BSI TP1: Surfaces paramétriques et élimination des parties cachées

Loïc Simon

September 16, 2015

L'objectif de ce TP est d'appréhender les bases de l'API d'OpenGL moderne au travers d'une sur-couche orientée objet, et qui servira de cadre pour ce TP et les suivants. Cette sur-couche repose, sur la librairie GLFW pour créer un contexte OpenGL Core Profile et gérer les interactions utilisateur.

1 Instructions

1.1 Récupération du code et premiers tests

Pour ce tp, comme les suivants vous devez continuer à travailler sur le répertoire git commun a tous les TPs. Pour rappel, pour compiler le projet en ligne de commande, vous devez exécuter les commandes suivantes:

```
cd $HOME/Bureau/TPSI # adapter en fonction de votre configuration
cd GLLabs
cd build
make
```

Une fois la compilation terminée, vous disposez d'un exécutable que vous pouvez tester:

```
./Minimal/minimal
```

Dans ce TP vous travaillerez uniquement sur le fichier `Minimal/minimal.cpp`. Dans son état original, l'exécutable créé, affiche une scène composée d'un cube multicolore.

Attention! N'oubliez pas de commiter régulièrement. Pour ce faire, vous pouvez utiliser la commande:

```
git commit -am "A nice log message" # Adapter le message en conséquence.
```

1.2 Structure du répertoire

Pour rappel, le répertoire est constitué de plusieurs dossiers ayant chacun leur utilité:

- `resources` est destiné a stocker les ressources de type GLSL, textures, maillage.
- `include` contient un unique header `lightGlAPI.hpp` qui implémente l'API de la sur-couche d'OpenGL utilisée dans les TP.
- `utils` contient des routines de plus bas niveau, utilisées dans l'API précédente. Ces fonctionnalités proviennent pour la plupart du tutoriel: `http://www.opengl-tutorial.org/`. Ces routines ne seront normalement pas utilisées directement.
- `Minimal` contient le code nécessaire pour ce premier TP. D'autres répertoire du même type seront ajoutés pour les autres TPs.

1.3 Système d'exercices

D'autre part, dans le code, des sections sont réservées pour les réponses aux exercices. Elles sont du type:

```
1    if(controls.exercice==i)
2            /*!todo Instructions pour l'exercice i*/
```

Une telle section doit être remplie pour l'exercice `i` en suivant les instructions en commentaire. Certaines sections sont identiques pour plusieurs exercices, et certains exercices peuvent être concernés par plusieurs sections. *Respectez bien cette structure !*

Au départ le code est incomplet pour tous les exercices à part l'exercice 0 (qui correspond à la scène de base). Lorsque vous aurez répondu, vous pourrez naviguer entre les différents exercices en appuyant sur la touche 'e' pour passer à l'exercice suivant, la touche 'E' pour le précédent, ou utiliser le pavé numérique pour passer directement à un exercice donné.

Vous pouvez aussi obtenir une aide sur les différent raccourci clavier en tapant 'h'. L'aide s'affichera sur le terminal.

2 Exercices

2.1 Exercice 0: Compréhension

Prenez le temps d'analyser le code dont vous disposez déjà, de le comprendre et de poser des questions sur les parties qui vous semblent obscures.

Q1. Expliquez les étapes principales pour envoyer au GPU la description d'un maillage.

Q2. Comment est spécifié le lien entre les variables du shaders ("in") et les attributs correspondant du VAO? Donnez les appels de routines OpenGL qui concrétisent ce lien.

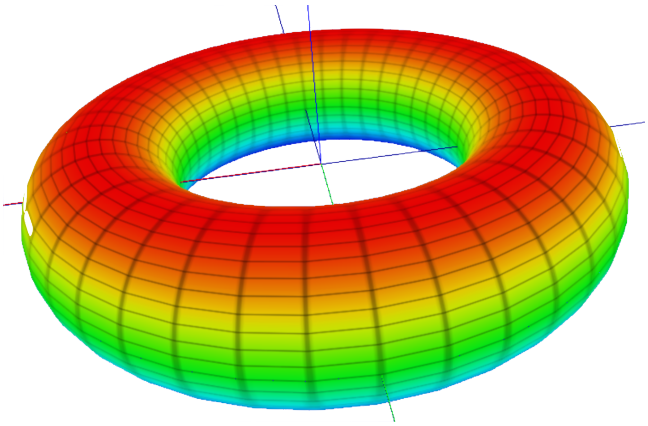

Figure 1: Le tore en dégradé de couleur.

2.2 Exercice 1: Activation du back-face culling

Q3. Le rendu original est erroné. Expliquez à l'aide d'un schéma la raison de ce problème.

Activez le "back-face culling" pour éliminer ce problème.

2.3 Exercice 2: Création d'un tore

En vous inspirant des fonctions équivalentes pour le cube, remplissez le corps des fonctions utiles à la définition et au dessin du tore.

Pour créer le maillage du tore vous utiliserez la définition du tore en tant que surface paramétrée. Dans un premier temps, le tore sera coloré entièrement en rouge, puis en dégradé linéaire allant du rouge au bleu (cf. Figure 1).

Vous vous arrangerez par ailleurs pour que le centre tore soit placé à une distance 1 du centre du cube.

Q4. Expliquez à l'aide d'un schéma la façon dont le tore est paramétré. Vous fournirez la formule de la paramétrisation, et l'expliquerez via le schéma.

Q5. Expliquez comment vous avez obtenu la normale en un point de la surface du tore. Vous utiliserez une formule analytique basée sur la paramétrisation du tore.

Q6. Expliquez comment vous avez entrepris le placement du tore. Attention, il est déconseillé de réaliser ce placement en modifiant les coordonnées des vertices, pourquoi?

2.4 Exercice 3: z-buffer

Normalement à ce point du TP les dessins du cube et du tore doivent se mélanger de façon incohérente. Activez le z-buffer pour éliminer ces incohérences.

Q7. Expliquez le pourquoi des incohérences mentionnées. Illustrez de façon graphique.

Q8. Expliquez à quoi sert le z-buffer. Vous pourrez vous appuyer sur une description pseudo-code de l'algorithme.

2.5 Exercice 4: Application indépendante

Intégrez les fonctionnalités de ce TP (cube, tore, back-face culling, z-buffer) dans le code indépendant `fromScratch/myOwnOpenGLProg.cpp`. Par contre, vous n'utiliserez pas la sur-couche du fichier `lightGlAPI.h`.

3 Compte-rendu du TP

Tous les TPs sont a effectuer en binôme. Seul le code devra être rendu. Pour rappel, le rendu du code sera fait grâce à votre répertoire git public qui doit être opérationnel via le chemin suivant:

```
1  $HOME/public_html/gits/GLLabs.git
```

Votre chargé de TP pourra alors le récupérer via la commande:

```
1  git clone http://www.ecole.ensicaen.fr/~YOURLOGIN/gits/GLLabs.git
```

Aucun rapport n'est obligatoire. Toutefois je vous conseille de savoir répondre aux questions posées dans les énoncés, car elles récapitulent les points clés de ces TP.

Bon courage!