Introduction à la programmation 3D

Travaux pratiques

Au début de ce TP/TD, vous recevrez une archive zip contenant une base de code. Ce code permet d'afficher un *maillage triangulaire* à l'aide d'openGL.

1. Nous commencerons par l'analyser ensemble pour vous familiariser avec.
2. Vous devez faire évoluer ce code au fur et à mesure du TP, pour répondre aux questions.
3. Vous devez déposez votre code, fichier image de résultat et une phrase sur l'état de votre TP sur le moodle.
4. À la fin de ce TP, vous serez familiés avec :
   — La (une) structure basique de maillage
   — les normales
   — les couleurs
   — les vertex arrays, normal arrays, color arrays (rendu à l'aide de glDrawElements)

1 Base de code

Téléchargez l'archive.
Nous allons le compiler et l'analyser ensemble dans un premier temps.

2 Rendu de maillages

Le fichier tp.cpp contient une méthode `drawTriangleMesh()`, qui contient du code openGL basique permettant d'afficher un maillage triangulaire. Ce code est obsolète (non compatible avec les versions récentes d'openGL).

```
    void drawTriangleMesh( Mesh const & i_mesh ) {
        // This code is deprecated.
        glBegin(GL_TRIANGLES);
        for(unsigned int tIt = 0 ; tIt < i_mesh.triangles.size(); ++tIt) {
            Vec3 p0 = i_mesh.vertices[i_mesh.triangles[tIt][0]];
            Vec3 n0 = i_mesh.normals[i_mesh.triangles[tIt][0]];

            Vec3 p1 = i_mesh.vertices[i_mesh.triangles[tIt][1]];
            Vec3 n1 = i_mesh.normals[i_mesh.triangles[tIt][1]];

            Vec3 p2 = i_mesh.vertices[i_mesh.triangles[tIt][2]];
            Vec3 n2 = i_mesh.normals[i_mesh.triangles[tIt][2]];

            glNormal3f( n0[0] , n0[1] , n0[2] );
            glVertex3f( p0[0] , p0[1] , p0[2] );
            glNormal3f( n1[0] , n1[1] , n1[2] );
            glVertex3f( p1[0] , p1[1] , p1[2] );
            glNormal3f( n2[0] , n2[1] , n2[2] );
            SglVertex3f( p2[0] , p2[1] , p2[2] );
    }
    glEnd();
    }
```

2.1 Exercice

Remplacez ce code et utilisez à la place des vertex arrays.
Pour cela, commencez par les activer :

```
    glEnableClientState (GL_VERTEX_ARRAY);
```

Si les coordonnées des sommets 3D sont écrites dans un tableau `std::vector<float> positionArray` sous la forme `xyzxyzxyzxyz` alors vous pouvez envoyer ce tableau au GPU à l'aide de :

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

Si les indices des coins des triangles à afficher sont écrits dans un tableau `std::vector<unsigned int> triangleArray` sous la forme $c_0c_1c_2c_0c_1c_2c_0c_1c_2\ldots$, alors vous pouvez demander au GPU d'afficher les triangles correspondants à l'aide de :

```
glDrawElements(GL_TRIANGLES, triangleArray.size() , GL_UNSIGNED_INT, (GLvoid*)(&triangleArray[0]));
```

1. Lisez la documentation de `glVertexPointer` sur opengl.org
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.
3. Lisez la documentation de `glDrawElements` sur opengl.org
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.

Vous devriez obtenir ce résultat (figure 1) :

FIGURE 1 – Affichage obtenu à l'aide des positions uniquement (exercice 2.1).

2.2 Exercice

Ajouter l'affichage des normales :

1. Lisez la documentation de `glNormalPointer` sur opengl.org
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.
3. activer les normal arrays (`glEnableClientState (GL_NORMAL_ARRAY);`)
4. envoyer le tableau de normales au GPU à l'aide de
   `glNormalPointer (GL_FLOAT, 3*sizeof (float), (GLvoid*)(&normalArray[0]));`

Vous devriez obtenir ce résultat (figure 2) :

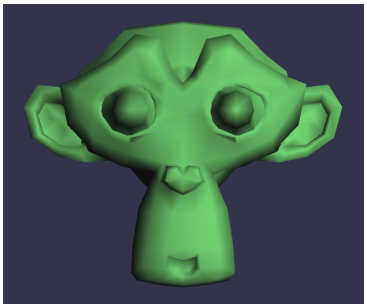

FIGURE 2 – Affichage obtenu à l'aide des positions et des normales (exercice 2.2).

2.3 Exercice

Ajouter l'affichage des couleurs :

1. Lisez la documentation de `glColorPointer` sur opengl.org
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.
3. activer les color arrays (`glEnableClientState (GL_COLOR_ARRAY);`)
4. activer l'utilisation des matériaux (`glEnable(GL_COLOR_MATERIAL);`)
5. envoyer le tableau de normales au GPU à l'aide de
```
glColorPointer (3, GL_FLOAT, 3*sizeof (float), (GLvoid*)(&colorArray[0]));
```

Note Le premier paramètre est le nombre de canaux de couleurs que doit trouver le GPU dans le color array. Si Nous avions mis 4 float par sommet (rgba à la place de rgb), nous aurions du spécifier 4 à la place de 3.
Le deuxième paramètre indique le type des éléments que le GPU doit trouver dans le tableau (en l'occurence, des float).
Le troisième paramètre indique la taille mémoire attendue par couleur associé à un sommet donné.
Le quatrième paramètre indique l'adresse du début du tableau à transmettre au GPU.