Visual Rendering: Vertex Transforms

CS 6384 Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

Computer Graphics and Computer Vision

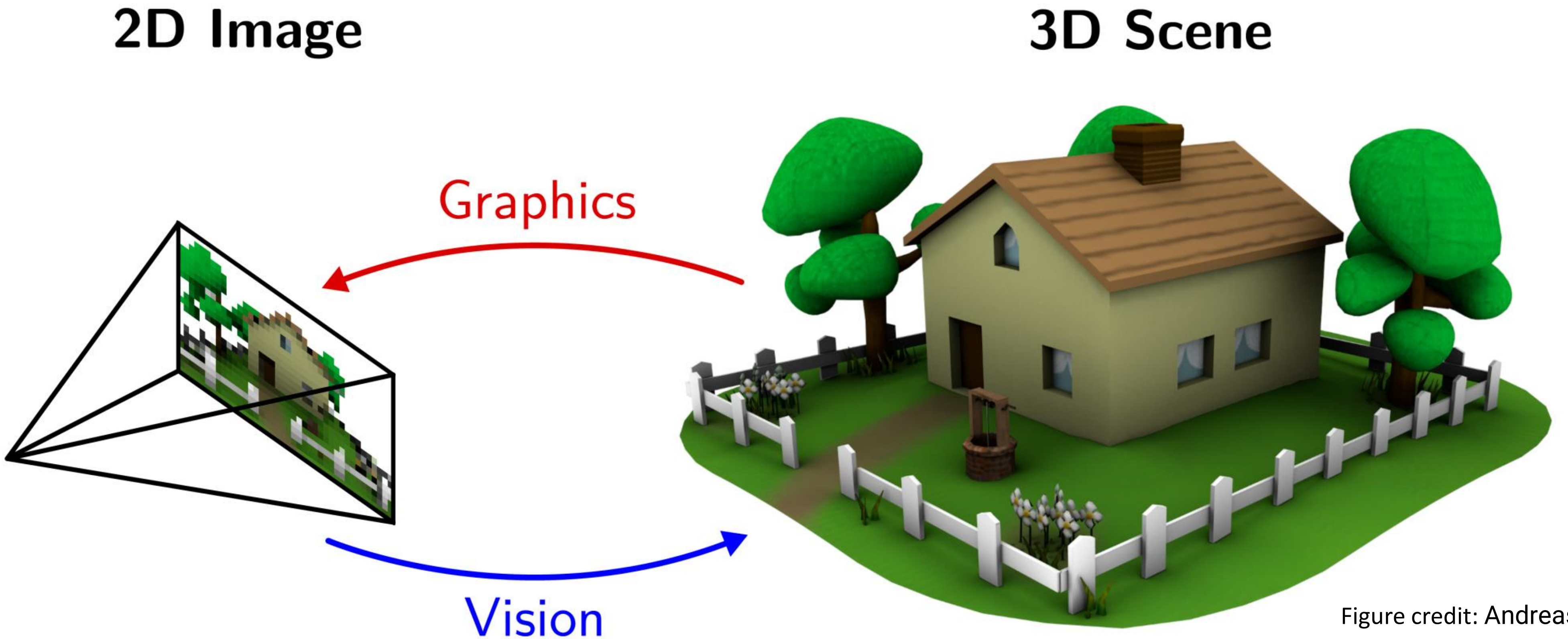
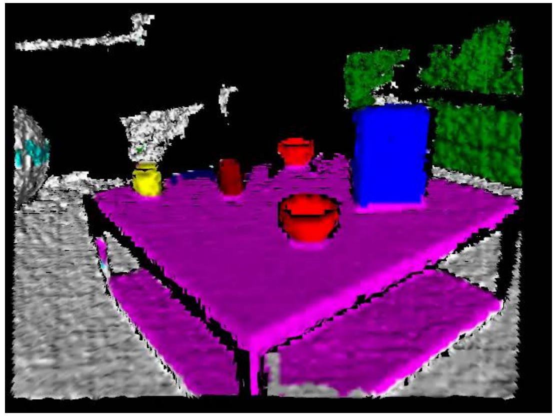


Figure credit: Andreas Geiger

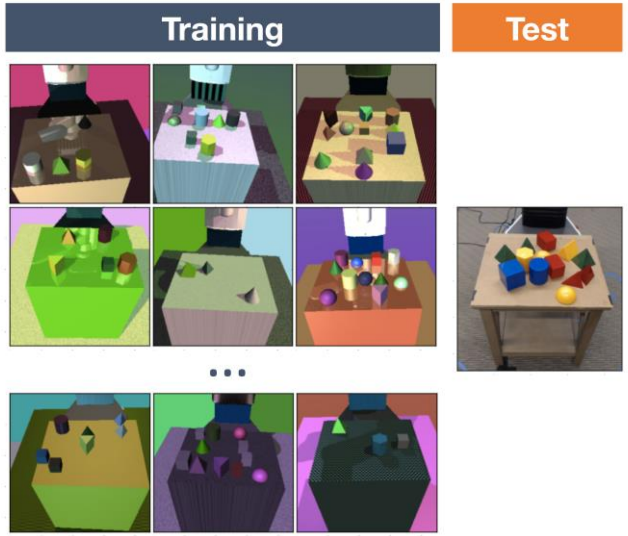
Visual Rendering

- 3D reconstruction

KinectFusion
Newcombe et al. 2011

- Synthetic data for training

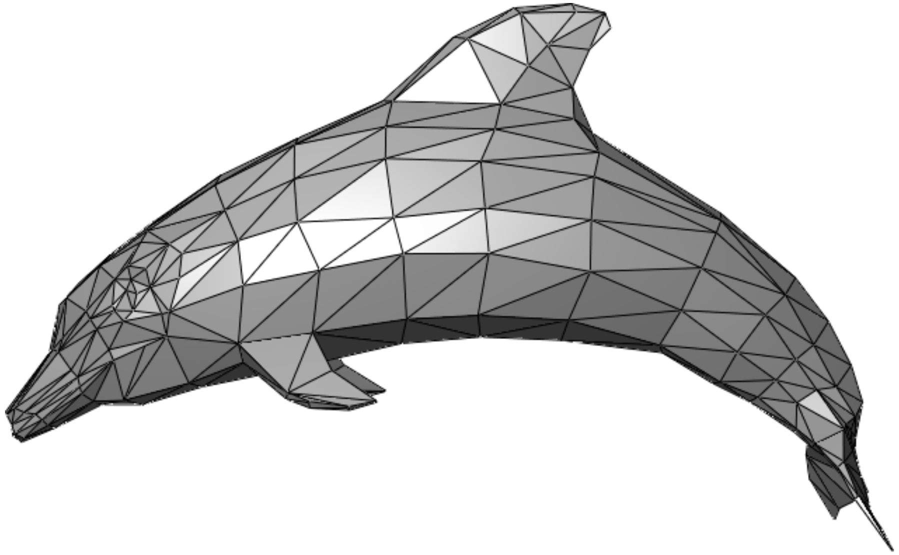


Domain Randomization
Tobin et al., 2017

- Interactive environments

iGibson
Xia et al. 2021

3D Triangle Meshes



Face-Vertex Meshes

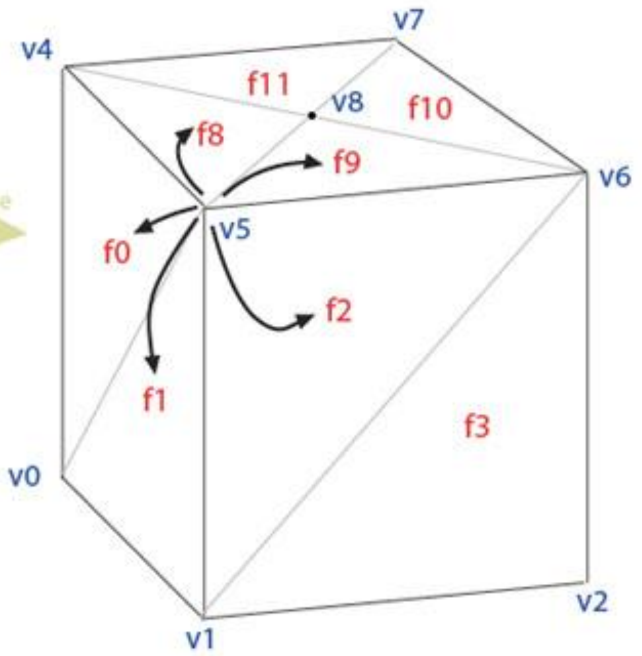
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

