Curve and surface modeling

– a CAGD approach based on OpenGL and C++ –

Ágoston Róth

Department of Mathematics and Computer Science, Babeș-Bolyai University, Cluj-Napoca, România

(agoston.roth@gmail.com)

Seminar 1 – February 21 & 28, 2022

Coordinate transformations

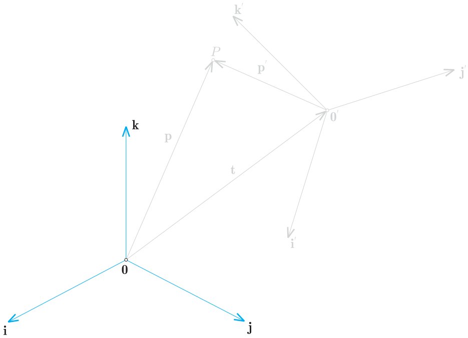

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system $\mathbf{0ijk}$.
- Let $\mathbf{0'i'j'k'}$ also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems $\mathbf{0ijk}$ and $\mathbf{0'i'j'k'}$.

Coordinate transformations

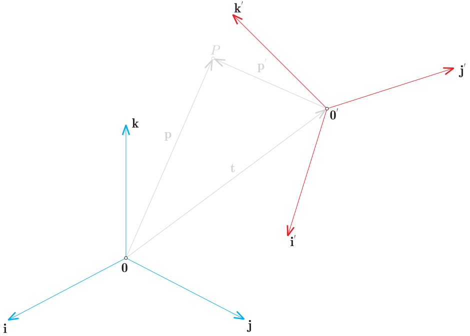

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system $\mathbf{0ijk}$.
- Let $\mathbf{0'i'j'k'}$ also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems $\mathbf{0ijk}$ and $\mathbf{0'i'j'k'}$.

Coordinate transformations

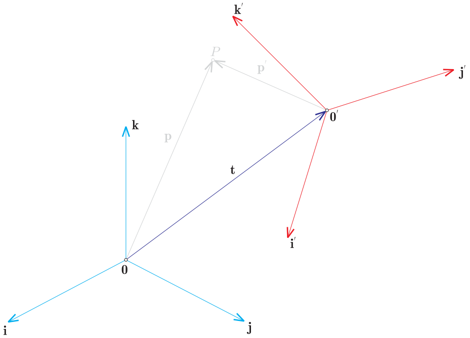

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system **0ijk**.
- Let **0′i′j′k′** also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems **0ijk** and **0′i′j′k′**.

Coordinate transformations

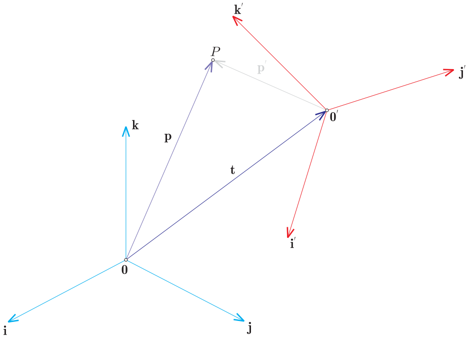

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system $\mathbf{0ijk}$.
- Let $\mathbf{0'i'j'k'}$ also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems $\mathbf{0ijk}$ and $\mathbf{0'i'j'k'}$.

Coordinate transformations

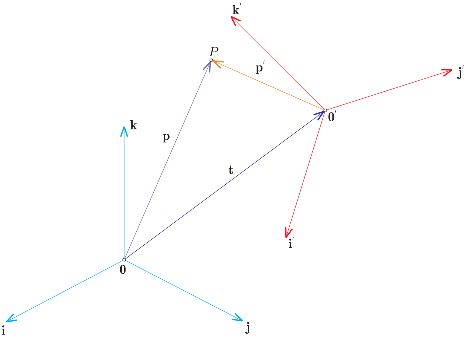

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system $\mathbf{0ijk}$.
- Let $\mathbf{0'i'j'k'}$ also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems $\mathbf{0ijk}$ and $\mathbf{0'i'j'k'}$.

Coordinate transformations

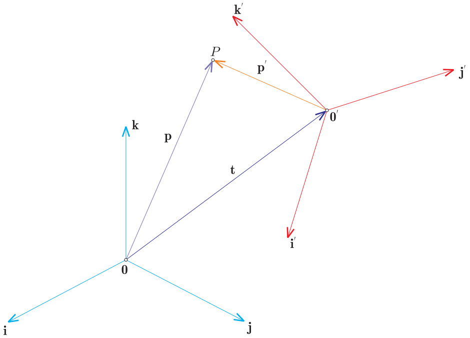

Fig. 1: Position vectors of a point in two different Cartesian, right-handed and orthonormal coordinate systems.

- Consider the classical Cartesian right-handed and orthonormal coordinate system **0ijk**.
- Let $\mathbf{0}'\mathbf{i}'\mathbf{j}'\mathbf{k}'$ also be a Cartesian right-handed and orthonormal coordinate system.
- Let $\mathbf{t}(t_x, t_y, t_z)$ be the position vector of $\mathbf{0}'$ in the old coordinate system.
- Let $\mathbf{p}(x, y, z)$ the position vector of a point $P \in \mathbb{R}^3$ in the old coordinate system.
- Denote by $\mathbf{p}'(x', y', z')$ the position vector of the point P in the new coordinate system.
- Our goal is to determine the transformation and its inverse between coordinate systems **0ijk** and $\mathbf{0}'\mathbf{i}'\mathbf{j}'\mathbf{k}'$.

Proposition ($\mathbf{p}'(x', y', z') \stackrel{?}{\rightarrow} \mathbf{p}(x, y, z)$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & t_x \\ i'_y & j'_y & k'_y & t_y \\ i'_z & j'_z & k'_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}.$$

Proof.

- Based on Fig. 1 we can write:

Proposition ($\mathbf{p}'(x', y', z') \overset{?}{\to} \mathbf{p}(x, y, z)$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & t_x \\ i'_y & j'_y & k'_y & t_y \\ i'_z & j'_z & k'_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}.$$

Proof.

- Based on Fig. 1 we can write:

$$\begin{aligned} \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \mathbf{p}' + \mathbf{t} \\ &= (x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}') + \mathbf{t} \\ &= x' \begin{bmatrix} i'_x \\ i'_y \\ i'_z \end{bmatrix} + y' \begin{bmatrix} j'_x \\ j'_y \\ j'_z \end{bmatrix} + z' \begin{bmatrix} k'_x \\ k'_y \\ k'_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \end{aligned}$$

Proposition ($\mathbf{p}'(x', y', z') \xrightarrow{?} \mathbf{p}(x, y, z)$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & t_x \\ i'_y & j'_y & k'_y & t_y \\ i'_z & j'_z & k'_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}.$$

Proof.

- Based on Fig. 1 we can write:

$$\begin{aligned} \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \mathbf{p}' + \mathbf{t} \\ &= (x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}') + \mathbf{t} \\ &= x' \begin{bmatrix} i'_x \\ i'_y \\ i'_z \end{bmatrix} + y' \begin{bmatrix} j'_x \\ j'_y \\ j'_z \end{bmatrix} + z' \begin{bmatrix} k'_x \\ k'_y \\ k'_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \end{aligned}$$

Proposition ($\mathbf{p}'(x', y', z') \xrightarrow{?} \mathbf{p}(x, y, z)$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & t_x \\ i'_y & j'_y & k'_y & t_y \\ i'_z & j'_z & k'_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}.$$

Proof.

