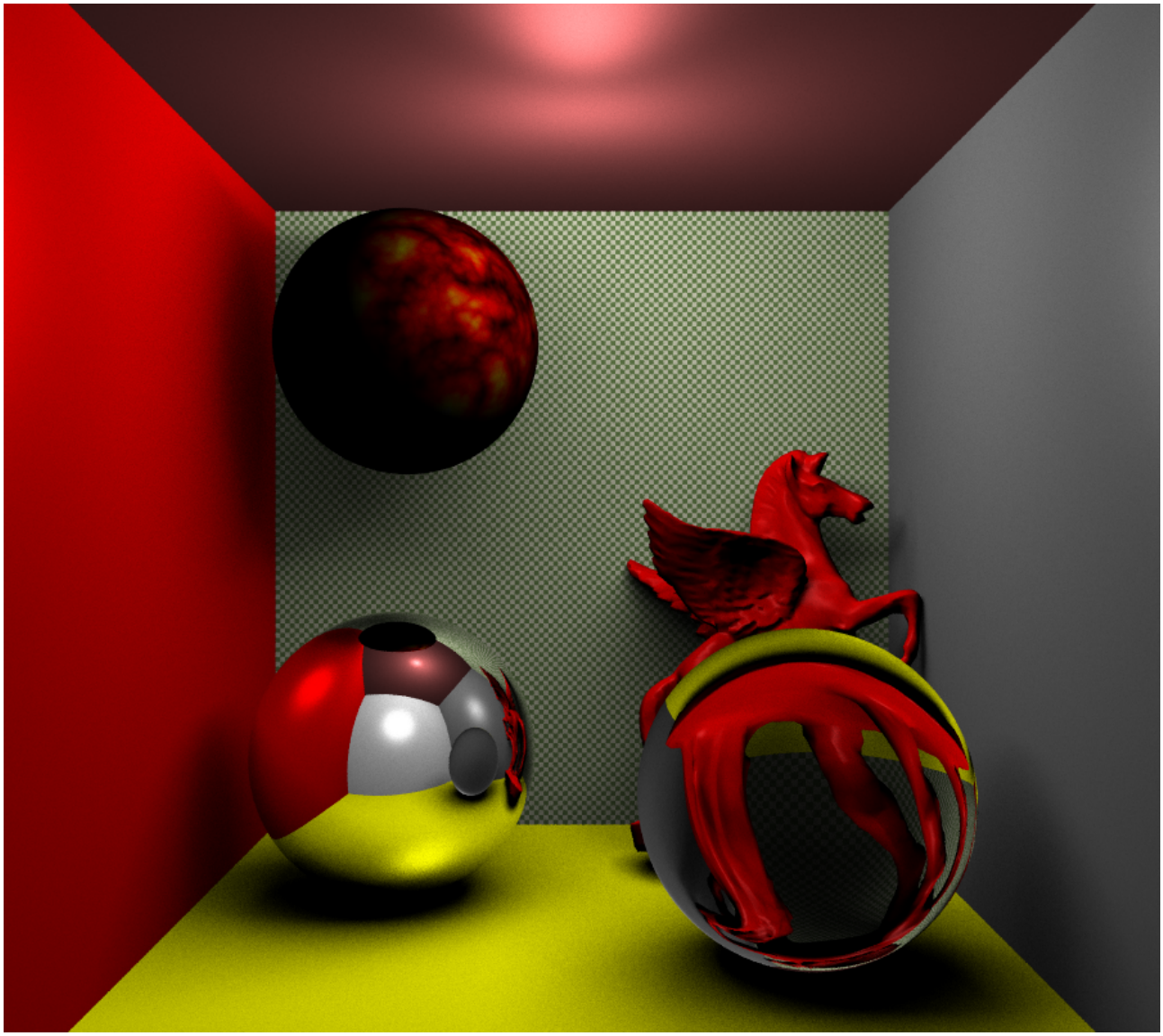
HAI719I – Programmation 3D

Rendu projet phase 4

Léo Hafdane – e22202516 – Github

16/01/2026



M1 Informatique – IMAGINE
Faculté des Sciences
Université de Montpellier.

1 Settings et préréglages

Lors de la phase 3, j'ai mis en place un système de paramètres globaux (dans `Settings`) ainsi que des préréglages modifiables. Cette partie introduit plusieurs paramètres : `ENABLE_GLASS`, `AIR_INDEX_MEDIUM` et `MAX_LEAF_SIZE`, ainsi que les nouveaux préréglages `PHASE_4_REFRACTION` et `PHASE_4_KDTREE`. On peut toujours changer de préréglage avec p et P.