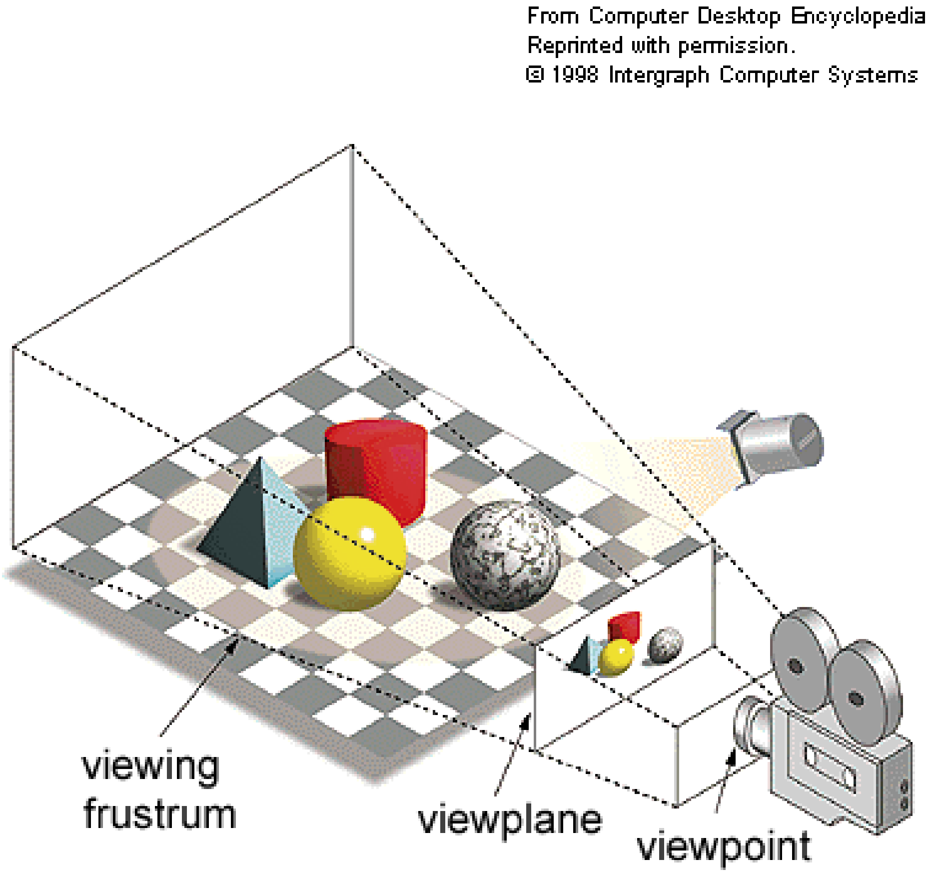
example



From Wikipedia

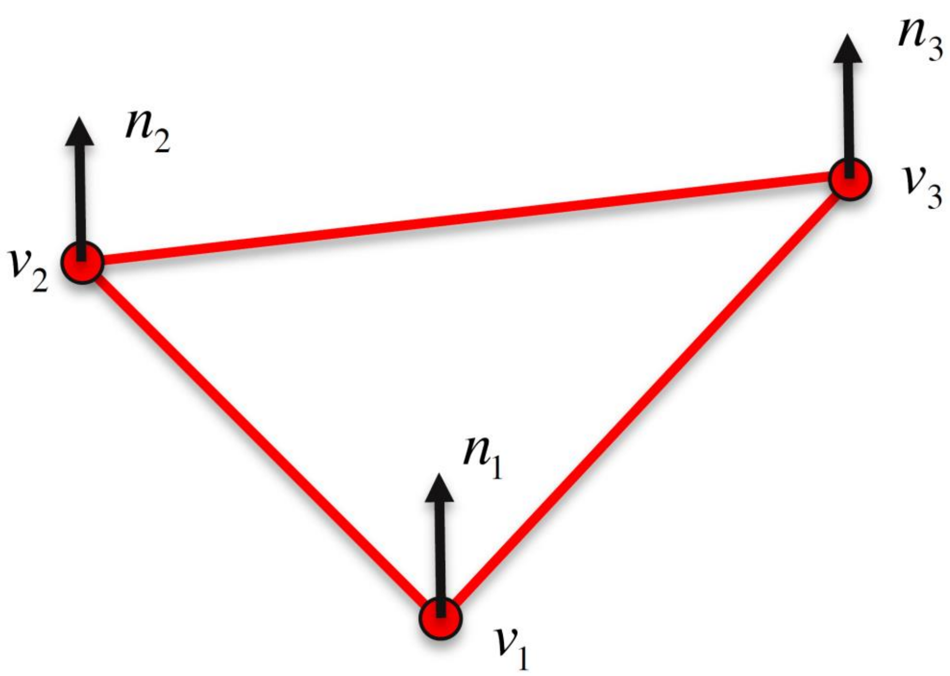
Visual Rendering

- Converting 3D scene descriptions into 2D images
- The graphics pipeline
  - Geometry + transformations
  - Cameras and viewing
  - Lighting and shading
  - Rasterization
  - Texturing

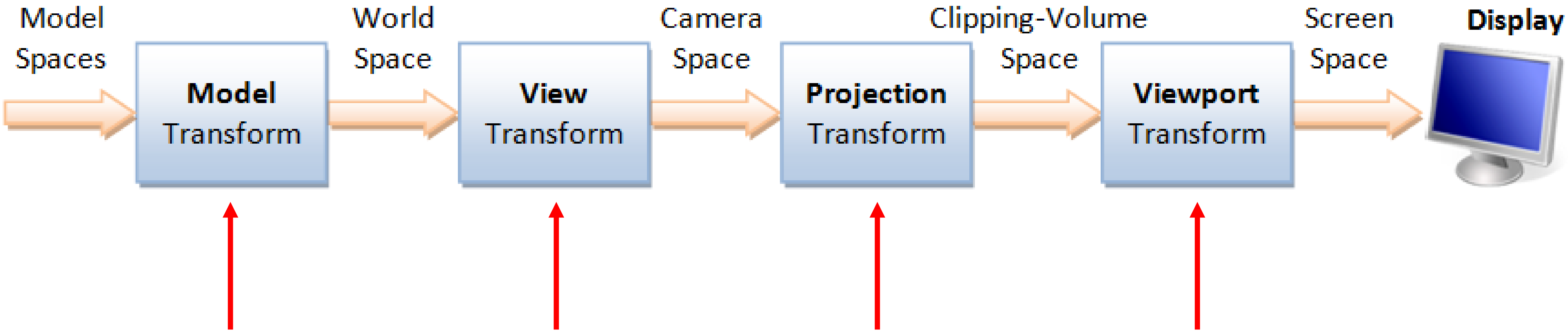


Primitives

- Vertex: 3D point v(x, y, z)
- Triangle (Face): 3D vertices
- Normal: 3D vector per vertex describing surface orientation $\mathbf{n} = (n_x, n_y, n_z)$



