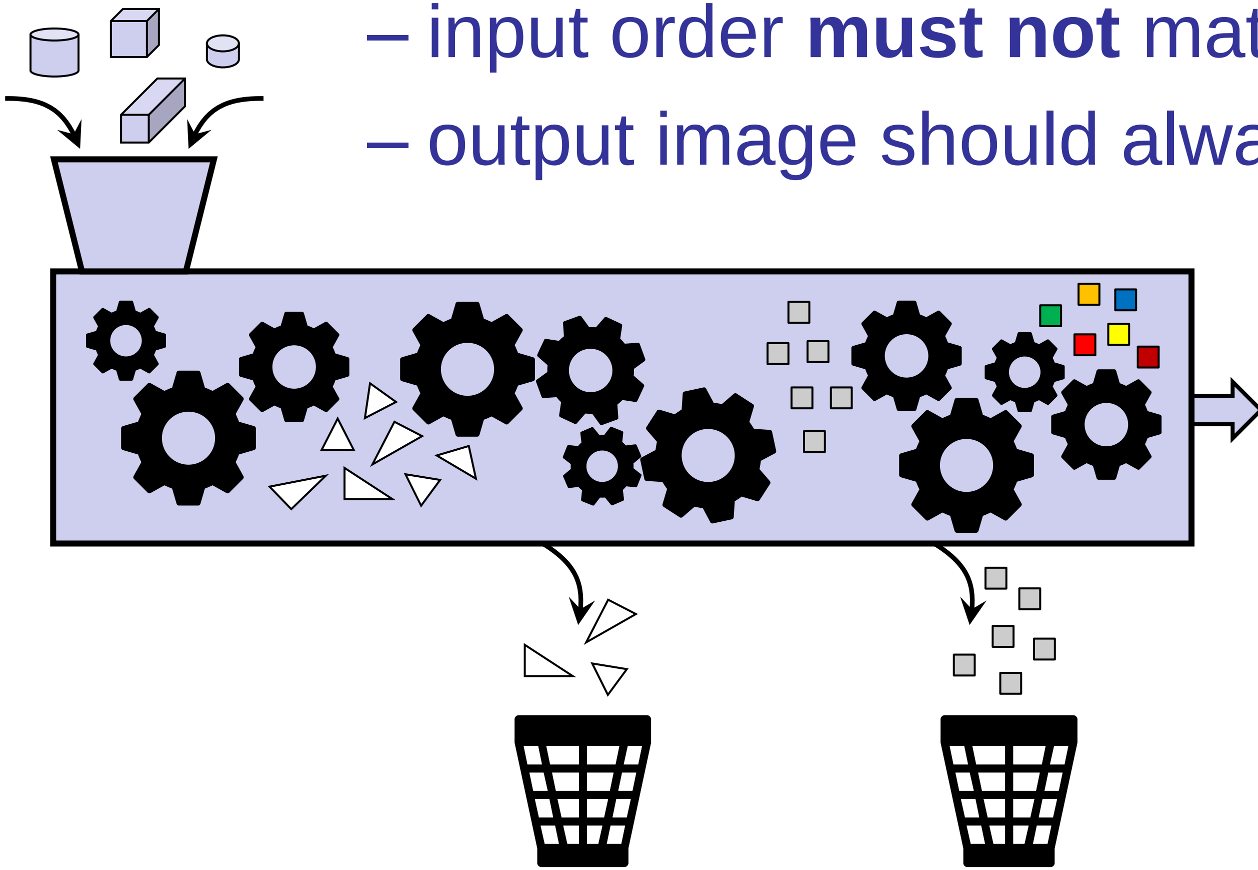
Computer Graphics

Viewing & Projections

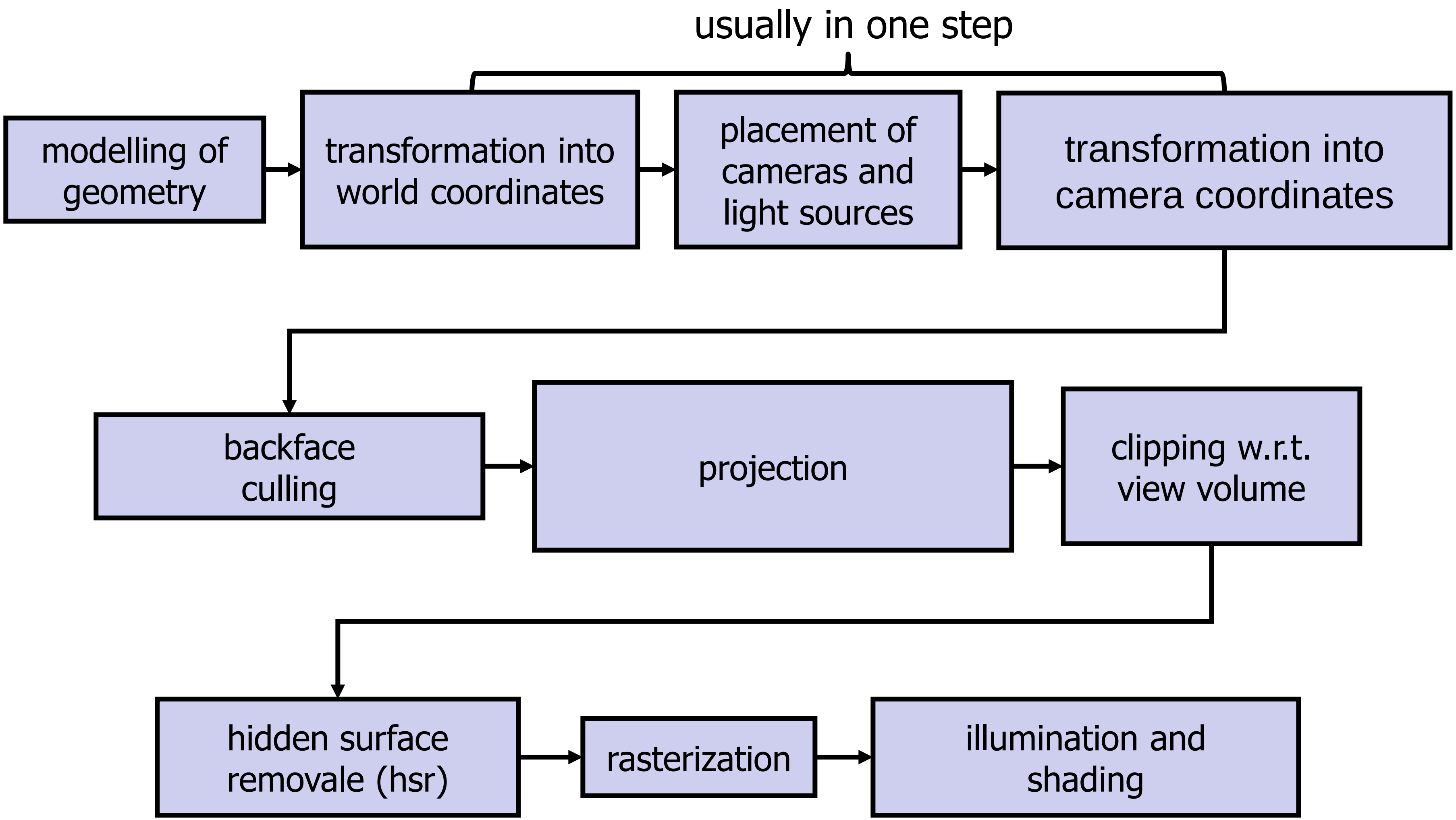
Overview

- general approach: a pipeline
  → process from model to final image
  - input order **must not** matter
  - output image should always be correct



iStock

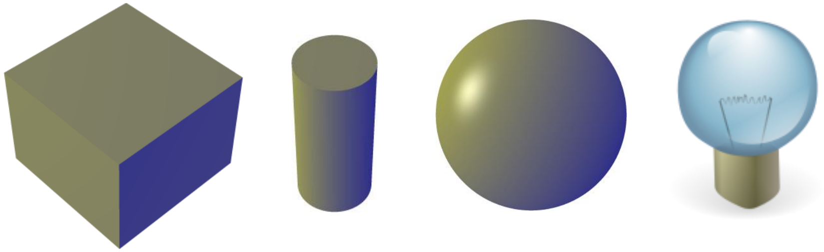
Rendering Pipeline



Rendering Pipeline

- first part of the pipeline: transformation into camera coordinates
  - **model-view transformation**: 1st stage, arranging model w.r.t. camera
    - requires camera reference (coordinate system)
  - **projection**: 2nd transformation stage, coordinate transformation to 2D
    - requires complete camera model
    - many different projections possible

Model-View Transformation



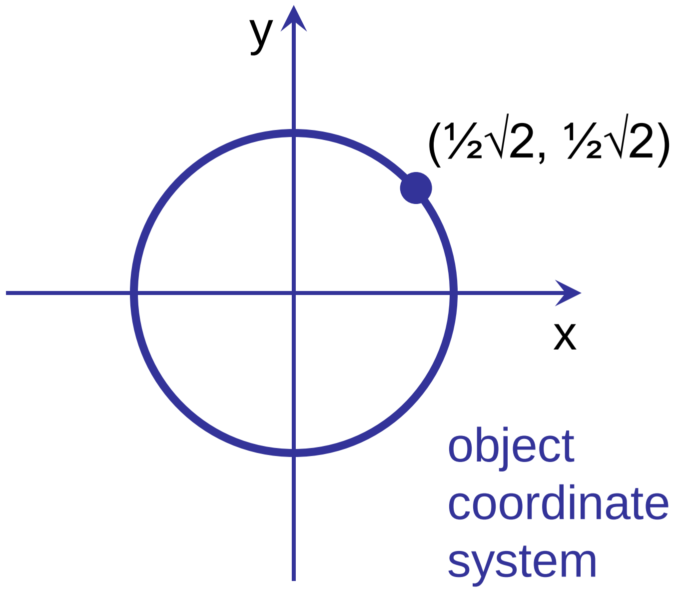
- input / foundations:
  - object definitions (including lights etc.)
  - including object positions & orientations
  - transformations in 3D
  - camera position & parameters

$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- problem:
  - how to arrange objects in space?

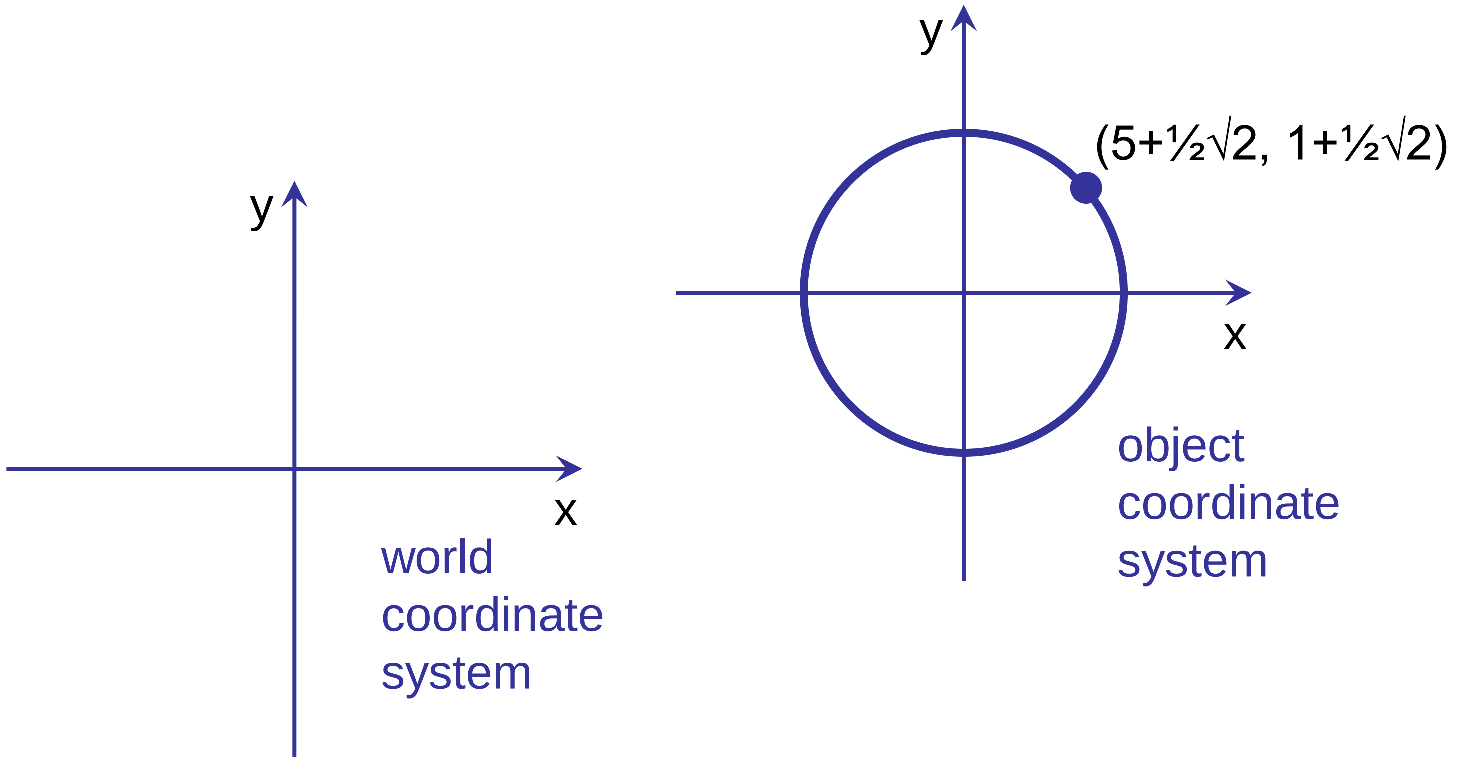
Model-View Transformation

- 1st idea: transformation of all objects into a (the) world coordinate system



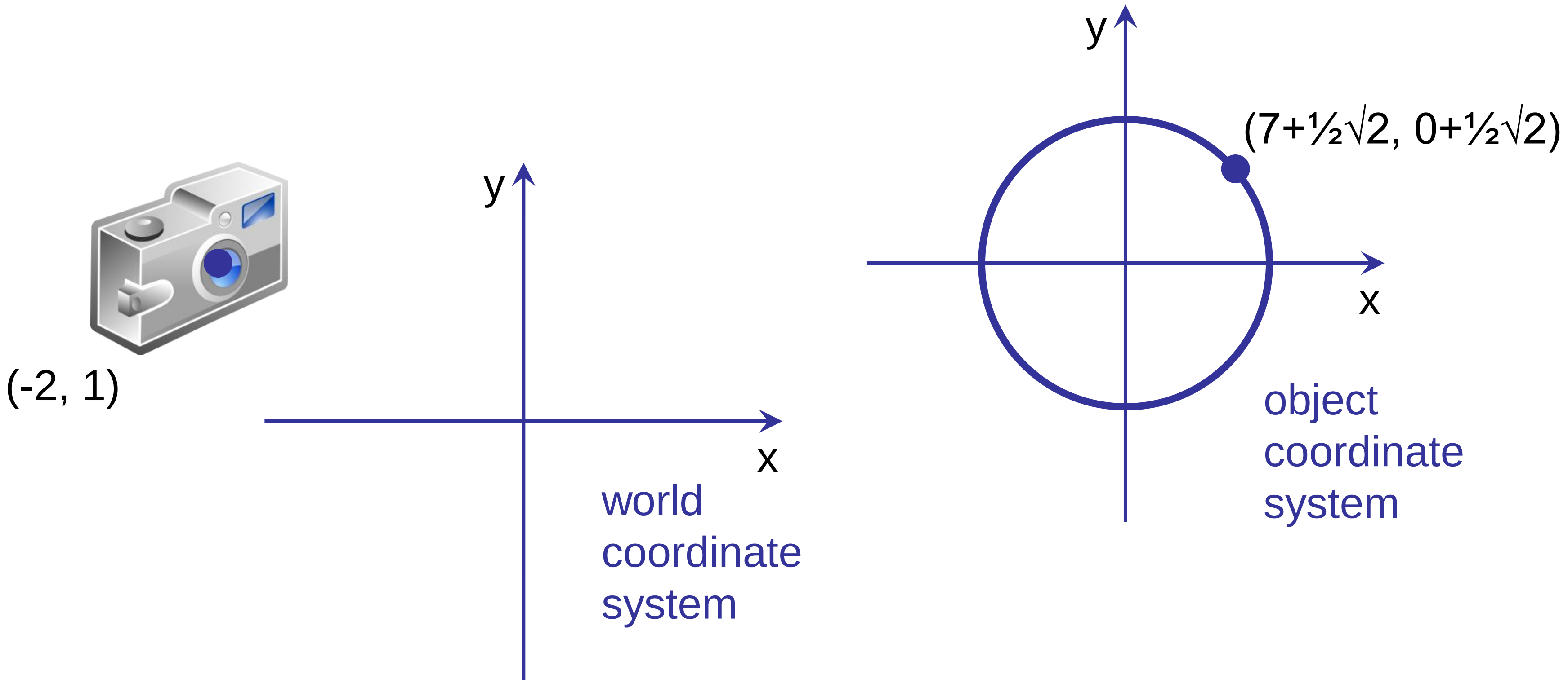
Model-View Transformation

- 1^{st} idea: transformation of all objects into a (the) world coordinate system



