Transformations in Graphics

Hendrik Speleers

Transformations in Graphics

Università di Roma Tor Vergata

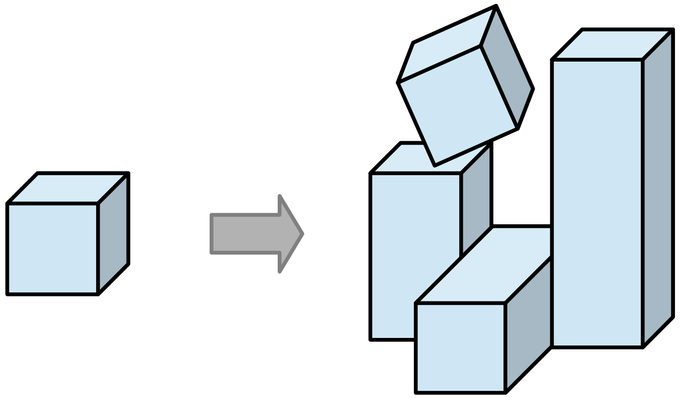
- Overview
  - Homogeneous coordinates
  - Affine transformations
    - 2D and 3D
    - Changing coordinate systems
  - Viewing in 3D
    - Camera setup
    - Perspective projection
    - Canonical view volume: 3D clipping

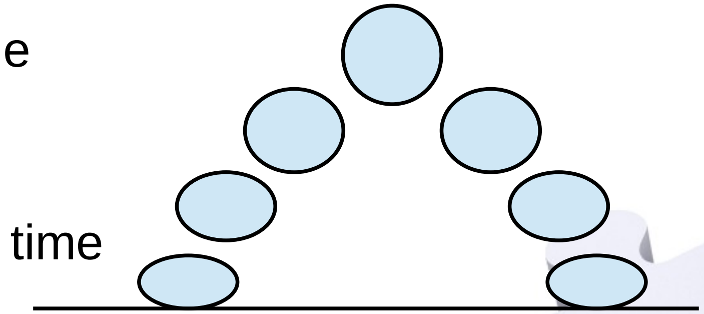
Transformations in Graphics

- Coordinate systems
  - Homogeneous coordinates
    - Key concept in computer graphics
    - Why? Points and vectors can now be mixed in operations
  - Points: (x, y, z, 1)
  - Vectors: (x, y, z, 0)
  - Some operations
    - Subtraction: (*, *, *, 1) − (*, *, *, 1) = (*, *, *, 0)
    - Addition: (*, *, *, 1) + (*, *, *, 0) = (*, *, *, 1)
    - Affine linear combinations of points produce another point

Transformations in Graphics

- Transformations
  - Translations, rotations, scaling, …
- Why are transformations useful?
  - Constructing complex objects
    - They are usually composed of simple objects
  - Moving camera around
    - Different views on the same scene
  - Computer animation
    - Translate/rotate/warp object over time





Transformations in Graphics

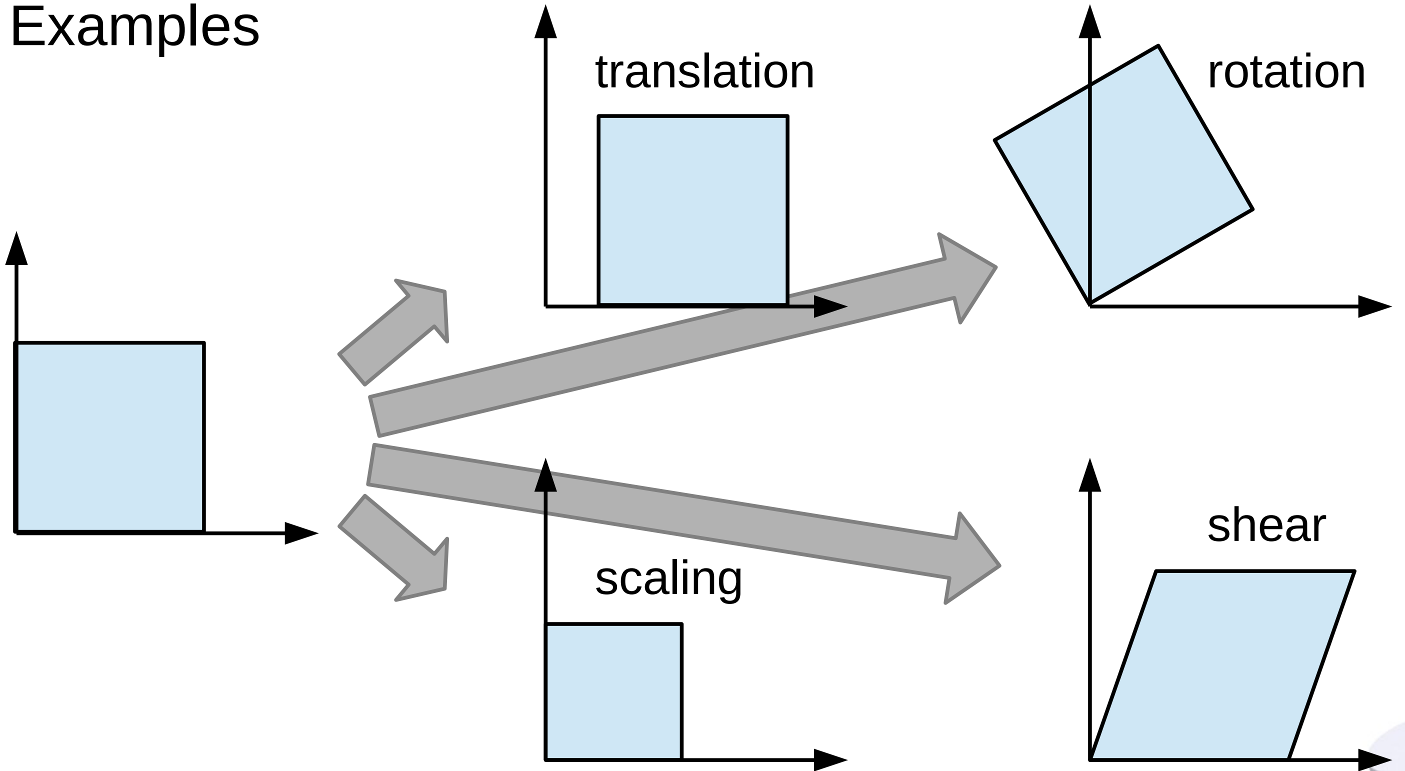
- 2D affine transformations
  - Coordinates of Q are linear combination of coordinates of P

