Structure de données II : Rapport de l'étape préliminaire du projet

Groupe 5 :
HUYLENBROECK Florent
DACHY Corentin

Année Académique 2018-2019
Bachelier en Sciences Informatiques

Faculté des Sciences, Université de Mons

Contents

1 Introduction

Pour ce travail, voici les consignes qui nous on été demandées :

> Pour se familiariser avec les arbres BSP (*Binary space partitions*), il vous est demandé de réaliser l'exercice préliminaire suivant :
>
> Etant donné un arbre BSP représentant une scène dans un plan (ensemble de segments) et deux points x et y dans ce plan, donnez un algorithme recursif en pseudo-code qui indique si le segment d'extrémités x et y appartient à la scène. Veuillez accompagner votre algorithme :
>
> - d'une explication de son fonctionnement; et
> - d'une discussion autour de sa complexité (ne vous limitez pas au pire des cas).
>
> Nous convenons, pour cet exercice, que les segments contenus dans un même nœud de l'arbre BSP sont stockés dans une liste chaînée.
>
> **Remarque** : Cet exercice préliminaire n'est qu'une mise en route du projet. Il ne sera pas nécessaire à la résolution du problème principal.

2 Résolution

2.1 Pseudo-code

Algorithm 1 belongsToScene

Détermine si un segment dont les deux extrêmités données sous formes de points en entrée appartient à l'arbre BSP donné en entrée.

Entrées : BSP : Partition de recherche binaire
On assume que chaque noeud contient l'équation de la droite qu'il décrit et (facultatif) le segment qui lui est confondu, et chaque feuille contient un segment, décrit par une paire de points $S(S.x, S.y)$ et $S'(S'.x, S'.y)$.
A : Point correspondant à une extremité du segment.
Ce point a pour coordonnées ($A.x$,$A.y$).
B : Point correspondant à l'autre extremité du segment.
Ce point a pour coordonnées ($B.x$,$B.y$).
Sorties : Boléen, vrai si le segment appartient à la scene, faux sinon.
Effets : /

1: **procedure** BELONGSTOSCENE(BSP, A, B)
2: $d \leftarrow$ COEFFICIENTANGULAIRE(A, B)
3: **retourner** RECHERCHER(BSP, A, B, d)
4: **end procedure**

Algorithm 2 coefficientAngulaire

Calcule le coefficient angulaire d'un segment.

Entrées : A : Racine de la sous-partition de recherche binaire que l'on doit chercher
B : Point que l'on recherche.
Sorties : Le coeficient angulaire de la droite passant par A et B.
Une valeur sentinelle $+\infty$ sera retournée si la pente est verticale.
Effets : /

1: **procedure** COEFICIENTANGULAIRE(A, B)
2: **if** $A.x - B.x == 0$ **then**
3: **retourner** $+\infty$
4: **else**
5: **retourner** $(A.y - B.y)/(A.x - B.x)$
6: **end if**
7: **end procedure**

Algorithm 3 rechercher

Recherche récursivement un segment dans un arbre BSP.

Entrées : BSP : Partition de recherche binaire
On assume que chaque noeud contient l'équation de la droite qu'il décrit et (facultatif) le segment qui lui est confondu, et chaque feuille contient un segment, décrit par une paire de points $S(S.x, S.y)$ et $S'(S'.x, S'.y)$.
P : Point, premiére extremité du segment recherché dans le BSP
B : Point, deuxième extremité du segment recherché dans le BSP
d : Entier (ou valeur sentinelle $+\infty$), coeficient angulaire du segment recherché.
Sorties : Boléen, vrai si le segment PB appartient au BSP
Effets : /

```
1: procedure RECHERCHER(BSP, P, B, d)
2:     S[] ← nouvelle liste vide
3:     LOCALISER(BSP, P, S[])
4:     REDUIRE(S[], d)
5:     if S[] vide then
6:         retourner False
7:     else
8:         for segment in S[] do
9:             P' ← extremité de segment qui n'est pas P
10:            if P' == B then
11:                retourner True
12:            else
13:                if P' sur un bord then
14:                    retourner RECHERCHER(BSP, P', B, d)
15:                end if
16:            end if
17:        end for
18:    end if
19:    retourner False
20: end procedure
```

Algorithm 4 localiser

Recherche récursivement un point donné dans les segments d'un arbre BSP

Entrées : $root$: Racine de la sous-partition de recherche binaire où l'on doit chercher
P : Point que l'on recherche.
$return[]$: Liste contenant les résultats de la recherche.
Sorties : /
Effets : La liste $return[]$ est mise à jour.

```
1: procedure LOCALISER(root, P, return[])
2:     if root est une feuille then
3:         if P ∈ root then
4:             ajouter root dans return[]
5:         end if
6:     else
7:         res ← résultat de la résolution de l'équation de la droite décrite par root avec P.x et P.y
8:         if res ≥ 0 then
9:             LOCALISER(root+, P, return[])
10:        else if res ≤ 0 then
11:            LOCALISER(root−, P, return[])
12:        else
13:            if P ∈ root then
14:                ajouter root dans return[]
15:            end if
16:            LOCALISER(root+, P, return[])
17:            LOCALISER(root−, P, return[])
18:        end if
19:    end if
20: end procedure
```

Algorithm 5 reduire

Réduit un ensemble de segments pour ne garder que ceux qui ont un coefficient angulaire donné.

Entrées : $S[]$: Ensemble de segments à réduire.
d : Entier (ou valeur sentinelle $+\infty$), coeficient angulaire du segment recherché.
Sorties : /
Effets : La liste $S[]$ ne contient plus que les segments qui ont un coefficient angulaire d.

```
1: procedure REDUIRE(S[], d)
2:     for all elements s de S[] do
3:         sd ←COEFICIENTANGULAIRE(s.A, s.B)
4:         if sd ≠ d then
5:             retirer s de S[]
6:         end if
7:     end for
8: end procedure
```

2.2 Discussion de la complexité