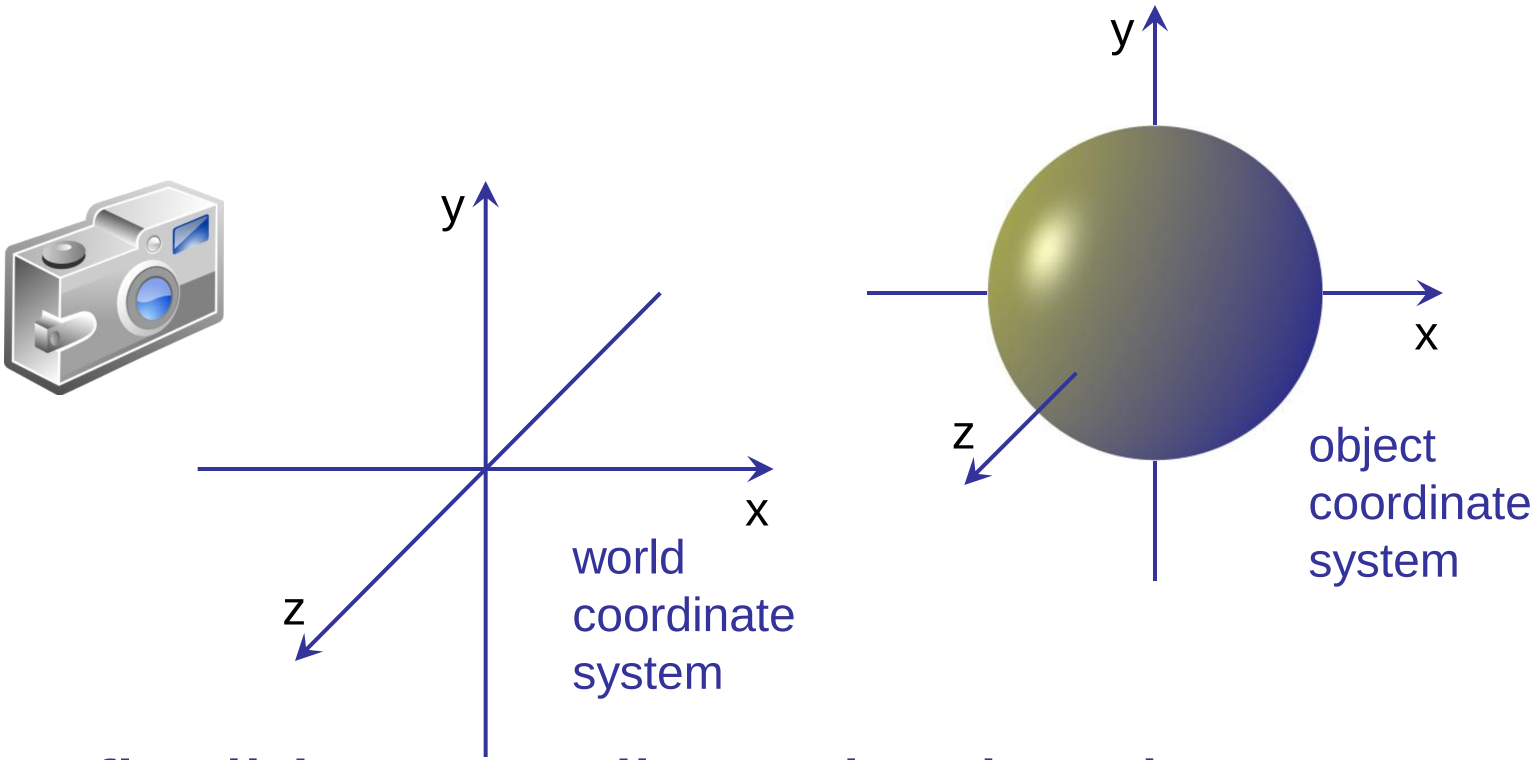
Model-View Transformation

- 1^{st} idea: transformation of all objects into a (the) world coordinate system



Model-View Transformation

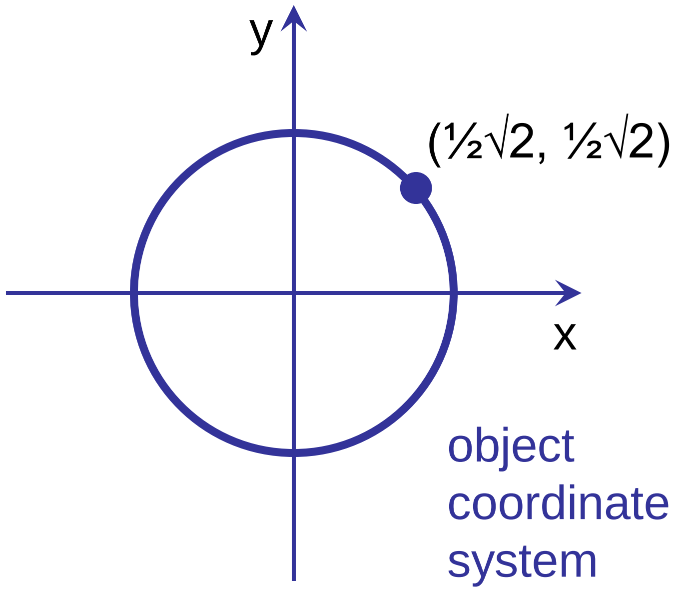
- 1^{st} idea: transformation of all objects into a (the) world coordinate system



- not flexible: complicated animation

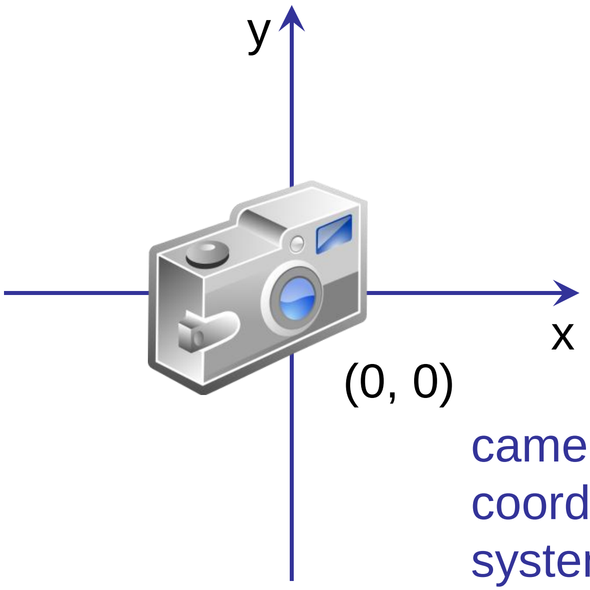
Model-View Transformation

- 2nd idea: transformation **directly** into camera coordinates: object-dependent

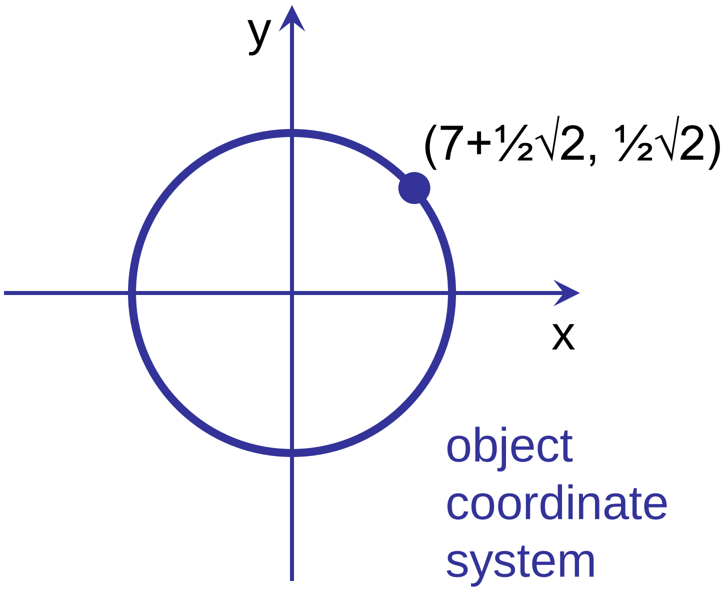


Model-View Transformation

- 2^{nd} idea: transformation **directly** into camera coordinates: object-dependent



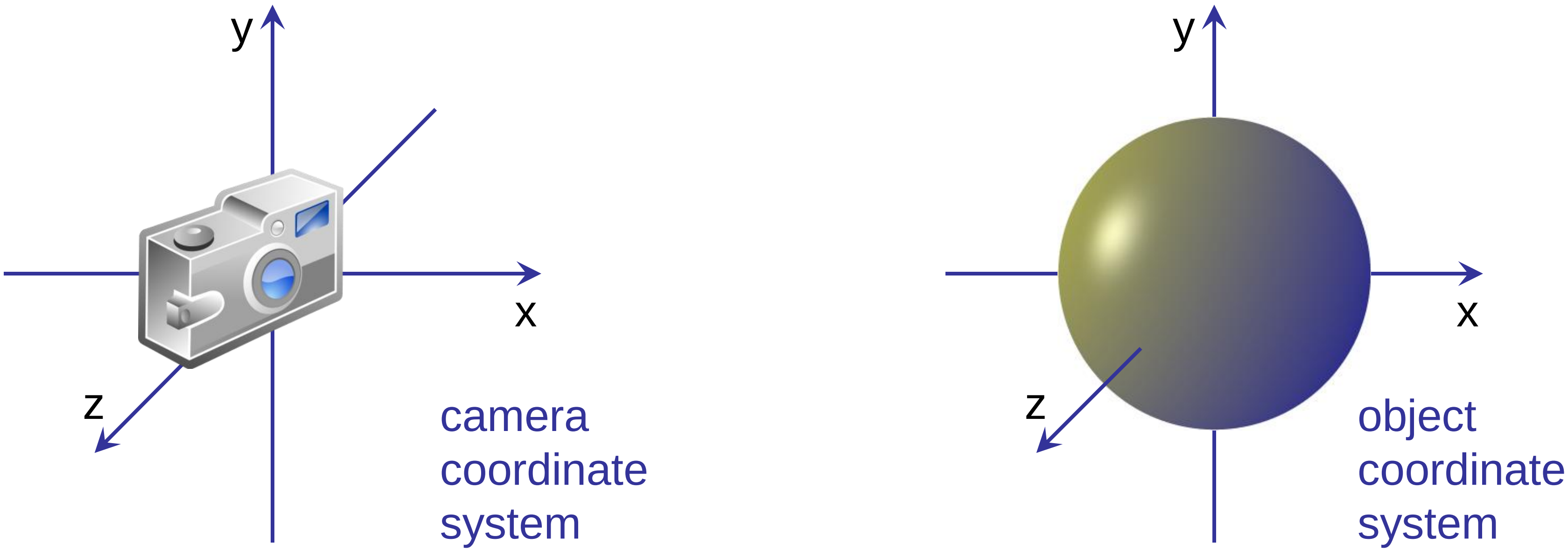
camera
coordinate
system



object
coordinate
system

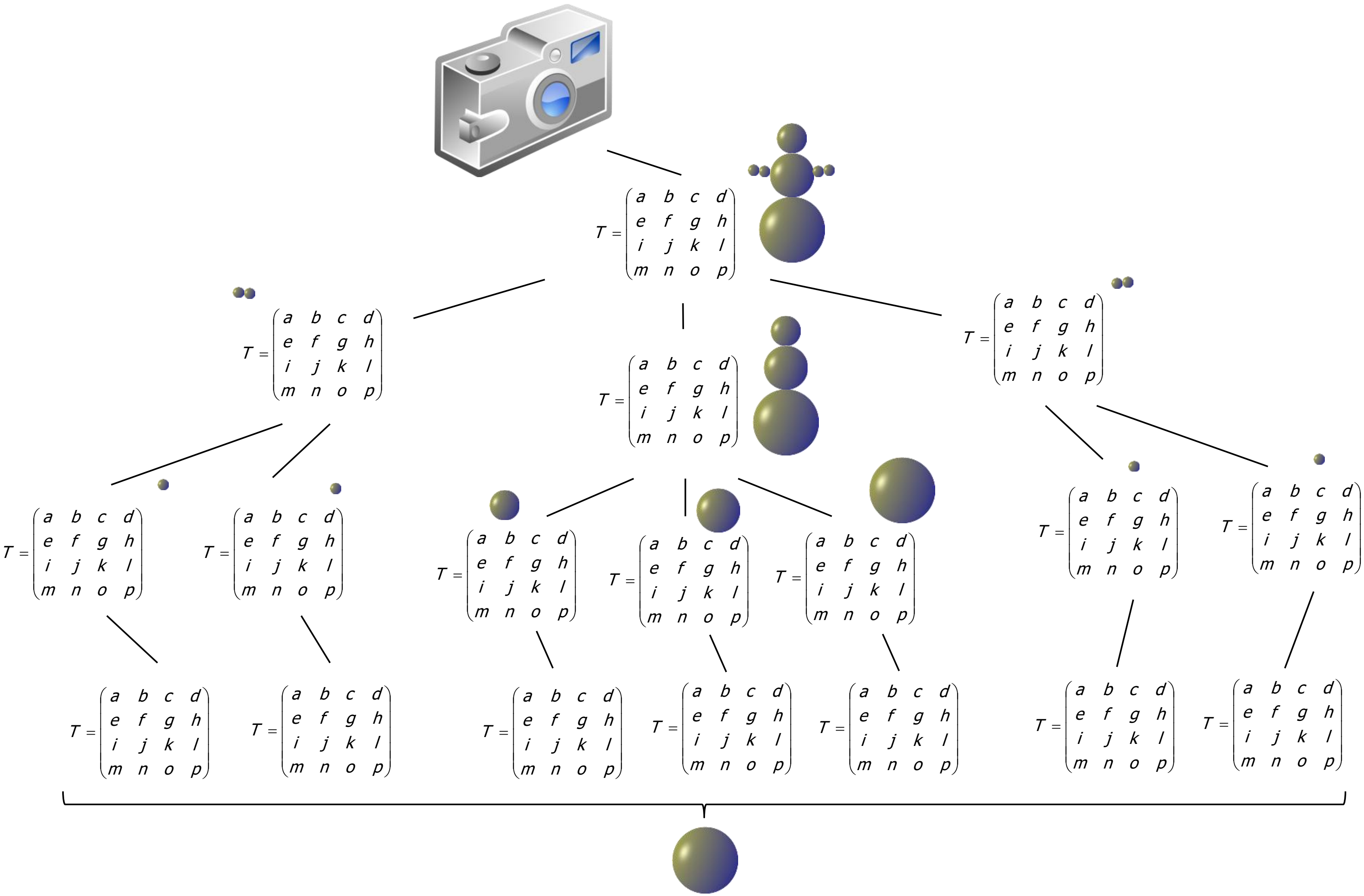
Model-View Transformation

- 2nd idea: transformation **directly** into camera coordinates: object-dependent



- each object has its own model-view matrix
- hierarchies possible, objects reusable

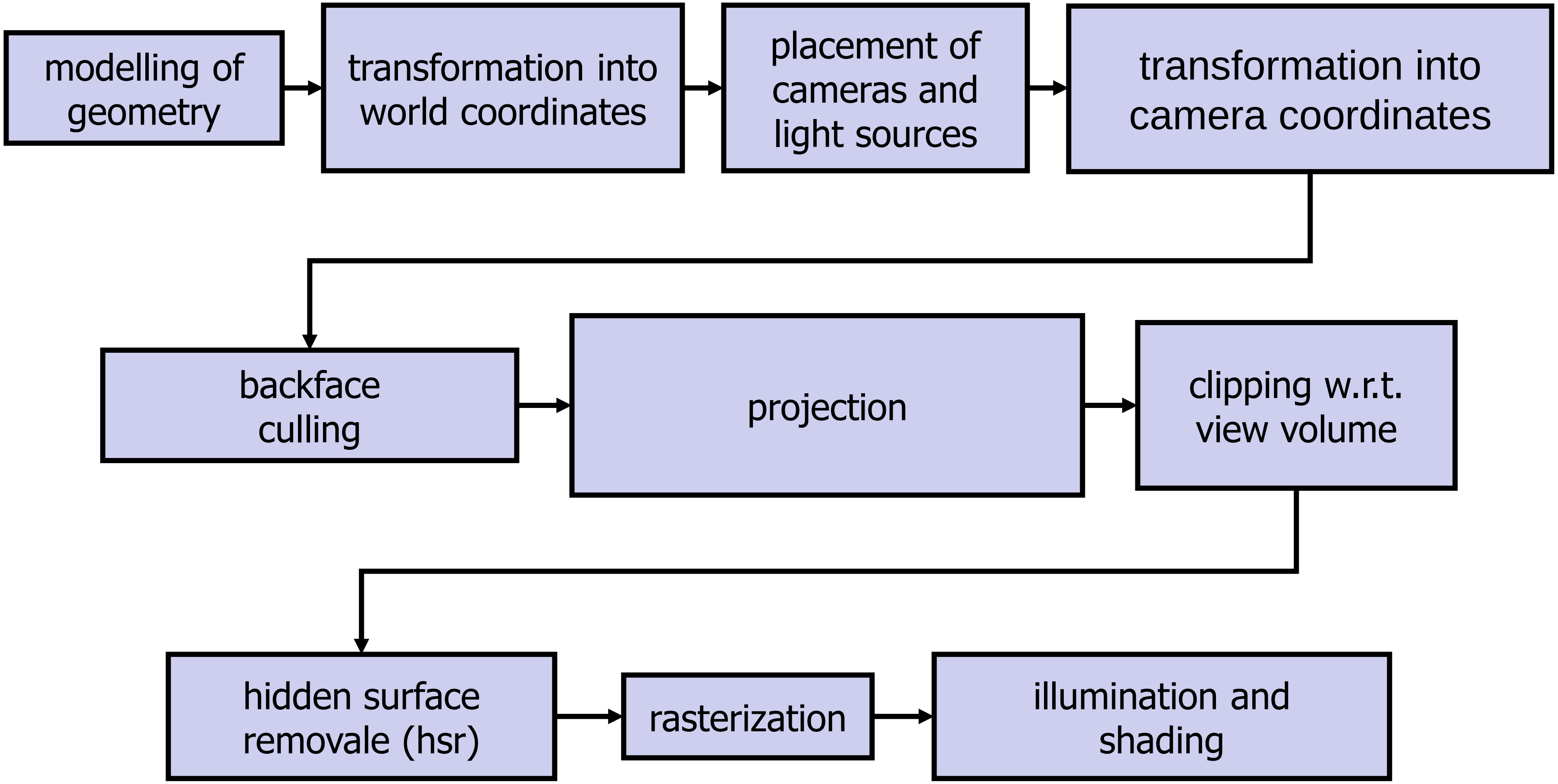
Model-View Transformation



Model-View Transformation

- model-view transformation steps:
  1. translate object origin to camera location
  2. rotate to align coordinate axes
  3. possibly also scaling
- this process is used in OpenGL:
  no explicit world coordinates!
- object & camera locations and orientations may be specified in a world coordinate system (e.g., in modeling systems)

