Synthèse d'image 3D

Rapport TP1

Barros Mikel-ange – Scouflaire Thomas

Lien vers le code : https://forge.univ-lyon1.fr/p1508726/rendu-si3d

Partie 2 - Affichage et shader :

Dessiner un seul objet se fait en définissant un vao pour l'objet et en appelant le shader sélectionné avec glDrawArray, ceci permet de dessiner facilement un objet. Le shader sera quand à lui écrit en glsl et permettra l'affichage de notre objet. Cette partie ne nous a posé aucun problème étant donné que nous avions travaillé sur cela en M1 dans l'ue SI.

Cependant, pour afficher plusieurs instances du même objet faire un draw par objet est une solution lourde et peu efficace. Afin de pallier a ce problème, nous pouvons utiliser la fonction glDrawArraysInstanced et choisir de stocker les informations de positions dans un Shader Storage Buffer Object (ou via un uniform si les données sont de petites tailles). Cela permet de récupérer ces informations dans le shader et de les utiliser pour dessiner les différentes instances de notre objet. GlDrawArrayInstanced est une fonction de opengl qui permet de faire avancer la variable instanceID de opengl et de dessiner l'ensemble des objets entre l'objet à l'id first et l'objet à l'id finale.

Partie 3 - Animation et interpolation :

Afin de faire l'interpolation entre le temps 1 et le temps 2 nous passons les informations sur les deux animations au shader via un vao et nous calculons la position de chaque point du mesh à un instant t. Pour pouvoir calculer la position du point , au temps t (compris entre t=0 et t=1) , il nous faut donc connaître le temps depuis le dernier changement d'animation et l'intervalle de temps entre deux animations It. La formule pour interpoler p entre t0 et t1 serait alors :

$p(t)=p(t0) * (t/It) +p(t1) *(1-t/It)$ pour chaque point.

Partie 4 - Materials

Afin de récupérer les matériaux de nos objets et de les afficher le passage par uniform est le plus simple, mais limite beaucoup le nombre de materials et d'objets affichables. Pour éviter cela, on va juste envoyer un uniform contenant les informations des différents matériaux que l'on utilise (habituellement max 255). Une fois cela fait, nous ajoutons un attribut à chacun de nos point, cet attribut correspond à l'indice du matériaux utilisé par ce point et nous permet de connaître les informations de couleurs pour notre point.

Ce résultat fonctionne (dans la limite de 255 materiaux)et permet de limiter l'utilisation de la mémoire accordée à un uniform, cependant ca nous limite quand même dans le nombre de matériaux utilisés et dans la manière de les utiliser. L'utilisation d'un Shader Storage Buffer Object permettrait de palier ce problème.

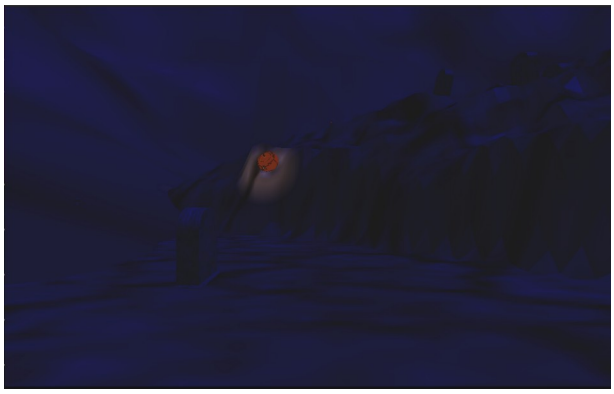
Partie 5 – Deffered Shader

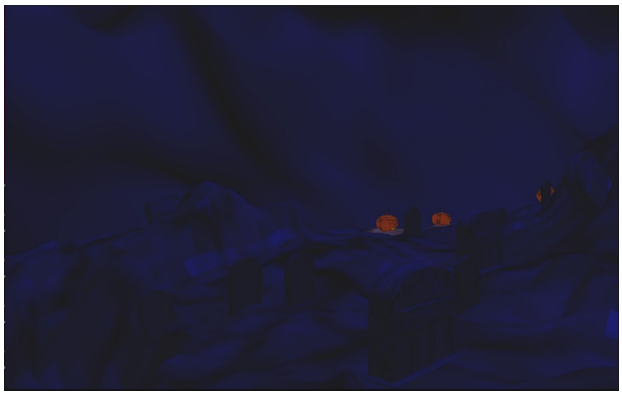
Le deffered shading consiste à calculer les différentes informations (positions, normal, couleurs) dans une première passe, puis de calculer l'éclairage ambiant lors d'un deuxième passage. Cela permet d'optimiser légèrement les performances.
Dans notre code cela ce traduit en plusieurs étapes :
Tout d'abord, nous allons créer les textures sur lesquelles nous voulons dessiner, ainsi que l'ensemble des vao associés à nos mesh.
Une fois cela fait, nous allons dessiner chacun des objets de la scène sur une texture à l'aide d'un shader.
Puis enfin, nous plaquerons nos textures sur un quad au centre de l'image et nous afficherons ce quad.

