TP n°3 : Maillages et Scènes

Le but du TP est de définir une structure de données représentant des maillages et de créer des scènes complexes.

Téléchargez l'archive du TP3 qui est sur Moodle et décompressez-la. Il est possible que les raccourcis des dossiers `libs` et `data` (liens logiques entre dossiers) ne fonctionnent pas. Vous devrez alors soit les refaire, soit recopier les dossiers directement.

Le TP est en plusieurs parties successives, liées au cours. Seule la partie « 02-Mesh » est totalement nécessaire aux autres. Les autres sont assez indépendantes.

Le fait que, sur certains systèmes (Windows), les dossiers `libs` et `data` ne puissent pas être des liens logiques pose un problème au navigateur : le « cross-origin resource sharing » est interdit, voir https://developer.mozilla.org/fr/docs/Web/HTTP/CORS. Pour éviter cette erreur, il y a un script Python à lancer dans chaque projet, `server.py`, et qui consiste en un serveur HTTP sur http://localhost:8000 — cette page correspond à `main.html`, et ce script redirige toutes les références à `libs` et `data` vers les dossiers au dessus.

1. Apprentissage de la bibliothèque glMatrix : 00-Maths

Il y a des exercices assez variés, pour comprendre à quoi servent les opérateurs sur les vecteurs et les matrices. Le corrigé est là, il reste à le comprendre entièrement et à en tirer quelque chose.

Par exemple, vous pourriez mettre en œuvre le calcul du vecteur miroir de V par rapport à U (pas demandé, sauf si vous voulez vous perfectionner).

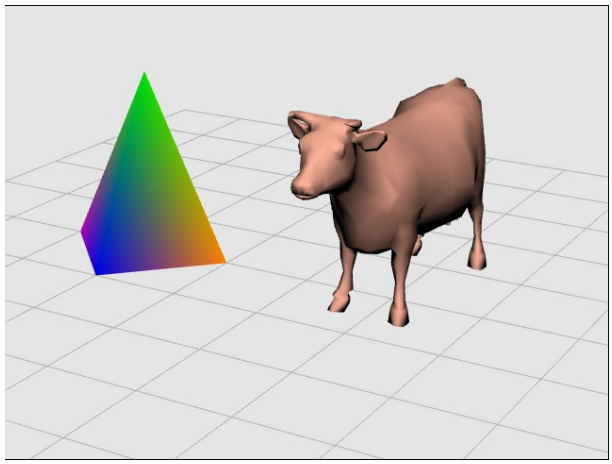
À noter que ces calculs sont généralement faits en GLSL dans le shader. Ceux qu'on fait dans le programme JavaScript sont la plupart du temps des préparations de matrices.

2. Un exemple de scène sans Mesh : 01-SansMesh

Juste pour montrer les limites de cette approche. On n'ira pas loin comme ça.

Cette scène peut servir à expérimenter les transformations matricielles : composition de translations et rotations.

3. Complétion de la classe Mesh : 02-Mesh



Le cours explique différents aspects de la classe `Mesh`. Vous allez devoir les programmer. Son source est dans le dossier `libs` car elle est utilisée dans tous les projets du TP.

3.1. Relations entre maillage, sommets et triangles

Compléter les constructeurs de `Vertex` et de `Triangle`.

3.2. Remplissage des VBOs

Ensuite, complétez la méthode `buildVBOs` de la classe `Mesh`. Elle doit remplir tous les VBOs demandés par les shaders : `m_VertexBufferId`, `m_ColorBufferId`, `m_NormalBufferId` et `m_FacesIndexBufferId`.

Lorsque ce sera fait, vous devriez voir apparaître la pyramide ainsi que la vache. Par contre, cette dernière est dessinée en noir, parce que ses normales sont nulles.

3.3. Calcul des normales

Compléter les méthodes `computeNormal` des classes `Vertex` et `Triangle`.

