

# Structure de données II

## Rapport de projet

HUYLENBROECK Florent  
DACHY Corentin  
BA3 Info

15 avril 2019

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Notions théoriques . . . . .	3
1.2	Enoncé du problème . . . . .	3
<b>2</b>	<b>Calcul de la complexité de l'algorithme du peintre</b>	<b>4</b>
<b>3</b>	<b>Comparaison des heuristiques</b>	<b>5</b>
<b>4</b>	<b>Mode d'emploi du logiciel de test</b>	<b>7</b>
<b>5</b>	<b>Conclusion</b>	<b>8</b>

# 1 Introduction

## 1.1 Notions théoriques

- Une **scène**  $S$  est un ensemble de segments colorés dans un repère fini.
- Un **BSP** (*Binary Space Partition* ou *partition binaire de l'espace*)  $\tau$  est un arbre binaire permettant de représenter une scène en fragmentant celle-ci en sous-espaces convexes, lesquels contiennent au plus un segment. Son fonctionnement est le suivant :

Soit  $v$  la racine de  $\tau$ .

- Si  $v$  est une feuille, alors  $S(v)$  (sous-ensemble de segments de la scène contenus dans  $v$ ) contient 0 ou 1 segment.
- Sinon  $v$  est un noeud interne du BSP. Dans ce cas,  $v$  contient une équation de droite de la forme  $\ell \equiv ax + by + c = 0$  et un ensemble  $S(v)$   
Le fils gauche  $v^-$  de  $v$  sera la racine du sous-BSP  $\tau^-$  décrivant la partie de la scène située sous la droite  $\ell$ , et le fils droit  $v^+$  de  $v$  sera la racine du sous-BSP  $\tau^+$  décrivant la partie de la scène située au-dessus de la droite  $\ell$ .  
 $S(v)$  contient les segments totalement inclus à la droite  $\ell$ . Leur nombre n'est pas limité comme dans les feuilles.

Si une droite de séparation  $\ell_v$  d'un noeud interne  $v$  intersecte un segment en un point autre qu'une extrémité, ce segment est coupé en deux parties et celles-ci sont ajoutées dans  $\tau^-$  et  $\tau^+$  en conséquence.

- La **taille**  $t$  d'un BSP est le nombre total de segments contenus dans chacun des noeuds de celui-ci.
- Une **heuristique** de construction d'un BSP intervient au moment de choisir le segment de la scène par lequel passera la prochaine droite de séparation  $\ell$ .

## 1.2 Enoncé du problème

Pour ce projet, il nous a été demandé à partir de fichier *.txt* au format Scene2D de :

- Construire un BSP de la scène décrite dans le fichier.
- Appliquer l'algorithme du peintre sur ce BSP.
- Fournir une application ayant deux modes, console et graphique, afin de tester différentes heuristiques de construction pour les BSP. L'application console doit aussi mesurer la hauteur/taille du BSP, le temps CPU pris par la construction et par l'exécution de l'algorithme du peintre (*Voir section suivante*).
- Trois heuristiques nous ont été imposées pour ce projet :
  - RANDOM - Le segment est choisi au hasard parmi  $S(v)$ .
  - ORDERED - Le segment choisi est le premier dans  $S(v)$ .
  - FREE\_SPLIT - Le segment choisi est le premier dans  $S(v)$  dont les deux extrémités sont chacune sur une droite de séparation  $\ell$ . Si aucun segment satisfaisant ces conditions n'est trouvé dans  $S(v)$ , alors le segment  $y$  est choisi au hasard.

## 2 Calcul de la complexité de l'algorithme du peintre

Introduisons la notion d'*oeil*  $e$  dans une scène  $S$ . Celui-ci servira de point de vue des éléments de la scène. L'algorithme du peintre est un algorithme simple de rendu d'une scène sur un objet de dimension inférieure. (*Par exemple, afficher la vue en 2D par un oeil d'une scène en 3D.*) Il permet de déterminer quels fragments de la scène sont visible par l'oeil en fonction de sa position dans le repère.