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = MP$$

  - Properties
    - Preservation of affine linear combinations
    - Preservation of lines
    - Preservation of parallelism of lines
    - Preservation of relative ratios
    - Areas are scaled with $|\det(M)|$

Transformations in Graphics

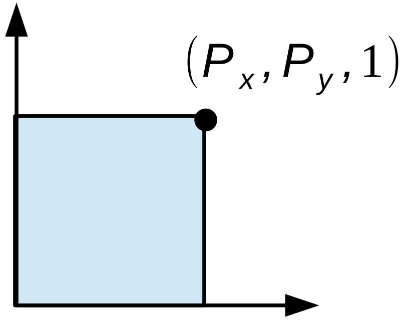
- 2D affine transformations
  - Examples

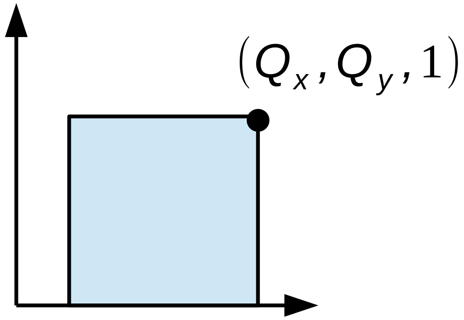


Transformations in Graphics

Università di Roma
Tor Vergata

- 2D affine transformations
  - Translation





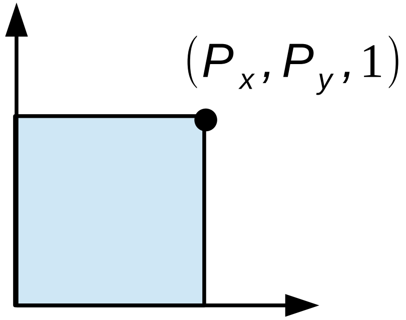
$$Q_x = P_x + T_x$$
$$Q_y = P_y + T_y$$

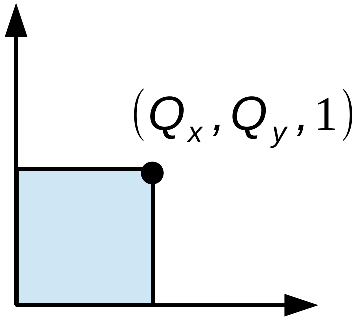
$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = TP$$

Transformations in Graphics

Università di Roma Tor Vergata

- 2D affine transformations
  - Scaling





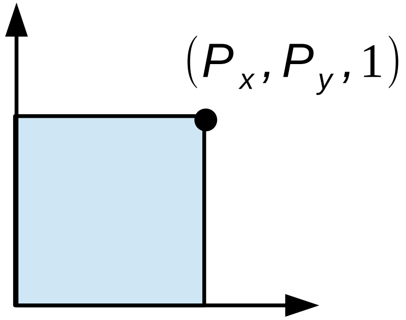
$$Q_x = S_x P_x$$
$$Q_y = S_y P_y$$

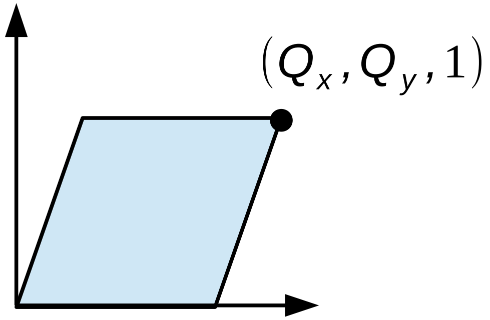
$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = SP$$

Transformations in Graphics

Università di Roma Tor Vergata

- 2D affine transformations
  - Shear



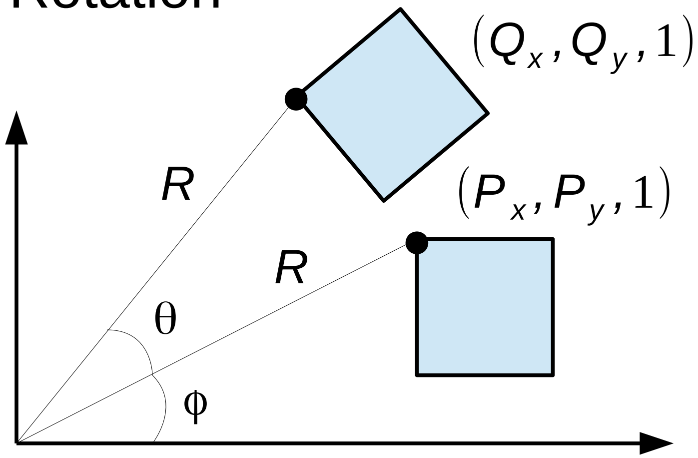


$$Q_x = P_x + hP_y$$
$$Q_y = P_y$$

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = S_h P$$

Transformations in Graphics

- 2D affine transformations
  - Rotation



$$P_x = R\cos\phi$$
$$P_y = R\sin\phi$$

$$Q_x = R\cos(\phi+\theta)$$
$$Q_y = R\sin(\phi+\theta)$$

$$\cos(\phi+\theta) = \cos\phi\cos\theta - \sin\phi\sin\theta$$
$$\sin(\phi+\theta) = \sin\phi\cos\theta + \cos\phi\sin\theta$$