2 Réfraction

Pour la réfraction, je me suis aidé de cette partie de page wikipédia (en plus de celle-ci) pour comprendre le phénomène et de ce site pour m'aider à implémenter la réfraction en elle-même. Tout ceci se retrouve dans ma fonction `Scene::computeRefractionRay`.

Puisque ce calcul dépend de la densité du matériau que le rayon quitte et de celle dans laquelle il entre, il faut que les rayons gardent en mémoire les objets par lesquels ils sont passés ainsi que leur densité. La structure `Ray` contient maintenant une liste initialisée avec un `index_medium` à 1, celui de l'air. Je suis parti du principe que la caméra ne se trouve jamais dans un solide.

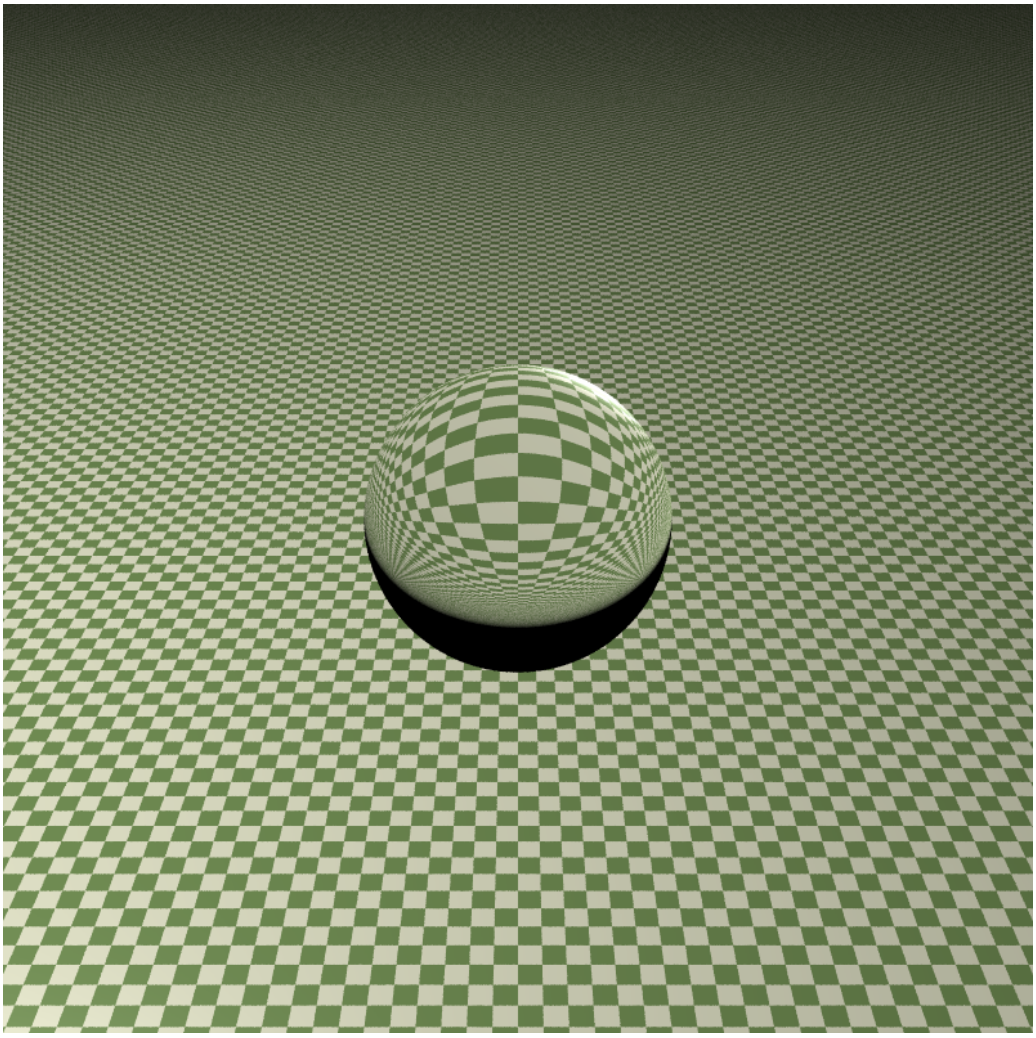


FIGURE 1 – Cliquez sur l'image pour voir une démonstration vidéo (boucle).
Lien vers la démonstration vidéo.

