Moteurs 3D Temps Réel
TP1 : Premiers shaders

G.Gilet

15 septembre 2017

Au cours de ce TP, nous allons commencer par nous familiariser avec le moteur GobLim qui sera utilisé au cours de ces TP.

1 Premiers Pas

Commencez par vous familiariser avec l'architecture du moteur, en essayant de repérer :

— La classe SampleEngine et les fonctions *init()* et *render()* du moteur
— Le **Material** utilisé pour le premier affichage ainsi que les shaders associés.

2 Modification du *vertex program*

Les quelques exercices de la section suivante porteront uniquement sur le *vertex program* (**MainVS.glsl** du materiau **ColorMaterial**.

2.1 Premiers exercices

Voici quelques premiers exercices traitant de la couleur pour se familiariser avec les *vertex program* :

— Assignez comme couleur à chaque sommet la valeur absolue de sa position dans l'espace objet (il peut être nécessaire de multiplier cette valeur par une constante arbitraire afin de mieux visualiser les positions)
— Assignez maintenant à chaque sommet la valeur absolue de sa position dans l'espace image. Attention, nous sommes ici en dimension 4, il faut donc diviser les coordonnées de la position par la dernière coordonnée pour revenir dans le domaine cartésien (Utilisez pour cela les fonctionnalités de glsl, par exemple : $P.xyz/P.w$).
— Assignez comme couleur à chaque sommet la valeur absolue de la normale en chaque sommet. (Rappel : il faut pour cela déclarer une nouvelle variable **in** (le flux de données entrant contenant l'information de normale, sur le canal 2, cf. exemple du cours))

2.2 Déformation du lapin

Quelques exercices plus complexes traitant de la déformation du lapin, afin de se familiariser avec les *Vertex Program*.

— Déplacez, dans l'espace de l'objet (avant projection), chacun des sommets ayant une coordonnée X supérieure à 0 de 0.03 unités le long de sa normale. Il suffit pour cela d'ajouter 0.03 fois le vecteur normal à la position du sommet. Attention, la variable Position est en lecture seule, il est nécessaire de créer une nouvelle variable, de copier le contenu de *Position* et (après déformation) de projeter ce nouveau sommet dans l'espace image.
— Déplacez, dans l'espace de l'objet, chacun des sommets ayant une coordonnée Y supérieure à 0 de 0.2 unités le long de son axe X. Il suffit d'augmenter la valeur de la coordonnée x de la position du sommet.
— Réalisez cette dernière opération dans l'espace écran (après projection). Que constatez vous ?

2.3 Interaction avec le CPU

— Calculez une couleur sur le CPU et transmettez la au vertex program. Pour cela, il est nécessaire de :
  — Déclarer une nouvelle variable dans le bloc **uniform** du VP.
  — Déclarer une variable de type **GPUvec3*** dans le **Material** correspondant (la classe *ColorMaterial*)
  — D'initialiser (dans le constructeur de la classe *ColorMaterial*) cette variable avec l'adresse mémoire du la variable déclarée dans le VP. Il faut pour cela faire appel au gestionnaire de variables (**uniform()**) du VP (cf Cours)
  — D'utiliser enfin (toujours dans le constructeur) la fonction *Set* de cette variable afin d'inscrire la valeur choisie dans la mémoire graphique

2.4 Déformation du lapin (bis)

Utilisez une variable *temps* (définie par le CPU et) afin de réaliser une animation du lapin basée sur le temps. Commencez par déformer les sommets du lapin dans l'espace objet en fonction du cosinus de cette variable temps (par exemple,pour un sommet (x, y, z) : $x = x + 0.1 * cos(y * time)$). Réalisez ensuite la même opération dans l'espace image et commentez.
Le *vertex program* permet (entre autre) de modifier la géométrie sommet par sommet. A vous d'expérimenter !

3 Modification du *Fragment Program*

Voici quelques exercices permettant de manipuler les fragments :
— Assignez à chaque fragment une couleur dépendant de sa position à l'écran. Utilisez pour cela la variable GLSL *gl_FragCoord*, qui contient la position en pixel du fragment. Attention, cette position étant en pixel, il faudra diviser cette valeur par la largeur de la fenêtre.

4 Un shader cohérent

La plupart des programmes reposent sur une interaction entre vertex et *fragment program*. Voici quelques exemples à réaliser :

— Créez une variable contenant la position dans l'espace objet de chaque sommet, calculez une couleur pour chaque zone de l'objet (par exemple, une couleur spécifique pour les $Z > 0$ et une autre pour les $Z <= 0$) et transmettez cette valeur au *fragment program*.
— Recommencez la même opération en transmettant cette fois la position dans l'espace objet au *fragment program* (cela nécessite de créer une nouvelle variable) et en effectuant le calcul de couleur dans le *fragment program*. Que constatez vous ?
— Définissez au sein du *vertex program* une déformation de la géométrie à l'aide d'une formule de votre choix. Utilisez également cette formule pour déterminer divers paramètres (par exemple la déformation subie par le sommet) qui vous permettrons de calculer la couleur d'un fragment dans le *fragment program*.