TP1 - Programmation 3D

Introduction à la programmation 3D

Université de Montpellier

Léa Serrano M1 IMAGINE

15 Septembre 2022

Lien de mon git pour ce tp : https ://github.com/LeaSerrano/M1-IMAGINE-Prog3D-TP1.git

Dans ce tp, nous allons chercher à afficher un objet issu d'un fichier OFF en 3D. Pour cela, nous allons devoir tracer ses vertices et ses triangles, ses normales et lui donner une couleur.

Table des matières

1 Exercice 1

Question 1

Lisez la documentation de ***glVertexPointer*** *sur opengl.org*

gLVertexPointer est une fonction d'OpenGL qui prend plusieurs paramètres en entrée : le nombre de coordonnées par vertice, le type de chaque coordonnée, une taille, et un pointeur vers les coordonnées du premier élément des vertices.
C'est cette fonction va nous permettre l'affichage des vertices de notre objet.

Question 2

Munissez la class Mesh d'un ***std : :vector<float>*** *contenant toutes positions des sommets. Ce vecteur doit être rempli une fois pour toutes, par exemple dans une fonction* ***Mesh : :buildVertexArray()*** *que l'on appelle après le chargement du maillage*

Nous allons donc ajouter dans notre classe Mesh, un vecteur stockant une liste de positions que nous allons remplir grâce aux vertices qui sont dans notre Mesh.

```
std::vector<float> positionArray;

void buildVertexArray() {
    positionArray.clear();
    for(unsigned int i = 0 ; i < vertices.size() ; i++) {
        positionArray.push_back(vertices[i][0]);
        positionArray.push_back(vertices[i][1]);
        positionArray.push_back(vertices[i][2]);
    }
}
```

Ex1-2 : création du vecteur contenant les positions des sommets et remplissage de ce vecteur

Question 3

Lisez la documentation de ***glDrawElements*** *sur opengl.org*

glDrawElements est une fonction d'OpenGL qui va prendre plusieurs paramètres en entrée : le mode de tracé que l'on veut, le nombre d'éléments à afficher, le type des indices et un pointeur vers l'endroit où sont stockés les indices.
Cette fonction va nous permettre l'affichage des triangles de notre objet.

Question 4

Munissez la class Mesh d'un ***std : :vector<unsigned int>*** *contenant tous les triangles les uns à la suite des autres. Ce vecteur doit être rempli une fois pour toutes, par exemple dans une fonction* ***Mesh : :buildTriangleArray()*** *que l'on appelle après le chargement du maillage*

Nous allons maintenant ajouter dans notre classe Mesh, un vecteur qui stocke une liste de triangles que nous allons remplir grâce aux triangles de notre Mesh.

```
std::vector<unsigned int> triangleArray;

void buildTriangleArray() {
    triangleArray.clear();
    for(unsigned int i = 0 ; i < triangles.size() ; i++) {
        triangleArray.push_back(triangles[i][0]);
        triangleArray.push_back(triangles[i][1]);
        triangleArray.push_back(triangles[i][2]);
    }
}
```

Ex1-4 : création du vecteur contenant les triangles des sommets et remplissage de ce vecteur

A la fin, nous allons afficher notre objet avec uniquement ses vertices et ses triangles. Voici le résultat obtenu :

Ex1 : rendu de l'objet après l'exercice 1

2 Exercice 2

Question 1

Lisez la documentation de ***glNormalPointer*** *sur opengl.org*

glNormalPointer est une fonction d'OpenGL qui va prendre plusieurs paramètres en entrée : le type de données, une taille et le pointeur vers l'endroit où sont les coordonnées de nos normales.
Cette fonction va donc nous permettre de tracer les normales de notre objet.

Question 2

Munissez la class Mesh d'un ***std : :vector<float>*** *contenant toutes normales des sommets sous la forme xyzxyzxyzxyz.... Ce vecteur doit être rempli une fois pour toutes, par exemple dans une fonction* ***Mesh : :buildVertexArray()*** *que l'on appelle après le chargement du maillage*

Nous allons donc ajouter dans notre Mesh, un vecteur contenant les normales aux sommets. Ensuite nous allons remplir ce vecteur avec l'attribut des normales du Mesh.

```
std::vector<float> normalArray;

void buildNormalArray() {
    normalArray.clear();
    for(unsigned int i = 0 ; i < normals.size() ; i++) {
        normalArray.push_back(normals[i][0]);
        normalArray.push_back(normals[i][1]);
        normalArray.push_back(normals[i][2]);
    }
}
```

Ex2-2 : création du vecteur contenant les normales des sommets et remplissage de ce vecteur

Question 3

Activer les normal arrays ***(glEnableClientState(GL_NORMAL_ARRAY);)***

On va donc activer les normal arrays que l'on va ajouter dans la fonction d'initialisation. Sachant qu'on avait déjà activé les vertex arrays.

