triée par rapport à la verticale b dans l'escalier maximal (au sens d'inclusion d'ensembles) (b, e, Q) et la liste restante L' .
- $\text{Fusioner}(b, e, R', Q, R)$ fusionne la liste R' triée par rapport à la verticale e avec l'escalier (b, e, Q) en sortant la liste R triée par rapport à e .

Ainsi ChercherDansBande divise la liste initiale L en deux, Q et L' ; maintenant

$$\text{Int}_{b,e}(L, L) = \text{Int}_{b,e}(L', L') \cup \text{Int}_{b,e}(L', Q) \cup \text{Int}_{b,e}(Q, Q)$$

où $\text{Int}_{b,e}(Q, Q) = \emptyset$ et $\text{Int}_{b,e}(L', L')$ est trouvé par l'appel récursif.

Il faut remarquer aussi que Diviser obtient aussi les positions des segments de L' dans l'escalier Q , de manière que les intersections d'un segment de L' avec les marches de Q puissent être trouvées en balayant Q à partir de la position donnée par Diviser dans les deux sens jusqu'à trouver une marche qui n'intersecte pas le segment.

Passons maintenant à l'algorithme principal. Balaban décrit au fait deux algorithmes, dont le premier est un peu plus simple, mais asymptotiquement sub-optimal (complexité $O(n \log^2 n + k)$, où n est le nombre de segments et k le nombre d'intersections), et le deuxième est optimal ($O(n \log n + k)$) mais plus compliquée.

L'algorithme sub-optimal est le suivant

```

begin
   $L := \{\text{le segment qui commence sur la première verticale}\}$ 
   $I := \text{l'ensemble initial moins } L \text{ moins le segment qui}$ 
    finit sur la dernière verticale
  ChercherDansArbre(première vert., dernière vert.,  $L$ ,  $I$ ,  $R$ )
end
avec:
procedure ChercherDansArbre( $b, e, L, I, R$ )
  if  $b$  et  $e$  consécutives
    then ChercherDansBande( $b, e, L, R$ )
    else
       $c := \text{la verticale qui divise en deux l'ensemble}$ 
        des verticales comprises entre  $b$  et  $e$ 
      Diviser( $b, e, L, Q, L_1$ )
      trouver  $\text{Int}_{b,e}(Q, L_1)$ 
       $I_1 := \{s \in I \mid s \text{ intérieur pour } (b, c)\}$ 
      ChercherDansArbre( $b, c, L_1, I_1, R_1$ )
       $s := \text{le segment dont une extrémité}$ 
        if  $s$  commence sur  $c$ 
          then  $L_2 := R_1 \cup \{s\}$ 
          else  $L_2 := R_1 \setminus \{s\}$ 
        endif
       $I_2 := \{s \in I \mid s \text{ intérieur pour } (c, e)\}$ 
      ChercherDansArbre( $c, e, L_2, I_2, R_2$ )
      trouver  $\text{Int}_{b,e}(Q, I)$ 
      Fusionner( $b, e, R_2, Q, R$ )
      trouver  $\text{Int}_{b,c}(Q, R_2)$ 
    endif
end

```

Dans cet algorithme, la seule opération qui demande un temps plus grand que $O(n \log n + k)$ est *trouver* $\text{Int}_{b,e}(Q, I)$, car elle nécessite une recherche binaire de la position de chaque segment de I dans Q . Pour éviter ceci, la solution consiste à garder les positions des segments de I trouvés dans les appels récursifs de *TreeSearch*; en ce but il faut garder des relations entre les escaliers générés au différents niveaux d'appel de *TreeSearch*.

2. CAS PARTICULIERS ET DÉGÉNÉRÉES

Le premier problème à l'implantation de cet algorithme est le fait que l'algorithme ne traite pas explicitement les cas particuliers:

- si l'intersection de deux segments se trouve exactement sur une verticale;

- si le point d'intersection se trouve sur une extrémité de segment;
- si deux extrémités de segment se trouvent sur la même verticale;
- si un segment est vertical;
- si deux segments se superposent (ont la même droite-support).

