Structures de données II - Rapport de Projet.

Dubuc Xavier & Hannecart Aurore

3 mai 2010

Table des matières

---

1 Introduction

Notre projet consistait à déterminer ce que voit un oeil d'une scène en deux dimensions sur un segment en une dimension. Pour celà, nous avons dû nous familiariser avec les arbres **BSP** et implémenter leur construction ainsi que l'algorithme du peintre permettant d'interroger un tel arbre et afficher ce qui est vu par un oeil fournit en paramètre. L'énoncé du problème principal est donc : «**Etant donné un arbre BSP représentant une scène dans un plan (ensemble de segments) et un oeil (un point dans ce plan), donner un algorithme qui affiche la représentation de ce que voit l'oeil**». Nous fournissons dès lors 2 programmes :

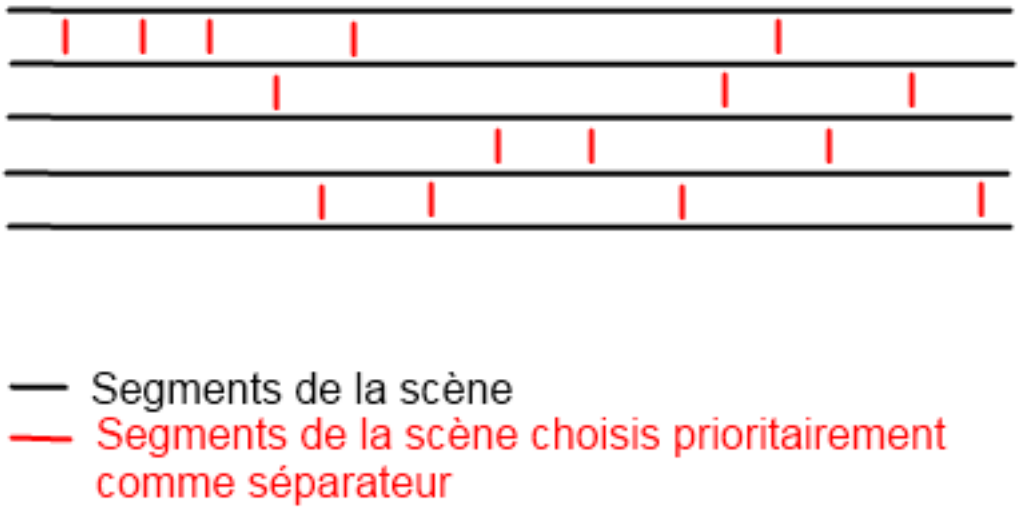
- le premier, ***Test_GUI.java***, est un programme interactif permettant de charger un fichier contenant une scène en 2D, choisir la position du point de vue, choisir une des heuristiques de construction des arbres **BSP** disponibles et finalement afficher la représentation graphique de ce que voit l'oeil en appliquant l'algorithme du peintre.
- le second, ***Test_Heuristiques.java***,est un programme *console* qui permet de comparer les différentes heuristiques de construction des arbres **BSP**, mais qui n'affiche pas de représentation graphique de ce que voit l'oeil à l'écran. Ces heuristiques seront comparées en fonction des critères suivants : la **taille** de l'arbre **BSP** généré par l'heuristique, la **hauteur** de cet arbre, le **temps CPU** utilisé pour construire cet arbre ainsi que le **temps CPU** utilisé pour appliquer l'algorithme du peintre sur cet arbre.

2 Algorithme du peintre

Cet algorithme nous permet d'interroger l'arbre **BSP** que l'on a créé afin de répondre à notre problème, il est donc primordial de connaître la complexité de celui-ci dans le pire des cas. Après réflexion nous trouvons que cette complexité est en $O(n)$ avec n qui est la taille de l'arbre **BSP** (en sachant que la taille de cet arbre est en fait le nombre de segments ou fragments de segments contenus dans l'arbre). En effet, afin d'obtenir un ordre ainsi que la projection des segments sur la droite orthogonale à la vue, il faut parcourir chaque segment/fragment de segments que contient l'arbre afin de lui appliquer la projection et pour chaque segment, on applique de simples instructions en $O(1)$ d'où la complexité totale en $O(n)$.

