



**POLYTECHNIQUE
MONTREAL**

LE GÉNIE
EN PREMIÈRE CLASSE

DÉPARTEMENT DU GÉNIE INFORMATIQUE ET
GÉNIE LOGICIEL

INF8702

INFOGRAPHIE AVANCÉE

TP1 :
INTRODUCTION AUX VBOS, IBOS, VAOS
ET AUX NUANCEURS EN GLSL

OBJECTIFS DU LABORATOIRE

Le but de ce premier laboratoire est de familiariser l'étudiant avec un environnement OpenGL 4+ et ainsi qu'avec la transmission de données par tampons OpenGL vers des nuanceurs de sommets et de fragments simples. L'étudiant sera amené à se familiariser avec l'utilisation de nuanceurs différents pour des objets différents.

Ce premier laboratoire n'est pas noté. Le barème de correction à la fin du présent document est présenté à titre indicatif uniquement.

MATÉRIEL

On met à votre disposition les fichiers suivants :

- Une solution Visual Studio contenant le projet **TP1** et le projet utilitaire **ObjParser** (ce dernier ne sera pas utilisé cette semaine, mais est disponible à ceux souhaitant se familiariser avec ce module dès maintenant). La structure de base de la solution sera utilisée pour la majorité des prochains laboratoires. Il vous est donc utile de l'inspecter et de vous y familiariser dès maintenant.
- Deux nuanceurs (*basicSommets.glsl*, *basicFragments.glsl*) pour l'affichage d'un plan fait de quadrilatères
- Deux nuanceurs (*cubeSommets.glsl*, *cubeFragments.glsl*) pour l'affichage d'un cube.

CONTRÔLES

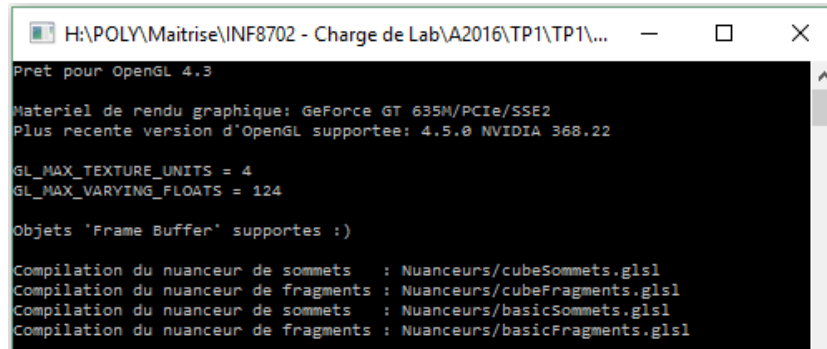
Le projet fourni vous permet quelques contrôles de base avec la souris et le clavier grâce à GLFW afin de vous aider à visualiser/débugger l'environnement. Vous serez amenés au cours des prochains laboratoires à ajouter certaines fonctionnalités de contrôle.

Entrée(s)	Effet
R	Active / désactive la rotation du cube
A, S, W, D	Effectue des translations de caméra
Q	Quitte l'application
I	Active l'affichage par seconde dans la console de la position de la caméra et du temps de calcul inter-trame
P	Modifie la matrice projection pour activer/désactiver la perspective
C	Permet d'activer/désactiver la rotation de caméra grâce aux mouvements de souris
Mouvements de souris	Change la direction de la caméra si <code>CVar : :mouseControl</code> est vrai.

1. Se familiariser avec la structure du projet Visual Studio

Ceci n'est pas un requis mais vous aidera grandement à accélérer vos prochains travaux si vous jetez déjà un coup d'œil aux différents modules de la solution, telles les classes CMaterial, et CLumiere.

