Computer Graphics Labs

Abel J. P. Gomes

LAB. 5

Department of Computer Science and Engineering
University of Beira Interior
Portugal
2020

LAB. 5

PROJECTIONS AND 3D VISUALIZATION

Lab. 5

PROJECTIONS AND 3D VISUALIZATION

In this lab, we deal with 3D scenes and their visualization in 3D space. We use 3D geometric transformations to put 3D objects in the scene, projective transformations to project the scene on a virtual plane in the 3D space, and a viewer immersed in the scene that looks at the scene objects.
Thus, as usual in 2D engineering drafting, to draw a 3D object on a paper sheet we need three entities:

- The viewer (you or me!) that looks at the object;
- The object (or a generic scene);
- The projection plane (paper sheet) where the object is projected.

1. Learning Goals

At the end of this lab, **you should be able to**:

1. To learn building 3D scenes up.
2. Master 3D transformations in computer graphics; you should be able to construct a scene together with objects moving around. These transformations are also used to move a bot or avatar in virtual environments such as, for example, in a First-Person Shooter (FPS) game.
3. Master the details behind the 3D viewing pipeline; you should be able to move the camera/viewer in the scene.

2. 3D Transformations in OpenGL / GLM

Building up a 3D graphics application in OpenGL requires defining the MVP matrix, where M denotes the modelling matrix, V the view matrix, and P the projection matrix. Let us now detail how does the viewing pipeline work for us.

2.1. OpenGL/GML modelling matrix

Like 2D transformations, GLM provides transformations to handle objects and scenes in 3D. We handle each object via its modelling matrix M, which is a result of multiplying various geometric transformations matrices. For example, if we rotate and then translate an object, M is the product of a rotation matrix and a translation matrix.

Have a look at the following web link for further details about the MVP matrix:

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

2.2. OpenGL/GML projections

To display a scene (i.e., a set of several objects) on computer screen, we need to define a 3D domain through an OpenGL/GML projection function. This function returns the projection matrix P, i.e., one of the component matrices of the MVP matrix.

GLM provides three sorts of projection: orthographic, perspective, and frustrum.

The 3D **orthographic projection** is defined by the following GLM function prototype:

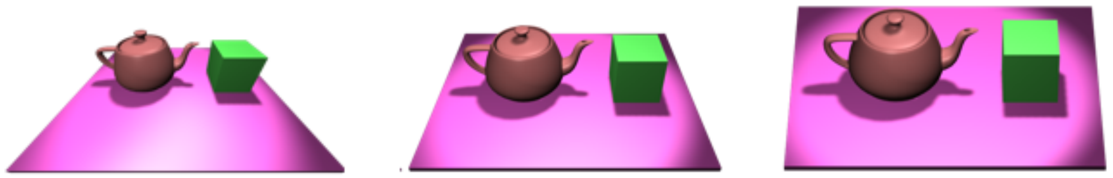
```
glm::mat4 glm::ortho(
       float left, float right,
       float bottom, float top,
       float zNear, float zFar);
```

whose arguments represent the left, right, bottom, top, front, and back planes of the frustum. The viewer (or camera) is located at the infinitum. Note that in 3D we have two additional planes, say the front and back planes. The front plane defined by `zNear` is the projection plane.

The 3D **perspective projection** is defined by the following GLM function prototype:

```
glm::mat4 perspective(
       float fovy, float aspect, float zNear, float zFar);
```

where `zNear` and `zFar` represent (distances from the viewer to) the front and back planes, respectively, `aspect` the aspect ratio, and `fovy` the field-of-view angle.



fovy=122º　　fovy=54º　　fovy=10º

Figure 1: Viewing a scene with distinct field-of-view angles (abusively taken from https://knowww.eu/nodes/59b8e93cd54a862e9d7e40e3).

In turn, the 3D **frustum projection** is defined by the following GLM function prototype:

```
glm::mat4 glm::frustum(
       float left, float right,
       float bottom, float top,
       float zNear, float zFar);
```

which has the same arguments as the ortographic projection function. The difference is that the viewer is at a finite distance from the projection plane `zNear`.

It is worth noting that the **window-viewport mapping** is automatically performed from the the projection plane `zNear` onto the viewport. Also, for more details about the OpenGL/GLM transformations in 3D, the reader is referred to lab. 02.

Let us recall that a viewport is defined as a region of the desktop window that you wish to map the domain window defined by some projection function defined above. Interestingly, we may define multiple viewports for a single domain window, so that we can display the same scene onto different viewports. Furthermore, we can display distinct domain windows (of the same scene) onto distinct viewports.

