TP N°8
Dessin 3D (charger un modèle OBJ)

Les buffers d'index :

Les IBO permettent de réutiliser les sommets communs entre plusieurs triangles sans les dupliquer. Par exemple, pour la base de la pyramide, on obtient :

```
vec3(1.0f, 0.0f, 1.0f), vec3(0.0f, 0.0f, 1.0f)   –>  index 0
vec3(-1.0f, 0.0f, 1.0f), vec3(0.0f, 0.0f, 1.0f)  –>  index 1
vec3(-1.0f, 0.0f, -1.0f), vec3(0.0f, 0.0f, 1.0f)  –>  index 2
vec3(-1.0f, 0.0f, -1.0f), vec3(0.0f, 0.0f, 1.0f)  –>  index 2
vec3(1.0f, 0.0f, -1.0f), vec3(0.0f, 0.0f, 1.0f)  –>  index 3
vec3(1.0f, 0.0f, 1.0f), vec3(0.0f, 0.0f, 1.0f)  –>  index 0
```

On crée un buffer qui contient des entiers, trois pour chaque triangle, et qui font référence aux attributs des sommets. On remplace ainsi, la duplication des attributs d'un sommet (position, normale, couleur, …) par la duplication de l'index qui est un entier. Ceci réduit l'espace mémoire utilisé.

- Déclarer et remplir le tableau d'index :

```
unsigned int vertex_index []={ … }
```

- Redéfinir le tableau vertices[] sans répéter les sommets communs.

- Créer le IBO, le lier au VAO avec la cible GL_ELEMENT_ARRAY_BUFFER :

```
GLuint IBO;
glGenBuffers(1, &IBO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 18*sizeof(unsigned int), vertex_index, GL_STATIC_DRAW);
```

- Utiliser glDrawElements pour dessiner la pyramide au lieu de glDrawArrays.

```
glDrawElements(GL_TRIANGLES, 18, GL_UNSIGNED_INT, (void*)0);
```

Charger un modèle 3D :

Dans cette partie nous allons charger un modèle simple de format « obj » créé avec blender.
Le fichier obj contient : les coordonnées des sommets, les coordonnées de texture et les normales.
Nous allons utiliser le type std::vector pour mettre les données au lieu du tableau qu'on utilise d'habitude. Std::vector est aussi un tableau mais avec une taille modifiable.

```
std::vector<vec3>  position;
std::vector<vec2>  texture;
std::vector<vec3>  normals;
```

Nous allons, lire le fichier obj ligne par ligne pour récupérer les données indexées, on les stocke dans des vectors temporaires puis arrivé aux lignes qui contiennent les facettes (elles commencent avec « f ») on envoie les données dans les vectors définitifs en suivant les indexes (le code est fourni).
Dans le main, on appelle la fonction loadOBJ qui prend en paramètres le nom du fichier obj et les trois vectors :
loadOBJ("torus.obj", position, texture, normals);

Il faudra ensuite envoyer les données aux buffers et spécifier les attributs. Par exemple, pour les positions :

```
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, position.size() * sizeof(vec3), &position[0], GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void*)0 );
```

Dans ce TP on va utiliser les normales comme couleur :

```
glGenBuffers(1, &VBOColor);
glBindBuffer(GL_ARRAY_BUFFER, VBOColor);
glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(vec3), &normals[0], GL_STATIC_DRAW);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,3 * sizeof(GLfloat), (void*)0 );
```

On ne va pas utiliser les coordonnées de texture pour le moment. Il reste plus qu'à dessiner l'objet avec *glDrawArrays*.