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
```

Ex2-3 : activation des normal arrays

Question 4

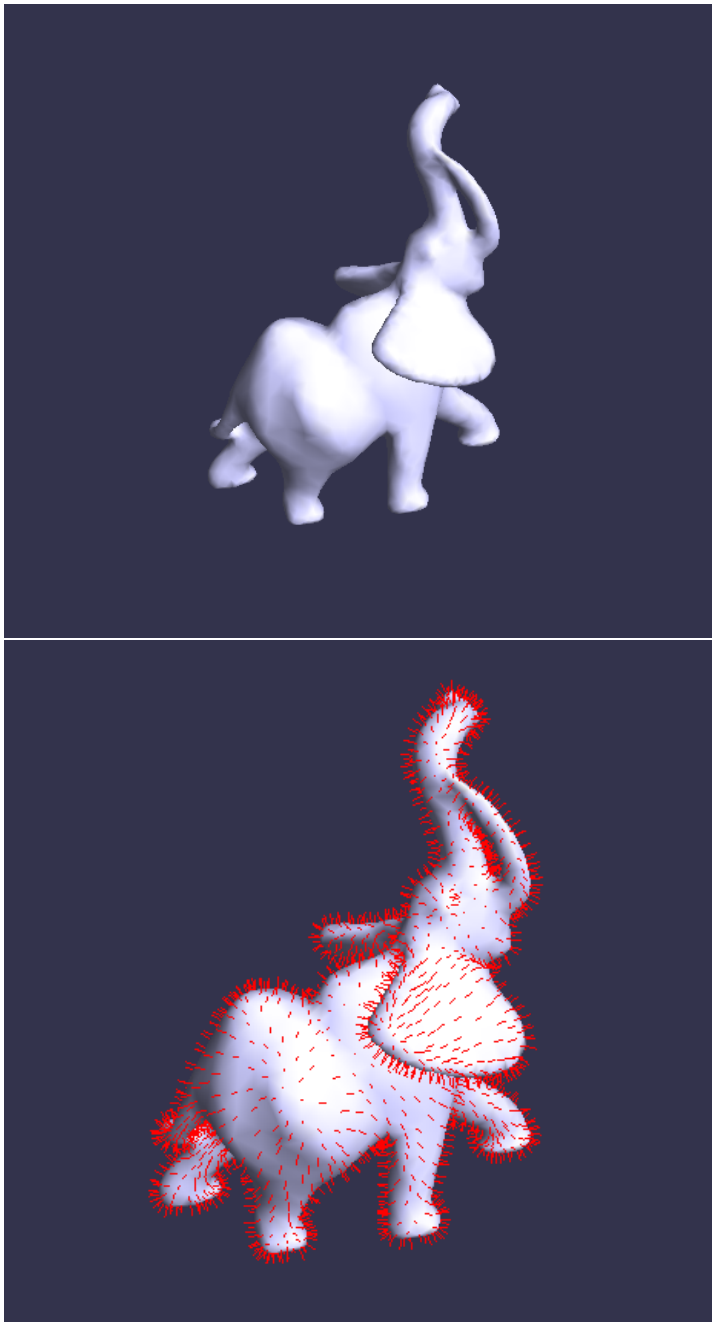
Envoyer le tableau de normales au GPU à l'aide de ***glNormalPointer(GL_FLOAT, 3 * sizeof(float), (GLvoid*)(& normalArray [0]));***

On va donc ajouter ce code au même endroit que là où on l'a fait précédemment avec les vertex.

```
glVertexPointer(3, GL_FLOAT, 3*sizeof(float), (GLvoid*)(&mesh.positionArray[0]));
glNormalPointer(GL_FLOAT, 3*sizeof(float), (GLvoid*)(&mesh.normalArray[0]));
```

Ex2-4 : envoi du tableau de normales au GPU

A la fin de cet exercice, nous avons obtenu le résultat suivant :



Ex2 : rendu de l'objet après l'exercice 2

On voit maintenant que nous avons les ombres sur notre objet et lorsque l'on appuie sur "n" alors les normales sont affichées.

3 Exercice 3

Question 1

Lisez la documentation de ***glColorPointer*** *sur opengl.org*

La fonction glColorPointer est une fonction d'OpenGL qui va prendre en entrée : le nombre d'éléments par coordonnées (nous on en a 3 avec x, y, z), le type de chaque couleur, une taille et le pointeur du premier élément de la liste des couleurs.
C'est cette fonction qui va donc nous permettre d'appliquer des couleurs sur l'objet.

Question 2

Munissez la class Mesh d'un ***std : :vector<float>*** *contenant toutes couleurs des sommets sous la forme rgbrgbrgbrgb... (valeurs entre 0 et 1). Ce vecteur doit être rempli une fois pour toutes, par exemple dans une fonction* ***Mesh : :buildColorArray*** *(que l'on appelle après le chargement du maillage*

Nous allons donc encore ajouter un vecteur dans le mesh, comme nous l'avons déjà fait pour les vertices et les normales, mais là nous allons récupérer les normes de notre objet afin d'avoir un résultat assez joli. Nous allons normaliser ces normes en prenant comme valeur dans notre vecteur, la racine carrée de la norme au carré.

