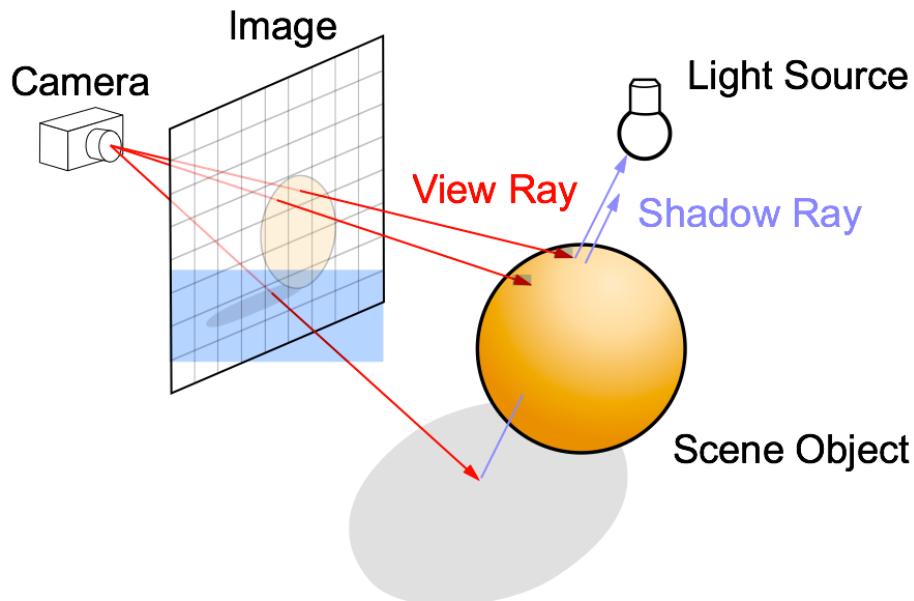
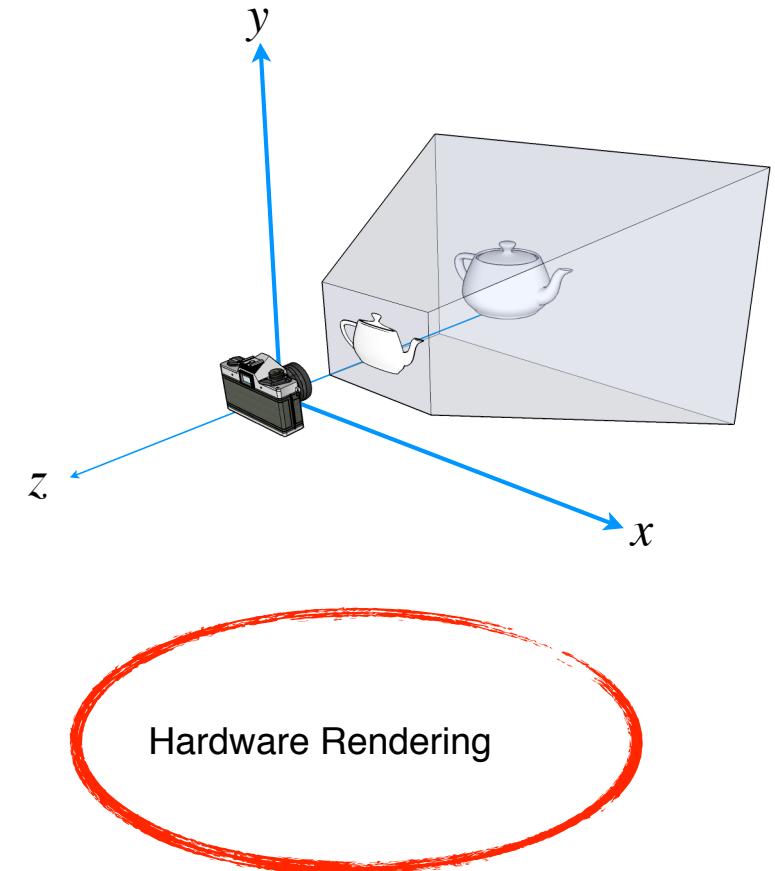


# Rendering : Hardware / Software

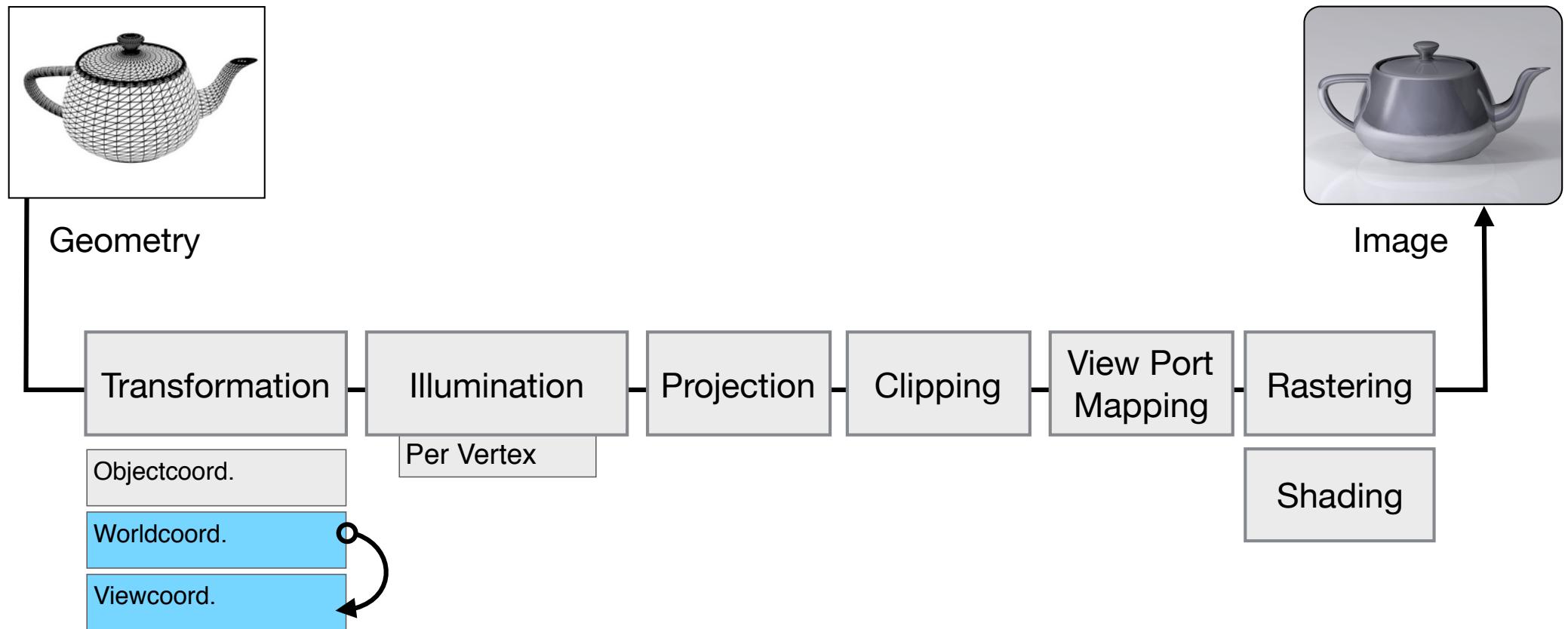


Software Rendering (Raytracing...)

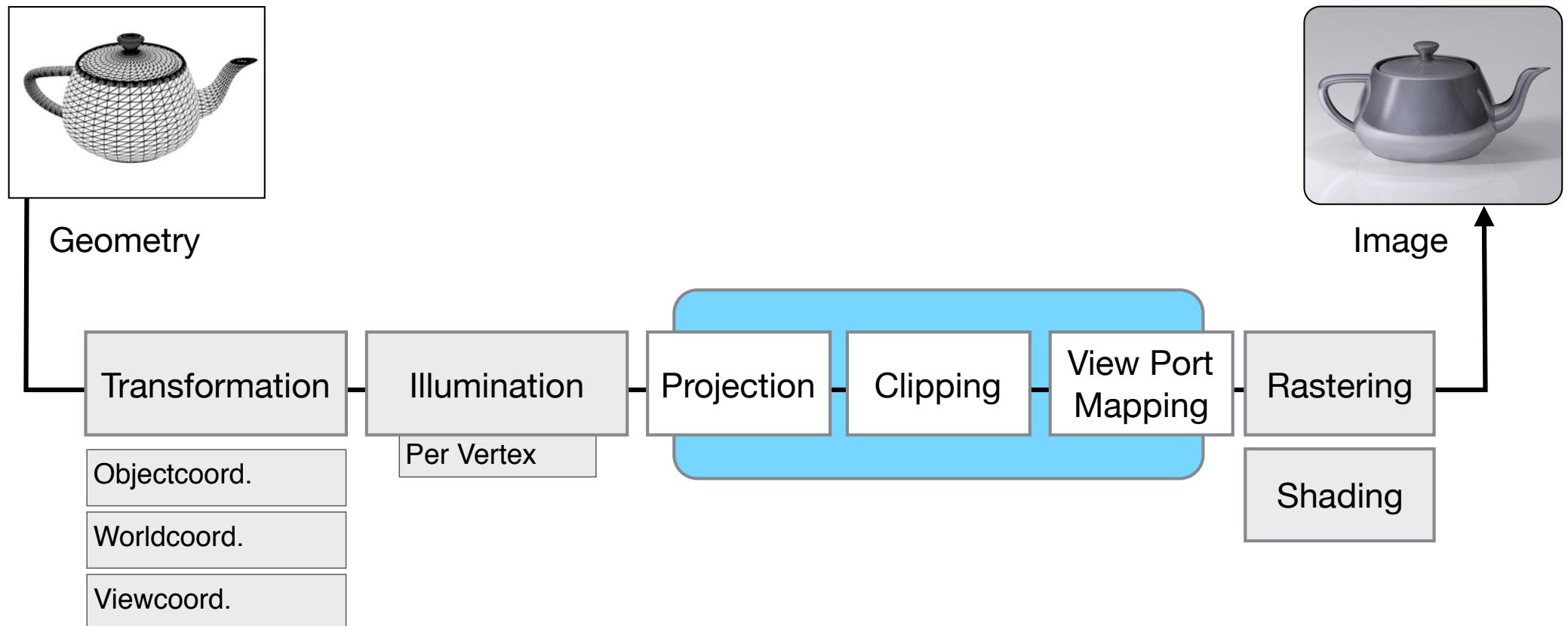


Hardware Rendering

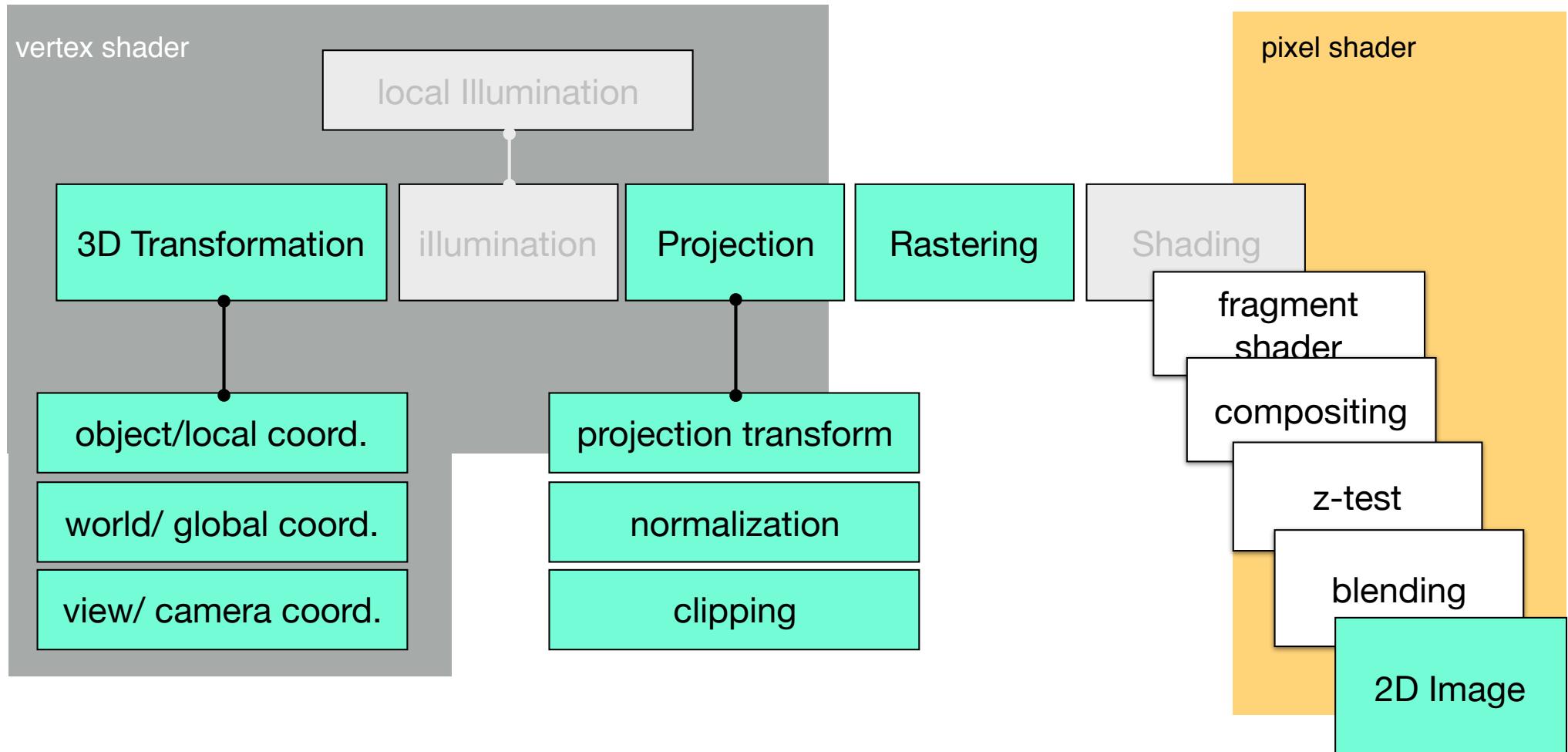
# *pipeline*



# pipeline

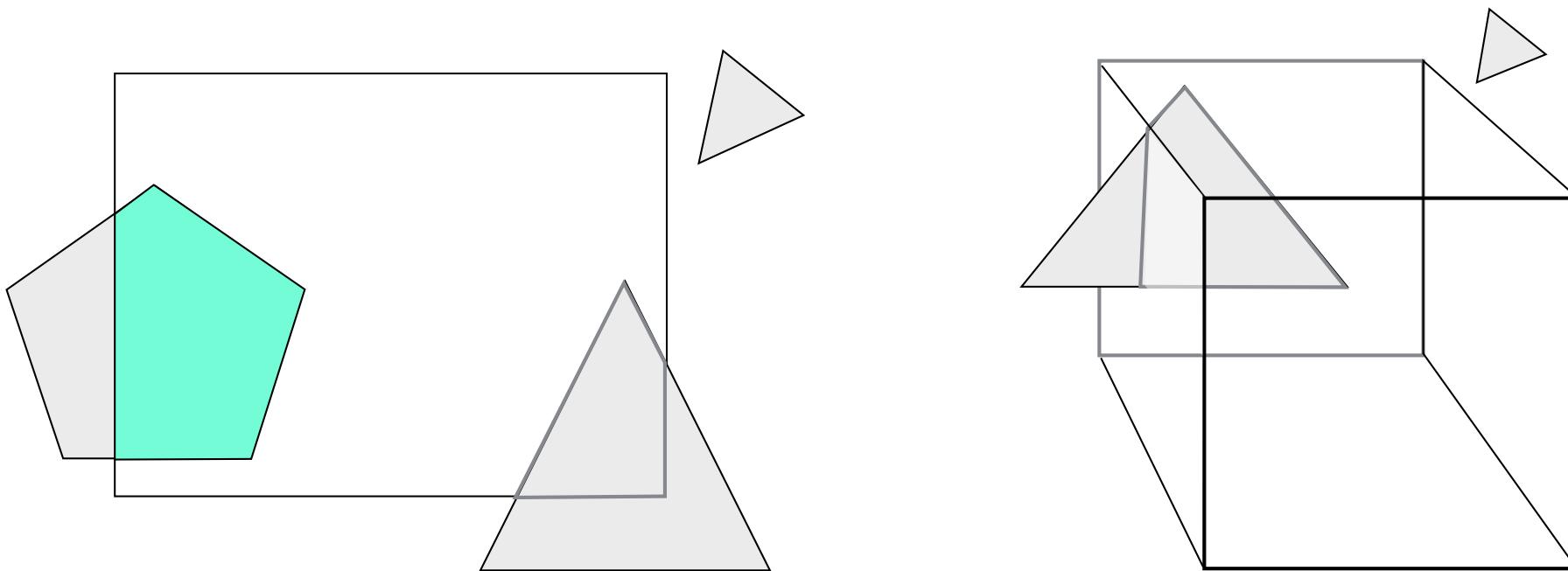


# Programmable Graphics Pipeline



## ■ Idee des Clipping in der Rendering Pipeline:

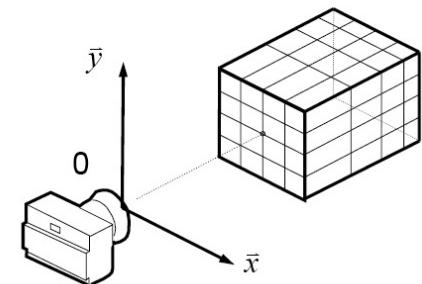
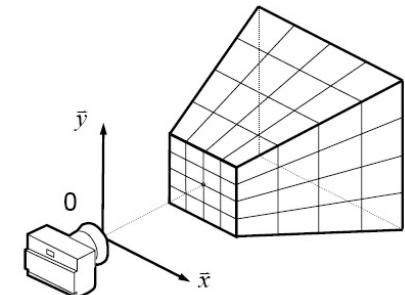
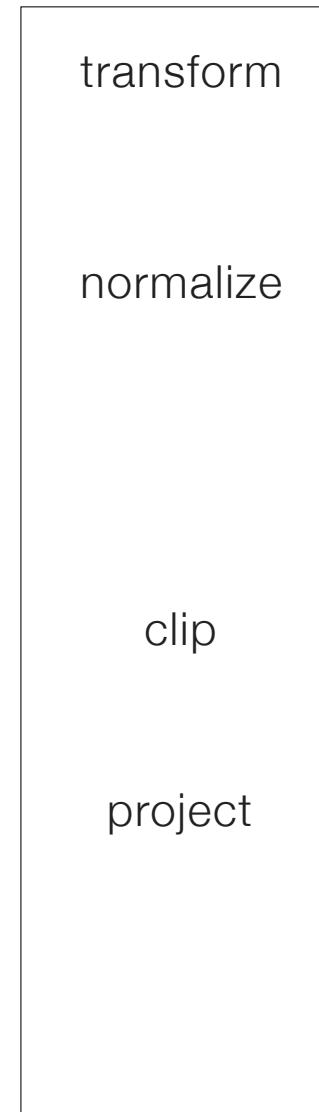
- anstatt Clipping der 2D-Objekte am Viewplane (Sichtfenster),
- Clipping der 3D-Objekte am Viewing-Frustum (Ansichtsvolumen)
- Erinnerung: Algorithmen gelten nur für achsenparallele Ebenen
- Verfahren teuer-> Überlegung : so wenig wie möglich reinschicken



