

Projet API : Algorithme du peintre

DUNAND Quentin
GATTAZ Rémi
VIAL-GRELIER Aymeric

8 Décembre 2014



Table des matières

1	Introduction	3
2	Liste des paquetages	3
2.1	machine_seq	3
2.2	off_file	3
2.3	off_struct	4
2.4	ps_file	4
2.5	tri_packet	4
3	Utilisation de l'application	5
3.1	Compilation des sources	5
3.2	algo_peintre	5
3.3	algo_peintre_perf	5
4	Tests	6
5	Evaluation de la mémoire	7
6	Evaluation des performances	8
7	Difficultés rencontrées	9
8	Perspectives d'amélioration	10
8.1	Gestion des arguments	10
8.2	Evaluation des performances	10

1 Introduction

Voici le compte rendu du projet "*Algorithme du peintre*". Vous trouverez ci-joint à ce compte rendu le répertoire "source" dans lequel se trouveront l'intégralité des sources de ce projet.

1. Introduction
2. Liste des paquetages
3. Utilisation de l'application
4. Tests
5. Evaluation de la mémoire
6. Evaluation des performances
7. Difficultés rencontrées
8. Perspectives d'amélioration

2 Liste des paquetages

Voilà la liste des paquetages dans le projet présenté par ordre alphabétique

2.1 machine_seq

Ce paquetage est l'implémentation d'une machine séquentielle utilisée pour parcourir une donnée de type `pForme_List_T` (voir 2.3).

Description des procédures du paquetage :

- `demarrer` : Initialise la machine séquentielle
- `finDeSequence` : Retourne le booléen `true` si la machine séquentielle à fini de parcourir les données
- `avancer` : Change l'élément courant par le suivant
- `elementCourant` : Retourne l'élément courant

Ce paquetage dépend du paquetage `off_struct`.

2.2 off_file

Ce paquetage contient la procédure prenant en entrée un fichier `.off` et le parcourt pour rendre en sortie un pointeur sur un tableau de points et une liste chaînée de formes.

Description de la procédure du paquetage :

- `file_to_sommets_formes` : procédure qui permet de retourner le pointeur vers le tableau de points et la liste de formes

Ce paquetage dépend du paquetage `off_struct`.

2.3 off_struct

Ce paquetage contient la description de l'architecture des données utilisées tout au long du projet.

Description des types :

- Sommet : type composé de 3 flottants
- Sommet_T : tableau de Sommets
- pSommet_T : pointeur sur un Sommet_T
- Integer_T : tableau d'entiers
- pInteger_T : pointeur sur un Integer_T
- Forme : type composé d'un pSommet_T et d'un entier
- Forme_List_Element : liste chaînée de Formes
- Forme_List_T : tableau de Forme_List
- pForme_List_T : pointeur sur un Forme_List_T

Description des procédures :

- getMinMaxXYZ : retourne les valeurs maximums et minimums prises sur les trois axes des Sommets d'un tableau de Sommets
- libererSommet_T : libération d'un tableau de Sommets
- libererInteger_T : libération d'un tableau d'entiers
- libererForme_List : libération d'une liste de formes
- libererForme_List_T : libération d'un tableau de listes de formes

2.4 ps_file

Ce paquetage contient la procédure permettant de créer le fichier .ps

Description des procédures :

- forme_list_t_to_ps : prend en paramètre une chaîne pour le nom du fichier .ps à créer, une liste de formes à écrire dans le fichier, le tableau des sommets référencés dans les formes, la taille (entier) de la liste des formes, 4 entiers pour les maximums/minimums de X et Y.

Ce paquetage dépend du paquetage off_struct et machine_seq.

2.5 tri_packet

Ce paquetage contient la procédure permettant de trier selon l'algorithme de tri par paquet la liste de formes. La procédure retourne ensuite un tableau de liste.

- triParPaquet : prend en entrée une liste de formes, un tableau contenant les Sommets des formes, un entier pour le nombre total de formes, le minimum et le maximum de Z. Retourne en sortie le tableau trié de listes triées.

Ce paquetage dépend du paquetage off_struct.

3 Utilisation de l'application

3.1 Compilation des sources

Le dossier source contient un makefile. Pour construire les executables, il faut donc executer la commande suivante :

```
make
```

Si la compilation s'est bien déroulée, les exécutables `algo_peintre` et `algo_peintre_perf` ont été créés.

