DÉPARTEMENT DU GÉNIE INFORMATIQUE ET
GÉNIE LOGICIEL

INF8702
INFOGRAPHIE AVANCÉE

TP2 :
ÉCLAIRAGE PAR SOMMETS ET FRAGMENTS
TEXTURAGE, SKYBOX ET MIPMAP
PLACAGE DE RELIEF ET DE DÉPLACEMENTS

OBJECTIFS DU LABORATOIRE

Pour ce TP, nous plongeons dans le vif du sujet des nuanceurs de sommets et fragments et nous allons explorer plusieurs utilités courantes qu'ils ont dans une application graphique. Nous utiliserons ici les classes *CMatériau* et *CLumière* pour produire les effets visuels escomptés.

Vous travaillerez ici avec 3 modèles 3D différents qui utiliserons chacun leur propres nuanceurs : un maillage BSplinaire, un *skybox* et une grille de quadrilatères (comme dans le TP1) qui représentera une carte étudiante de Polytechnique.

1. **Éclairage, texturage et *mipmapping* - Gazon**
   Votre première tâche sera d'afficher une surface gazonnée et courbée (avec le maillage BSplinaire). Il vous faudra ici appliquer une texture, un matériel, ainsi que de l'éclairage ambient et diffus par fragment sur le modèle. Vous serez emmenés à appliquer du mipmapping également pour rendre le résultat du texturage plus intéressant.
2. **Texturage cubique – Skybox**
   Votre deuxième tâche sera d'englober la scène dans un Skybox pour améliorer le réalisme du rendu de l'environnement.
3. **Éclairage, multi-texturage, placage de déplacement, et placage de de relief – Carte Poly**
   Votre troisième tâche sera d'afficher un modèle de carte étudiante de Polytechnique possédant plusieurs propriétés intéressantes :
   - Possède des textures différentes sur chaque face
   - Souple/déformable
   - Possède un côté plus spéculaire et bosselé

***Note* :** À partir de maintenant, les TP doivent se faire en équipe de deux. S'il y a un nombre de personnes impair dans la classe, une personne seule pourra être acceptée.

MATÉRIEL

On met à votre disposition les choses suivantes :

- La solution Visual Studio intégrant un projet intitulé **TP2**
- Deux nouvelles classes :
  - *CSkybox*
  - *CGazon*
- Trois paires de nuanceurs pour le rendu des trois modèles :
  - *carteSommets.glsl & carteFragments.glsl*
  - *skyboxSommets.glsl & skyboxFragments.glsl*
  - *gazonSommets.glsl & gazonFragments.glsl*
- Quatre nouvelles textures :
  - cartePolyRecto.bmp
  - cartePolyVerso.bmp
  - gazon.bmp
  - uffizi_cross_LDR.bmp (skybox)
- Des nouvelles fonctionnalités clavier:
  - N : permuter le mode de filtrage en *minification*
  - M : permuter le mode de filtrage en *magnification*

- X : permuter l'état du bruit de Perlin
- E : permuter l'état de l'animation de la carte
- R : permuter l'état de rotation de la carte
- 1 : permuter l'état de la lumière ponctuelle
- 2 : permuter l'état de la lumière spot
- 3 : permuter l'état de la lumière directionnelle

DESCRIPTION DES TÂCHES

GAZON

1. **Éclairer le gazon**

   Dans l'état actuel de la solution Visual Studio, vous devriez pouvoir afficher un maillage BSplinaire qui se fait attribuer une couleur par défaut :

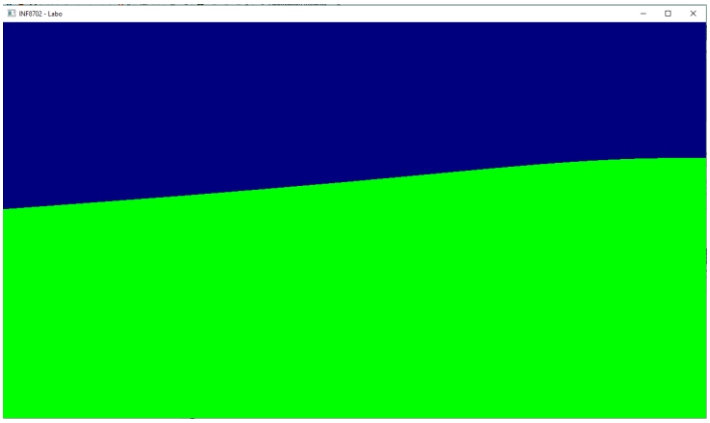

Figure 1 : Affichage initial de la scène. Le maillage BSplinaire possède une couleur par défaut.

