Fakultät für Informatik
Labor für Computergrafik
Prof. Dr. G. Umlauf

Hochschule Konstanz
University of Applied Sciences

Konstanz, 06.03.2023

Assignment 1

„Geometric Modeling“

Deadline 26.04.2026.

Preliminary remarks:
Do **not** use functions from OpenGL, GLUT or GLAUX, to compute projections and rotations! Use the provided vector and matrix classes.

Framework for the assignments:
Download the zip-file for the assignments from the web page of the course:

- It contains a VC-project.
- The VC-project contains:
  - **`Aufgabe01.cpp:`**
    Framework for the usage of the display functions (for refresh of the double buffer) and a keyboard function to control the program.
  - **`color.h, AffineGeometry.h, AffineMap.h:`**
    Implementations of a color, point, vector and matrix class. Some methods are implemented exemplarily. Implement the missing methods, if necessary, reusing the provided implementation.
  - **`viewSystem.h, viewSystem.cpp:`**
    Implementation of a view system consisting of an eye point (**`EyePoint`**), a view direction (**`ViewDir`**) and an image plane (**`ViewUp, ViewHor`**) in homogeneous coordinates. The view system realizes the projektions, global coordinaten transformations and affine transformations. It offers three modi to the used implementation of the affine transformations:
    - **`VIEW_MATRIX_MODE, VIEW_FORMULA_MODE:`** Implementation using 4x4 matrices (working).
    - **`VIEW_QUATERNION_MODE:`** Implementation using quaternions (see Part 2).
  - **`quader.h, quader.cpp:`**
    Implementation of a class for the representation of cuboid objects.
  - **`quaternion.h, quaternion.cpp:`**
    Implementation of a class for the representation of quaternions.

The functionality of your implementation will be tested using the source code!

Part 1 (General orientations)

In the framework, the scene can only be viewed from the perspective of the view system (**`ViewSystem`**), which might have an arbitrary position with respect to the world. The view coordinate system is defined by:

- A general position of the eye point (in homogenous coordinates) **`EyePoint`**.
- A general position view direction **`ViewDir`**.
- A general position view-up vector **`ViewUp`**.
- An additional (implicitly given) direction **`ViewHor = ViewDir x ViewUp`**.

Implement in **`viewSystem.cpp`** the methods

`AffineMap viewSystem::getViewToWorld()` and
`AffineMap viewSystem::getWorldToView()`

computing the position and pose of the view coordinate system with respect to the world coordinate system using a 4×4 (see **Figure 1**) and vice versa. This yields a matrix transforming points in world coordinates to points in view coordinates.

When these methods are implemented, the scene can be manipulated in **`VIEW_FORMULA_MODE`** (used in **`Aufgabe01.cpp`**) using the keys from Part 2.

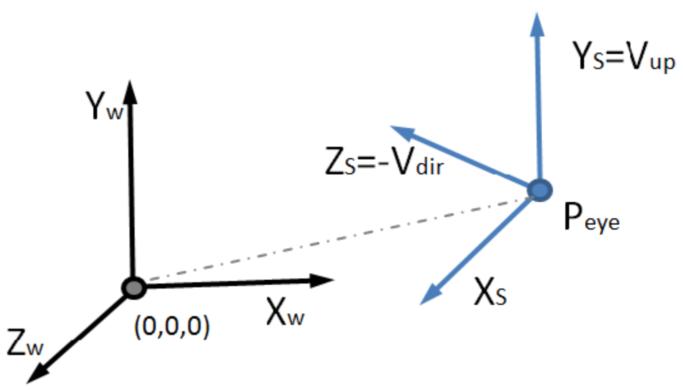

Figure 1 General view transformations.

Part 2 (Quaternions)

Extend your program such that it can be used in **`VIEW_QUATERNION_MODE`**. To this end, implement the missing methods and operations for the keyboard interaction in **`viewSystem.cpp`** and **`quaternion.cpp`**:

a. **`X`**, **`Y`** and **`Z`** rotates the view coordinate system in positive direction around x-, y- and z-axis of the world coordinate system and **`x`**, **`y`** and **`z`** rotates in negative direction around the respective axis.

b. **`A`**, **`B`** and **`C`** rsp. **`a`**, **`b`** and **`c`** rotate the view coordinate system in the respective directions around the axis of the view coordinate system: **`A,a`**: **`ViewDir`**-Vector, **`B,b`**: **`ViewUp`**-Vector, **`C,c`**: **`ViewHor`**-Vector.

c. **`U`**, **`V`**, **`W`**, **`u`**, **`v`** and **`w`** translate the view coordinates system in the directions of the axis of the world coordinate system: **`U,u`**: x-axis, **`V,v`**: y-axis, **`W,w`**: z-axis (not part of the assignment).

d. **`R`** performs a reset of the view coordinate system in the initial position and (not part of the assignment).

e. **`f`** and **`F`** change the focal length of the view system (not part of the assignment).

After the initialization, the world and view coordinate systems coincide.

Part 3 (Interpolation of rotations)

Implement in your program three methods to interpolate rotations. Choose a suitable start and end pose of the scene and interpolate between these two poses sensible intermediate poses using the following three approaches, see [Shoemaker 1985] and [Kremer 2008].

a. Linear Interpolation (LERP) between both poses.

b. Spherical linear interpolation (SLERP) between both poses.

c. Normalized SLERPs (NLERP) between both poses.