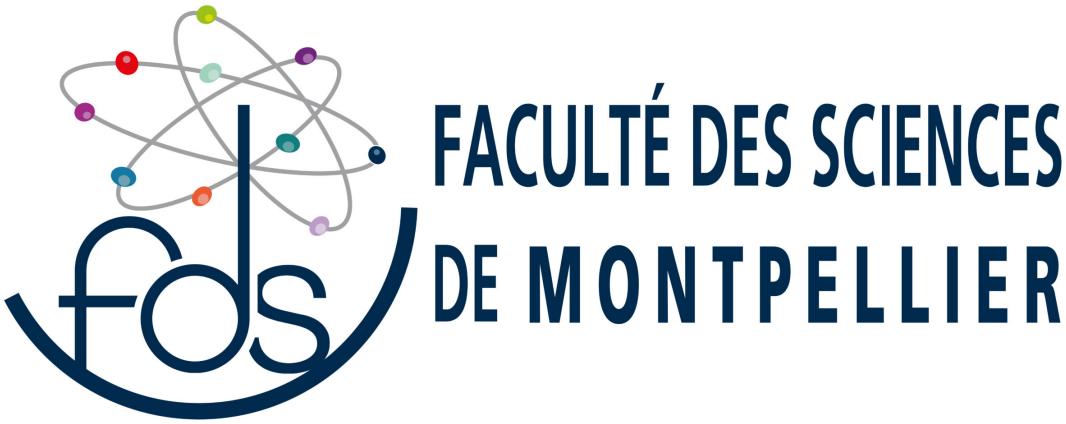
Université de Montpellier

M1 - IMAGINE - Programmation 3D
Compte Rendu TP3 - Transformations

Etudiant :
Guillaume Bataille

Encadrant :
Noura Faraj
Marc HARTLEY

Année 2022-2023



Sommaire

Contexte Aujourd'hui nous avons vu des révisions et un approfondissement des transformations matricielle. Cela permet de faire évoluer des objets dans une scène, de faire bouger des repères, mais aussi de gérer la caméra, les vues et faire un rendu de projection.

Objectif Le but de ce TP est de se familiariser avec les transformations 3D représentées sous forme matricielle. Cela va se traduire par des manipulations de matrices et des tranformations, d'abord sur une chaise et ensuite sur la camera et enfin sur des astres.

Utilisation :

Compiler et lancer : make et ./tp

Changer d'exercice affiché : espace

Translation de la matrice de transmformation : q et d (x) / z et s (y)

Scale de la matrice de transformation : + / n

Rotation de la matrice de transformation autour de l'origine : i et k (x) / o et l (y) / p et m (z)

Aligner la matrice de transformation avec 1,1,1 en Y : v

Zoom de la camera : w avant et x arrière

Deplacer la caméra : t f g h

1. Question 1 - Chaise

1.1 Variable uniforme mat4 et premières matrices de transformation identités

1.1.1 Côté Vertex_shader

On decide de d'ajouter 3 uniform mat4, ce seront nos matrices M V P et nous les utilisons afin de calculer la gl_Posiition.

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertices_position_modelspace;

// NEW
uniform mat4 transform_mat;
uniform mat4 view_mat;
uniform mat4 project_mat;


//TODO create uniform transformations matrices Model View Projection
// Values that stay constant for the whole mesh.

void main(){

        // TODO : Output position of the vertex, in clip space : MVP * position
        // Ordre des multiplication de matrice important !
        gl_Position = project_mat * view_mat * transform_mat * vec4(vertices_position_modelspace,1) ;

}

```

1.1.2 Côté C++

Pour remplir nos uniform mat4 côté shader, nous initialisons 3 glm : :mat4 comme matrices 4x4 identités.

```
glm::mat4 mat_t = glm::mat4(1); // Ma matrice de transformation initialisé a identité
glm::mat4 mat_v = glm::mat4(1); // Ma matrice de vue initialisé a identité
glm::mat4 mat_p = glm::mat4(1); // Ma matrice de projection initialisé a identité
```

Maintenant qu'on à nos mat4, nous envoyons ces derniers aux shaders dans la méthode draw.