$$Q = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = RP$$

Transformations in Graphics

Università di Roma Tor Vergata

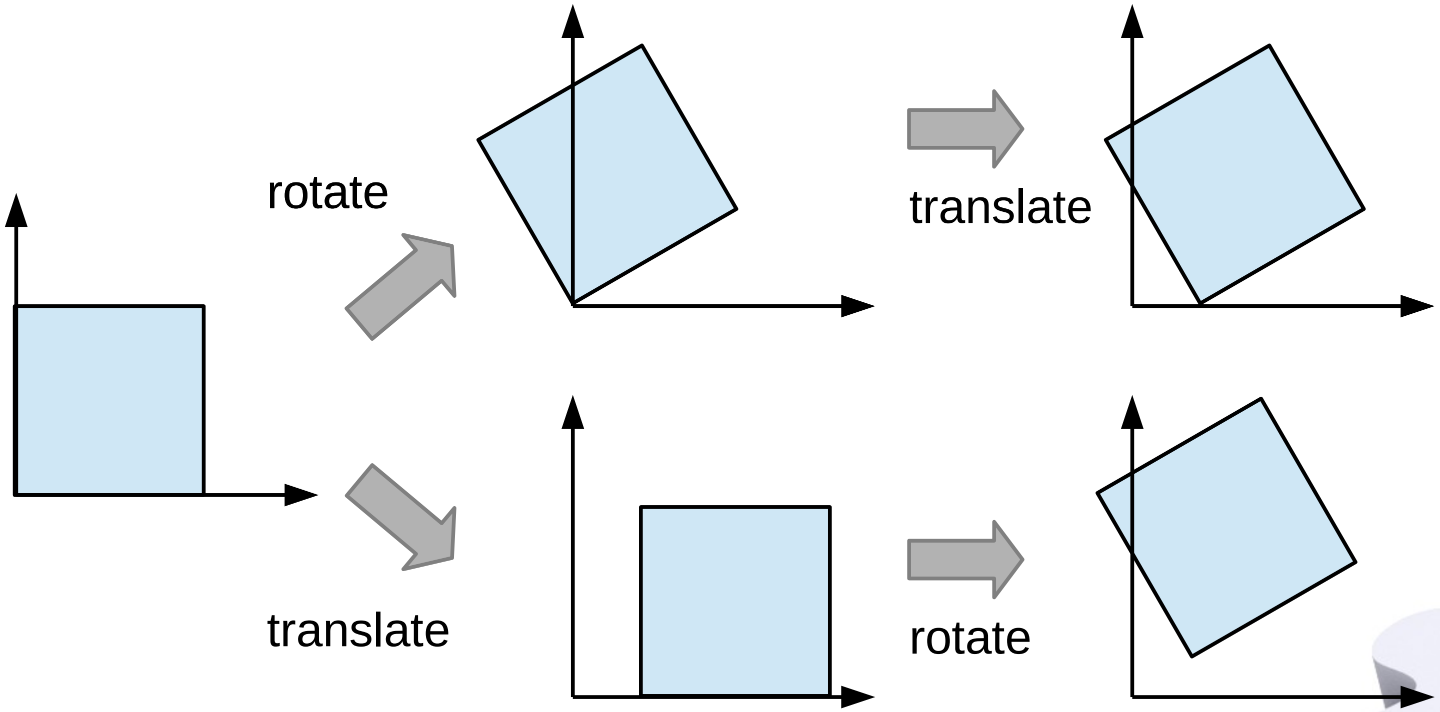
- 2D affine transformations
  - Undo transformation by inverting matrix

$$T^{-1}=\begin{pmatrix}1 & 0 & -T_x\\ 0 & 1 & -T_y\\ 0 & 0 & 1\end{pmatrix}\quad S^{-1}=\begin{pmatrix}1/S_x & 0 & 0\\ 0 & 1/S_y & 0\\ 0 & 0 & 1\end{pmatrix}\quad S_h^{-1}=\begin{pmatrix}1 & -h & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\quad R^{-1}=\begin{pmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}$$

  - Composite transformations
    - Window-to-viewport transform: scaling + translation
    - Example: Rotation around a point: $Q=(T^{-1}RT)P$
      - Translate rotation center to origin (T)
      - Rotate around origin (R)
      - Translate origin back to rotation center (T^{-1})

Transformations in Graphics

- 2D affine transformations
  - Composite transformations: Order is important!!!