On peut également exprimer cette complexité en terme de nombre de noeuds que l'arbre contient, notons ce nombre m :

- Nombre de noeuds visités : dans tous les cas on visitera tout l'arbre, afin d'obtenir tous les segments qu'il contient, on a donc m noeuds visités.
- Nombre de références vides visitées : aucune, les arbres étant implémentés pour que le cas de base soit une feuille vide.
- Coût par noeud : Mis à part quelques instructions en $O(1)$, nous nous attachons surtout aux 2 voire 3 boucles exécutées, l'une concerne les segments/fragments de segment contenus dans le noeud, les 2 autres les segments/fragments de segments contenus dans les fils. Toutes ces boucles misent ensemble, vu qu'elles se suivent, le coût total sera donc la somme des coûts des boucles et ce coût sera au maximum égal à la taille de l'arbre (vu qu'il s'agit de tous les segments). Cependant il est excessif de dire que le coût est en $O(n)$, car, par exemple pour une noeud assez bas dans l'arbre, il est évident que ces 3 boucles ne vont pas parcourir toute les segments/fragments de segments que contient l'arbre. Plaçons nous dans le pire des cas, on sait que comme l'algorithme dépend de la taille de l'arbre, ce pire des cas sera lorsque cette taille sera maximale par rapport au nombre de segments donnés. Il faut donc qu'à chaque séparation, le maximum de segments soient coupés, mais surtout qu'à chaque étape il y ait au moins un segment coupé, si possible. Dans cette optique, nous pensons que le pire des cas serait une sorte de quadrillage et la découpe se ferait à chaque fois via des droites parallèles avant de se faire dans l'autre sens (cf image ci-dessous pour plus de clarté).



On a donc que chaque segment noir sera coupé lors des premières découpes (on peut imaginer beaucoup plus de segments rouges pour augmenter la taille encore, le dessin est juste à titre indicatif afin d'appuyer notre idée), et par la suite chacun de ces segments partitionnera le petit espace restant en 2 parties contenant à chaque fois des parties de segments noirs restantes ou rien. Notre raisonnement sera donc le suivant, à chaque étape de séparation par un segment rouge, le segment rouge choisi est contenu dans le noeud, il coupe les segments noirs en 2 et possède donc de chaque coté un ensemble de segments rouges entiers et de parties de segments noirs. On peut donc affirmer que chaque noeud possède un ensemble de segments à partitionner de taille $n_{pere} - 1$ au maximum et **il devra donc parcourir $n_{pere} - 1$ segments dans les 3 boucles décrites plus haut dans le pire des cas.** Cette quantité étant majorée par n, on a donc une **complexité dans le pire des cas en $O(m*n)$**.

Comme dit précédemment, ce pire des cas est très excessif, on calcule donc la complexité en moyenne, et au lieu d'avoir $O(m*n)$ instructions, on va décomposer le nombre de segments visités au fur et à mesure que l'on descend dans l'arbre, on a donc n pour le premier noeud, $n-1$ pour le second, ... jusqu'à ce que l'on arrive à 0. On considère que chaque noeud a une probabilité équivalente d'être visité équivalente à $\frac{1}{m}$. On a donc au final une complexité totale en $\frac{1}{m} * O\left(n + (n-1) + (n-2) + (n-3) + ... + 0\right) = O\left(\frac{1}{m}\sum_{i=0}^{n} i\right) = O\left(\frac{1}{m}\frac{n(n-1)}{2}\right) = O\left(\frac{n^2}{m}\right)$. Cette complexité, bien qu'en moyenne, n'en est pas moins calculée sur la base du pire des cas concernant la taille de l'arbre, et on peut montrer que dans ce cas, la taille de l'arbre sera toujours inférieure ou égale au nombre de noeuds de l'arbre (c'est assez évident, vu que chaque noeud ne contient qu'un seul segment ou fragment au maximum, on a donc au moins autant de noeuds que de segments/fragments auxquels il faut ajouter les éventuels noeuds vides). On peut donc conclure que la complexité est en $O\left(\frac{m^2}{m}\right) = O(m)$.

3 Test des heuristiques

Nous avions à implémenter 3 heuristiques, l'heuristique **linéaire** qui consiste à prendre simplement le premier segment de l'ensemble comme séparateur, l'heuristique **aléatoire** qui consiste à prendre un segment au hasard et finalement l'heuristique de **Teller** définie ci dessous.
Soit S un ensemble de segments dans $\mathbb{R}^2$. Soit T, un arbre **BSP** associé à l'ensemble de segments S. On rappelle que chaque noeud interne v de T contient l'équation d'une droite d_v. Le sous-arbre gauche de v est un arbre **BSP** pour le plan d_v^- et le sous-arbre droit d_v^+. A chaque noeud v de l'arbre on peut donc associer :

- une région R_v du plan (déterminée par les droites définies dans les noeuds allant de la racine jusqu'au noeud v) ;
- un ensemble I_v de segments se trouvant dans R_v : soit un segment S, soit un fragment d'un segment de S.