3.2 algo_peintre

`algo_peintre` est un programme qui prend en entrée un fichier `off` et qui crée un fichier `ps`, contenant la projection sur `z` de la forme définie dans le fichier `.off`.

`algo_peintre` se lance de la façon suivante :

```
algo_peintre path/to/off/file path/to/ps/file
```

Pour visualiser le fichier `.ps`, nous vous conseillons d'utiliser le programme `gv`.

3.3 algo_peintre_perf

`algo_peintre_perf` est un programme qui permet d'évaluer les performances de `algo_peintre`. Prennant en entrée un fichier `off`, ce programme va exécuter l'algorithme de `algo_peintre` 1000 fois et stocker les performances de chaque exécutions dans un fichier `csv`.

`algo_peintre_perf` se lance de la façon suivante :

```
algo_peintre_perf path/to/off/file path/to/csv/file
```

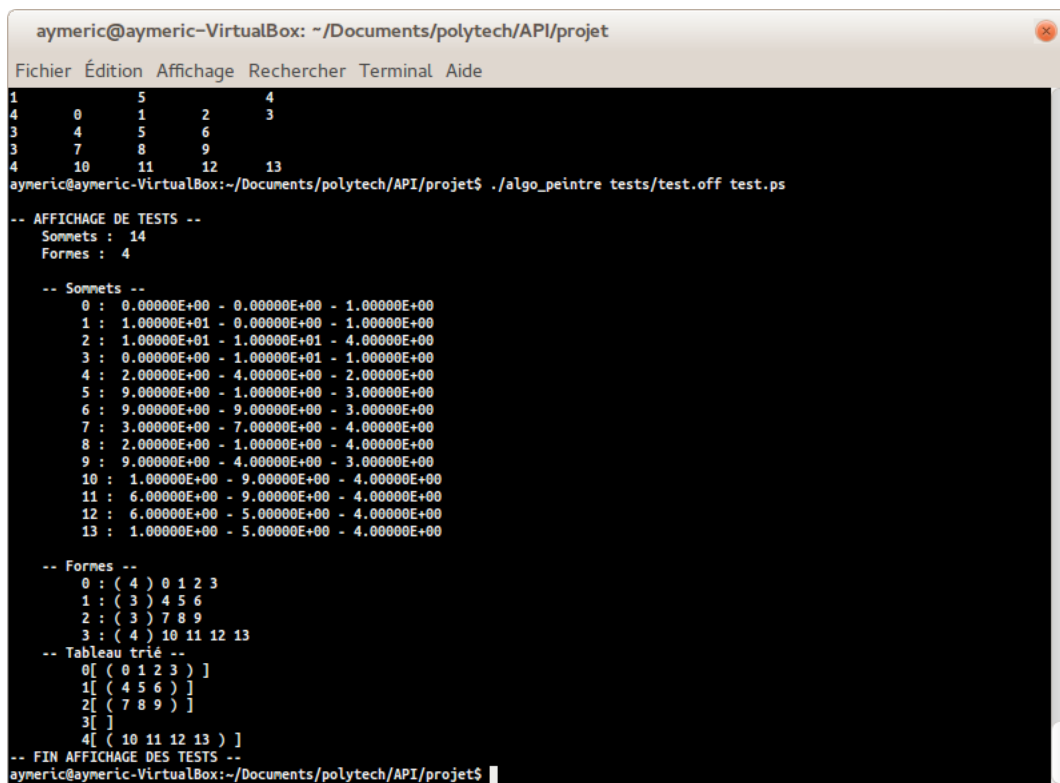
4 Tests

Pour tester le programme nous avons constitué trois fichiers de test nommés : single.off, empty.off et test.off.

Le fichier single.off contient trois points et trois formes qui sont composées uniquement de un point. La génération du fichier .ps fonctionne et retourne un fichier .ps qui ne contient que l'entête et la queue de n'importe quel fichier ps.

Le fichier empty.off est un fichier .off vide. C'est à dire qu'il ne contient aucun Sommet et aucune forme. Pour autant, le fichier est bien formé et contient donc les deux premières lignes de tout fichier off. Encore une fois, la génération du fichier .ps fonctionne et retourne un fichier .ps qui ne contient que l'entête et la queue de n'importe quel fichier ps.