Transformations in Graphics

Università di Roma Tor Vergata

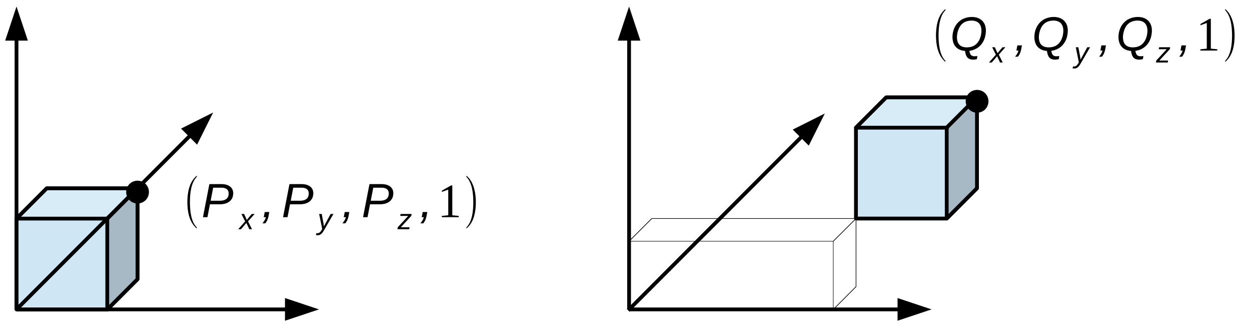
- 3D affine transformations
  - Same idea as 2D, but now 4x4 matrices

$$Q=\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix}=\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}=MP$$

  - Properties
    - Preservation of affine linear combinations
    - Preservation of lines and planes
    - Preservation of parallelism of lines and planes
    - Preservation of relative ratios
    - Volumes are scaled with $|\det(M)|$

Transformations in Graphics

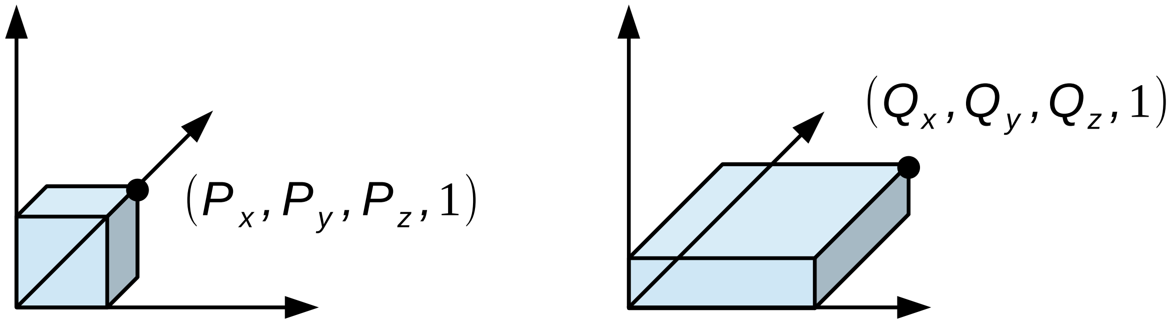
- 3D affine transformations
  - Translation



$$Q = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = TP$$

Transformations in Graphics

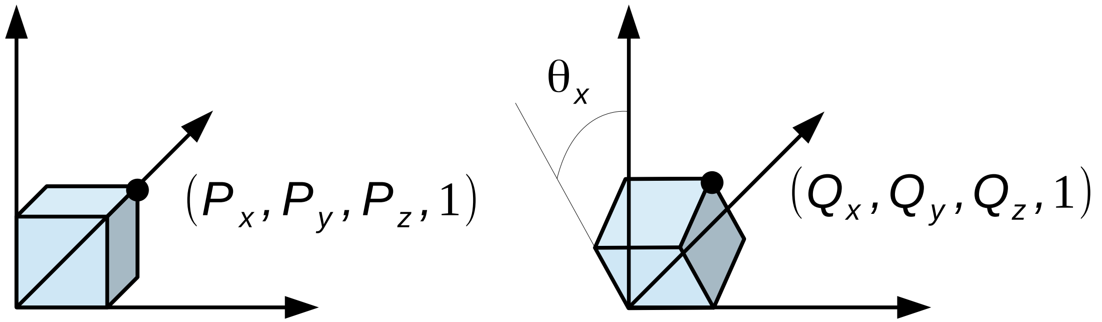
- 3D affine transformations
  - Scaling



$$Q = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = SP$$

Transformations in Graphics

- 3D affine transformations
  - Rotation around *X*-axis (similar for other axes)



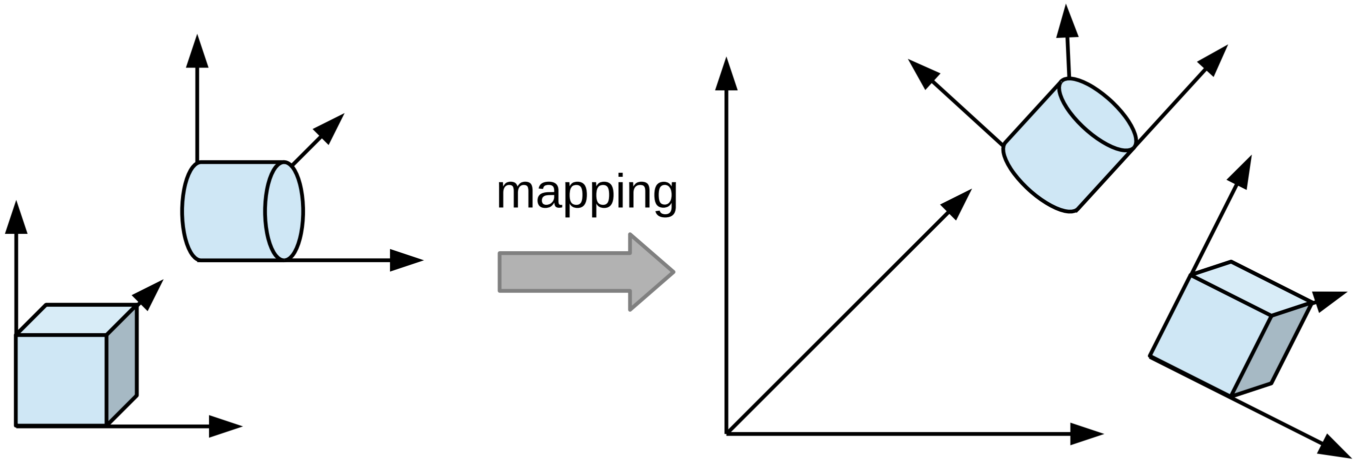
$$Q=\begin{pmatrix}Q_x\\Q_y\\Q_z\\1\end{pmatrix}=\begin{pmatrix}1 & 0 & 0 & 0\\0 & \cos\theta_x & -\sin\theta_x & 0\\0 & \sin\theta_x & \cos\theta_x & 0\\0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}P_x\\P_y\\P_z\\1\end{pmatrix}=R_x P$$