# *project transform, normalize & clip*

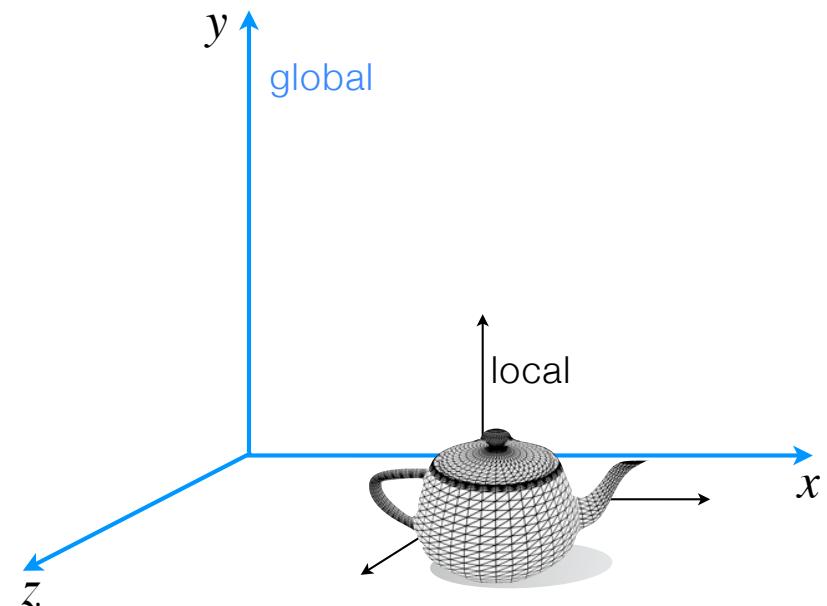
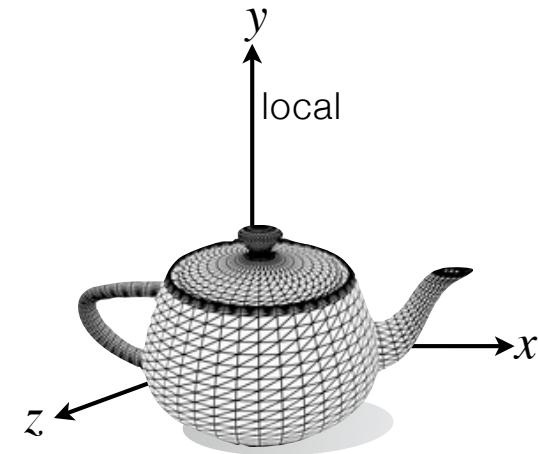
## ■ Beliebigen View Frustum vereinfachen

- Zentriert am Ursprung / Achsenparallel / -z
- Quader in Würfel verformen und zentrieren
- Würfel normieren
- Culling
  - Faces mit der Normale nach hinten weglassen
- Clipping
  - Faces komplett oder zum Teil abschneiden
- Die Projektion
  - ist dann nur noch eine orthogonale (parallel) Projektion auf die vordere Ebene des Einheitsvolumen
  - Weglassen der Z-Koordinate



# Koordinatensysteme

- Lokales (Object) Koordinatensystem
- $x, y, z$  Koordinate im Bezug auf den „Schwerpunkt“, pivot Punkt, oder lokales Koordinatensystem
- Globales (World) Koordinatensystem
- Position im Bezug auf die Weltkoordinaten
- Wo steht das Objekt in der Szene



# Koordinatensysteme

## ■ View (Kamera) Koordinatensystem

### ■ Kameradefinition

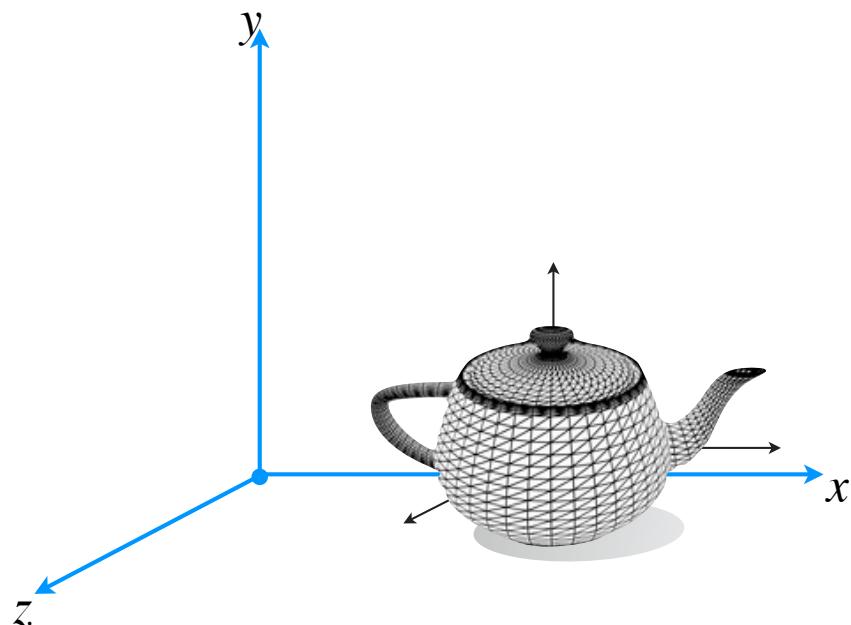
3 Fragen



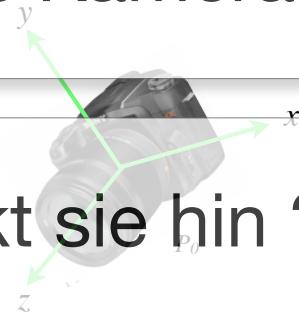
Viewpoint

Centerpoint (COI - Center of interest)

up Vector



Wo ist die Kamera ?

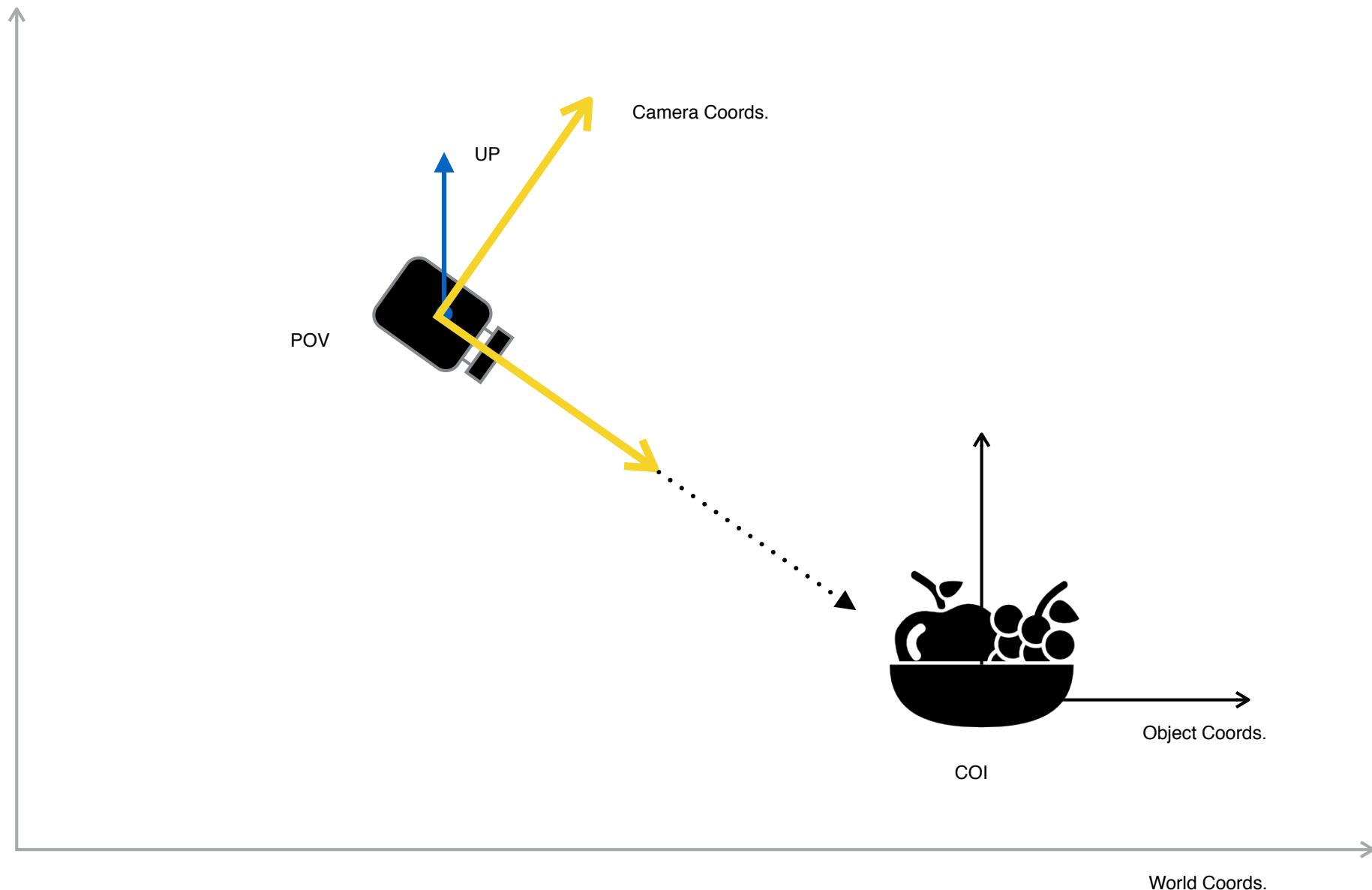


Wo guckt sie hin ?

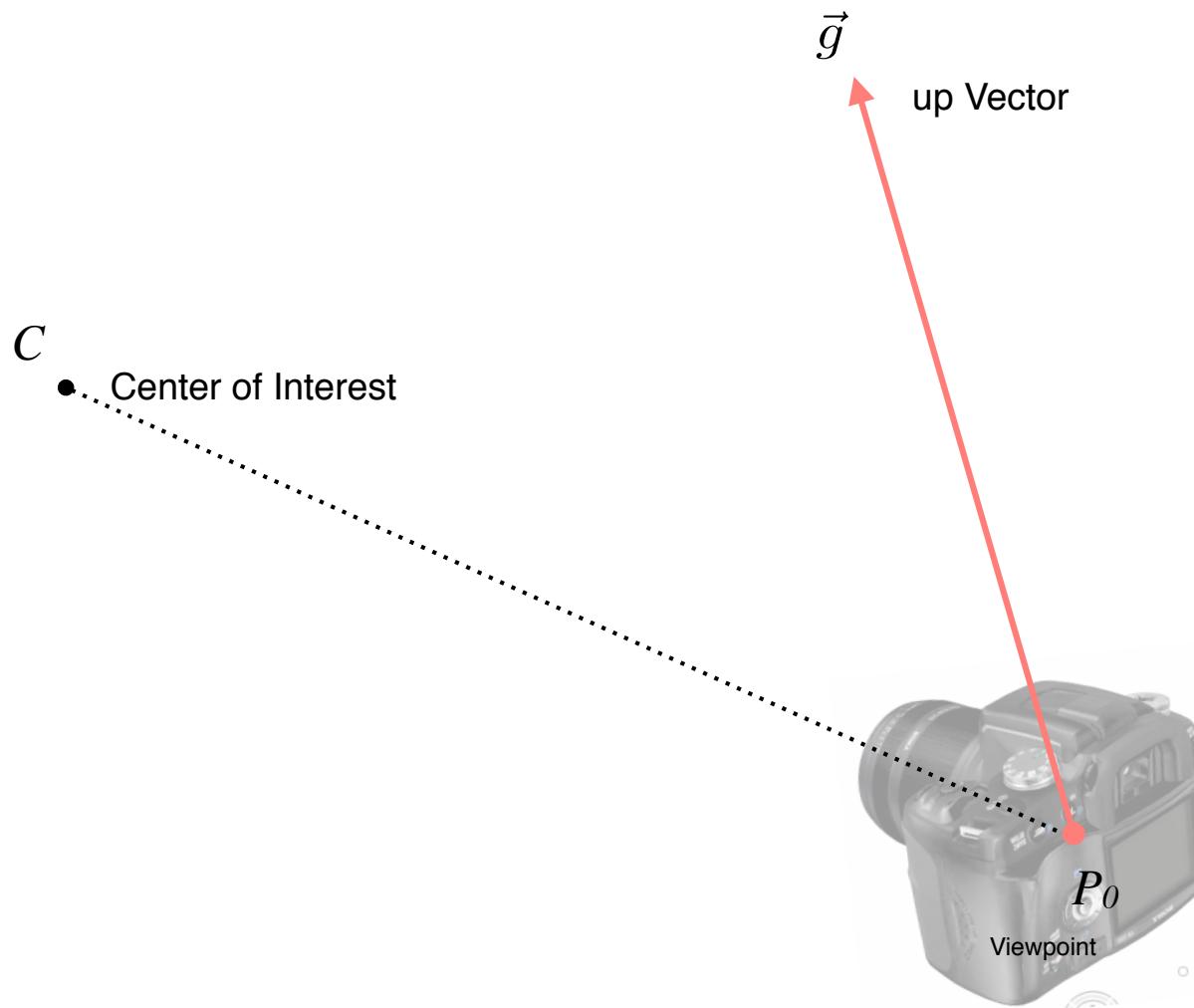
Wo ist oben ?

Aus diesen Parametern wird das Kamerakoordinatensystem „bauen“

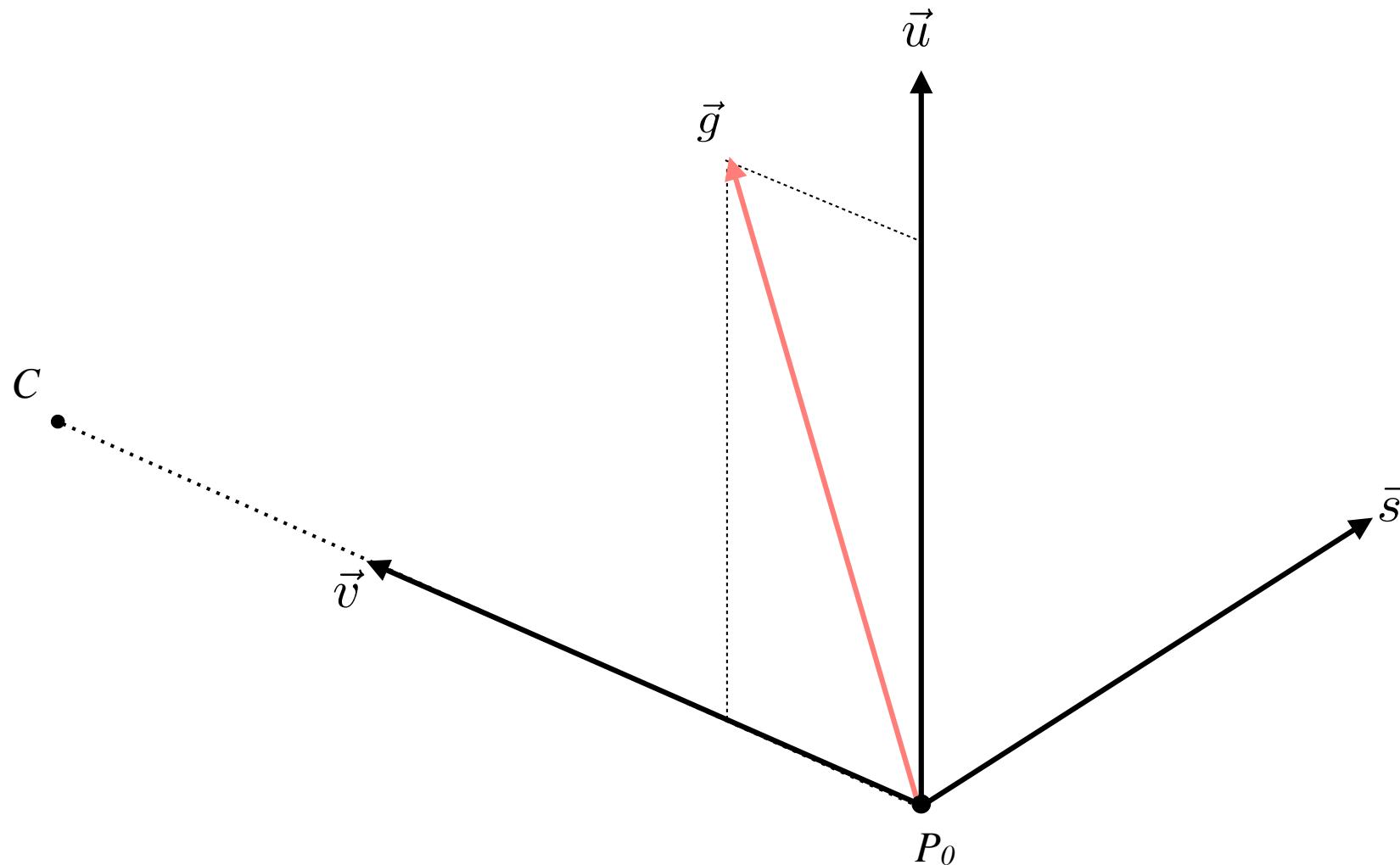
# Koordinatensysteme



# Kamera Koordinatensystem



# Kamera Koordinatensystem



# View Coordinates

- gegeben:

- Viewpoint:  $P_0$
- Center of Interest  $C$  (*Center*)
- User-Up Vector  $\vec{g}$

- gesucht:

- Camera Coord.sys.  $(\vec{v}, \vec{u}, \vec{s})$  (view,Camera-Up, side)



Wie erzeugen wir den View-Vektor?

$$\vec{v} = C - P_0$$



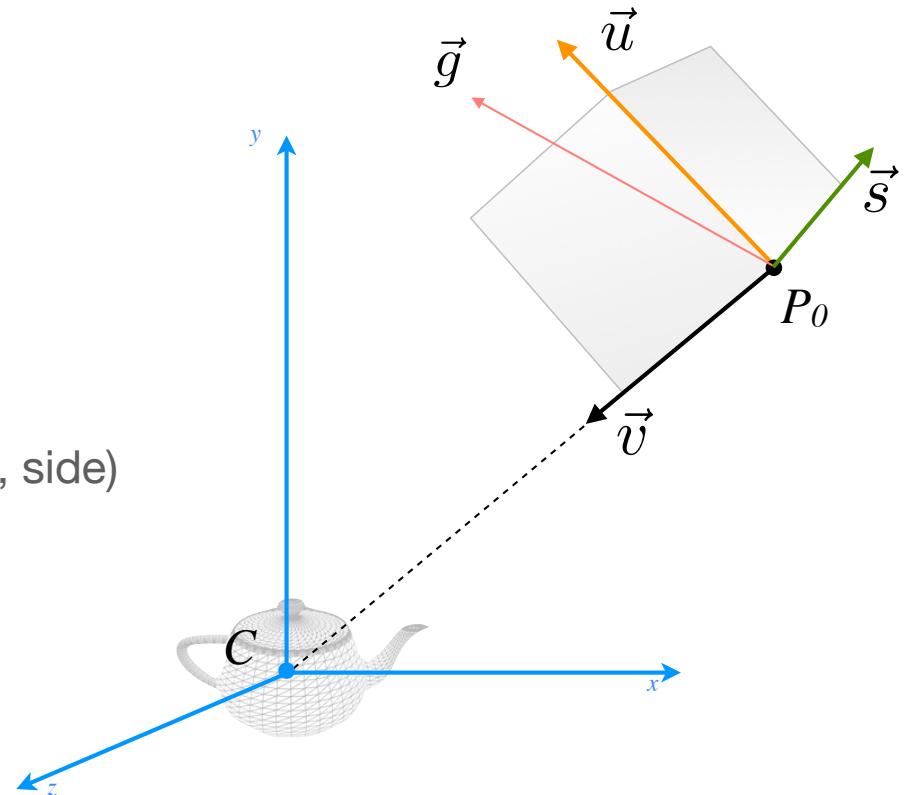
Wie erzeugen wir den Side-Vektor?

$$\vec{s} = \vec{v} \times \vec{g}$$

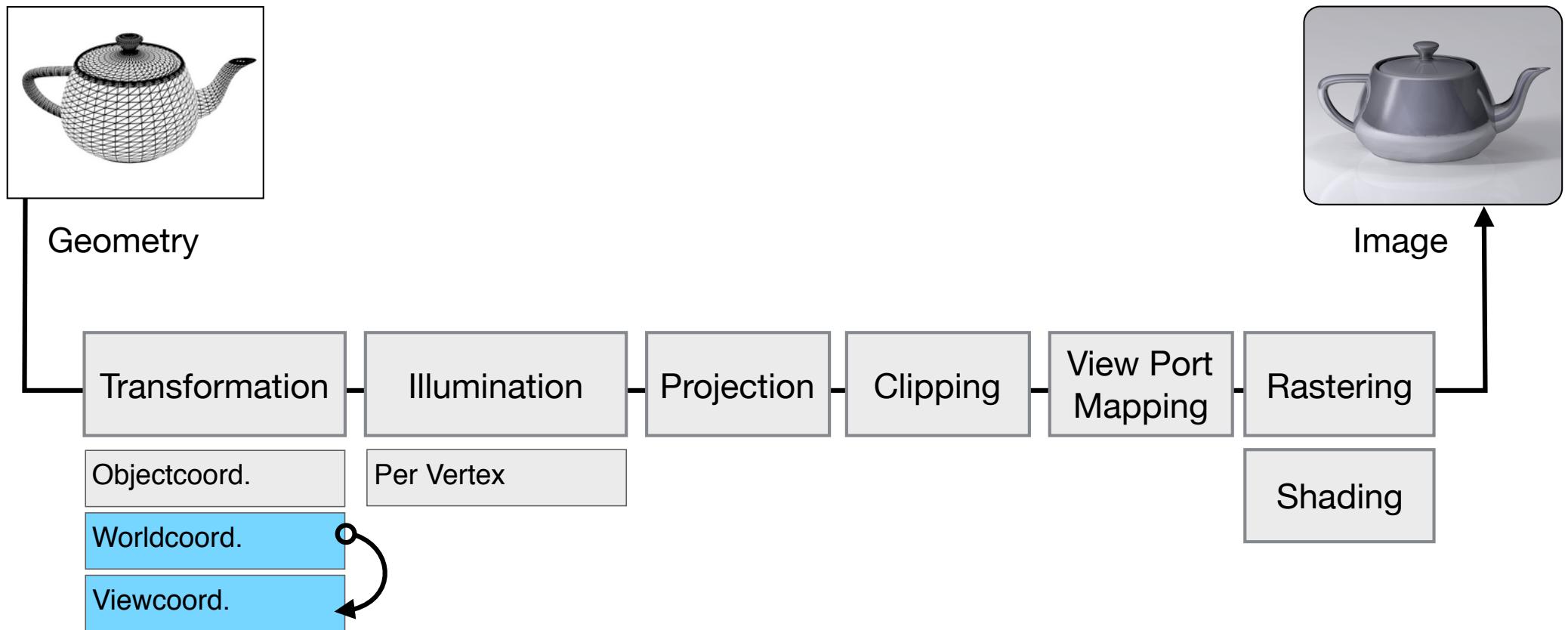


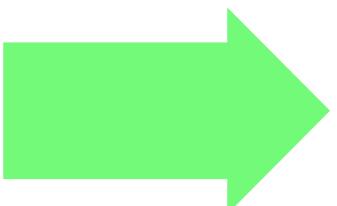
Wie erzeugen wir den Camera-Up Vektor

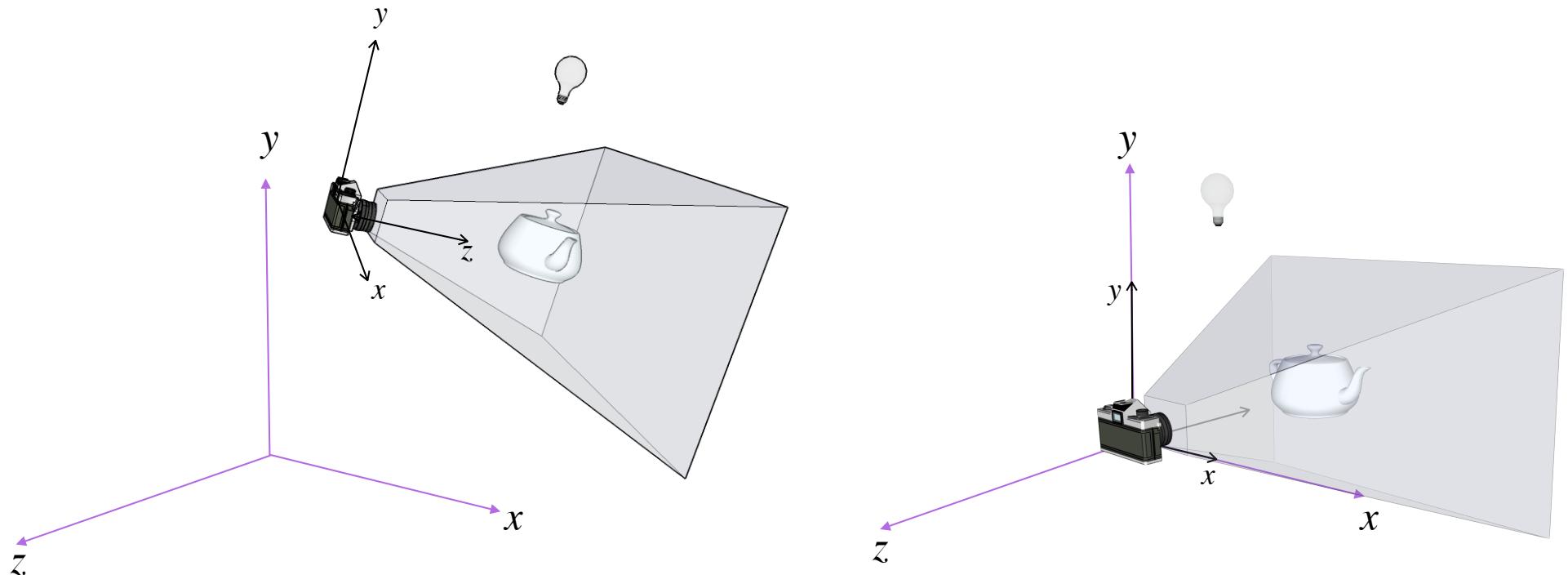
$$\vec{u} = \vec{s} \times \vec{v}$$



# *pipeline*



**WORLD**  **CAMERA**

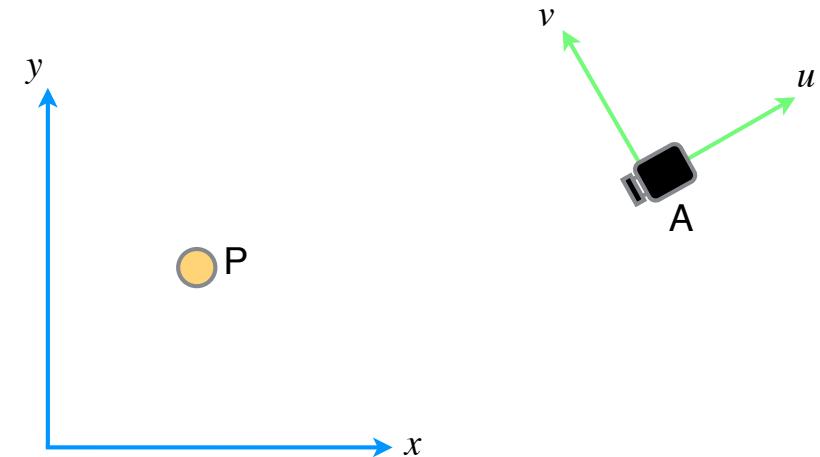


# Koordinaten Transformation

- Wie kommt man in das Kamerakoordinatensystem?

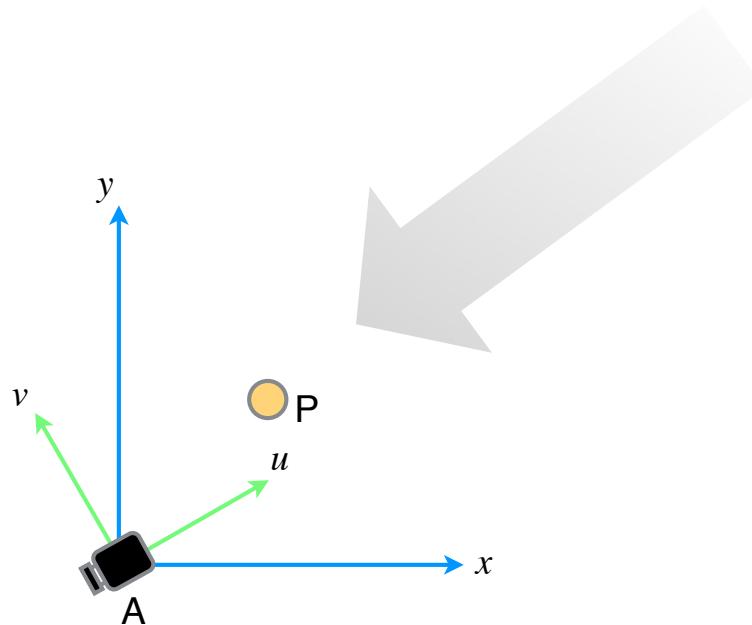
- Gegeben:  $P(x,y)$

- Gesucht:  $P(u,v)$



Translation in den Ursprung

$$T = \begin{pmatrix} 1 & 0 & -A_x \\ 0 & 1 & -A_y \\ 0 & 0 & 1 \end{pmatrix}$$



# Koordinaten Transformation

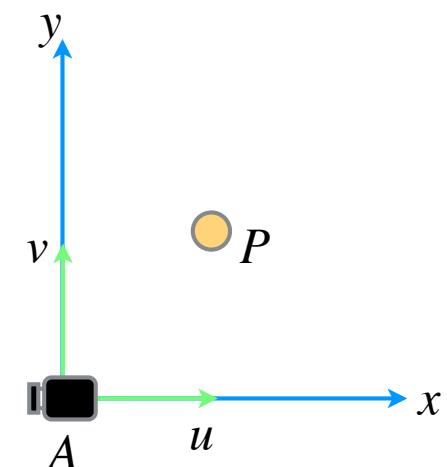
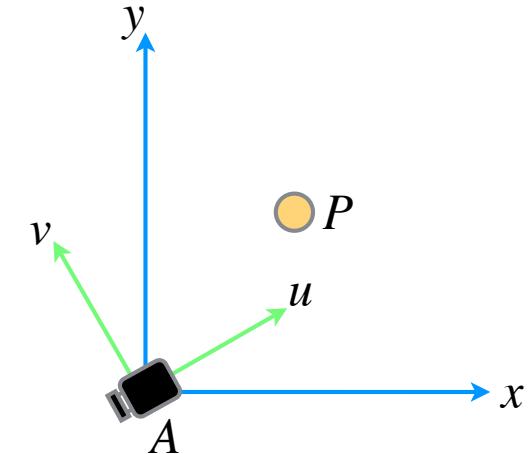
Translation in den Ursprung

$$T = \begin{pmatrix} 1 & 0 & -A_x \\ 0 & 1 & -A_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation, so dass die Achsen übereinstimmen

$$R = \begin{pmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (?) \text{ wie werden die Winkel berechnet?}$$

$$M = \begin{pmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -A_x \\ 0 & 1 & -A_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & -A_x u_x - A_y u_y \\ v_x & v_y & -A_x v_x - A_y v_y \\ 0 & 0 & 1 \end{pmatrix}$$



# Koordinaten Transformation

Praktisch werden aber die Objekte transformiert.

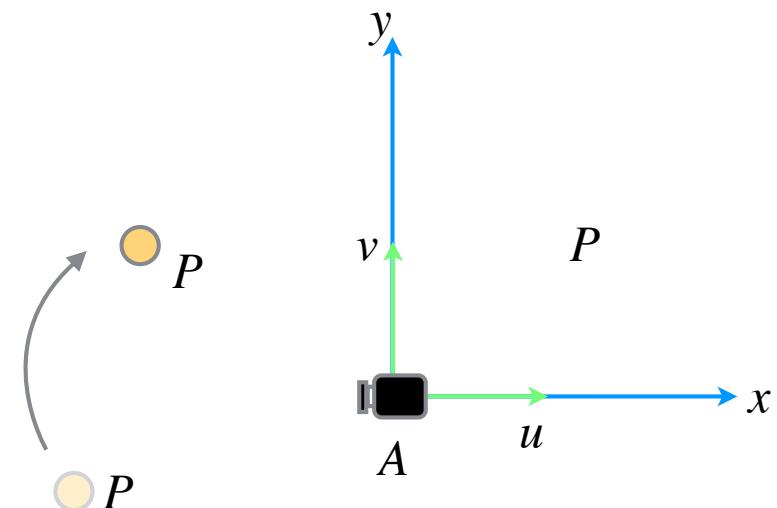
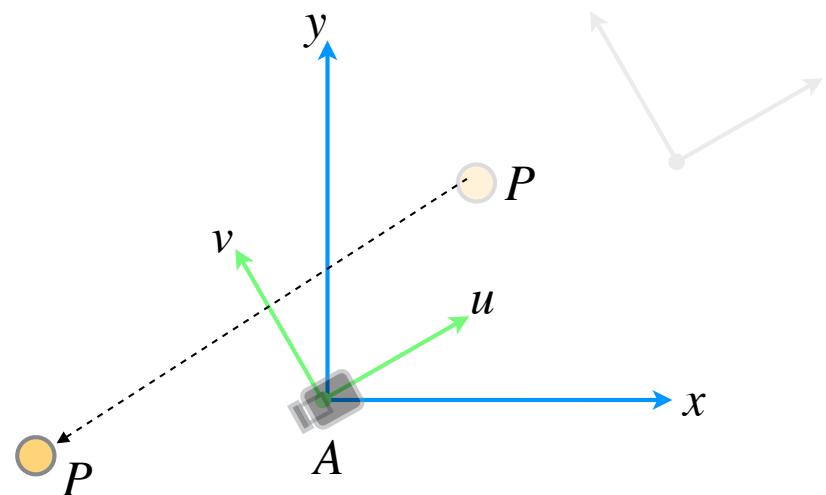
$$T = \begin{pmatrix} 1 & 0 & -A_x \\ 0 & 1 & -A_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation, so dass die Achsen übereinstimmen.

