Stefan Johansson
stefanj@cs.umu.se

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE

3D Studio—Part 2

Projection and camera

Purpose

In this part, we will extend our software from Part 1 to handle different projections and using the camera model. The aim is to get a good understanding of the different transformations a single vertex must undergo before it is shown on the screen, in particularly the different aspects of the camera model. We also add a GUI.

Specification

Extend the code from Part 1 such that, in addition to the model matrix *M*, the shader also receives a view matrix *V*, and a projection matrix *P*:

```
gl_Position = P*V*M*vPosition
```

At start, the camera should be located at (0.0, 0.0, 2.0, 1.0) with the reference point at (0.0, 0.0, 0.0, 1.0) and an up-vector (0.0, 1.0, 0.0, 0.0). This is used to define the *V* matrix. Using the GUI, you get values for either *top* and *far,* or *FOV* (Field of view) and *far*. Use these values together with a suitable *near* (chosen by you) to define *left* and *bottom*. This is then used to define your frustum for *P*. The *M* matrix can be computed just like in Part 1. Opening an OBJ-file should now, however, be done in the GUI.

Aspect ratio

Set the viewport as the entire window. Adjust left and right in the frustum so that the aspect ratio of the front plane of the frustum equals the one of the window.

Graphical User Interface

Under 'Files/Project Resources' on Canvas, you find the zip-file *project-part2gui-ImGui.zip* containing some code you may use to get a basic GUI functionality going using Dear ImGui, https://github.com/ocornut/imgui. See ImGui_Readme.pdf in the package for instructions on how to integrate the code into your existing project. You can, if you like, write your own GUI.

GUI functionality

From the GUI the projection type and parameters are set.

Perspective	Use a *perspective* projection.
- Field of view	Change the angle of *field of view* (angle between top and bottom planes), [160°, 20°] (default 60°).
- Far	Specifies the *far* value of the frustum
Parallel	Use a *parallel* projection (default orthographic).
- Top	Specifies the *top* value of the frustum

- Far	Specifies the *far* value of the frustum
- Oblique scale	Ratio (a) between z- and xy-plane, [0,1] (default 0.0).
- Oblique angle	*Degree* (φ) of obliqueness (angle between DOP and PP), [15°, 75°] (default 45°).

For parallel projection the projection matrix $P = M_{\text{orth}} * S * T * H(\varphi)$, where

$$H(\varphi) = \begin{bmatrix} 1 & 0 & a * \cos(\varphi) & 0 \\ 0 & 1 & a * \sin(\varphi) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the product $S*T$ is the scaling and translation performed on the frustum to get it to NDC. However, we do not need to explicitly multiply with the M_{orth} matrix, hence we get $P=S*T*H(\varphi)$. Note that if $a = 0.0$ we have *orthographic* projection and if $0 < a \leq 1$ an *oblique* projection. Especially, an oblique projection with $a = 0.5$ is called *cabinet* projection and with $a = 1$ *cavalier* projection.

Camera functionality

The camera should be possible to rotate around the camera's x- and y-axis using the mouse. A tip is to only activate mouse rotation when the shift key or a mouse button is pressed. The camera should also be translated using the following keys:

Up (E)	Moves p_0 and p_{ref} relative the camera's positive y-axis.
Down (Q)	Moves p_0 and p_{ref} relative the camera's negative y-axis.
Right (D)	Moves p_0 and p_{ref} relative the camera's positive x-axis.
Left (A)	Moves p_0 and p_{ref} relative the camera's negative x-axis.
Forward (W)	Moves p_0 and p_{ref} relative the camera's negative(!) z-axis.
Backwards (S)	Moves p_0 and p_{ref} relative the camera's positive z-axis.

- When tilting the camera up or down with the mouse, the look-at-point should be rotated around the camera's x-axis. When rotating left or right, the look-at-point should be rotated around the camera's y-axis.
- Each time the camera moves, a new look-at-point p_{ref} should be computed such that the look-at-point always is at certain distance d from the camera. Notice the following (see also the handouts for *Coordinate Systems*):

$$V = M_{wc}T(-p_0)$$

$$\Rightarrow$$

$$V^{-1} = T(-p_0)^{-1}M_{wc}^{-1} = T(p_0)M_{wc}^{T}$$

and

$$p^{(camera)} = Vp^{(world)}$$
$$p^{(world)} = V^{-1}p^{(camera)}$$

therefore

$$V^{-1}p_0^{(camera)} = V^{-1}[0 \;\; 0 \;\; 0 \;\; 1]^T = p_0^{(world)}$$

and, equivalently

$$V^{-1}[0 \;\; 0 \;\; -d \;\; 1]^T = p_{ref}^{(world)}$$
$$V^{-1}[0 \;\; 1 \;\; 0 \;\; 0]^T = v_{up}^{(world)}$$

That is, an update of the camera position p_0, reference point p_{ref}, or up-vector v_{up} is easy to express in camera coordinates, which can be multiplied with V^{-1} to get them back to world coordinates. For example, a translation y of the camera along its positive y-axis (up) can be expressed as below, resulting in an updated camera position and reference point which can be used to update V.

$$\begin{aligned}
\hat{p}_0^{(world)} &\leftarrow V^{-1}[0 \;\; y \;\; 0 \;\; 1]^T \\
\hat{p}_{ref}^{(world)} &\leftarrow V^{-1}[0 \;\; y \;\; -1 \;\; 1]^T \\
\hat{v}_{up} &\leftarrow v_{up} \\
\hat{V} &\leftarrow view(\hat{p}_0, \hat{p}_{ref}, \hat{v}_{up})
\end{aligned}$$

Note that if the camera is not tilted sideways (along the z-axis), the v_{up} vector does not need to be updated.

Tips

The model matrix M may lose its orthogonality after a while. This is caused by accumulated error in the computation of the rotation. This is something you do not need to consider, but if you want to solve it there are several solutions to this. One is to normalize the columns of M regularly (make it orthogonal again). Another solution is to use quaternion, but they also may suffer from some errors.

Instructions

This is an individual assignment. The code should follow good programming practice and if you are using any libraries that are not pre-installed, include them in separate subfolders, for example ./lib and ./include, and set appropriate parameters in the Makefile. No written report is required.

Due date is at latest December 8, 2023, 12.00. Times for demonstration are provided December 5 and 8, 10:15-12:00, in the computer labs, if nothing else is agreed upon.