Transformations in Graphics

- 3D affine transformations
  - Composite transformations
    - Same ideas as 2D
    - Example: Rotation around arbitrary axis *U*: $Q=\left(R_y^{-1}R_z^{-1}R_xR_zR_y\right)P$
      - 2 rotations such that *U* is aligned with *X*-axis
      - *X*-rotation over desired angle
      - Undo the 2 rotations to restore *U* to the original direction
  - Columns in matrix reveal transformed coordinate frame
    - First 3 columns: mapped X/Y/Z-axes
    - Last column: mapped origin

Transformations in Graphics

- Changing coordinate systems
  - Most natural approach
    - Objects are modeled in their own coordinate system
    - Compute coordinates of transformed object in world coordinate system



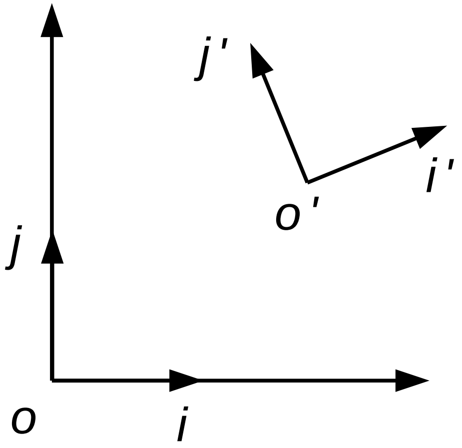
Object coordinate systems World coordinate system

Transformations in Graphics

Università di Roma Tor Vergata

- Changing coordinate systems
  - Global vs. local coordinate system
    - $o = (0, 0, 1)$; unit vectors $i = (1, 0, 0)$, $j = (0, 1, 0)$
    - $o' = (m_{13}, m_{23}, 1)$; unit vectors $i' = (m_{11}, m_{21}, 0)$, $j' = (m_{12}, m_{22}, 0)$
    - Transformation matrix M

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix}$$



Transformations in Graphics

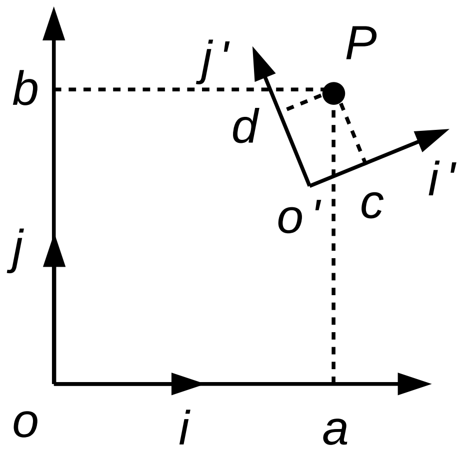
Università di Roma Tor Vergata

- Changing coordinate systems
  - Transformation matrix *M*
    - Transforms $\langle o, i, j \rangle$ into $\langle o', i', j' \rangle$

      $o' = M o \qquad i' = M i \qquad j' = M j$

    - Transforms local coordinates of *P* into global coordinates of *P*

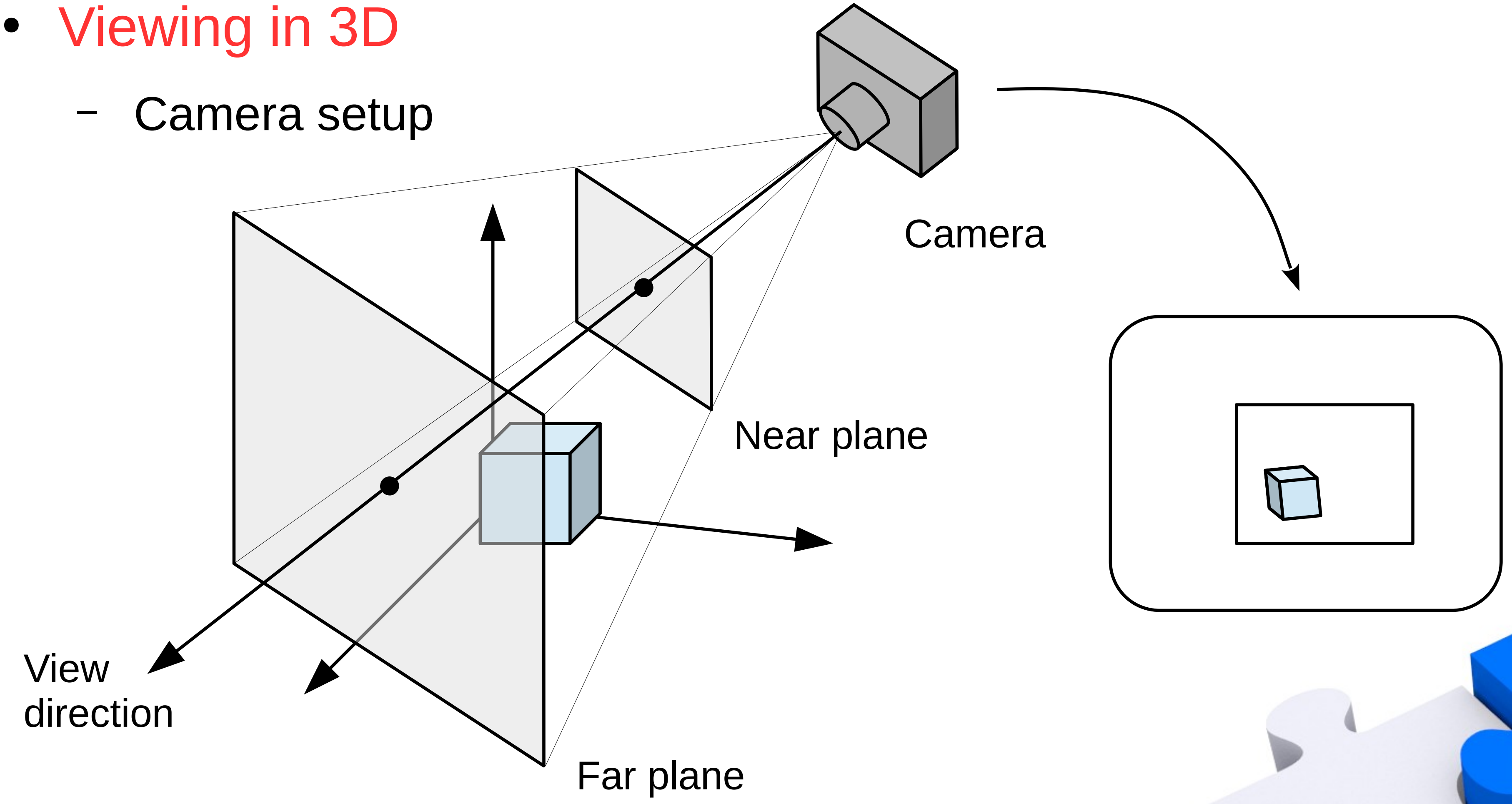
      $\begin{pmatrix} a \\ b \\ 1 \end{pmatrix} = M \begin{pmatrix} c \\ d \\ 1 \end{pmatrix}$ Modeling Transformation