Rendering Pipeline

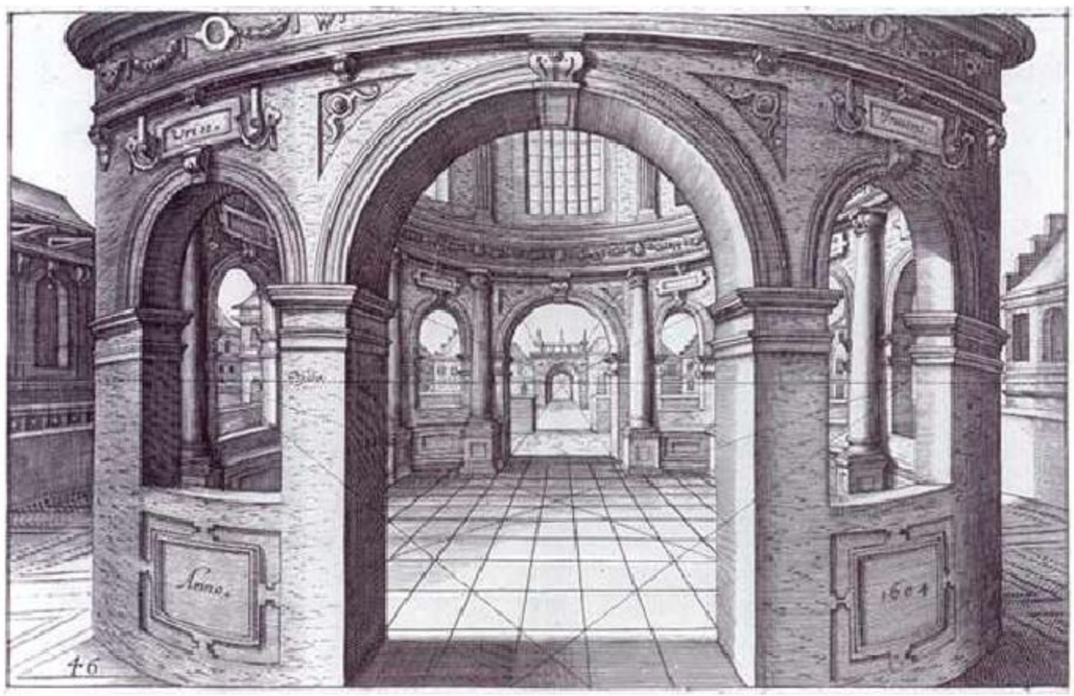


Projections

Introduction and Classification

Introduction

A painting [the rendering] is the intersection of a visual pyramid [view volume/view frustum] at a given distance, with a fixed center [center of projection] and a defined position of light, represented by art with lines and colors [the cg pipeline and its primitives] on a given surface [the projection plane/frame buffer]. (Alberti, 1435)

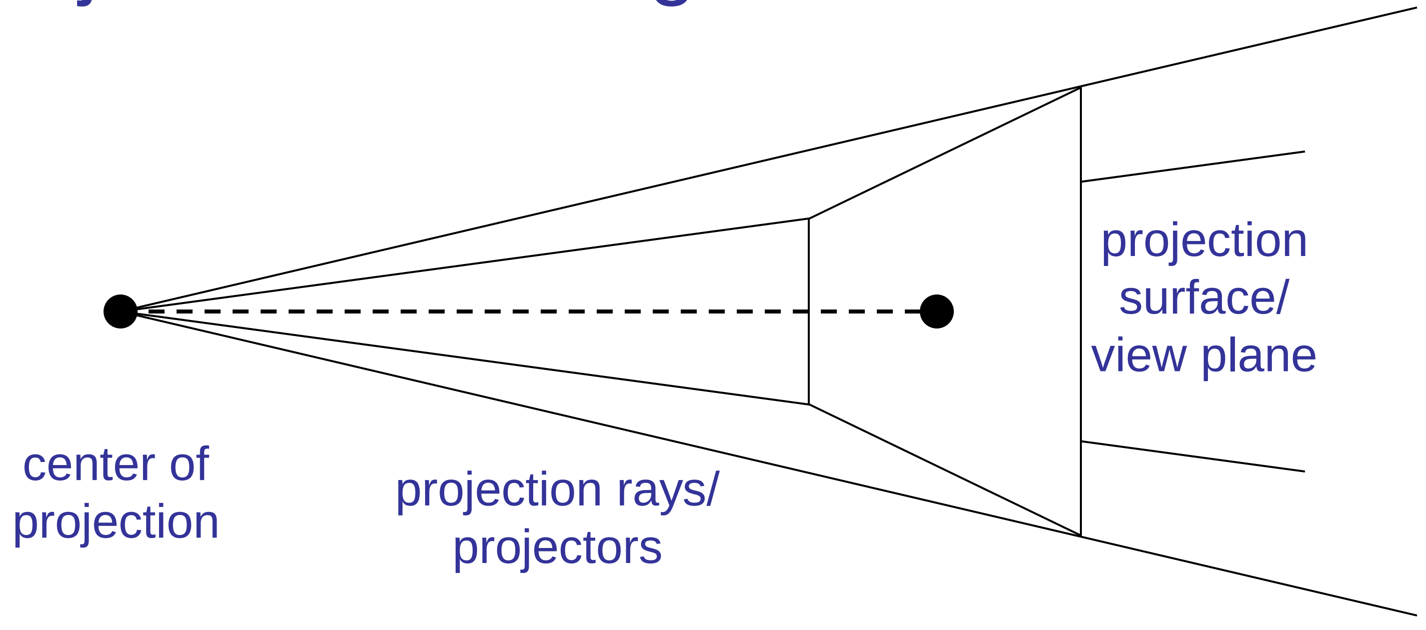


Hans Vredeman de Vries: Perspektiv 1604

Introduction

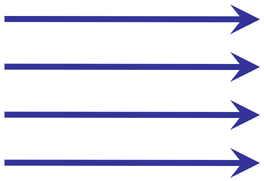
planar projection:

- projection rays are straight lines
- projection surface/view plane is planar
- projections of straight lines are also straight

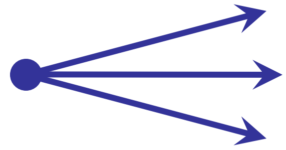


Introduction – Terms

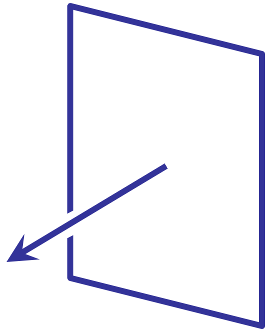
- parallel projection: characterized by **direction of projection** (dop)



- perspective projection: **center of projection** (cop)

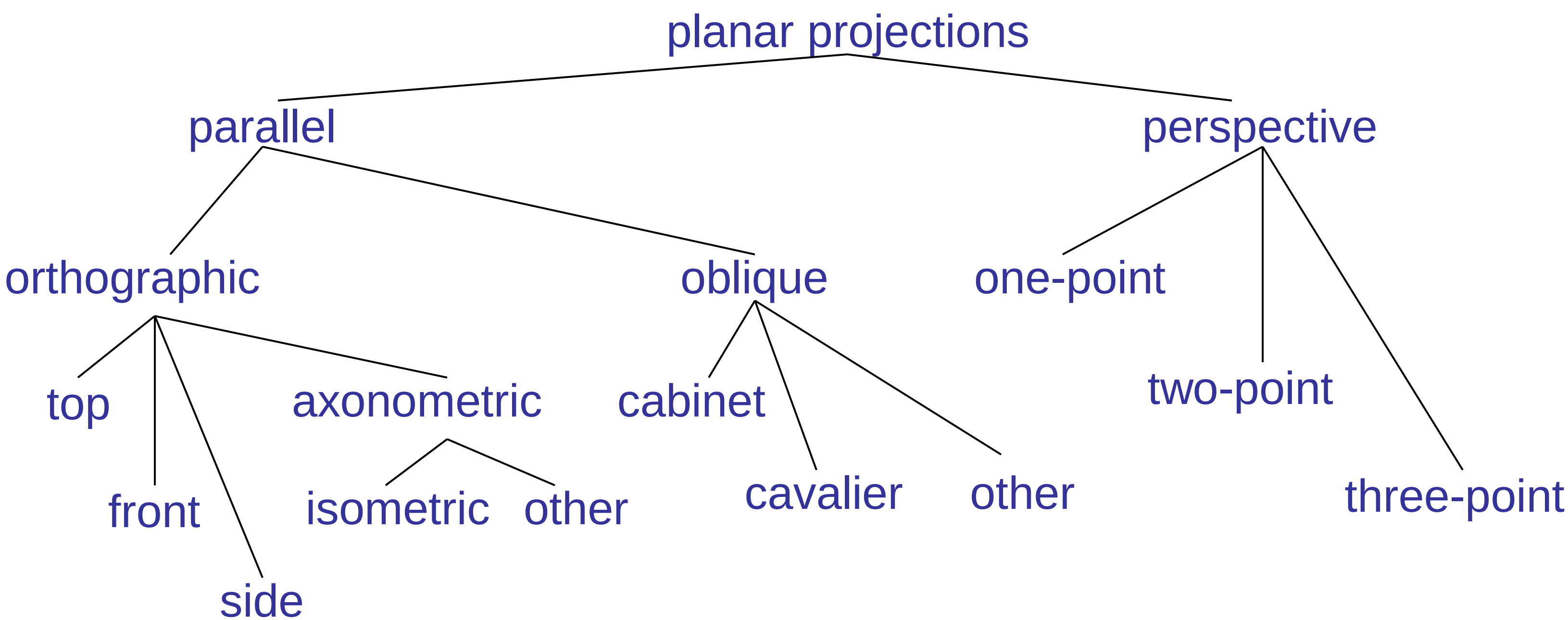


- projection on **view plane**
- vector perpendicular to view plane: **view plane normal** (vpn)



- rays characterizing projection: **projectors** (parallel or diverging from cop)

Classification of Planar Projections



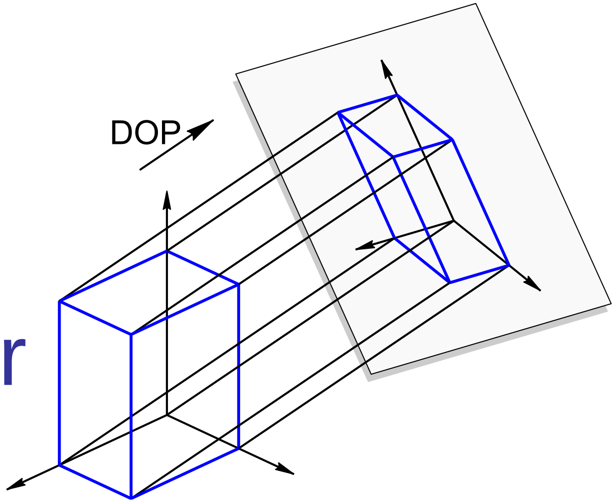
- parallel projections:
  all projectors parallel to each other
- perspective projections:
  projectors diverge from cop

Projections

Parallel Projections

Parallel Projections

- cop at infinity
- projectors are parallel
- no foreshortening: distant objects do not appear smaller
- parallel lines remain parallel
- classes of parallel projections:
  - depending on angles between vpn and dop
  - depending on dop and coordinate axes
- parallel projection in CG: some rendering computations faster



Parallel Projections

- typical applications:
  - architectural drawings (e.g., buildings which are mainly characterized by perpendicular angles)
  - medical visualizations
- reason:

  some lengths can be measured in the projected image

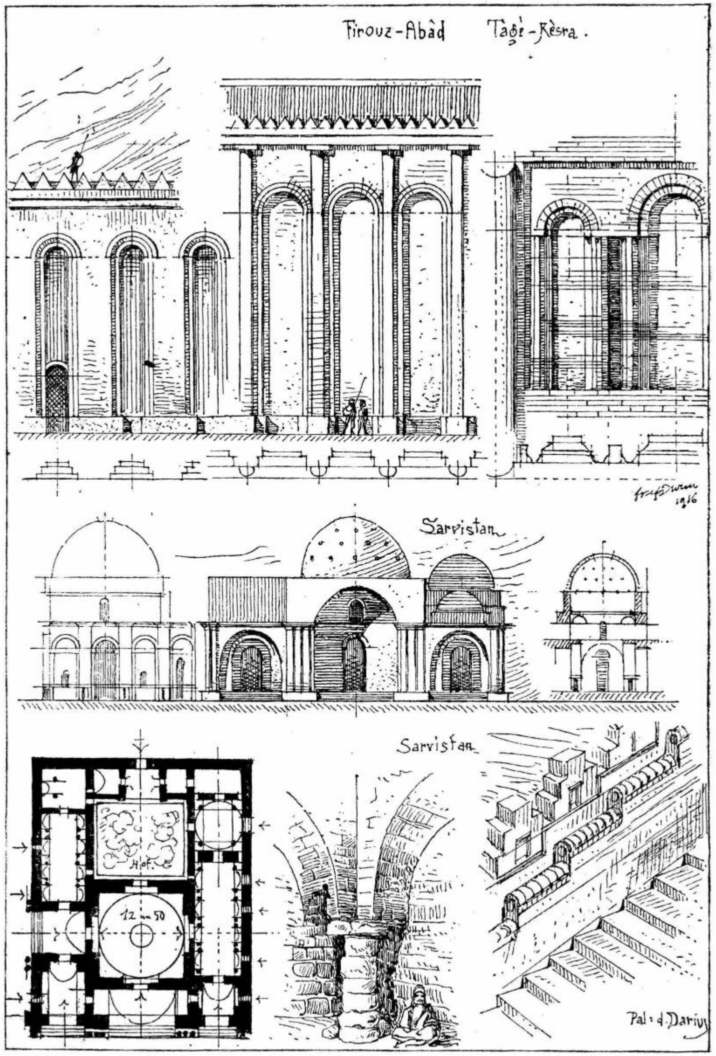
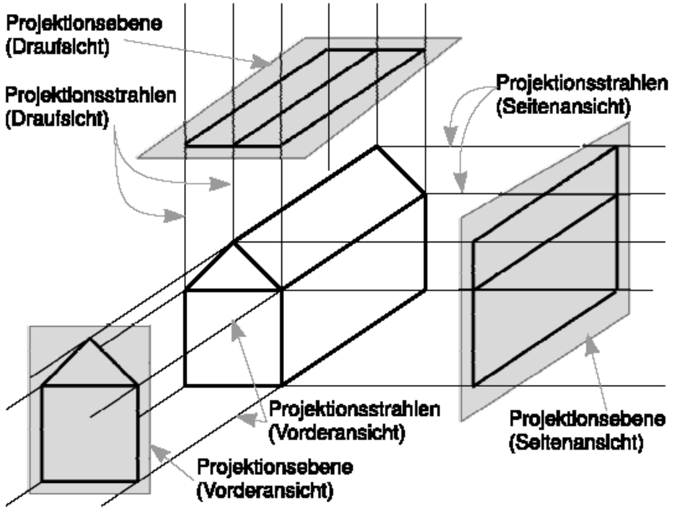