Le fichier test.off est quand à lui un fichier qui nous a permis de tester notre tri. En effet, il contient des valeurs basiques et nous avons donc pu déterminer la valeur attendue du tri et donc vérifier si le tri par paquet était correct. Le résultat obtenu était celui conforme à nos attentes. Notre tri semble donc fonctionner.



```
aymeric@aymeric-VirtualBox: ~/Documents/polytech/API/projet
Fichier Édition Affichage Rechercher Terminal Aide
1      5      4
4      0      1      2      3
3      4      5      6
3      7      8      9
4      10     11     12     13
aymeric@aymeric-VirtualBox:~/Documents/polytech/API/projet$ ./algo_peintre tests/test.off test.ps

-- AFFICHAGE DE TESTS --
  Sonnets : 14
  Formes : 4

-- Sonnets --
  0 : 0.00000E+00 - 0.00000E+00 - 1.00000E+00
  1 : 1.00000E+01 - 0.00000E+00 - 1.00000E+00
  2 : 1.00000E+01 - 1.00000E+01 - 4.00000E+00
  3 : 0.00000E+00 - 1.00000E+01 - 1.00000E+00
  4 : 2.00000E+00 - 4.00000E+00 - 2.00000E+00
  5 : 9.00000E+00 - 1.00000E+00 - 3.00000E+00
  6 : 9.00000E+00 - 9.00000E+00 - 3.00000E+00
  7 : 3.00000E+00 - 7.00000E+00 - 4.00000E+00
  8 : 2.00000E+00 - 1.00000E+00 - 4.00000E+00
  9 : 9.00000E+00 - 4.00000E+00 - 3.00000E+00
 10 : 1.00000E+00 - 9.00000E+00 - 4.00000E+00
 11 : 6.00000E+00 - 9.00000E+00 - 4.00000E+00
 12 : 6.00000E+00 - 5.00000E+00 - 4.00000E+00
 13 : 1.00000E+00 - 5.00000E+00 - 4.00000E+00

-- Formes --
  0 : ( 4 ) 0 1 2 3
  1 : ( 3 ) 4 5 6
  2 : ( 3 ) 7 8 9
  3 : ( 4 ) 10 11 12 13

-- Tableau trié --
  0[ ( 0 1 2 3 ) ]
  1[ ( 4 5 6 ) ]
  2[ ( 7 8 9 ) ]
  3[ ]
  4[ ( 10 11 12 13 ) ]
-- FIN AFFICHAGE DES TESTS --
aymeric@aymeric-VirtualBox:~/Documents/polytech/API/projet$
```

FIGURE 1 – trace de l'exécution avec tests/test.off

A la suite de l'exécution de ces tests et la visualisation des fichiers .ps produits avec des fichiers .off plus complexes, nous avons déterminé que notre programme fonctionnait correctement.

5 Evaluation de la mémoire

Pour ce projet nous pensions important de tester notre gestion de la mémoire. Par conséquent nous avons effectué des recherches pour trouver un utilitaire qui nous permettrait de vérifier ceci.

Nous avons trouvé **valgrind** qui permet entre autre de tester la mémoire allouée et libérée lors de l'exécution d'un programme.