Vertex Transforms



Arrange Objects in the world

Position and orientate the camera

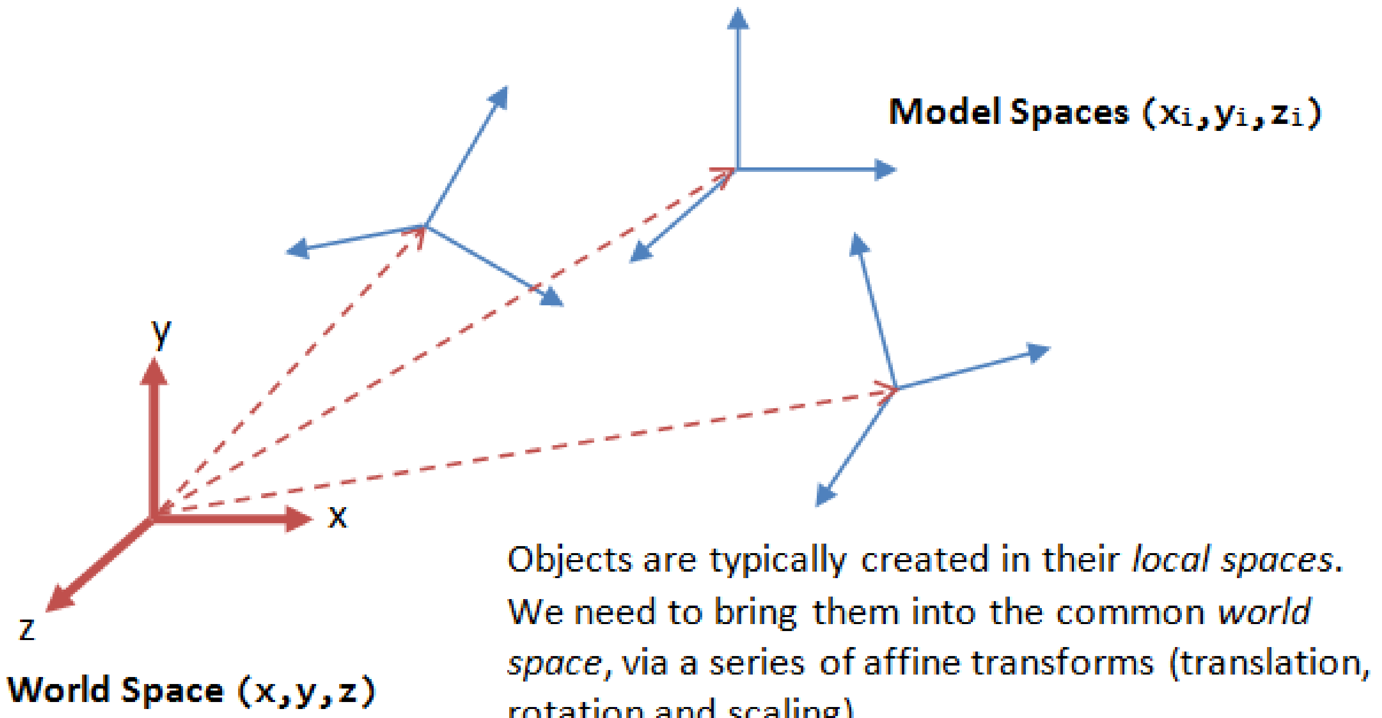
Select a camera lens, adjust focus length

Convert vertices to window coordinates

https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Model Transform

- Transform each vertex from object coordinates to world coordinates
  - 3D rotation and 3D translation



Objects are typically created in their *local spaces*. We need to bring them into the common *world space*, via a series of affine transforms (translation, rotation and scaling).

Object coordinates

Model Transform

- translation $T(d)=\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- scale $S(s)=\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- rotation $R_z(\theta)=\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $R_x=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $R_y=\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Vertex $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

Model Transform

- Combine transformations

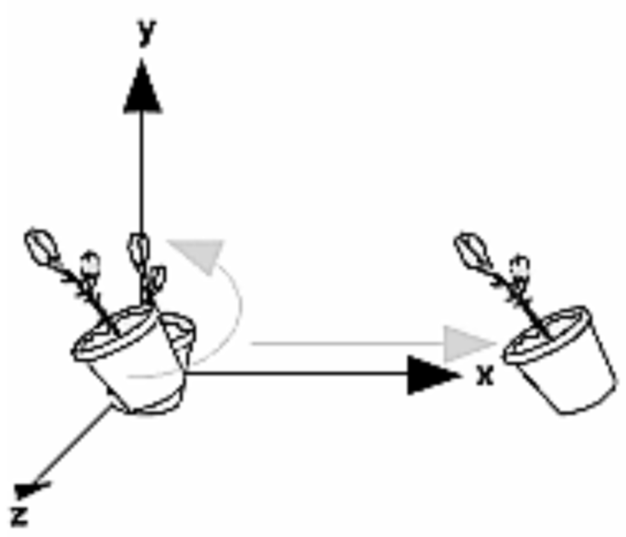
$$v' = T \cdot S \cdot R_z \cdot R_x \cdot T \cdot v$$

- Inverse

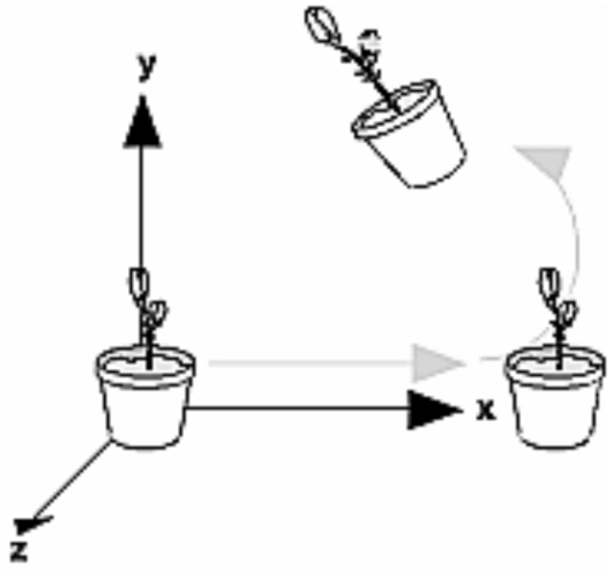
$$\begin{aligned} v &= \left(T \cdot S \cdot R_z \cdot R_x \cdot T\right)^{-1} \cdot v' \\ &= T^{-1} \cdot R_x^{-1} \cdot R_z^{-1} \cdot S^{-1} \cdot T^{-1} \cdot v' \end{aligned}$$

Model Transform

- Rotation and translation are not commutative (the order matters)



