ETH zürich

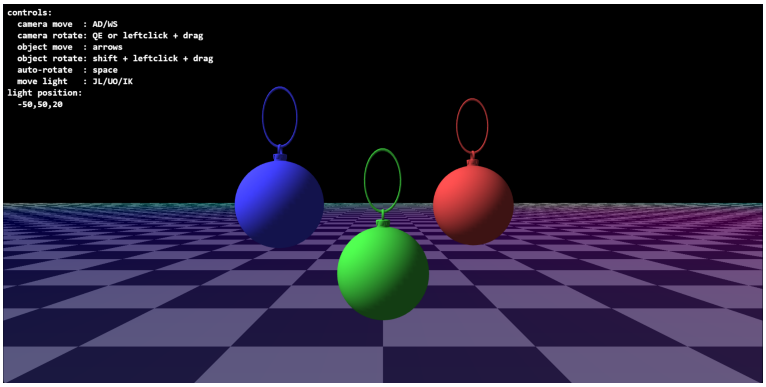
computer graphics laboratory

Prof. Dr. Markus Gross

Visual Computing

Exercise 09: Matrices & Quaternions

Hand-out Date: 19 November 2024



Goals

- Learning about transformations typically used in 3D graphics applications.
- Getting used to different representations of transformations.
- Implementing a series of transformations in a 3D scene.

Resources

The lecture slides and exercise slides are accessible via the Visual Computing Course Web Page.

Tasks

1. Theory

- **Short Questions**:

  (i) What are the three coordinate systems used to describe the scene?

  (ii) State one advantage of using homogeneous coordinates.

  (iii) Given two homogeneous points $\mathbf{p_1} = \begin{bmatrix}4 & 3 & 2 & 1\end{bmatrix}^\top$ and $\mathbf{p_2} = \begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}^\top$, compute the Euclidean displacement vector $\mathbf{d} \in \mathbb{R}^3$ from $\mathbf{p_1}$ to $\mathbf{p_2}$.

- **Homogeneous Coordinates**:

  (i) Decompose the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & -1 & 0 & 47 \\ 1 & 0 & 0 & 11 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  into a linear transformation $\mathbf{L}$ and a translation $\mathbf{T}$, such that $\mathbf{A} = \mathbf{LT}$ (Note: not $\mathbf{TL}$).

  (ii) Describe in words what the transformations $\mathbf{L}$ and $\mathbf{T}$ represent.

  (iii) In which order are $\mathbf{L}$ and $\mathbf{T}$ processed, if applied to a point in 3D?

  (iv) Let $\mathbf{n}$ be the normal of a plane $\mathbf{H}$ in "Hessian normal form", i.e., $\mathbf{H}$ can be defined as $ax + by + cz = -d$. Thus, for all points $\mathbf{p} \in \mathbf{H}$, we have $\mathbf{p}^\top \mathbf{n} = 0$. The linear transformation $\mathbf{A}$ given above maps all $\mathbf{p} \in \mathbf{H}$ to points $\mathbf{p}'$ in a second plane $\mathbf{H}'$ ($\mathbf{p}' = \mathbf{Ap} \in \mathbf{H}'$). Our goal is to compute a matrix $\mathbf{B}$ that maps $\mathbf{n}$ to the normal $\mathbf{n}'$ of the plane $\mathbf{H}'$ such that $\mathbf{n}' = \mathbf{Bn}$ (Hint: Compute $\mathbf{B}$ using the decomposition $\mathbf{A} = \mathbf{LT}$).

- **Quaternions**:

  (i) Assemble the unit quaternion $\mathbf{q} = c + xi + yj + zk$ which describes a rotation of 60° around the rotation axis $\mathbf{u} = \begin{bmatrix} 3 & 0 & 4 \end{bmatrix}^\top$. Simplify the quaternion as much as possible.

  (ii) Directly compute the quaternion of the inverse rotation from $\mathbf{q}$.

  (iii) Which elementary quaternion operation have you used in (ii)?

  (iv) Given the quaternion $\mathbf{r} = \begin{bmatrix} c & x & y & z \end{bmatrix}^\top = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^\top$. What rotation does $\mathbf{r}$ correspond to?

  (v) Specify the rotation matrix that corresponds to $\mathbf{r}$.

2. Practice

In this part, you will implement different transformations in code. First, you will construct the model matrix and the view matrix. In a second step, you will add functionality for handling user input (i.e., translations and rotations based on keyboard and mouse events). Download the code from the GitHub repository and get familiar with the code in `ex08.js`. Run the code in a browser (refer to the GitHub repository and the exercise slides for details). You should see a black screen with white text on the upper left explaining the key instructions.

- **ModelView Transform**: Find the part in the code marked as "Task 1". Your task is to complete the functions `getCameraMatrix()`, `getViewMatrix()`, and `getModelMatrix()`.

(i) Implement the `getCameraMatrix()` function by following the code comments. The camera should look down the negative z-axis. Use the result in `getViewMatrix()` to compute the view matrix. If implemented correctly, you should see a tiled floor.

(ii) Next, implement the model matrix in `getModelMatrix()` using the described transformations. Make sure that the transformations are applied in the correct order. If implemented correctly, you will see the mesh from the previous exercise appear. Press the space key to rotate the mesh automatically.

- **User Input**: Find the part in the code marked as "Task 2". In this part, you will complete the functions needed to handle user input for translating and rotating the mesh and the camera, and changing the light position. The keyboard and mouse event handling is already implemented for you.

  (i) First, complete `moveCamera(...)`. The camera should move by a specific distance either forward/backward, or sideways, depending on the 'sideways' flag.

  (ii) Next, complete `rotateCameraX(...)` and `rotateCameraY(...)`. Think about what the origin of rotation is before applying the rotation.

  (iii) Finally, complete `moveLight(...)` by adding a displacement vector 'd' to the light position.

  If implemented correctly, you will be able to rigidly transform the mesh using the keyboard and the mouse as indicated by the instructions on the upper left of the screen.