Rapport CG - TP 3

Startresse Guillard
startresse.guillard@gmail.com

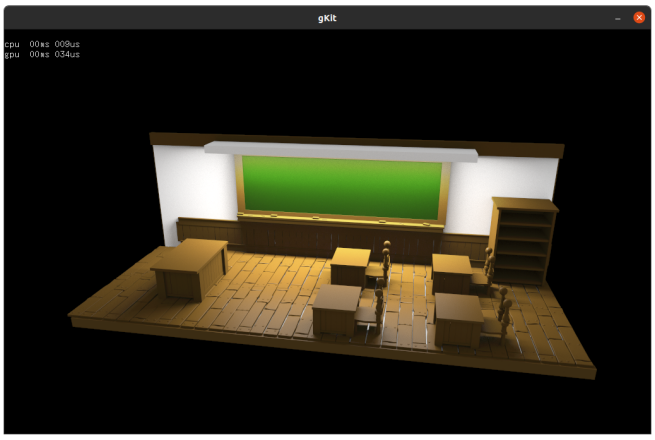

Fig. 1. Rendu final

I. Vidéo démo

https://youtu.be/ZRVq3qCNzlI

II. Travail effectué

Concrètement, ce TP est une version temps réelle de ce qui a été implémenté dans le TP2 de SI.

A. BVH cousu

Pour améliorer les performances de l'intersecteur, un BVH a été mis en place. Pour le parcours itératif du BVH, les noeuds ont été cousus en leur donnant un next() et un skip(). Il contiennent aussi une boite englobante et un indice de triangle (-1 : interne, > 0 : feuille).

B. Buffers et storage textures

Voici la liste des buffer et storage textures ainsi que leur utilité.

1) Triangle buffer: Contient tous les triangles de la scène. Les triangles sont 3 points 3D et 1 indice de matériel.

2) Material buffer: Contient toutes les matières de la scène.

3) Sources: Contient tous les indices des sources de lumière de la scène.

4) Nodes buffer: Contient les noeuds cousus pour le parcours du BVH. Le vecteur donné en initialisation est créé par la structure BVH à la fin du build().

5) Ray texture: Cette storage texture contient tous les rayons lancés par le premier shader. C'est une texture 3D (2D array) dont les profondeurs correspondent respectivement à l'origine du rayon, sa direction et ses information d'intersection (tmax, u, v, triangle_id). Ce rayon sera réutilisé par les autres shader pour avoir de la cohérence notamment au niveau de l'aléatoire (voir II-G1)

6) Seed texture: Contient les graines aléatoires utilisées pour l'aléatoire sur carte graphique. Ces valeurs sont initialisées aléatoirement (stl) au lancement du programme, et à chaque fin de shader, la valeur sera changée en accord avec les tirages aléatoires fait pendant le shader. Ainsi à la prochaine frame, la seed sera différente et tous les tirages aléatoires seront indépendant de ce de la frame précédente. On remarque que cela fonctionne car l'image se débruite au cours de l'accumulation.

7) Color texture: Cette texture est sur 4 layers. Chaque shader va écrire la valeur résultat de l'illumination dans le canal dédié. Le premier est pour le L0, le second pour l'éclairage direct, le 3ème pour l'éclairage indirect et le dernier pour l'éclairage ambiant. Cette texture est remise à zéro à chaque frame.

8) Result texture: Sur les mêmes canaux que Color texture, cette texture va enregistrer l'accumulation temporelle des couleurs sur tous les canaux puis sommer dans un 5ème canal qui sera l'image affichée. L'accumulation est une moyenne de toutes les frames précédente. La première frame est sauvegardée telle quelle. La frame x sera sauvegardée selon la formule $accu\_color = (x * pred\_color + new\_color)/(x+1)$ pour chaque canal.

C. Intersection

Tout d'abord, le shader tp3_intersect.glsl va lancer 1 rayon par pixel et stocker le résultat dans la storage texture Ray. Le rayon lancé en chaque pixel est bruité ($+x | x \in [-0.5, 0.5]$). Cela permettra à l'accumulation temporelle d'anti-aliaiser en plus de débruiter. Ce shader permet aux 3 prochains shaders de s'exécuter en parallèle (car ils partent tous du même rayon stocké dans la texture).

D. Éclairage ambiant

Voir fig 2. Ce shader calcule l'éclairage ambiant. En chaque point (et à chaque frame) le shader envoie un rayon dans une direction aléatoire (Compendium 35.) et si ce rayon n'intersecte aucun triangle dans toute la scène on le considère éclairé. On pourrait réduire la taille du rayon si on ne voudrait calculer que localement l'ambiant.

E. Éclairage direct

Voir fig 3. Ce shader calcule l'éclairage direct (L0 et L1). Pour chaque point on tire un rayon jusqu'à chaque source et on regarde si ce rayon est intersecté. On applique un éclairage de Blinn-Phong.

