Programmation 3D

Compte-rendu TP3

Transformations

Louis Jean
Master 1 IMAGINE
Université de Montpellier

20 octobre 2023

Table des matières

1 Introduction

Ce TP s'est concentré sur l'exploration approfondie des transformations 3D en utilisant des représentations matricielles, essentielles pour la manipulation d'objets dans un espace tridimensionnel. À travers diverses tâches, nous avons manipulé des transformations géométriques et finalement créé un mini système solaire animé.
Veuillez noter qu'ayant terminé le TP sur ma machine à domicile, j'ai du supprimer le dossier *third_party* et installer *glm-0.9.7.1* sans quoi il m'était impossible de compiler. Cependant, mon code fonctionne sur les machines de la faculté, en rajoutant le dossier *third_party* dans l'arborescence et en commentant quelques directives de préprocessing.

2 Exercice 1

Dans cet exercice, nous avons joué avec des maillages de chaises et de Suzanne et leur avons appliqué diverses transformations.
Pour la première question, il suffisait simplement de rajouter une matrice de transformation dans le code C++ et de la passer au vertex shader.

```
glm::mat4 identityMatrix = glm::mat4(s: 1.0f);
GLuint transformationMatrixID = glGetUniformLocation(programID,"transformationMatrix");
glUniformMatrix4fv(transformationMatrixID,1,GL_FALSE,(const GLfloat *)&identityMatrix[0]);
```

Figure 1: Ajout d'une matrice de transformation dans le code C++

```
uniform mat4 TransformationMatrix;
```

Figure 2: Ajout d'une matrice de transformation dans le vertex shader

```
gl_Position = TransformationMatrix * vec4(vertices_position_modelspace,1);
```

Figure 3: Application de cette matrice de transformation

Il était ensuite demandé de rajouter des contrôles au clavier pour pouvoir déplacer le maillage de chaise affiché à l'écran.
J'ai pu implémenter ceci en complétant la fonction key. Voici quelques exemples (je n'ai pas tout mis dans le compte-rendu pour que cela reste digeste).

```
case '+':
    scaleTransformation += glm::vec3(scalar: 0.1f);
    TransformationMatrix = glm::mat4(s: 1.0f);
    TransformationMatrix = glm::scale(m: TransformationMatrix, v: scaleTransformation);
    TransformationMatrix = glm::translate(m: TransformationMatrix, v: translationTransformation);
    break;
```

Figure 4: + pour le zoom sur la maillage (mise à niveau du maillage en réalité)

```
case 'z':
    translationTransformation += glm::vec3(x: 0.0f,y: 0.1f,z: 0.0f);
    TransformationMatrix = glm::mat4(s: 1.0f);
    TransformationMatrix = glm::translate(m: TransformationMatrix, v: translationTransformation);
    TransformationMatrix = glm::scale(m: TransformationMatrix, v: scaleTransformation);
    break;
```

Figure 5: z pour déplacer le maillage vers le haut

```
case 'q':
    translationTransformation -= glm::vec3(x: 0.1f,y: 0.0f,z: 0.0f);
    TransformationMatrix = glm::mat4(s: 1.0f);
    TransformationMatrix = glm::translate(m: TransformationMatrix, v: translationTransformation);
    TransformationMatrix = glm::scale(m: TransformationMatrix, v: scaleTransformation);
    break;
```

Figure 6: q pour déplacer le maillage vers la gauche

Par la suite, une petite série de transformations était à réaliser. En premier lieu, il fallait diviser la taille du maillage de la chaise par 2 et la placer en bas à gauche de la fenêtre, puis ajouter une nouvelle chaise, placée de manière miroir à la première. Après ceci, il fallait ajouter une troisième chaise, et lui inscrire une rotation controlâble depuis le clavier. Enfin, il fallait changer la rotation de cette chaise, afin qu'elle tourne autour de son centre de gravité.

```
TransformationMatrix = glm::translate(glm::mat4(1.0f),glm::vec3(-0.5f,-1.0f,0)) * glm::scale(glm::mat4(1.0f),glm::vec3(0.5f,0.5f,0.5f));
TransformationMatrix2 = glm::translate(glm::mat4(1.0f),glm::vec3(0.5f,-1.0f,0)) * glm::scale(glm::mat4(1.0f),glm::vec3(-0.5f,0.5f,0.5f));
```