The OpenGL function prototype for creating a viewport is as follows:

```
glViewport(GLint x, GLint y, GLsizei width, GLsizei height );
```

2.3. The viewer (or camera) in OpenGL / GLM

Finally, to display a scene onto a viewport, we need to specify where the viewer (or camera) is in 3D space.

Theoretically, the viewer is located at the infinitum when we use the **orthographic projection** because the scene is viewed along parallel lines of sight that are perpendicular to the projection plane (or drawing plane). In practice, when we use the orthographic projection function `glm::ortho`, we assume the viewer is located at the origin (0,0,0,); otherwise, we could not see anything of the scene.

By default, the viewer is also located at the origin (0,0,0,) when using either the **frustum projection** function `glm::frustum` or **perspective projection** function `glm::perspective`.

OpenGL/GML allows use to place the viewer at an arbitrary point of the scene using the following function:

```
glm::mat4 glm::lookAt(
        glm::vec3 const & eye,
        glm::vec3 const & center,
        glm::vec3 const & up);
```

where `eye` denotes the viewer position (or eye point), `center` the point to which the viewer is looking at, and up the vector perpendicular to the vector (center-eye); that is, up is the y-axis of the viewer, while (center-eye) defines its z-axis locally.

When the eye is not at the global origin of the scene, the function `glm::lookAt` produces a view matrix V that internally places the viewer at the origin, with its local y and z axes aligned with the global y and z axes of the scene, respectively, without changing the positioning of the viewer relative to the scene. That is, we change the scene from global coordinates to viewer coordinates. Obviously, to fly through a scene, we essentially change the eye point and redraw.

For further details about the camera, have a look at the following web links:

http://www.songho.ca/opengl/gl_camera.html

and

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

and

https://learnopengl.com/Getting-started/Camera

2.4. The MVP matrix

Finally, let us see an example that shows how to build up the *MVP* matrix for a graphics application in modern OpenGL/GLM:

```
glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);
glm::mat4 View       = glm::lookAt(
        glm::vec3(0,0,5), // camera is at (0,0,5), in world space (or scene space)
        glm::vec3(0,0,0), // and looks at the origin
        glm::vec3(0,1,0)  // head is up (set to 0,-1,0 to look upside-down)
);
glm::mat4 Model      = glm::mat4(1.0f);
glm::mat4 MVP        = Projection * View * Model;
```

Note the projection matrix appears before the view matrix and model matrix. Why?

3. Example: The Cube

The following program draws a cube. The program is available at:

http://www.di.ubi.pt/~agomes/cg/teoricas/cube.zip

Questions:

(1) Which are the objects of the scene?
(2) Which is the location of the viewer?
(3) Where is the projection plane?

4. Programming Exercises

1. Re-write the previous example to create a large planar floor defined by a square with the following diagonal vertices: (-100,-100) and (100,100). This floor is in the plane XZ. Also, create and place 12 cubes on the floor.
2. Re-write the previous program to move a cube around the scene. The cube moves on the plane XZ. The allowed movements are: translation along x-axis; translation along z-axis; and rotation about the y-axis. Hint: use the keys x, y, and z for moving the cube interactively.

3. Re-write the previous program in a way that the moving object around in the scene is now the viewer. Use the arrow keys to move around the scene.
4. Build up a 3D house with a single door and no windows. Also, use with distinct color for each part of the house, namely: walls, roof, and door.
5. Enhance the previous program to rotate the house around the z-axis. The counterclockwise rotation is done using the mouse left button, while the clockwise rotation is done using the mouse right button.
6. Change the previous program to include a skyscraper (*arranha-céus*, in Portuguese) on the opposite side of the street. Do not use building windows this time.
7. Change the previous program to include the door number on each building.
8. Change the program in Exercise 2 to move any cube on the floor.
9. Change the program in Exercise 8 to move any cube on the floor with obstacle avoidance; that is, when two cubes collide, the moving cube must stop to avoid overlapping of both cubes.
10. Change the program in Exercise 9 so that when the moving cube collide with a static cube, the moving cube must jump to the top of the static cube.
11. Change the program in Exercise 4 to get an exploded view of the house. An exploded view means to put the secondary house parts away from the primary house part (i.e., the walls). The displacement of the secondary parts must be smooth and controlled by two keys, the first to get away from and the second to get close to the primary part.
12. Change the program in Exercise 4 to build a street with 5 houses of different sizes.