```

// Model matrix : an identity matrix (model will be at the origin) then change

GLuint id_t = glGetUniformLocation(programID, "transform_mat");
glUniformMatrix4fv(id_t, 1, false, &mat_t[0][0]);

// View matrix : camera/view transformation lookat() utiliser camera_position camera_target camera_up

GLuint id_v = glGetUniformLocation(programID, "view_mat");
glUniformMatrix4fv(id_v, 1, false, &mat_v[0][0]);

// Projection matrix : 45 Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units

GLuint id_p = glGetUniformLocation(programID, "project_mat");
glUniformMatrix4fv(id_p, 1, false, &mat_p[0][0]);

```

1.2 Manipulation clavier

Comme on envoie la matrice mat_t au shaders, on décide de généraliser ce qui a été fait pour les scale,translation et rotation dans le tp précédent en utilisant une matrice de transformation (mat_t) qu'on va configurer avec les fonctions glm a chaque appui sur une certaine touche.

1.2.1 Scale

```
        // SCALE

    case '+': //Press + key to increase scale
        mat_t = glm::scale(mat_t, glm::vec3(1.05, 1.05, 1.05));
        break;

    case '-': //Press - key to decrease scale
        mat_t = glm::scale(mat_t, glm::vec3(0.95, 0.95, 0.95));
        break;
```

1.2.2 Translation

```
    case 'd': //Press d key to translate on x positive
        mat_t = glm::translate(mat_t, glm::vec3(0.005, 0., 0.));
        break;
```

```
4
5       case 'q': //Press q key to _translate on x negative
6           mat_t = glm::translate(mat_t, glm::vec3(-0.005, 0., 0.));
7           break;
8
9       case 'z': //Press z key to vec_translate on y positive
10          mat_t = glm::translate(mat_t, glm::vec3(0., 0.005, 0.));
11          break;
12
13      case 's': //Press s key to vec_translate on y negative
14          mat_t = glm::translate(mat_t, glm::vec3(0., -0.005, 0.));
15          break;
```

1.2.3 Rotation

```
1       // ROTATE
2       case 'i': //Press i key to rotate on x positive
3           mat_t = glm::rotate(mat_t, 3.14f / 4, glm::vec3(1.0f, 0.0f, 0.0f));
4           break;
5
6       case 'k': //Press k key to rotate on x positive
7           mat_t = glm::rotate(mat_t, -3.14f / 4, glm::vec3(1.0f, 0.0f, 0.0f));
8           break;
9
10      case 'o': //Press o key to rotate on y positive
11          mat_t = glm::rotate(mat_t, 3.14f / 4, glm::vec3(0.0f, 1.0f, 0.0f));
12          break;
13
14      case 'l': //Press l key to rotate on y megative
15          mat_t = glm::rotate(mat_t, -3.14f / 4, glm::vec3(0.0f, 1.0f, 0.0f));
16          break;
17
18      case 'p': //Press p key to rotate on y positive
19          mat_t = glm::rotate(mat_t, 3.14f / 4, glm::vec3(0.0f, 0.0f, 1.0f));
20          break;
21
22      case 'm': //Press m key to rotate on y negative
23          mat_t = glm::rotate(mat_t, -3.14f / 4, glm::vec3(0.0f, 0.0f, 1.0f));
24          break;
```

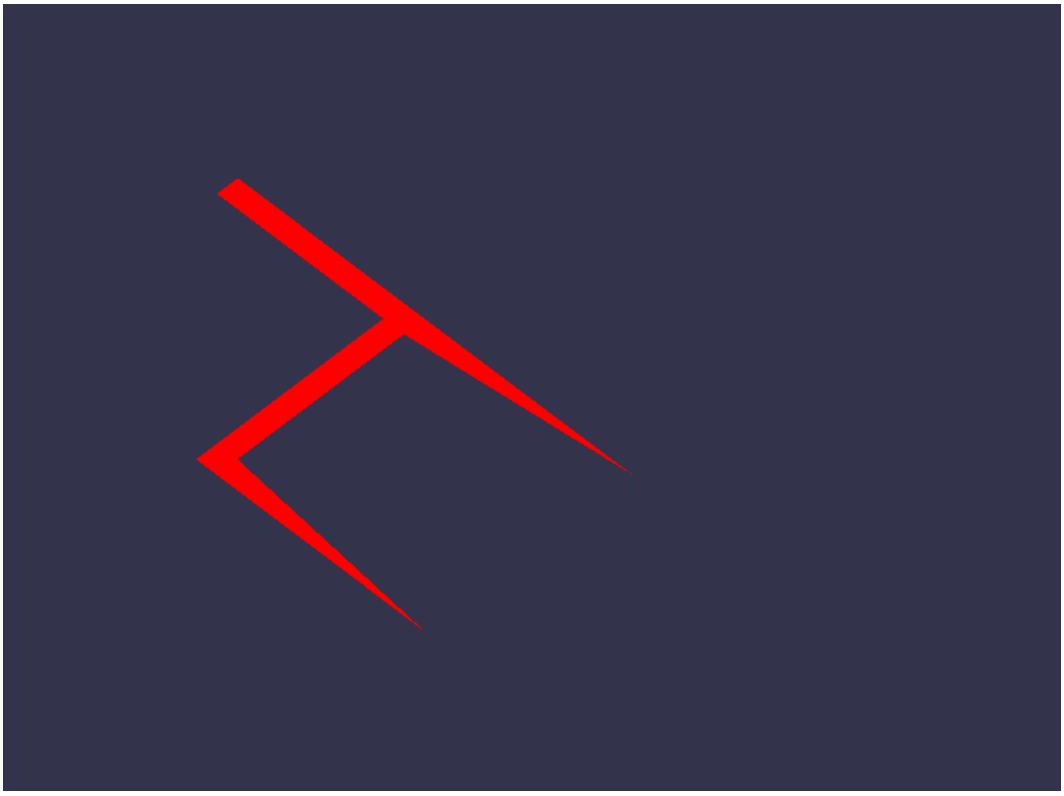


FIGURE 1.1 – Chaise de base après translation vers le bas, scale x 2, rotation de 180° en y et rotation de 45° sur z

1.3 Transformation manuelle de la scène

Pour cette partie, on n'utilise aucun input clavier, tout est fait manuellement depuis draw (sauf le dernier).

1.3.1 Chaise 1 : Au sol a gauche et deux fois plus petite