A ce point il est utile de regarder un peu la preuve de correction de l'algorithme. On s'aperçoit alors que les demandes sur les prédicats sont les suivantes:

- (1) Si deux segments ne s'intersectent pas dans une bande, alors leur ordre relative sur les verticale gauche et droite doit être la même;
- (2) Si un segment est localisé entre deux marches m_i et m_{i+1} d'un escalier $Q_{b,e}$ et il n'intersecte ni m_i , ni m_{i+1} , alors il ne doit intersecter aucune autre marche de $Q_{b,e}$;
- (3) Un segment ne peut intersecter que des marches consecutives d'un même escalier.

Prenons ces cas un à la fois.

Si le point d'intersection se trouve exactement sur une verticale, il suffit de considerer q'il se trouve "un peu à droite" ou "un peu à gauche", donc dans une des deux bandes délimitées par la verticale en question. Mettons-le donc à gauche. Maintenant pour garder la cohérence, si deux segments s'intersectent sur une verticale, il faut mettre en premier sur la liste triée celui qui se trouvera en premier sur la verticale suivante (car ils ne s'intersecteront plus dans la bande de droite), c'est-à-dire le premier en ordre trigonométrique dans le demi-plan droit.

Maintenant si le point d'intersection coïncide avec une extrémité, il se trouvera donc sur une verticale, et conformément au paragraphe précédent il va être considéré dans la bande de gauche. S'il coïncide avec l'extrémité droite du segment, il va être traité comme s'il était à l'intérieur; par contre s'il coïncide avec l'extrémité gauche il ne sera pas vu car le segment n'existe pas dans la bande gauche. Ce cas d'intersection doit donc être traité lors de l'insertion du segment sur la liste de segments correspondante à cette verticale. *Remarque:* si au moment de l'insertion l'autre segment est une marche, l'intersection sera vue normalement par l'algorithme.

S'il existe plusieurs extrémités de segment sur la même verticale, il y a deux solutions: soit on considère des bandes de longueur zero, soit on prépare la liste triée des extrémités (avec les segments correspondants) et on la fusionne avec la liste des segments associés à la verticale. J'ai pris la deuxième approche. En faisant ainsi, le traitement d'une verticale ne consiste plus seulement à ajouter ou effacer un segment (obtenir la liste L_2 à partir de la liste R_1), mais à fusionner la liste L_1 avec la liste des segments qui commencent sur la verticale traitée et à effacer par la même occasion les segments qui finissent sur la verticale traitée. En même temps, on garde une liste de segments verticaux, qu'on met au jour chaque fois qu'on rencontre une extrémité d'un segment vertical.

Enfin, le dernier problème concerne le cas où deux segments partagent la même droite support et leur intersection est un segment. Pour les segments verticaux, ceci n'est pas dérangeant, leur intersection étant trouvée lors du traitement de la

verticale sur laquelle ils se trouvent. Si, au contraire, les segments sont obliques ou horizontaux, ils s'intersectent dans plusieurs bandes en agrandissant le nombre d'intersections et donc le temps de calcul. (n segments superposés peuvent donner ainsi jusqu'à $\frac{1}{6}(2n^3 + n)$ "points" d'intersection). Dans ce cas on peut considérer qu'il s'intersectent toujours en un seul point; soit ce point le point le plus à droite de l'intersection. L'ordre de ces segments sur une liste associée à une verticale n'a pas alors d'importance.

3. EVALUATION DES PRÉDICATS

Il est connu que les algorithmes géométriques sont très sensibles aux erreurs numériques; d'ici la nécessité de faire les calculs exactes, ou au moins que les valeurs de vérité des divers prédicats évalués pendant l'exécution de l'algorithme soit cohérentes; sinon il y a le risque que la réponse de l'algorithme ne soit ni même une approximation du résultat réel, ou même que l'algorithme ne s'arrête pas.

Les calculs exactes d'autre part sont chères car non supportés directement par le matériel de l'ordinateur. Et dans tous les cas, plus longue soit la représentation, plus long est le temps de calcul. L'opération la plus chère en terme de demande de chiffres significatives sur la représentation étant la multiplication, on voit l'intérêt de réduire le plus possible le degré des polynômes qui apparaissent dans l'évaluation des prédicats.