- Based on Fig. 1 we can write:

$$\begin{aligned} \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \mathbf{p}' + \mathbf{t} \\ &= (x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}') + \mathbf{t} \\ &= x' \begin{bmatrix} i'_x \\ i'_y \\ i'_z \end{bmatrix} + y' \begin{bmatrix} j'_x \\ j'_y \\ j'_z \end{bmatrix} + z' \begin{bmatrix} k'_x \\ k'_y \\ k'_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \end{aligned}$$

Proposition ($\mathbf{p}'(x', y', z') \stackrel{?}{\to} \mathbf{p}(x, y, z)$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & t_x \\ i'_y & j'_y & k'_y & t_y \\ i'_z & j'_z & k'_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}.$$

Proof.

- Based on Fig. 1 we can write:

$$\begin{aligned} \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \mathbf{p}' + \mathbf{t} \\ &= (x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}') + \mathbf{t} \\ &= x' \begin{bmatrix} i'_x \\ i'_y \\ i'_z \end{bmatrix} + y' \begin{bmatrix} j'_x \\ j'_y \\ j'_z \end{bmatrix} + z' \begin{bmatrix} k'_x \\ k'_y \\ k'_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \end{aligned}$$

□

Proposition ($\mathbf{p}(x, y, z) \xrightarrow{?} \mathbf{p}'(x', y', z')$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{t} \cdot \mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t} \cdot \mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t} \cdot \mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

Proposition ($\mathbf{p}(x, y, z) \xrightarrow{?} \mathbf{p}'(x', y', z')$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\left[\begin{array}{c} x' \\ y' \\ z' \\ 1 \end{array}\right] = \left[\begin{array}{cccc} i'_x & i'_y & i'_z & -\mathbf{t}\cdot\mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t}\cdot\mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t}\cdot\mathbf{k}' \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} x \\ y \\ z \\ 1 \end{array}\right].$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

$$\begin{aligned} x' &= \mathbf{p}'\cdot\mathbf{i}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{i}' = \mathbf{p}\cdot\mathbf{i}' - \mathbf{t}\cdot\mathbf{i}' = \left(xi'_x + yi'_y + zi'_z\right) - \mathbf{t}\cdot\mathbf{i}', \\ y' &= \mathbf{p}'\cdot\mathbf{j}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{j}' = \mathbf{p}\cdot\mathbf{j}' - \mathbf{t}\cdot\mathbf{j}' = \left(xj'_x + yj'_y + zj'_z\right) - \mathbf{t}\cdot\mathbf{j}', \\ z' &= \mathbf{p}'\cdot\mathbf{k}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{k}' = \mathbf{p}\cdot\mathbf{k}' - \mathbf{t}\cdot\mathbf{k}' = \left(xk'_x + yk'_y + zk'_z\right) - \mathbf{t}\cdot\mathbf{k}' \end{aligned}$$

after which we switch to the homogeneous representation of coordinates.

Proposition ($\mathbf{p}(x, y, z) \stackrel{?}{\to} \mathbf{p}'(x', y', z')$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{t}\cdot\mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t}\cdot\mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t}\cdot\mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

$$\begin{aligned} x' &= \mathbf{p}'\cdot\mathbf{i}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{i}' = \mathbf{p}\cdot\mathbf{i}' - \mathbf{t}\cdot\mathbf{i}' = \left(xi'_x + yi'_y + zi'_z\right) - \mathbf{t}\cdot\mathbf{i}', \\ y' &= \mathbf{p}'\cdot\mathbf{j}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{j}' = \mathbf{p}\cdot\mathbf{j}' - \mathbf{t}\cdot\mathbf{j}' = \left(xj'_x + yj'_y + zj'_z\right) - \mathbf{t}\cdot\mathbf{j}', \\ z' &= \mathbf{p}'\cdot\mathbf{k}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{k}' = \mathbf{p}\cdot\mathbf{k}' - \mathbf{t}\cdot\mathbf{k}' = \left(xk'_x + yk'_y + zk'_z\right) - \mathbf{t}\cdot\mathbf{k}', \end{aligned}$$

after which we switch to the homogeneous representation of coordinates.

Proposition $(\mathbf{p}(x,y,z) \xrightarrow{?} \mathbf{p}'(x',y',z'))$

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{t}\cdot\mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t}\cdot\mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t}\cdot\mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

$$\begin{aligned} x' &= \mathbf{p}'\cdot\mathbf{i}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{i}' = \mathbf{p}\cdot\mathbf{i}' - \mathbf{t}\cdot\mathbf{i}' = \left(xi'_x + yi'_y + zi'_z\right) - \mathbf{t}\cdot\mathbf{i}', \\ y' &= \mathbf{p}'\cdot\mathbf{j}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{j}' = \mathbf{p}\cdot\mathbf{j}' - \mathbf{t}\cdot\mathbf{j}' = \left(xj'_x + yj'_y + zj'_z\right) - \mathbf{t}\cdot\mathbf{j}', \\ z' &= \mathbf{p}'\cdot\mathbf{k}' = (\mathbf{p}-\mathbf{t})\cdot\mathbf{k}' = \mathbf{p}\cdot\mathbf{k}' - \mathbf{t}\cdot\mathbf{k}' = \left(xk'_x + yk'_y + zk'_z\right) - \mathbf{t}\cdot\mathbf{k}' \end{aligned}$$

after which we switch to the homogeneous representation of coordinates.

Proposition ($\mathbf{p}(x, y, z) \xrightarrow{?} \mathbf{p}'(x', y', z')$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{t} \cdot \mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t} \cdot \mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t} \cdot \mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

$$\begin{aligned} x' &= \mathbf{p}' \cdot \mathbf{i}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{i}' = \mathbf{p} \cdot \mathbf{i}' - \mathbf{t} \cdot \mathbf{i}' = \left(xi'_x + yi'_y + zi'_z\right) - \mathbf{t} \cdot \mathbf{i}', \\ y' &= \mathbf{p}' \cdot \mathbf{j}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{j}' = \mathbf{p} \cdot \mathbf{j}' - \mathbf{t} \cdot \mathbf{j}' = \left(xj'_x + yj'_y + zj'_z\right) - \mathbf{t} \cdot \mathbf{j}', \\ z' &= \mathbf{p}' \cdot \mathbf{k}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{k}' = \mathbf{p} \cdot \mathbf{k}' - \mathbf{t} \cdot \mathbf{k}' = \left(xk'_x + yk'_y + zk'_z\right) - \mathbf{t} \cdot \mathbf{k}', \end{aligned}$$

after which we switch to the homogeneous representation of coordinates.