Fig. 18. – Palacios de Firouz-Abad y de Sarvistán

Parallel Projections: Special Cases

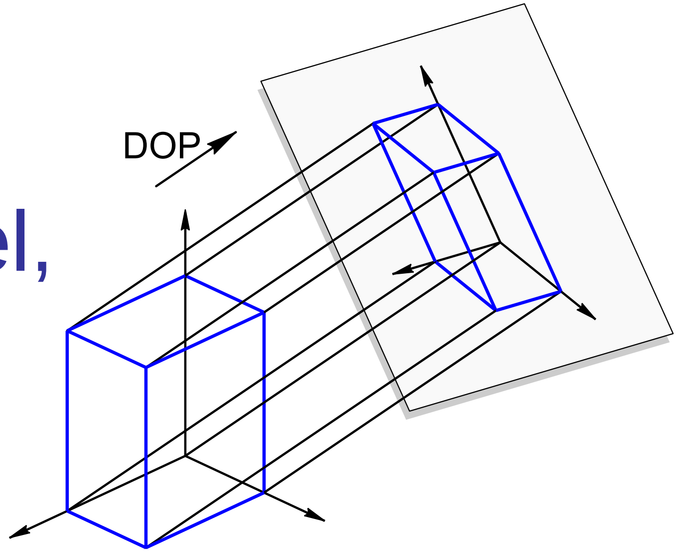
- top projection, side projection, front projection
- orthographic projections
- view plane is perpendicular to one of the coordinate axis or parallel to one of the coordinate planes or dop co-linear to one of the coordinate axes
- used in architecture



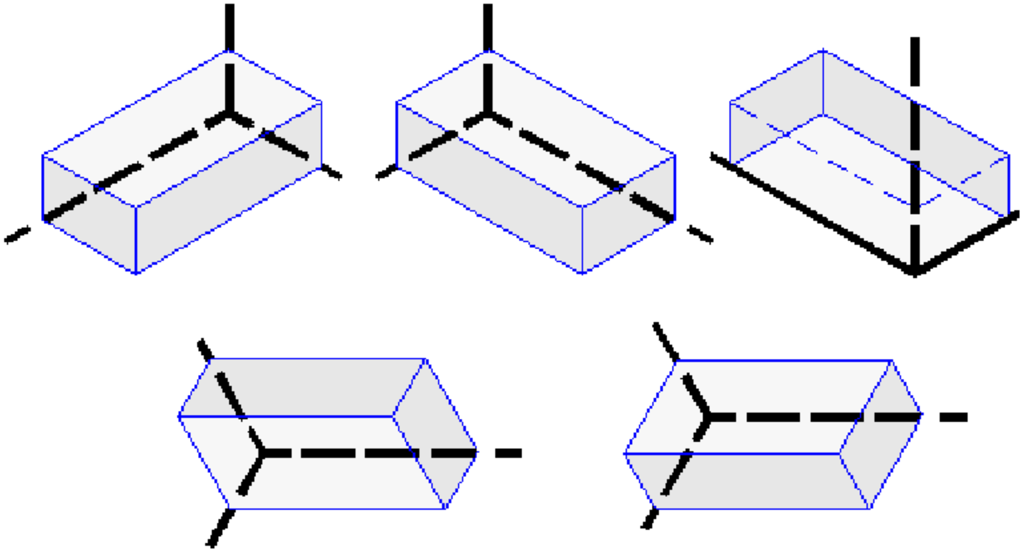
Angel (2000)

Axonometric Parallel Projections

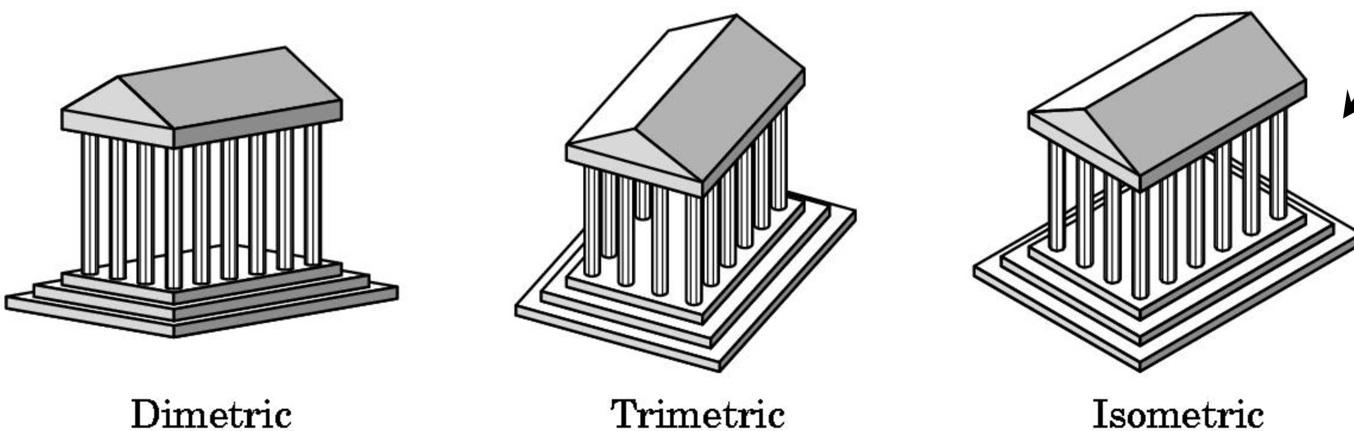
- dop **not** co-linear to one of the coordinate axes
- parallel lines remain parallel, angles are not maintained
- special cases:
  - **isometric** projection: same angle of dop to all coordinate axes
  - **dimetric** projection: same angle of dop to only two coordinate axes
  - **trimetric** projection: angle of dop to coordinate axes is different for all axes



Axonometric Parallel Projections



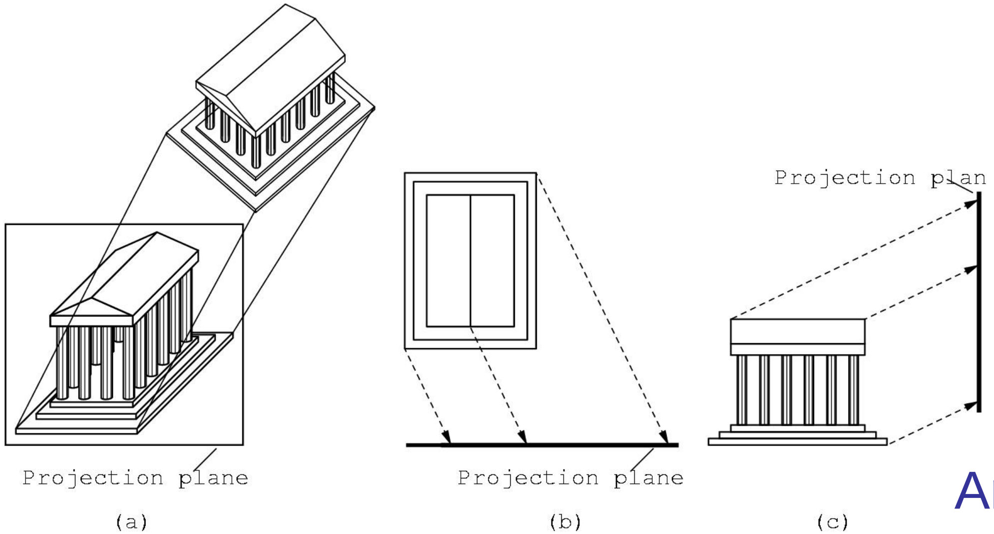
isometric projections



Angel (2000)

Oblique Parallel Projections

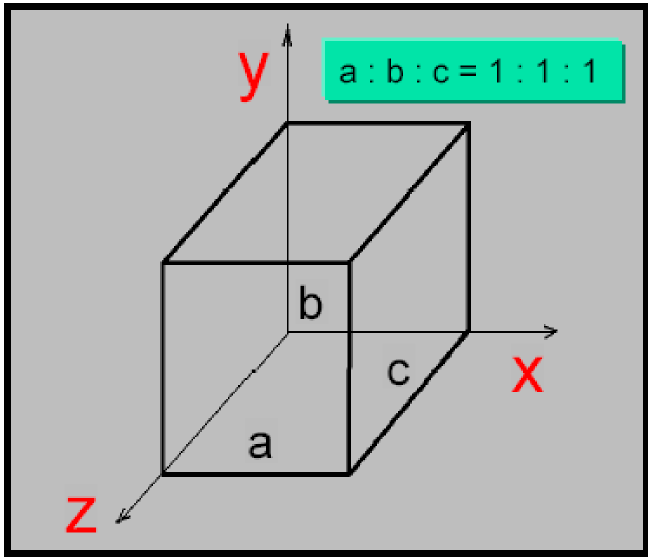
- direction of projection (dop) is not perpendicular to view plane normal (vpn)
- better spatial perception than with orthographic projections
- view plane usually perpendicular to one system plane

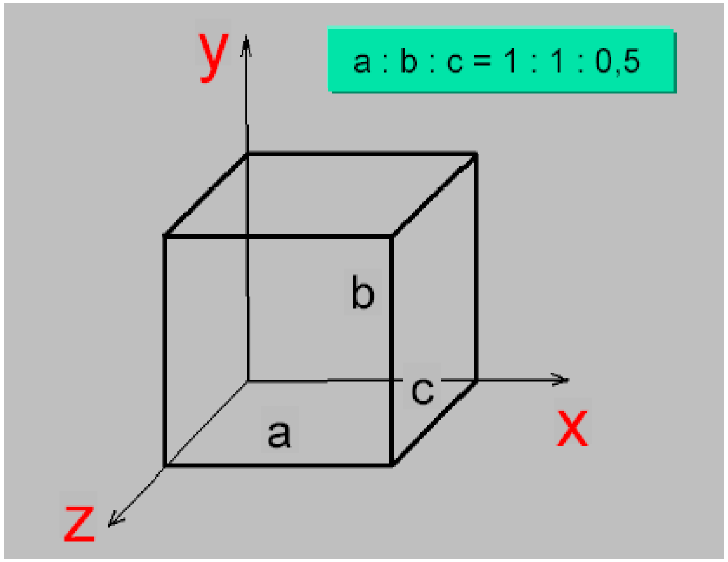


Angel (2000)

Oblique Parallel Projections

- Cavalier projection: 45° angle between dop and view plane
- Cabinet projection: 63.4° angle between dop and view plane (arctan(2)), *z*-lengths shorter by ½
- these are not perspective projections!





M . Haller, FH Hagenberg, (2002)

Oblique Parallel Projections

- Cavalier projection, 45° between dop & view plane

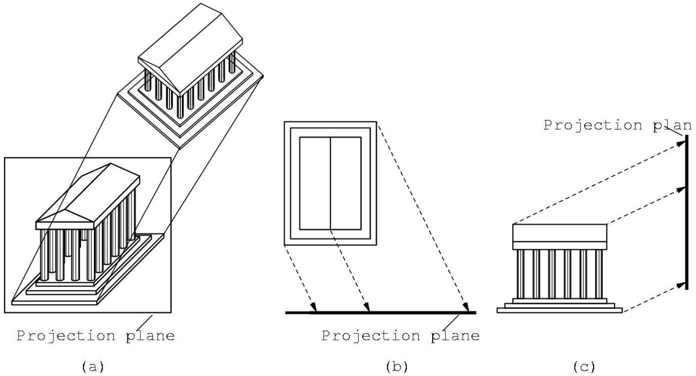
for α = 45°
and α = 30°

Bender/Brill (2003)

- Cabinet projection, 63.4° between dop & view plane

for α = 45°
and α = 30°

Bender/Brill (2003)



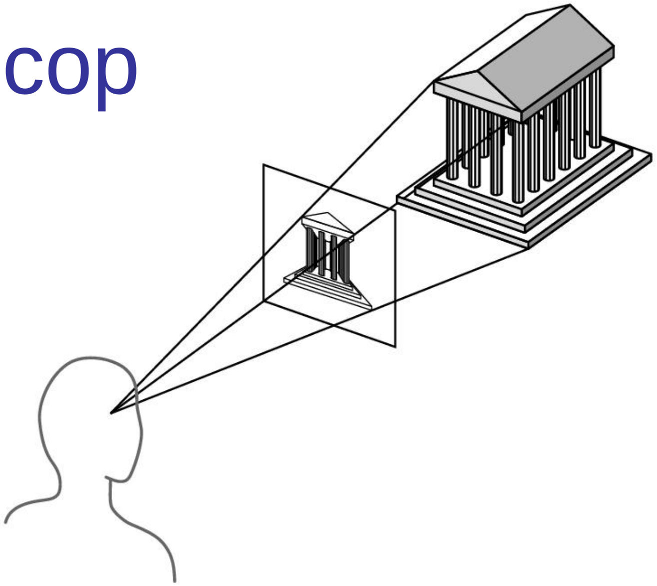
Angel (2000)

Projections

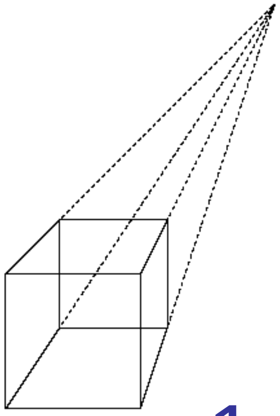
Perspective Projections

Perspective Projections

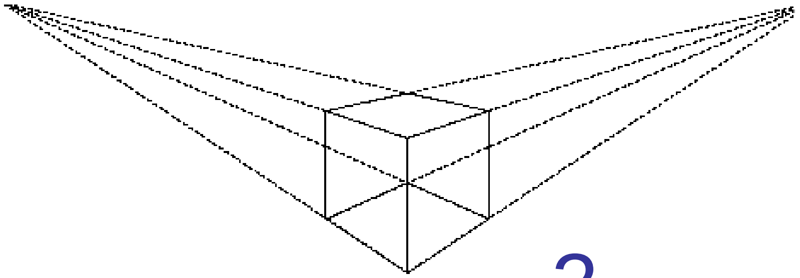
- viewer placed into cop
- cop not at infinity
- classification by vanishing points



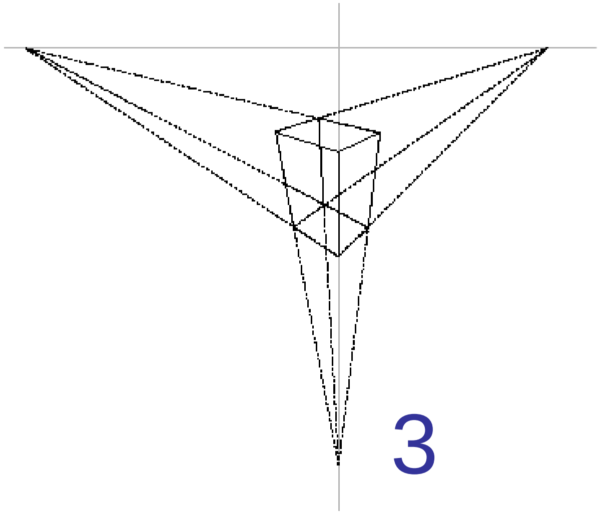
Angel (2000)



