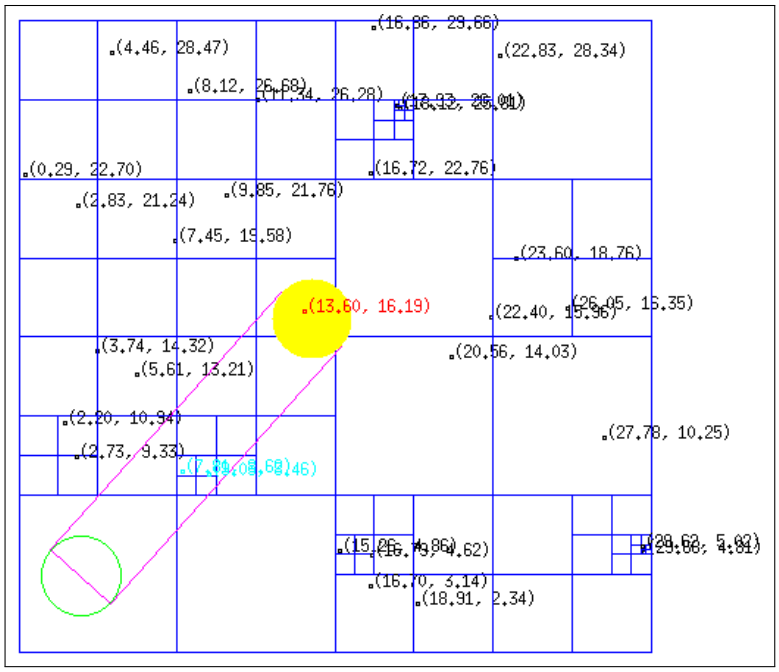
IPRF - « QuadTree »

Rapport

Quentin Barrand
quentin@quba.fr

3 mai 2015

Ce document a été réalisé dans le cadre du projet final du module IPRF « Programmation fonctionnelle » de la formation FIPA de l'ENSIIE.
Dans ce projet, nous étudions l'opportunité d'utiliser une structure de données de type arbre 4-aire (« QuadTree ») pour stocker des points appartenant à un repère orthonormé ; le but est de détecter des collisions entre un disque en déplacement et des obstacles matérialisés par ces points.
Le QuadTree étudié dans ce projet n'est pas extensible (les bornes du plan donné à l'initialisation sont fixes et positives) et possède une taille de panier égale à 1, ce qui signifie qu'un seul objet peut être stocké par nœud terminal.
Ce projet est l'occasion d'étudier plusieurs stratégies de stockage d'objets en mémoire et d'approfondir l'utilisation de la bibliothèque standard d'OCaml pour des usages graphiques.



Organisation du projet

Pour chaque section, les fonctions à écrire sont dans les fichiers `partX.ml` et le code interactif (tel que les tests graphiques) dans les fichiers `partX.test.ml`. Cette organisation permet d'importer le code des sections précédentes à l'aide de directives `#use` sans provoquer des appels de fonction inattendus.
Le rendu est donc constitué des fichiers suivants :

```
/
├── bonus.ml ..... Bonus
├── coord.ml ..... Partie 1
├── display.ml ..... Fourni par le sujet
├── part1.test.ml ..... Partie 1 - tests
├── part2.test.ml ..... Partie 2 - tests
├── part3.ml ..... Partie 3
├── part3.test.ml ..... Partie 3 - tests
├── part4.ml ..... Partie 4
├── part4.test.ml ..... Partie 4 - tests
├── part5.ml ..... Partie 5
├── part5.test.ml ..... Partie 5 - tests
├── quadtree.ml ..... Partie 2
├── rect.ml ..... Partie 1
├── report.pdf ..... Le présent rapport au format PDF
├── report.tex ..... Le présent rapport au format LaTeX
├── simulation1.ml ..... Fourni par le sujet
└── simulation2.ml ..... Fourni par le sujet
```

1 Échauffement sur les rectangles

Question 1. Voir `rect.ml`.

Question 2. Voir `rect.ml`.

Question 3. Voir `rect.ml`.

Question 4. Voir `rect.ml`.

Question 5. Voir `rect.ml`.

Question 6. Voir `rect.ml`.

2 La structure de données QuadTree

Question 7. Pour stocker une collection de points, on pourrait utiliser quatre grandes familles de structures de données :

Ensembles ou listes On stocke les points les uns à la suite des autres dans une liste.
Occupation mémoire : minimale ($n \times taille(n)$).
Algorithmes d'accès : très peu efficaces (parcours de la liste - $\mathcal{O}(n)$).