Soient un noeud v donné et un segment $s \in I_v$. On note d la droite supportant le segment s. Nous pouvons alors calculer, parmi l'ensemble de segments I_v, les valeurs suivantes :

- le nombre f_d de segments I_v intersectés par d ;
- la proportion σ_d de segments intersectés par d définie par $\sigma_d = \dfrac{f_d}{|I_v|}$.

L'idée de l'heuristique de Teller est la suivante : on choisit la droite d qui :

- maximise σ_d si $\sigma_d \geq \tau$,
- minimise f_d sinon,

où τ est un seuil fixé.

Nous détaillons ici nos tests en fonction des 3 heuristiques que nous avons dû implémenter, nous utilisons l'heuristique de Teller avec un seuil de 0.5.

Voici un tableau regroupant les **temps d'exécution (en secondes) pour l'algorithme de création des arbres** appliqué à un fichier de chaque type :

Scène	Nombre de segments	Heuristique Linéaire	Heuristique Aléatoire	Heuristique de Teller
Octangle	14	0	0	0
EllipsesMedium	720	0.06	0.02	11.92
RandomSmall	2939	0.04	0.08	34.87
RectanglesHuge	16800	0.12	0.04	343.08

Voici un tableau regroupant les **temps d'exécution (en secondes) pour l'application de l'algorithme du peintre** à un fichier de chaque type :

Scène	Nombre de segments	Heuristique Linéaire	Heuristique Aléatoire	Heuristique de Teller
Octangle	14	0	0	0
EllipsesMedium	720	0.24	0.04	0.26
RandomSmall	2939	0.16	0.08	0.24
RectanglesHuge	16800	0.26	0.08	0.28

Voici un tableau regroupant les **hauteurs des arbres BSP** crées par l'algorithme de création des arbres appliqué à un fichier de chaque type :

Scène	Nombre de segments	Heuristique Linéaire	Heuristique Aléatoire	Heuristique de Teller
Octangle	14	11	10	11
EllipsesMedium	720	720	130	720
RandomSmall	2939	54	28	121
RectanglesHuge	16800	46	15	43

Voici un tableau regroupant les **tailles des arbres BSP** crées par l'algorithme de création des arbres appliqué à un fichier de chaque type :

Scène	Nombre de segments	Heuristique Linéaire	Heuristique Aléatoire	Heuristique de Teller
Octangle	14	14	15	14
EllipsesMedium	720	720	738	720
RandomSmall	2939	4541	3922	3537
RectanglesHuge	16800	16400	16413	16402

Remarques :

- Il est normal que l'heuristique de **Teller** prenne plus de temps car l'algorithme permettant de définir la droite à utiliser parcourt 2 fois la liste des segments qui lui est passée en paramètre alors que les 2 autres heuristiques ne font qu'accéder à l'un des segments.
- L'heuristique de **Teller** augmente le temps d'exécution pour la construction de l'arbre, cependant l'algorithme du peintre prend moins de temps (lorsque le seuil est plus petit que 0.5, voir plus bas). Cependant, il est à remarquer que l'heuristique aléatoire a des performances semblables voir meilleures que celles de l'heuristique.
- On remarque que pour les scènes de rectangles, l'heuristique de **Teller** n'est pas très efficace car la construction prend beaucoup de temps pour peu d'amélioration au niveau de l'algorithme du peintre.

Nous allons maintenant détailler un peu nos tests sur l'heuristique de Teller en modifiant la valeur du seuil.

Pour les temps d'exécution de construction de l'arbre :

Scène	Nombre de segments	$\tau = 0.75$	$\tau = 0.5$	$\tau = 0.1$	$\tau = 0.01$	$\tau = 0.001$
Octangle	14	0	0	0	0	0
EllipsesMedium	720	10.38	10.42	10.5	10.46	0.36
RandomSmall	2939	30.34	30.36	30.46	2.52	2.54
RectanglesHuge	16800	329.52	321.22	313.94	313.36	39.31

Pour les temps d'exécution d'application de l'algorithme du peintre :

Scène	Nombre de segments	$\tau = 0.75$	$\tau = 0.5$	$\tau = 0.1$	$\tau = 0.01$	$\tau = 0.001$
Octangle	14	0	0	0	0	0
EllipsesMedium	720	0.2	0.26	0.26	0.2	0.06
RandomSmall	2939	0.28	0.24	0.26	0.12	0.12
RectanglesHuge	16800	0.28	0.28	0.31	0.29	0.17