Proposition ($\mathbf{p}(x, y, z) \xrightarrow{?} \mathbf{p}'(x', y', z')$)

If (i'_x, i'_y, i'_z), (j'_x, j'_y, j'_z) and (k'_x, k'_y, k'_z) denote in the old coordinate system the components of vectors $\mathbf{i}'$, $\mathbf{j}'$, and $\mathbf{k}'$, respectively, then

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & i'_y & i'_z & -\mathbf{t} \cdot \mathbf{i}' \\ j'_x & j'_y & j'_z & -\mathbf{t} \cdot \mathbf{j}' \\ k'_x & k'_y & k'_z & -\mathbf{t} \cdot \mathbf{k}' \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Proof.

- By projecting the position vector $\mathbf{p}'$ onto the unit vectors of the new coordinate system, we can successively write:

$$\begin{aligned} x' &= \mathbf{p}' \cdot \mathbf{i}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{i}' = \mathbf{p} \cdot \mathbf{i}' - \mathbf{t} \cdot \mathbf{i}' = \left(x i'_x + y i'_y + z i'_z\right) - \mathbf{t} \cdot \mathbf{i}', \\ y' &= \mathbf{p}' \cdot \mathbf{j}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{j}' = \mathbf{p} \cdot \mathbf{j}' - \mathbf{t} \cdot \mathbf{j}' = \left(x j'_x + y j'_y + z j'_z\right) - \mathbf{t} \cdot \mathbf{j}', \\ z' &= \mathbf{p}' \cdot \mathbf{k}' = (\mathbf{p} - \mathbf{t}) \cdot \mathbf{k}' = \mathbf{p} \cdot \mathbf{k}' - \mathbf{t} \cdot \mathbf{k}' = \left(x k'_x + y k'_y + z k'_z\right) - \mathbf{t} \cdot \mathbf{k}', \end{aligned}$$

  after which we switch to the homogeneous representation of coordinates. □

- In general, a point transformation could be any function mapping a set S onto another set or onto itself.
- However, often the set S has some additional algebraic or geometric structure and the term "transformation" refers to a function from S to itself which preserves this structure.
- We will study bijective, semi-affine and linear transformations.
- The matrix representation of a transformation is

$$\mathbf{p}' = M\mathbf{p},$$

where $\mathbf{p}$ and $\mathbf{p}'$ are the homogeneous coordinates of the original and of the transformed position vector, while M is a 4×4 matrix that represents the transformation itself.
- In what follows we assume that M is a bijective transformation, i.e. $\exists M^{-1}$ or equivalently $\det M \neq 0$.

Point transformations

- In general, a point transformation could be any function mapping a set S onto another set or onto itself.
- However, often the set S has some additional algebraic or geometric structure and the term "transformation" refers to a function from S to itself which preserves this structure.
- We will study bijective, semi-affine and linear transformations.
- The matrix representation of a transformation is

$$\mathbf{p}' = M\mathbf{p},$$

where $\mathbf{p}$ and $\mathbf{p}'$ are the homogeneous coordinates of the original and of the transformed position vector, while M is a 4×4 matrix that represents the transformation itself.
- In what follows we assume that M is a bijective transformation, i.e. $\exists M^{-1}$ or equivalently $\det M \neq 0$.

Point transformations

- In general, a point transformation could be any function mapping a set S onto another set or onto itself.
- However, often the set S has some additional algebraic or geometric structure and the term "transformation" refers to a function from S to itself which preserves this structure.
- We will study bijective, semi-affine and linear transformations.
- The matrix representation of a transformation is

$$\mathbf{p}^{'} = M\mathbf{p},$$

  where $\mathbf{p}$ and $\mathbf{p}^{'}$ are the homogeneous coordinates of the original and of the transformed position vector, while M is a 4×4 matrix that represents the transformation itself.
- In what follows we assume that M is a bijective transformation, i.e. $\exists M^{-1}$ or equivalently $\det M \neq 0$.

Point transformations

- In general, a point transformation could be any function mapping a set S onto another set or onto itself.
- However, often the set S has some additional algebraic or geometric structure and the term "transformation" refers to a function from S to itself which preserves this structure.
- We will study bijective, semi-affine and linear transformations.
- The matrix representation of a transformation is

$$\mathbf{p}' = M\mathbf{p},$$

where $\mathbf{p}$ and $\mathbf{p}'$ are the homogeneous coordinates of the original and of the transformed position vector, while M is a 4×4 matrix that represents the transformation itself.

- In what follows we assume that M is a bijective transformation, i.e. $\exists M^{-1}$ or equivalently $\det M \neq 0$.

Point transformations

- In general, a point transformation could be any function mapping a set S onto another set or onto itself.
- However, often the set S has some additional algebraic or geometric structure and the term "transformation" refers to a function from S to itself which preserves this structure.
- We will study bijective, semi-affine and linear transformations.
- The matrix representation of a transformation is

  $$\mathbf{p}' = M\mathbf{p},$$

  where $\mathbf{p}$ and $\mathbf{p}'$ are the homogeneous coordinates of the original and of the transformed position vector, while M is a 4×4 matrix that represents the transformation itself.
- In what follows we assume that M is a bijective transformation, i.e. $\exists M^{-1}$ or equivalently $\det M \neq 0$.

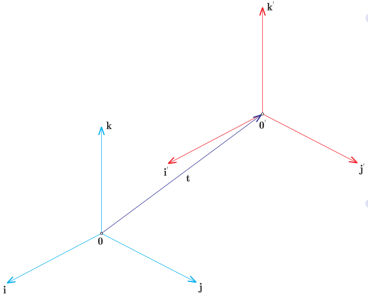

Fig. 2: Translation

- Consider the translation vector $\mathbf{t}(t_x, t_y, t_z)$.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glTranslatef(GLfloat x, GLfloat y, GLfloat z);
GLvoid glTranslated(GLdouble x, GLdouble y, GLdouble z);
```

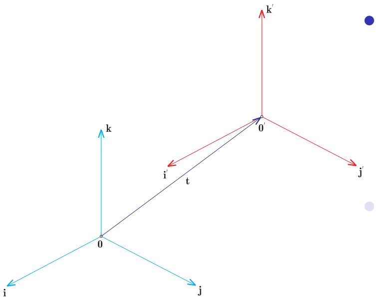

Fig. 2: Translation

- Consider the translation vector $\mathbf{t}(t_x, t_y, t_z)$.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glTranslatef(GLfloat x, GLfloat y, GLfloat z);
GLvoid glTranslated(GLdouble x, GLdouble y, GLdouble z);
```

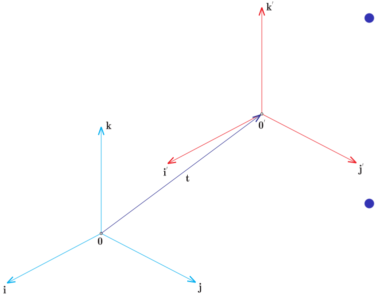

Fig. 2: Translation

- Consider the translation vector $\mathbf{t}(t_x, t_y, t_z)$.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glTranslatef(GLfloat x, GLfloat y, GLfloat z);
GLvoid glTranslated(GLdouble x, GLdouble y, GLdouble z);
```

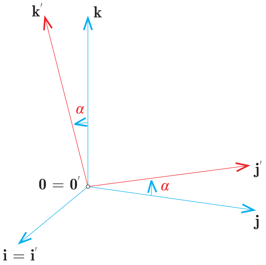

Fig. 3: Rotation around axis $\mathbf{0}x$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 1.0f, 0.0f, 0.0f);

...

glRotated(angle_d, 1.0, 0.0, 0.0);
```