Dictionnaires On stocke les points dans un dictionnaire dont les clés sont les coordonnées des points.
Occupation mémoire : modérée (on considère que pour éviter la plupart des

collisions sur la fonction de hachage, il est nécessaire de d'utiliser un ensemble environ 5 fois plus grand que le nombre d'éléments à stocker – $5 \times n \times taille(n)$).
Algorithmes d'accès : très efficaces lorsque l'on connaît les coordonnées exactes du point (temps quasi-constant qui dépend de la fonction de hachage choisie et du nombre d'éléments possédant la même clé), peu efficaces dans les autres cas (recherche de tous les points contenus dans une restriction du plan initial par exemple) puisqu'il est nécessaire de parcourir tout le dictionnaire ($\mathcal{O}(n)$).

Matrices On stocke un tableau à deux dimensions en mémoire, et on stocke le point à ses coordonnées dans le tableau.
Occupation mémoire : très importante (de l'ordre de $((Abs_{max} - Abs_{min}) \times (Ord_{max} - Ord_{min})) \times taille(n)$).
Algorithmes d'accès : très efficaces (temps quasi-constant qui dépend seulement du nombre d'éléments possédant la même clé).

Arbres n-aires Les arbres permettent de combiner une utilisation mémoire modérée et une recherche efficace. L'emploi du QuadTree, arbre 4-aire, est particulièrement justifié pour travailler avec des points 2D, puisqu'il découpe si nécessaire le plan en rectangles successifs et que tout point est logiquement contenu dans un rectangle. Ainsi, la recherche de tous les points contenus dans une restriction du plan initial est très rapide.
Occupation mémoire : modérée ($\leq 4 \times n \times taille(n)$).
Algorithmes d'accès : efficaces ($\mathcal{O}(\log n)$).

Question 8. Voir `quadtree.ml`.

Question 9. Voir `quadtree.ml`.

Question 10. Voir `quadtree.ml`.

Question 11. Voir `quadtree.ml`.

Question 12. Voir `quadtree.ml`.

Question 13. Voir `quadtree.ml`.

Intuitivement, on pourrait être tenté d'utiliser `list_of_quadtree`, puis de supprimer l'élément choisi dans la liste, puis de faire appel à `quadtree_of_list` pour reconstruire un QuadTree à partir de cette nouvelle liste ; la fonction `list_remove` implémente cette méthode.
Pour éviter l'opération en $\mathcal{O}(n)$ qui consiste à reconstruire le QuadTree après avoir supprimé un objet dans la liste, on peut écrire une fonction `clean_qt` qui devrait permettre d'avoir une simplification plus performante du QuadTree que si l'on utilisait la méthode décrite plus haut. Malgré de nombreux tests, nous n'avons pas trouvé de cas dans lesquels cette fonction produirait des résultats incorrects.

3 Représentation graphique d'un QuadTree et tests

Question 14. Fonctions de `display.ml` :

`draw_data` Dessine une donnée stockée dans une feuille `Leaf of coord * 'a` d'un QuadTree. La fonction de conversion de `'a` vers `String` ainsi que la couleur de dessin doivent être passées en paramètres.

`draw_quadtree` Dessine récursivement un QuadTree. Commence par dessiner en bleu le rectangle dans lequel est contenu le QuadTree, puis, en fonction de la nature de l'élement `cell` :

— ne fait rien si l'élément est `Empty` ;
— fait appel à `draw_data` si l'élément est `Leaf` ;
— s'appelle récursivement pour chaque sous-QuadTree si l'élément est `Node`.

`wait_and_quit` Attend un appui sur une touche du clavier puis ferme la fenêtre de dessin.

`init` Initialise un environnement de dessin en fonction de la taille du rectangle passé en paramètre et ouvre la fenêtre. Retourne un triplet de `Float` nommé `dparams` dans le reste du projet et utilisé par toutes les fonctions de dessin.

`simple_test` Appelle la fonction `draw_quadtree` à l'aide du triplet `dparams` (obtenu par un précédent appel à la fonction `init`) et d'une fonction de conversion vers le type `String` pour dessiner le QuadTree passé en paramètre.

Question 15. Voir `part3.ml`.

La fonction `new_simple_test` prend en paramètre deux coordonnées `c` et `c'` et un entier `n`. Elle insère dans un QuadTree vide délimité par le rectangle formé par `c` et `c' n` objets dont les coordonnées sont obtenues aléatoirement à l'aide du module `Random`. Elle affiche le QuadTree ainsi complété, puis lors de l'appui sur une touche, le vide et l'affiche de nouveau. De cette façon, on peut vérifier que tous les objets sont bien supprimés.

