INGI1122 — Projet 2017 : Rendu Glouton

Optimisation du rendu de faces de voxels dans un jeu de type Minecraft

Guillaume Maudoux, Xavier Gillard et Charles Pecheur

31 mars 2017

Modalités

Le projet se divise en trois parties de deux semaines et s'étale donc sur six semaines. Le projet se fera par groupes de trois étudiants. Pensez à vous inscrire dans un groupe sur Moodle.

Chaque partie du projet se fera en deux temps : une semaine et demie pour rédiger votre solution, et trois jours pour commenter celle d'autres groupes. Il y aura donc trois échéances de remise de travaux à rendre et trois échéances de relecture croisée.

mardi 28 mars midi Remise de la partie 1 ;
vendredi 31 mars midi Relecture croisée de la partie 1 ;
mardi 25 avril midi Remise de la partie 2 ;
vendredi 28 avril midi Relecture croisée de la partie 2 ;
mardi 9 mai midi Remise de la partie 3 ;
vendredi 12 mai midi Relecture croisée de la partie 3 ;

Description du projet

Le but de ce projet est de spécifier, d'implémenter et de prouver la validité d'un programme de rendu de surfaces. Ce programme permettra de diminuer le nombre de rectangles nécessaires pour représenter un objet en utilisant des rectangles plus grands là où c'est possible. Votre programme devra fournir une solution *localement* optimale.

Cette application est largement inspirée du comportement des jeux vidéos utilisant les voxels [1], comme Minecraft, ou Antichamber. Comme documentation sur le sujet, on trouve l'excellent blog "Meshing in a Minecraft Game" [2] ou encore le projet GreedyMesh [3] sur Github.

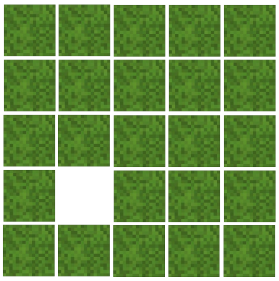
Dans ces jeux, un grand nombre de rectangles doivent être dessinés par la carte graphique (au moins un par face de cube visible). Une optimisation possible est de fusionner plusieurs rectangles adjacents en un rectangle plus grand.

Concrètement, l'algorithme à implémenter va tester toutes les paires de rectangles et les fusionner tant que c'est possible. La solution est localement optimale quand aucune paire de rectangles ne peut être exactement recouverte par un rectangle plus grand, mais n'est pas nécessairement optimale au sens du nombre de rectangles utilisés. Atteindre la solution optimale est un problème NP complet. Une telle solution n'est pas souhaitable dans un jeu vidéo ou l'on cherche à maximiser le nombre d'images par seconde. En revanche, optimiser localement le nombre de rectangles s'avère rentable.

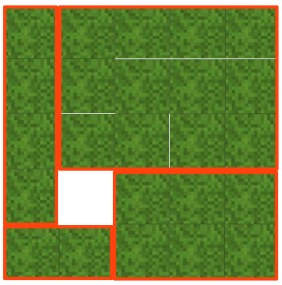
Par exemple, voici une forme à base de carrés pour laquelle on propose trois couvertures. La première est optimale, la seconde localement optimale et la dernière n'est pas suffisamment optimisée.

Il est à noter qu'une couverture ne contient jamais de rectangles qui se chevauchent.

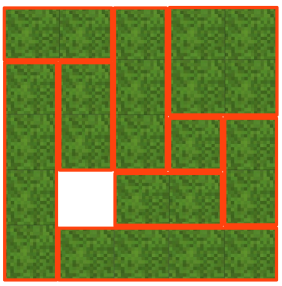
1. `https://fr.wikipedia.org/wiki/Voxel#Jeux_vidéo_utilisant_les_voxels`
2. `http://0fps.wordpress.com/2012/06/30/meshing-in-a-minecraft-game/`
3. `https://github.com/roboleary/GreedyMesh`



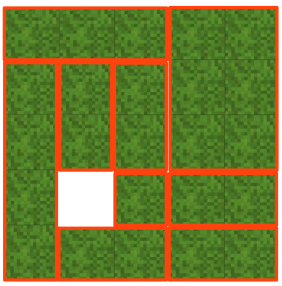
(a) Grille à couvrir



(b) Couverture optimale



(c) Couverture localement optimale



(d) Couverture à optimiser davantage

FIGURE 1 – Exemples de couvertures

Première partie

La description ci dessus reste volontairement vague. La première étape de ce projet consistera donc à établir la théorie du problème, puis à spécifier les méthodes à implémenter.

La théorie du problème Afin de comprendre et d'expliquer le problème posé, on vous demande de décrire formellement le problème et la tactique qui sera utilisée pour le résoudre. Il ne s'agit pas de programmer, mais bien d'analyser et de décomposer le problème. Vous devez décrire :

- le problème posé en termes formels et précis ;
- la stratégie utilisée pour le résoudre, en particulier la décomposition en sous-problèmes ;
- la représentation des données intermédiaires manipulées, et les opérations nécessaires sur cette représentation ;
- les fonctions, prédicats et propriétés utiles à la résolution.

