POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

INF2705 Infographie

Travail pratique 2

Transformation matricielle et textures

Département de génie informatique et génie logiciel
Polytechnique Montréal
Tristan Rioux, Hiver 2025

Table des matières

1 Description globale

1.1 But

Le but de TP est de permettre à l'étudiant de mettre en pratique les notions de transformation matricielle afin de peupler une scène graphique et d'animer des modèles 3d. Il sera en mesure d'utiliser des textures afin d'ajouter des détails aux objets. Le travail fera l'utilisation des fonctions de glm `glm::scale`, `glm::rotate`, `glm::translate`, `glm::perspective` et `glm::ortho`, puis les fonctions d'OpenGL `glGenTextures`, `glBindTexture`, `glTexImage2D` et `glTexParameteri`.

1.2 Travail demandé

Le tp2 est une continuation du tp1 ; il reprend la même forme de celui-ci avec les différentes scènes. Cette fois-ci, une seule scène est à remplir. Certaines implémentations de classe du tp1 ont été déplacées dans la librairie, **vous n'avez pas à ajouter de fichier .cpp pour le travail**.

Prenez note que le face culling est activé dans le main. L'ordre des points détermine le sens de la face. Un mauvais ordre pourrait cacher votre face sans que ce ne soit voulu.

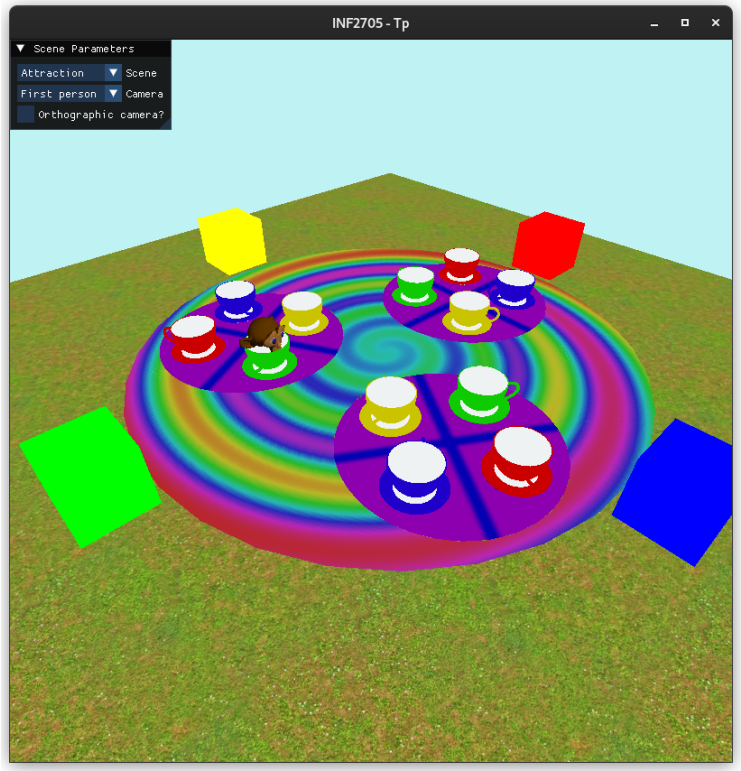

FIGURE 1 – Rendu final du tp2

Partie 1 : chargement et animation de modèles 3d

En temps normal, il est commun de faire les modèles 3d dans une autre application spécialisée pour la modélisation plutôt que remplir des tableaux de points directement dans le code. Dans votre cas, les modèles ont déjà été fait et une librairie de chargement de fichier .obj est disponible. Il faudra tout de même implémenter le chargement des données du modèle dans la carte graphique. Pour ce faire, vous devez remplir la classe Model en initialisant le VBO, EBO, drawCommand et spécifier l'attribut de position et de coordonnée de texture dans le VAO.

Pour commencer, on utilisera les shaders `colorUniform.*.glsl`. On aura une version modifiée du shader de transformation du tp1 qui ne possède que l'attribut de position en entrée dans le vertex shader. Dans le shader de fragment, on remplacera l'attribut de couleur en entrée par une variable uniform. Pour différencier les objets, on pourra temporairement utiliser cette variable uniforme pour changer la couleur du modèle. Utiliser le ShaderProgram `colorUniform` dans la classe Resources.