Un appel à cette fonction avec l'insertion d'une centaine d'objets est disponible dans le fichier `part3.test.ml`.

Question 16. Voir `part3.ml`.

En utilisant `rect_length` et `rect_height`, dans la fonction `bad_draw_quadtree`, on observe que certains segments délimitant les bordures du QuadTree sont plus épais que d'autres (voir Fig. 1 et 2). Le problème peut être lié à plusieurs conversions de nombres flottants en entiers :

— Lorsque l'on utilise la fonction `draw_quadtree`, la conversion de nombre flottant en nombre entier a lieu une fois que le calcul sur les composantes flottantes `sx` / `sy`, `rect_left r` / `rect_right r` / `rect_top r` / `rect_bottom r` et `z` est terminé ; on a donc un seul arrondi.
— Lorsque l'on utilise la fonction `bad_draw_quadtree`, on ajoute le résultat arrondi en entier du calcul sur les deux composantes flottantes `z` et `rect_length r` / `rect_height r` à la composante entière déjà arrondie `x1` ou `x2` ; on fait alors deux arrondis, qui peuvent provoquer le dessin de traits non superposés lorsque l'on dessine plusieurs fois le même QuadTree. Ces traits proches mais non superposés apparaissent dans la fenêtre graphique comme des traits plus épais.

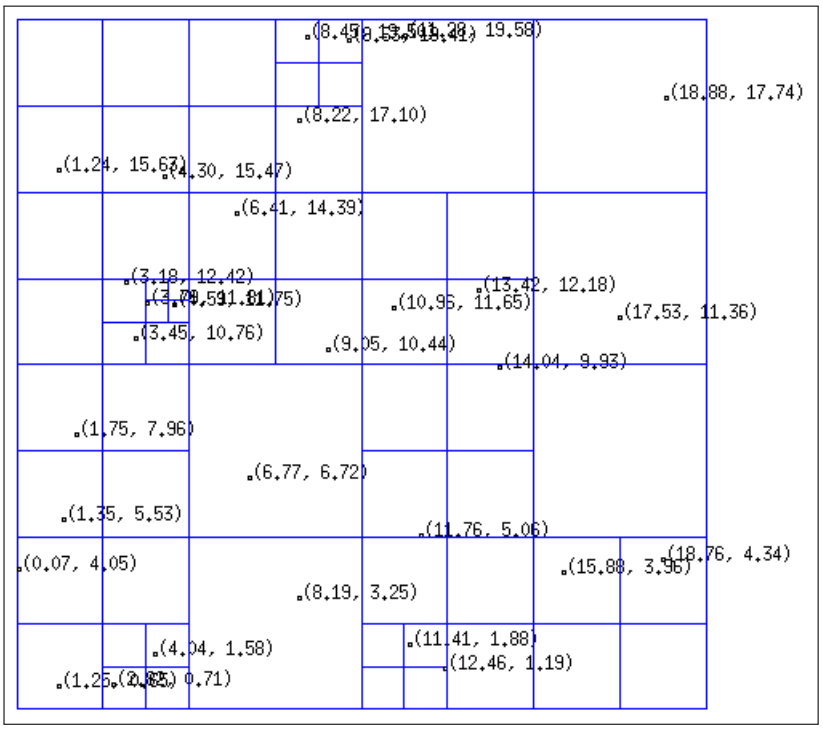


FIGURE 1 – Un exemple de dessin d'un QuadTree `q` en utilisant la fonction `simple_test` fournie par le sujet.