Votre première tâche sera donc d'ajouter aux nuanceur `gazonSommet.glsl` les calculs d'éclairage manquants pour la contribution Ambiante et la contribution Diffuse. Vous devrez d'abord modifier la fonction

- `gazonSommet.glsl :: main ()`

… afin de bien calculer les vecteur lumières (allant de la surface à vers les lumières).

Vous devrez ensuite compléter les fonctions

- `gazonSommet.glsl :: pointLight()`
- `gazonSommet.glsl :: spotLight()`
- `gazonSommet.glsl :: directionalLight ()`

… afin de calculer ces contributions en fonctions des différentes lumières présentes dans la scène.

Finalement, il vous faudra modifier la fonction

- `gazonSommet.glsl :: flight ()`

… afin d'effectivement remplacer la couleur par défaut par la couleur calculée.

Vous devriez maintenant avoir une surface correctement éclairée par sommets :

Figure 2 : Éclairage de la surface avec la lumière ponctuelle.

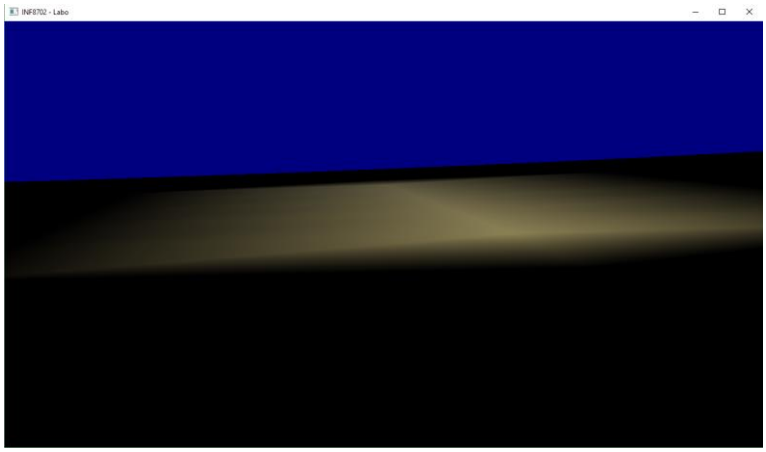

Figure 3 : Éclairage de la surface avec la lumière "spot".

2. Appliquer une texture au gazon

La surface est donc éclairée mais ne ressemble pas à du gazon ! C'est parce qu'elle ne possède pas de texture. Il faut donc ajouter la contribution de texture. Lors de la création du modèle, la construction et le chargement de textures sont effectués par les classes `CModeleAbstrait` et `CTexture2D`. Il vous est donc conseillé de d'y jeter un coup d'œil afin de bien comprendre comment l'information se rend vers les nuanceurs.

Pour avoir un résultat visuel intéressant, il est nécessaire de passer les coordonnées de texture aux nuanceur de fragments pour effectuer l'échantillonnage de la texture par fragment. Vous devrez donc compléter

- `gazonSommet.glsl :: main()`
- `gazonfragments.glsl :: main()`

… afin de bien ajouter la contribution de texture.