3 KdTree

3.1 Bounding box

Avant de faire mon KdTree, j'ai eu besoin de coder une axis-aligned bounding box (AABB). En effet, lors d'une intersection avec le KdTree, on vérifie dans un premier temps si le rayon touche la AABB. Cela nous permettra de savoir dans quelle branche aller pendant le calcul d'intersection. De plus, cela permets également d'éviter de faire des calculs d'intersection sur une mesh qu'on est sûr de ne pas toucher, accélérant ainsi tous les autres rayons.

Pour construire une AABB, il suffit d'itérer sur tous les triangles et de garder en mémoire les composants minimum et maximum de chaque axe. La AABB finale étant le cube formé par $min_{x,y,z}$ et $max_{x,y,z}$.

3.2 Benchmarks

Pour évaluer l'efficacité de mon KdTree, j'ai réalisé des benchmarks testant, pour chaque mesh disponible, une taille maximale de feuille de 0 ainsi que les premières valeurs 2^n et $2^n + 2^{n+1}$. Cela m'a donné un gros fichier csv : *raw_mesh_benchmark.csv*. J'ai dans un premier temps extrait les moyennes dans un tableau à deux entrées (trop grand pour que je l'affiche mais il est ici). J'ai ensuite extrait les graphiques suivants :

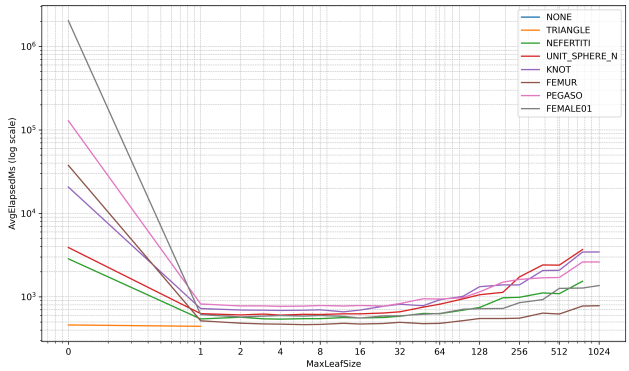


FIGURE 2 – Temps moyen vs la taille max de feuille pour chaque mesh
Ici, on peut voir que l'utilisation d'un KdTree réduit énormément le temps de calcul. On peut également voir qu'à partir de 32 triangles maximum par feuille, ce gain de performance commence à réduire significativement. Je n'ai malheureusement pas mesuré le temps de construction du KdTree par rapport au nombre de max de triangle par feuilles mais on peut faciement deviner que plus ce nombre diminue, plus le KdTree sera long à construire. L'entre deux parfait semble être autour de 32.

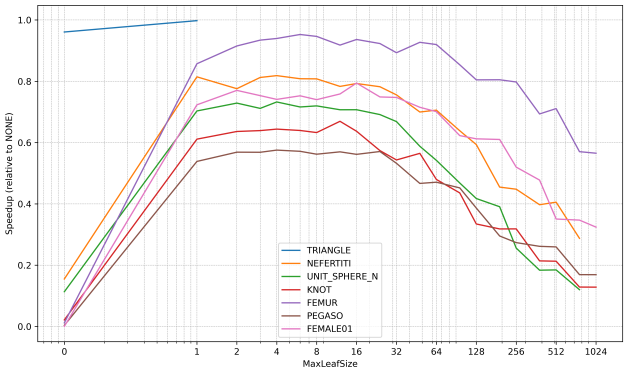


FIGURE 3 – Gain de temps relatif vs taille max de feuille pour chaque mesh
Ici, la métrique "speedup" décrit le gain de temps au temps de calcul de la scène sans Mesh. il est calculé comme étant $speedup = \frac{reference}{avgMeshTime}$. Parrallèlement à l'autre diagramme, on peut mieux voir que le gain de temps est maximal autour de 8.

3.3 Construction

Un KdTree est défini par son ensemble de triangles T où $|T| = n$, une AABB, un paramètre α décrivant le nombre maximal de triangles dans une feuille, s son axe de coupe ainsi que deux enfants k_1 et k_2. L'idée du KdTree va de pair avec sa construction :

- Si $n < \alpha$, alors le KdTree est une feuille, il n'a pas d'enfants mais garde son ensemble de triangles.
- Sinon, le KdTree doit avoir deux enfants, avec deux nouveaux ensembles de triangles et deux nouvelles AABB :
  1. Trier T et calculer C, le centroïde médian sur l'axe s.
  2. Séparer T en deux tableaux T_1 et T_2 tels que T_1 contient tous les triangles qui ont au moins un point à gauche de C sur l'axe s (resp. T_2 à droite).
  3. Calculer les deux nouvelles AABB.
  4. Construire les deux enfants de la même manière mais avec $s = (s + 1)\%3$.