Figure 7: Code pour effectuer les transformations décrites ci-dessus

Figure 8: Première étape

Figure 9: Deuxième étape

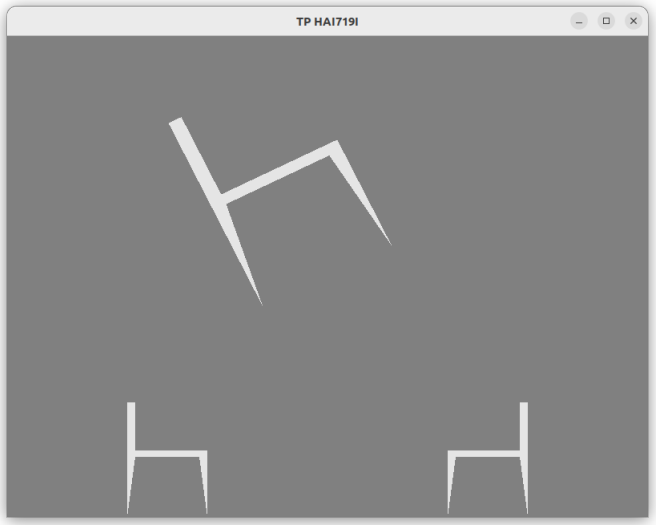

Figure 10: Troisième étape

En m'appuyant sur le cours, j'ai réalisé cet enchaînement de transformations pour permettre à la troisième chaise de faire une rotation selon son centre de gravité lors d'un appui sur la touche 3.

```
case '3': {
    radiansChaise3 += 10.0f;
    glm::mat4 trans1 = glm::translate(glm::mat4(1.0f),glm::vec3(0.0f,-0.5f,0.0f));
    glm::mat4 rot = glm::rotate(glm::mat4(1.0f),glm::radians(radiansChaise3),glm::vec3(0.0f,0.0f,1.0f));
    glm::mat4 trans2 = glm::translate(glm::mat4(1.0f),glm::vec3(0.0f,0.5f,0.0f));
    TransformationMatrix3 = (trans2 * rot) * trans1;
    glUniformMatrix4fv(transformationMatrixID,1,GL_FALSE,(const GLfloat *)&TransformationMatrix3[0]);
    break;
}
```

Figure 11: Code pour la dernière étape

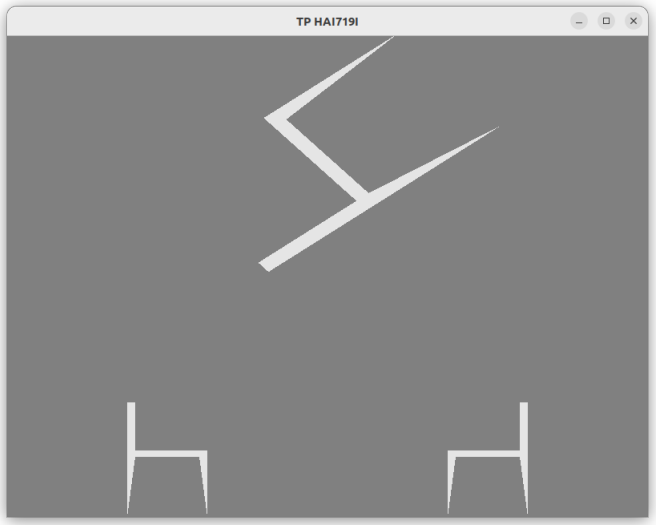

Figure 12: Dernière étape, après plusieurs appuis sur la touche 3

Enfin, il fallait changer le modèle de la chaise par celui de Suzanne, pour ensuite y appliquer une rotation 3D contrôlée au clavier.

```
case 'r':
    TransformationMatrix = glm::rotate(m: glm::mat4(s: 1.0f),angle: glm::radians(degrees: radiansSuzanne),v: glm::vec3(x: 1.0f,y: 1.0f,z: 1.0f));
    radiansSuzanne+=10.0f;
    break;
```