Pour les hauteurs :

Scène	Nombre de segments	$\tau = 0.75$	$\tau = 0.5$	$\tau = 0.1$	$\tau = 0.01$	$\tau = 0.001$
Octangle	14	11	11	9	9	9
EllipsesMedium	720	720	720	720	720	105
RandomSmall	2939	121	121	120	58	58
RectanglesHuge	16800	43	43	43	43	27

Pour les tailles :

Scène	Nombre de segments	$\tau = 0.75$	$\tau = 0.5$	$\tau = 0.1$	$\tau = 0.01$	$\tau = 0.001$
Octangle	14	14	14	16	16	16
EllipsesMedium	720	720	720	720	720	809
RandomSmall	2939	3537	3537	4014	5952	5952
RectanglesHuge	16800	16402	16402	16402	16402	16422

Remarques :

- On remarque que plus τ est petit, plus le nombre de segments coupés augmente (ça favorise le premier cas, cas où l'on maximise la proportion de segments coupés) et donc plus la taille augmente.
- On remarque également que plus τ diminue, plus le temps de construction de l'arbre diminue et plus la hauteur de l'arbre diminue. C'est normal, vu que l'on favorise le découpage des segments, on évite, en pratique, quasi tous les cas où l'arbre contient des feuilles vides. La hauteur diminuant, les appels récursifs sont réduits et celà explique la diminution du temps **CPU** consacré à la création de l'arbre **BSP**.
- On remarque une grande différence entre les différents résultats pour la taille pour le *Random* car les segments sont placés de manière assez aléatoires et donc pour chaque segment choisi il y a de grandes chances d'en couper d'autres. On remarque exactement l'inverse dans le *Rectangle*, les segments étant placés de manière à avoir une grand nombre de segments colinéaires.

4 Reflexion sur le projet

Nous avons eu le plus de problèmes lors de la projection des segments sur l'horitonzale afin d'afficher ce que voit la vue dans notre rectangle dans le programme graphique. Nous avons finalement trouvé un moyen qui fonctionne pas trop mal qui consiste à utiliser la droite horizontale passant par le point d'ordonnée minimale des segments projetés sur la droite perpendiculaire à la vue et de projeter tous ces segments sur la nouvelle droite. Pour projeter ceux-ci, nous calculons la taille (distance euclidienne entre les 2 extrêmités) des segments et nous la reportons sur la droite horizontale. Une fois que nous avons cette droite, nous calculons le rapport entre la taille du segment et la taille de la droite totale et nous l'adaptons à la taille du rectangle où l'on veut afficher, tout en gardant les proportions.

Un autre problème récurrent était que nous avions des difficultés à prendre en compte tous les cas possibles, nous plaçant toujours dans un cas trop spécifique et en occultant les autres possibles.

5 Utilisation du programme graphique

L'application se lance et ne peut être utilisée que lorsqu'un fichier de scène valable a été ouvert (via Fichier > Ouvrir ...), ensuite l'utilisation est assez instinctive, mais il y a quelques subtilités que nous détaillons ici :

- vous pouvez tracer une vue sur la scène avec un «*drag & drop*»,
- vous pouvez (dé)zoomer avec la molette de votre souris en maintenant la touche «*Control*»,
- vous pouvez placer la scène en zoom 1x en maintenant la touche «*Control*» et en cliquant avec le bouton gauche de la souris,
- vous pouvez afficher la scène entièrement en maintenant la touche «*Alt*» et en cliquant avec le bouton gauche de la souris,
- vous pouvez entrer les coordonnées de la vue et ensuite cliquer sur le pinceau afin de dessiner celle-ci (la vue doit être absolument dessinée avant de cliquer sur le «*V*» vert pour valider et afficher ce que voit la vue), (Les coordonnées sont et doivent être exprimées dans le repère JAVA)

6 Utilisation du programme console

Il suffit de suivre les instructions, cependant quelque chose d'important, pour lancer les gros fichiers, il est nécessaire d'augmenter la taille de la pile **JAVA** sinon le programme ne pourra pas fonctionner. Il suffit pour cela d'ajouter l'option «-Xss128m» lors de l'exécution du programme.

7 Remerciements

Nous avons tenu à glisser un petit mot à votre encontre afin de vous remercier pour la compréhension dont vous avez fait preuve en repoussant maintes fois la date de remise du projet. Un énorme merci à vous, car celà nous a énormément aidé.