Une fois compilée, la solution devrait produire un affichage semblable :



```
H:\POLY\Maitrise\INF8702 - Charge de Lab\A2016\TP1\TP1\...
Pret pour OpenGL 4.3
Materiel de rendu graphique: GeForce GT 635M/PCIe/SSE2
Plus recente version d'OpenGL supportee: 4.5.0 NVIDIA 368.22

GL_MAX_TEXTURE_UNITS = 4
GL_MAX_VARYING_FLOATS = 124

Objets 'Frame Buffer' supportes :)

Compilation du nuanceur de sommets : Nuanceurs/cubeSommets.glsl
Compilation du nuanceur de fragments : Nuanceurs/cubeFragments.glsl
Compilation du nuanceur de sommets : Nuanceurs/basicSommets.glsl
Compilation du nuanceur de fragments : Nuanceurs/basicFragments.glsl
```

Figure 1 : Affichage initial dans la console

Les détails devraient varier en fonction de la carte graphique utilisée. Une fenêtre bleue et vide devrait aussi apparaître :



Figure 2 : Résultat visuel initial

2. Se familiariser avec les différentes librairies utilisées :

- a. [OpenGL Extension Wrangler Library \(GLEW\)](#)
Nous permet d'accéder aux différentes fonctionnalités/definitions OpenGL.
- b. [GLFW](#)
Utilisée pour créer/gérer notre contexte et notre fenêtre OpenGL.
- c. [OpenGL Mathematics \(GLM\)](#)
Utilisée pour nos calculs d'algèbre linéaire (matrices/vecteurs).

3. Compiler et lier les nuanceurs et les programmes de nuanceurs

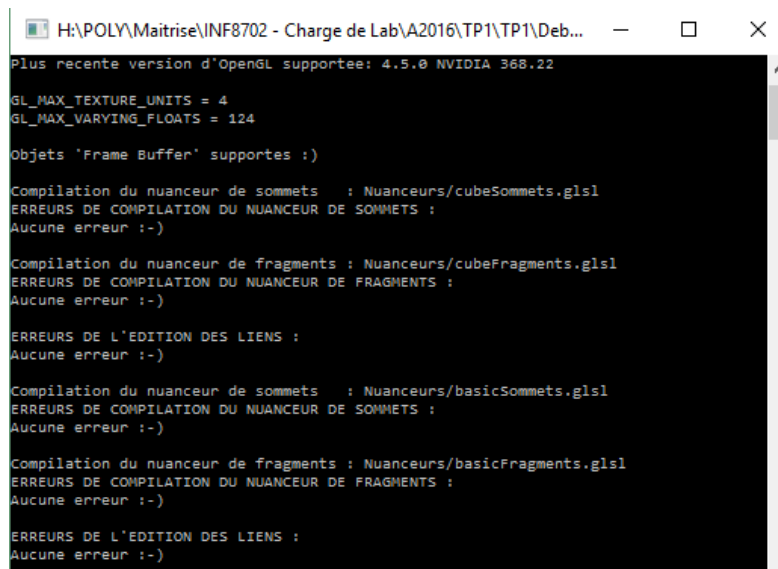
Les nuanceurs sont tout simplement des fichiers texte. Il faut donc d'abord charger leur contenu textuel dans des chaînes de caractères et leur faire passer ensuite le même processus de compilation et d'édition des liens que les différents fichiers d'un projet de programmation. Accessoirement, le projet met à votre disposition la fonction `::textFileRead()` qui réalise la lecture d'un fichier texte de nuanceur.

- Compléter la fonction :
 - `CNuanceurProg:: compilerEtLierNuanceurs ()`

4. Activer l'utilisation d'un programme de nuanceurs :

- Compléter la fonction :
 - `CNuanceurProg::activer()`

Une fois ces requis terminés, la console devrait afficher les messages suivants :



```
H:\POLY\Maitrise\INF8702 - Charge de Lab\A2016\TP1\TP1\Deb...
Plus recente version d'OpenGL supportee: 4.5.0 NVIDIA 368.22
GL_MAX_TEXTURE_UNITS = 4
GL_MAX_VARYING_FLOATS = 124
Objets 'Frame Buffer' supportes :)
Compilation du nuanceur de sommets : Nuanceurs/cubeSommets.glsl
ERREURS DE COMPILATION DU NUANCEUR DE SOMMETS :
Aucune erreur :-)
Compilation du nuanceur de fragments : Nuanceurs/cubeFragments.glsl
ERREURS DE COMPILATION DU NUANCEUR DE FRAGMENTS :
Aucune erreur :-)
ERREURS DE L'EDITION DES LIENS :
Aucune erreur :-)
Compilation du nuanceur de sommets : Nuanceurs/basicSommets.glsl
ERREURS DE COMPILATION DU NUANCEUR DE SOMMETS :
Aucune erreur :-)
Compilation du nuanceur de fragments : Nuanceurs/basicFragments.glsl
ERREURS DE COMPILATION DU NUANCEUR DE FRAGMENTS :
Aucune erreur :-)
ERREURS DE L'EDITION DES LIENS :
Aucune erreur :-)
```