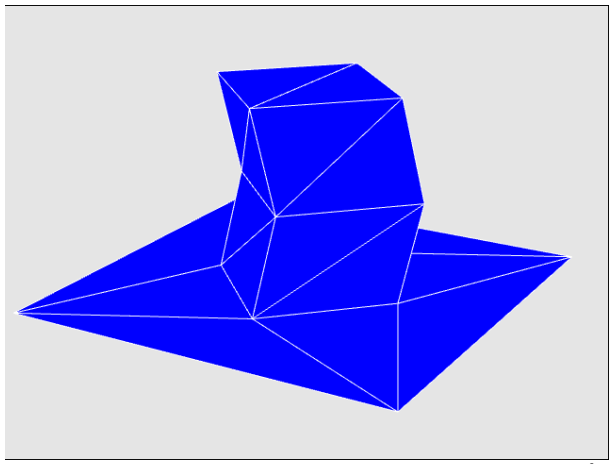
Lorsque ce sera fait, la vache sera dessinée correctement. Son shader gère une sorte d'éclairement diffus. Nous étudierons les questions d'éclairage et de matériaux la semaine prochaine.

3.4. Dessin des arêtes

Pour dessiner les arêtes, il faut compléter les méthodes `buildEdgesVBO` et `onDrawEdges`. La méthode de dessin doit dessiner des lignes le long des bords des triangles, donc il faut définir un autre VBO d'indices. Voici la démarche :

- `buildEdgesVBO` : il suffit de générer un VBO d'indices contenant 3*2 points : les trois côtés de chaque triangle, point de départ et point d'arrivée. Ce VBO doit s'appeler `m_EdgesIndexBufferId`.
- `onDrawEdges` : recopier le code de `onDraw` et enlever tout ce qui concerne les couleurs et les normales. Faire en sorte de tracer des lignes au lieu de triangles.
- Constructeur de la classe `Mesh` : il doit créer un shader spécifique pour dessiner les arêtes. Vous pouvez copier celui qui dessine la plaque de l'exercice 03-Extrusion. Le nouveau shader sera identifié par `m_EdgesShaderId`, et il aura des variables telles que `m_EdgesVertexLoc` et `m_EdgesMatPVMLoc`.

4. Programmation d'une extrusion : 03-Extrusion

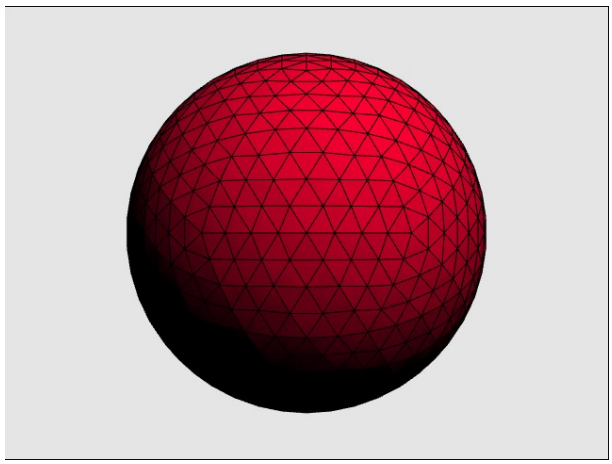


Le cours explique les principes d'une extrusion, il vous reste à la programmer dans la classe `Plaque`. Deux triangles sont à faire sortir du plan, comme une sorte de chapeau.

Il est assez compliqué, donc ce n'est pas demandé, de supprimer les cloisons construites par les arêtes partagées entre les triangles, à l'intérieur du chapeau. Il faudrait arriver à construire une sorte de contour regroupant les arêtes autour des triangles à extruder, et que c'est ce contour qu'il faut transformer en cloison. La modélisation de maillage proposée ne représente pas les arêtes, donc c'est compliqué à faire.

Vous en profiterez pour voir comment on trace les contours des polygones, en dessinant des arêtes à l'aide d'un VBO supplémentaire. Pour simplifier, j'ai repris le même shader pour donner la couleur.