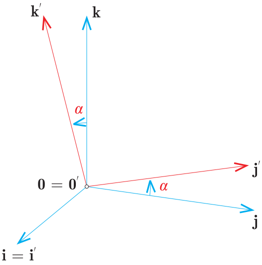

Fig. 3: Rotation around axis $\mathbf{0}x$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 1.0f, 0.0f, 0.0f);

...

glRotated(angle_d, 1.0, 0.0, 0.0);
```

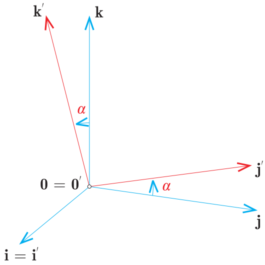

Fig. 3: Rotation around axis $\mathbf{0}x$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 1.0f, 0.0f, 0.0f);

...

glRotated(angle_d, 1.0, 0.0, 0.0);
```

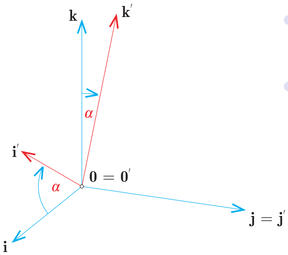

Fig. 4: Rotation around axis $\mathbf{0}y$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 1.0f, 0.0f);

...

glRotated(angle_d, 0.0, 1.0, 0.0);
```

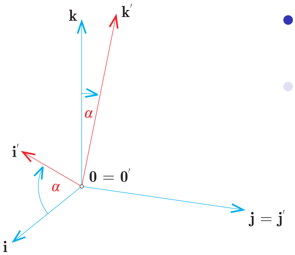

Fig. 4: Rotation around axis $\mathbf{0}y$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 1.0f, 0.0f);

...

glRotated(angle_d, 0.0, 1.0, 0.0);
```

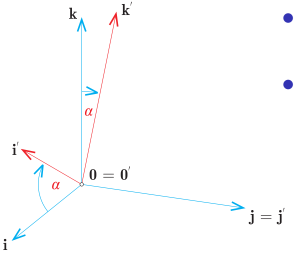

Fig. 4: Rotation around axis $\mathbf{0}y$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 1.0f, 0.0f);

...

glRotated(angle_d, 0.0, 1.0, 0.0);
```

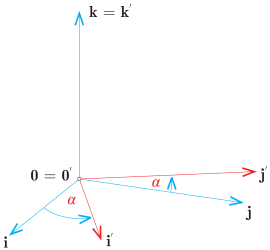

Fig. 5: Rotation around axis $\mathbf{0}z$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 0.0f, 1.0f);

...

glRotated(angle_d, 0.0, 0.0, 1.0);
```

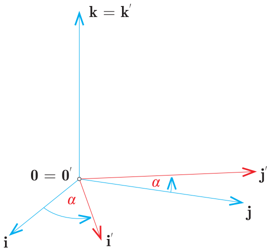

Fig. 5: Rotation around axis $\mathbf{0}z$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 0.0f, 1.0f);

...

glRotated(angle_d, 0.0, 0.0, 1.0);
```

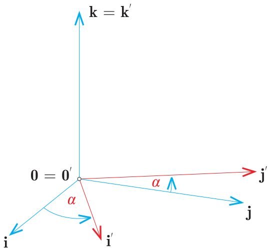

Fig. 5: Rotation around axis $\mathbf{0}z$.

- Consider the rotation angle $\alpha \in \mathbb{R}$.
- If $\alpha > 0$, then we rotate in counter-clockwise direction, otherwise, in clockwise direction.
- The transformation is described by the matrix

$$M = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  angle_f = ...; // in degrees
GLdouble angle_d = ...; // in degrees

...

glRotatef(angle_f, 0.0f, 0.0f, 1.0f);

...

glRotated(angle_d, 0.0, 0.0, 1.0);
```

- Consider the rotation angle $\alpha \in \mathbb{R}$ and the unit direction vector $\mathbf{u}(u_x, u_y, u_z)$.
- Consider the 3×3 transformation matrix

$$T = I_3 + U \sin\alpha + U^2 (1 - \cos\alpha),$$

where

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{bmatrix}$$

is a skew-symmetric matrix.
- The compact homogeneous representation of transformation T is

$$M = \begin{bmatrix} 1 + \left(u_y^2 + u_z^2\right)(\cos\alpha - 1) & u_z \sin\alpha + u_x u_y (1 - \cos\alpha) & -u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & 0 \\ u_x u_y (1 - \cos\alpha) - u_z \sin\alpha & 1 + \left(u_x^2 + u_z^2\right)(\cos\alpha - 1) & u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 0 \\ u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & -u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 1 + \left(u_x^2 + u_y^2\right)(\cos\alpha - 1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- You can use the OpenGL commands:

```
GLvoid glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
GLvoid glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
```

- Consider the rotation angle $\alpha \in \mathbb{R}$ and the unit direction vector $\mathbf{u}(u_x, u_y, u_z)$.
- Consider the 3×3 transformation matrix

$$T = I_3 + U \sin\alpha + U^2 (1 - \cos\alpha),$$

where

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{bmatrix}$$

is a skew-symmetric matrix.

- The compact homogeneous representation of transformation T is

$$M = \begin{bmatrix} 1 + \left(u_y^2 + u_z^2\right)(\cos\alpha - 1) & u_z \sin\alpha + u_x u_y (1 - \cos\alpha) & -u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & 0 \\ u_x u_y (1 - \cos\alpha) - u_z \sin\alpha & 1 + \left(u_x^2 + u_z^2\right)(\cos\alpha - 1) & u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 0 \\ u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & -u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 1 + \left(u_x^2 + u_y^2\right)(\cos\alpha - 1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- You can use the OpenGL commands:

```
GLvoid glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
GLvoid glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
```

- Consider the rotation angle $\alpha \in \mathbb{R}$ and the unit direction vector $\mathbf{u}(u_x, u_y, u_z)$.
- Consider the 3×3 transformation matrix

$$T = I_3 + U \sin\alpha + U^2 (1 - \cos\alpha),$$

where

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{bmatrix}$$

is a skew-symmetric matrix.
- The compact homogeneous representation of transformation T is

$$M = \begin{bmatrix} 1 + \left(u_y^2 + u_z^2\right)(\cos\alpha - 1) & u_z \sin\alpha + u_x u_y (1 - \cos\alpha) & -u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & 0 \\ u_x u_y (1 - \cos\alpha) - u_z \sin\alpha & 1 + \left(u_x^2 + u_z^2\right)(\cos\alpha - 1) & u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 0 \\ u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & -u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 1 + \left(u_x^2 + u_y^2\right)(\cos\alpha - 1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- You can use the OpenGL commands:

```
GLvoid glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
GLvoid glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
```

- Consider the rotation angle $\alpha \in \mathbb{R}$ and the unit direction vector $\mathbf{u}(u_x, u_y, u_z)$.
- Consider the 3×3 transformation matrix

$$T = I_3 + U \sin\alpha + U^2 (1 - \cos\alpha),$$

where

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{bmatrix}$$

is a skew-symmetric matrix.