$$R = \begin{pmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

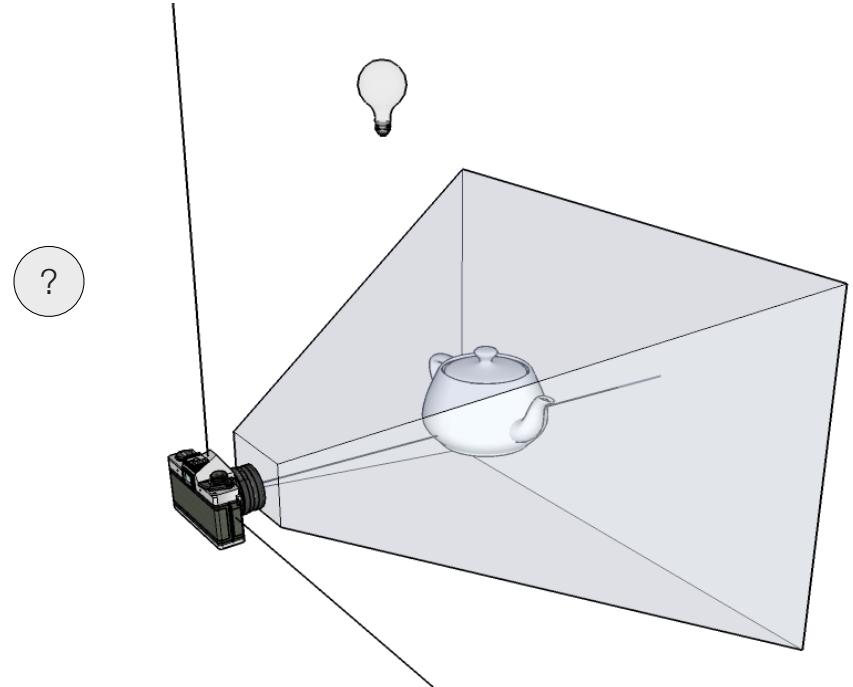
Wir bestimmen die spezielle Projektionsmatrix gültig für

1. Im Ursprung
2. Achsenparallel
3. in Richtung der negativen Z-Achse

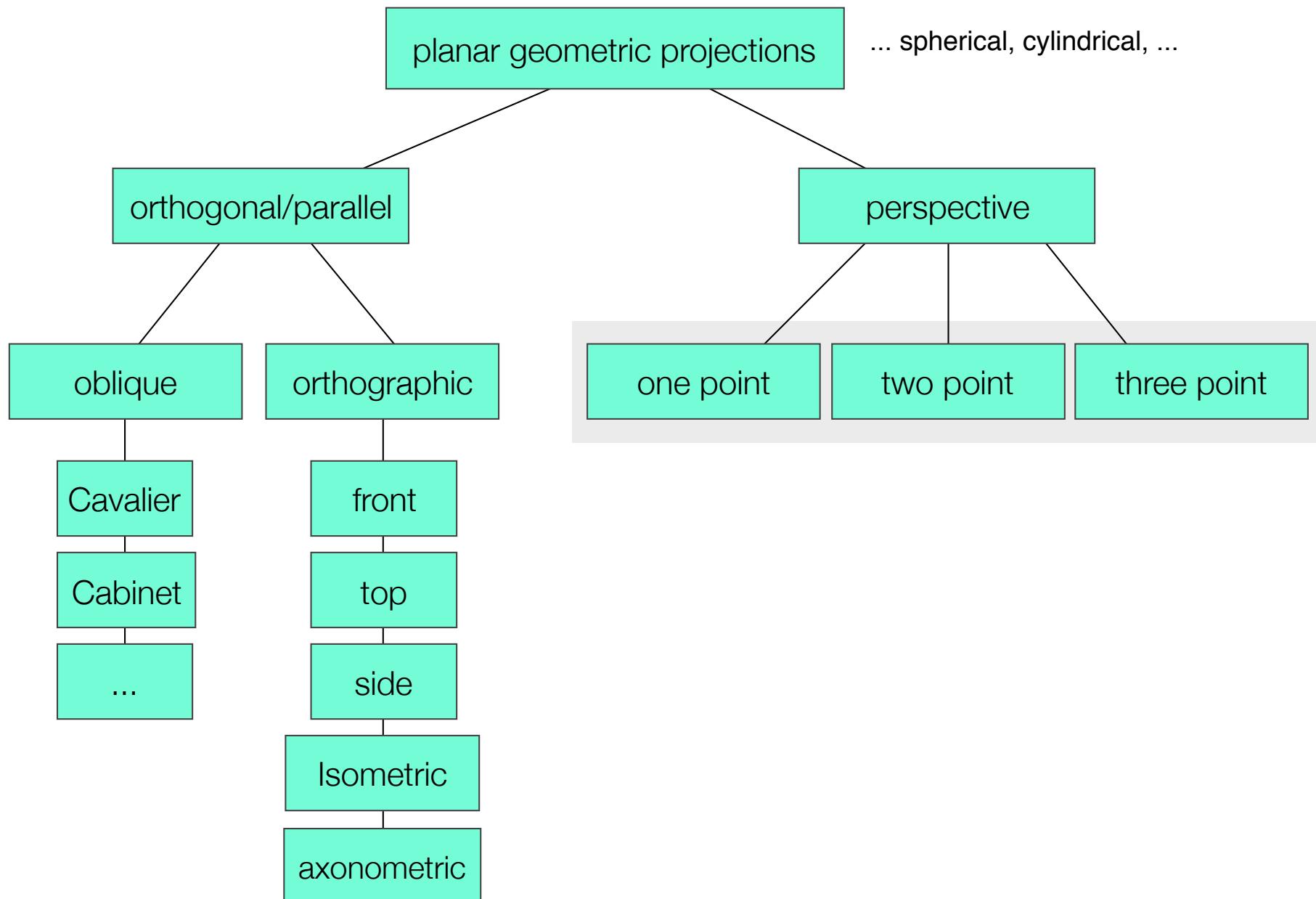


# Wo befinden wir uns ?

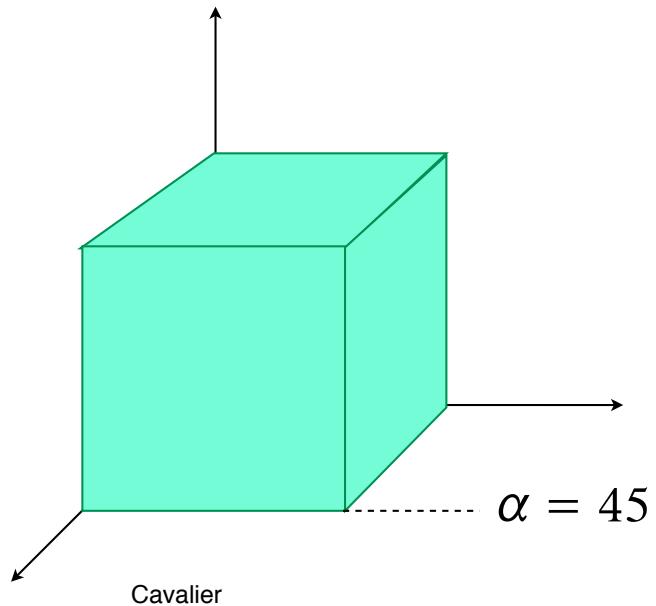
- Alle Objekt Transformationen sind durchgeführt
- Objekt ist im Worldmatrix transformiert
- View Koordinaten werden festgelegt mit
- ```
gluLookAt( 0.0, 0.0, 2.0,      // viewpoint
              0.0, 0.0, 0.0,      // center of interest
              0.0, 1.0, 0.0 )      // up - vector
```
- Transformation in Viewmatrix
- $p' = V \cdot (T \cdot S \cdot R \cdot T \cdot \dots \cdot R) \cdot p$  Viewmatrix
- An dieser Stelle wird die Beleuchtung berechnet.
- Danach erfolgt die Projektion
- und damit die Frage welche Projektionsart eingesetzt wird



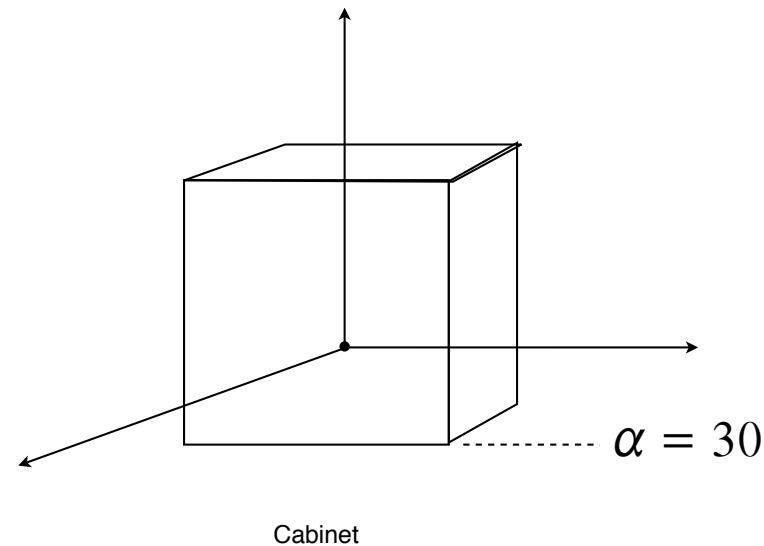
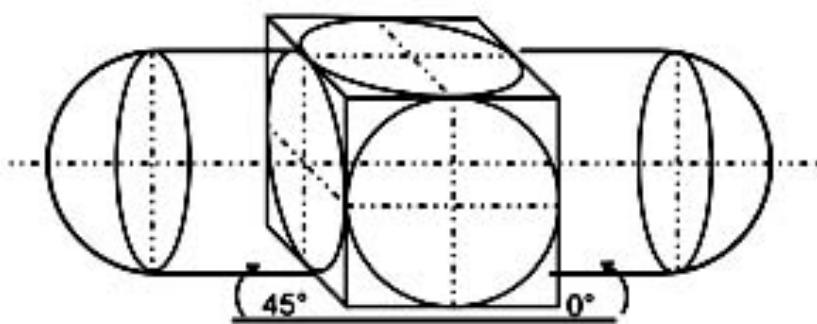
# Planar Geometric Projection



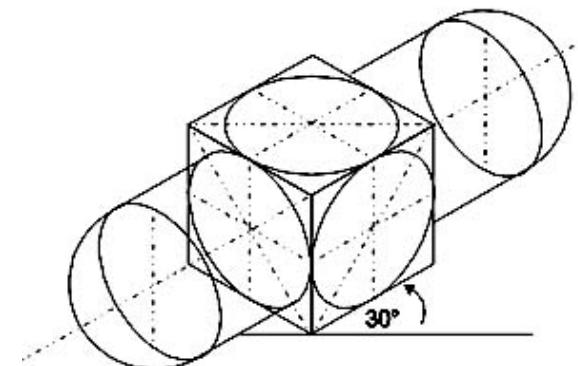
# Oblique parallel projection



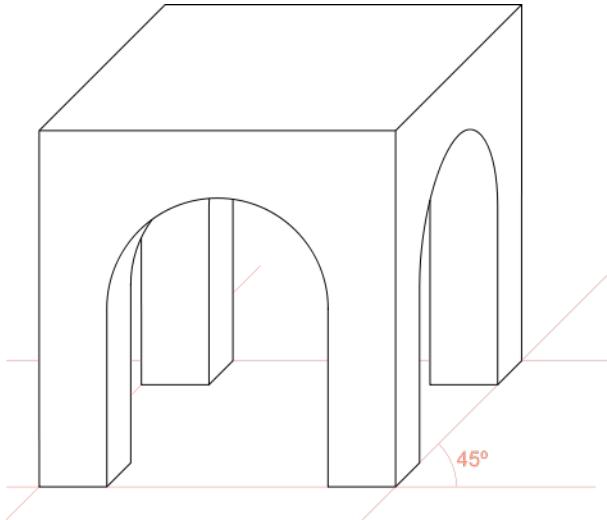
Längen bleiben erhalten



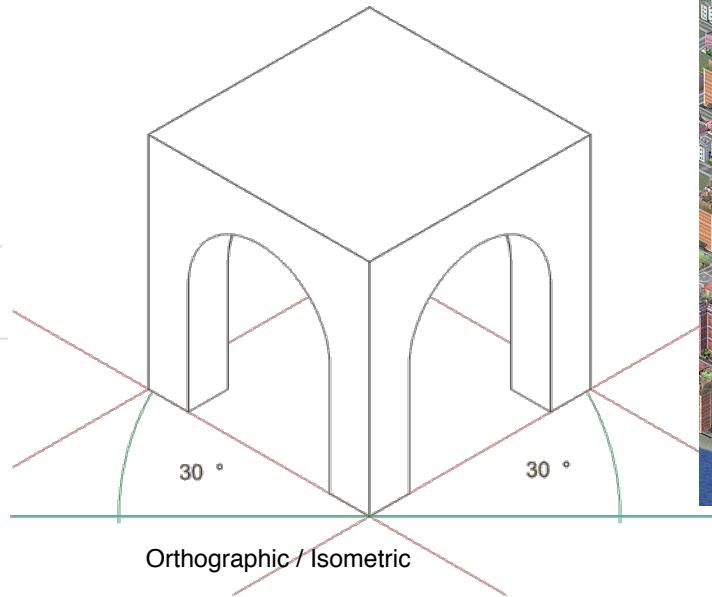
Längen sind um die Hälfte verkürzt



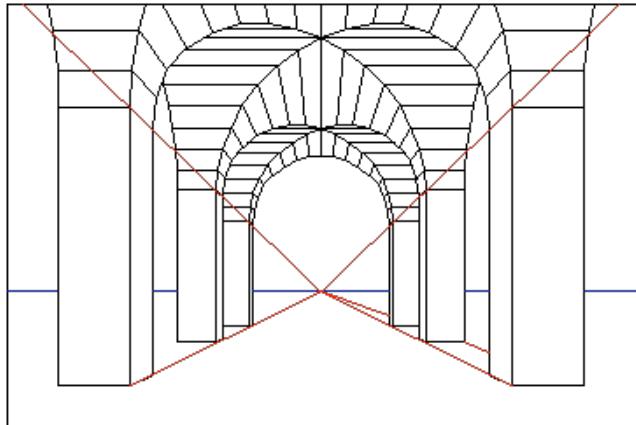
# Planar Geometric Projection



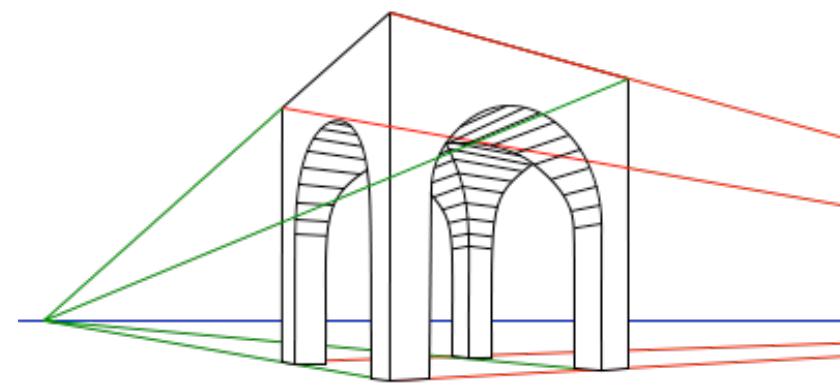
Oblique / Cavalier



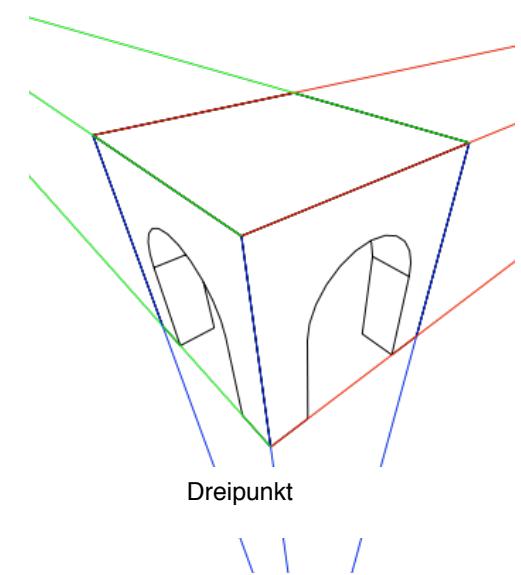
Orthographic / Isometric



Perspektive (Einpunkt / Zentral)



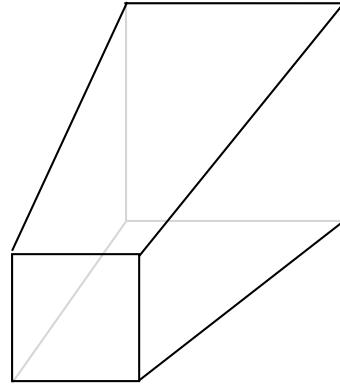
Zweipunkt



Dreipunkt

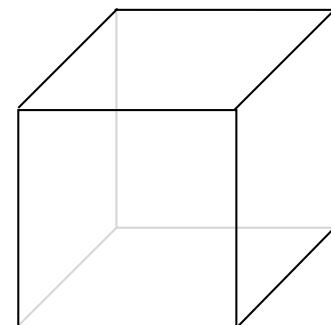
## ■ Perspective

- Vanishing points  
(Fluchtpunkte)
- wo ist das Auge?