5. Programmation d'une subdivision : 04-Subdivision



Le cours explique comment on subdivise un maillage. Chaque triangle est remplacé par quatre triangles bâtis sur les milieux des côtés. L'un des problèmes est la nécessité de partager les milieux des côtés, c'est à dire faire en sorte que d'un triangle à l'autre, quand on les subdivise tous les deux, les milieux créés soient les mêmes objets JavaScript, mais dans un premier temps, vous pouvez vous passer de le faire.

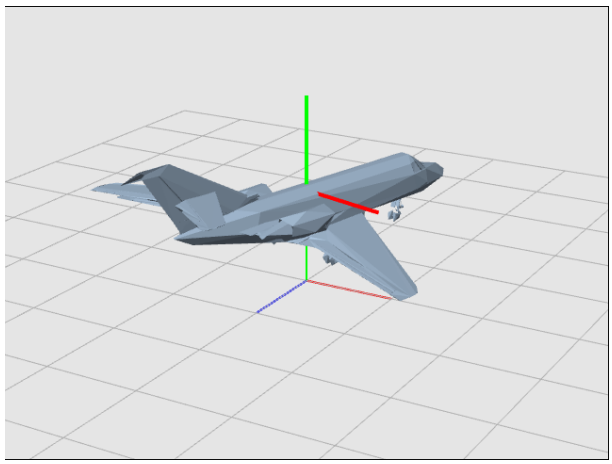
En fait, c'est assez compliqué à faire. Soient deux triangles contigus (A,B,C) et (D,B,A). Actuellement, le milieu entre A et B créé pour le premier triangle n'est pas celui du second triangle. Ils sont au même endroit, mais ce sont deux points distincts. Or il faudrait que ça soient les mêmes pour que le calcul des normales soit correct…

Une solution consiste à créer, sur chaque sommet, un dictionnaire (classe `Map`, allez voir sa documentation, notamment ses méthodes `get` et `set`) associant un sommet et le

milieu correspondant. Par exemple pour A, il y a un dictionnaire associant B au milieu entre A et B : milieu[B] = mAB, et pareil sur B mais inversé : milieu[A] = mAB. Alors quand on crée un nouveau sommet mXY, milieu entre X et Y, on le met à la fois dans X : milieu[Y] = mXY et dans Y : milieu[X] = mXY. De cette manière les milieux ne sont pas en double.

L'une des possibilités de la subdivision est de décaler les coordonnées, par exemple à une distance constante de 1 du centre. Il suffit de normaliser les coordonnées. En partant d'un solide de Platon, et en répétant cette subdivision lissée, on obtient une sphère.

6. Angles d'Euler : 05-Angles Euler



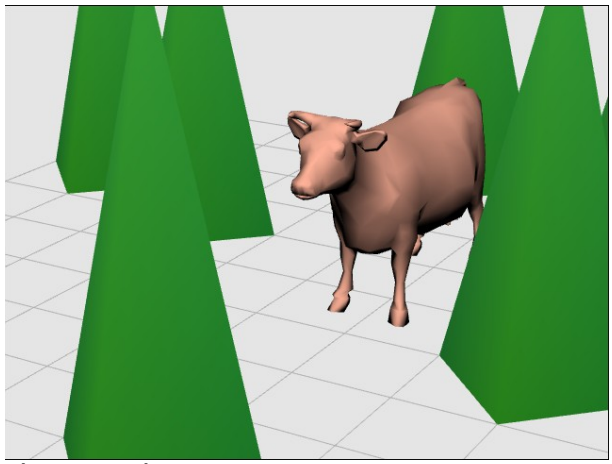
Il s'agit de jouer avec l'orientation de l'avion et de comprendre le rôle des rotations ainsi que l'ordre dans lequel les appliquer. Vous devrez ajouter les trois rotations lacet, roulis et tangage dans le bon ordre.

Vous êtes libre d'appliquer les rotations que vous voulez. Vous pouvez par exemple mettre l'avion en position de décollage, de virage en montant, de virage en descendant, d'atterrissage. Cette partie est uniquement pour votre compréhension de ces concepts.

Vous allez peut-être être gêné(e) par le fait de ne pas pouvoir orienter la scène à la souris. Dans ce cas, passez à la partie suivante puis revenez ici insérer le code nécessaire.