- The compact homogeneous representation of transformation T is

$$M = \begin{bmatrix} 1 + \left(u_y^2 + u_z^2\right)(\cos\alpha - 1) & u_z \sin\alpha + u_x u_y (1 - \cos\alpha) & -u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & 0 \\ u_x u_y (1 - \cos\alpha) - u_z \sin\alpha & 1 + \left(u_x^2 + u_z^2\right)(\cos\alpha - 1) & u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 0 \\ u_y \sin\alpha + (1 - \cos\alpha) u_x u_z & -u_x \sin\alpha + (1 - \cos\alpha) u_y u_z & 1 + \left(u_x^2 + u_y^2\right)(\cos\alpha - 1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- You can use the OpenGL commands:

```
GLvoid glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
GLvoid glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
```

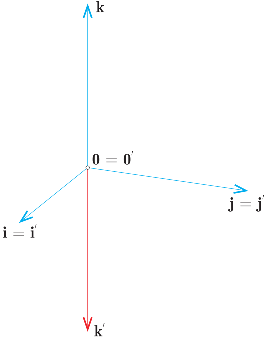

Fig. 6: Reflection.

- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
// Multiplies the current matrix with the one specified by m,
// and replaces the current matrix with the product.
glMultMatrixf(const GLfloat  *m);
glMultMatrixd(const GLdouble *m);

// example
GLfloat reflection[] = {1.0f, 0.0f,  0.0f, 0.0f,
                        0.0f, 1.0f,  0.0f, 0.0f,
                        0.0f, 0.0f, -1.0f, 0.0f,
                        0.0f, 0.0f,  0.0f, 1.0f};

...

glMultMatrixf(reflection);
```

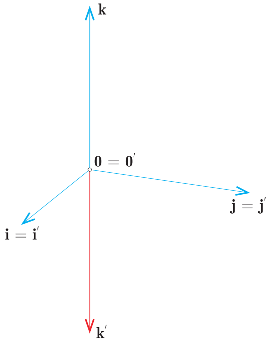

Fig. 6: Reflection.

- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
// Multiplies the current matrix with the one specified by m,
// and replaces the current matrix with the product.
glMultMatrixf(const GLfloat  *m);
glMultMatrixd(const GLdouble *m);

// example
GLfloat reflection[] = {1.0f, 0.0f,  0.0f, 0.0f,
                        0.0f, 1.0f,  0.0f, 0.0f,
                        0.0f, 0.0f, -1.0f, 0.0f,
                        0.0f, 0.0f,  0.0f, 1.0f};

...

glMultMatrixf(reflection);
```

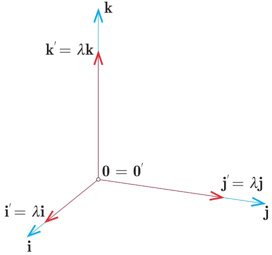

Fig. 7: Equal scaling along each axis.

- Let λ be a strictly positive real number. If $\lambda \in (0, 1)$ (resp. $\lambda > 1$), the transformation is a contraction (resp. dilation).
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ or } M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\lambda} \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  lambda_f = ...; // > 0
GLdouble lambda_d = ...; // > 0

...

glScalef(lambda_f, lambda_f, lambda_f);

...

glScaled(lambda_d, lambda_d, lambda_d);
```

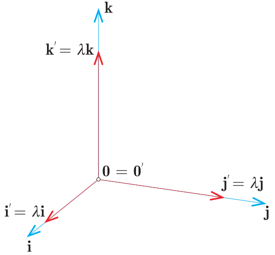

Fig. 7: Equal scaling along each axis.

- Let λ be a strictly positive real number. If $\lambda \in (0,1)$ (resp. $\lambda > 1$), the transformation is a contraction (resp. dilation).
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ or } M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\lambda} \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  lambda_f = ...; // > 0
GLdouble lambda_d = ...; // > 0

...

glScalef(lambda_f, lambda_f, lambda_f);

...

glScaled(lambda_d, lambda_d, lambda_d);
```

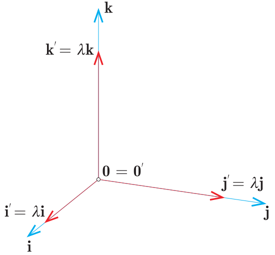

Fig. 7: Equal scaling along each axis.

- Let λ be a strictly positive real number. If $\lambda \in (0,1)$ (resp. $\lambda > 1$), the transformation is a contraction (resp. dilation).
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ or } M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\lambda} \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  lambda_f = ...; // > 0
GLdouble lambda_d = ...; // > 0

...

glScalef(lambda_f, lambda_f, lambda_f);

...

glScaled(lambda_d, lambda_d, lambda_d);
```

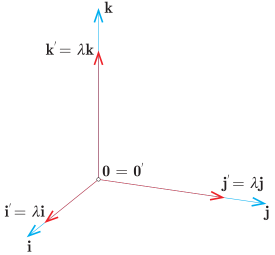

Fig. 7: Equal scaling along each axis.

- Let λ be a strictly positive real number. If $\lambda \in (0,1)$ (resp. $\lambda > 1$), the transformation is a contraction (resp. dilation).
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ or } M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\lambda} \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLfloat  lambda_f = ...; // > 0
GLdouble lambda_d = ...; // > 0

...

glScalef(lambda_f, lambda_f, lambda_f);

...

glScaled(lambda_d, lambda_d, lambda_d);
```

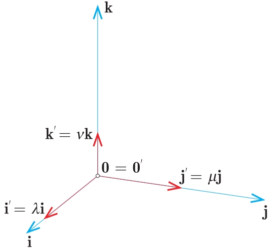

Fig. 8: Not necessarily equal scaling along each axis.

- Let λ, μ, ν be strictly positive real numbers.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \nu & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glScalef(GLfloat lambda, GLfloat mu, GLfloat nu);
GLvoid glScaled(GLdouble lambda, GLdouble mu, GLdouble nu);
```

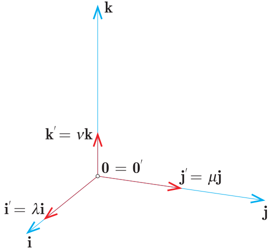

Fig. 8: Not necessarily equal scaling along each axis.

- Let λ, μ, ν be strictly positive real numbers.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \nu & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glScalef(GLfloat lambda, GLfloat mu, GLfloat nu);
GLvoid glScaled(GLdouble lambda, GLdouble mu, GLdouble nu);
```

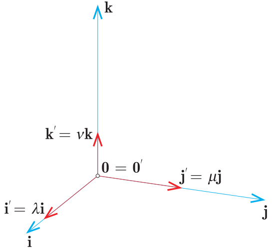

Fig. 8: Not necessarily equal scaling along each axis.

