Transformations et projections
TD n°2

Dans ce TP, vous allez repartir du projet précédent et vous allez y ajouter les fonctions nécessaires pour gérer le positionnement, l'orientation et la mise en perspective des objets 3D.

Dans ce projet, il vous faudra exclusivement compléter les parties du code source où le commentaire suivant est marqué

```
// TODO => TP02 //
```

Une classe `Object` a été ajoutée qui permet de décrire un objet : l'ensemble des sommets (`vertices`) ainsi que la manière dont ils sont reliés (`faces`). `faces` est de type `Face` (également une nouvelle classe) et contient principalement l'index des sommets enregistrés dans `vertices`. Une face est donc décrite de la manière suivante : le sommet numéro 2 est relié avec les sommets numéro 5, 8 et 9 (2, 5, 8 et 9 sont les indexes des sommets dans `vertices`).

Dans ce TD, on souhaite réaliser l'ensemble des transformations subies par les sommets avant d'être affichés (transformation puis projection). Les sommets et l'ensemble des informations concernant l'objet à dessiner sont contenues dans l'instance `myobject` de la classe `Object`. Commencez par comprendre à quoi servent les fonctions `load_box()` et `load_sphere()`.

Ensuite, vous observerez les modifications de la fonction `display()` et comment l'objet `myobject` est dessiné. Avant de dessiner cet objet, plusieurs opérations de mises à jour sont effectuées. En effet, à chaque appel de la fonction `display()`, il va falloir mettre à jour la transformation et la projection de l'ensemble des sommets de `myobject` : c'est le rôle des méthodes `update_transformation()` et `update_projection()`. Cependant, ces deux fonctions nécessitent des matrices en paramètres d'entrée : ces matrices sont calculées par les fonctions `transformation_matrix()` et `projection_matrix()`.

Exercice 1 *Complétez la fonction de transformation définie dans* ***`main.cpp`*** *de la manière suivante.*

```
mat4 transformation_matrix(double pitch, double yaw, double roll, vec3 \
    translation);
```

où ***`pitch`****,* ***`yaw`*** *et* ***`roll`*** *correspondent respectivement aux angles de rotations x, y et z et* ***`translation`*** *représente la translation dans la transformation. Cette fonction doit retourner une matrice combinant les rotations et les translations.*

Correction de l'exercice 1 – Il faut construire les différentes composantes de cette matrice de transformation que sont les matrices de rotation et de translation. La multiplication s'effectue dans le sens $T \cdot R_x \cdot R_y \cdot R_z$.
Conseil *: Les élèves ne sauront pas si leur code fonctionne avant le dernier exercice du TP. Le risque d'accumuler des erreurs et d'avoir à affronter un déboggage douloureux à la fin du TP est important. Une solution pour limiter ces risques est d'implémenter le minimum nécessaire à chaque étape pour pouvoir visualiser le cube à la fin. Dans l'exercice 1 seul la translation est indispensable. Les rotations peuvent attendre la fin du TP.*

Source 1 – Fonction renvoyant la matrice de transformation

```
mat4 transformation_matrix(double pitch, double yaw, double roll, vec3 \
    translation)
{
  mat4 rot_x_matrix;
  mat4 rot_y_matrix;
  mat4 rot_z_matrix;
  mat4 translation_matrix;
```

```
  pitch = pitch * M_PI / 180; // X axis
  yaw = yaw * M_PI / 180; // Y axis
  roll = roll * M_PI / 180; // Z axis

  ///////Rotation matrix ////////
  rot_x_matrix[1][1] =  cos(pitch);
  rot_x_matrix[1][2] =  sin(pitch);
  rot_x_matrix[2][1] = -sin(pitch);
  rot_x_matrix[2][2] =  cos(pitch);

  rot_y_matrix[0][0] =  cos(yaw);
  rot_y_matrix[0][2] = -sin(yaw);
  rot_y_matrix[2][0] =  sin(yaw);
  rot_y_matrix[2][2] =  cos(yaw);

  rot_z_matrix[0][0] =  cos(roll);
  rot_z_matrix[0][1] =  sin(roll);
  rot_z_matrix[1][0] = -sin(roll);
  rot_z_matrix[1][1] =  cos(roll);

  //////Translation matrix //////
  translation_matrix[3][0] = translation.x;
  translation_matrix[3][1] = translation.y;
  translation_matrix[3][2] = translation.z;

  ////Matrix product//////////////////////
  return (translation_matrix * rot_x_matrix * rot_y_matrix * \
     rot_z_matrix);
}
```