Transformations in Graphics

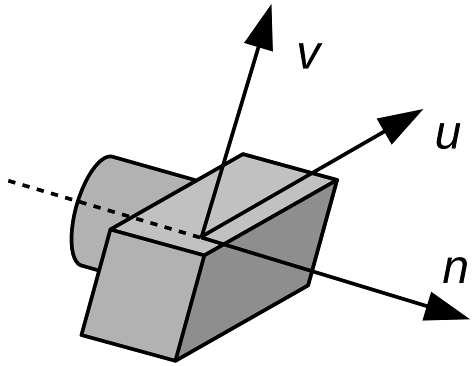
Università di Roma Tor Vergata

- Viewing in 3D
  - Camera setup



Transformations in Graphics

- Viewing in 3D
  - Camera definition: any position and any orientation (6 dof)
  - Attach coordinate system to camera
    - Origin (= eye): position of camera
    - *U*-axis: points ‘rightwards’
    - *V*-axis: points ‘upwards’
    - *N*-axis: opposite viewing direction



  - Angles of orientation of this system are called:
    - Pitch: around *U*-axis (nose up or down)
    - Yaw: around *V*-axis (nose left or right)
    - Roll: around *N*-axis

Transformations in Graphics

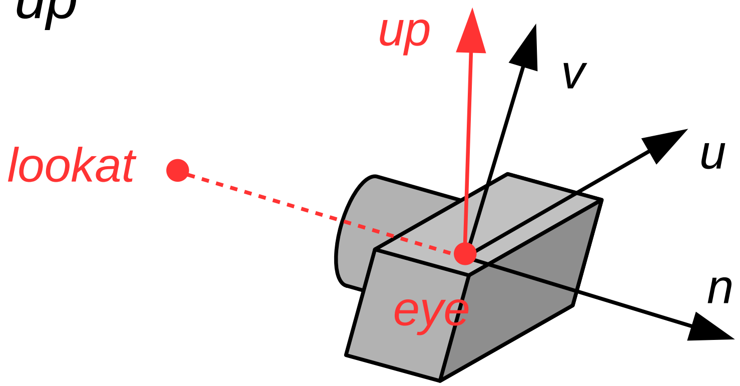
Università di Roma Tor Vergata

- Viewing in 3D
  - Suppose we have *eye*, *lookat*, and *up*

$$n = \frac{eye - lookat}{\|eye - lookat\|}$$

$$u = up \times n$$

$$v = n \times u$$



  - Change coordinates to camera system
    - From world system to camera system: matrix *V*
    - From object system to world system: matrix *M*
    - So... objects are expressed by