Figure 3 : Résultats textes après compilation des nuanceurs

5. Créer les VBOs, IBOs et VAOs pour afficher un cube coloré.

Pour dessiner des formes ou modèles 3D à l'aide de nuanceurs, nous devons fournir à ceux-ci les informations qui leur seront utiles à l'aide de tampons (buffers) OpenGL. Dans la fonction `::initialiserCube()` de `main.cpp`, nous vous fournissons des tableaux de valeurs pour le cube coloré que nous voulons afficher. Il vous reste donc à enregistrer ces valeurs dans des *Vertex Buffer Object*, de spécifier l'ordre d'utilisation de ces valeurs dans un *Index Buffer Object*, et finalement sauvegarder les états OpenGL nécessaires et les spécifications d'attributs des sommets dans le VAO.

- Complétez la fonction :
 - `::initialiserCube()`

6. Dessiner un cube coloré

Vous devez maintenant appliquer au cube les transformations indiquées, puis fournir aux nuanceurs les différentes matrices nécessaires à la transformation des positions vers le référentiel d'écran, et finalement dessiner les triangles du cube.

- Complétez la fonction :
 - `:: dessinerCube()`

7. Implémenter les nuanceurs simples *cubeSommets.glsl* et *cubeFragments.glsl*

Ici, il vous faut uniquement transformer les positions en référentiel d'écran et propager la couleur du nuanceur de sommets aux nuanceur de fragments.

- Complétez les fonctions :
 - `cubeSommets.glsl :: main()`
 - `cubeFragments.glsl :: main()`

Après ces étapes l'affichage devrait montrer un cube coloré en rotation :

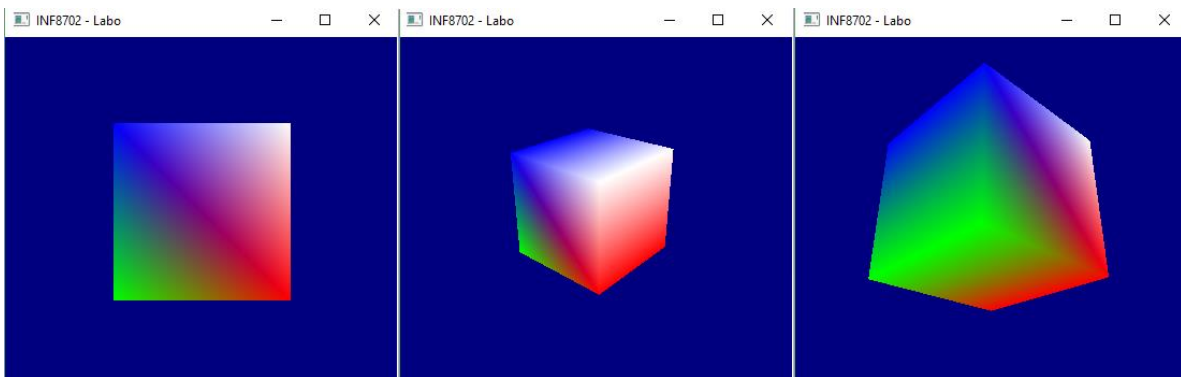


Figure 4 : Résultat visuel après complétion des requis 5, 6, 7

8. Afficher une grille de *quads* formant un plan

Vous devez ici utiliser la classe `CGrilleQuads` et afficher un plan fait de quadrilatères (*quads*). Les étapes vous permettant d'arriver à cela sont très similaires à celles pour afficher un cube, donc moins de détails sont donnés ici.