Rotate then Translate



Translate then Rotate

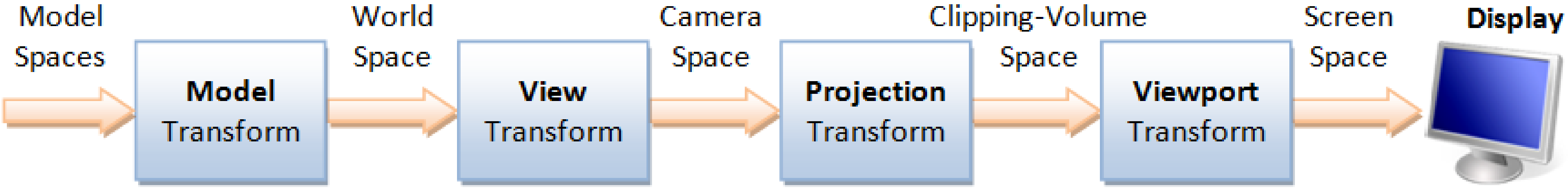
View Transform

- Transformation from world coordinate to camera or view coordinates

$$\mathbf{X}_{\mathrm{cam}} = R\mathbf{X} + \mathbf{t}$$

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

4x4



View Transform

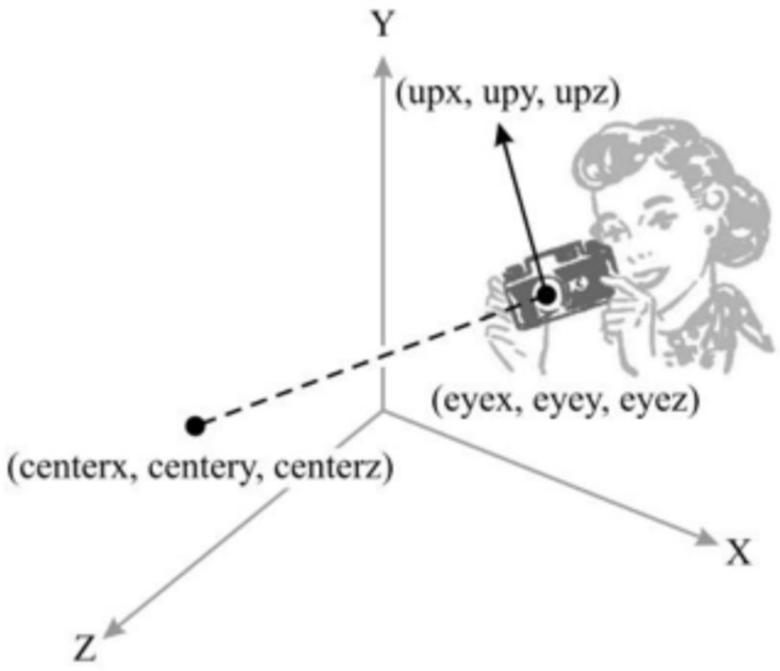
- Another way to specific the camera

  - eye position $eye = \begin{pmatrix} eye_x \\ eye_y \\ eye_z \end{pmatrix}$

  - reference position $center = \begin{pmatrix} center_x \\ center_y \\ center_z \end{pmatrix}$

    Look at

  - up vector $up = \begin{pmatrix} up_x \\ up_y \\ up_z \end{pmatrix}$



View Transform

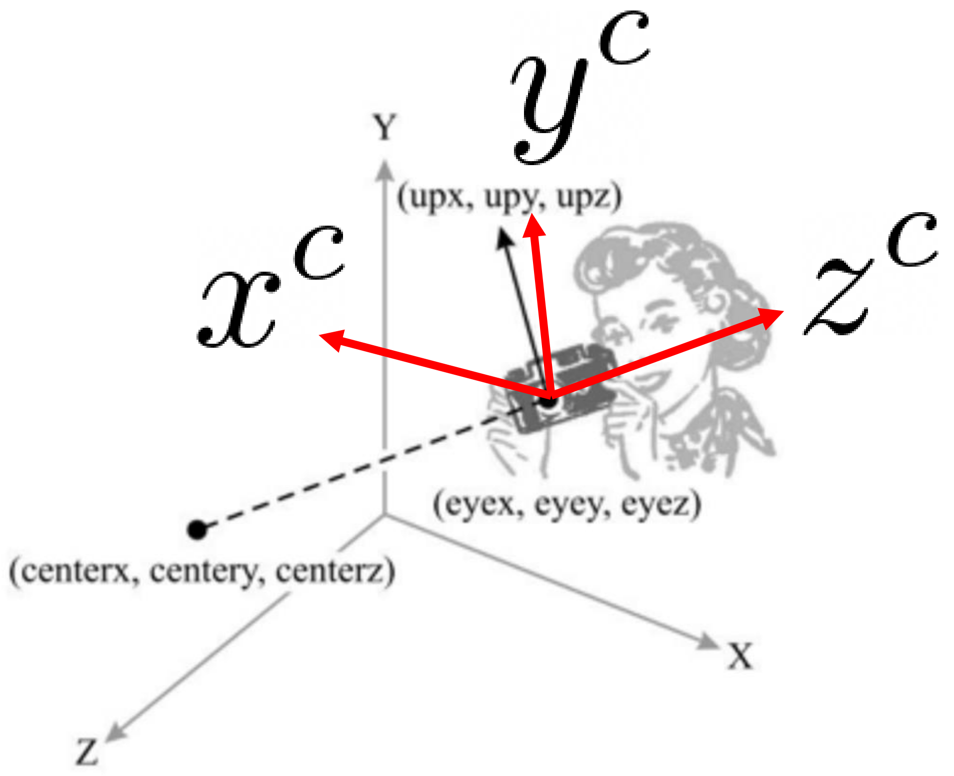
- Compute 3 vectors for the camera

$$z^c = \frac{eye - center}{\left\| eye - center \right\|}$$

$$x^c = \frac{up \times z^c}{\left\| up \times z^c \right\|}$$

$$y^c = z^c \times x^c$$

This can make sure y-axis is perpendicular to both x and z

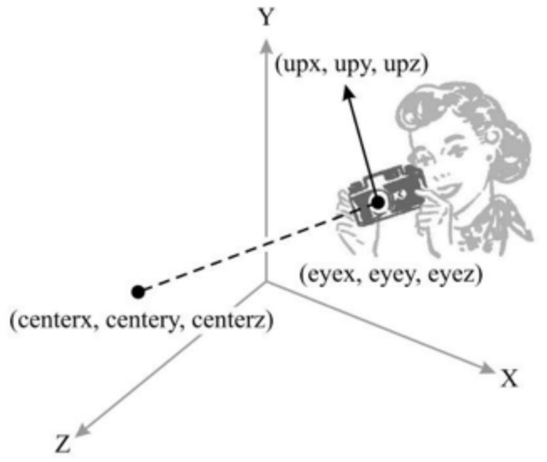


$$R^c = \begin{bmatrix} x^c & y^c & z^c \end{bmatrix}$$

Rotation from eye to world

View Transform

- Translation into eye position followed by rotation



$$M = R \cdot T(-e) = \begin{pmatrix} x_x^c & x_y^c & x_z^c & 0 \\ y_x^c & y_y^c & y_z^c & 0 \\ z_x^c & z_y^c & z_z^c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$z^c = \frac{eye - center}{\|eye - center\|}$$

$$R^c = [x^c \quad y^c \quad z^c]$$

$$R = R^{cT}$$

$$= \begin{pmatrix} x_x^c & x_y^c & x_z^c & -(x_x^c eye_x + x_y^c eye_y + x_z^c eye_z) \\ y_x^c & y_y^c & y_z^c & -(y_x^c eye_x + y_y^c eye_y + y_z^c eye_z) \\ z_x^c & z_y^c & z_z^c & -(z_x^c eye_x + z_y^c eye_y + z_z^c eye_z) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

