Université de Limoges

master ISICG limoges

Moteurs 3D Temps Réel
TP0 : Les premiers triangles

G.Gilet

22 septembre 2020

L'objectif de ce TP est de se familiariser avec les commandes OpenGL et les shaders, en partant d'une petite application permettant l'affichage en 2D. Vous trouverez l'application sur le serveur git de l'université (git.unilim.fr).

1 Le premier triangle

La première étape consiste à afficher un unique triangle en 2D dans la fenêtre avec la couleur de votre choix. Pour arriver à ce résultat, plusieurs étapes sont nécessaires :

- Commencez par déclarer et remplir un tableau de 3 sommets. Chaque sommet sera défini par un vecteur indiquant sa position dans l'espace. Nous utiliserons pour les types de données la bibliothèque glm (glm.g-truc.net). Attention toutefois à définir des coordonnées correctes pour chaque sommet.
- En suivant l'exemple donné dans le cours, créez un *Vertex Buffer Object* ainsi qu'un *Vertex Array Object* et chargez le tableau précédent en mémoire graphique. Prenez également soin de spécifier le format des données chargées sur GPU ainsi que les attributs correspondant.
- A l'aide de la fonction de lecture de fichier donnée, écrivez maintenant un *Vertex Shader* écrivant la coordonnée du sommet dans l'espace 2D ainsi qu'un *Fragment Shader* assignant une couleur unique à chaque fragment. Déclarez les objets *shaders* correspondants ainsi que l'objet *program* et activez les. Vous aurez besoin pour cela d'utiliser les fonctions **glCreateShader**, **glCompileShader**,**glCreateProgram**,**glUseProgram**,**glAttachShader**.
- Finalement, spécifiez les objets à utiliser dans la boucle de rendu et lancez l'affichage de votre triangle.

2 Ajoutons des triangles

Le principe maintenant est d'ajouter un deuxième triangle accolé au premier, afin d'afficher un polygone à 4 cotés à l'écran. Il ne s'agit pas toutefois de simplement dupliquer les sommets communs des triangles, mais de créer un tableau d'indices indiquant la topologie du maillage.

- Ajoutez en premier lieu un 4ème sommet dans votre tableau de sommets.
- Créez ensuite un tableau d'indices indiquant les indices des sommets composant chaque triangle et chargez le en mémoire graphique.

— Activez ensuite ce tableau dans votre *Vertex Array Object*. Vous pouvez pour cela utiliser la fonction **glVertexArrayElementBuffer**. Affichez maintenant votre nouvelle géométrie. Attention, la commande envoyant la géométrie dans le pipeline n'est plus la même !

3 Un nouvel attribut

Ajoutons maintenant depuis le CPU un attribut couleur différent pour chaque sommet.

— Créez un tableau contenant des couleurs, chargez le en mémoire graphique et activez l'attribut correspondant dans le *Vertex Arrray Object*.
— Modifiez vos shaders pour que la couleur de chaque sommet soit définie dans le *Vertex Shader* pour chaque sommet et transmise au *Fragment Shader*.

4 Des données uniformes depuis le CPU

Occupons nous maintenant d'envoyer des informations depuis le CPU vers le *Fragment Shader* afin de modifier la couleur des fragments en fonction du temps.

— Commencez par définir une variable variant au cours de chaque execution de la boucle de rendu (compteur, *timer*...).
— Declarez cette variable sur le GPU et transmettez la valeur correspondant en utilisant le mécanisme des variables *uniform* vu en cours.
— Bonus : Pour les plus avancés, essayez de définir cette variable GPU à l'aide d'un *Uniform Buffer Object*.