Dans le cas de l'algorithme de Balaban, les prédicats décrits dans la section précédente sont basés sur les trois suivants ($<_x$ signifie "l'abscisse du premier point est plus petite que celle du deuxième et $<_y$ signifie la même chose sur l'ordonnée):

- (1) étant données 2 points, p_1 et p_2 , est-ce que $p_1 <_x p_2$?
- (2) étant données 3 points, est-ce que p_3 se trouve à gauche, sur ou à droite de la droite p_1p_2 (orientée de p_1 vers p_2)
- (3) étant données 5 points $p_0 \dots p_4$ tels que $\{q\} = p_1p_2 \cap p_3p_4$, est-ce que $p_0 <_x q$ (q est inconnu et il n'est pas nécessaire de le connaître explicitement)?

En supposant une représentation cartésienne des points, l'évaluation des trois prédicats ci-dessus est équivalente à trouver le signe d'un polynôme de degré 1, 2 respectivement 3, irréductible. Pour évaluer ledit signe, il faut en principe travailler avec des nombres ayant 3 fois plus de chiffres que les coordonnées des points. Mais dans [2] il est montré que l'évaluation "presque exacte" du prédicat 3 est possible en travaillant en réels (IEEE 754 [5]) double-précision, les coordonnées des points étant simple-précision. Ce "presque exacte" veut dire que si le signe calculé du polynôme est -1 alors le signe réel est aussi -1 ; même chose pour $+1$; mais si le signe calculé est 0 alors on ne sait rien sur le signe réel.

4. LES PERFORMANCES

Dans le tableau ci-dessous sont marqués les temps de calcul utilisés (sur le même ordinateur) par l'algorithme naïf et l'algorithme sub-optimal et l'algorithme optimal de Balaban en utilisant chacun les prédicats géométriques de la bibliothèque

CGAL [3] (qui utilisent à leur tour les rationnels en précision illimitée de la bibliothèque LEDA [4]) et des predicates écrits en utilisant les réels double-précision.

Les exemples de test ont été générés soit aléatoirement, soit (pour les dernières 4) sous la forme d'une grille carrée.

Méthode génération	Nr. seg.	Nr. pairs	Temps naïf	Temps sub-opt CGAL	Temps opt CGAL	Temps sub-opt dbl-prec	Temps opt dbl-prec
aléat	100	215	1	1	4	1	1
aléat	200	886	3	3	11	1	2
aléat	800	14113	45	29	97	8	24
aléat	2000	6869	—	28	129	7	29
aléat	5000	43184	—	122	550	28	125
aléat	2000	89352	—	—	—	47	99
aléat	5000	555640	—	—	—	261	382
grille	800	2281	15	1	1	1	1
grille	1800	5221	84	1	1	1	1
grille	5000	14701	—	2	2	4	2
grille	20000	59401	—	9	8	16	8

TABLE 1. Temps d'exécution (en secondes) pour (en l'ordre) l'algorithme naïf, l'algorithme sous-optimal avec les prédicats implanté par CGAL, l'algorithme optimal avec les prédicats CGAL, l'algorithme optimal avec les prédicats évalués en double-précision et l'algorithme optimal avec les prédicats en double précision

On voit facilement que pour des exemples de taille raisonnable, l'algorithme sous-optimal asymptotiquement se comporte nettement mieux, le gain d'un facteur $\log n$ (7–13 dans les exemples ci-dessus) étant contrebalancé par la perte due aux complications de l'algorithme. Il faut remarquer aussi le gain de vitesse obtenu en remplaçant les rationnels en précision illimitée (bibliothèque LEDA) par les réels double-précision de la machine.

REFERENCES

- [1] I. BALABAN, An Optimal Algorithm for Finding Segment Intersections, Proceedings of the Eleventh Annual Symposium on Computational Geometry, Vancouver, Canada, June 5-7, 1995, pg. 211-219
- [2] Jean-Daniel BOISSONNAT, Franco P. PREPARATA, Robust Plane Sweep for Intersecting Segments, Rapport de recherche no. 3270 septembre 1997, Institut National de Recherche en Informatique et en Automatique
- [3] <http://www.cgal.org/>
- [4] <http://www.mpi-sb.mpg.de/LEDA/leda.html>
- [5] <http://grouper.ieee.org/groups/754/>

BABEŞ-BOLYAI UNIVERSITY, FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, RO-400084 CLUJ-NAPOCA, ROMANIA

E-mail address: rlupsa@cs.ubbcluj.ro