$Q = V\,M\,P$ Viewing + Modeling Transformation

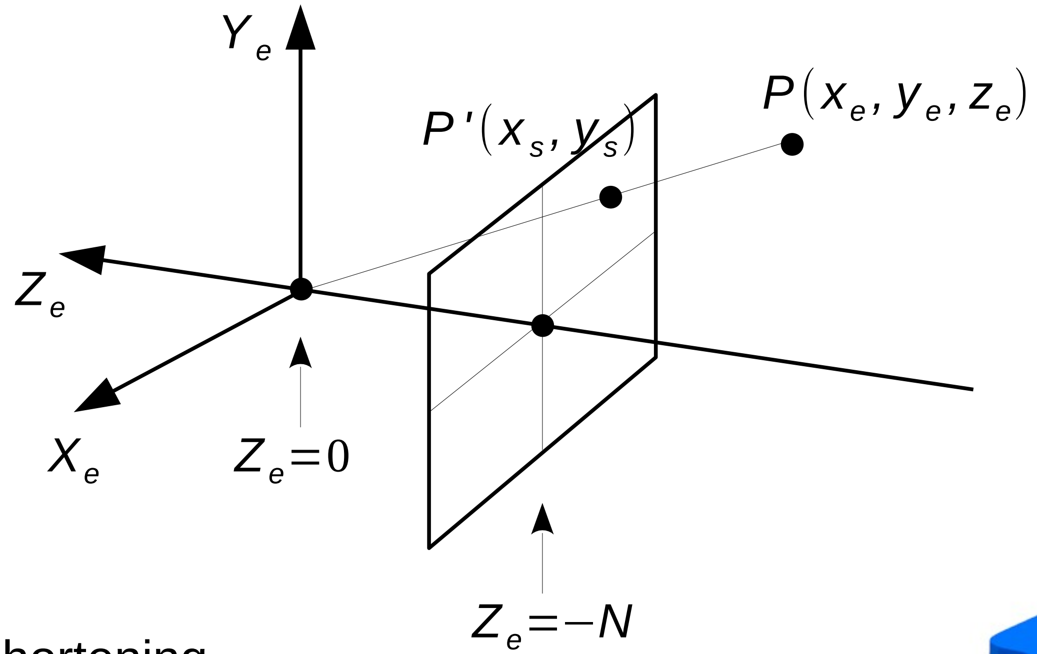
Transformations in Graphics

- Viewing in 3D
  - All objects are now expressed in camera system
  - What’s left to do?
    - Perspective projection
    - 3D clipping
      - Cut everything outside view pyramid
    - Depth
      - Needed for removal of hidden points

Transformations in Graphics

- Viewing in 3D
  - Perspective projection
    - Project 3D point on 2D plane

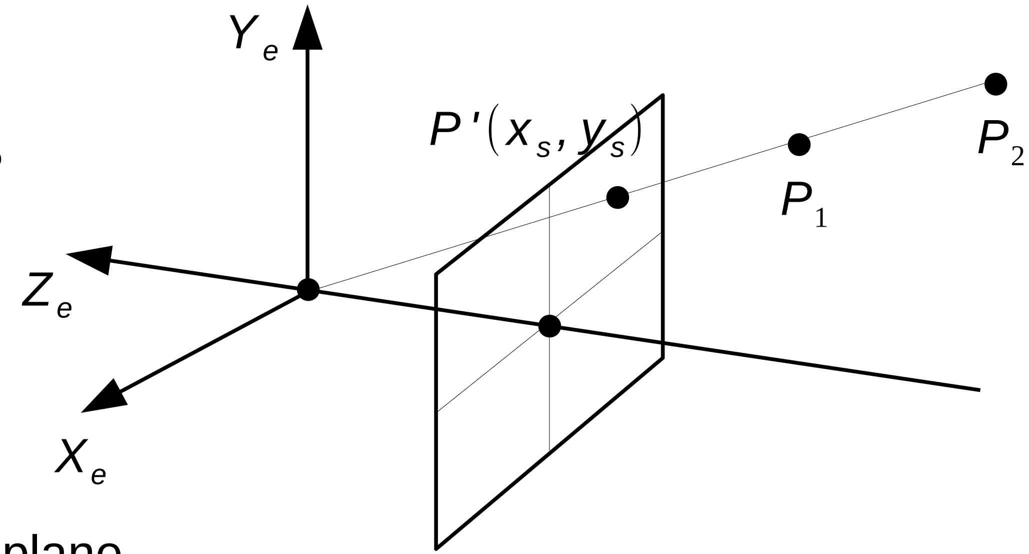
$$x_s = \frac{N}{-z_e} x_e \qquad y_s = \frac{N}{-z_e} y_e$$



  - Properties:
    - Division by z_e: perspective foreshortening
    - Effect of N: scaling of the picture
    - Straight lines project to straight lines

Transformations in Graphics

- Viewing in 3D
  - Adding depth
    - Which point is closer: P_1 or P_2 ?

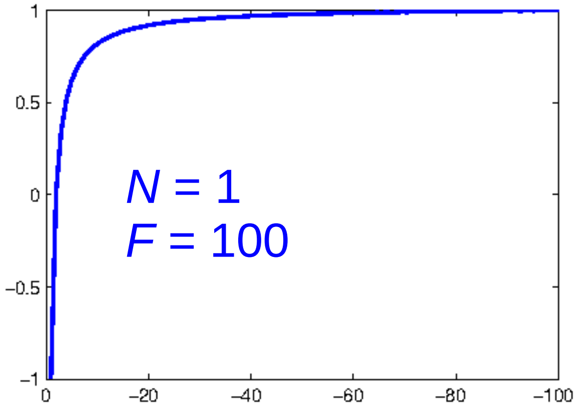


    - Maintain a depth function
      - Same denominator z_e
      - Pseudo-depth = −1 at near plane
      - Pseudo-depth = +1 at far plane

$$z_s = \frac{a z_e + b}{-z_e} \qquad a = \frac{-(F+N)}{F-N} \qquad b = \frac{-2FN}{F-N}$$

Transformations in Graphics

- Viewing in 3D
  - Hidden surfaces: *Z*-buffer
    - During rasterizing
    - Interpolate pseudo-depth between vertices
    - Store depth of pixel in *Z*-buffer
    - If new depth < old depth: recolor pixel
  - Artefacts with *Z*-buffer
    - Pixel-precision (one value per pixel)
    - Pseudo-depth interpolated, not real depth!

