Computer Graphics Fall, 2014

TP 4

Submission date: *10 October*

Problem. The main goal of this assignment is for you to add the model matrix to your myObject3D class.

1. Add the model matrix object to your `myObject3D` class:

   ```
   glm::mat4 model_matrix;
   ```

2. Your `main.cpp` should include:

   ```
   #include <glm/glm.hpp>
   #include <glm/gtc/matrix_transform.hpp>
   #include <glm/gtx/transform.hpp>
   ```

3. Initialize it to the identity matrix in the constructor:

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

4. Add the uniform variable for this matrix in your shaders:

   ```
   uniform mat4 mymodel_matrix;
   ```

5. In the myObject3D::display function, set this matrix for the shaders to use (you will have to pass the `shaderprogram` variable to the display function):

   ```
   glUniformMatrix4fv(glGetUniformLocation(shaderprogram, "mymodel_matrix"),
   1, GL_FALSE, &model_matrix[0][0]);
   ```

6. In the vertex shader, from now on you should also use this matrix for transformations:

   ```
   gl_Position = myprojection_matrix * myview_matrix * mymodel_matrix * vertex_modelspace;
   ```

7. Similarly, in both vertex and fragment shaders, appropriately use this matrix also (for example, in computing the current fragment position).

8. Write functions in your `myObject3D` class to change this matrix, say to add translation to your object, or scaling or rotation. For example, here is what the translation function in `myObject3D` could look like, as well as the rotation function:

   ```
   void translate(double x, double y, double z)
   {
       glm::mat4 tmp = glm::translate(glm::vec3(x,y,z));
       model_matrix = tmp * model_matrix;
   }

   void rotate(double axis_x, double axis_y, double axis_z, double angle)
   {
       glm::mat4 tmp = glm::rotate((float) angle, glm::vec3(axis_x, axis_y, axis_z));
       model_matrix = tmp * model_matrix;
   }
   ```

9. Also remember that the normal matrix will also change with the model matrix (as the object changes!). So you will also need to pass the `view_matrix` to the `myObject3D::display` function, update it, and then set it for the shaders. So part of your display function could include:

```
void displayObject(GLuint shaderprogram, glm::mat4 viewmatrix)
{
    glUniformMatrix4fv(glGetUniformLocation(shaderprogram, "mymodel_matrix"), 1,
                       GL_FALSE, &model_matrix[0][0]);
    glm::mat3 normal_matrix = glm::transpose(glm::inverse(glm::mat3(viewmatrix*model_matrix)));
    glUniformMatrix3fv(glGetUniformLocation(shaderprogram, "mynormal_matrix"), 1,
                       GL_FALSE, &normal_matrix[0][0]);
```

⋮

10. For efficiency, you could pass just one matrix, the `view * model` matrix to the shaders. This is only for efficiency; your code should work without it.