```
void buildColorArray() {
    colorArray.clear();
    for(unsigned int i = 0 ; i < normals.size() ; i++) {
        colorArray.push_back(sqrt(normals[i][0]*normals[i][0]));
        colorArray.push_back(sqrt(normals[i][1]*normals[i][1]));
        colorArray.push_back(sqrt(normals[i][2]*normals[i][2]));
    }
}
```

Ex3-2 : création du vecteur contenant les couleurs de l'objet et remplissage de ce vecteur avec les normales normalisées

Question 3

Activer les color arrays ***(glEnableClientState(GL_ COLOR_ ARRAY) ;)***

Nous allons donc ajouter les color arrays comme nous l'avons fait précédemment avec les normales et vertices.

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
```

Ex3-3 : activation des color arrays

Question 4

Activer l'utilisation des matériaux ***(glEnableClientState(GL_ COLOR_ MATERIAL) ;)***

Nous avons activé l'utilisation des matériaux afin de permettre aux matériaux de l'objet de prendre la bonne couleur.

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_COLOR_MATERIAL);
```

Ex3-4 : activation des color materials

Question 5

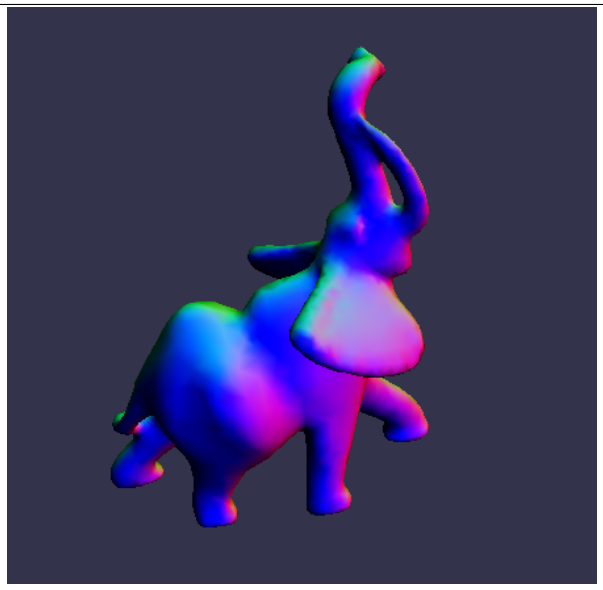
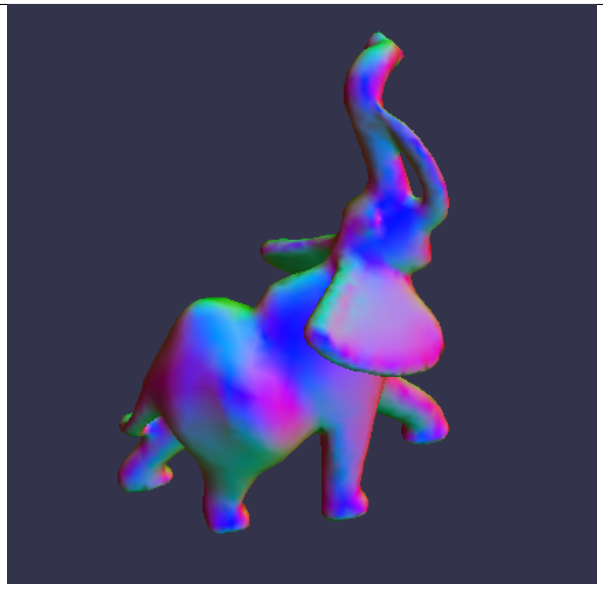
Envoyer le tableau de couleurs au GPU à l'aide de ***glColorPointer(3, GL_FLOAT, 3 * sizeof(float), (GLvoid*)(& colorArray[0])) ;***

On va donc ajouter cette ligne auprès des deux autres.

```
glVertexPointer(3, GL_FLOAT, 3*sizeof(float), (GLvoid*)(&mesh.positionArray[0]));
glNormalPointer(GL_FLOAT, 3*sizeof(float), (GLvoid*)(&mesh.normalArray[0]));
glColorPointer(3, GL_FLOAT, 3*sizeof(float), (GLvoid*)(&mesh.colorArray[0]));
```

Ex3-5 : envoi du tableau de couleurs au GPU

Après avoir réalisé tout ce code, le résultat que l'on va obtenir est le suivant (attention il faut bien penser à normaliser les normales que l'on va mettre dans la liste des couleurs) :

Objet final avec une couleur non normalisée	Objet final avec une couleur normalisée
	

On pourrait aussi choisir la couleur de notre objet en entrant directement des valeurs dans notre vecteur de couleurs.

4 Conclusion

Pour conclure, nous pouvons dire que nous avons plutôt réussi ce tp. En effet, l'objectif de départ était :

- d'afficher les vertices et les triangles : on a bien un éléphant comme c'était attendu
- d'ajouter les normales : on a bien des ombres sur notre objet et on peut les afficher en appuyant sur "n"
- d'afficher des couleurs : on voit que notre éléphant n'est pas tout blanc

On a découvert, avec ce tp, plusieurs fonction OpenGL qui nous ont permis d'obtenir notre résultat.

Je pense que c'est un tp très important car il pose la base de la modélisation 3d d'un objet avec OpenGL.