- Let λ, μ, ν be strictly positive real numbers.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \nu & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The corresponding OpenGL commands are:

```
GLvoid glScalef(GLfloat lambda, GLfloat mu, GLfloat nu);
GLvoid glScaled(GLdouble lambda, GLdouble mu, GLdouble nu);
```

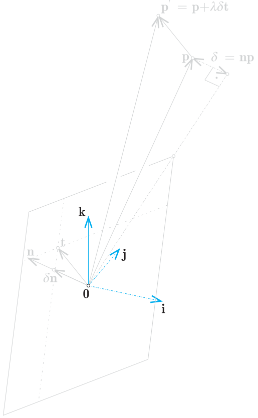

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

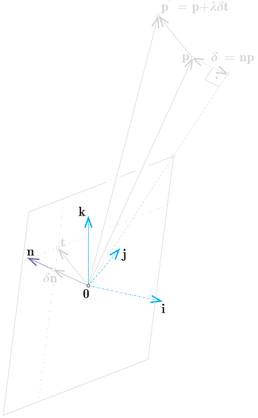

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

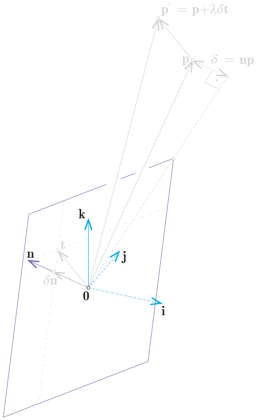

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

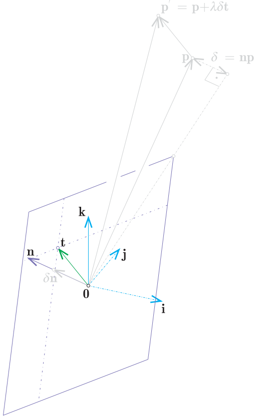

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

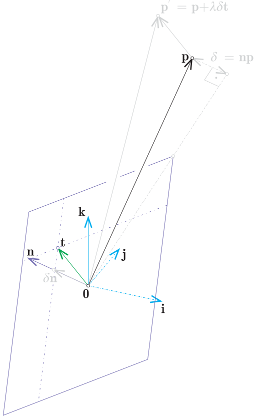

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

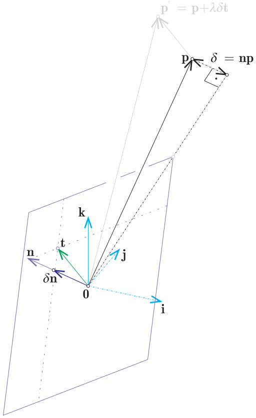

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine transformations

Shearing

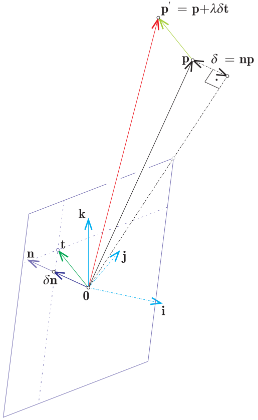

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 + \lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1 + \lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1 + \lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

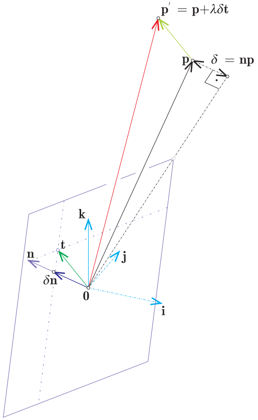

Fig. 9: Shearing.

- Consider the orthonormal, right-handed Cartesian coordinate system **0ijk**.
- Let $\mathbf{n}(n_x, n_y, n_z)$ be the unit normal vector of a hyperplane.
- Consider the unit direction vector $\mathbf{t}(t_x, t_y, t_z)$ within this hyperplane.
- Let $\mathbf{p}(x, y, z)$ be the position vector of point in space, the signed distance of which from the hyperplane is $\delta = \mathbf{n} \cdot \mathbf{p}$.
- Move the point $\mathbf{p}$ to $\mathbf{p}'$ parallel to the direction vector $\mathbf{t}$ such that the movement is proportional to the signed distance of the point $\mathbf{p}$ from the hyperplane.
- The homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Any 4×4 regular matrix M, the fourth line of which is of the form $(0, 0, 0, c)$, $c \neq 0$, represents an affine transformation.

- Any 4×4 regular matrix M represents a projective transformation.

Relation between different types of transformations

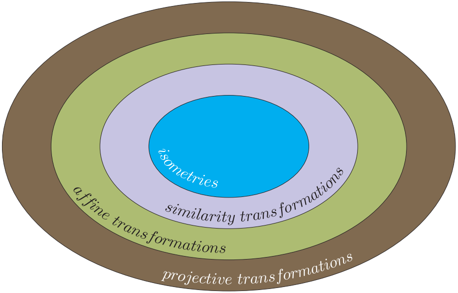

Fig. 10: Real subgroups of projective transformations are the groups of affine transformations, similarity transformations and isometries.

Order of transformations

- The effect of applying the sequence of transformation matrices $M_1, M_2, \ldots, M_n$ to a point $\mathbf{p}$ is equivalent to the effect of the single (product) transformation

$$M = M_n M_{n-1} \cdots M_1.$$

Central projection

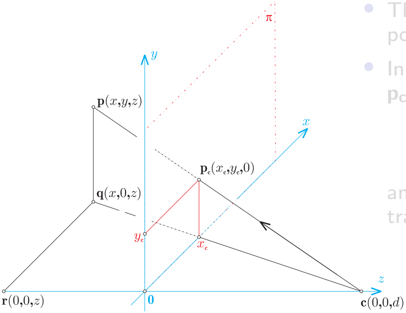

Fig. 11: Central projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- The viewer is located on the axis $\mathbf{0}z$ at the point/centrum $\mathbf{c}(0, 0, d)$.
- In this case the coordinates of the projected point $\mathbf{p_c}(x_\mathbf{c}, y_\mathbf{c}, z_\mathbf{c})$ are

$$x_\mathbf{c} = x\frac{d}{d-z},\ y_\mathbf{c} = y\frac{d}{d-z},\ z_\mathbf{c} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}.$$

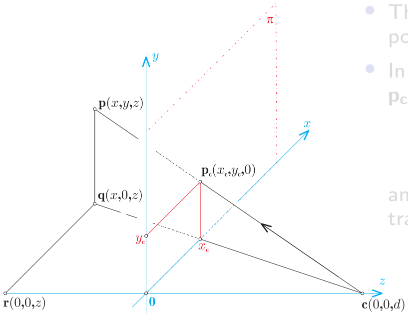

Fig. 11: Central projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- The viewer is located on the axis $\mathbf{0}z$ at the point/centrum $\mathbf{c}(0, 0, d)$.
- In this case the coordinates of the projected point $\mathbf{p_c}(x_\mathbf{c}, y_\mathbf{c}, z_\mathbf{c})$ are

$$x_\mathbf{c} = x\frac{d}{d-z},\ y_\mathbf{c} = y\frac{d}{d-z},\ z_\mathbf{c} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}.$$

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- The viewer is located on the axis $\mathbf{0}z$ at the point/centrum $\mathbf{c}(0, 0, d)$.
- In this case the coordinates of the projected point $\mathbf{p_c}(x_\mathbf{c}, y_\mathbf{c}, z_\mathbf{c})$ are

$$x_\mathbf{c} = x\frac{d}{d-z},\ y_\mathbf{c} = y\frac{d}{d-z},\ z_\mathbf{c} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}.$$

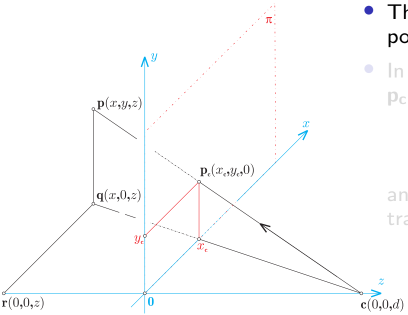

Fig. 11: Central projection.