---

### Algorithm 1 painter's algorithm

---

**Entrées :**     $v$  : Noeud, racine du BSP.  
                   $e$  : Point, position de l'oeil dans la scène.

**Sorties :**         /

**Effets :**            Les appels à la fonction de dessin sont effectués dans l'ordre correct d'affichage pour l'oeil  $e$ .

```

1: procedure PAINTERSALGORITHM( $A, B$ )
2:   if  $v$  est une feuille then
3:     Dessiner les fragments de  $S(v)$ 
4:   else
5:     if  $e$  est au-dessus de la droite de séparation  $\ell$  de  $v$  then
6:       PAINTERSALGORITHM( $v^-, e$ )
7:       Dessiner les fragments de  $S(v)$ 
8:       PAINTERSALGORITHM( $v^+, e$ )
9:     else if  $e$  est en dessous de la droite de séparation  $\ell$  de  $v$  then
10:      PAINTERSALGORITHM( $v^+, e$ )
11:      Dessiner les fragments de  $S(v)$ 
12:      PAINTERSALGORITHM( $v^-, e$ )
13:     else
14:       PAINTERSALGORITHM( $v^+, e$ )
15:       PAINTERSALGORITHM( $v^-, e$ )
16:     end if
17:   end if
18: end procedure

```

---

**NOTE :** la fonction *Dessiner les fragments de  $S(v)$*  est en  $O(s)$  où  $s$  est le nombre de segments dans  $S(v)$ .

La complexité de l'algorithme du peintre va dépendre de la taille de  $\tau$ . Elle va donc dépendre de la fragmentation des segments de  $S$  lors de la construction de  $\tau$ .

On remarque en effet que tout le BSP est parcouru par l'algorithme car dans chaque cas, en partant d'un noeud interne, on applique l'algorithme sur ses deux fils. La récursion s'arrête uniquement lorsqu'on arrive à une feuille. Nous sommes donc pour l'instant en  $O(n)$  où  $n$  est le nombre de noeud de  $\tau$

Ensuite, à chaque noeud, l'algorithme dessine tous les segments contenus dans  $S(v)$  (sauf quand  $e$  se situe sur la droite de séparation  $\ell$  du noeud, auquel cas il n'est pas nécessaire de dessiner les fragments dans  $S(v)$  car ils ne seront pas visible depuis  $e$ ). Nous avons donc une complexité de  $O(n \cdot s)$  ce qui donne, par la manière dont nous avons défini la taille d'un BSP,  $O(t)$ .

### 3 Comparaison des heuristiques

Pour commencer la comparaison des heuristiques, regardons au taux de fragmentation des segments lors de la construction du BSP. Calculons ce taux comme ceci :

$$\text{Taux} = \frac{t}{\text{nombre de segments de la scène}}$$

Voici les résultats pour chaque scène :

Scène (nombre de segments)	RANDOM	ORDERED	FREE SPLITS
RectangleSmall (660)	1.04	1.01	1.04
RectangleMedium (1800)	1	1	1
RectangleLarge (5940)	1.01	1	1.01
RectangleHuge (16800)	1	1	1
RandomSmall (2939)	1.75	1.21	1.751
RandomMedium (11045)	1.75	1.11	1.75
RandomLarge (23504)	1.7	1.14	1.7
RandomHuge (47017)	1.66	1.06	1.66
EllipsesSmall (200)	1.05	1	1.05
EllipsesMedium (720)	1.04	1	1.04
EllipsesLarge (4500)	1.01	N/A	1.01

On remarque une similarité entre les heuristiques RANDOM et FREE\_SPLITS. Cette similarité avait déjà été annoncée dans l'ouvrage de référence pour ce projet, ce résultat est donc normal.

On y observe aussi un net avantage de l'heuristique ORDERED par rapport aux deux autres. Cette heuristique serait donc une bonne candidate pour construire des BSP sur lesquels l'algorithme du peintre est très rapide.

Regardons maintenant au temps CPU (en ms) de construction des BSP :

Scène (nombre de segments)	RANDOM	ORDERED	FREE SPLITS
RectangleSmall (660)	1.708	3.480	1.322
RectangleMedium (1800)	3.644	4.890	3.277
RectangleLarge (5940)	17.525	54.432	17.210
RectangleHuge (16800)	83.955	426.679	81.534
RandomSmall (2939)	10.090	13.802	10.371
RandomMedium (11045)	63.745	149.688	60.021
RandomLarge (23504)	204.669	778.286	197.410
RandomHuge (47017)	645.144	2313.319	671.375
EllipsesSmall (200)	0.963	2.012	0.599
EllipsesMedium (720)	2.99	26.976	2.572
EllipsesLarge (4500)	37.554	N/A	34.945

Cette fois-ci, on remarque un retard de l'heuristique ORDERED par rapport aux deux autres (lesquelles sont à nouveau similaires).