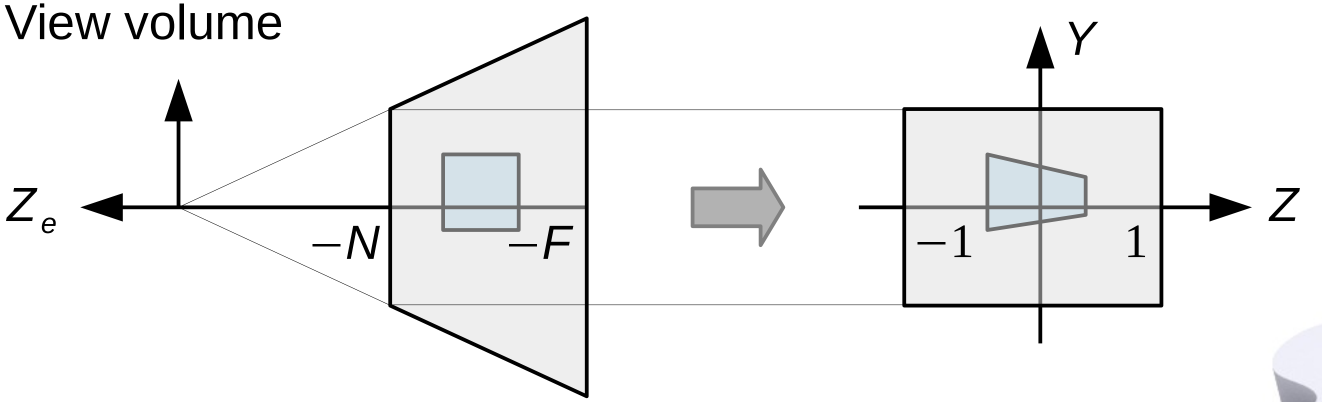


Transformations in Graphics

- Viewing in 3D
  - Perspective transform
    - Projection + depth testing: transformation matrix?

$$x_s = \frac{N}{-z_e} x_e \qquad y_s = \frac{N}{-z_e} y_e \qquad z_s = \frac{a z_e + b}{-z_e}$$

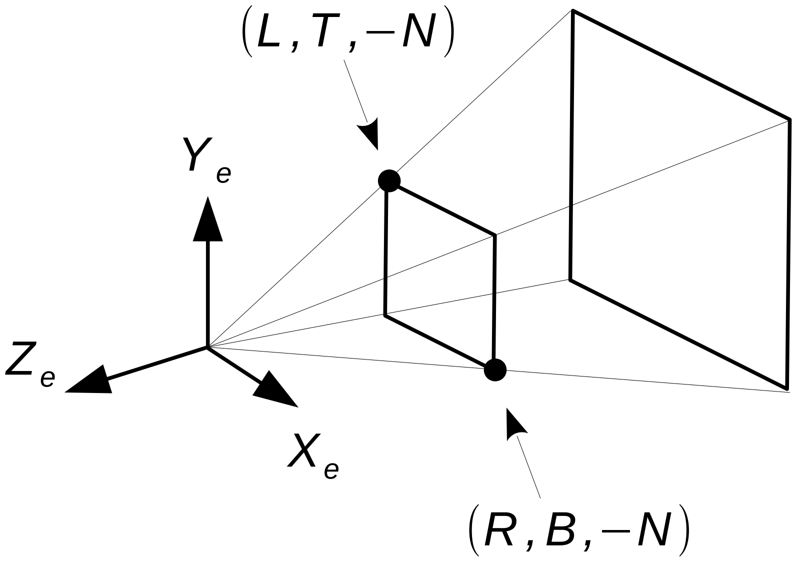
    - View volume



Transformations in Graphics

- Viewing in 3D
  - Perspective transform

$$\begin{pmatrix} \frac{2N}{R-L} & 0 & \frac{R+L}{R-L} & 0 \\ 0 & \frac{2N}{T-B} & \frac{T+B}{T-B} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

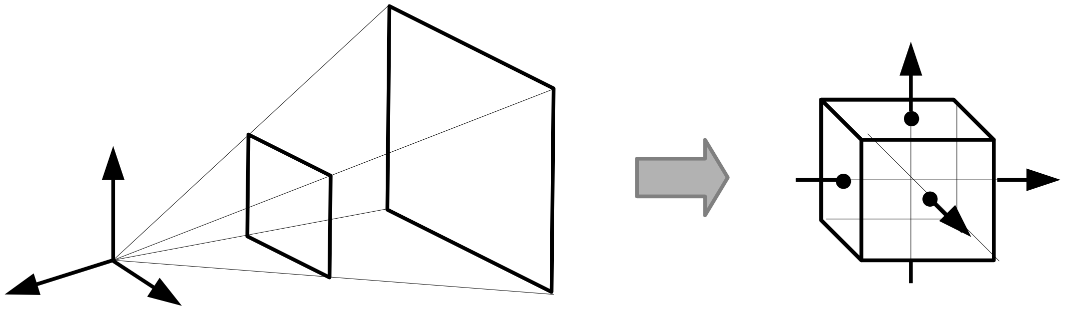


- From view pyramid to unit box [−1, 1] x [−1, 1] x [−1, 1]
  - Perspective + additional scaling and translation
- Homogeneous coords have 4th value != 1 (Division by $-z_e$ required)

Transformations in Graphics

Università di Roma Tor Vergata

- Viewing in 3D
  - Canonical view volume (CVV)
    - We have transformed everything into a unit box



  - 3D clipping
    - Four sides of view pyramid ($x = -1, 1$ and $y = -1, 1$)
    - Near and far planes ($z = -1, 1$)
    - Clipping against CVV is very efficient

Transformations in Graphics

- Viewing in 3D
  - Putting it all together
    - Every point is transformed by the modeling transformation
    - … then the viewing transformation
    - … then the perspective transformation
    - … then clip against the CVV
    - … then keep the 2D perspective coordinates
    - … then do the window-to-viewport transformation
  - This can all be specified in OpenGL!