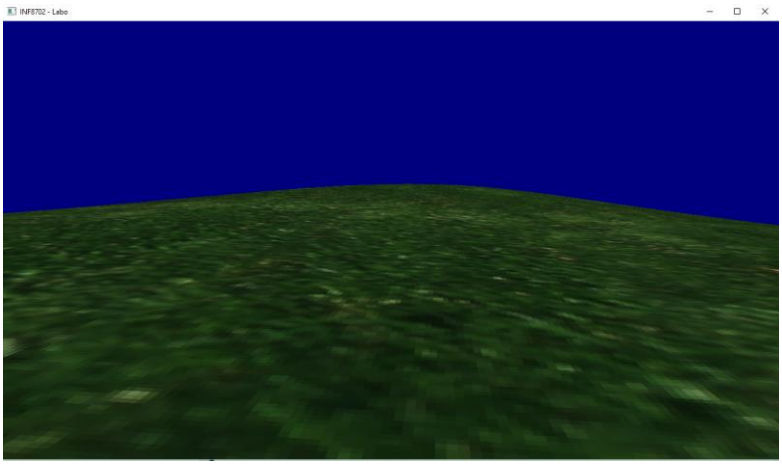

Figure 4 : Éclairage du gazon avec texture.

3. Illuminer par fragments

Vous aurez sûrement remarqué que l'éclairage avec seulement la lumière « spot » d'activée donne un faisceau mal défini que ne ressemble pas beaucoup à notre idée d'un « spot ». C'est parce que le programme de nuanceurs que vous avez complété effectue un éclairage par sommet. Cela signifie que les calculs d'éclairage utilisant les normales et les vecteurs « lumières » sont effectués par sommet, puis leurs résultats interpolés linéairement dans les nuanceurs de fragments. Avec un nombre limité de sommets, cela donne des artéfacts non souhaitables :

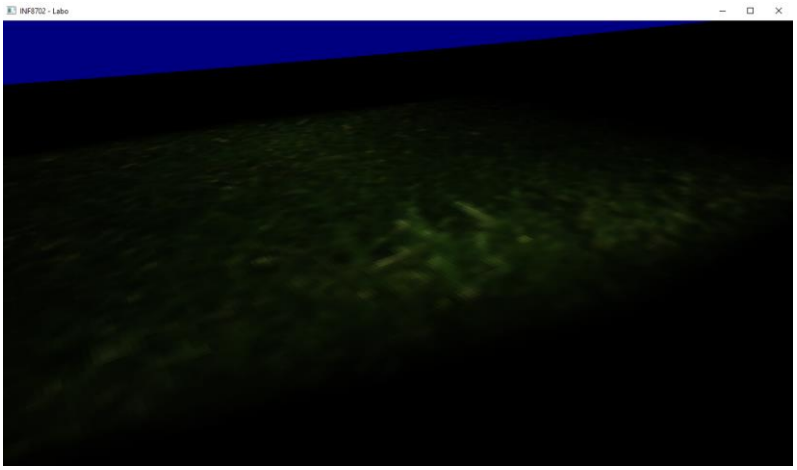

Figure 5 : Éclairage par sommets avec seulement la lumière "spot".

On vous demande donc de transférer ces calculs d'éclairage vers le nuanceur de fragments. Vous devrez donc fournir au nuanceur de fragments les normales et vecteurs lumières qui seront interpolés, au lieu de fournir des couleurs.

Vous devrez donc modifier les fichiers suivants

- `gazonSommet.glsl :: main()`
- `gazonfragments.glsl :: main()`

… afin d'effectuer un éclairage par fragments pour le modèle du gazon.

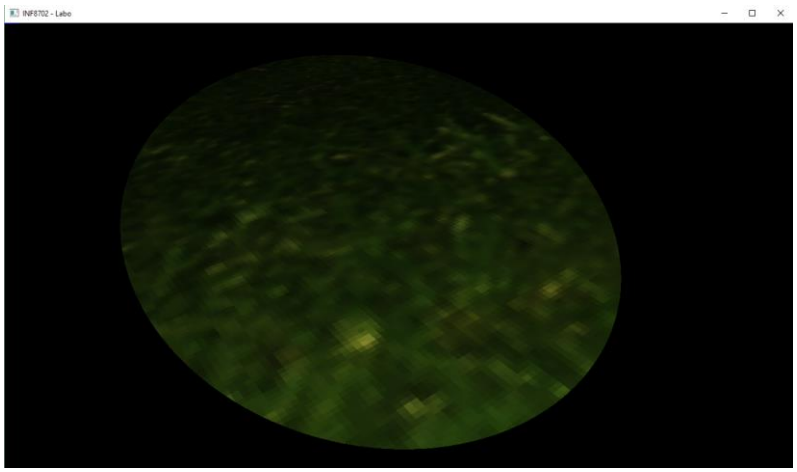

Figure 6 : Éclairage du gazon par fragment avec la lumière "spot".