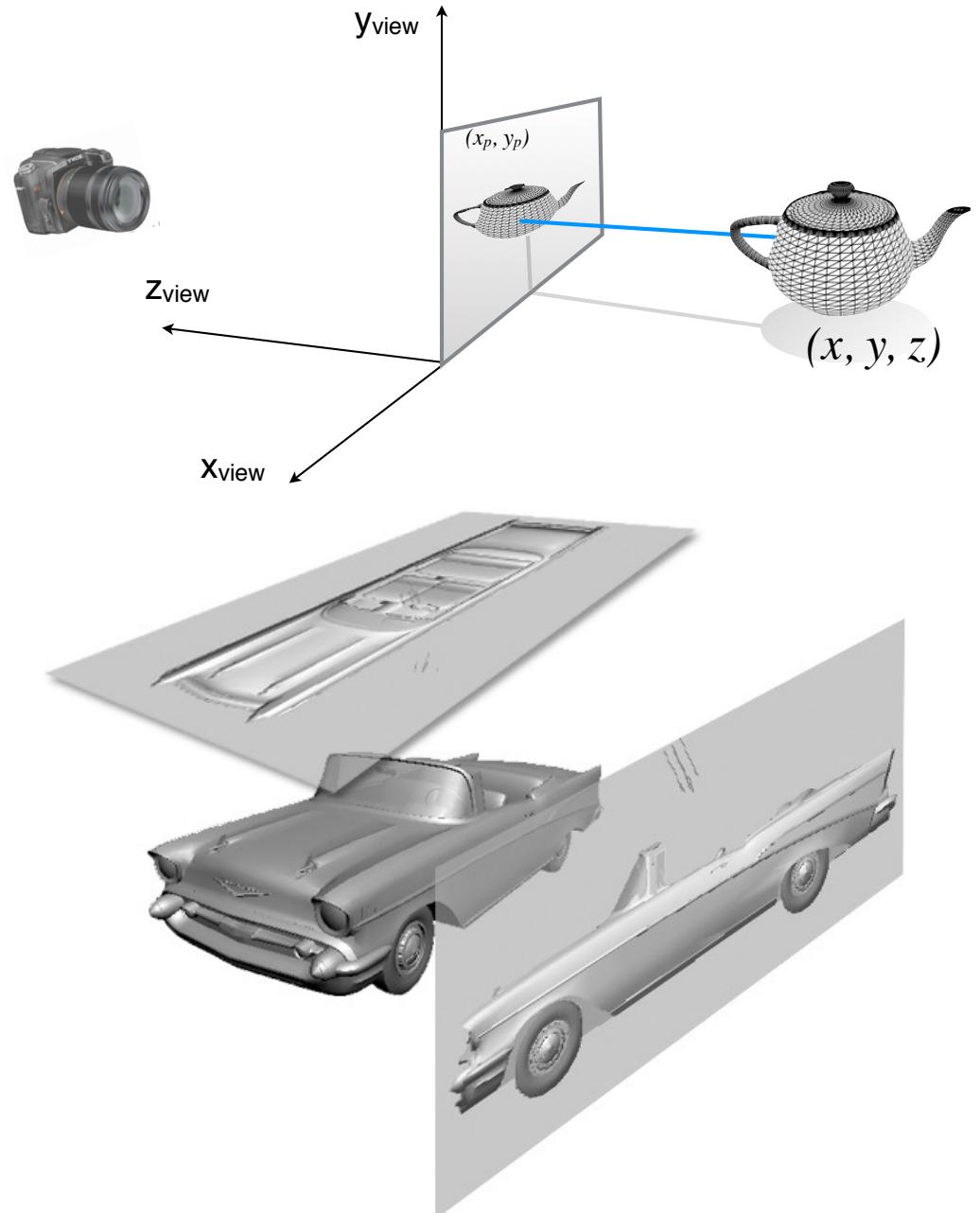
## ■ Orthogonal

- Parallelprojektion
- wo ist das Auge?



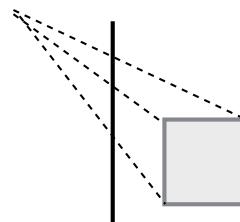
# Orthogonal Projections

- Orthographic projection
  - Top, Side, Front
- Bekannt aus
  - Architektur
  - CAD-Anwendungen
- Viewplane perpendicular (senkrecht) to one principal axis
- z.B.  $x_p=x$ ;  $y_p=y$ ;  $z=0$
- Z-Wert wird für die Verdeckung ausgewertet
- Längen bleiben gleich
- Wichtig für die Modellierung!!!



# oblique parallel projection

- Eine orthogonale Sicht reicht nicht, um ein Objekt zu beurteilen



- perspektivische Projektion:

- Vorteile:

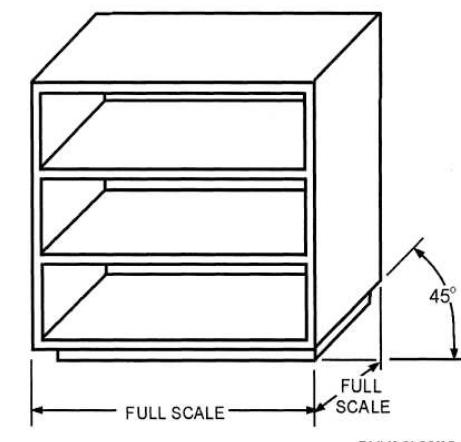
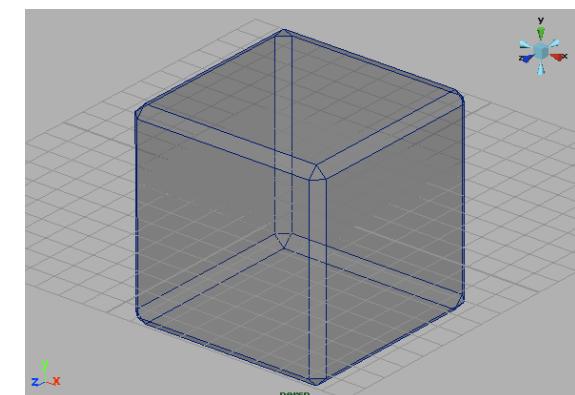
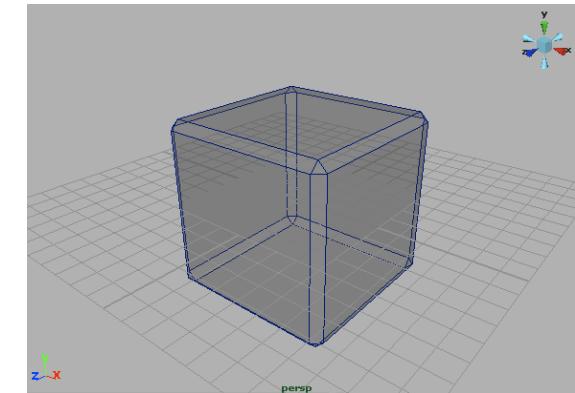
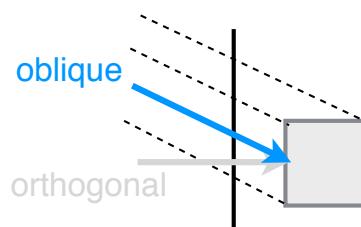
- realitätsnah
  - entspricht dem normalen Sehvorgang

- Nachteile

- Längen schwer abzuschätzen
  - Ergebnis abhängig von Kameraparameter

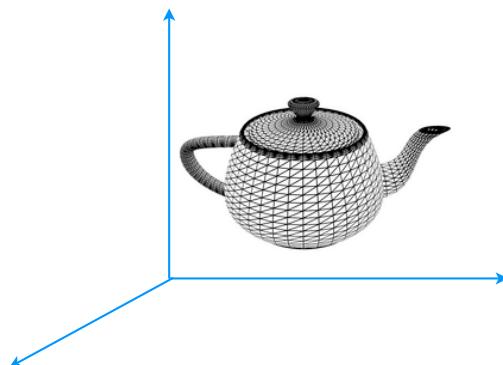
- Darstellungsmethode für Designer, Architekten und Ingenieuren mit Maßeinheiten

- Längen bleiben gleich
  - Wichtig für Bemaßungen und Verhältnisse

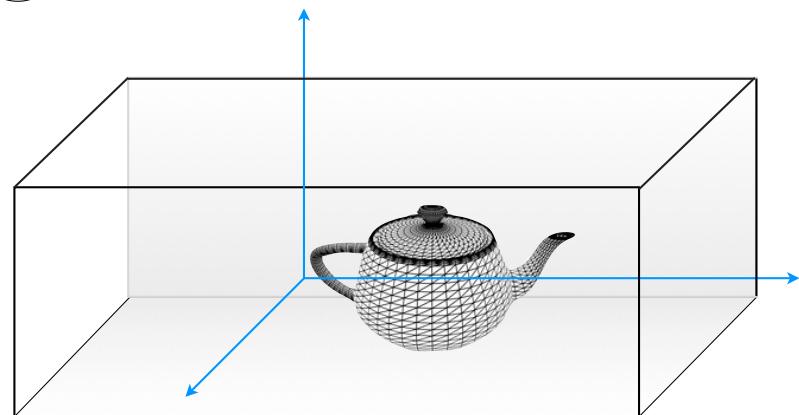


# Orthogonal Projection - steps

1

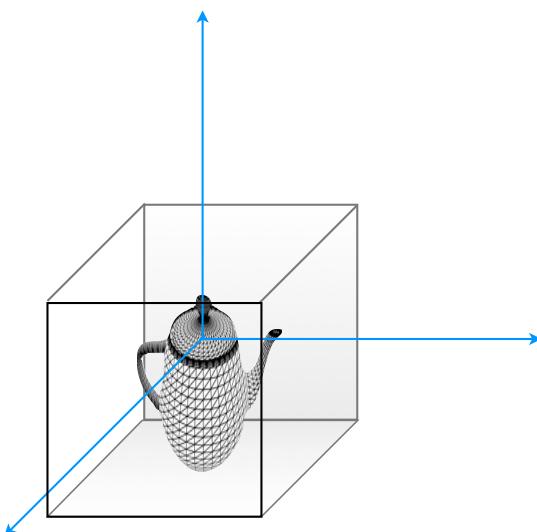


2



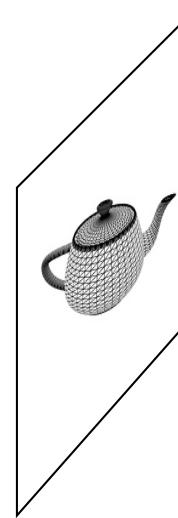
orthogonal projection view volume  
Kanonisches Volumen

3



normalized view volume

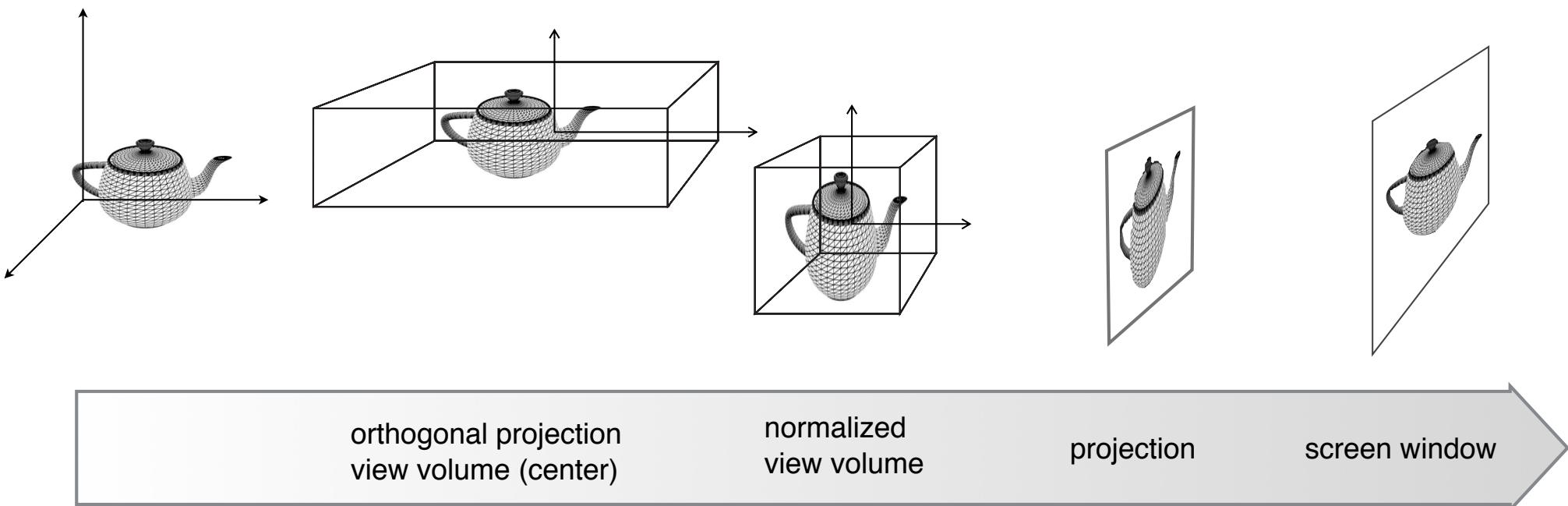
4



projection



- Position im Weltkoordinatensystem
- Position im Kamerasystem
- Forderung: Orthogonale Projektion
- Viewpoint & -Volume festlegen
- View Volume zentrieren
- View Volume normalisieren
- Back Face Culling
- Clipping - mit achsenparallelen Ebenen (+1,-1)
- Eine Koordinate eliminieren (Projektion)
  - Tiefenwert für Verdeckung nutzen
- View Port Transformation
  - Position in Screen Koordinatensystem



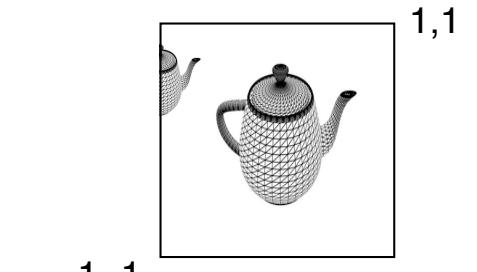
# Warum Normierung ?

- Orthogonal Projection View Volume
- Normierung auf Einheitsvolumen (-1,1)
- „Entzerren“ des „Sichtquadrates“ auf die endgültige Fenstergröße des Sichtfensters
  - Mit den normierten Werten können die Werte einfach zum „Pixel-Wert“ umgerechnet werden.  
(Viewporttransformation)
  - Culling (Back Face Removal)
    - Testen ob der Z Wert einen negativen Wert hat
  - Clipping der Polygone
    - Schnitttests und Clipping-Verfahren gegen Achsenparallele Ebenen
    - Alle Gleichungen und Berechnungen einfacher, wenn die Ebenen mit -1,+1 zu definieren sind.

1



2



-1,-1

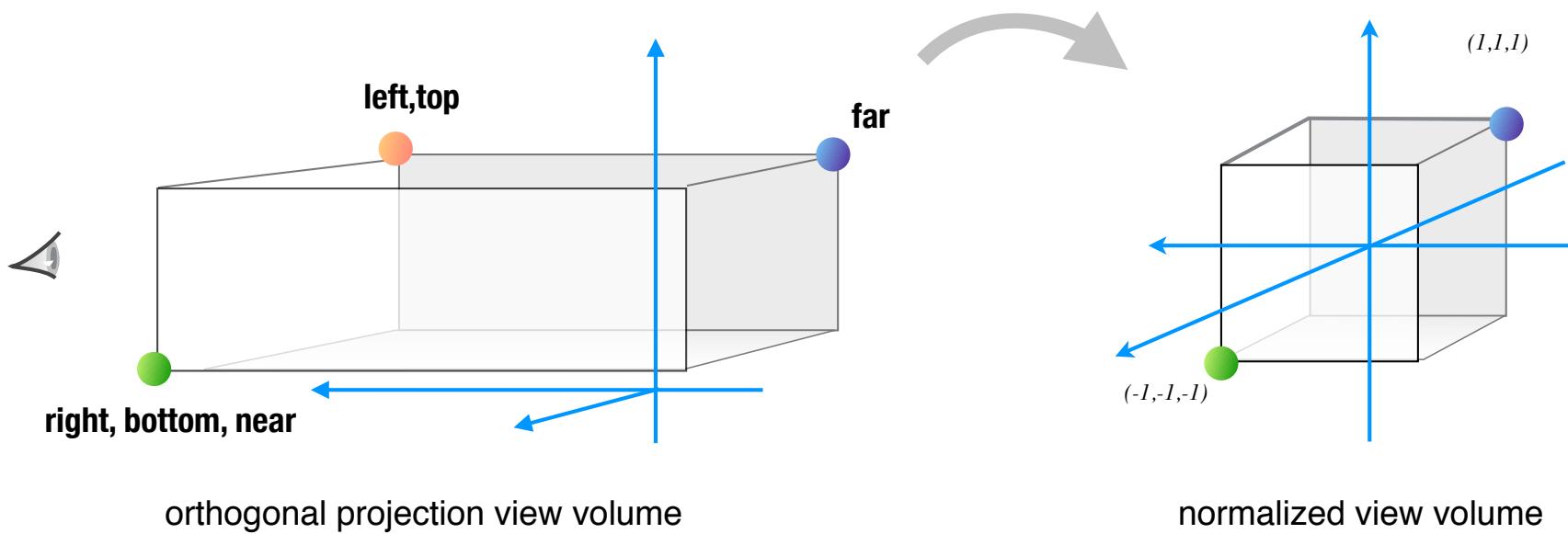
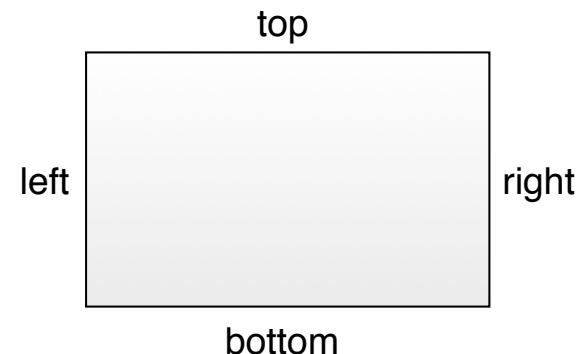
3



768X576  
1024X768  
...