Central projection

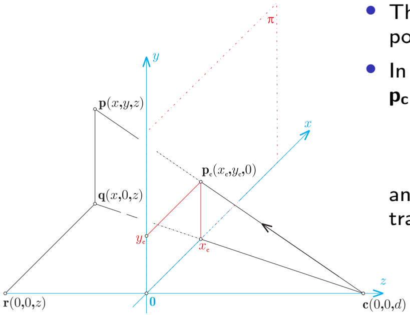

Fig. 11: Central projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- The viewer is located on the axis $\mathbf{0}z$ at the point/centrum $\mathbf{c}(0, 0, d)$.
- In this case the coordinates of the projected point $\mathbf{p_c}(x_\mathbf{c}, y_\mathbf{c}, z_\mathbf{c})$ are

$$x_\mathbf{c} = x\frac{d}{d-z},\ y_\mathbf{c} = y\frac{d}{d-z},\ z_\mathbf{c} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}.$$

Central projection

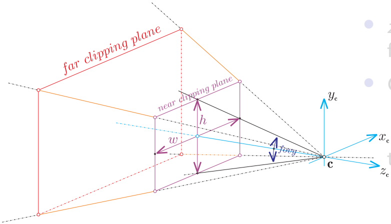

Fig. 12: Central projection using OpenGL.

- *fovy* specifies the field of view angle, in degrees, in the y direction.
- *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).
- z_{near} specifies the distance from the viewer to the near clipping plane (always strictly positive).
- z_{near} specifies the distance from the viewer to the far clipping plane (always strictly positive).
- Given

$$f = \cot \frac{fovy}{2},\ aspect = \frac{w}{h},$$

the generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2z_{far}z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Central projection

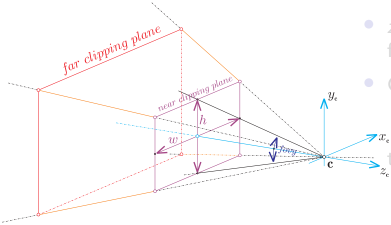

Fig. 12: Central projection using OpenGL.

- *fovy* specifies the field of view angle, in degrees, in the y direction.
- *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).
- z_{near} specifies the distance from the viewer to the near clipping plane (always strictly positive).
- z_{near} specifies the distance from the viewer to the far clipping plane (always strictly positive).
- Given

$$f = \cot\frac{fovy}{2},\ aspect = \frac{w}{h},$$

the generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2z_{far}z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Central projection

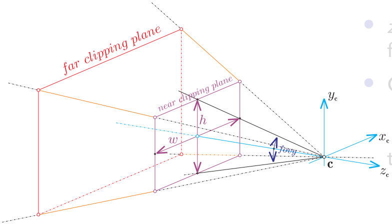

Fig. 12: Central projection using OpenGL.

- *fovy* specifies the field of view angle, in degrees, in the y direction.
- *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).
- z_{near} specifies the distance from the viewer to the near clipping plane (always strictly positive).
- z_{near} specifies the distance from the viewer to the far clipping plane (always strictly positive).
- Given

$$f = \cot\frac{fovy}{2}, \; aspect = \frac{w}{h},$$

the generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2z_{far}z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Central projection

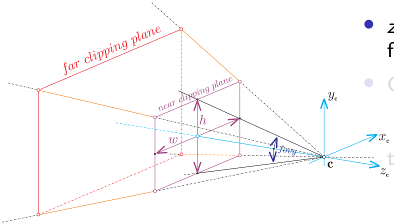

Fig. 12: Central projection using OpenGL.

- *fovy* specifies the field of view angle, in degrees, in the y direction.
- *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).
- z_{near} specifies the distance from the viewer to the near clipping plane (always strictly positive).
- z_{near} specifies the distance from the viewer to the far clipping plane (always strictly positive).
- Given

$$f = \cot \frac{fovy}{2}, \; aspect = \frac{w}{h},$$

the generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2 z_{far} z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Central projection

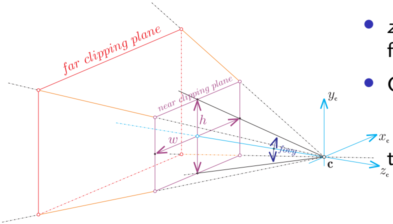

Fig. 12: Central projection using OpenGL.

- *fovy* specifies the field of view angle, in degrees, in the y direction.
- *aspect* specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).
- z_{near} specifies the distance from the viewer to the near clipping plane (always strictly positive).
- z_{near} specifies the distance from the viewer to the far clipping plane (always strictly positive).
- Given

$$f = \cot\frac{fovy}{2},\ aspect = \frac{w}{h},$$

the generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2z_{far}z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Example

```
// creating the central projection matrix
GLdouble fovy = ..., aspect = ...,  z_near = ... , z_far = ...;

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, z_near, z_far);

// creating the model view matrix
GLdouble    d           = ...;
GLdouble    eye[3]      = {0.0, 0.0, d},
            center[3]   = {0.0, 0.0, 0.0},
            up[3]       = {0.0, 1.0, 0.0};

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2], up[0], up[1], up[2]);
```

Parallel projection

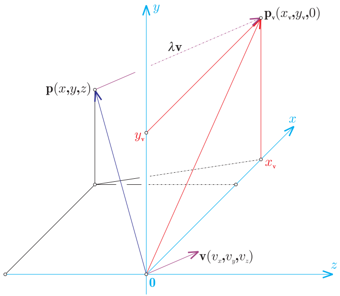

Fig. 13: Parallel projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- Consider the common direction vector $\mathbf{v}(v_x, v_y, v_z)$ of the parallel projecting rays. Assume that $\mathbf{v} \nparallel \pi$.
- In this case the coordinates of the projected point $\mathbf{p_v}(x_\mathbf{v}, y_\mathbf{v}, z_\mathbf{v})$ are

$$x_\mathbf{v} = x - \frac{v_x}{v_z} z,\; y_\mathbf{v} = y - \frac{v_y}{v_z} z,\; z_\mathbf{v} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Special case: orthogonal projection ($\mathbf{v} \perp \pi$, i.e. $v_x = v_y = 0$).

Parallel projection

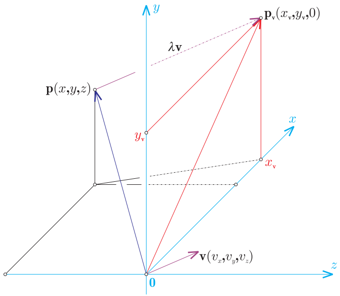

Fig. 13: Parallel projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- Consider the common direction vector $\mathbf{v}(v_x, v_y, v_z)$ of the parallel projecting rays. Assume that $\mathbf{v} \nparallel \pi$.
- In this case the coordinates of the projected point $\mathbf{p}_{\mathbf{v}}(x_{\mathbf{v}}, y_{\mathbf{v}}, z_{\mathbf{v}})$ are

$$x_{\mathbf{v}} = x - \frac{v_x}{v_z}z,\ y_{\mathbf{v}} = y - \frac{v_y}{v_z}z,\ z_{\mathbf{v}} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Special case: orthogonal projection ($\mathbf{v} \perp \pi$, i.e. $v_x = v_y = 0$).