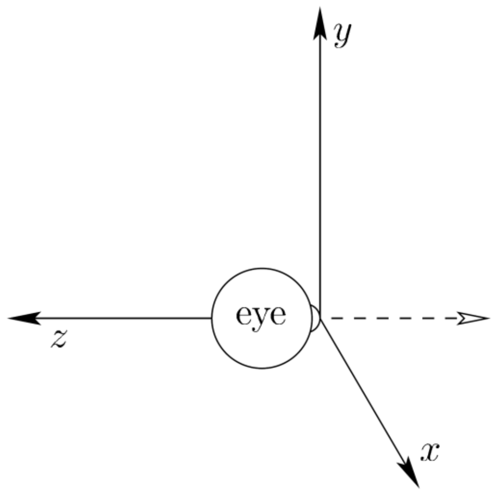
$$x^c = \frac{up \times z^c}{\|up \times z^c\|}$$

$$y^c = z^c \times x^c$$

View Transform

- Most graphics APIs has a function called lookat to compute the view transform matrix

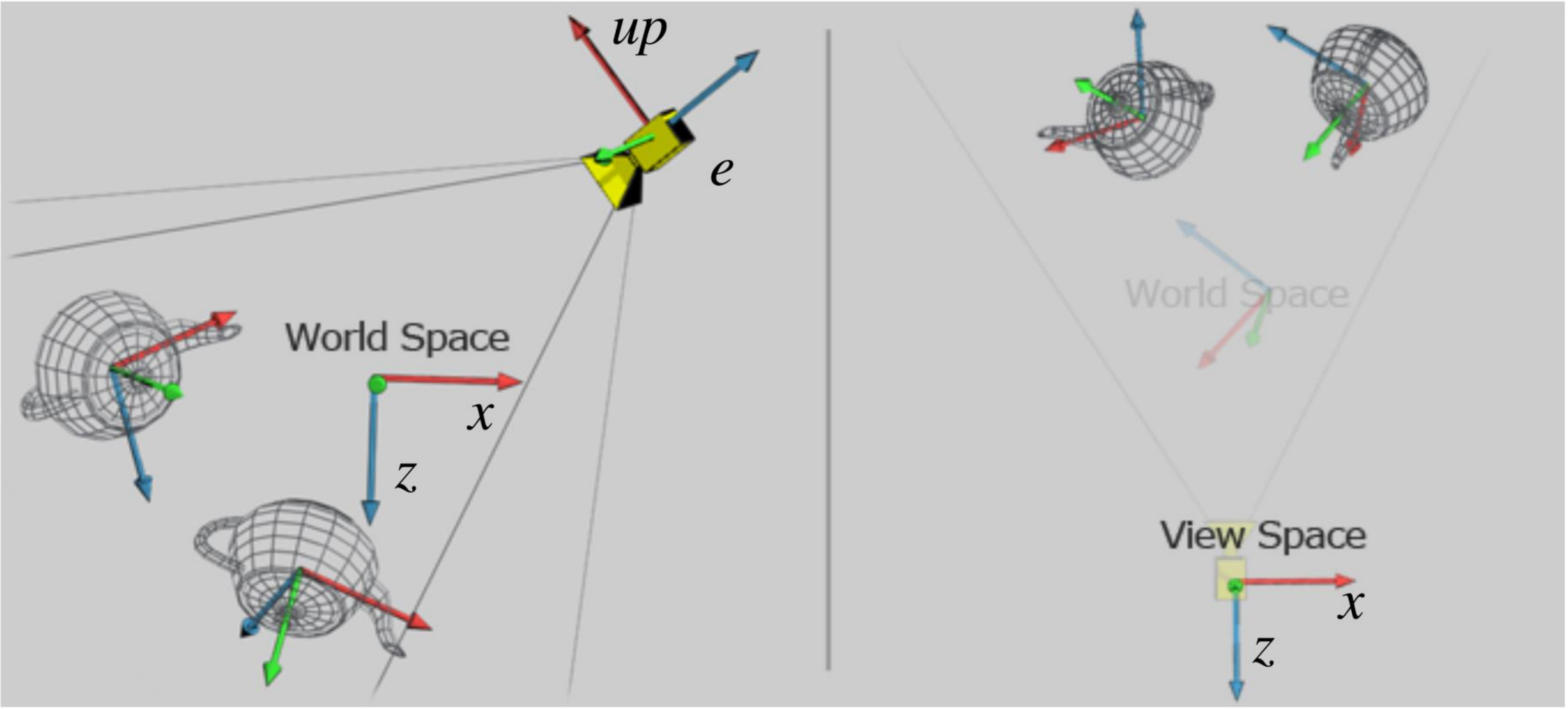
- In camera coordinates, the camera looks into negative z



- *Modelview matrix* is the combined model and view transformation matrix

View Transform

- In camera coordinates, the camera looks into negative z



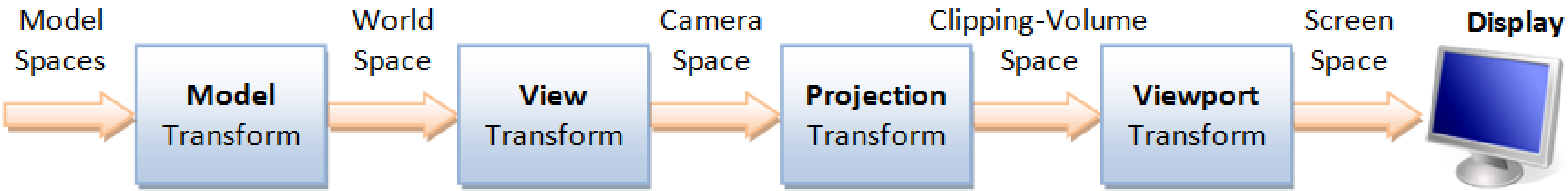
vodacek.zvb.cz

Projection Transform

- Similar to choose lens and sensor of camera, specify field of view and aspect of camera
  - Perspective projection
  - Orthographic projection