4. **Utiliser plusieurs niveaux de *mipmaps***

Vous aurez peut-être remarqué que la grande étendue de ce modèle 3D dans la scène donne lieu à des artéfacts de crénelage dans les parties éloignées et de la pixellisation dans les zones rapprochées de la caméra. Ces effets sont indésirables et on désire ici les atténuer. On vous demande donc d'implémenter l'anticrénelage de la texture gazon par l'utilisation de la technique du mipmapping.

Pour ce faire, il vous faudra modifier la fonction

- `CTexture2D :: chargerTexture()`

… afin de générer la pyramide des mipmaps lors du chargement de gazon.bmp.

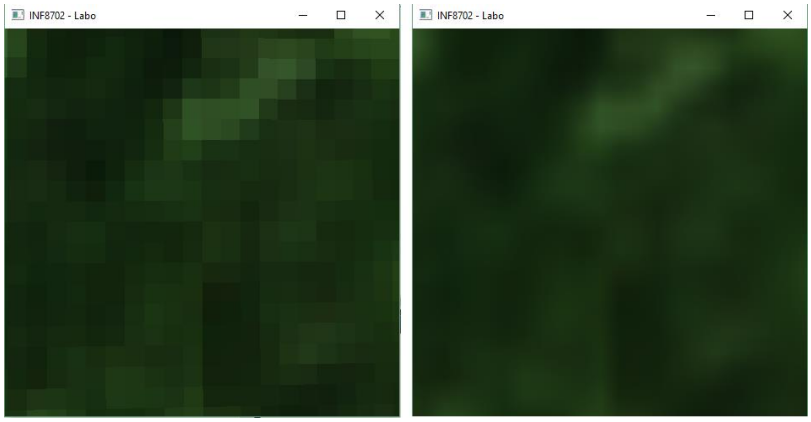

Figure 7 : Différence entre l'affichage du gazon sans *magnification* (gauche) et avec *magnification* (droite)

SKYBOX

1. **Activer le texturage dans les nuanceurs**

   Nous voulons ici se débarrasser de la couleur de fond par défaut pour placer notre scène dans un environnement intéressant. Nous utiliserons une technique grandement répandue de l'utilisation d'un skybox. La construction du modèle est déjà effectuée dans l'application, il ne vous reste plus qu'à l'afficher.

   - Complétez :
     - skyboxSommets.glsl ::main()
     - skyboxFragments.glsl ::main()

   Afin d'activer le texturage du cube de la skybox. Vous remarquerez que nous n'utilisons pas de coordonnées de texture ici. Nous n'en avons pas besoin.

   Vous devrez dé-commenter la ligne dans

     - `::dessinerScene()`

   … afin d'activer l'affichage du skybox. Une fois cela terminé, vous devriez pouvoir voir le skybox :

Figure 8 : Affichage initial du skybox après texturage.

2. **Dessiner le skybox en arrière-plan**

   Vous remarquerez qu'une fois le texturage complété, nous ne pouvons plus voir la totalité du gazon. C'est parce que notre skybox est moins volumineux que le gazon. Nous pourrions évidemment agrandir le cube qui le représente, mais nous aurons recours à une technique plus « propre » et non dépendante de la taille de nos modèles. Vous devez modifier la fonction

   - `::dessinerScene()`

   … afin de s'assurer avec des directive OpenGL que le dessin de la skybox sera toujours fait arrière-plan. Une fois ce requis complété, vous devriez avoir un rendu similaire à ceci :