Exercice 2 *Complétez la fonction de projection définie dans* ***main.cpp*** *de la manière suivante.*

```
mat4 projection_matrix(double focal);
```

où ***focal*** *représente la focale de la matrice de projection. Cette fonction doit retourner une matrice permettant de projeté les sommets.*

Correction de l'exercice 2 –

Source 2 – Fonction renvoyant la matrice de projection

```
mat4 projection_matrix(double focal)
{
  mat4 proj_matrix;
  proj_matrix[0][0] = focal*window.get_sample();
  proj_matrix[1][1] = focal*window.get_sample();
  proj_matrix[2][0] = window.get_width()*window.get_sample() / 2;
  proj_matrix[2][1] = window.get_height()*window.get_sample() / 2;

  return proj_matrix;
}
```

À présent que vous savez calculer les matrices de transformation et de projection, il vous faut compléter les méthodes de la classe `Object` permettant d'appliquer les transformations et les projections aux sommets.

Dans la classe `Object`, l'ensemble des sommets est contenu dans l'attribut `vertices`. L'ensemble des sommets transformés vont être enregistrés dans l'attribut `vertices_transformed`; l'ensemble des sommets projetés vont être enregistrés dans l'attribut `vertices_projected`.

Exercice 3 *Mettez à jour l'attribut `vertices_transformed` en modifiant la fonction suivante du fichier `object.cpp`.*

```
void update_transformation(mat4 m);
```

N'oubliez pas de mettre à jour les normales contenues dans l'attribut `normals` de l'objet `myobject`; elles seront utiles dans le prochain TD.

Correction de l'exercice 3 –

***Conseil** : Dans un premier temps n'implémentez que la transformation des vertices. Le reste ne sera nécessaire que pour le prochain TP.*

Source 3 – Fonction mettant à jour la transformation des sommets

```
void Object::update_transformation(mat4 m)
{
  vec4 v_out, n_out;
  vertices_transformed.clear();
  for(unsigned int i=0; i<vertices.size(); i++)
  {
    v_out = m * vertices[i];
    vertices_transformed.push_back(v_out);
  }
  for(unsigned int i=0; i<faces.size(); i++)
  {
    faces[i].normal_transformed = m * faces[i].normal;
  }
  normals_transformed.clear();
  for(unsigned int i=0; i<normals.size(); i++)
  {
    n_out = m * normals[i];
    normals_transformed.push_back(n_out);
  }
}
```

Exercice 4 *Mettez à jour l'attribut `vertices_projected` en modifiant la fonction suivante du fichier `object.cpp`.*

```
void update_projection(mat4 m);
```

Correction de l'exercice 4 –

Source 4 – Fonction mettant à jour la transformation des sommets

```
void Object::update_projection(mat4 m)
{
```

```
  vec4 v_out;
  vertices_projected.clear();
  for(unsigned int i=0; i<vertices_transformed.size(); i++)
  {
    v_out = m * vertices_transformed[i];
    vertices_projected.push_back(vec2(v_out.x/v_out.z, v_out.y/v_out.z)\
       );
  }
}
```

Il ne reste plus qu'une étape pour pouvoir afficher les éléments de l'objet : il faut dessiner à l'aide de la fonction `draw_line()` (voir TD précédent) les faces de l'objet.

Exercice 5 *Complétez la fonction suivante pour dessiner dans la fenêtre l'objet* `myobject` *en mode* fil-de-fer.

```
void draw(Window & window);
```

Correction de l'exercice 5 –

Source 5 – Fonction dessinant les faces de l'objet

```
      case DRAW_WIRE :
        if(faces[i].visible)
        {
          for(unsigned int v=0; v<4; v++)
          {
            vec2 p1 = vertices_projected[faces[i].vertex_index[v]];
            vec2 p2 = vertices_projected[faces[i].vertex_index[(v+1)\
               %4]];
            window.draw_line(p1, p2, faces[i].color);
          }
        }
        break;
```