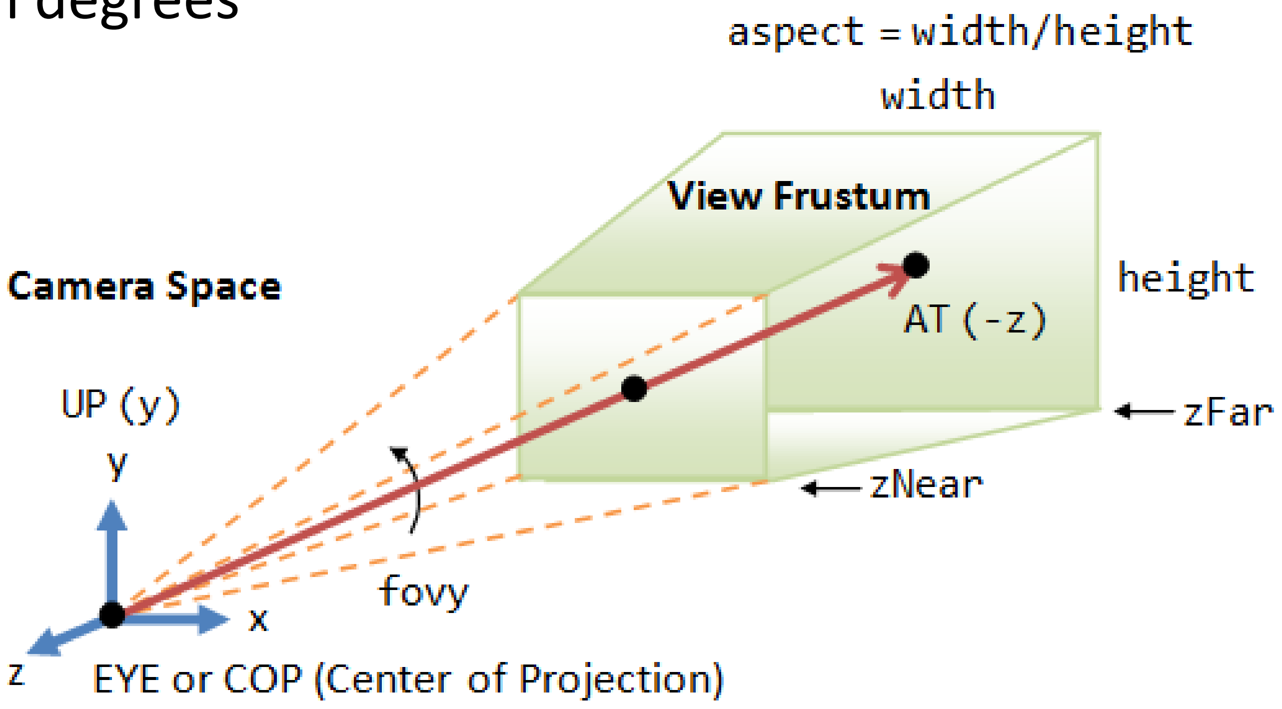
Camera model

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$



Projection Transform: Perspective Projection

- View frustum in perspective view (four parameters)
  - Fovy: total vertical angle of view in degrees
  - Aspect: ratio of width/height
  - zNear: near clipping plane
  - zFar: far clipping plane



Perspective Projection: The camera's view frustum is specified via 4 view parameters: fovy, aspect, zNear and zFar.

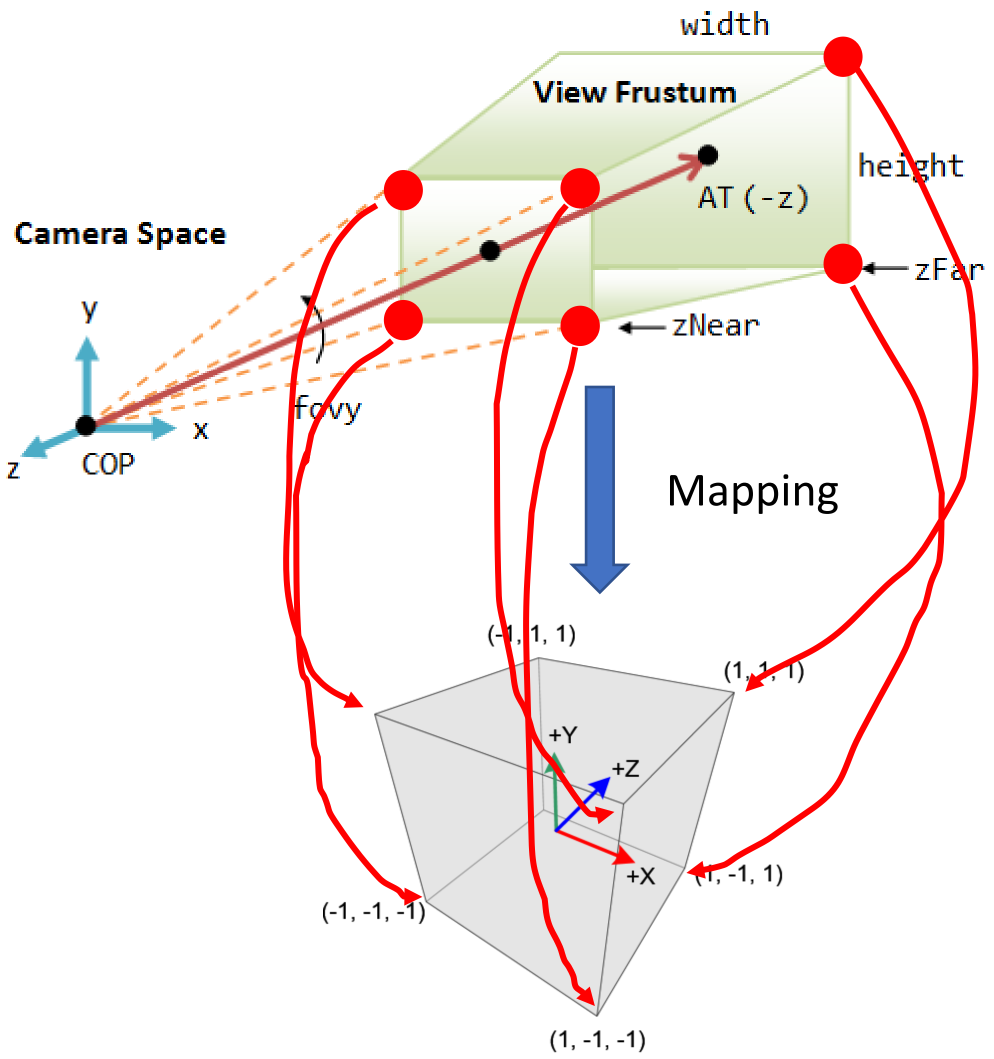
Projection Transform: Perspective Projection

- Clipping-Volume Cuboid 2x2x2

$$f = \cot(fovy/2) = \frac{zNear}{h/2}$$

$$M_{proj} = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -\frac{zFar + zNear}{zFar - zNear} & -\frac{2 \cdot zFar \cdot zNear}{zFar - zNear} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