```
1  glm::mat4 chaise1;
2  chaise1 = glm::scale(mat_t, glm::vec3(0.5, 0.5, 0.5));    // Scale / 2
3  chaise1 = glm::rotate(chaise1, 0.0f, directionVector()); // Rotate de 0
4  chaise1 = glm::translate(chaise1, glm::vec3(-1, -2, 0)); // Translate pour la poser au sol
```

FIGURE 1.2 – Chaise 1

1.3.2 Chaise 2, de même taille en face et fesant face a la premiere

```
glm::mat4 chaise1;
chaise1 = glm::scale(mat_t, glm::vec3(0.5, 0.5, 0.5));    // Scale / 2
chaise1 = glm::rotate(chaise1, 0.0f, directionVector()); // Rotate de 0
chaise1 = glm::translate(chaise1, glm::vec3(-1, -2, 0)); // Translate pour la poser au sol
```

FIGURE 1.3 – Chaise2

1.3.3 Chaise 3 a la position de base avec une rotation basique fesable avec un input clavier

```
glm::mat4 chaise3;
//Rotate via les input defini dans la question précédente
```

On constate que la rotation ici si fait autout de l'axe x y z global (aux centre de la scène).

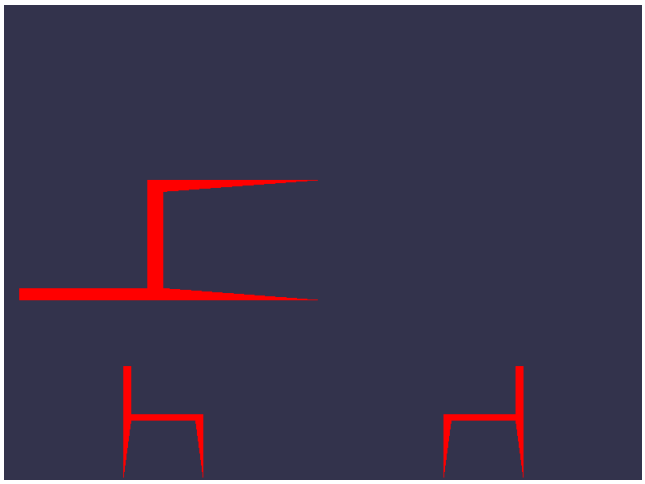


FIGURE 1.4 – Chaise3

1.3.4 Rotation autour du centre de gravité de la chaise (0,0.5,0)

On souhaite donc ici faire une rotation autour du centre de la chaise 3 (0,0.5,0).
Pour faire cette rotation et plus généralement pour tourner autour d'un point p, il faut effectuer dans l'ordre :
Translation1 de - p : cela mettra le centre de l'objet en 0,0,0 Rotation : cela effectuera la rotation voulue Translation2 retour de p : cela remettra l'objet a la position initiale. Pour que cela se passe dans cet ordre il faut definir mat_t = Translation2 * Rotation * Translation1.