1



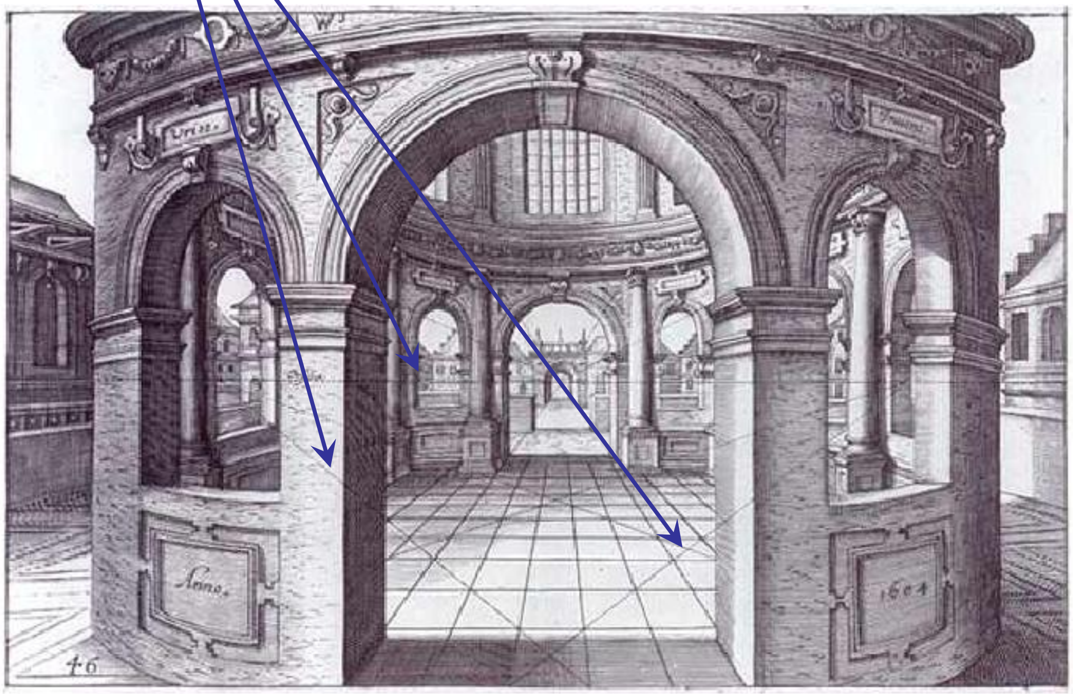
2



3

Perspective Projection

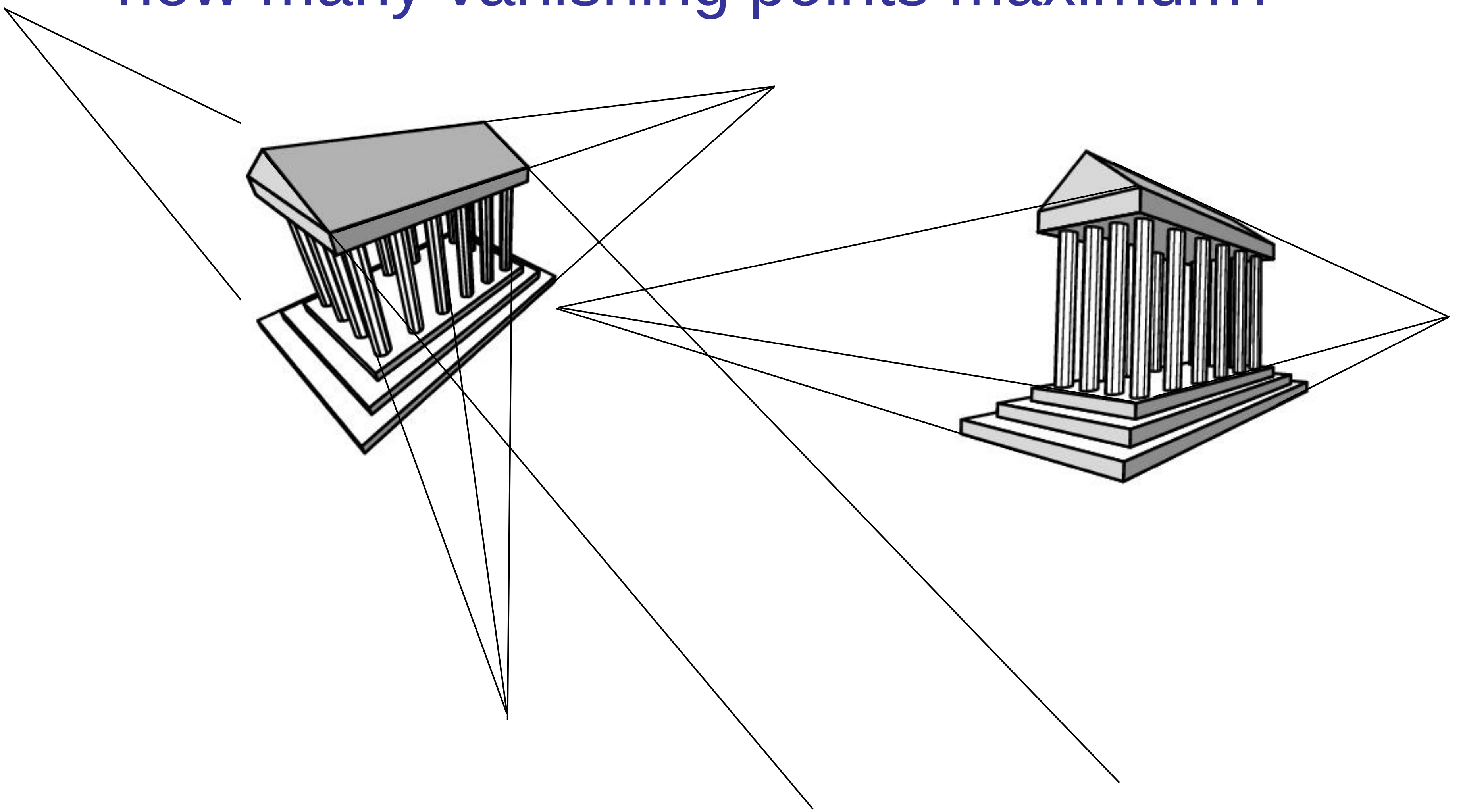
lines to construct the vanishing points, for correct drawing



Hans Vredeman de Vries: Perspektiv 1604

Perspective Projection: Vanishing Points

- how many vanishing points maximum?



Vanishing Points = 3 (roughly)

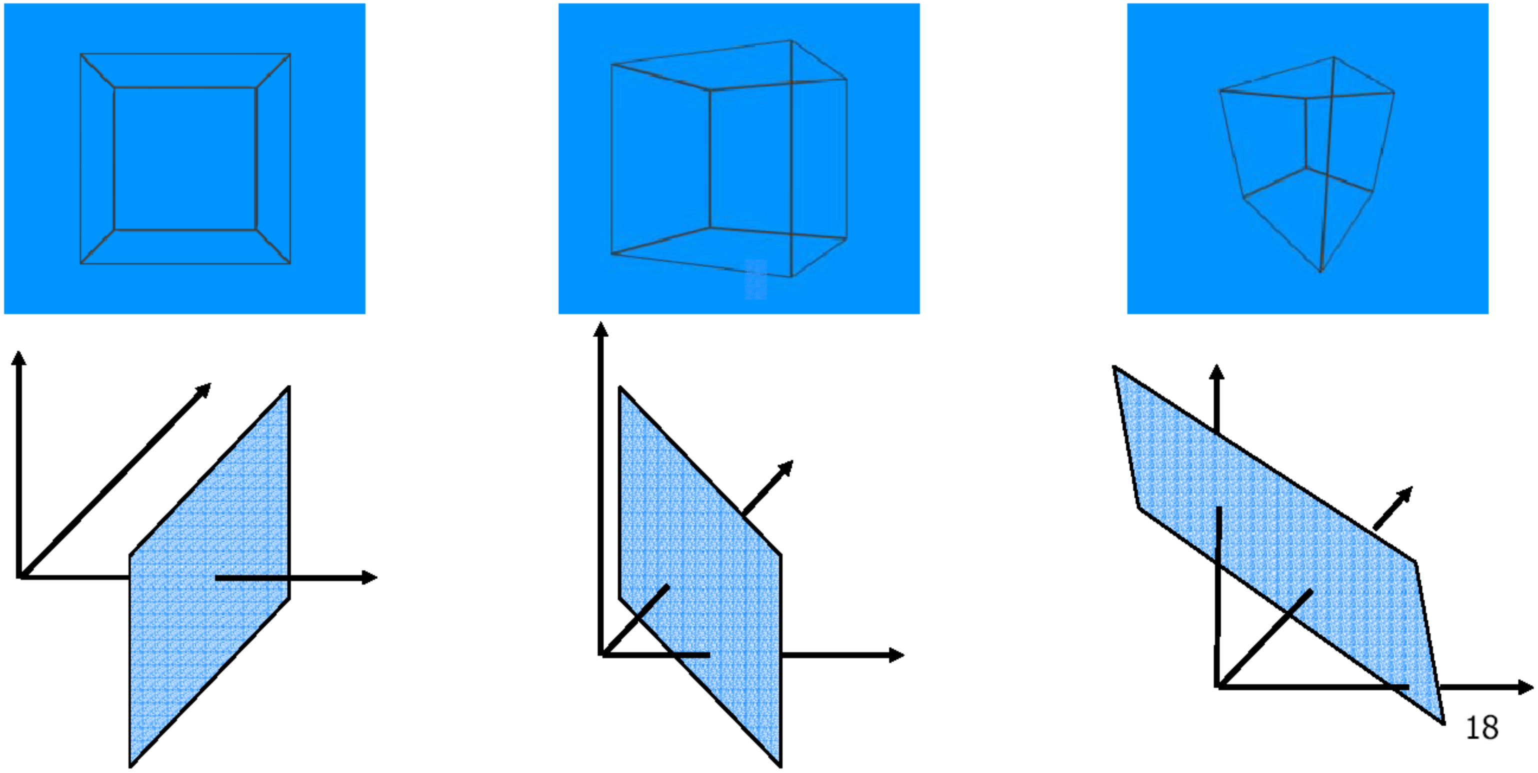
Wikipedia contributer Anthony Quintano

Vanishing Points >> 3

Wikipedia contributer Chensiyuan

Perspective Projection: Vanishing Points

- view plane in relation to coordinate axes



M . Haller, FH Hagenberg, (2002)

Perspective Projections

- foreshortening in all cases
- parallel lines and angles usually not preserved
- vanishing points depend on position of view plane with respect to coordinate axes
- **three** vanishing points if view plane intersects all coordinate axes,
  two if only two axes are intersected,
  one if only one axis is intersected

Projections

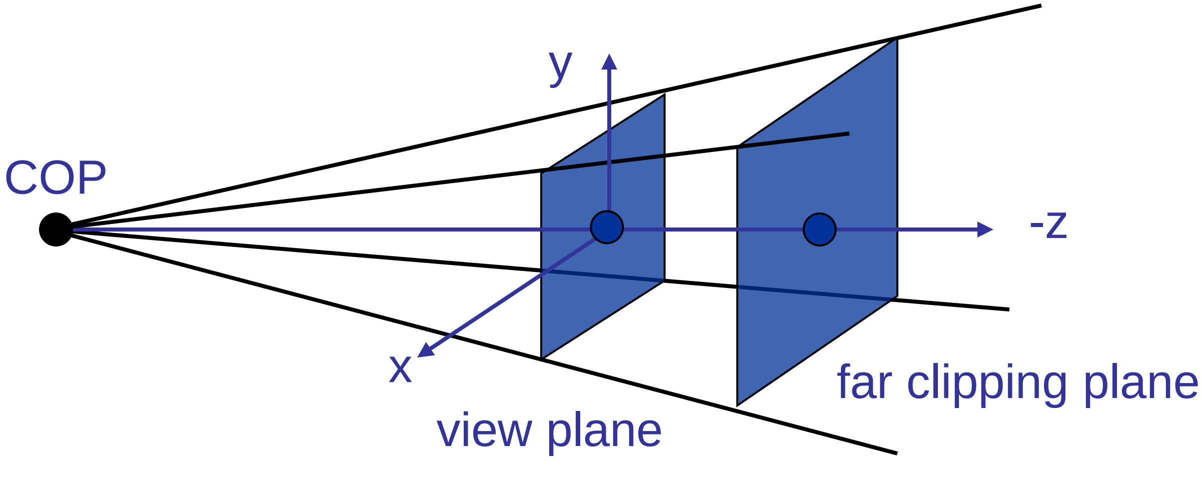
Camera Model for Computer Graphics

Perspective Projections: Camera Model

- inspired by real cameras
- parameters:
  - position, orientation
  - aperture
  - shutter time
  - focal length, lens type (zoom/wide)
  - depth of field
  - size of resulting image
  - aspect ratio (4:3, 16:9, 1.85:1, 2.35:1, 2.39:1)
  - resolution (digital) / granularity (analog)

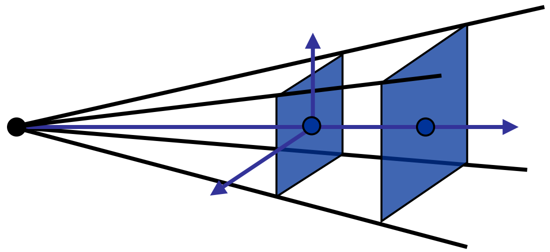
Perspective Projections: Camera Model

- simplified model in CG
  - position: point in 3D (= cop)
  - view direction: vector in 3D (vpn)
  - image specification: viewport
  - clipping planes for cutting off near and far objects (near and far clipping plane)



Perspective Projections: Camera Model

- differences between real and CG camera?