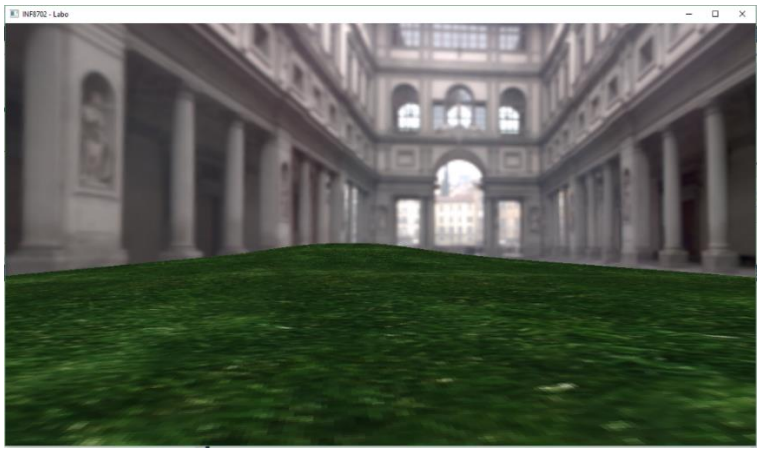

Figure 9 : Affichage du skybox en arrière-plan.

3. **Régler la perspective**

   Si vous bouger avec la caméra dans la scène vous remarquerez un drôle d'effet dû au fait de notre « petit » sybox et du « grand » gazon. Ces artéfacts sont facilement visibles en regardant vers le haut :

Figure 10 : Skybox avec déformation de perspective.

Un truc simple pour éviter ces artéfacts est de bouger le skybox avec la caméra. Vous devrez donc modifier la fonction

- `::dessinerSkybox()`

… afin de régler cet effet non-désirable en modifiant la matrice modèle du skybox. Le rendu ne devrait plus être déformé :

Figure 11 : Skybox sans déformation de perspective.

CARTE POLY

1. **Afficher la carte**
   Il vous suffira de dé-commenter une ligne dans

   - `::dessinerScene()`

   … afin ‘afficher une carte pour laquelle les calculs d’éclairage sont absents :

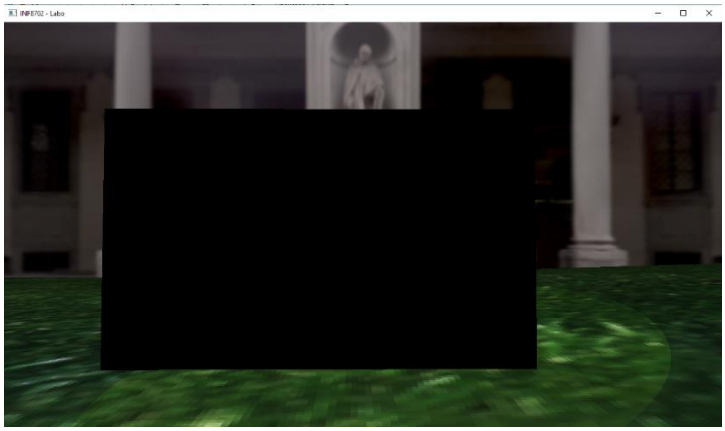

Figure 12 : Affichage initial de la carte.

2. Éclairer la carte

Il vous faudra compléter les fonctions d'éclairage de la carte afin de visualiser l'effet des lumières sur la grille de *quads.* Avec un pipeline de nuanceurs, rien ne nous empêche de calculer certaines contributions d'éclairage dans le nuanceurs de sommets et d'autres dans le nuanceurs de fragments. Ceci permet un certain contrôle au niveau de la vitesse de rendu et du réalisme visuel. Dans le cas présent, les contributions ambiantes et diffuses seront calculées au niveau des sommets, alors que la contribution spéculaire sera calculée au niveau des fragments. Vous devrez prendre en considération dans les calculs que nous voulons éclairer les deux côtés de la carte, peu importe l'ordre d'assignation des sommets dans le VBO.

Il vous faut donc compléter les fonctions

- `carteSommets.glgl::pointLight ()`
- `carteSommets.glgl::spotLight ()`
- `carteSommets.glgl::directionalLight ()`
- `carteSommets.glgl::frontLighting ()`
- `carteSommets.glgl::backLighting ()`
- `carteFragments.glgl::lightSpec ()`
- `carteFragments.glgl::main ()`

… afin d'activer l'éclairage sur la carte. Les calculs seront pour la plupart similaires à ceux effectués pour le modèle du gazon !