3.4 Intersection

L'intersection consiste en un simple parcours d'arbre, Si le KdTree actuel est une feuille alors il teste l'intersection à la bounding box des deux enfants. Et les parcours en premier la AABB ayant le plus petit t et l'autre ensuite.

3.4.1 Exemples Génération

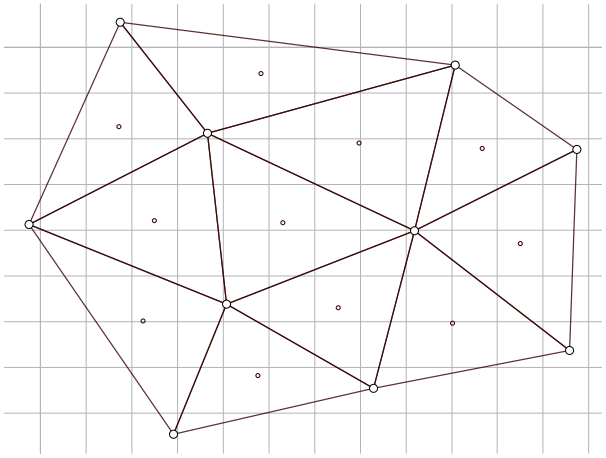


FIGURE 4 – Étape 0 de construction d'un KdTree, on a 11 Triangles

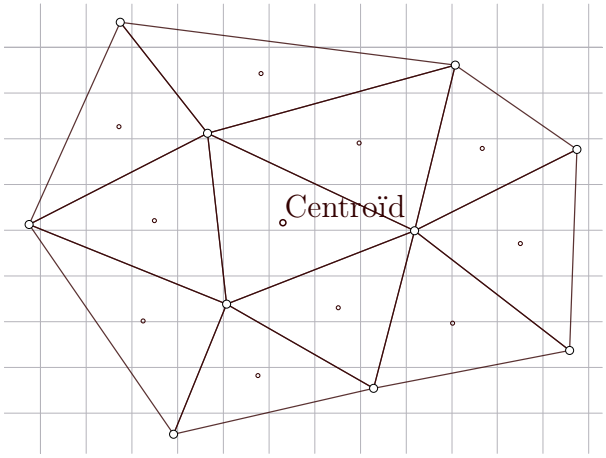


FIGURE 5 – Étape 1 de construction d'un KdTree, le centroïde médian est marqué

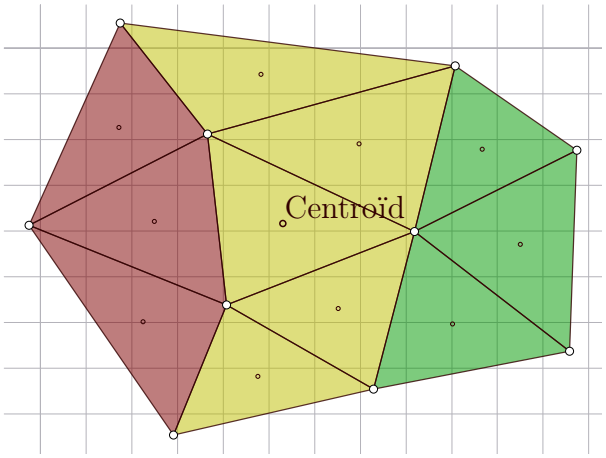


FIGURE 6 – Étape 2 de construction d'un KdTree, le centroïde médian est marqué

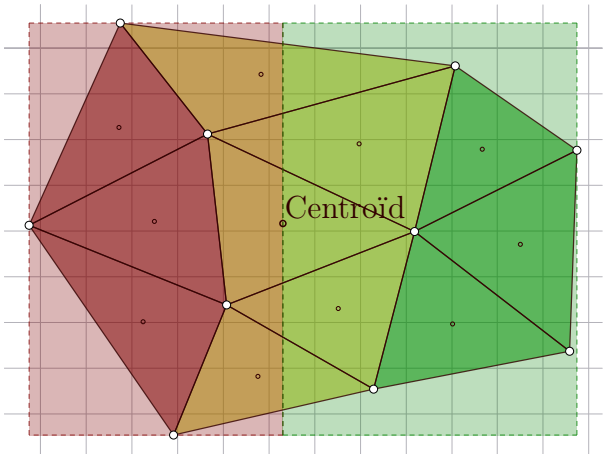


FIGURE 7 – Étape 3 de construction d'un KdTree, le centroïde médian est marqué

On a maintenant 1 profondeur de KdTree, la racine étant la grande boite qui englobe tout les tirangles, T_1 Contiens les triangles colorés en rouge et jaune tandis que T_2 contiens ceux en jaune et vert. On peut également voir leur propre AABB réprésentés en rouge et vert.