```
// On a float rotatechaise3 = 0; defini globalement
// Chaque appui sur b ou n incrémente(ou décrémente) cette variable
glm::mat4 chaise3, t1, r, t2;
// Calcul de l'angle de rotation courant en fonction du nomnbre d'incrément/decrement de rotatechase3
float rotate_angle = ((3.14) / 8) * rotatechaise3;
t1 = glm::translate(mat_t, glm::vec3(0, -0.5, 0));
r = glm::rotate(mat_t, (3.14f / 8) * rotatechaise3, glm::vec3(0.0f, 0.0f, 1.0f));
t2 = glm::translate(mat_t, glm::vec3(0, 0.5, 0));
chaise3 = t2 * r * t1; // Calculé dans cet ordre -> D'abord t1 puis r puis t2
```

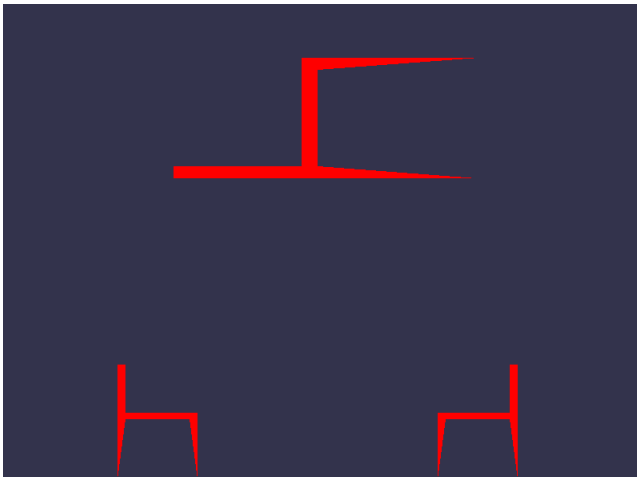


FIGURE 1.5 – Chaise 3 -Même nombre de rotation que la précédent mais on tourne autour du centre de la chaise

2. Suzanne

2.0.1 Rotation quelconque via clavier

FIGURE 2.1 – Suzanne de base

FIGURE 2.2 – Suzanne post rotation autour de x

FIGURE 2.3 – Suzanne post rotation autour de y

FIGURE 2.4 – Suzanne post rotation autour de z

2.0.2 Alignement de l'axe y de suzanne (0,1,0) avec le vecteur (1,1,1) des coordonées monde.

Suzanne semble centrer sur les axes du repère monde (0,0,0).
Je décompose mon problème en 2 étapes : aligner suzanne sur l'axe 1,1,0 puis sur l'axe 1,1,1.
Ces deux étapes sont combinable avec glm :

```
/* OLD
// Calcul des angles necessaires pour aligner l'axe y avec le vecter 1,1,1 du repaire monde
float l_vec = glm::length(glm::vec3(1, 1, 1));
float l_xy = glm::length(glm::vec2(1, 1));
float x_angle = acos(l_xy / l_vec);
float z_angle = acos(1. / l_xy);

// Application to align suzanne:

mat_t = glm::rotate(mat_t, x_angle, glm::vec3(1.0f, 0.0f, 0.0f)); // En X
mat_t = glm::rotate(mat_t, z_angle, glm::vec3(0.0f, 0.0f, 1.0f)); // En Z*/

//NEW -  -45 car sinon On est dans le mauvais sens (antihoraire)
mat_t = rotate(mat_t, -45.f, glm::vec3(1., 0., 1.));

```

3. View et Projection

Auparavant, j'avais mes matrices view et projection en tant que matrices identités, ce qui occulte tout l'aspect 3D du rendu. Essayons dans cet exercice de les prendre en compte !

3.1 Projection

```
//perspective prends 4 arguments : Le champe de vision, le ratio, et les bornes definissant la resolution
    // Projection matrix : 45 Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
    mat_p = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);
    GLuint id_p = glGetUniformLocation(programID, "project_mat");
```

3.2 View

```
//lookAt prends 3 arguments:Le vecteur position de la camera, le verteur vers la cible(Z) et le vecteur up(Y)
    // View matrix : camera/view transformation lookat() utiliser camera_position camera_target camera_up
    mat_v = glm::lookAt(camera_position, camera_target, camera_up);
    GLuint id_v = glGetUniformLocation(programID, "view_mat");
```

3.3 Mouvement de camera

Pour effectuer les mouvements de caméra, on a besoin de l'axe X de la camera, on l'obtient avec le produit vectoriel entre cameratarget et up.