Figure 13 : Éclairage de la carte par lumière directionelle.

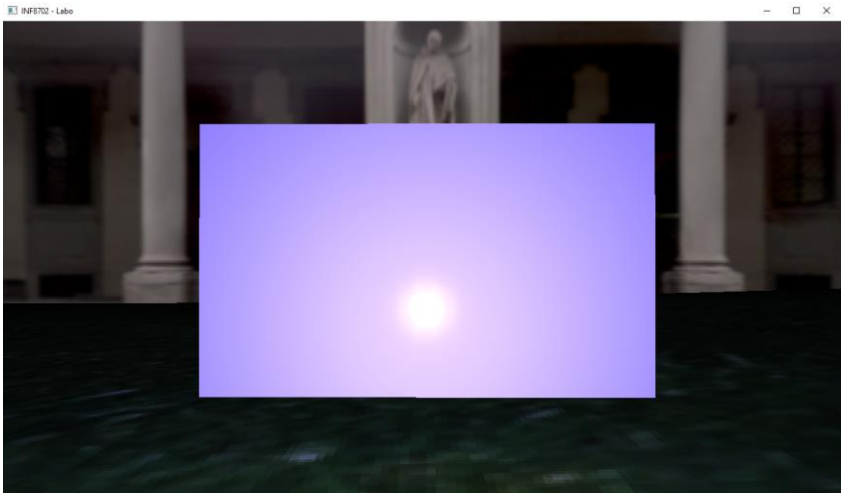

Figure 14 : Éclairage de la carte par lumière ponctuelle.

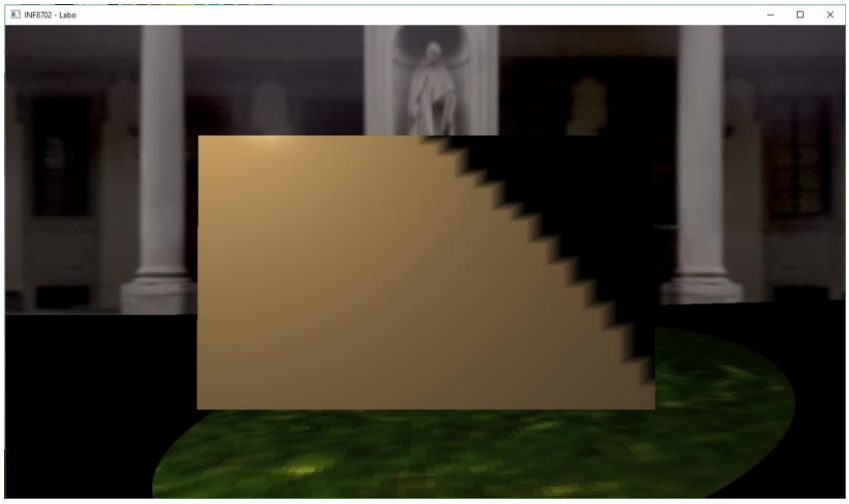

Figure 15 : Éclairage de la carte par lumière "spot".

3. **Texturer la carte**

   Avec une procédure similaire au gazon, vous devrez texturer la carte de Polytechnique. La différence étant ici le fait 2 textures différentes doivent être utilisées pour les deux faces de la carte.

   Complétez les fonctions

   - `carteSommets.glgl::main ()`
   - `carteFragments.glgl::main ()`

   … afin d'ajouter la contribution des textures à l'éclairage de la carte. Attention : les caractères devraient être lisibles sur les deux faces de la carte!