Figure 13: Code pour implémenter la rotation de Suzanne sur 3 axes en appuyant sur la touche r

Finalement, il était demandé d'appliquer une rotation afin que l'axe vertical de Suzanne (vecteur (0,1,0)) soit aligné avec le vecteur (1,1,1) du repère monde.
Voici la manière dont j'ai implémenté ceci et le résultat.

```
case 'l': { // Pour aligner l'objet sur le vecteur (1,1,1) (sans modèle MVP)
    glm::mat4 ViewMatrixObjet = glm::lookAt(eye: glm::vec3(scalar: 0.0f),center: glm::vec3(x: 1.0f,y: 1.0f,z: 1.0f),up: glm::vec3(x: 0.0f,y: 0.1f,z: 0.0f));
    TransformationMatrix = ViewMatrixObjet * TransformationMatrix;
}
```

Figure 14: Code pour implémenter le repositionnement selon le vecteur (1,1,1) du repère monde

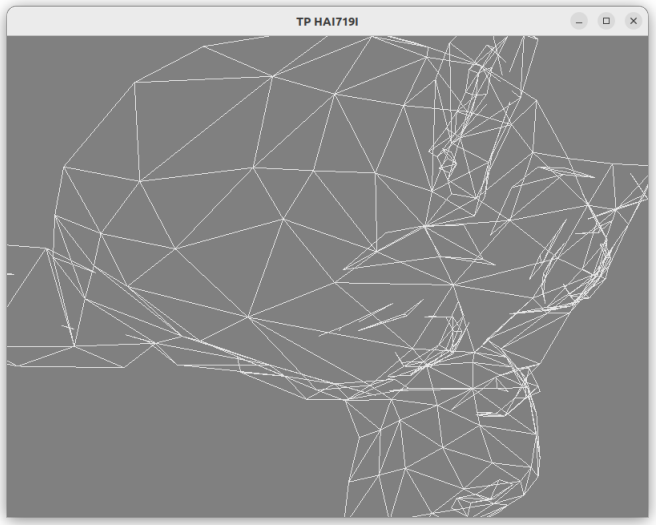

Figure 15: Résultat de ce repositionnement

3 Exercice 2

Dans cet exercice, le but était d'implémenter le modèle MVP (Model View Projection).
Voici comment je m'y suis pris.

```
glm::mat4 ModelMatrix = glm::mat4(1.0f);
```

Figure 16: Création de la matrice Model

```
glm::mat4 ViewMatrix = glm::lookAt(camera_position,camera_target,camera_up);
```

Figure 17: Création de la matrice View

```
glm::mat4 ProjectionMatrix = glm::perspective(glm::radians(45.0f),4.0f / 3.0f, 0.1f, 100.0f);
```

Figure 18: Création de la matrice Projection

```
uniform mat4 ModelMatrix;
uniform mat4 ViewMatrix;
uniform mat4 ProjectionMatrix;
```

Figure 19: Déclaration des variables uniformes nécessaires dans le vertex shader

```
GLuint ModelMatrixLocation = glGetUniformLocation(programID,"ModelMatrix");
GLuint ViewMatrixLocation = glGetUniformLocation(programID,"ViewMatrix");
GLuint ProjectionMatrixLocation = glGetUniformLocation(programID,"ProjectionMatrix");
```

Figure 20: Récupération des locations de ces variables uniformes

```
glUniformMatrix4fv(ModelMatrixLocation,1,GL_FALSE,(const GLfloat *)&ModelMatrix[0]);
glUniformMatrix4fv(ViewMatrixLocation,1,GL_FALSE,(const GLfloat *)&ViewMatrix[0]);
glUniformMatrix4fv(ProjectionMatrixLocation,1,GL_FALSE,(const GLfloat *)&ProjectionMatrix[0]);
```

