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

2.1.1 Algorithme belongsToScene

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 : /

```
procedure BELONGSTOSCENE(BSP, A, B)
    d ← COEFFICIENTANGULAIRE(A, B)
    retourner RECHERCHER(BSP, A, B, d)
end procedure
```

2.1.2 Algorithme coefficientAngulaire

Algorithm 2 coefficientAngulaire

Calcule le coefficient angulaire d'un segment.

Entrées : A : Point, première extrêmité du segment, de coordonnées ($A.x$,$A.y$).
B : Point, deuxième extrêmité du segment, de coordonnées ($B.x$,$B.y$).
Sorties : Le coeficient angulaire de la droite passant par A et B.
Une valeur sentinelle $+\infty$ sera retournée si la pente est verticale.
Effets : /

```
procedure COEFICIENTANGULAIRE(A, B)
    if A.x − B.x == 0 then
        retourner +∞
    else
        retourner (A.y − B.y)/(A.x − B.x)
    end if
end procedure
```

2.1.3 Algorithme rechercher

Algorithm 3 rechercher

Recherche récursivement un segment dans un arbre BSP.

Entrées :	BSP :	Partition de recherche binaire.
	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 :		/

```
procedure RECHERCHER(BSP, P, B, d)
    S[] ← nouvelle liste vide
    LOCALISER(BSP, P, S[])
    REDUIRE(S[], d)
    if S[] vide then
        retourner False
    else
        for segment in S[] do
            if P ∈ segment then
                P′ ← extremité de segment qui n'est pas P
                if P′ == B then
                    retourner True
                else if P′ sur un bord then
                    retourner RECHERCHER(BSP, P′, B, d)
                end if
            end if
        end for
    end if
    retourner False
end procedure
```

2.1.4 Algorithme localiser

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. On assume que *root* possède un attribut d étant l'équation de la droite décrite par *root* et S qui contient le segment (*root* est une feuille) ou les segments (*root* est un noeud) contenus dans *root*, sous la forme de deux points par segments (s'il y en a plusieurs, ils seront contenus dans une liste chainée). $root+$ représente le sous-arbre au dessus de $root.d$, et $root-$ représente le sous-arbre en dessous de $root.d$.
- P : Point que l'on recherche, de coordonnées ($P.x$,$P.y$).
- $return[]$: Liste des segments (paires de points) contenant le point recherché.

Sorties : /

Effets : Les segments contenant P ont été ajoutés à $return[]$.

```
procedure LOCALISER(root, P, return[])
    if root est une feuille then
        if P ∈ root.S then
            ajouter root dans return[]
        end if
    else
        res ← résultat de la résolution de root.d par le point P.x et P.y
        if res ≥ 0 then
            LOCALISER(root+, P, return[])
        else if res ≤ 0 then
            LOCALISER(root−, P, return[])
        else
            for segment in root.S do
                if P ∈ segment then
                    ajouter segment dans return[]
                end if
            end for
            LOCALISER(root+, P, return[])
            LOCALISER(root−, P, return[])
        end if
    end if
end procedure
```

2.1.5 Algorithme reduire

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.

```
procedure REDUIRE(S[], d)
    for all elements s de S[] do
        sd ← COEFICIENTANGULAIRE(s.x, s.y)
        if sd ≠ d then
            retirer s de S[]
        end if
    end for
end procedure
```

2.2 Discussion de la complexité

Dans l'étude de la complexité de nos algorithmes, nous allons nommer :

- s le nombre de segments dans la scène.
- h la hauteur de l'arbre BSP.
- l la largeur de l'arbre BSP.

2.2.1 Algorithme coefficientAngulaire

Dans le pire des cas comme dans un cas moyen, l'appel à cet algorithme se fait en $O(1)$.

2.2.2 Algorithme reduire

Dans le pire des cas, $S[]$ contient tous les segments de la scéne. Ce cas n'est envisageable dans notre procédure que lorsque tous les segments de la scène ont une extrêmité en commun. Il s'agira alors d'une complexité en $O(s)$ (on suppose que l'on retire les segments de la liste par index, donc en $O(1)$).

Dans le cas moyen par contre, $S[]$ ne contiendra que quelques segments. Le corp de la boucle ne contenant que des instructions en $O(1)$, l'algorithme aura pour complexité $O(\text{nombre de segments dans } S[])$.

Pour la suite de la discussion, nous considérerons une complexité de $O(s)$ pour cet algorithme. Nous considérerons aussi que, dans le pire des cas, la liste $S[]$ contiendra, après application de la fonction, s segments, contre 1 dans un cas moyen.

2.2.3 Algorithme localiser

Chaque appel récursif de cet algorithme "descend" d'un noeud dans l'arbre. La condition d'arrêt est que l'on atteint une ou plusieurs feuilles. Les instructions autres que les appels à la fonctions étant en 0(1) (affectation, ajout d'un élément à une liste) et que l'on considère (lignes 13:17) que $root.S$ contient dans le pire des cas tous les segments de la scène, et dans un cas moyen un segment, on aura, dans le pire des cas une complexité en $0(h + s)$, et dans un cas moyen $O(h)$.

Note : il est possible qu'une récursion appelle deux fois la fonction (*cf* lignes 18:19, mais cela est négligeable car $O(2h) \in O(h)$.

2.2.4 Algorithme rechercher

Dans le pire des cas, le segment recherché traverse tout l'arbre BSP. On aura donc un nombre d'appel récursifs égal au nombre l de feuilles de l'arbre. On a donc une complexité de $O(h + s + s$ (lignes 3:4) $+l \cdot s$ (lignes 7:18 si $S[]$ contient tous les segments de la scène)). Donc $O(h + s + l \cdot s)$.

Dans un cas moyen la complexité sera $O(h + s + l)$ par les considérations précédentes.

2.2.5 Algorithme belongsToScene

Nous avons donc au final, un algorithme ayant comme complexité dans le pire des cas $O(1 + h + s + s \cdot l)$ donc $O(h + s + s \cdot l)$ et dans un cas moyen $O(1 + h + s + l)$ donc $O(h + s + l)$.