# Normalization Transform (Orthogonal)

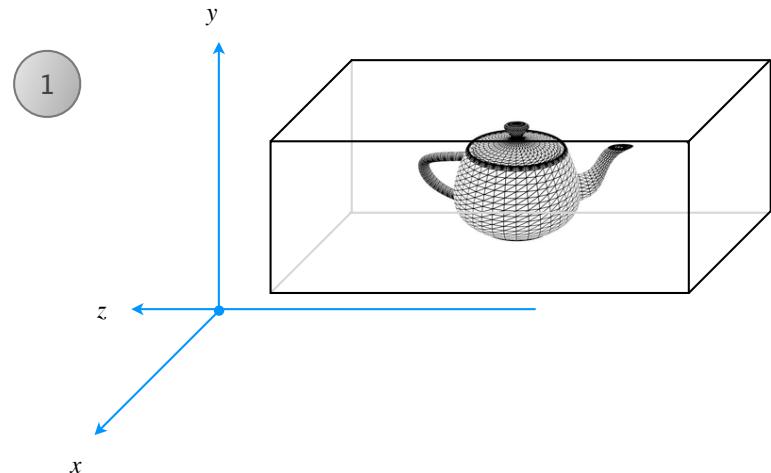
- Bisher ist die Viewplane zu bestimmen mit
  - (top,bottom, left, right)
- Einführung des achsenparallelen Sichtvolumens  
AABB (Axisaligned Bounding Box) definiert durch 6-Tupel
  - ( $n, f, l, r, b, t$ ) ( $near, far, left, right, bottom, top$ )
- Einführung von “Clipping Plane”
  - near, far
- Normalization, Clipping, Screencoordinates



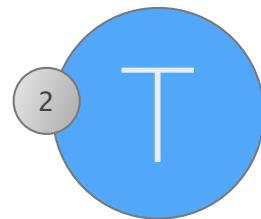
`glOrtho(left,right,bottom,top,near,far)`

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

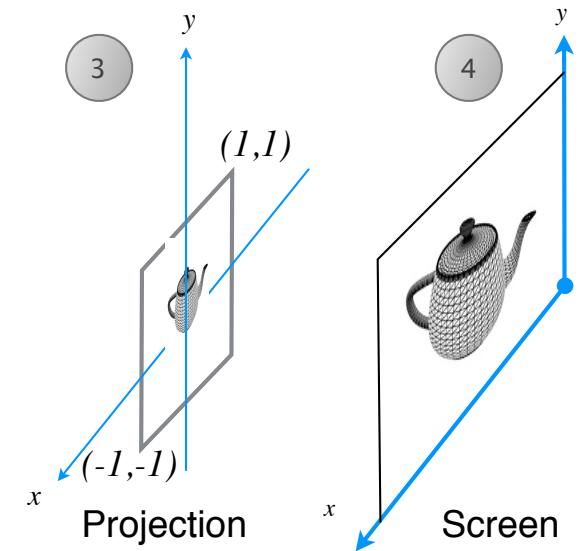
# Was macht die Matrix ?



arbitrary orthogonal projection view volume



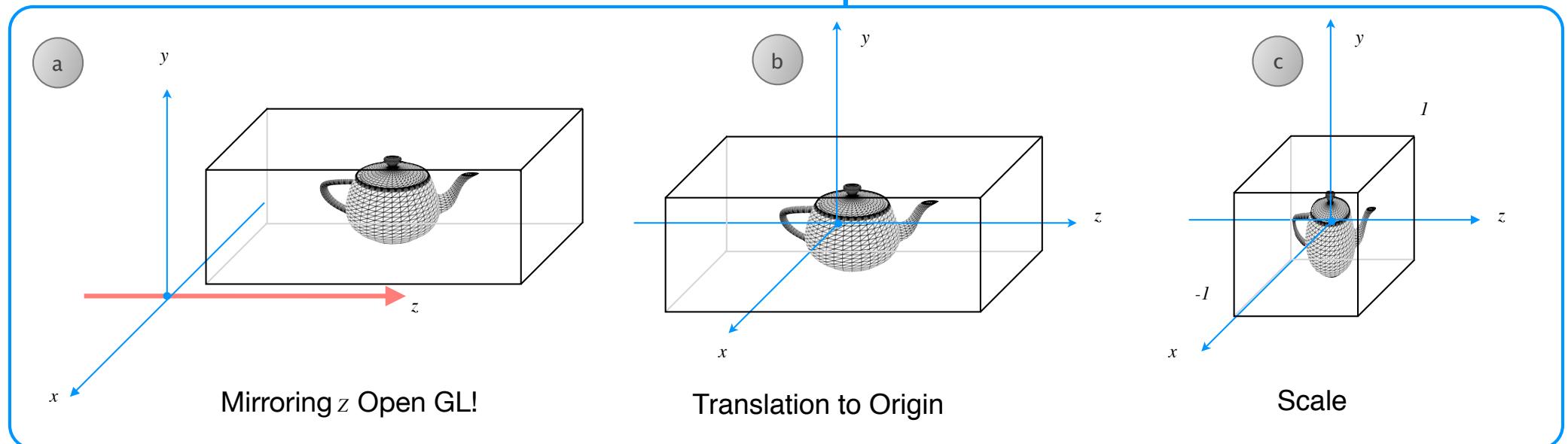
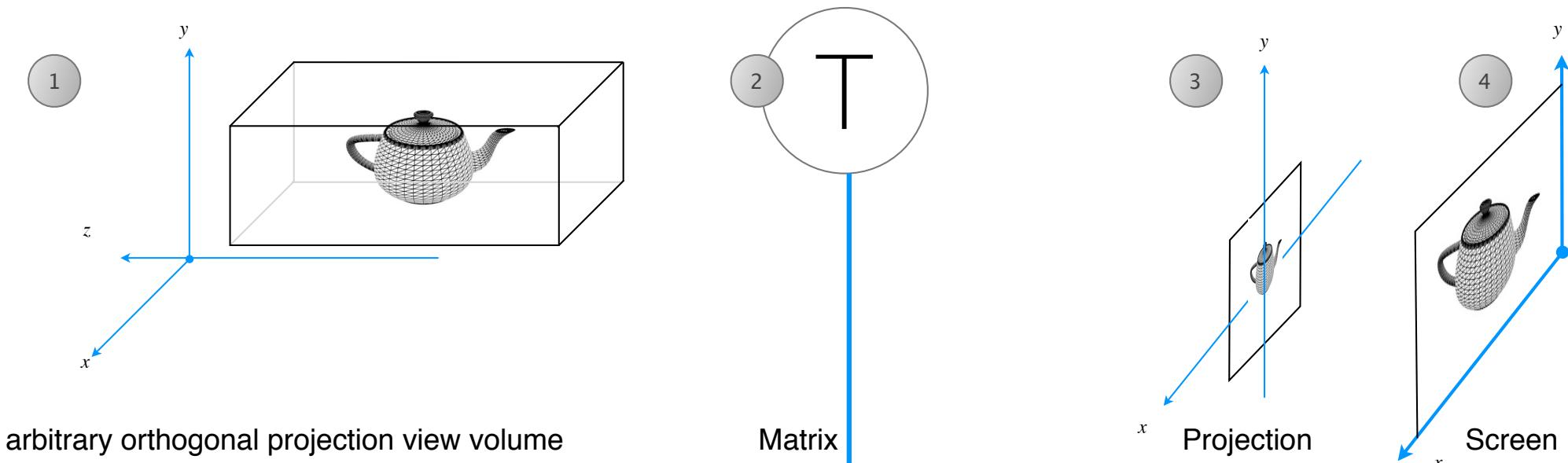
Matrix



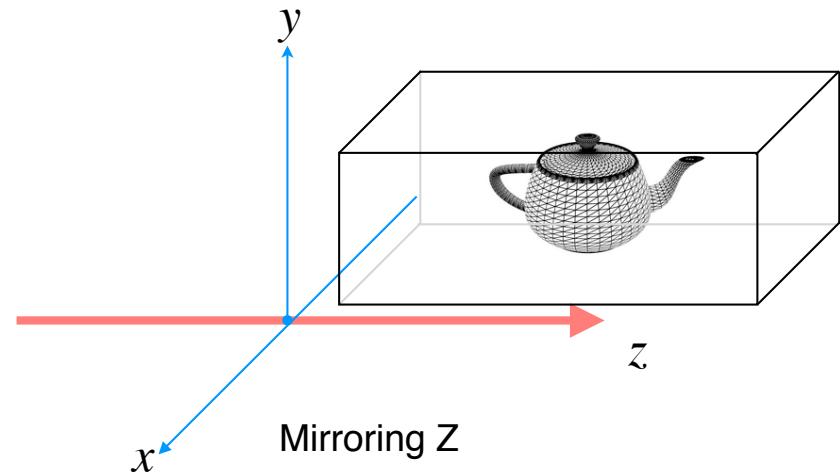
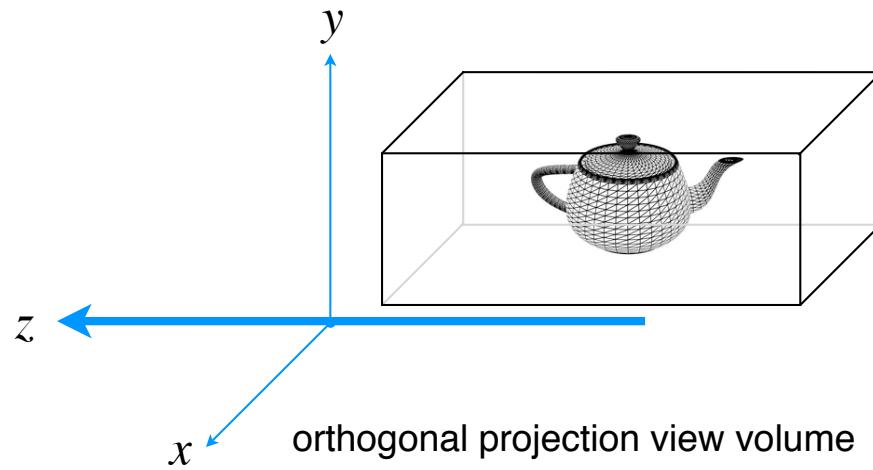
Projection

Screen

# Was macht die Matrix ?

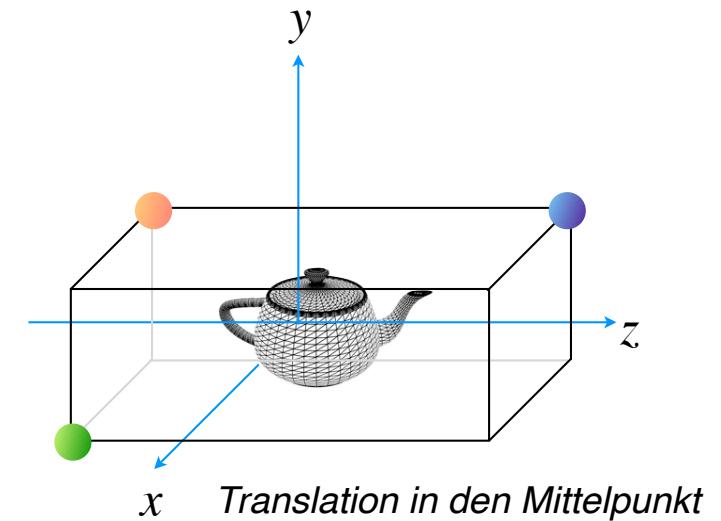
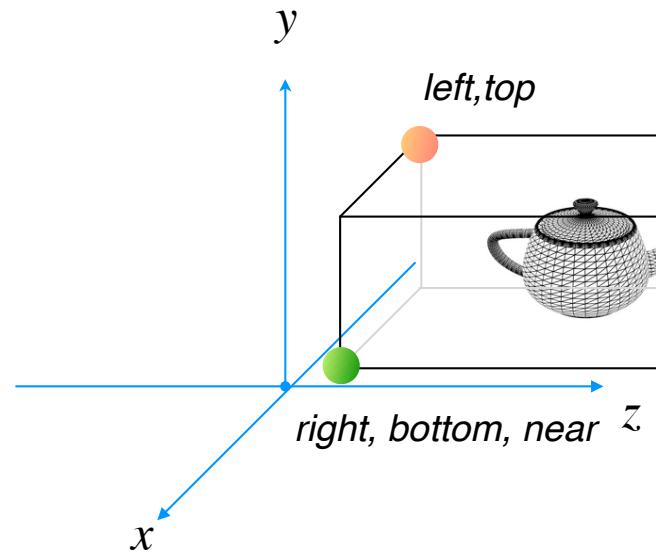


# Mirroring



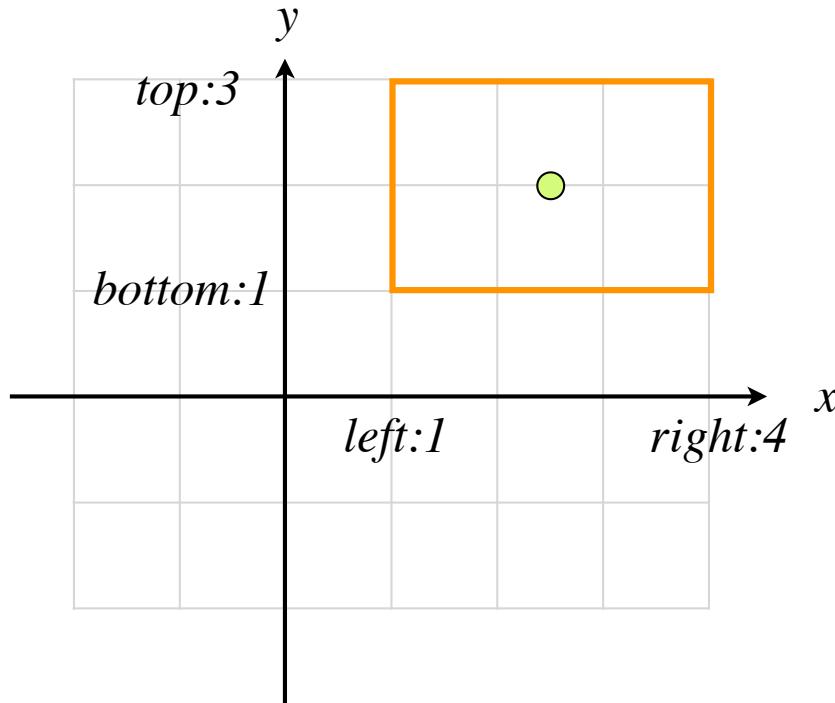
$$M_{Sp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Translate to Origin



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

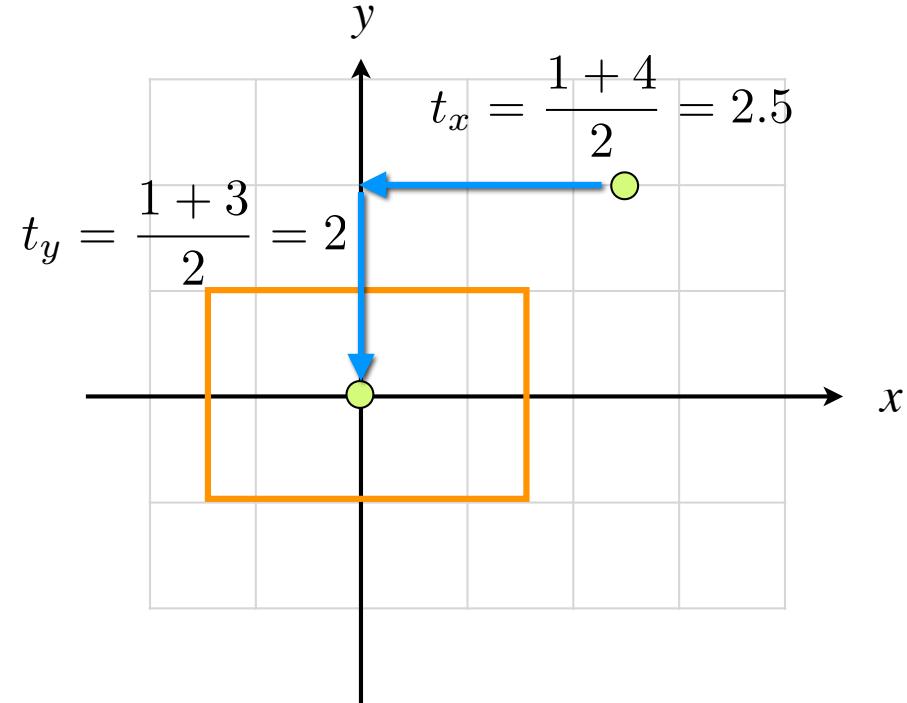
# Translate to Origin



Allgemeine Position des Mittelpunktes  
in X-Richtung:

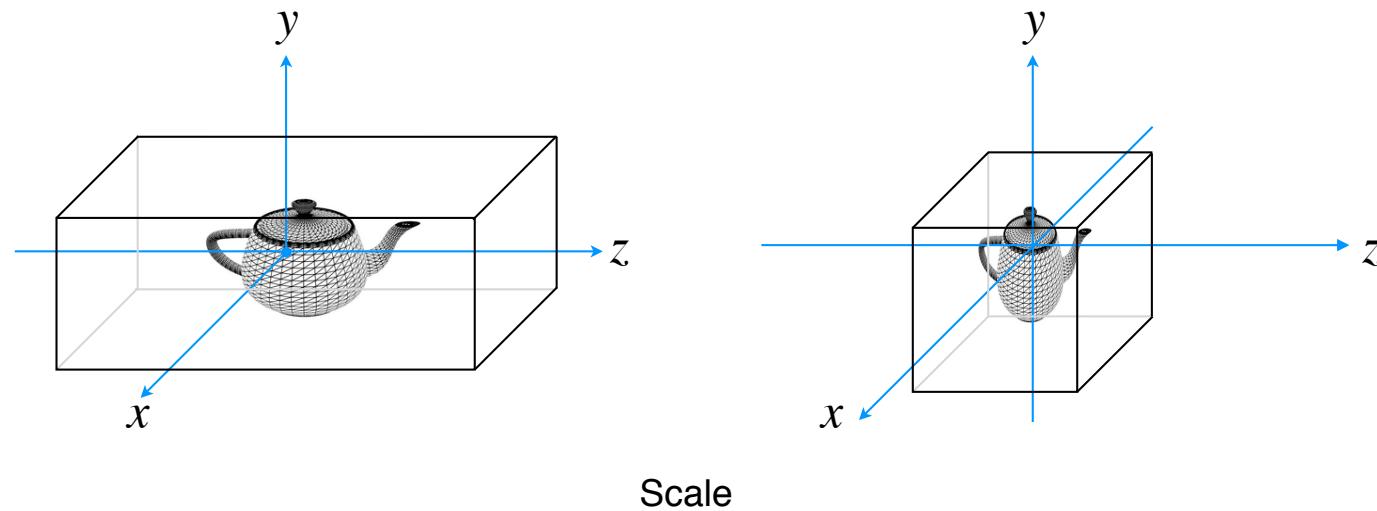
$$l + \frac{1}{2}(r - l) = \frac{l + r}{2}$$