Dans la scène, il faudra construire la matrice mvp pour les différents modèles :

- La matrice de projection orthographique projete l'origine au centre de l'écran et possède un frustum carré de taille `SCREEN_SIZE_ORTHO`. Le near plane est à 0.1 et le far plane à 300.
- La matrice de projection de perspective est identique à celle du cube du tp1, mais le far plane est à 300 (fov de 70 degrée, aspect ratio de la taille de la fenêtre, near plane 0.1).
- La matrice de vue en première personne est obtenable à partir de la méthode getCameraFirstPerson. Il faudra faire des rotations selon l'orientation de la caméra et une translation selon la position de la caméra (glm : :lookAt ne sera pas accepté, appliquer l'inverse des transformations).
- La matrice de vue en troisième personne est obtenable à partir de la méthode getCameraThirdPerson. L'utilisation de glm : :lookAt et des coordonnées sphériques sera utile. On utilisera un rayon constant de 36 pour la position de la caméra. Puisqu'on voudra initialement avoir la caméra au niveau du sol, un décalement de $\pi/2$ sera appliqué sur l'orientation en x et y. La caméra fixe l'origine de la scène.

Pour les transformations des objets, **vous ne devriez pas avoir de trigonométrie dans le code des transformations, utilisez des matrices de rotation et translation**. Il est recommandé de garder les animations pour la fin :

- Quatres cubes sont placé au point cardinaux à $(30, 0)$, $(-30, 0)$, $(0, 30)$ et $(0, -30)$ pour aider à se repérer dans la scène. Ils reposent sur le plan xz à une hauteur $y = 3$. Ils sont de couleur uniforme (aucune texture n'est donnée).
- Le sol de taille d'arète 90 est en dessous du manège à -0.1.
- La grande plateforme est centrée à l'origine. Elle possède une animation de son angle en y de `m_largePlatformAngle`.
- Il y a 3 groupes de tasses. Un groupe de tasse est constitué d'une petite plateforme et de quatre tasses. Le centre des groupes de tasse sont sur un cercle de rayon 15, séparer par 120 degrée d'arc de cercle. De plus, chaque groupe de tasse possède une animation de leur angle en y de `m_smallPlatformAngle`.
- Pour chaque tasse dans un groupe, elles sont placées sur un cercle de rayon 6 par rapport au centre du groupe, séparer par 90 degrée d'arc de cercle. Chaque tasse possède une animation de leur angle en y de `m_cupsAngles`.

— Une tasse est accompagné d'une assiette. La tasse doit être remonté de 0.12 en y pour être déposé sur l'assiette.
— Un singe est placé dans une tasse ! Sa position et orientation en y ont été calculé pour vous, il ne reste qu'à faire les appliquer. De plus, une mise à l'échelle de 2 sur tous les axes doit être fait. Le singe n'est pas dessiner si on prend sa vue (`m_cameraMode == 2`).

Il est recommandé d'ajouter des méthodes privées afin de réduire la duplication de code et de vous aidez à résoudre le problème. Organiser votre code de façon réduire le nombre de multiplication matriciel. Baser vous sur le principe de stack de matrice vu en classe pour vous aidez (sans avoir besoin d'implémenter la classe, le principe reste applicable).

Partie 2 : textures

Malgré les nouveaux ajouts que nous avons fait à la scène, elle manque clairement de détail. Une technique couramment utilisée pour facilement rajouter du détail est d'utiliser des textures.

Essayer de dessiner les objets en changeant le moins souvent l'état d'OpenGL (les shaders et textures notamment, voir section 1e).

Partie 2a : Chargement des textures

Pour ce faire, il va falloir charger les textures de chaques modèles et les appliquer sur ceux-ci lorsqu'il sera temps de les dessiner avec la méthode `use`. Une classe `Texture2D` contenant du code pour le chargement des images vous est fournis. Vous devez complété son constructeur pour charger l'image en tant que texture, ainsi que les autres méthodes pour utiliser la texture.

Utiliser les méthodes `setFiltering`, `setWrap` et `enableMipmap` à l'extérieur du constructeur (dans la scène).

Vous allez devoir charger les textures suivantes et les appliquer sur les objets respectifs :

"`largePlatform.png`", "`smallPlatform.png`", "`cupTextureAtlas.png`", "`grassSeamless.jpg`" et "`suzanneTextureShade.png`".

On ne veut pas que les textures se répètent sur les objets, à l'exception du sol (voir section 1c).

On veut que les textures aient un fini lisse, à l'exception de la grande plateform où on veut voir les pixels de façon plus définie.