Pour finir, intéressons nous au temps CPU (en ms) écoulé lors de l'application de l'algorithme du peintre sur le BSP, pour un oeil  $e$  situé en  $(0, 0)$  dans le repère.

<b>Scène (nombre de segments)</b>	<b>RANDOM</b>	<b>ORDERED</b>	<b>FREE_SPLITS</b>
RectangleSmall (660)	0.057	0.034	0.029
RectangleMedium (1800)	0.075	0.050	0.045
RectangleLarge (5940)	0.250	0.244	0.237
RectangleHuge (16800)	0.635	0.681	0.553
RandomSmall (2939)	0.302	0.065	0.306
RandomMedium (11045)	1.412	0.133	1.251
RandomLarge (23504)	3.166	0.357	2.806
RandomHuge (47017)	5.620	0.287	6.156
EllipsesSmall (200)	0.035	0.014	0.011
EllipsesMedium (720)	0.057	0.064	0.032
EllipsesLarge (4500)	0.426	N/A	0.413

L'heuristique FREE\_SPLITS est légèrement plus avantageuse sur les scènes structurées (rectangles, ellipses) mais l'heuristique qui se démarque le plus est ORDERED sur les scènes aléatoires, conformément à nos attentes lors de l'observation du taux de fragmentation.

## 4 Mode d'emploi du logiciel de test

### Mode graphique

Ouvrir un terminal, se déplacer dans le dossier *code* du logiciel et exécuter la commande :

```
$ ant testgui
```

Le logiciel consiste en une fenêtre séparée en deux parties. La partie du haut permet de visualiser le BSP et celle du bas ce que voit l'oeil. Afin d'ouvrir un BSP, il faut se rendre dans le menu *Actions* → *Open* et choisir un fichier *Scene2D* valide. Il est possible de changer l'heuristique de construction via le menu *Heuristics*.

### Mode console

Ouvrir un terminal, se déplacer dans le dossier *code* du logiciel et exécuter la commande :

```
$ ant test
```

Il est possible de spécifier les conditions de départ des test grâce aux paramètres suivants :

- *path* - Le chemin jusqu'au fichier *Scene2D*. La valeur par défaut est "octogone.txt".
- *n* - Le nombre de répétitions du test. La valeur par défaut est 100.
- *ex* - L'abscisse de l'oeil. La valeur par défaut est 0.
- *ey* - L'ordonnée de l'oeil. La valeur par défaut est 0.

Pour modifier ces paramètres, il faut utiliser le modificateur *-D*.

```
$ ant test -Dpath=somepath.txt
```

Pour modifier plusieurs paramètres, plusieurs *-D* sont nécessaires.

```
$ ant test -Dpath=somepath.txt -Dn=1000 -Dex=2.718 -Dey=3.141
```

### Javadoc

Ouvrir un terminal, se déplacer dans le dossier *code* du logiciel et exécuter la commande :

```
$ ant doc
```

## 5 Conclusion

### Difficultés rencontrées

Grâce à l'ouvrage de référence qui nous a été donné avec l'énoncé du projet, comprendre les structures d'arbres BSP n'a pas été difficile. L'exercice préliminaire nous a demandé une petite réflexion mais l'intuition nous est vite venue.

En ce qui concerne le projet principal, l'implémentation des BSP n'a pas été difficile. La partie la plus chronophage a été l'interface graphique, et plus particulièrement la fonction *Dessiner les fragments de  $S(v)$*  de l'algorithme du peintre, décrit à la deuxième section.

### Résultat de la comparaison des différentes heuristiques

En ce qui concerne les heuristiques, ORDERED est la meilleure parmi les trois qu'il nous a été demandé d'implémenter. Le temps de construction des BSP selon cette heuristique est un peu plus long, mais une fois construit, l'application de l'algorithme du peintre se fait beaucoup plus rapidement.

Dans nos tests, ce gain de performance était surtout visible sur les scènes *random*. Cela renforce l'idée que ORDERED est meilleure que les deux autres car en pratique, les scènes ont un aspect aléatoire et non pas structuré.

### Apports

Un point intéressant de ce projet est qu'il est un exemple concret de l'invention d'une structure de donnée pour résoudre un problème. Nous avons en effet tous déjà expérimenté des milieux en 3 dimensions, et il est donc facile de se rendre compte des problèmes que ces structures ont permise de régler.

Lors de futurs projets, il sera intéressant de chercher à résoudre des problèmes en les structurant correctement, avant de penser à concevoir un algorithme de résolution.