4 Bonus

4.1 Textures

J'ai implémenté les textures. Je n'ai pas réimplémenté les paramètres d'OpenGL (`GL_CLAMP_TO_EDGE`, etc...) mais elles fonctionnent. J'ai introduit le paramètre `ENABLE_TEXTURES`.

4.2 GPU

J'ai créé `Renderer`, une surclasse de `Scene` qui gère tout le côté rendu, que ce soit celui basé sur le CPU ou le GPU. Mon accélération GPU fonctionne avec un compute shader. Je l'ai implémentée en gardant l'idée de création d'image spontanée et non en visant un temps réel. Je compile alors le shader, transfère les données et l'exécute à la suite à chaque création d'image. Maintenant on peut appuyer sur r pour faire un rendu sur CPU et R pour le faire sur GPU.

4.2.1 Données de la scène

Pour faire passer les données de la scène, j'ai utilisé des storage buffers car ils me laissaient la liberté de faire passer autant d'objets que je le voulais. Un storage buffer vient cependant avec une petite complication : l'alignement des données. En effet, toute donnée doit être alignée selon des règles un peu trop complexes pour que je les réexplique ici. Cependant, après avoir compris ces règles et exploré Internet, j'ai découvert que WebGPU utilisait les mêmes règles qu'un storage buffer std430 et qu'il existait un site permettant de visualiser l'alignement des structures WGSL. J'ai donc fait ma petite page.

Pour pouvoir implémenter des mesh, j'ai créé de buffer qui contiens l'ensemble des vertex des mesh de la scène (resp. triangles). La structure Mesh quand à elle contiens juste un nombre de vertex, et un offset dans ce buffer (resp. triangles).

4.2.2 Limitations

Malheureusement, je n'ai pas eu le temps de faire fonctionner les textures. J'ai essayé d'implémenter un `SAMPLER_2D_ARRAY` mais il ne fonctionnait pas. Je n'ai pas non plus implémenté le KdTree, même si cela aurait été possible, je ne me voyais pas passer autant de temps à déboguer un KdTree sur le GPU. Avec le passage sur le GPU, le nombre de passages maximum dans un objet réfractant d'un rayon a été limité à 100. Je ne crois pas que je pouvais le contourner avec un buffer. Cependant, il est quand même très rare qu'un rayon passe dans 100 profondeurs de milieux différents donc je pense que ça va.

4.2.3 Benchmarks

Cette fois ci j'ai fait tourné un test qui comparait les performances CPU et GPU par rapport à une scène dont le nombre d'objets augmentent.

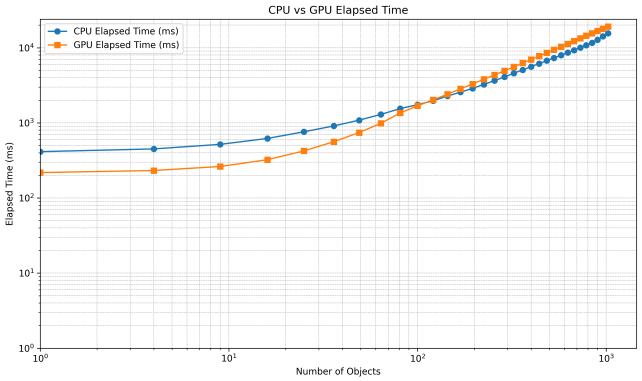


FIGURE 8 – Gain de temps relatif vs taille max de feuille pour chaque mesh

Ici, on peut voir que le GPU est plus efficace que le CPU sur des petites scène mais on peut aussi voir que la tendence s'inverse autour de 100 objets. Cela peut s'expliquer soit le fait que les transferts mémoire ont beaucoup plus d'effet sur le GPU que sur le CPU. Mais puisqu'une scène de 100 objets fait approximativement $12KiB$, ce qui semble peux pour que cela soit la cause. Cela peut aussi s'expliquer par le fait que mon compute shader n'est pas très efficace, ce qui est probable puisque c'est mon premier et que j'essayait surtout de le faire fonctionner.