

3D Studio—Part 2

Projections 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 use 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 :

$$\text{gl_Position} = P * V * M * \text{vPosition}$$

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* 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.

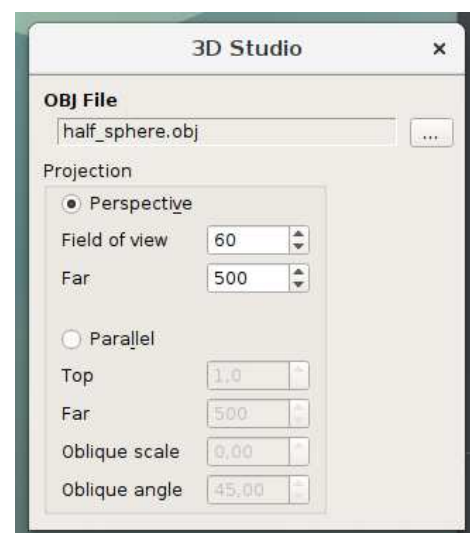
Aspect ratio

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

Graphical User Interface

Under Resources/Project on Cambro, there is a zip-file *project-part2gui.zip* containing some code you may use to get a basic GUI functionality going. This code can be used to create a window that looks like in the picture to the right. You can, if you like, write your own GUI.

The GUI is specified in the XML-file `ass2widget.ui` and the GUI callback functions in `ass2widget.cpp` and



ass2widget.h. To include the GUI in your own project, add these three files to the project and add the following lines in main.cpp:

```
#include "ass2widget.h"
```

and in the main() function

```
Ass2Widget widgets;  
widgets.show();
```

GUI functionality

From the GUI the projection type and parameters are set.

- | | |
|-----------------|------------------------------------------------------------------------------------------------------------|
| Perspective | Use a <i>perspective</i> projection. |
| - Field of view | Change the angle of <i>field of view</i> (angle between top and bottom planes), [160°, 20°] (default 60°). |
| - Far | Specifies the <i>far</i> value in the frustum |
| Parallel | Use a <i>parallel</i> projection (default orthographic). |
| - Top | Specifies the <i>top</i> value in the frustum |
| - Far | Specifies the <i>far</i> value in the frustum |
| - Oblique scale | Ratio (a) between z- and xy-plane, [0,1] (default 0.0). |
| - Oblique angle | <i>Degree</i> (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 and translated using the following keys:

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

- When tilting the camera up or down, 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 *Geometric Transformations*):

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

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

and

$$p^{(camera)} = V p^{(world)}$$

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

therefore

$$V^{-1} p_0^{(camera)} = p_0^{(world)}$$

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

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

$$\Rightarrow V^{-1} [0 1 0 0]^T = v_{up}^{(world)}$$

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 .

$$\hat{p}_0^{(world)} \leftarrow V^{-1} [0 y 0 1]^T$$

$$\hat{p}_{ref}^{(world)} \leftarrow V^{-1} [0 y -1 1]^T$$

$$\hat{v}_{up} \leftarrow v_{up}$$

$$\hat{V} \leftarrow view(\hat{p}_0, \hat{p}_{ref}, \hat{v}_{up})$$

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 also they may suffer from some errors.

Instructions

This is an individual assignment. The code should follow good programming practice and be compliant and executable on the computers in MA416 and

MA426. If you are using any libraries that are not pre-installed, include them in the separate subfolders ./lib and ./include and set appropriate parameters in the Makefile. No written report is required.

Due date is at latest November 30, 2017, 8.15. Time for demonstration is provided November 30, 8:15-12:00, in MA416 and MA426, if nothing else is agreed upon.