Pour les modèles 3D, l'attribut de coordonnée de texture est déjà défini dans le .obj. Le chargement de celle-ci et la configuration de l'attribut a déjà été fait dans la partie 1.

Partie 2b : Utilisation de texture dans les shaders

Vous allez devoir vous basez sur le shader de couleur uniform pour le shader de texture.

Pour le vertex shader, on a besoin de la matrice mvp appliqué sur les positions en entrée. On ajoute l'attribut de coordonnée de texture (vec2) en entrée et en sortie. La sortie est identique à l'entrée.

Pour le fragment shader, nous n'utiliserons plus le uniform de couleur. On utilise un uniform de texture avec les coordonnées de texture en entrée pour donnée la couleur au fragment.

Partie 2c : Répétition de texture

Le sol possède une texture dites "seamless". Une texture seamless est idéale pour couvrir une très grande région tout en ayant des textures relativement petites en espace mémoire. Pour ce faire, il faut nécessairement configurer cette texture dans le mode répétition pour pouvoir couvrir la totalité de la surface. De plus, il faudra activer le mipmap pour cette texture spécifiquement afin de retirer les artéfacts de rendu. On veut une progression lisse entre les niveaux de mipmaps.

Dans le cas du sol, sa texture sera répété 6 fois sur le long d'une arète. Une initialisation de VBO, EBO, VAO et de draw call comme au tp1 a été réalisé pour vous. Il faudra spécifier les attributs dans le constructeur et définir les coordonnées de textures.

Partie 2d : Utilisation de texture atlas

Pour la texture des tasses et des assiettes, au lieu d'être obliger de changer à plusieurs reprises la texture, on utilise un atlas de texture. Un atlas de texture combine plusieurs textures semblables ensembles dans une texture plus grande pour éviter des changements d'états trop fréquent.

Un nouveau vertex shader doit être implémenté (`cup.vs.glsl`) pour diviser les coordonnées selon le nombre de rangés et colonnes de l'atlas, puis donner le décalement pour aller chercher la bonne sous texture. Il faut envoyé un uniform de type entier pour avoir l'index de la tasse (entre 0 et 3 inclusivement). Pour calculer le décalement, la fonction `modf` sera très utile (ou l'opération modulo et division entière).

Pour les assiettes, on utilise un autre uniform de type entier pour savoir si on dessine une tasse ou une assiette. On l'utilisera pour appliquer un diviseur et décalement différent sur les coordonnées de texture, puisque la grosseur des sous-images est différente.

On utilisera le même fragment shader que celui des textures.

Partie 2e : Réduction des changements d'états

Afin de réduire le changement d'états d'OpenGL, le calcul des matrices devrait être fait au début de la frame et mémoriser sur la stack dans des variables locales. Cela va permettre de faire les bind de texture une seule fois et de dessiner tous les objets qui en ont besoin l'un à la suite des autres. Au final, les textures devraient être bind une seule fois par frame.

2 Exigences

2.1 Exigences fonctionnelles

TBH

2.2 Exigences non fonctionnelles

De façon générale, le code que vous ajouterez sera de bonne qualité. Évitez les énoncés superflus (qui montrent que vous ne comprenez pas bien ce que vous faites !), les commentaires erronés ou simplement absents, les mauvaises indentations, etc.

2.3 Remise

Faites la commande « `make remise` » afin de créer l'archive « **INF2705_remise_TPn.zip** » que vous déposerez ensuite dans Moodle. (Moodle ajoute automatiquement vos matricules ou le numéro de votre groupe au nom du fichier remis.)

Ce fichier `zip` contient tout le code source du TP (`makefile`, `*.h`, `*.cpp`, `*.glsl`, `*.txt`).

ANNEXES

A Liste des commandes

Touche	**Description**
`ESC`	Quitter l'application
`SPACE`	Afficher/Cacher la souris
`W`	Déplacer la caméra vers l'avant
`S`	Déplacer la caméra vers l'arrière
`A`	Déplacer la caméra vers la gauche
`D`	Déplacer la caméra vers la droite
`Q`	Déplacer la caméra vers le bas
`E`	Déplacer la caméra vers le haut
`UP`	Tourner la caméra sur l'axe +x
`DOWN`	Tourner la caméra sur l'axe -x
`LEFT`	Tourner la caméra sur l'axe +y
`RIGHT`	Tourner la caméra sur l'axe -y