```
        // CAMERA MOVEMENTS
    case 'f':
        camera_position += glm::normalize(glm::cross(camera_target, camera_up)) * cameraSpeed;

        //mat_v = glm::translate(mat_v, glm::vec3(cameraSpeed, 0, 0));
        break;

    case 'h':
        camera_position -= glm::normalize(glm::cross(camera_target, camera_up)) * cameraSpeed;
        break;
    case 't':
        camera_position += cameraSpeed * camera_up;
```

```
13          break;
14
15      case 'g':
16          camera_position -= cameraSpeed * camera_up;
17          break;
18
```

On constate que lorsqu'on fait deplacer la camera, elle pointe toujours vers le 0,0,0. C'est tout l'interet de lookAt qui va conserver vers ou la camera regarde peut, importe les transformation qu'elle va subir.

FIGURE 3.1 – Suzanne avec zoom out x2 / Camera mise en haut à droite (X++ et Y++)

4. Question 3 - Mini Système solaire

On cherche ici a faire tourner des objets entre eux dans un mini système solaire.
Afin d'avoir une rotation constante, j'utilise le tick obtenu par deltatime que j'incrémente dans ma variable "spin".

4.1 Soleil

```
1       // SOLEIL
2
3       glm::mat4 soleil = mat_t;
4       soleil = glm::rotate(soleil, spin / 5.f, glm::vec3(0, 1, 0)); // Rotation avec spin/5
5       soleil = scale(soleil, glm::vec3(0.5, 0.5, 0.5));             // Scale
6
```

4.2 Terre

```
1       // TERRE
2
3       glm::mat4 terre = mat_t;
4       terre = glm::rotate(terre, spin / 5.f, glm::vec3(0, 1, 0));                    // Rotation avec spin/5
5       terre = glm::translate(terre, glm::vec3(-1.5, 0, 0));                          // Translate initial
6       glm::mat4 lune = terre; // A cet endroit on peut initialiser la lune
7       terre = glm::rotate(terre, glm::radians(23.0f), glm::vec3(1.0f, 0.0f, 0.0f)); //INCLINAISON
8       terre = rotate(terre, spin * 2, glm::vec3(0, 1, 0));                           // Rotation sur elle même
9       terre = glm::scale(terre, glm::vec3(0.2, 0.2, 0.2));                           // Scale
10
11
```

4.3 Lune

```
    // LUNE

    //INCLINAISON TERRE/LUNE
    lune = glm::rotate(lune, glm::radians(5.0f), glm::vec3(1.0f, 0.0f, 0.0f));
    lune = glm::rotate(lune, spin, glm::vec3(0, 1, 0));
    lune = glm::translate(lune, glm::vec3(-0.5, 0, 0));                         // TRANSLATE INITIAL
    lune = glm::rotate(lune, glm::radians(6.0f), glm::vec3(0.0f, 1.0f, 0.0f)); //INCLINAISON de la lune
    lune = glm::scale(lune, glm::vec3(0.1, 0.1, 0.1));                          // SCALE

```

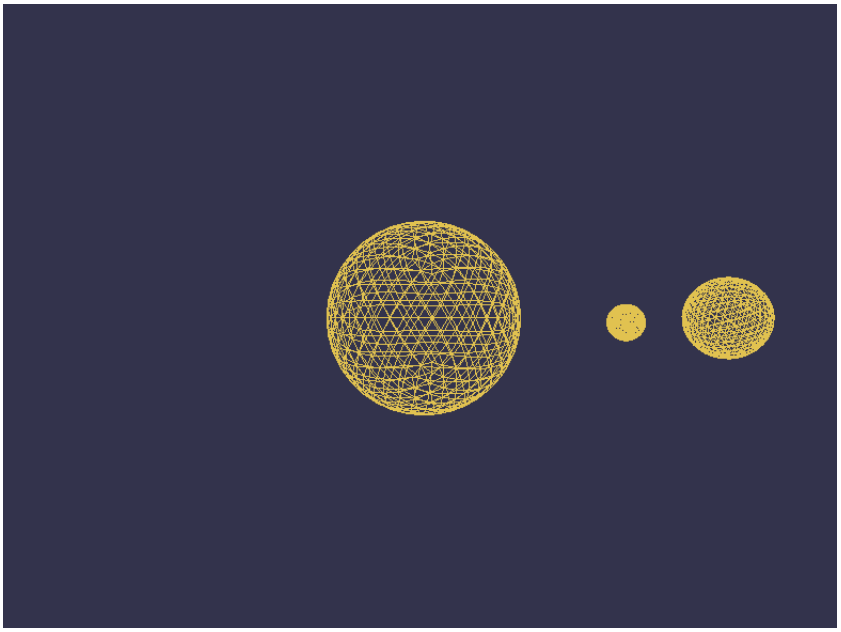


FIGURE 4.1 – Visuel de mon système solaire