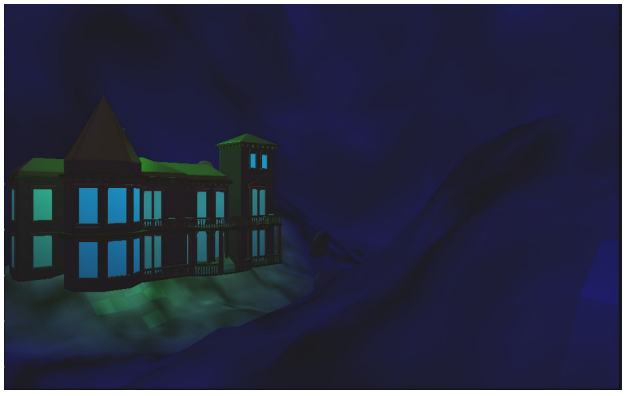
Scène finale :

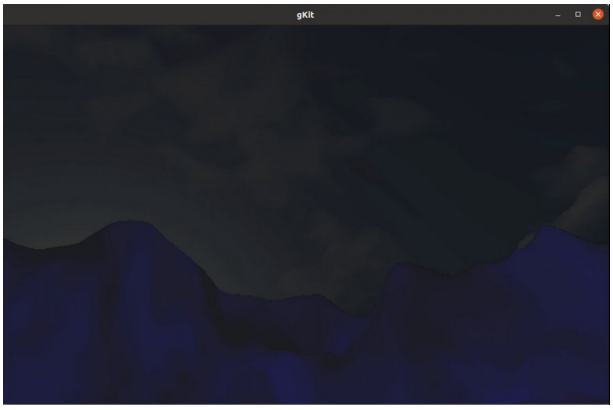
La scène final représente un cimetière à halloween. Pour cela, on suit la caméra qui se déplace le selon un chemin prédéfinis. Afin de rendre le cimetière plus réaliste de nombreuses tombes citrouilles et monstres ont été insérés ce qui rend la scène assez lourde. Afin de l'optimiser nous dessinons les différents objets avec des glDrawArraysInstanced ,cela permet d'éviter les boucles et les multiples draw qui ralentiraient l'application , notamment au niveau cpu. De plus, nous dessinons le tout avec une méthode de deffered shader ce qui rends le calcul gpu plus simple en découpant les taches de rendu d'objets et les taches de rendu lumineux. La plupart des meshs utilisés dans cette scène ne possèdent pas de materiaux associés, on utilise donc des textures pour assigner des couleurs à nos objets.
Enfin afin de rajouter du réalisme à la scène un effet de cube map a été rajouté pour modéliser le ciel.

Nous voulions vous mettre une vidéo mais la capture avait beaucoup de lag, nous vous laisseront donc découvrir la scène par vous même

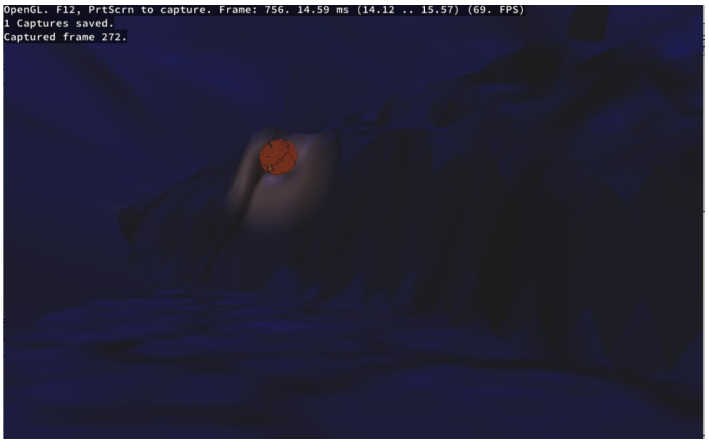








Performances de la scène finale
Capture prise sur render doc, le processeur graphique utilisé étant une radeon Rx vega 10 (intégré CPU)



La moyenne sur la scène est de 68fps.