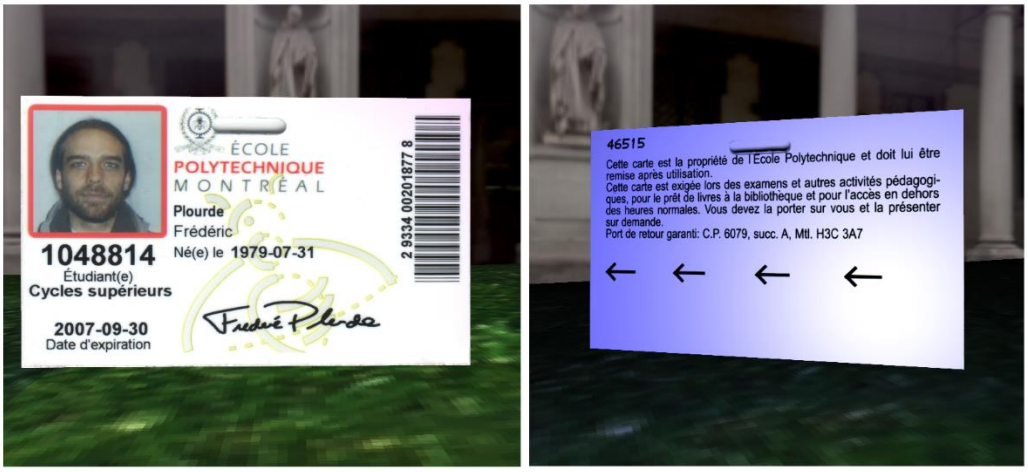

Figure 16 : Carte texturée sur les deux faces.

4. Placage de déplacement

Les calculs d'éclairage et de texturage ne sont pas les seules tâches possibles à effectuer dans nos nuanceurs de sommets et de fragments. Dans notre cas, nous voulons exprimer la souplesse de la carte de Polytechnique en implémentant un placage de déplacement procédural. Il s'agit donc d'implémenter un placage de déplacements de forme sinusoïdale sur l'ensemble des sommets du modèle 3D.

Vous devrez modifier la fonction

- `carteSommets.glgl::animation ()`

… afin de calculer correctement les grandeurs suivantes après le placage du déplacement du sommet :

- Position (la nouvelle position du sommet)
- Normal (la normale au sommet après le déplacement)
- Tangent (la tangente au sommet après le déplacement)

Pour vous aider, on inclut l'implémentation du delta-déplacement absolu du sommet en fonction de divers paramètres fournis en entrée et provenant de l'implémentation en OpenGL/C++ (coordonnées d'objet X et Y du sommet à modifier, et le temps courant) dans la fonction `carteSommets.glgl::animation()`. On y retrouve aussi le calcul de l'angle *thêta* utile au calcul de la normale et de la tangente.

Figure 17 : Résultats du placage visuel avec lumière ponctuelle (gauche) et lumière "spot" (droite).

5. **Implémenter un placage de relief pré-calculé de Perlin**

Vous avez vu en classe l'utilisation du bruit de Perlin pour texturer des objets en RGB. Mais pourquoi ne pas utiliser ce bruit de Perlin pour moduler directement les normales XYZ du placage de relief ?

Pour recréer l'effet des bosselures que vous pouvez observer sur le recto de votre carte d'identité de Polytechnique, on vous demande ici d'implémenter un placage de relief de Perlin. Il est dit « pré-calculé », car la carte de normales générée par un processus aléatoire de Perlin est déjà calculée pour vous côté application OpenGL puis injectée dans le nuanceur de fragments par l'entremise d'une texture RGB (appelée « normalMap ») qui pourra être lue afin de déterminer directement la normale perturbée XYZ de chacun des fragments de la carte d'identité.

La fonction `main.cpp::chargerBruitPerlin()`, côté application OpenGL, se charge de construire et de charger la texture de Perlin utilisée. Elle vous est fournie d'emblée. Vous n'avez pas à la modifier.

C'est plutôt du côté des nuanceurs qu'on vous demande de travailler. Vous devrez donc modifier la fonction suivante :

- `cartePolySommets.glsl :: tsTransform()`

… afin d'effectuer la conversion en espace tangent et de calculer correctement les varying Light0HV, Light1HV et Light2HV qui serviront à obtenir la contribution spéculaire correcte du côté nuanceur de fragments (ce sont les « half-vectors » du modèle de **Blinn-Phong**, nécessaires pour calculer les contributions spéculaires).