Le test donne la trace suivante :

```
==30854==  
==30854== HEAP SUMMARY:  
==30854==    in use at exit: 0 bytes in 0 blocks  
==30854== total heap usage: 23,232 allocs, 23,232 frees, 873,148 bytes allocated  
==30854==  
==30854== All heap blocks were freed -- no leaks are possible  
==30854==  
==30854== For counts of detected and suppressed errors, rerun with: -v  
==30854== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```

FIGURE 2 – trace de valgrind

Nous pouvons donc affirmer que l'exécution du programme se déroule sans fuite de mémoire.

6 Evaluation des performances

Afin de tester les performances du programme nous avons rajouté des instructions `ada.Real_time` afin de vérifier le temps d'exécution des différentes fonctions. Nous sommes conscient du fait que ces valeurs sont fortement liées à la capacité de la machine choisie pour effectuer ces tests. Notre programme est composé en 5 fonctions principales dont nous relevons les temps d'exécutions. Les fonctions sont les suivantes :

1. lecture du fichier `.off`
2. récupération des maximums en x,y et z
3. tri par paquet
4. écriture du fichier `.ps`
5. déallocation

La complexité des fonctions 1,2,4,5 est de n puisque l'on parcourt les données en entrée une seule et unique fois. De plus, puisque l'on considère que les valeurs à trier sont réparties de façon homogène, le tri par paquet à lui aussi une complexité de n . La complexité générale de notre programme est donc n .

Pour obtenir un résultat proche de la réalité, le test de performance exécute le programme 1000 fois. Grâce au grand nombre de valeurs obtenues nous pouvons nous faire une idée globale de la rapidité d'exécution du programme.

Le fichier ci-dessous est un extrait du fichier `.csv` produit par le programme de test de performance sur le fichier `cow.off`.

Les temps d'exécutions présents dans le fichier `.csv` sont exprimés en ms.

	A	B	C	D	E	F	G	H
1	nbSommets	nbFormes	TTotat	TLectureOff	TGetMax	TTri	TEcriturePS	TLibération
2	2904	5804	43	9	0	1	32	1
3	2904	5804	41	10	0	1	29	1
4	2904	5804	41	9	0	1	30	1
5	2904	5804	41	9	0	1	29	1
6	2904	5804	39	10	0	1	27	1
7	2904	5804	45	8	0	1	35	1
8	2904	5804	36	8	0	1	26	1
9	2904	5804	36	8	0	1	26	1
10	2904	5804	37	8	0	1	27	1
11	2904	5804	45	8	0	1	35	1
12	2904	5804	37	8	0	1	27	1
13	2904	5804	37	8	0	1	27	1
14	2904	5804	42	12	0	1	28	1
15	2904	5804	41	8	0	1	31	1
16	2904	5804	41	8	0	1	31	1
17	2904	5804	38	8	0	1	28	1

FIGURE 3 – 8 premières exécutions sur `tests/models/cow.off`

7 Difficultés rencontrées

Nous n'avons pas eu de difficultés particulières au cours de ce projet. En revanche, nous avons rencontré une erreur que nous ne sommes toujours pas en mesure d'expliquer complètement.

Lors du tri par paquet, nous copions entièrement les éléments de la liste donnée en entrée pour former le tableau de liste de sortie. Cependant, lors de la première écriture de la fonction nous avons fait un erreur. En effet, des tableaux d'entiers dynamiquement alloués n'étaient pas recopiés mais la référence partagée. Cela liait donc le cycle de vie des deux objets.

Pendant la libération des espaces mémoires, nous n'avions cependant pas d'erreur. En effet, malgré le fait que nous libérions des tableaux d'entiers plusieurs fois, aucune exception ne survenait.

Tant que nous n'exécutons pas le programme plusieurs fois à la suite, le programme semblait fonctionner sans erreur. En revanche, lorsque nous avons créé le fichier `algo_peintre_perf` afin de faire des tests de performances, et donc que nous avons fait boucler le programme, nous avons été confronté à un problème. En effet, le processus se mettait dans l'état suspendu lors du premier appel à "new" de la seconde itération du programme.

Nous avons cherché sur des forums une explication possible. Mais c'est finalement grâce à l'outil `valgrind` que nous avons pu découvrir cette libération double qui causait cette anomalie.

Pour régler cette erreur, nous avons donc modifié notre paquetage de tri par paquets afin de ne plus lier les éléments de la liste d'entrée et les éléments du tableau de sortie.

Pour comprendre l'erreur, suite à ce constat, nous avons alors essayé de répliquer ce problème avec une structure de données plus simple. Nous avons pu constater que libérer plusieurs fois à la suite un même pointeur ne cause pas d'exception. Cependant, notre problème n'a pas été reproduit puisque nous n'avons pas réussi à reproduire la mise en pause du programme.

8 Perspectives d'amélioration

8.1 Gestion des arguments

Pour ne pas perdre de temps sur une problématique peu intéressante dans le cadre de ce projet, nous avons géré les arguments de l'application de façon très basique. Il serait cependant intéressant de les gérer en utilisant la norme `gnu`. On pourrait alors obtenir un programme avec les spécifications d'utilisation suivantes :

```
Usage: algo_peintre [-i file1] [-o file2] [--log-perf]
```

```
-i --input-file file1    : use file1 as the input file  
-o --output-file file2   : use file2 as the output file  
--log-perf number file3  : return in stdout the  
                           performance of the programm
```

8.2 Evaluation des performances

Avec la version du programme définie au point précédent, il ne serait pas nécessaire d'avoir une version `algo_peintre_perf` du programme pour effectuer les tests de performances du programme. En effet, il suffirait de faire un script qui construirait un fichier `.csv` en effectuant plusieurs fois l'exécution du programme avec le paramètre `-log-perf`.