Parallel projection

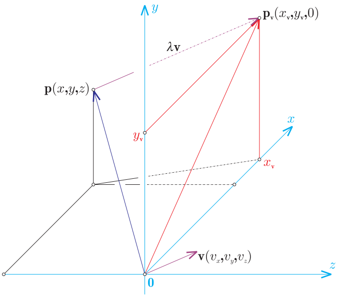

Fig. 13: Parallel projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- Consider the common direction vector $\mathbf{v}(v_x, v_y, v_z)$ of the parallel projecting rays. Assume that $\mathbf{v} \nparallel \pi$.
- In this case the coordinates of the projected point $\mathbf{p}_{\mathbf{v}}(x_{\mathbf{v}}, y_{\mathbf{v}}, z_{\mathbf{v}})$ are

$$x_{\mathbf{v}} = x - \frac{v_x}{v_z}z,\ y_{\mathbf{v}} = y - \frac{v_y}{v_z}z,\ z_{\mathbf{v}} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Special case: orthogonal projection ($\mathbf{v} \perp \pi$, i.e. $v_x = v_y = 0$).

Parallel projection

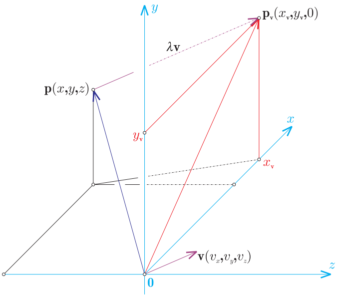

Fig. 13: Parallel projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- Consider the common direction vector $\mathbf{v}(v_x, v_y, v_z)$ of the parallel projecting rays. Assume that $\mathbf{v} \nparallel \pi$.
- In this case the coordinates of the projected point $\mathbf{p}_{\mathbf{v}}(x_{\mathbf{v}}, y_{\mathbf{v}}, z_{\mathbf{v}})$ are

$$x_{\mathbf{v}} = x - \frac{v_x}{v_z}z,\ y_{\mathbf{v}} = y - \frac{v_y}{v_z}z,\ z_{\mathbf{v}} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Special case: orthogonal projection ($\mathbf{v} \perp \pi$, i.e. $v_x = v_y = 0$).

Parallel projection

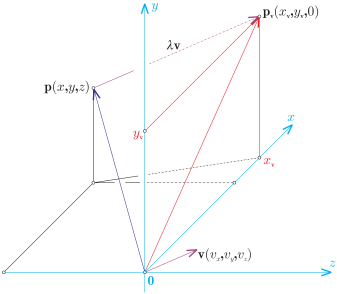

Fig. 13: Parallel projection.

- Plane $\mathbf{0}xy$ is considered as the image plane π.
- The points $\mathbf{p}(x, y, z)$ of the object that is projected onto the plane π lie in the negative half-space of π.
- Consider the common direction vector $\mathbf{v}(v_x, v_y, v_z)$ of the parallel projecting rays. Assume that $\mathbf{v} \nparallel \pi$.
- In this case the coordinates of the projected point $\mathbf{p_v}(x_\mathbf{v}, y_\mathbf{v}, z_\mathbf{v})$ are

$$x_\mathbf{v} = x - \frac{v_x}{v_z} z,\ y_\mathbf{v} = y - \frac{v_y}{v_z} z,\ z_\mathbf{v} = 0,$$

and the homogeneous representation of the transformation is

$$M = \begin{bmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Special case: orthogonal projection ($\mathbf{v} \perp \pi$, i.e. $v_x = v_y = 0$).

Orthogonal projection

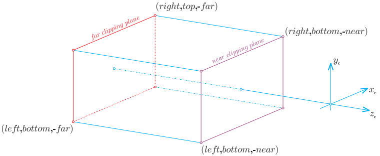

Fig. 14: Orthogonal projection using OpenGL.

- *left*, *right* specify the coordinates for the left and right vertical clipping planes.
- *bottom*, *top* specify the coordinates for the bottom and top horizontal clipping planes.
- *near*, *far* specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.
- The generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{2}{far-near} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where

$$t_x = -\frac{right+left}{right-left},\ t_y = -\frac{top+bottom}{top-bottom},$$

$$t_z = -\frac{far+near}{far-near}.$$

Orthogonal projection

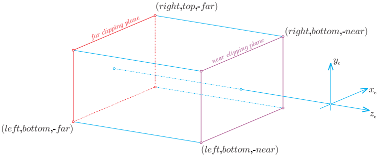

Fig. 14: Orthogonal projection using OpenGL.

- *left*, *right* specify the coordinates for the left and right vertical clipping planes.
- *bottom*, *top* specify the coordinates for the bottom and top horizontal clipping planes.
- *near*, *far* specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.
- The generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{2}{far-near} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where

$$t_x = -\frac{right+left}{right-left},\ t_y = -\frac{top+bottom}{top-bottom},$$

$$t_z = -\frac{far+near}{far-near}.$$

Orthogonal projection

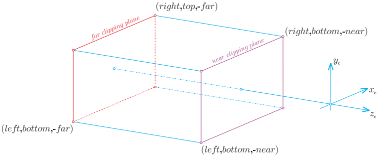

Fig. 14: Orthogonal projection using OpenGL.

- *left*, *right* specify the coordinates for the left and right vertical clipping planes.
- *bottom*, *top* specify the coordinates for the bottom and top horizontal clipping planes.
- *near*, *far* specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.
- The generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{2}{far-near} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where

$$t_x = -\frac{right+left}{right-left},\ t_y = -\frac{top+bottom}{top-bottom},$$

$$t_z = -\frac{far+near}{far-near}.$$

Orthogonal projection

- *left*, *right* specify the coordinates for the left and right vertical clipping planes.
- *bottom*, *top* specify the coordinates for the bottom and top horizontal clipping planes.
- *near*, *far* specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.
- The generated trasnformation matrix is

$$M = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & t_x \\ 0 & \frac{2}{top - bottom} & 0 & t_y \\ 0 & 0 & \frac{2}{far - near} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where

$$t_x = -\frac{right + left}{right - left},\ t_y = -\frac{top + bottom}{top - bottom},$$

$$t_z = -\frac{far + near}{far - near}.$$

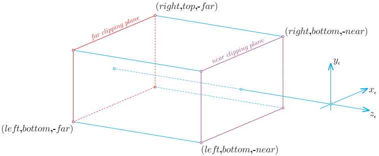

Fig. 14: Orthogonal projection using OpenGL.

Example

```
// creating the orthogonal projection matrix
GLdouble left = ...,     right = ...,
         bottom = ...,   top = ...,
         near = ...,     far = ...;

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bottom, top, near, far);

...
```