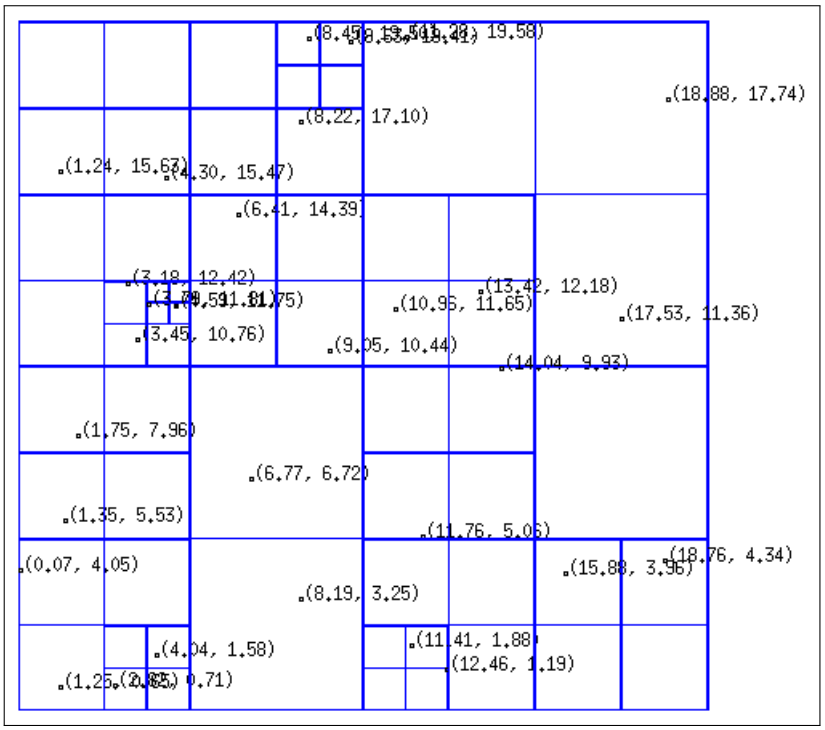


FIGURE 2 – Un exemple de dessin du même QuadTree `q` que dans la Fig. 1, en utilisant cette fois la fonction buggée `bad_simple_test`.

4 Placement du disque

Question 17. Voir `part4.ml`.

Question 18. Voir `part4.ml`.

Question 19. Voir `part4.ml`.

Question 20. Fonctions de `simulation1.ml` :

`get_point` Retourne les coordonnées du point sur lequel se trouve le pointeur de la souris lors du déclenchement du clic.

`get_disk` Retourne les coordonnées du centre et la longueur du rayon du cercle désigné par une action « clic - glisser » (début du clic - déplacement du pointeur - relâchement du pointeur).

`draw_disk` Dessine à disque d'une couleur spécifiée à des coordonnées spécifiées, plein ou non.

`draw_disk_with_collisions` Dessine un disque :
- de couleur jaune s'il recouvre des points du QuadTree (obtenus grâce à `collision_disk` de `part4.ml`), et fait appel à `draw_data` de `display.ml` pour dessiner ces points en rouge le cas échéant ;
- de couleur verte sinon.

`simulation_placement` Fait appel aux fonctions `init` et `draw_quadtree` de `display.ml` et aux fonctions `get_disk` et `draw_disk_with_collisions` pour dessiner un cercle défini à la souris par l'utilisateur et ses éventuelles collisions avec les points du QuadTree.

5 Déplacement du disque et détection de collision

Question 21. Voir `part5.ml`.

Question 22. Voir `part5.ml`.

Question 23. Voir `part5.test.ml`.

Fonctions de `simulation2.ml` :

`draw_trail_with_collisions` Dessine la zone survolée pendant le cercle pendant son déplacement depuis son point de départ vers son point d'arrivée. Fait appel à `collision_trail` pour obtenir une liste tous les points survolés. Redessine ensuite tous ces points en cyan.

`simulation_move` Idem que `simulation_placement` de `simulation1.ml`, sauf qu'après avoir défini un cercle, un second clic de souris utilise la fonction `draw_trail_with_collisions` pour déplacer le cercle vers cette seconde position. Un exemple de fonctionnement de cette fonction est disponible plus bas (voir Fig. 3).

Dans `part5.ml`, on trouvera également les fonctions `get_new_destination` et `new_simulation_move` qui implémentent le bonus de la question. Une illustration du fonctionnement de ces fonctions est disponible plus bas (voir Fig. 4).