Le maillage d'avion qui est fourni est outrageusement simplifié parce que le maillage d'origine est énorme et un peu trop lent à charger. Ce qu'il faudrait, c'est définir un autre format de fichier que OBJ, un format qui serait quasiment une vidange des VBO afin d'être lu et exploité très rapidement. D'autre part, cet avion n'a pas de belles couleurs. Les matériaux et les textures sont au programme de la prochaine semaine.

7. Caméra type plateau tournant : 06-Camera plateau



Les pyramides sont censées représenter des arbres. C'est moins coûteux que des arbres définis par de nombreux polygones, surtout que les maillages d'arbres qu'on peut trouver sur internet font largement appel aux textures, c'est à dire des images, afin de réduire le nombre de polygones. Si on devait modéliser chaque feuille à l'aide de triangles, le maillage compterait des centaines de milliers de facettes. Les textures sont au programme du cours de la semaine prochaine.

La classe `Scene` ne gère pas complètement correctement la souris. Les mouvements avec le bouton enfoncé doivent se traduire par des modifications sur les variables `m_Elevation` et `m_Azimut`. Également, dans la fonction d'affichage, il faut tenir compte de ces angles.

Faites en sorte qu'on puisse avancer ou reculer par rapport à la scène à l'aide des touches Z et S ; elles doivent agir sur la variable `m_Distance`.

D'autres touches pourraient permettre de décaler le pivot des rotations latéralement : Q et D, ainsi que A et W pour monter et descendre.

8. Caméra première personne

Ensuite, faites une copie du projet 6 et nommez-le « 07–Camera 1P ». Votre travail consiste à implémenter la caméra de type première personne. Son fonctionnement est expliqué dans le cours.

Vous veillerez à ce que les mouvements soient agréables : la souris doit notamment être ralentie et les « pas » des touches Z,S,Q,D adaptés afin de faire des mouvements agréables. En fait, il faudrait adapter ce domaine : il est trop restreint pour une caméra première personne. Il faudrait espacer les objets. Vous pouvez aussi élargir le champ de vision.

Faites en sorte qu'on ne puisse pas aller sous le sol ni s'élever dans les airs.

Rajoutez quelques actions avec les touches : H pour réinitialiser la caméra, E pour « zoomer » c'est à dire rendre le champ de vision plus étroit, et R pour élargir le champ de vision.

Il est possible de gérer la touche espace pour sauter, mais c'est un peu compliqué : il faut appliquer les équations différentielles d'Euler pour gérer le mouvement. Pour commencer, il faut ajouter au moins une variable : la vitesse verticale, Vy (on peut ajouter un vecteur vitesse complet, voir la remarque finale ci-dessous). Ensuite, à chaque instant dans la méthode onDrawScene, il faut calculer ceci :

Vy = Vy + g*dt ; // g étant la constante de gravitation, -9.81 sur Terre, mais à adapter ; dt est le temps écoulé depuis le précédent calcul (il faut enregistrer le Utils.Time précédent et calculer la différence).

Ensuite, il faut modifier InvPosCam.y = InvPosCam.y + Vy*dt, et empêcher que la valeur soit inférieure à la hauteur normale de vue.

Pour finir, le rôle de la touche espace consiste à donner une vitesse initiale positive. Ça va donc faire monter la caméra, dans un premier temps, mais avec l'écoulement du temps, cette vitesse va être grignotée peu à peu, devenir négative, faire redescendre la caméra. Si on gère un vecteur vitesse complet, alors il faut l'initialiser avec des composantes x et z dépendantes de la direction de vision et aussi des touches Z et autres qui sont enfoncées en même temps que l'espace…

9. Rendu de TP

Déposer un fichier zip contenant votre dossier du TP3, contenant le dossier `libs` (dont le fichier `Mesh.js`) et un seul exemplaire du dossier `data` (au niveau de la racine de votre projet). Faites attention à la taille du fichier zip.

Important : vos projets doivent fonctionner avec le script `server.py`, tels quels, sortis de l'archive, sur une autre machine et système que le vôtre.