Figure 21: Passage des matrices remplies au vertex shader

```
gl_Position = ProjectionMatrix * ViewMatrix * ModelMatrix * vec4(vertices_position_modelspace,1);
```

Figure 22: Utilisation du modèle MVP dans le vertex shader

Pour terminer cet exercice, il fallait implémenter un déplacement vertical et horizontal de la caméra. Voici ce que j'ai écrit.

```
case 'u':
    camera_position += glm::vec3(x: 0.0f,y: cameraSpeed,z: 0.0f);
    break;

case 'j':
    camera_position -= glm::vec3(x: 0.0f,y: cameraSpeed,z: 0.0f);
    break;

case 'h':
    camera_position -= glm::vec3(x: cameraSpeed,y: 0.0f,z: 0.0f);
    break;

case 'k':
    camera_position += glm::vec3(x: cameraSpeed,y: 0.0f,z: 0.0f);
    break;
```

Figure 23: Implémentation des déplacements de la caméra

4 Exercice 3

Cet exercice proposait la mise en place d'un mini système solaire animé, en utilisant le modèle MVP.
Voici la solution que j'ai élaborée.
Le code suivant est intégré dans la fonction draw().

```
glm::mat4 ModelMatrixSun = glm::mat4(s: 1.0f);
float sunSunSpeed = 0.1f;
angleSunSun += sunSunSpeed * deltaTime;
ModelMatrixSun = glm::scale(m: ModelMatrixSun,v: glm::vec3(x: 0.5f,y: 0.5f,z: 0.5f));
ModelMatrixSun = glm::rotate(m: ModelMatrixSun,angle: glm::radians(degrees: angleSunSun),v: glm::vec3(x: 0.0f,y: 1.0f,z: 0.0f));
```

Figure 24: Code pour le Soleil

```
// Terre
glm::mat4 ModelMatrixEarth = glm::mat4(s: 1.0f);
float earthSunSpeed = 5.0f;
float earthEarthSpeed = 50.0f;
angleEarthSun += earthSunSpeed * deltaTime;
angleEarthEarth += earthEarthSpeed * deltaTime;
angleEarthSun = fmod(x: angleEarthSun,y: 360.0f);
angleEarthEarth = fmod(x: angleEarthEarth,y: 360.0f);
glm::vec3 axeInitial = glm::vec3(x: 0.0f,y: 1.0f,z: 0.0f);
glm::mat4 tiltAxe = glm::rotate(m: glm::mat4(s: 1.0f),angle: 23.0f,v: glm::vec3(x: 0.0f,y: 0.0f,z: 1.0f));
glm::vec4 axeTilted4 = tiltAxe * glm::vec4(xyz: axeInitial,w: 0.0f);
glm::vec3 axeTilted = glm::vec3(v: axeTilted4); // Tentative d'appliquer une rotation de 23 degrés à l'axe Y autour de l'axe Z
ModelMatrixEarth = glm::rotate(m: ModelMatrixEarth, angle: glm::radians(degrees: angleEarthSun), v: glm::vec3(x: 0.0, y: 1.0, z: 0.0));
ModelMatrixEarth = glm::translate(m: ModelMatrixEarth, v: glm::vec3(x: 1.0f, y: 0.0f, z: 0.0f)); // Translation pour l'orbite
ModelMatrixEarth = glm::rotate(m: ModelMatrixEarth, angle: glm::radians(degrees: angleEarthEarth), v: axeTilted);
ModelMatrixEarth = glm::scale(m: ModelMatrixEarth,v: glm::vec3(x: 0.15f,y: 0.15f,z: 0.15f));

glUniformMatrix4fv(ModelMatrixLocation,1,GL_FALSE,(const GLfloat *)&ModelMatrixEarth[0]);

glDrawElements(mode: GL_TRIANGLES,count: indices.size(),type: GL_UNSIGNED_SHORT,indices: (void*)0);
```