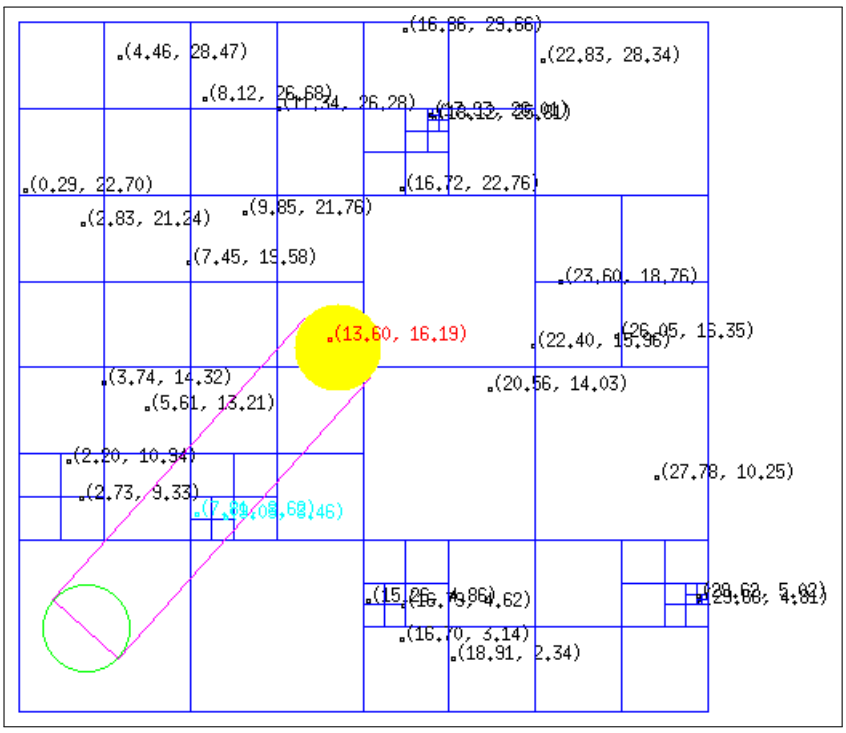


FIGURE 3 – Illustration du fonctionnement de la fonction `simulation_move`.

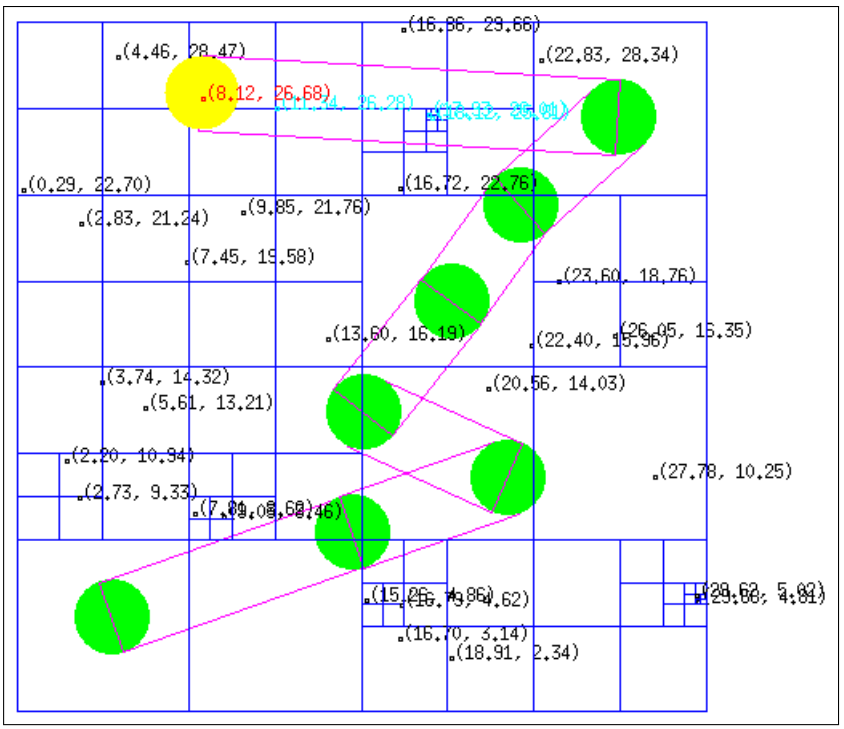


FIGURE 4 – Illustration du fonctionnement de la fonction bonus `new_simulation_move`.

Bonus

A l'origine écrite pour débugger `clip`, la fonction `graphical_clip` du fichier `bonus.ml` doit permettre de réaliser de façon graphique une restriction du plan, comme suggéré dans la Fig. 2 du sujet. Elle prend en paramètre un QuadTree et une fonction d'affichage des données du QuadTree ; cependant, bien qu'elle semble produire un QuadTree correct, il est impossible de l'afficher correctement.

Conclusion

L'arbre 4-aire (« QuadTree ») possède des propriétés très intéressantes pour le stockage de données géographiques dans un plan. D'une part, les algorithmes d'accès à une donnée en particulier sont raisonnablement rapides ($\mathcal{O}(\log n)$) ; d'autre part, l'empreinte mémoire est faible en comparaison avec d'autres structures de données de performance comparable. Les QuadTrees permettent également de restreindre assez facilement l'espace de recherche à une sous-partie du plan initial (voir la fonction `clip` dans la partie 4), et la visualisation de leur contenu pour le débuggage est assez intuitive.
Dans ce projet, nous avons vu un cas d'utilisation typique des QuadTrees : la détection de collisions. Les algorithmes développés plus haut sont couramment utilisés dans le monde des jeux vidéos par exemple puisqu'ils permettent de ne tester la collision de l'objet en mouvement qu'avec une partie des points du plan du plan complet, ce qui réduit considérablement les calculs.
Il est également possible de généraliser cette structure de données pour gérer des objets en trois dimensions dans l'espace, en utilisant des arbres 8-aires (« OcTrees »).