***Attention** : Vous verrez que toutes les informations de sommets sont fournies dans un seul VBO pour la grille. Cela a certaines implications dans l'initialisation des attributs et leur déclaration dans le nuanceur. Pour ce premier labo, seulement les positions seront utilisées, mais cela va changer au prochain TP.*

- Complétez les fonctions :
 - `CGrilleQuads :: creerLaGrille()`
 - `CGrilleQuads :: dessiner()`
 - `:: initialisation()`
 - `:: dessinerGrille()`
 - `grilleSommets.glsl :: main()`
 - `grilleFragments.glsl :: main()`

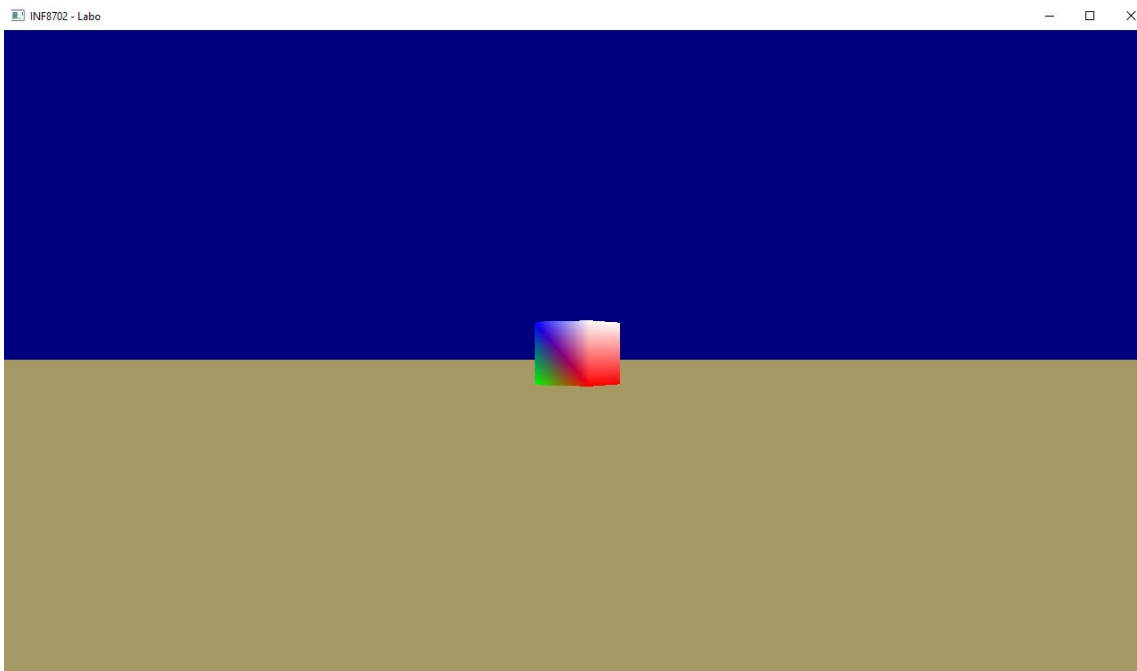


Figure 5 : Résultat visuel après complétion de la tâche 8

BARÈME DE CORRECTION

Description des requis	Points alloués
Compilation, et <i>linking</i> des nuanceurs	0.50
Créations et utilisation des VBOs, IBOs, et VAOs	0.75
Création des matrices modèle	0.25
Nuanceurs du cube fonctionnels	0.50
Nuanceurs du plan fonctionnels	0.50
Lisibilité du code	0.50
Total	3.00

QUESTIONS

1. Pourquoi la couleur de fond est-elle bleue ?
2. Quelle instruction indique à OpenGL de ne pas simplement donner aux fragments la couleur du dernier objet dessiné (ce qui rendrait le cube invisible vu de haut – on ne verrait que le plan) ?
3. Pourrions-nous utiliser plusieurs VBOs et IBOs pour un seul VAO ?
4. En quoi est-ce que l'utilisation de VAOs est intéressante ?

- Pipeline de rendu OpenGL :
 - https://www.opengl.org/wiki/Rendering_Pipeline_Overview
- Tutoriel/Rappel sur les matrices utilisant GLM (contient beaucoup de tutoriels intéressants) :
 - <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>
- Meilleure documentation OpenGL pour démêler les versions :
 - <http://docs.gl/>
- *Quick references* :
 - <https://www.khronos.org/files/opengl43-quick-reference-card.pdf>