Pinhole Camera

17th century illustration

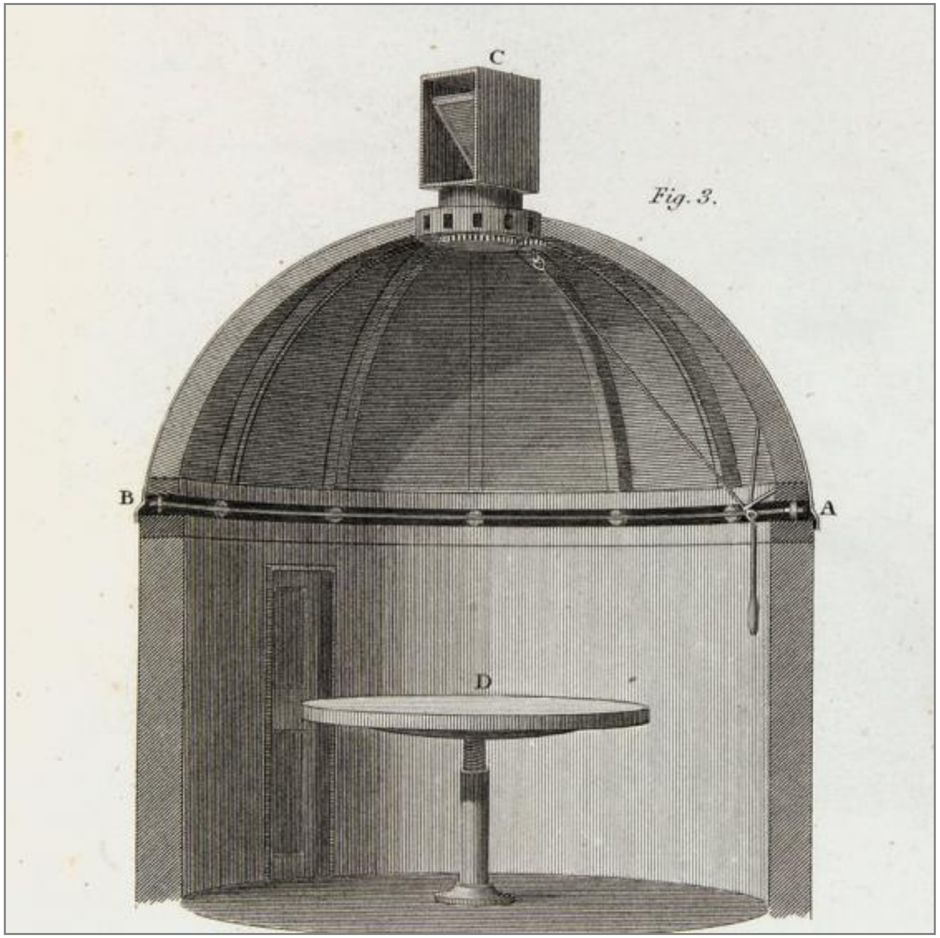
Pinhole Camera

Ulli Purwin

Pinhole Camera

Cardboard Pinhole Hasselblad

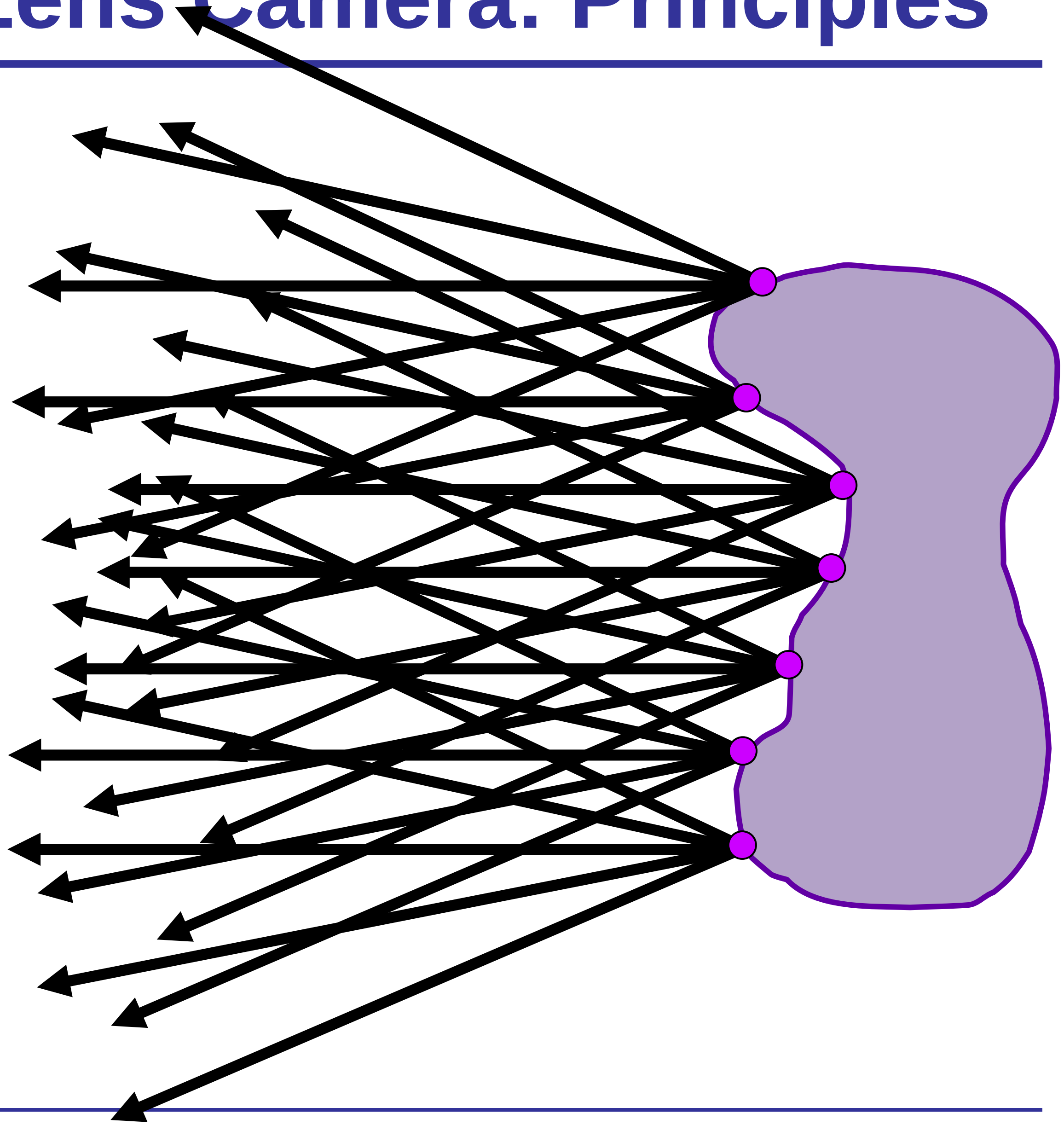
Pinhole Camera



Camera Obscura, Greenwich Royal Observatory

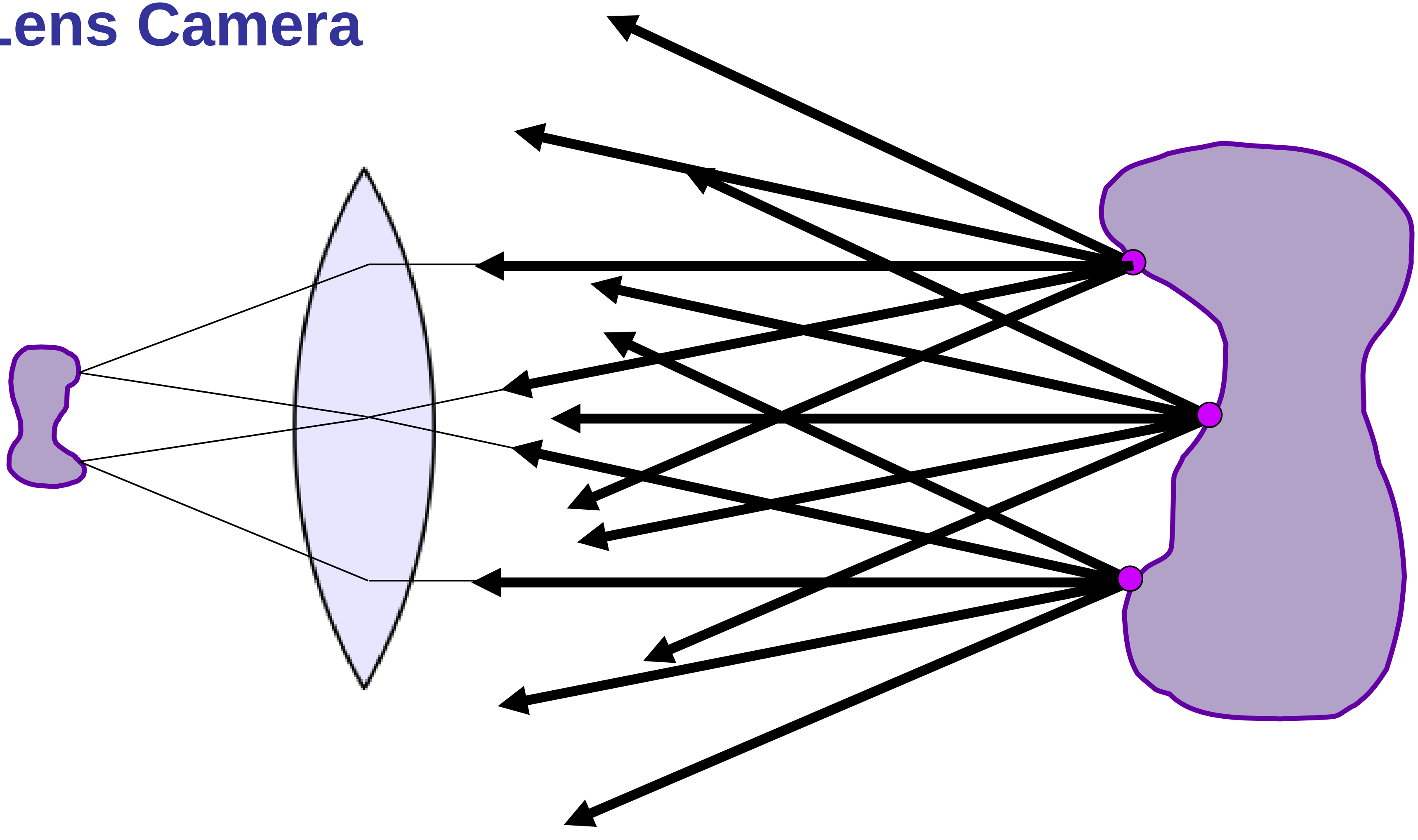
Pinhole vs. Lens Camera: Principles

Plenoptic function or **light field**



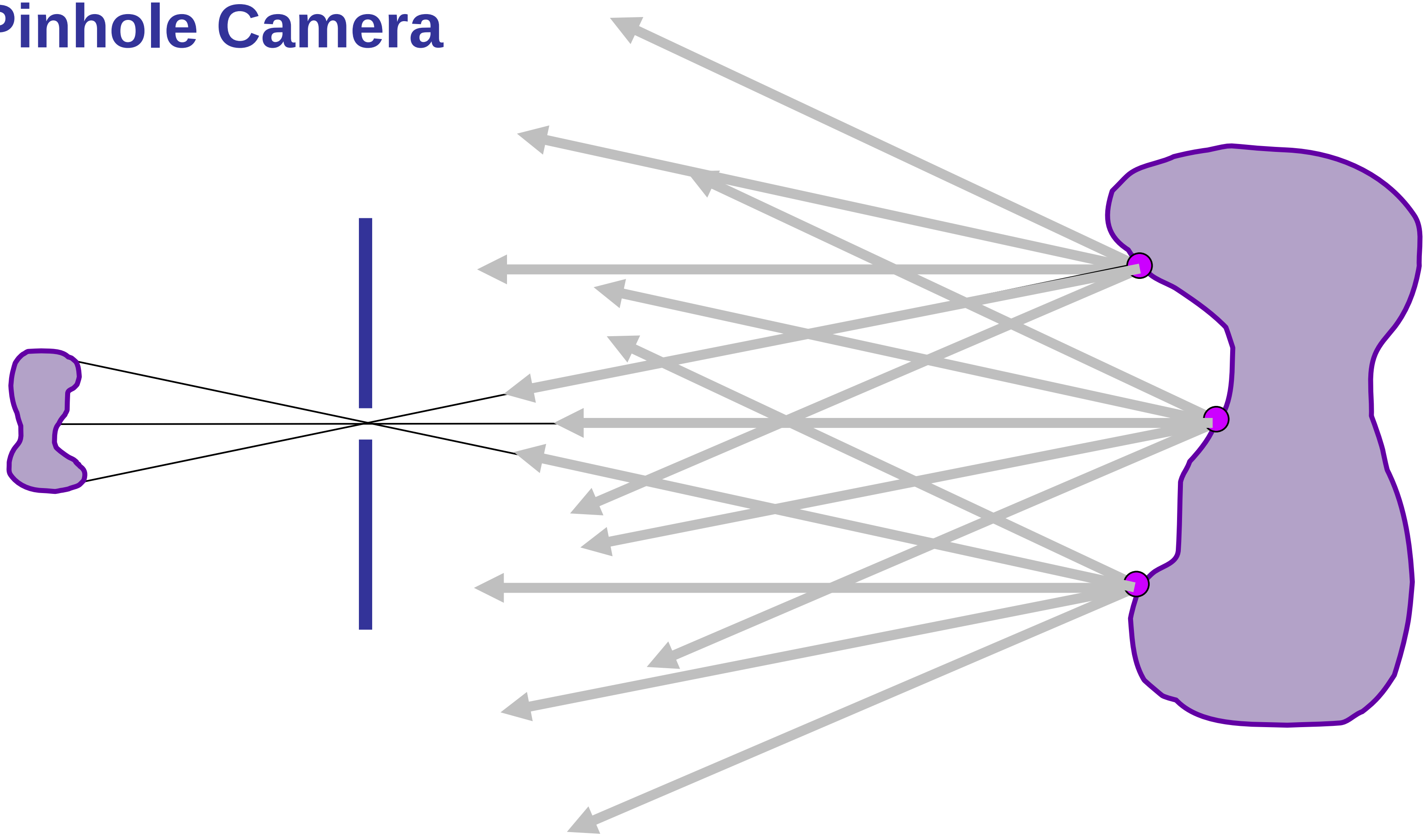
Pinhole vs. Lens Camera: Principles

Lens Camera



Pinhole vs. Lens Camera: Principles

Pinhole Camera



Lens Flare

Lens Flare

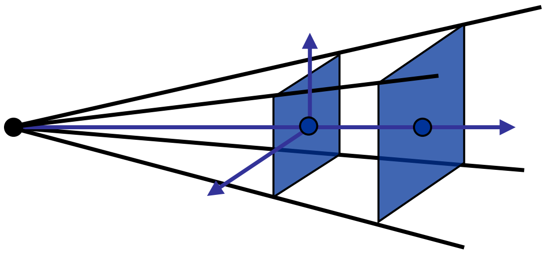
Motion Blur

Motion Blur

cg, with motion blur **cg, without motion blur**

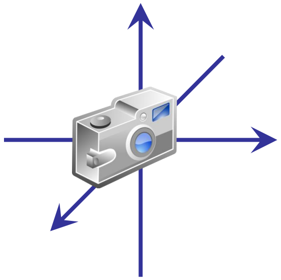
Perspective Projections: Camera Model

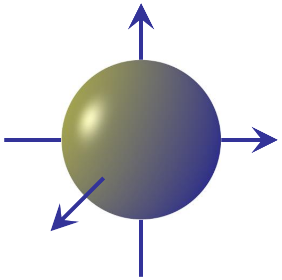
- differences between real and CG camera?
  - position of view plane w.r.t. COP
  - orientation of the image
  - type of camera (pinhole vs. lens)
    - type of “refraction”
    - lens effects (lens flare)
    - depth of field: none vs. existing
  - types of possible projections
  - picture taking times: 0 vs. >0
  - existence of far clipping plane
  - shape of view volume



Camera Model: Realization

- recall: model-view-transformation:
  - transformation from model coordinate system into camera coordinate system in one step (without projection)
  - there is no world coordinate system
  - models are usually specified in or w.r.t. the point of origin of the coordinate system
  - updated as different objects are processed



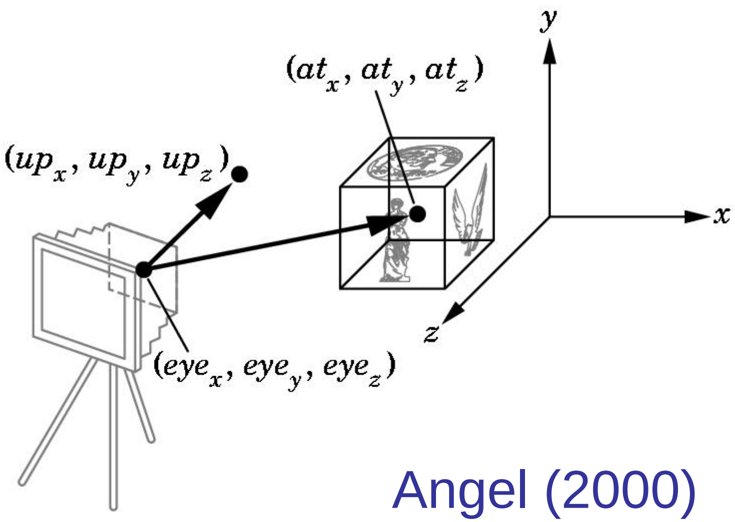


Camera Model: Realization

- model-view transformation: represented as matrix in homogeneous coordinates
- identity matrix by default
- has to be modified to
  - arrange objects in the scene
  - move camera so that scene is visible
  - different for each object at each frame of animation
- after model-view transformation: projection

Camera Model: Realization

- camera specification
  - position of the camera (cop)
  - view direction or look-at point
  - up direction to specify camera coordinate system – why?
  - computation of perpendicular basis for camera coordinate system using cross products – how?



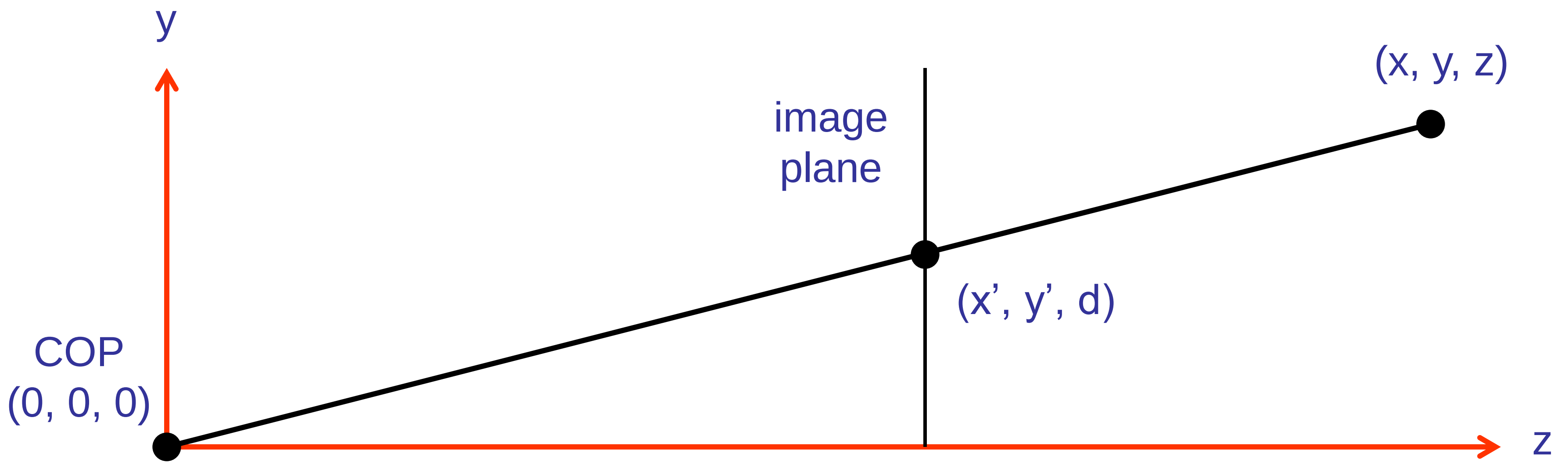
Angel (2000)

Projections

Computing the Projection

Perspective Projections: Math

- view plane parallel to *x*-*y*-plane, distance d
- COP at origin (0, 0, 0); view plane
- input: point (x, y, z) to be projected
- projected point: x', y', d

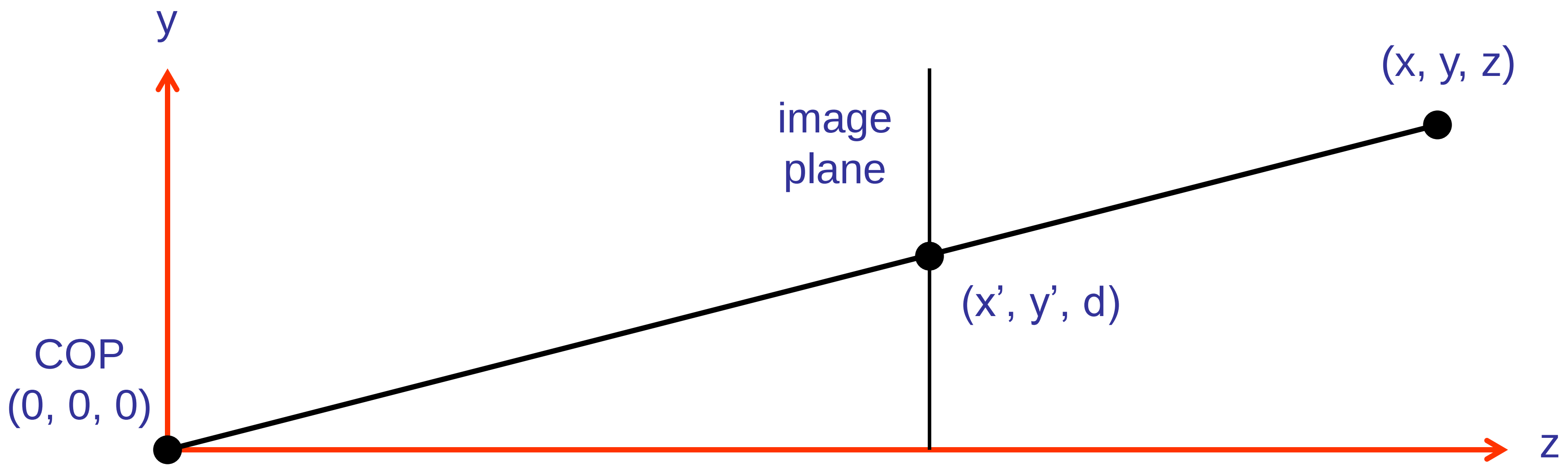


Perspective Projections: Math

- projection line: parametric equation

$$\frac{x'}{d} = \frac{x}{z} \qquad x' = \frac{d \cdot x}{z} = \frac{x}{z/d}$$

$$\frac{y'}{d} = \frac{y}{z} \qquad y' = \frac{d \cdot y}{z} = \frac{y}{z/d}$$



Perspective Projections: Math

- description as projection matrix

$$x' = \frac{d \cdot x}{z} = \frac{x}{z/d}$$

$$y' = \frac{d \cdot y}{z} = \frac{y}{z/d}$$

- matrix in homogeneous coordinates

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \Leftrightarrow \begin{pmatrix} x' \\ y' \\ d \\ 1 \end{pmatrix}$$

NEVER set the distance of front clipping plane to camera position to zero!!!

