HAI719I – Programmation 3D TP Textures

Fabien Caballero

December 23, 2022

Contents

1 Question 1

1.1 fonction TriangleVArray

Dans le code ci-dessous on crée nos buffers (glGenBuffers) avec le nombre de buffers que l'on créé et le buffer utilisé ensuite on dit que l'on vas travailler sur ce buffer (glBindBuffer) puis on précise au buffer sur lequel on travaille quelles données il vas contenir.

```
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * sizeof(g_vertex_buffer_data),
    g_vertex_buffer_data, GL_STATIC_DRAW);
```

De même que précédemment pour le buffer des couleurs

```
glGenBuffers(1, &colorbuffer);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * sizeof(g_color_buffer_data),
    g_color_buffer_data, GL_STATIC_DRAW);
```

1.2 fonction clearBuffers

Ces 2 lignes permettent de vider et supprimer les buffers une fois qu'ils ne sont plus utilisés.

```
glDeleteBuffers(1,&vertexbuffer);
glDeleteBuffers(1,&colorbuffer);
```

1.3 fonction draw

Dans le code ci-dessous pour chaque buffer on précise qu'on travaille sur tel ou tel buffer,

```
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
```

Puis ensuite on précise comment est ce que les données vont être envoyées au shader, typiquement le shader vas travailler point par point et dans notre vertexbuffer on a une suite de float il faut donc lui dire que l'on prend les float 3 par 3 (correspont à x y et z d'un point) avec un décalage de 0 (décalage lorsque on utilise un buffer pour plusieurs données Le premier paramètre précise le layout de ce buffer (utile pour le shader).

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3*sizeof(GLfloat), (void *)0);
```

puis on lui dit que ce buffer sera utiliser dans le shader.

```
glEnableVertexAttribArray(0);
```

Même chose

```
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3*sizeof(GLfloat), (void *)0);
glEnableVertexAttribArray(0);
```

On dessine le triangle

```
glDrawArrays(GL_TRIANGLES,0,3);
```

On désactive les buffers

```
glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
```

2 Question 2

Ici on déclare, dans le vertexshader, 2 variables de type uniform (static pour tout les objets) qui serviront à gérer l'échelle et la translation des objets

```
uniform float scale;
uniform vec3 translation;
```

Afin que les changements soient appliqués on doit appliquer ces changements à la position de l'objet

```
gl_Position =  vec4(vertexPosition_modelspace*scale+translation,1);
```

Avec la fonction key on peut changer dynamiquement ces attribut sans oublier de faire glUniform à chaque fois que la valeur est changée. (nom donné lors du glUniform est le même que celui dans le vertexshader)

3 Question 3

Dans le vertexshader on récupère notre color (depuis le buffer fait précédemment)

```
layout(location = 1) in vec3 color;
```

on créé notre sortie qui sera transmise au fragmentshader

```
out vec3 o_color;
```

puis dans le main du même fichier on assigne les valeurs de notre entrée à notre sortie

```
o_color=color;
```

Dans le fragmentshader on créé une variable d'entrée qui récupèrera la sortie du vertexshader

```
in vec3 o_color;
```

et on utilise cette variable pour gérer notre couleur

```
FragColor = vec4(o_color[0],o_color[1],o_color[2], 1.);
```

4 Question 4

Pour cette question il s'agit de faire la même chose que précédemment sauf que cette fois-ci on vas le faire dans la structure Mesh et on vas avoir un buffer en plus contenant tout les sommets des triangles de notre objet pour le simple triangle juste 3 mais pour l'avion beaucoup plus. Et lors de l'affichage il s'agit cette fois-ci de dessiner les éléments à partir de notre buffer d'indices (liste indexée de sommets).

(Afin de différencier et d'avoir quelque chose de différent du triangle précédent j'ai changé les couleurs en synthèse soustractive)

4.1 fonction initbuffer de la structure Mesh

```
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);

glBufferData(GL_ARRAY_BUFFER, sizeof(Vec3) * vertices.size(), vertices.data(),
    GL_STATIC_DRAW);

glGenBuffers(1, &colorbuffer);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);

glBufferData(GL_ARRAY_BUFFER, sizeof(Vec3) *normals.size(), normals.data(), GL_STATIC_DRAW);

std::vector<unsigned int> indicesSommets;
for (size_t i = 0; i < triangles.size(); i++)
```

```
13    {
14
15      indicesSommets.push_back(triangles[i][0]);
16      indicesSommets.push_back(triangles[i][1]);
17      indicesSommets.push_back(triangles[i][2]);
18    }
19  glGenBuffers(1, &elementbuffer);
20  glBindBuffer(GL_ARRAY_BUFFER, elementbuffer);
21  glBufferData(GL_ARRAY_BUFFER, sizeof(unsigned int)*3*triangles.size(), indicesSommets.data()
        , GL_STATIC_DRAW);
```

4.2 fonction clearBuffers de la structure Mesh

```
1  glDeleteBuffers(1, &vertexbuffer);
2  glDeleteBuffers(1, &colorbuffer);
3  glDeleteBuffers(1, &elementbuffer);
```

4.3 fonction draw de la structure Mesh

```
1  glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
2  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void *)0);
3  glEnableVertexAttribArray(0);
4
5  glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
6  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (void *)0);
7  glEnableVertexAttribArray(1);
8
9  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, elementbuffer);
10 glDrawElements(GL_TRIANGLES, 3 * triangles.size(), GL_UNSIGNED_INT, (void *)0);
11
12 glDisableVertexAttribArray(0);
13 glDisableVertexAttribArray(1);
```