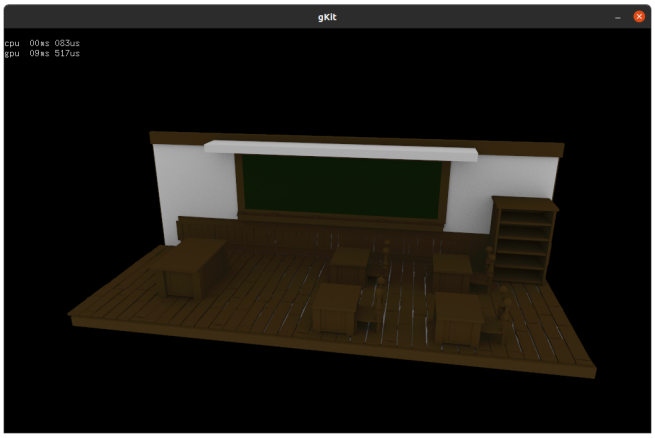

Fig. 2. Éclairage ambiant

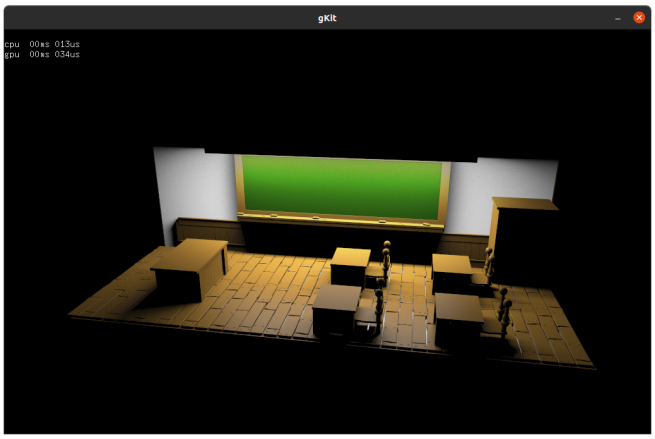

Fig. 3. Éclairage direct

F. Éclairage indirect

Voir fig 4. Ce shader calcule l'éclairage indirect (L2). En chaque point d'intersection le shader va renvoyer un rayon aléatoirement, et s'il touche de la géométrie on va calculer le L1 en ce point. J'ai essayé de réutiliser le L1 précalculé (technique utilisée dans la TP 2) mais le recalcul est assez rapide pour que le gain de performance ne soit pas flagrant, car dans les 2 cas on doit renvoyer un rayon. Si on envoyait plusieurs rayons par frame cela aurait peut être été un gain de performance. On aurait aussi pu récupérer du L1 débruité ce qui aurait été plus précis qu'un L1 à un seul rayon.

G. Accumulation temporelle

Du à la nature de l'aléatoire du programme, calculer plusieurs images donne des résultats différents. Le programme accumule 1000 frames, après il affiche simplement ce qui a déjà été calculé.

1) Anti-Aliaising: Pour chaque image le léger décalage du rayon dans le pixel va dé-aliaiser l'image automatiquement.

2) Denoising: Pour chaque rayon lancé (L1, ambiant, etc...) la direction aléatoire sera différentes, l'accumulation d'image revient donc à calculer les intégrales avec de plus en plus d'échantillon, ce qui tend vers la solution voulue. On aurait pu envoyer des rayons le long de la spirale de Fibonacci, cela aurait fait un bruit structuré mais le résultat convergerait plus vite.

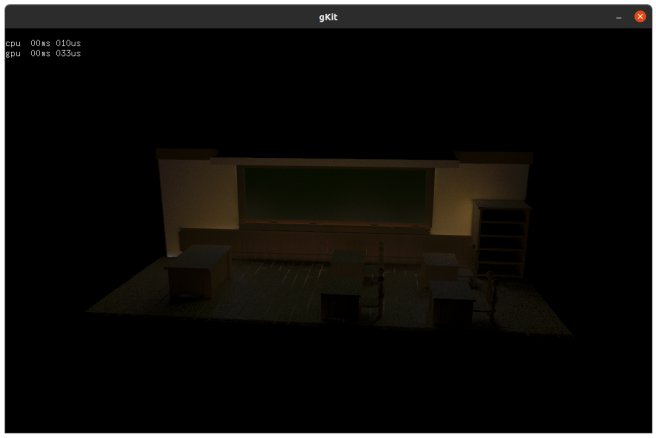

Fig. 4. Éclairage indirect

III. Améliorations

A. Plus de lumière

Contrairement au TP2, le calcul de la lumière indirecte se fait avec 1 rayon par sources, ce qui endommage gravement la framerate plus il y a de sources de lumières (et pose les mêmes problèmes pour le calcul correct de l'éclairement). Il faudrait donc choisir une source aléatoirement parmi toutes les sources avant d'en prendre un point aléatoire. On pourrait pondérer la probabilité de tirer une source en fonction de sa distance avec le point, car les sources plus lointaines éclairent moins.

B. Denoising

Actuellement le dénoising (et l'anti aliaising) et font naturellement par l'accumulation temporelle des images. On pourrait ajouter (surtout pour les premières frames) un filtre pour débruiter l'image.

C. Recalcul complet non nécessaire

À chaque mouvement de caméra l'image entière est détruite et on recommence le calcul de 0. Il aurait été intéressant de trouver les points qui avaient simplement translaté dans la scène pour repartir d'une base. Cependant, l'éclairage de Blinn-Phong induisant des reflets selon l'angle avec la caméra, il y aurait eu des corrections à faire.

D. BVH cousu plus compact

Le BVH cousu fait d'abord une construction de BVH classique, et ces noeuds sont ensuite retriés pour coudre (ajouter un next et un skip), les indices de Noeuds ne sont donc pas leur ordre dans le buffer, et il est obligatoire de garder le next. De plus l'intersection avec le next pourrait être plus intelligente (BBox dans le parent, etc...) et plus cohérente.

E. Matériaux/lighting plus performant

Le TP a été fait en comparant le résultat du rendu instantané (par accumulation) de Blender. Le rendu est relativement plus rapide que Cycles temps réel (mais je crois que Blender fait le sien sur CPU...) mais diffère par rapport à tous les paramètres de lighting (roughness, subcolor, etc...) et le rendu n'est ultimement pas le même.