Breite des Rechtecks



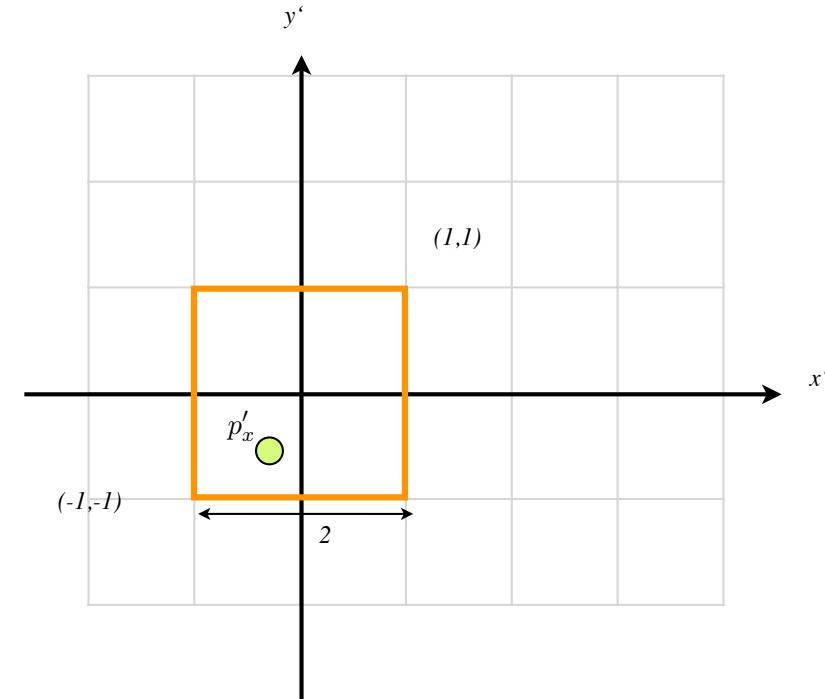
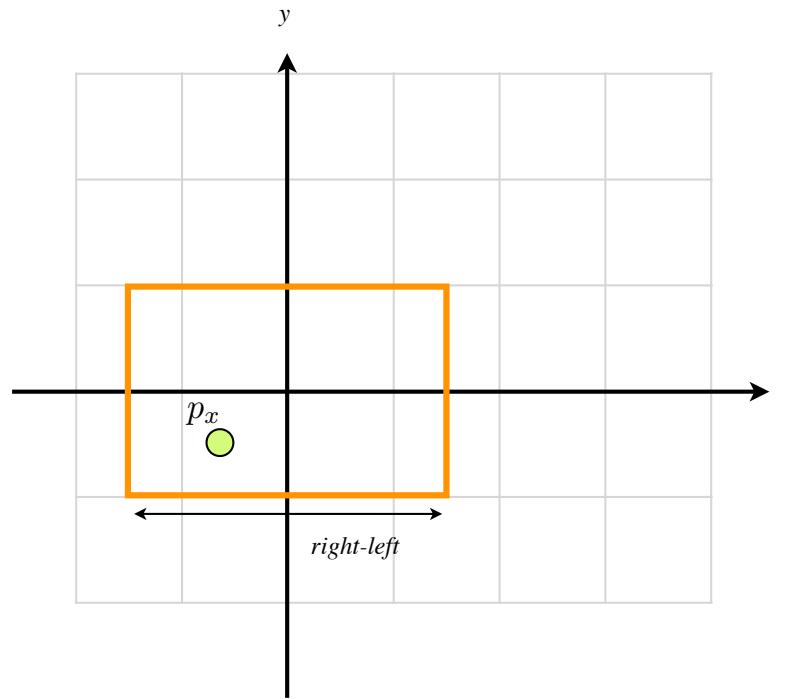
Für das Volumen:  $M_T = \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$

# Scale to uniform



$$M_T = \begin{pmatrix} \frac{2}{r-t} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Scale to uniform



$$\text{Allgemein: } \frac{p'_x}{p_x} = \frac{2}{r-l}$$

$$\text{damit: } p'_x = \frac{2}{r-l} \cdot p_x$$

$$\text{Für das Volumen: } M_T = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Normalization Transform

Die Konkatenation der Transformationen:

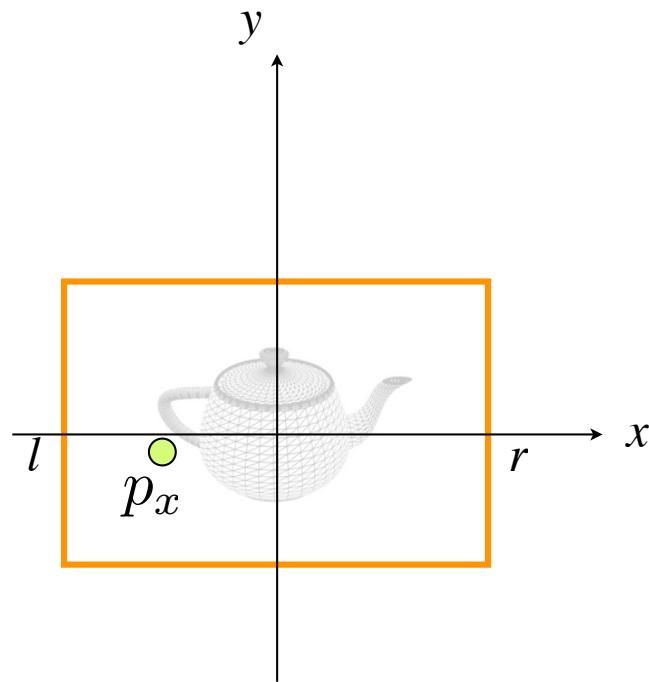
$$M = \begin{pmatrix} \frac{2}{r-t} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1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

$M_{\text{Ortho}}$        $M_{R \sim L}$

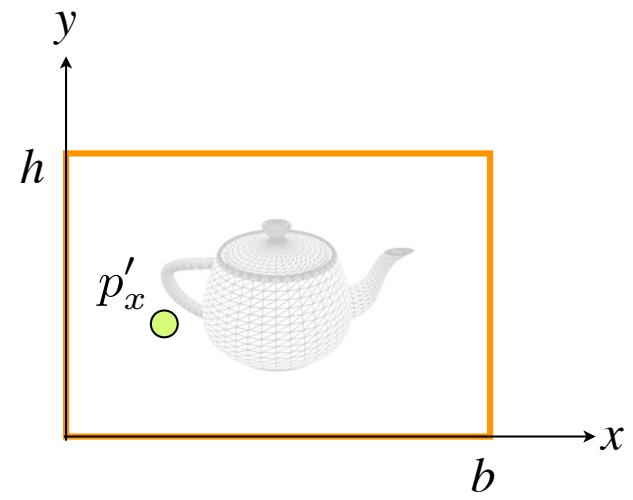
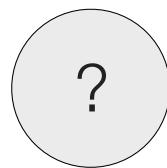
$$M = \begin{pmatrix} \frac{2}{r-t} & 0 & 0 & -\frac{l+r}{r-t} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

glOrtho(left,right,bottom,top,near,far)

# Viewport Transformation

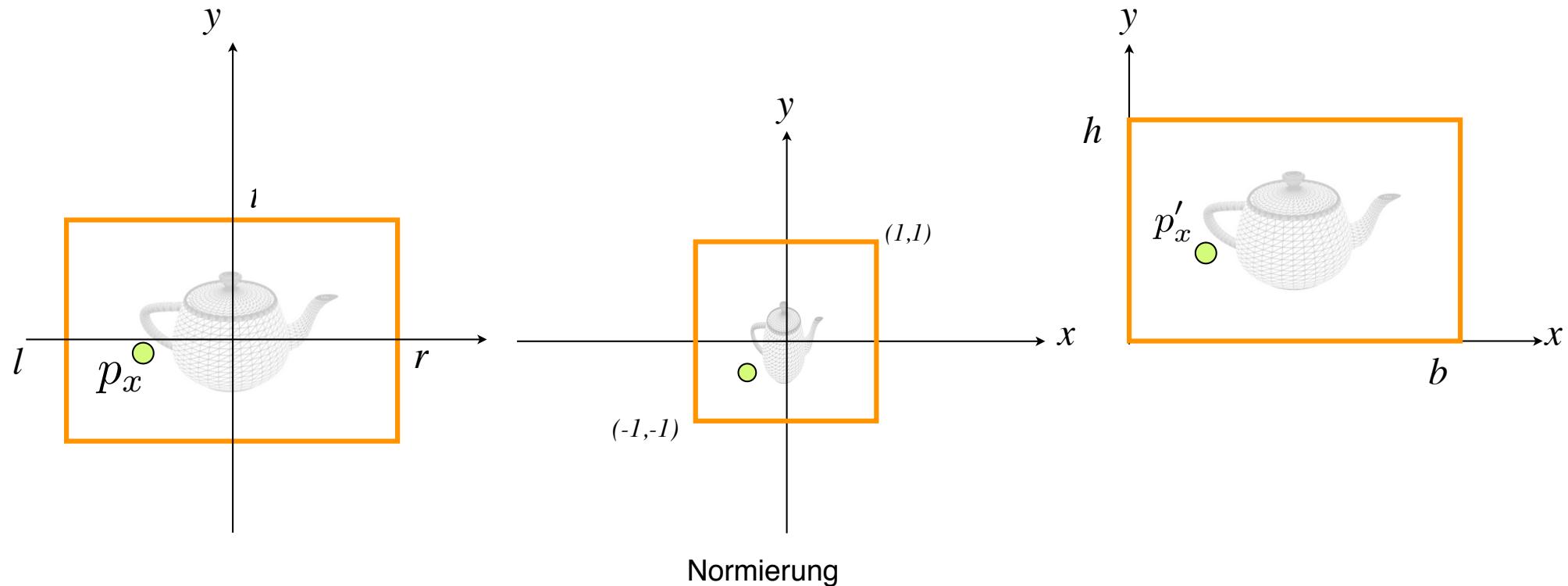


right / left / top in Weltkoordinaten

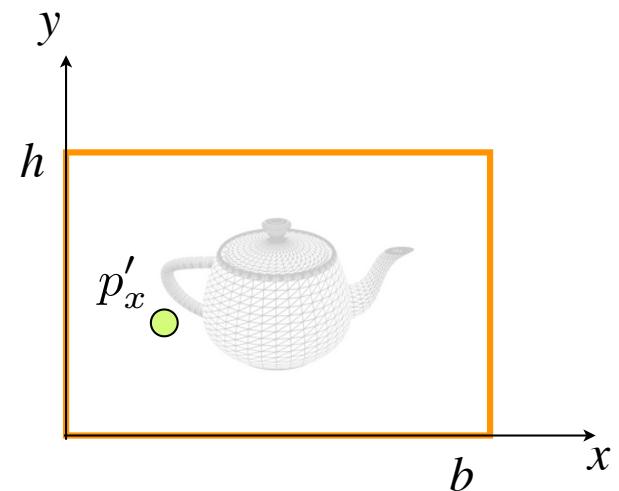
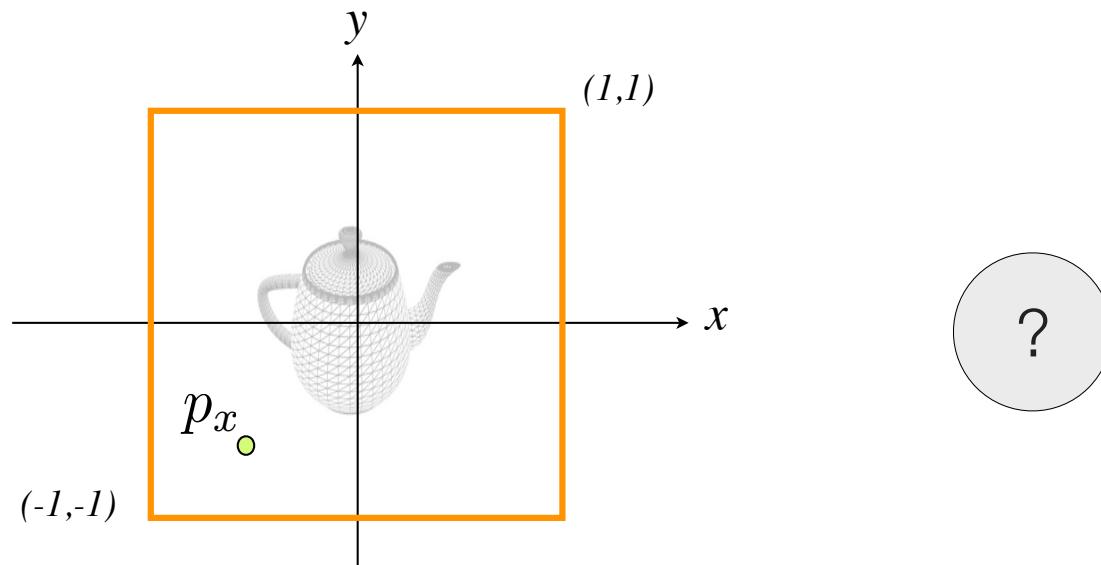


Höhe / Breite in Pixel

# Viewport Transformation

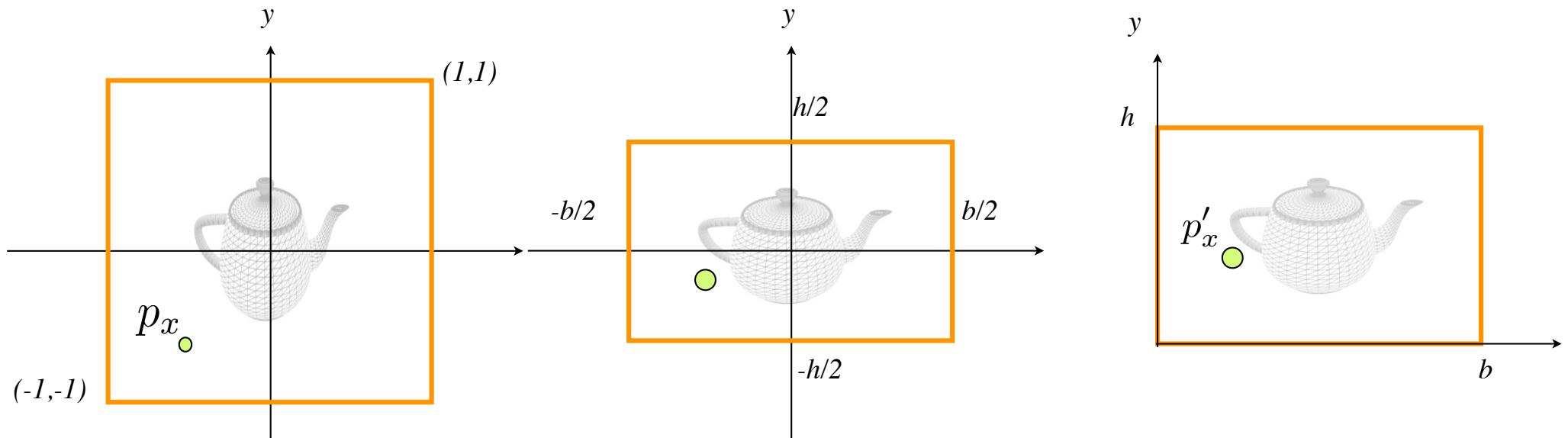


# Viewport Transformation



Entzerrnen des Einheits-Quadrates auf das Sichtfenster

# Viewport Transformation



$$\begin{aligned} p'_x &= \frac{b}{2} \cdot p_x + \frac{b}{2} \\ p'_y &= \frac{h}{2} \cdot p_y + \frac{h}{2} \\ p'_z &= \frac{1}{2} \cdot (p_z + 1) \quad \text{Wertebereich des } Z \text{ Buffers?} \end{aligned}$$

p e r s p e k t i v e n



# Perspective Projection in OpenGL

```
void gluPerspective(GLdouble fov, GLdouble aspect, GLdouble near, GLdouble far);
```

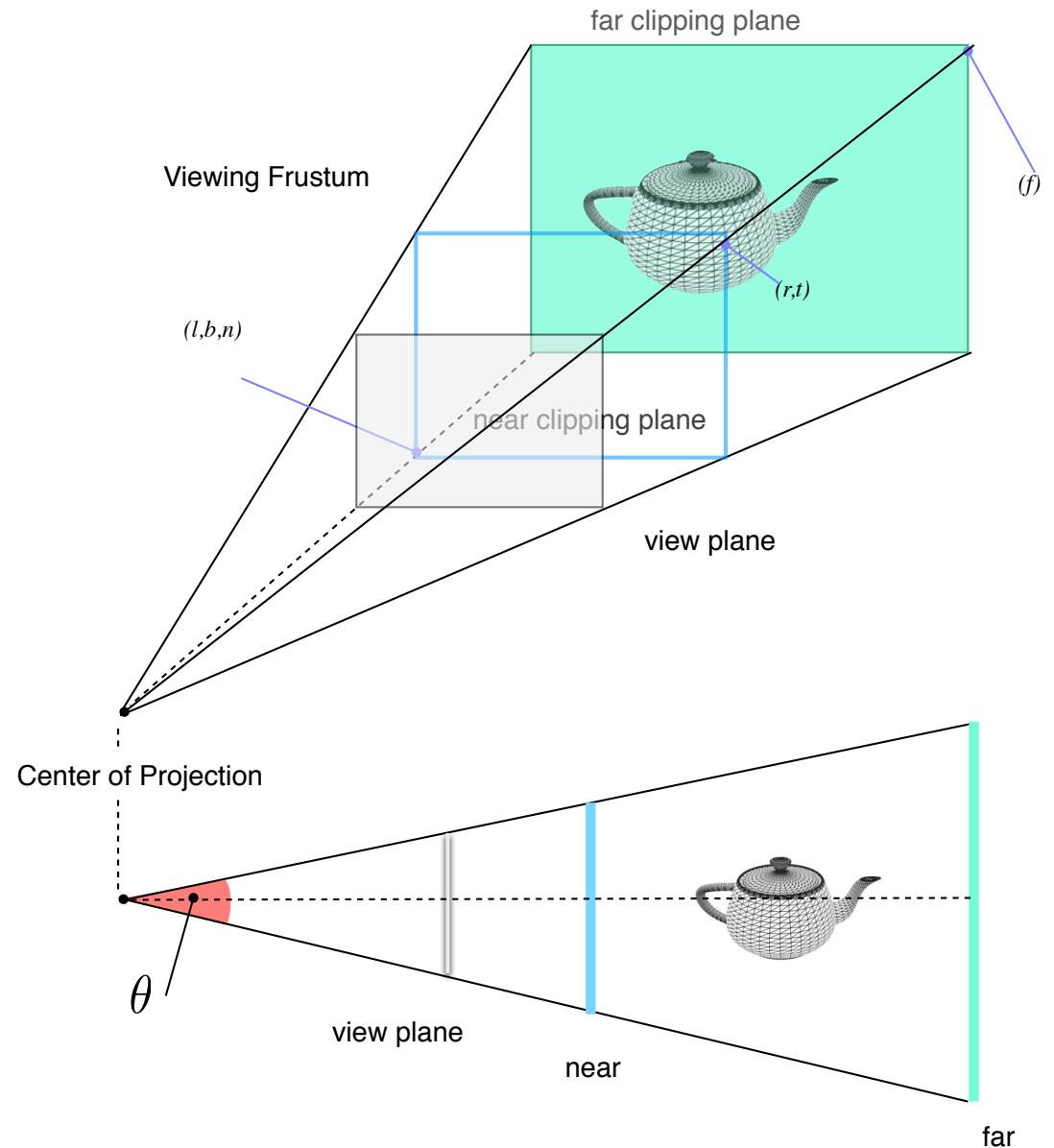
$$M = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} - \frac{2fn}{f-n} & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$