Figure 25: Code pour la Terre

```
// Lune
float moonOrbitSpeed = 10.0f;
angleMoonEarth += moonOrbitSpeed * deltaTime;
angleMoonEarth = fmod(x: angleMoonEarth,y: 360.0f);
glm::mat4 ModelMatrixMoon = ModelMatrixEarth;
ModelMatrixMoon = glm::translate(m: ModelMatrixMoon,v: glm::vec3(x: 1.0f,y: 1.0f,z: 0.0f));
ModelMatrixMoon = glm::rotate(m: ModelMatrixMoon,angle: glm::radians(degrees: angleMoonEarth),v: glm::vec3(x: 0.0f,y: 1.0f,z: 0.0f));
ModelMatrixMoon = glm::scale(m: ModelMatrixMoon,v: glm::vec3(x: 0.25f,y: 0.25f,z: 0.25f));

glUniformMatrix4fv(ModelMatrixLocation,1,GL_FALSE,(const GLfloat *)&ModelMatrixMoon[0]);

glDrawElements(mode: GL_TRIANGLES,count: indices.size(),type: GL_UNSIGNED_SHORT,indices: (void*)0);
```

Figure 26: Code pour la Lune

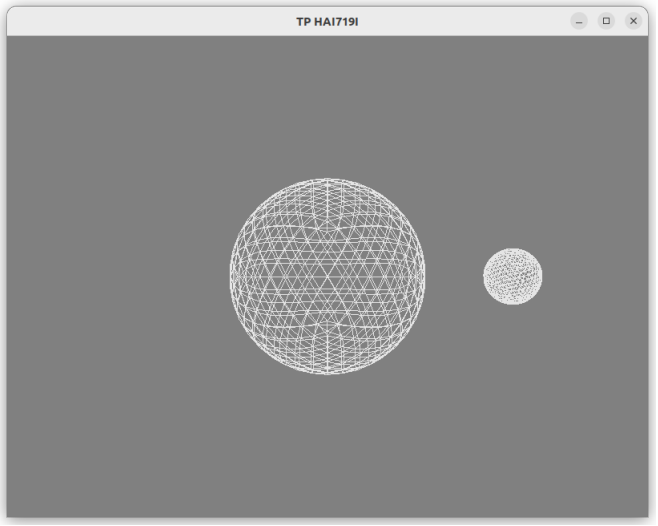

Figure 27: Rendu du Soleil et de la Terre

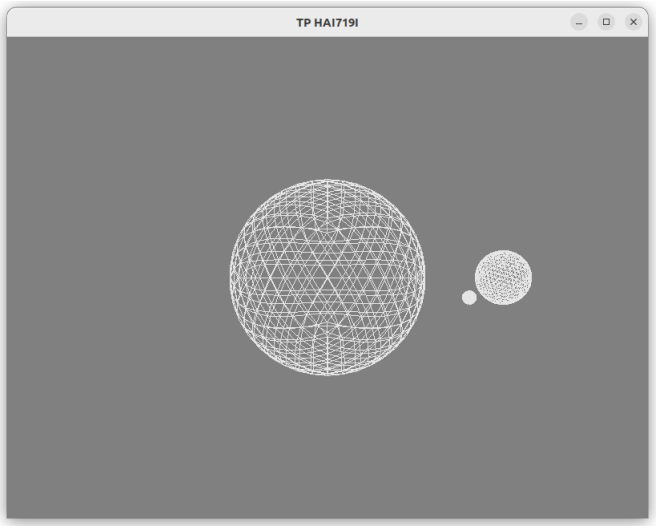

Figure 28: Rendu du Soleil, de la Terre et de la Lune

N.B : La Terre tourne sur elle-même selon un axe vertical ayant subi une inclinaison de 23 degrés. Elle tourne aussi autour du Soleil. La Lune tourne autour de la Terre.
Améliorations possibles : Un bon choix aurait été de créer une classe Planete pour faciliter l'implémentation de ce système solaire. Ma version, codée en dur, n'est vraiment pas optimale. J'aurai aussi pu rajouter des textures et des lumières pour rendre cela plus sympathique.

5 Conclusion

Ce TP a consolidé ma compréhension des transformations 3D. Ces connaissances sont cruciales pour des explorations graphiques futures plus complexes.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.