Vous devrez également modifier la fonction :

- `cartePolyFragments.glgl:: main()`

… afin de d'être en mesure d'échantillonner la carte de normales pour le fragment en cours et de perturber correctement la normale du fragment considéré.

Figure 18 : Bruit de Perlin correctement affiché avec une lumière ponctuelle.

Figure 19: Bruit de Perlin correctement affiché avec une lumière "spot".

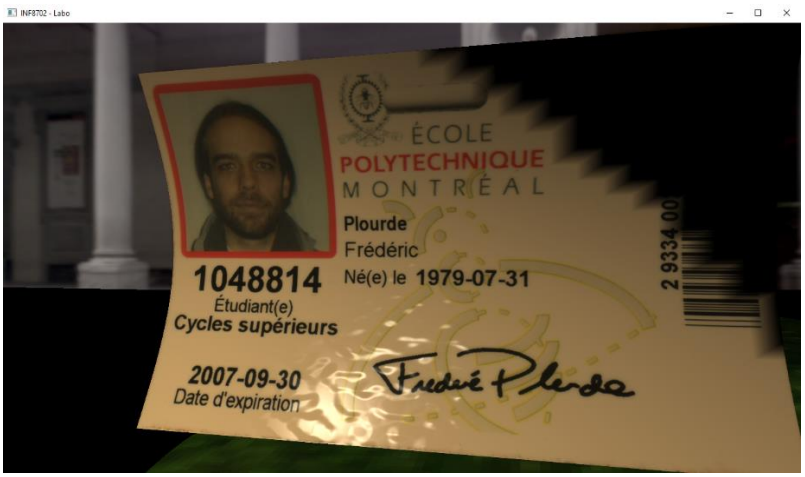

Figure 20 : Bruit de Perlin correctement affiché avec une lumière "spot" et le placage de déplacement.

QUESITONS

1. **Pourquoi est-ce nous semblons avoir deux niveaux de *magnification* et *minification* des textures avant même d'avoir construit des *mipmaps* ?**
2. **Nous vous avons demandé de calculer l'ajout spéculaire du modèle de Blinn-Phong. Dans quelle(s) circonstance(s) est-ce que ce modèle est avantageux ?**
3. **Quel aurait été l'effet visuel de perturber les normales pour le bruit de Perlin dans le nuanceur de sommets ? Pourquoi ?**
4. **Pourquoi utiliser l'espace tangent ?**
5. **Comment pourriez-vous visualiser n'importe quelle fonction 2D d'un domaine connu et limité avec un programme de nuanceurs en vous basant sur ce que vous avez accompli dans ce TP ? Énumérez les étapes.**

BARÈME DE CORRECTION

Description des requis	**Points alloués**
Gazon éclairé par fragments	1.00
Gazon adéquatement texturé	0.25
Mipmapping correctement implémenté	0.25
Skybox bien texturé	0.25
Skybox en arrière-plan	0.25
Skybox sans déformation de perspective	0.25
Carte bien éclairée (ambiant + diffus)	0.75
Carte bien éclairée (spéculaire)	0.25
Carte bien texturée (recto-verso)	0.50
Placage de déplacements fonctionnel	1.00
Conversion vers l'espace tangent fonctionnel	1.00
Placage de bruit de Perlin	0.25
Code lisible et bien commenté	0.50
Questions	3.50
Total	**10.00**

REMISE

- On demande une seule remise par équipe.
- **Ne remettez que les fichiers modifiés** durant le TP.
- Remettez un seul fichier .ZIP contenant :
  - Les fichiers modifiés (.cpp, .h, .glsl)
  - Un document (.txt, .pdf) contenant les réponses aux questions posées.
  - Un document facultatif (.txt, .pdf) contenant des commentaires sur un ou des aspects du TP (instructions, code fourni, bogues trouvés, etc.) dans le but de rendre les TP plus intéressants dans le futur.
- Nommez le fichier avec vos matricules et le nom du TP
  - Ex : « *MATRICULE1_MATRICULE2_TP2.zip* »