Par exemple, le problème qui consiste à donner l'étendue d'un tableau a ($\max(a) - \min(a)$) peut se décomposer comme suit :

> Pour trouver l'étendue d'un tableau donné, nous allons d'abord trouver sa valeur maximale et sa valeur minimale, puis renvoyer leur différence.
>
> Pour calculer le maximum (resp. minimum) d'un tableau, nous allons le parcourir entièrement en mémorisant à chaque étape la plus grande (resp. petite) valeur rencontrée.

N'hésitez pas à vous aider de schémas et de dessins pour représenter vos idées, et à joindre ceux-ci à votre rapport. Vous pouvez aussi vous inspirer de la théorie du problème de la solution du devoir 2.

La spécification En plus de la théorie du problème, il vous est demandé de spécifier formellement les éléments du programme à développer. Votre programme devra comporter :

— une représentation pour les rectangles,

— une représentation pour les ensembles de rectangles, dont le type concret sera une classe `Couverture` contenant un tableau de rectangles.

— une fonction `canMerge(Rectangle, Rectangle): bool` qui détermine si deux rectangles a et b peuvent être fusionnés en un seul.

— une fonction `merge(Rectangle, Rectangle): Rectangle` qui prend deux rectangles et retourne un rectangle fusionné.

— une méthode `improve()` de la classe `Couverture` qui fusionne au moins deux rectangles, si c'est possible. Si cette méthode ne fusionne aucun rectangle, c'est que la `Couverture` est déjà localement optimale

— une méthode `optimize()` de la classe `Couverture` qui calcule une couverture localement optimale.

— une méthode `contains(x: int, y: int)` de la classe couverture, qui indique si la tuile à l'indice (x, y) est couverte.

Vous devez donc donner, sous forme mathématique (pas en Dafny) :

— un type abstrait de donnée pour les rectangles, le type concret correspondant, les invariants de représentation et la fonction d'abstraction,

— une classe `Couverture` qui représente un ensemble recouvert par des rectangles, le type abstrait correspondant et les fonctions *abs* et *ok*

— des spécifications formelles complètes pour les méthodes `canMerge`, `merge`, `improve`, `optimize` et `contains`.

Vous serez attentifs à développer les éléments demandés de manière complète, rigoureuse et claire. Pour la spécification, il est recommandé d'introduire des prédicats intermédiaires pour alléger vos pré- et post-conditions.

Encore une fois, il n'est pas nécessaire d'écrire du code pour cette partie. En particulier, il n'est pas demandé d'écrire le corps des fonctions ci-dessus.

Délivrables

Vous devez remettre **un document pdf** sur le site Moodle du cours, dans la partie *Projet : Partie 1*, pour l'échéance reprise en tête de ce document. Veillez à inclure en couverture l'identification du cours, de l'année, du projet et les noms, prénoms et NOMA des étudiants. La relecture croisée se fera également au départ du site Moodle, dès l'échéance de la soumission.

Deuxième partie

Maintenant que vous avez décortiqué précisément le problème posé, nous vous demandons de fournir en Dafny une implémentation complète de vos méthodes. Votre code devra être exécutable par dafny, et sera fourni avec **trois tests** montrant le bon fonctionnement du programme.

Pour cette phase-ci, **il ne faut pas spécifier le programme** au delà de ce qui est strictement nécessaire pour permettre à dafny de le compiler. En pratique, il faudra quand même spécifier que vous n'accédez pas à des références nulles, que les indices de vos tableaux sont dans les bornes, et que vos boucles se terminent. Comme nombre d'entre vous l'ont remarqué durant le TP, c'est beaucoup plus facile de faire accepter un programme à Dafny si on n'écrit pas de postconditions. C'est ce qui est demandé ici. La preuve complète, sur base des spécifications de la phase 1 fera l'objet de la phase 3.

Votre programme devra rester le plus fidèle possible à la description faite à la phase 1. Si vous décidez de vous en écarter, vous devrez justifier votre décision et montrer pourquoi il n'était pas possible d'implémenter ce que vous aviez décrit.

En pratique, il faut implémenter une classe Couverture avec un constructeur qui reçoit une liste de rectangles et une méthode optimize() qui optimise (localement) la liste de rectangles. Toutes les méthodes de la phase 1 doivent être implémentées dans votre programme, dans la classe Couverture ou en dehors, à l'exception de la méthode improve, qui n'est que fortement conseillée.

Un schéma de programme vous est fourni pour vous lancer, et vous montrer les notations nécessaires au projet. Pour pouvoir exécuter le code, vous devrez utiliser l'option "`/compile:3`" ou utiliser la tâche INGInious "Execute Dafny". Rise4Fun ne permet pas d'exécuter le code Dafny, seulement de le vérifier. Si vous n'arrivez pas à installer Dafny, pensez à demander son installation via le Doodle `https://beta.doodle.com/poll/irmub7gy24k5drvm`. Si, comme c'est le cas pour l'instant, trop peu de gens sont intéressés, nous considèrerons que vous êtes capables de vous débrouiller.

Votre code est votre rapport ! Pensez donc à bien le commenter pour ses futurs lecteurs. En particulier, toutes vos méthodes et fonctions doivent être accompagnées d'un commentaire décrivant leur comportement et leur utilité, c'est à dire leur spécification informelle. C'est là que vous devez aussi justifier les différences entre votre design et votre implémentation. Le fichier "`groupeXX.dfy`" est à remettre sur Moodle selon les mêmes modalités que la phase 1.