Parallel Projections: Math

- view plane placed into *x*-*y*-plane
- projectors parallel to *z*-axis
- thus, *z*-values are projected to 0
- homogeneous matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 0 \\ 1 \end{pmatrix}$$

Other Parallel Projections

- oblique projection

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \frac{-\cos(\alpha)}{\tan(\beta)} & 0 \\ 0 & 1 & \frac{\sin(\alpha)}{\tan(\beta)} & 0 \\ 0 & 0 & \frac{-1}{\sin(\beta)} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{-\cos(\alpha)}{\tan(\beta)} & 0 \\ 0 & 1 & \frac{\sin(\alpha)}{\tan(\beta)} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Other Parallel Projections

- Cavalier projection: $\beta = 45^{\circ}$

$$P = \begin{pmatrix} 1 & 0 & -\cos(\alpha) & 0 \\ 0 & 1 & \sin(\alpha) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

no foreshortening of lengths in *z*-direction since $\cos^2\alpha + \sin^2\alpha = 1$

- Cabinet projection: $\beta = 63.4^{\circ}$

$$P = \begin{pmatrix} 1 & 0 & \frac{-\cos(\alpha)}{2} & 0 \\ 0 & 1 & \frac{\sin(\alpha)}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

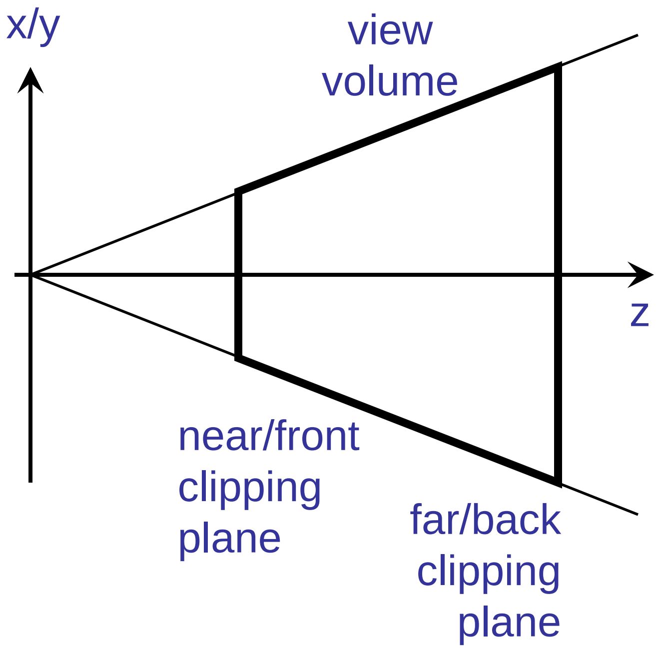
foreshortening by 2 of lengths in *z*-direction since $\cos^2(\alpha/2)+\sin^2(\alpha/2) = 1/2$

Projections: Problems

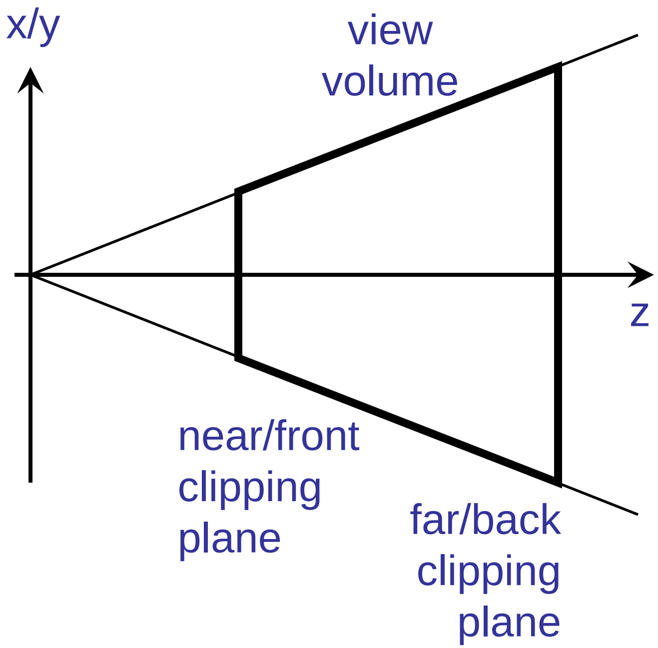
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \Leftrightarrow \begin{pmatrix} x' \\ y' \\ d \\ 1 \end{pmatrix}$$

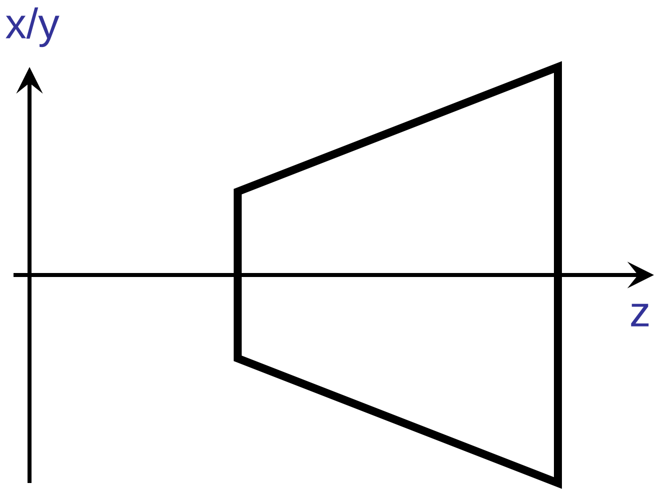
- view frustum is a pyramid shape
  $\longrightarrow$ complex clipping (expensive)
- all objects end up in a single plane
  $\longrightarrow$ no data for hidden surface removal
- solution: normalized processing with **canonical view volumes**