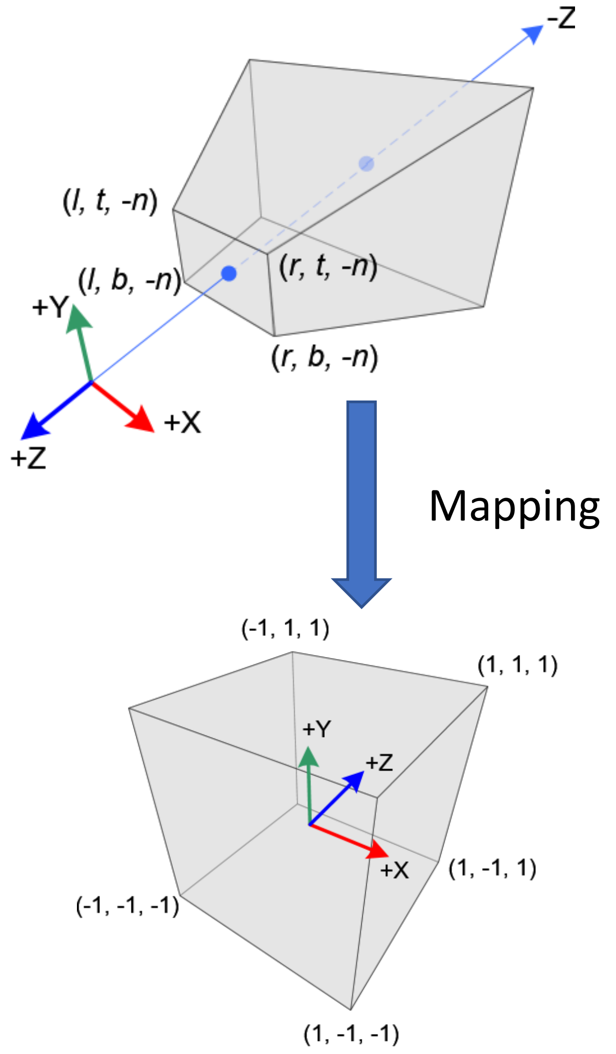
Flip z



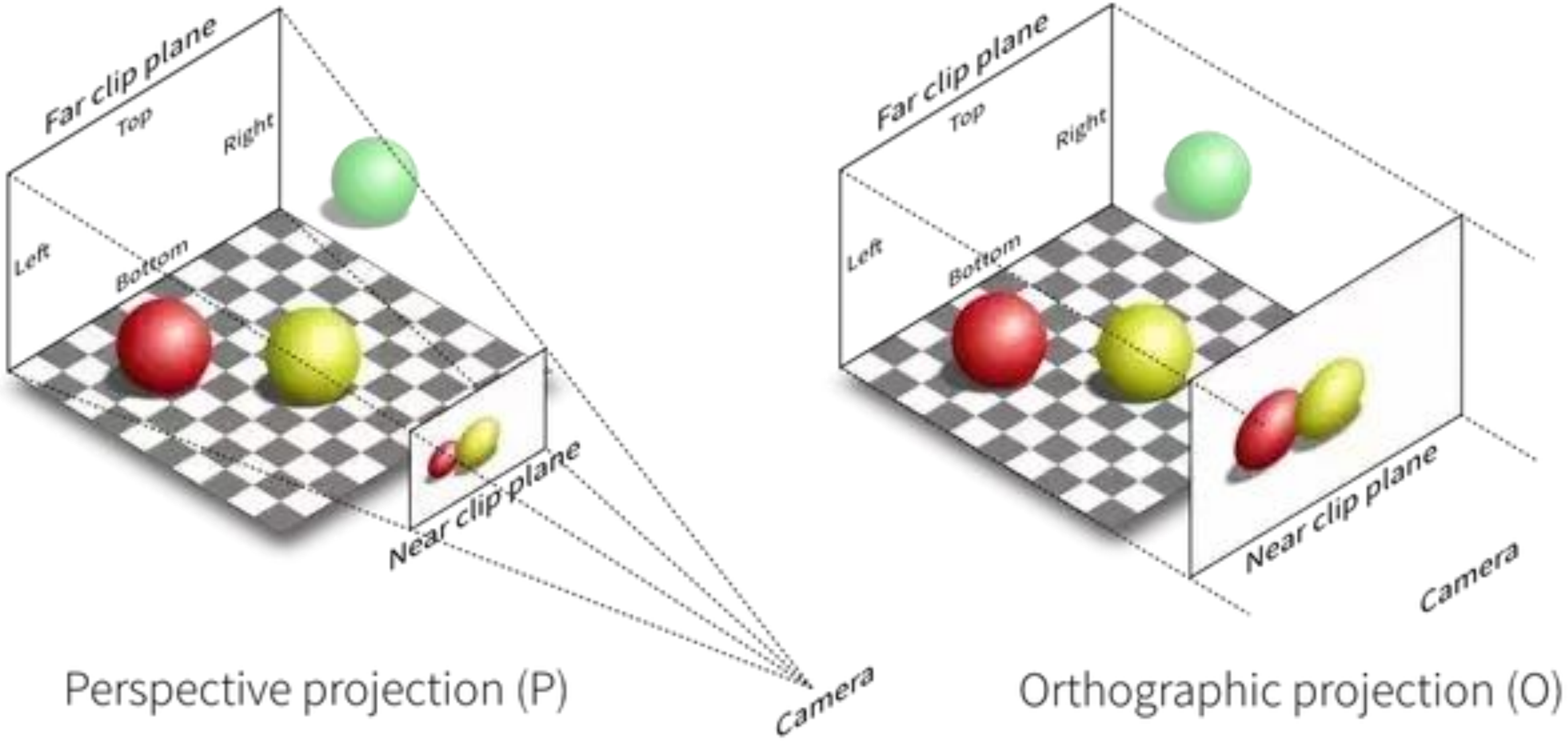
Projection Transform: Perspective Projection

- Specify the view frustum by left (l), right (r), bottom (b), and top (t) corner coordinates on near clipping plane (at zNear)

$$M_{proj} = \begin{pmatrix} \frac{2 \cdot zNear}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2 \cdot zNear}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{zFar+zNear}{zFar-zNear} & -\frac{2 \cdot zFar \cdot zNear}{zFar-zNear} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$



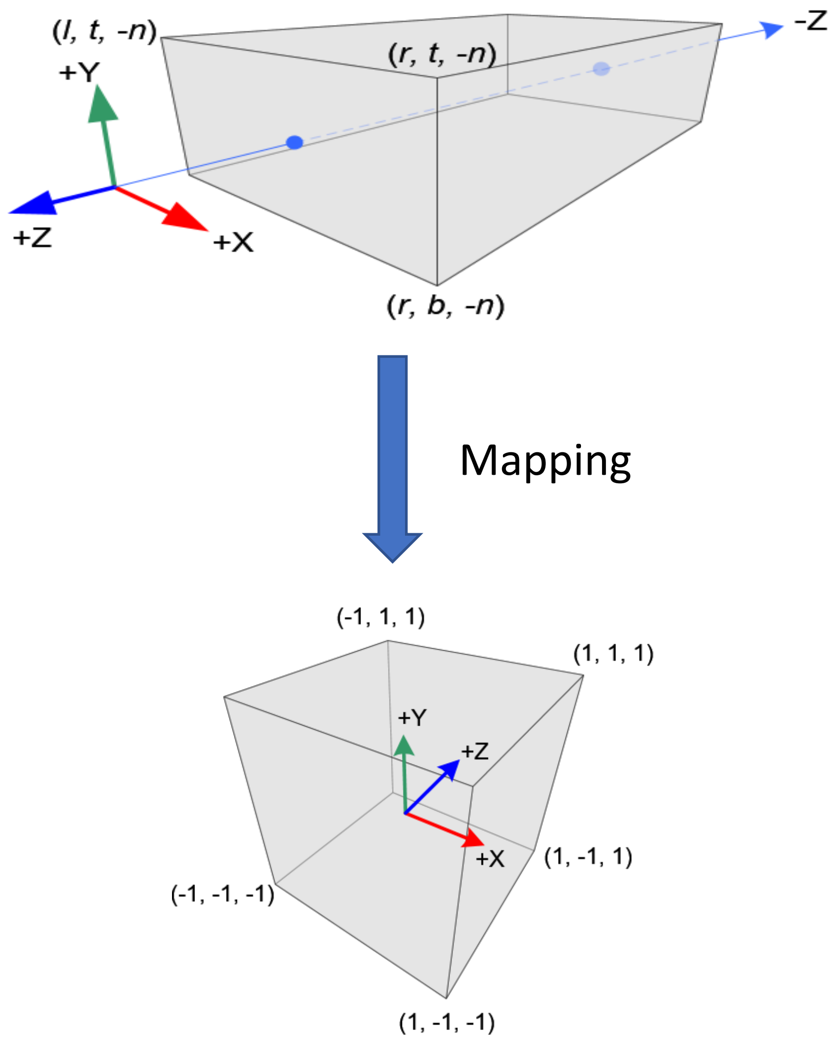
Orthographic Projection v.s. Perspective Projection



