Praktikum Echtzeit-Computergrafik Assignment 4 – Projective Geometry and Homogeneous Coordinates

Technische Universität München
Institut für Informatik
Lehrstuhl für Computergrafik & Visualisierung

Christoph Neuhauser, Simon Niedermayr – SS 24

In the last two assignments, you have generated a terrain field, converted this field into an .obj triangle mesh, and loaded the mesh into ShaderLabWeb (https://vmwestermann10.in.tum.de/). Furthermore, you wrote some simple shaders to shade the terrain in different ways. So far, we have taken the three `mat4` objects `mMatrix`, `vMatrix` and `pMatrix` for granted. In this assignment, you will learn more about transformations, projective geometry and homogeneous coordinates.

Introduction

In $\mathbb{R}^3$, the real 3D coordinate space, projective transformations are those transformations that preserve collinearities (i.e., that points that lie on a line still lie on a line after applying the transformation) and can be expressed through matrix multiplications when using so-called *homogeneous coordinates*.

A point in homogeneous coordinates is defined as $p = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$. Scalar multiples of p, i.e., $q = \lambda p$, $\lambda \in \mathbb{R}$, refer to the same point. We can dehomogenize the point p as $p' = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$ when $w \neq 0$.

When displaying an object on our screen, we usually apply three different matrices after each other.

1. The model matrix `mMatrix`. It is usually a concatenation of rotations, translations and scaling operations. It transforms the object space of an individual object/model to world space.

2. The view matrix `vMatrix`. It usually consists of a rotation and a translation. It transforms the world space into view space. In view space (sometimes also called camera space), the coordinate origin is at the position of the camera and the negative z axis (assuming a right-handed coordinate system as used by OpenGL, WebGL and Vulkan) points into the view direction. In some sources, $\texttt{vMatrix}^{-1}$ is called the camera matrix, in most others just the inverse view matrix.

3. The projection matrix `pMatrix`. It is a projective transformation that maps the view space to the normalized device coordinate space. The normalized device coordinate space is a cube of size $[-1, 1]^3$ ($[-1, 1]^2 \times [0, 1]$ in Vulkan and DirectX). These coordinates are then used by the rendering pipeline to get the screen coordinates $[0, w - 1] \times [0, h - 1]$.

The model and view matrix are usually affine transformations. Every affine transformation is a projective transformation, but not every projective transformation is an affine transformation. Affine transformations preserve collinearity and parallelism and are defined in 3D as $T_{affine} = \begin{pmatrix} T_{3x3} & t \\ \vec{0} & 1 \end{pmatrix}$. Below, different types of 3D affine transformations are shown.

- Translation: $T_{translation} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Rotation (counterclockwise around x axis): $T_{rotx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Rotation (counterclockwise around y axis): $T_{roty} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Rotation (counterclockwise around z axis): $T_{rotz} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Scaling: $T_{scale} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Shearing: $T_{shear} = \begin{pmatrix} 1 & \lambda & \mu & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Reflection (here by yz plane): $T_{reflection,xy} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Please try to keep the formulas above in mind. Important facts:

- Rotation matrices are part of the special orthogonal group $SO(n)$. This means that $T_{rot}^{-1} = T_{rot}^{T}$.
- Rotation, scaling, shearing and reflection matrices are linear transformations, which is a subgroup of affine transformations. Linear transformations preserve the origin of the space. Affine transformations can be expressed as a composition of a linear transformation and a translation.

For the projection matrix, usually two types of transformations are used.

- An orthographic projection. Orthographic projections maintain parallel lines.
- A perspective projection. Perspective projections use a field of view (FOV) angle defining the opening angle of the camera frustum (i.e., a cut-off pyramid).

To get an overview on the different types of projection matrices, please read the pages linked below.

- `https://learnwebgl.brown37.net/08_projections/projections_ortho.html`
- `https://learnwebgl.brown37.net/08_projections/projections_perspective.html`

Tasks

Task 1

Derive the transformation matrix that can be used for rotating an object around the origin of rotation $(1, 0, 0)^T$ counterclockwise by 90° around the z axis.

Hint: The formulas for the rotation matrices given above assume that the origin of rotation is at $(0, 0, 0)^T$. Thus, you need to first translate the origin of rotation to $(0, 0, 0)^T$ and then translate the origin of rotation back to its original position.

The result is given below as white text. You can see the solution by highlighting the text in your PDF viewer, or you can wait for the publication of the solutions for this assignment on Moodle.

Task 2

This task is an adapted form of an assignment originally written by Bernhard Kainz for a course offered by the London Imperial College.

ShaderLabWeb (`https://vmwestermann10.in.tum.de/`) provides an interface to `mMatrix`, `vMatrix` and `pMatrix`. The model matrix can be found in the `Uniforms` tab. `mMatrix` is the model matrix and does not need to be attached to anything since we are in full control of this in our case. The default model matrix centers the teapot at the origin. `vMatrix` is the view matrix and is attached to the according matrix in the host program. `pMatrix` is the projection matrix and attached to the view's projection matrix. Your task is to

1. Rotate the object 45° around axis (0.5, 0.5, 0.75).
2. Increase the scale of the object by 50%.
3. Translate the object to (0, 5, 0) followed by a 30° rotation around (0, 0, 1) (z-axis).
4. Reflect the object through a plane defined by its normal vector (0.7071, 0.7071, 0).
5. Shear the object along the x-axis to a general parallelepiped so that the top left edge of the cube is translated to (1, 1, 0).
6. Play with the field of view and change to orthographic projection afterwards.

Reset your matrices to default between the tasks.

Task 3

In the lecture accompanying this practical course you learn that normals need to be transformed using the matrix $(M^{-1})^T$ for the transformation matrix M. When using normals, one usually uses homogeneous coordinates $n = \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix}$ so that the normal vectors are not affected by translations.

Your task is to derive the matrix we need to use for transforming normal vectors for …

1. $M = R_x(\frac{\pi}{2})$, i.e., a counterclockwise rotation of $\frac{\pi}{2}$ around the x-axis.
2. $M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

The answers can again be seen by highlighting the white text below.