Projections: Canonical View Volumes

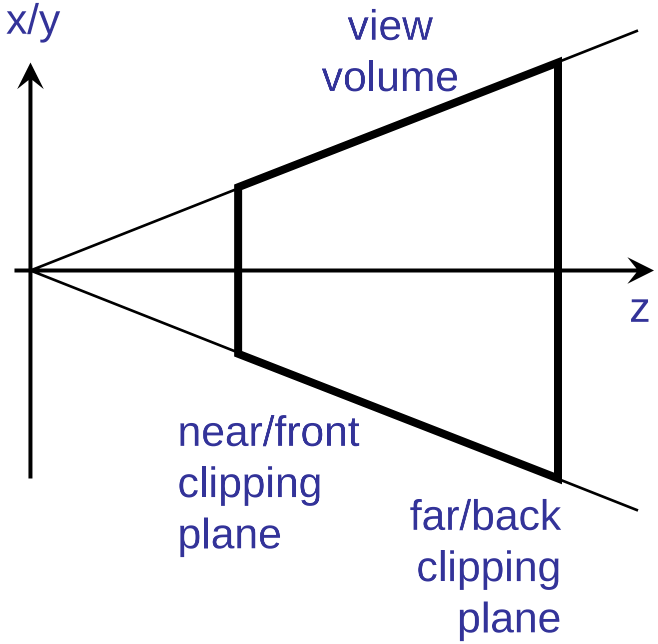


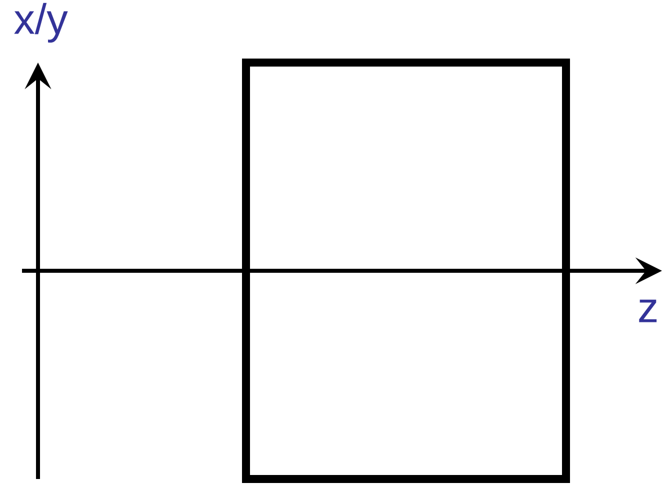
Projections: Canonical View Volumes



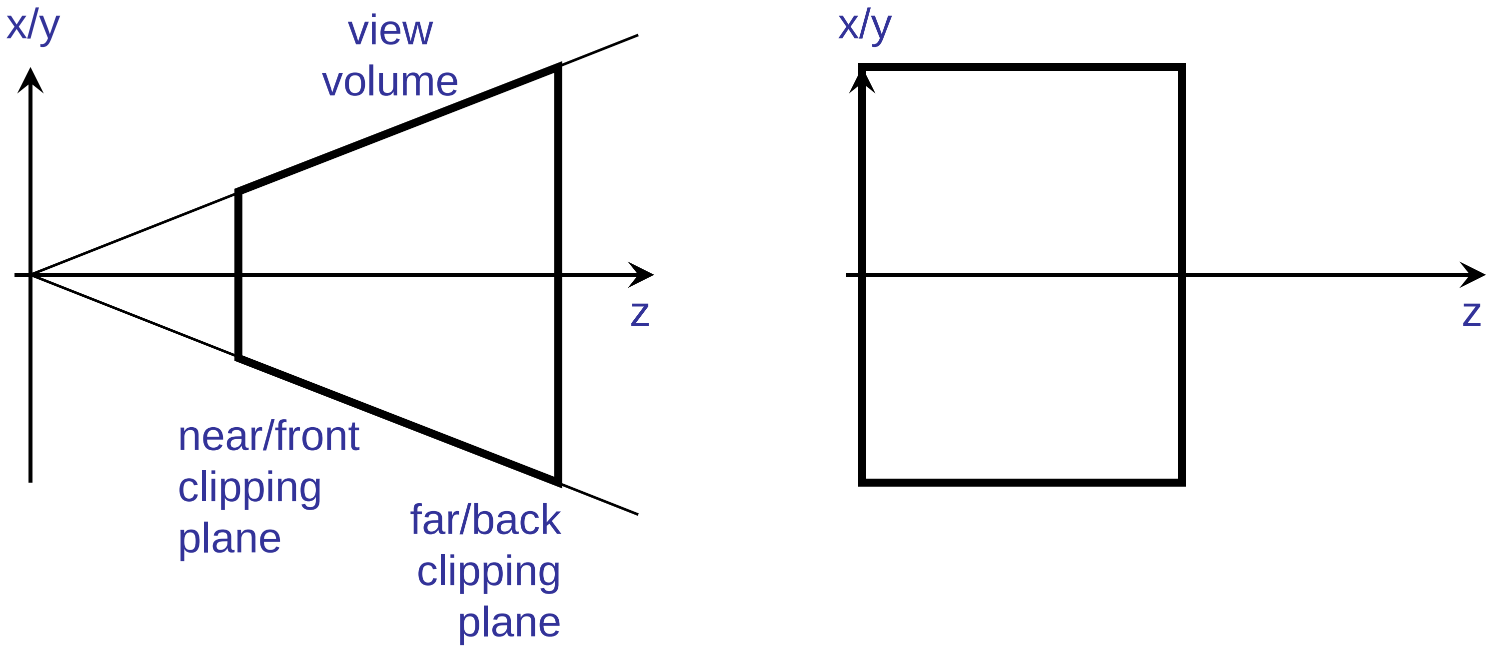


Projections: Canonical View Volumes

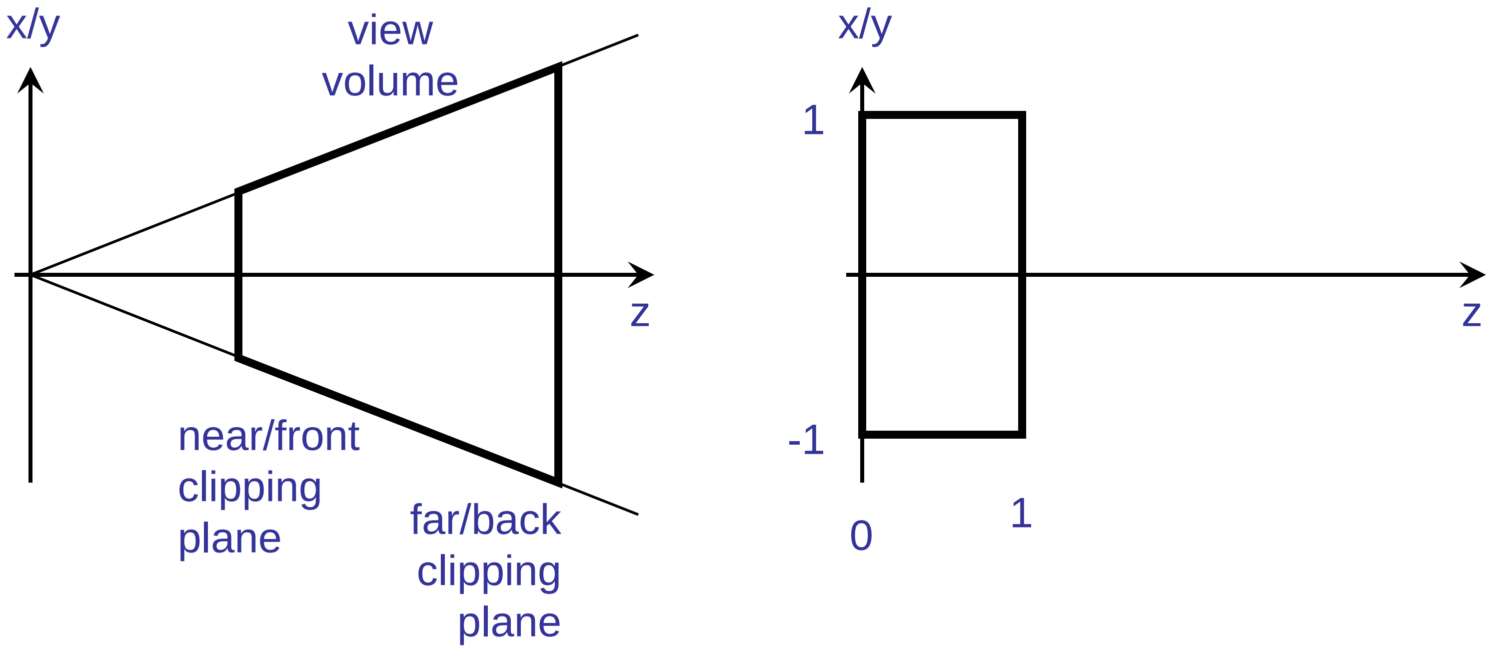




Projections: Canonical View Volumes

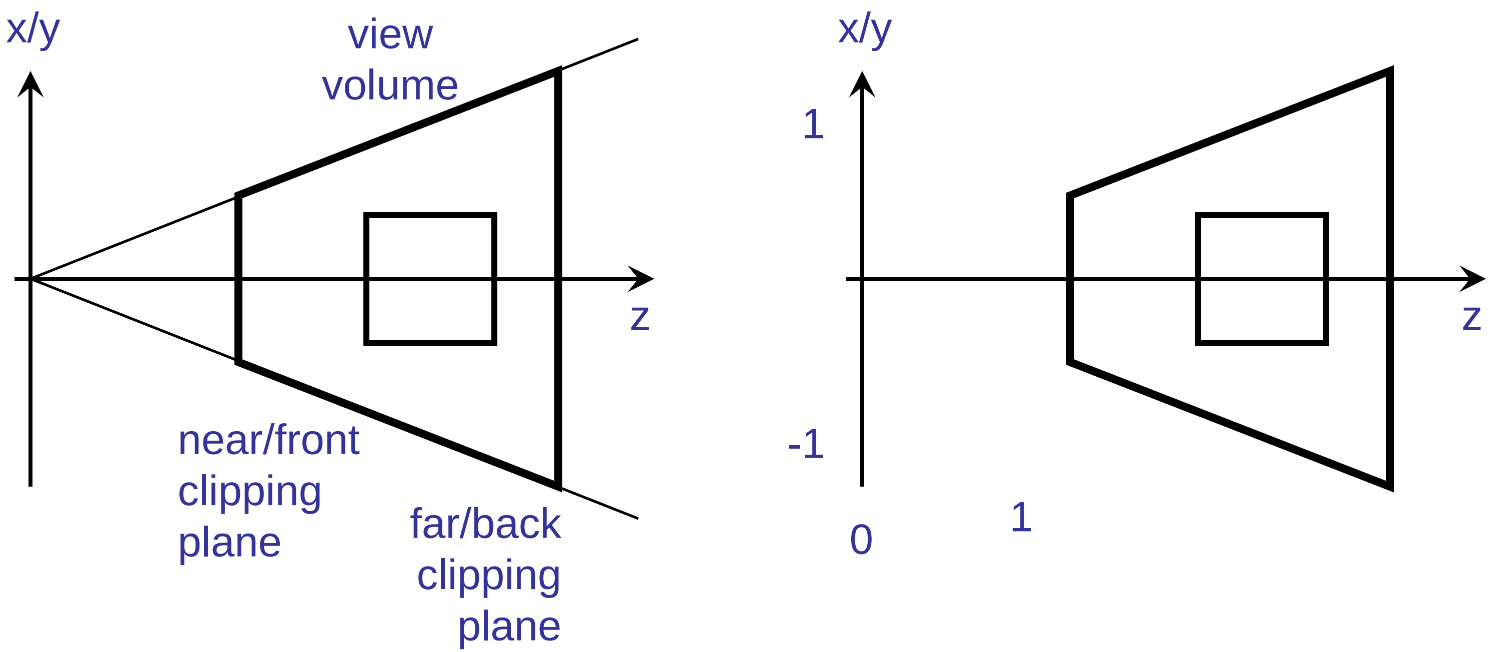


Projections: Canonical View Volumes



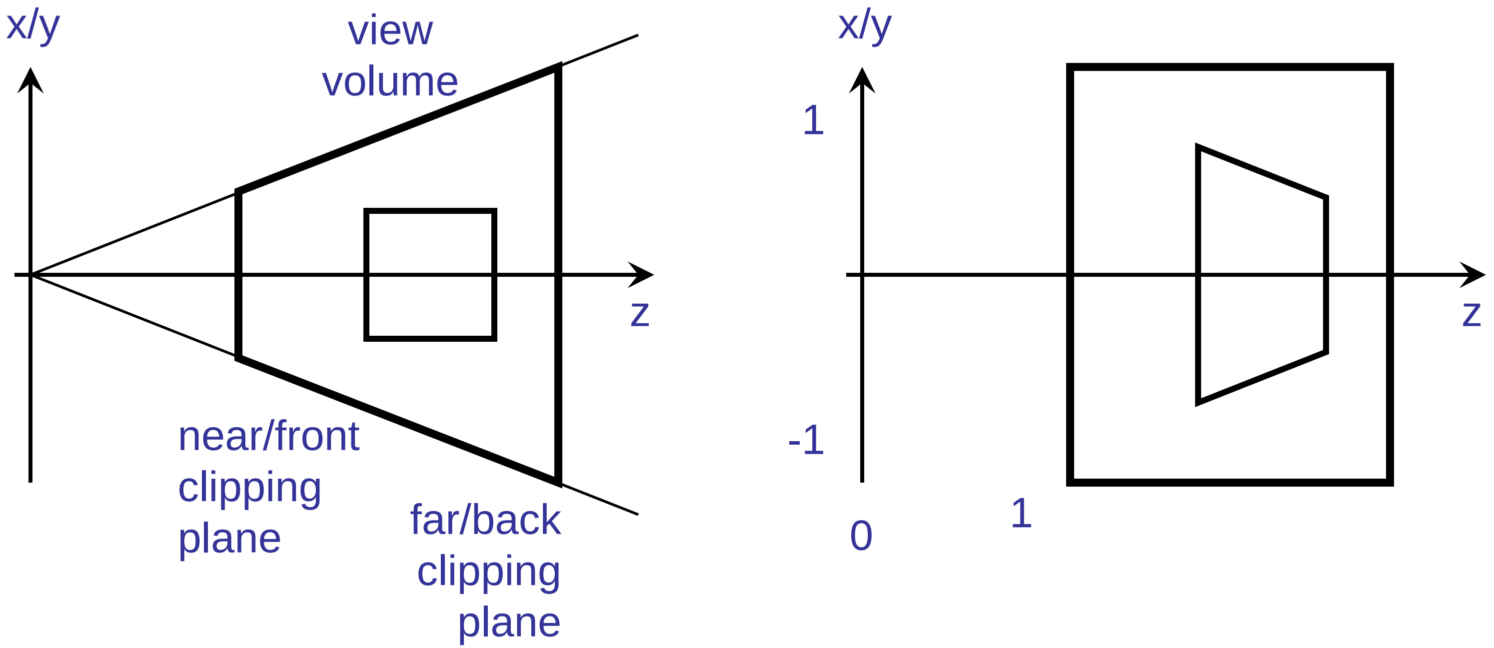
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



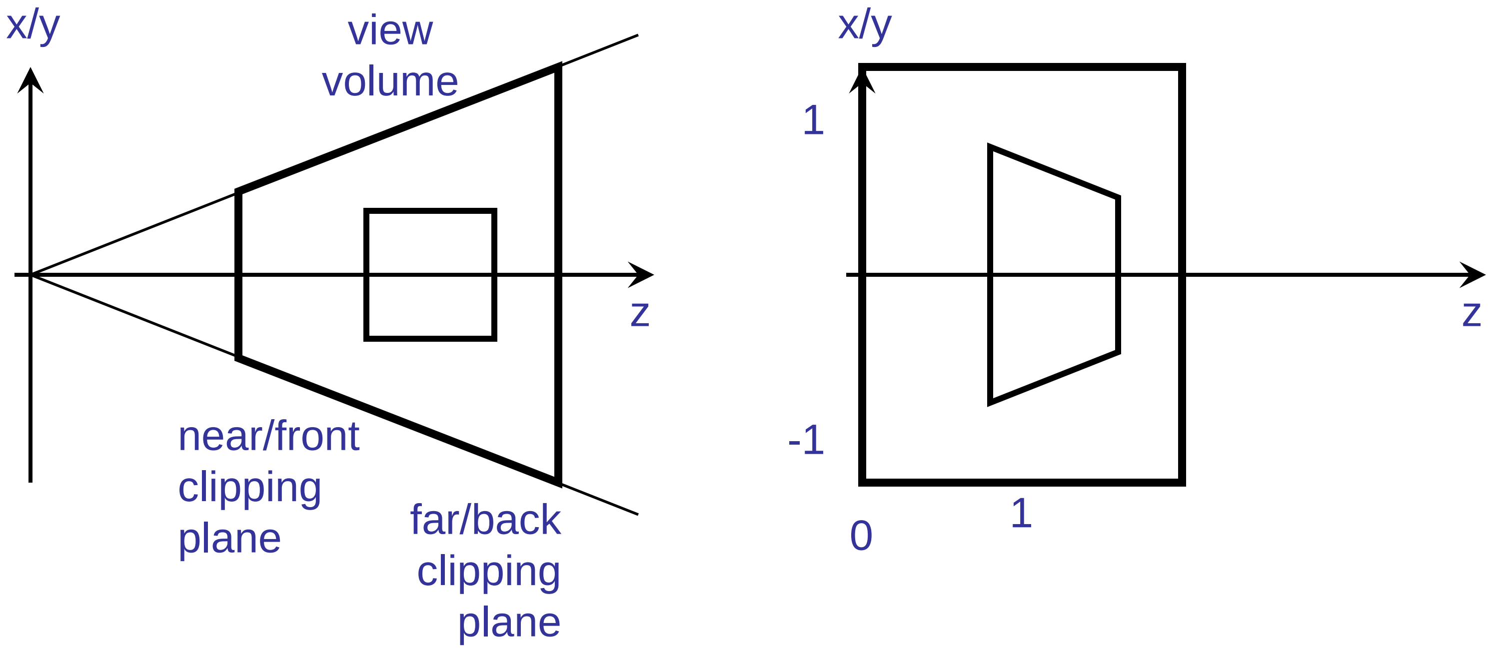
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



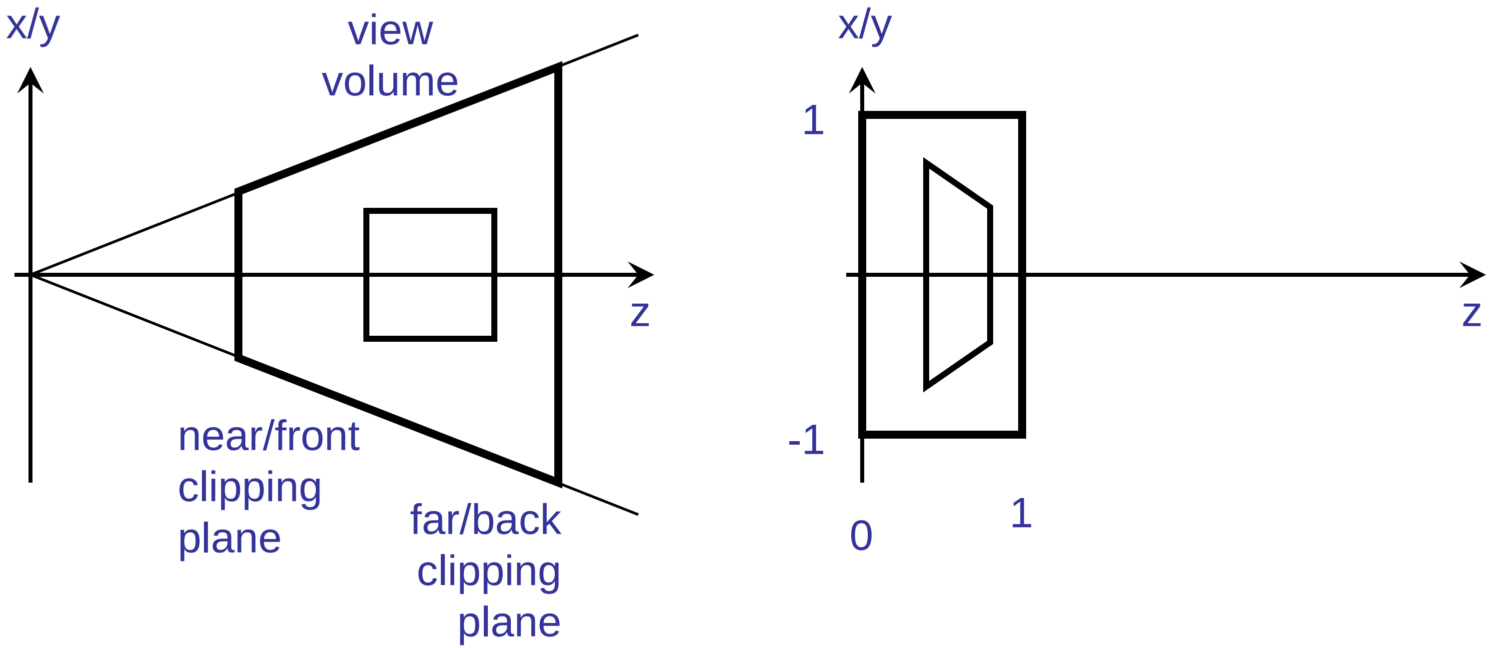
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



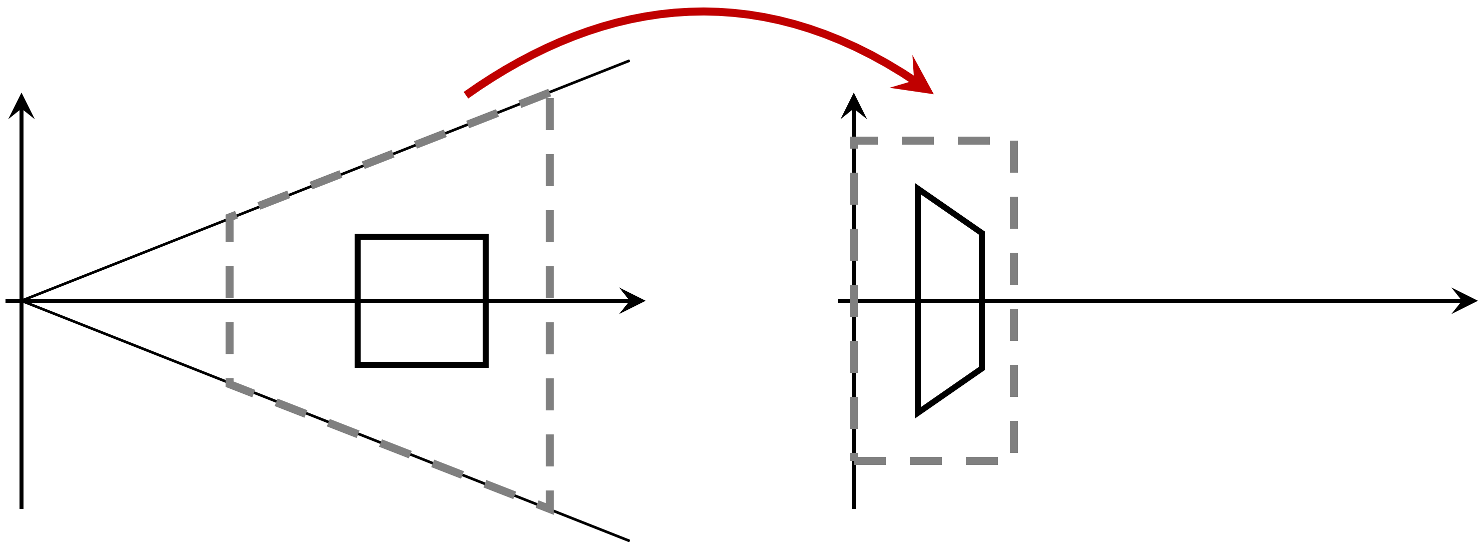
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Benefits of Canonical View Volumes



- it does not remove the **depth information** from the primitives (needed for hidden surface removal)
- after processing, the outside is nicely **box-shaped** and **aligned** to the camera coordinate system
  $\rightarrow$ better for later clipping off the “outside” geometry

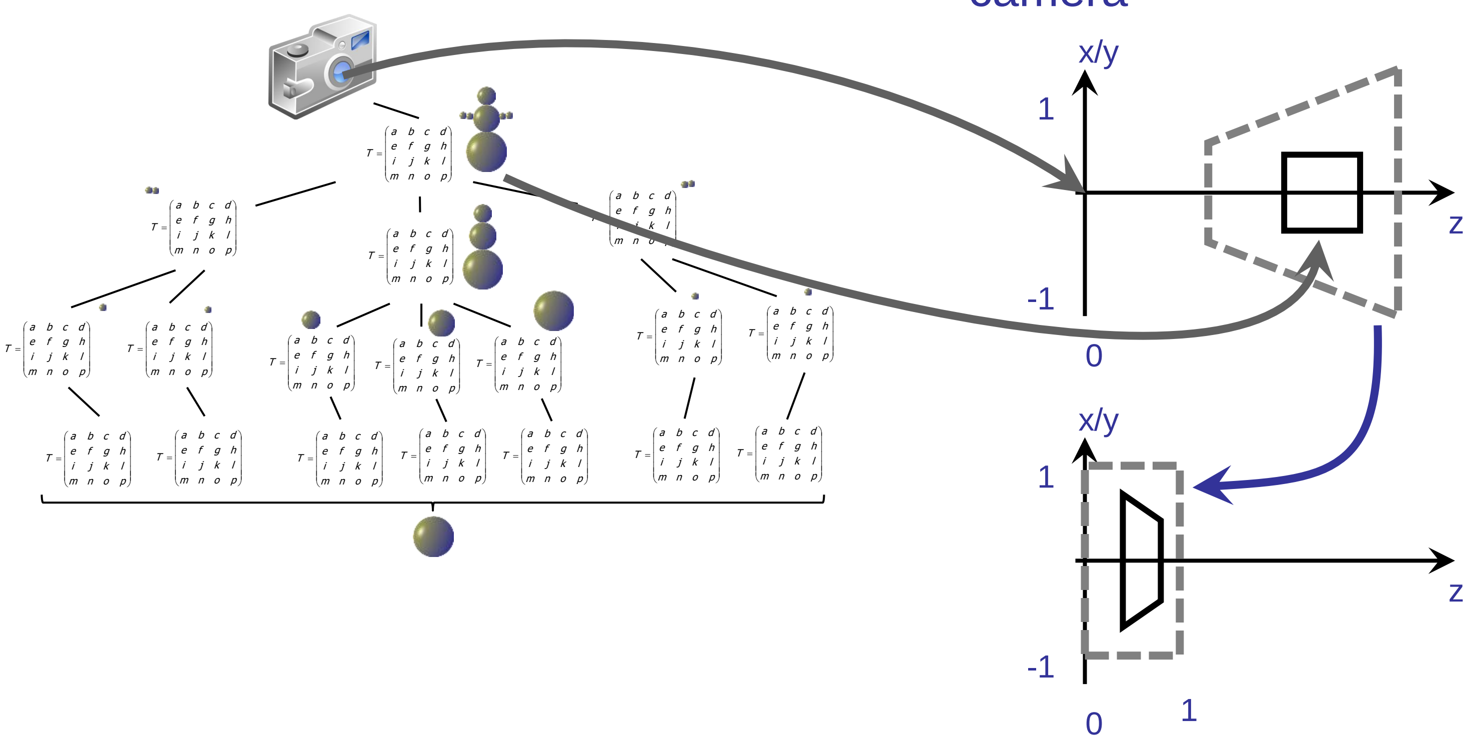
Model-View Transformation Realized

Model transformation:

- transforms from object coordinates to camera coordinates
- arranging objects w.r.t. camera
- different for every object or its parts

View transformation:

- transforms from camera coordinates to projected geometry
- perspective effects
- identical for all objects, unique to camera



Rendering Pipeline