Projection Transform: Orthographic Projection

- Camera is placed very far away (parallel projection)

$$M_{proj} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

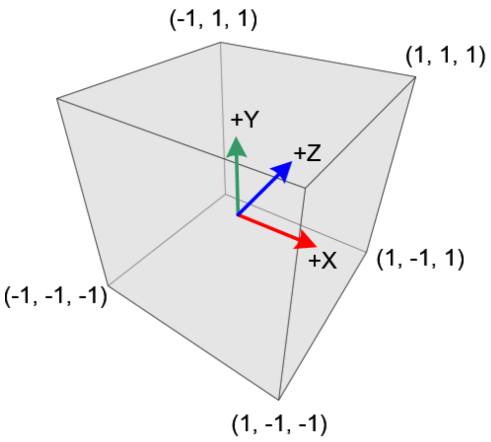


Modelview Projection Matrix

- Combine all the transformations

$$v_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot v = M_{proj} \cdot M_{mv} \cdot v$$

Vertex in clip space



Projection matrix

Modelview matrix

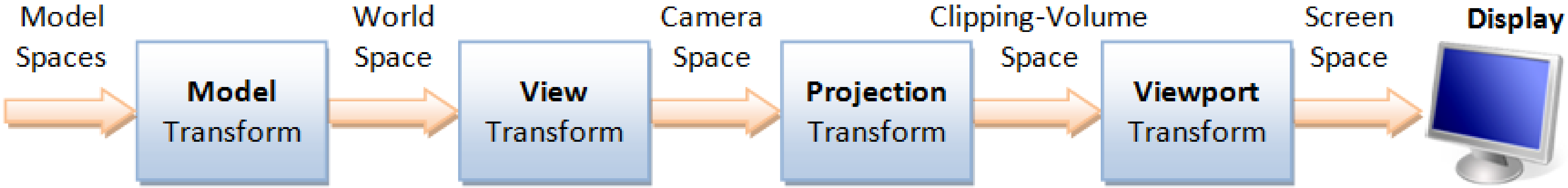
Viewport Transform

- Normalized Device Coordinate (NDC)

$$v_{clip} = \begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} \longrightarrow v_{NDC} = \begin{pmatrix} x_{clip} / w_{clip} \\ y_{clip} / w_{clip} \\ z_{clip} / w_{clip} \\ 1 \end{pmatrix} \begin{matrix} \in (-1,1) \\ \in (-1,1) \\ \in (-1,1) \\ \end{matrix}$$

vertex in clip space

vertex in NDC



Viewport Transform

- Define window as viewpoint (x, y, width, height)
  - (x, y) lower left corner of the viewport rectangle (default is (0, 0))
  - Width, height size of viewport rectangle in pixels

$$v_{NDC} = \begin{pmatrix} x_{clip} / w_{clip} \\ y_{clip} / w_{clip} \\ z_{clip} / w_{clip} \\ 1 \end{pmatrix} \longrightarrow v_{window} = \begin{pmatrix} x_{window} \\ y_{window} \\ z_{window} \\ 1 \end{pmatrix} \begin{matrix} \in (0, width) \\ \in (0, height) \\ \in (0,1) \\ \end{matrix}$$

vertex in NDC

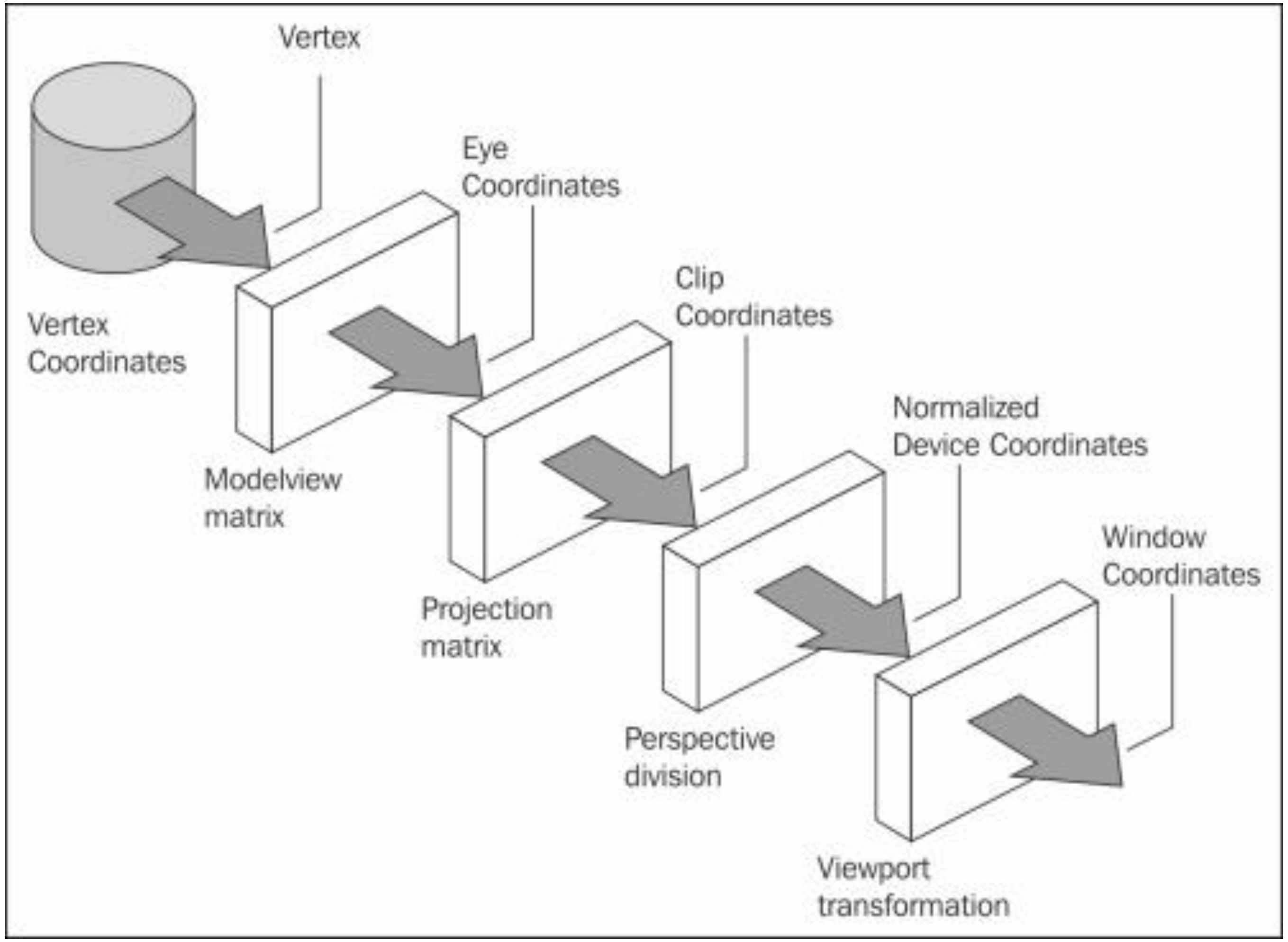
vertex in window coords

$$x_{window} = \frac{width}{2}(x_{NDC} + 1) + x$$

$$y_{window} = \frac{height}{2}(y_{NDC} + 1) + y$$

$$z_{window} = \frac{1}{2} z_{NDC} + \frac{1}{2}$$

Vertex Transform Pipeline



Further Reading

- 3D graphics with OpenGL, Basic Theory

https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

- Textbook: Shirley and Marschner “Fundamentals of Computer Graphics”, AK Peters, 2009