ETH zürich

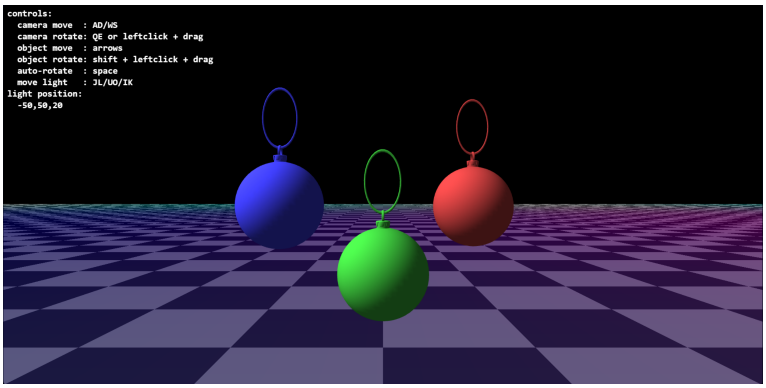
computer graphics laboratory

Prof. Dr. Markus Gross

Visual Computing

Solution 09: Matrices & Quaternions

Hand-out Date: 21 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?
  Answer: object coordinates, world coordinates, and camera coordinates.

  (ii) State one advantage of using homogeneous coordinates.
  Answer: All transformations can be expressed as matrix operations. Furthermore, an arbitrary number of affine and projective mappings, applied one after the other, can be combined in one single matrix.

(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 $\mathbf{p_2}$.
Answer: To compare $\mathbf{p_1}$ and $\mathbf{p_2}$ in $\mathbb{R}^3$, we must first make them homogeneous. This is done by simply dividing the first three components by the fourth component of the point vectors. The corresponding vectors in $\mathbb{R}^3$ are $\tilde{\mathbf{p}}_\mathbf{1} = \begin{bmatrix}4 & 3 & 2\end{bmatrix}^\top$ and $\tilde{\mathbf{p}}_\mathbf{2} = \frac{1}{4}\begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$, and we therefore have $\mathbf{d} = \tilde{\mathbf{p}}_\mathbf{2} - \tilde{\mathbf{p}}_\mathbf{1} = -\frac{1}{4}\begin{bmatrix}15 & 10 & 5\end{bmatrix}^\top$ as the displacement vector.

- **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 $\mathbf{TL}$).
Answer: From the lecture, we know how to decompose into $\mathbf{T'L'}$ where $\mathbf{L'}$ is a rotation that is applied first, and $\mathbf{T'}$ is a translation that is applied second. There, the corresponding translation vector would be $\mathbf{t'} = \begin{bmatrix}47 & 11 & 0\end{bmatrix}^\top$, and $\mathbf{L'}$ would represent a rotation around the z-axis by 90°. In this case here, the translation should be applied first, thus the translation is also affected by the rotation. We need to account for the rotation in the translation vector such that after rotation we get the same translation again. In other words, we rotate the translation into the other direction (i.e., by −90° around the z-axis) which gives $\mathbf{t} = \begin{bmatrix}11 & -47 & 0\end{bmatrix}^\top$. The rotation stays the same, which gives us

$$\mathbf{L} = \begin{bmatrix}0 & -1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix}1 & 0 & 0 & 11\\ 0 & 1 & 0 & -47\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

(ii) Describe in words what the transformations $\mathbf{L}$ and $\mathbf{T}$ represent.
Answer: $\mathbf{L}$ is a 90° rotation around the z-axis, and $\mathbf{T}$ is a translation by $\begin{bmatrix}11 & -47 & 0\end{bmatrix}^\top$.

(iii) In which order are $\mathbf{L}$ and $\mathbf{T}$ processed, if applied to a point in 3D?
Answer: The point is first translated by $\mathbf{T}$ and then rotated by $\mathbf{L}$.

(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 $\mathbf{A} = \mathbf{LT}$).
Answer: We have

$$\begin{aligned}\mathbf{n}'^{\top}\mathbf{p}' &= \mathbf{n}'^{\top}\mathbf{Ap} \\ &= \mathbf{n}'^{\top}\mathbf{LTp} \\ &= (\mathbf{T}^{\top}\mathbf{L}^{\top}\mathbf{n}')^{\top}\mathbf{p} \\ &= 0.\end{aligned}$$

Comparing this with $\mathbf{n}^{\top}\mathbf{p} = 0$ and noting that these are true for all $\mathbf{p} \in \mathbf{H}$, we see that a mapping from $\mathbf{n}$ to $\mathbf{n}'$ is given by $\mathbf{T}^{\top}\mathbf{L}^{\top}$. Hence, $\mathbf{n}' = (\mathbf{L}^{\top})^{-1}(\mathbf{T}^{\top})^{-1}\mathbf{n} = \mathbf{L}(\mathbf{T}^{\top})^{-1}\mathbf{n}$ since $\mathbf{L}$ is a unitary matrix. Thus, we get

$$\mathbf{B} = \mathbf{L}(\mathbf{T}^{\top})^{-1} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -11 & 47 & 0 & 1 \end{bmatrix}.$$

- **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.
  Answer: The rotation of an arbitrary 3D-point $\mathbf{p}$ by the rotation angle ϕ around an axis $\mathbf{n}$ is described by the quaternion operation $\mathbf{R_q} = \mathbf{qp\bar{q}}$, where point $\mathbf{p}$ is represented by a pure quaternion and $\mathbf{q}$ is a unit quaternion. This unit quaternion $\mathbf{q}$ can be written as $\mathbf{q} = \cos\frac{\phi}{2} + \sin\frac{\phi}{2}\mathbf{n}$. Thus, if we write $\mathbf{n} = \frac{\mathbf{u}^{\top}}{\|\mathbf{u}\|} = \begin{bmatrix}0.6 & 0 & 0.8\end{bmatrix}^{\top}$, this implies that we have $\mathbf{q} = \cos 30° + \sin 30°\mathbf{n} = \frac{\sqrt{3}}{2} + 0.3i + 0.4k$.

  (ii) Directly compute the quaternion of the inverse rotation from $\mathbf{q}$.
  Answer: $\mathbf{q}^{-1} = \dfrac{\bar{\mathbf{q}}}{\|\mathbf{q}\|_2} = \bar{\mathbf{q}} = \dfrac{\sqrt{3}}{2} - 0.3i - 0.4k.$

  (iii) Which elementary quaternion operation have you used in (ii)?
  Answer: Inversion, inversion of the unit quaternion, conjugation.

  (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?
  Answer: We write the quaternion in two different forms and look for correspondences. On one hand, $\mathbf{r} = \cos\frac{\phi}{2} + \sin\frac{\phi}{2}\mathbf{n}$ where $\mathbf{n} = (1i + 0j + 0k) = i$. On the other hand, we have $\mathbf{r} = 0 + 1i + 0j + 0 = i$. Thus, we have $\cos\frac{\phi}{2} + \sin\frac{\phi}{2}i = i$, which is true for $\phi = \pi + c \cdot 2\pi \quad \forall c \in \mathbb{Z}$. We conclude that $\mathbf{r}$ describes a rotation of 180° around the x-axis.

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

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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.