# Projection View Frustum

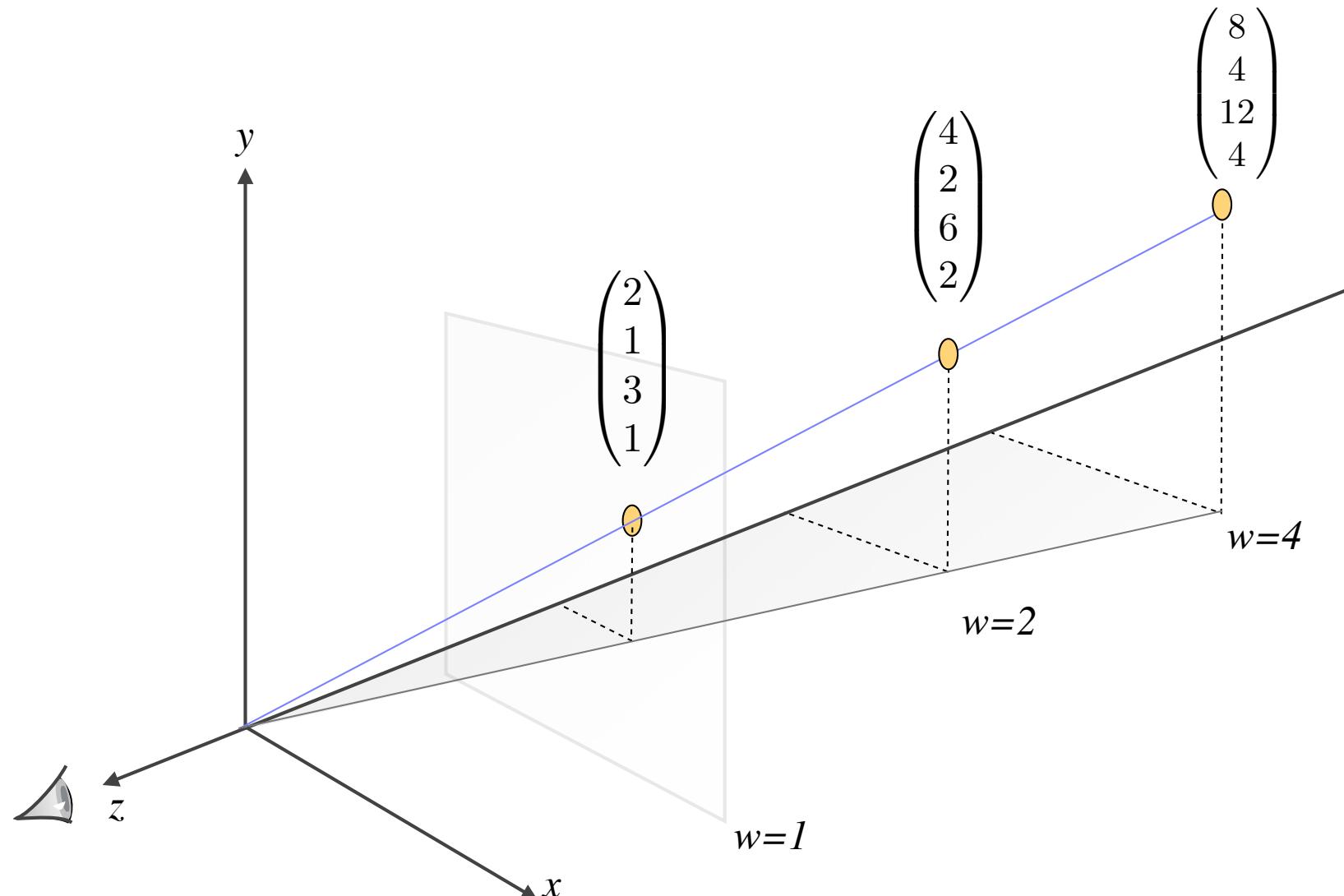
## Sichtvolumen

- Center of Projection (*Augpunkt*)
- Field of View  $\Theta$  (*fov*)
- Aspect ratio (*Seitenverhältnis*)
- Near Clipping Plane (*near*)
- Far Clipping Plane (*far*)
- üblich viewplane = nearplane
- Bei symmetrischen Sichtvolumen
  - $r=-l, t=-b$



```
void gluPerspective(GLdouble fov, GLdouble aspect, GLdouble near, GLdouble far);
```

# Homogene Koordinaten in der Perspektive



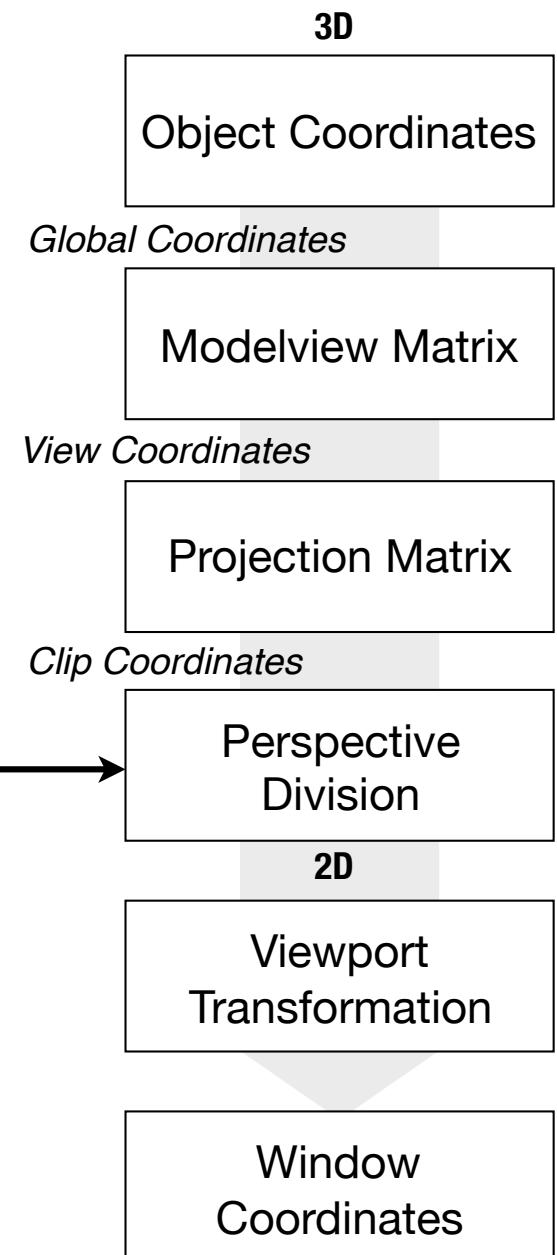
Division durch  $w$  (Homogenisierung) ergibt den  
projizierten Punkt in homogene Koordinaten

# Perspective Projection

- Erst die Division durch die  $w$ -Komponente werden die  $x, y$  Werte projiziert
- Division durch  $w$  (homogenisierung **ergibt** erst Projektion)
- Dies wird erst spät in der Pipeline durchgeführt

$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ p_z/n \end{pmatrix} \rightarrow \begin{pmatrix} \frac{n}{p_z} \cdot p_x \\ \frac{n}{p_z} \cdot p_y \\ n \\ 1 \end{pmatrix}$$

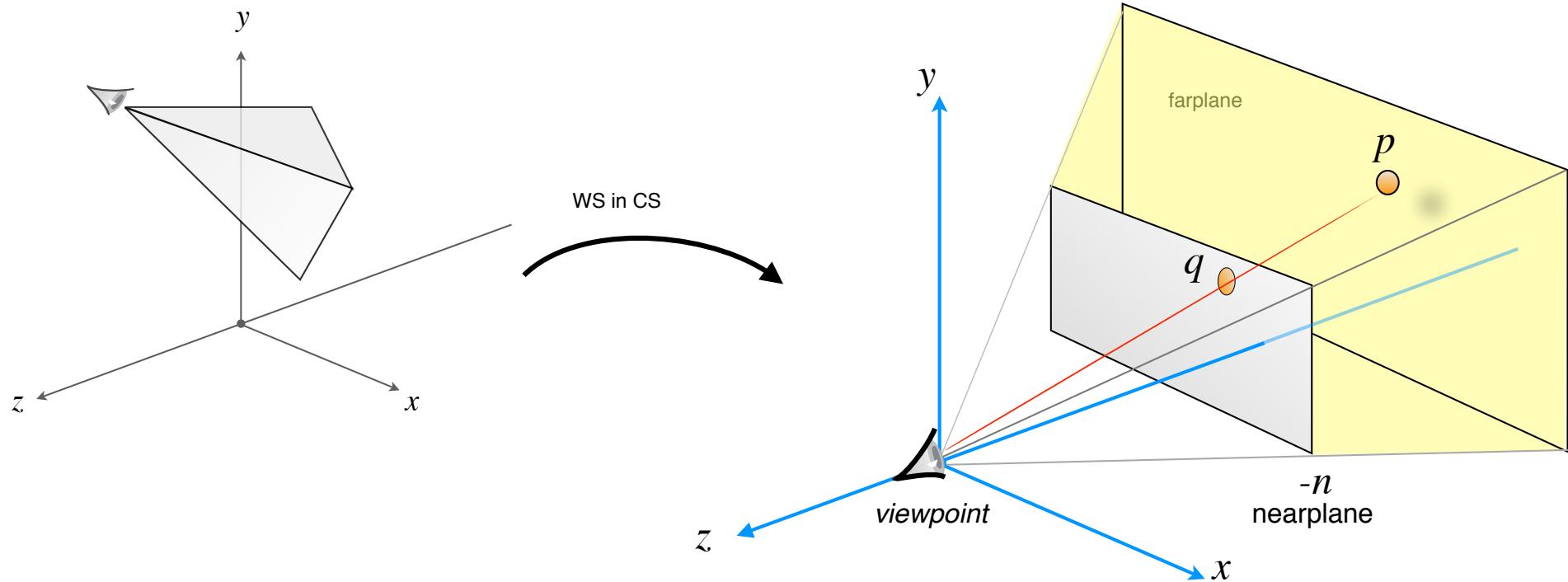
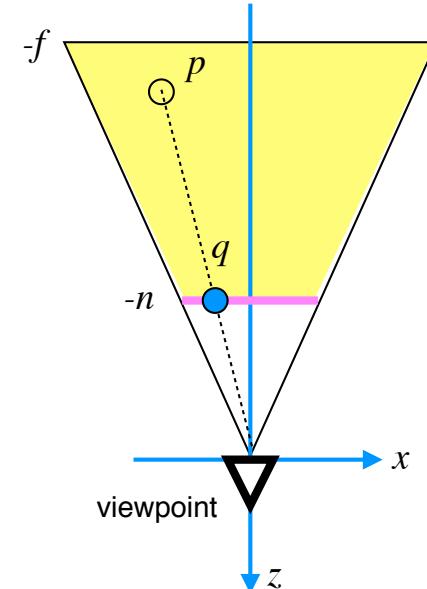
**DIVISION DURCH W**



# Perspective Projection

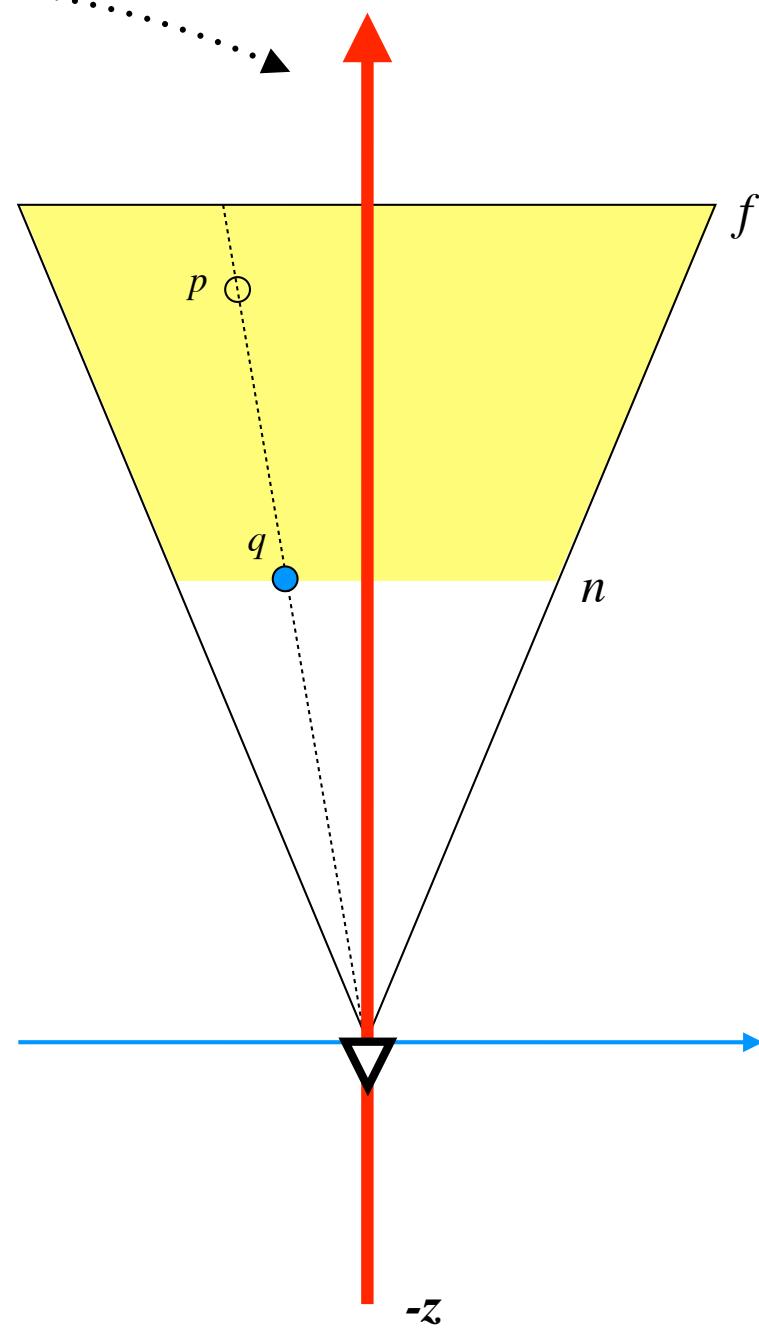
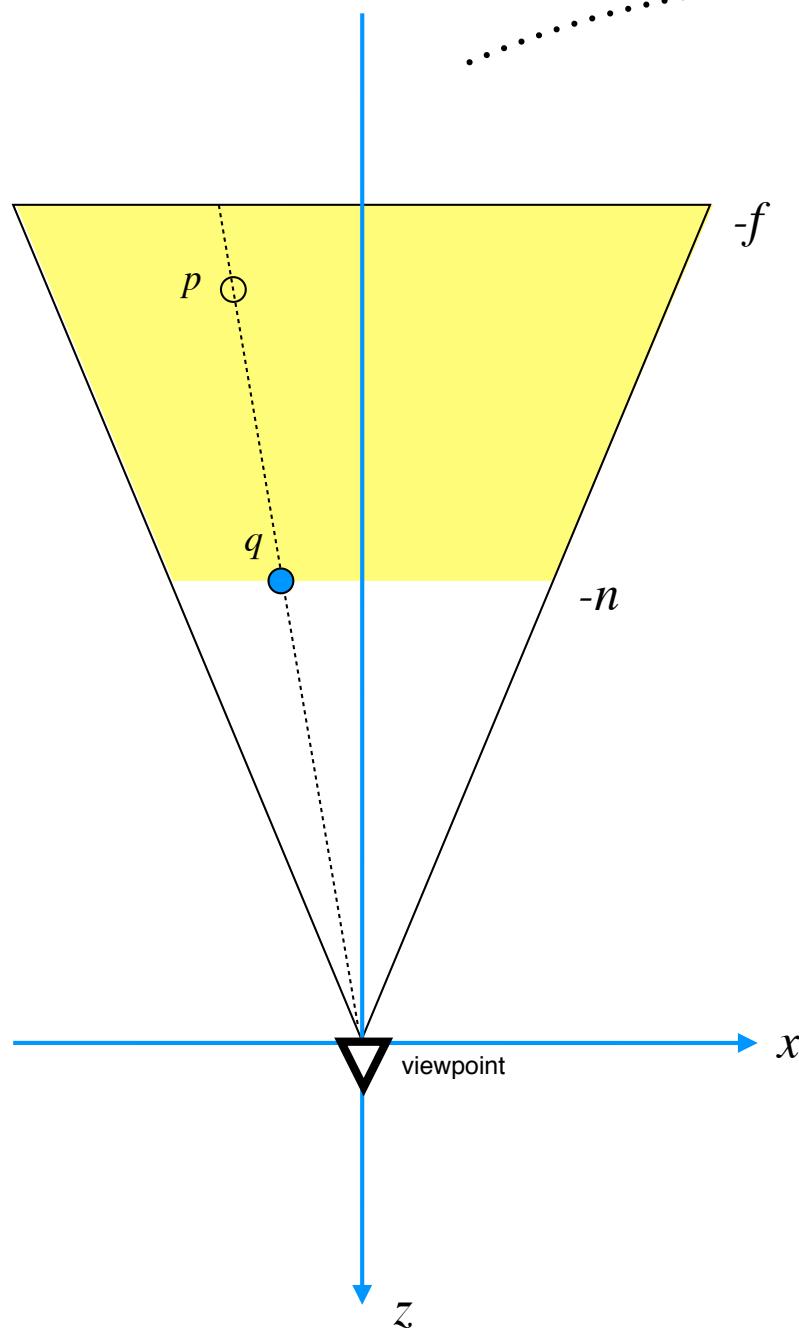
- Transformation vom beliebigen Viewing Frustum in:

- Viewpoint im Nullpunkt
- UV-Viewplane koplanar zur x,y Ebene
- damit: Viewing Frustum in der negativen z-Ebene
- Punkt  $p=(p_x, p_y, p_z)$  : abbilden auf  $q=(q_x, q_y, -n)$



# Perspektivische Projektion

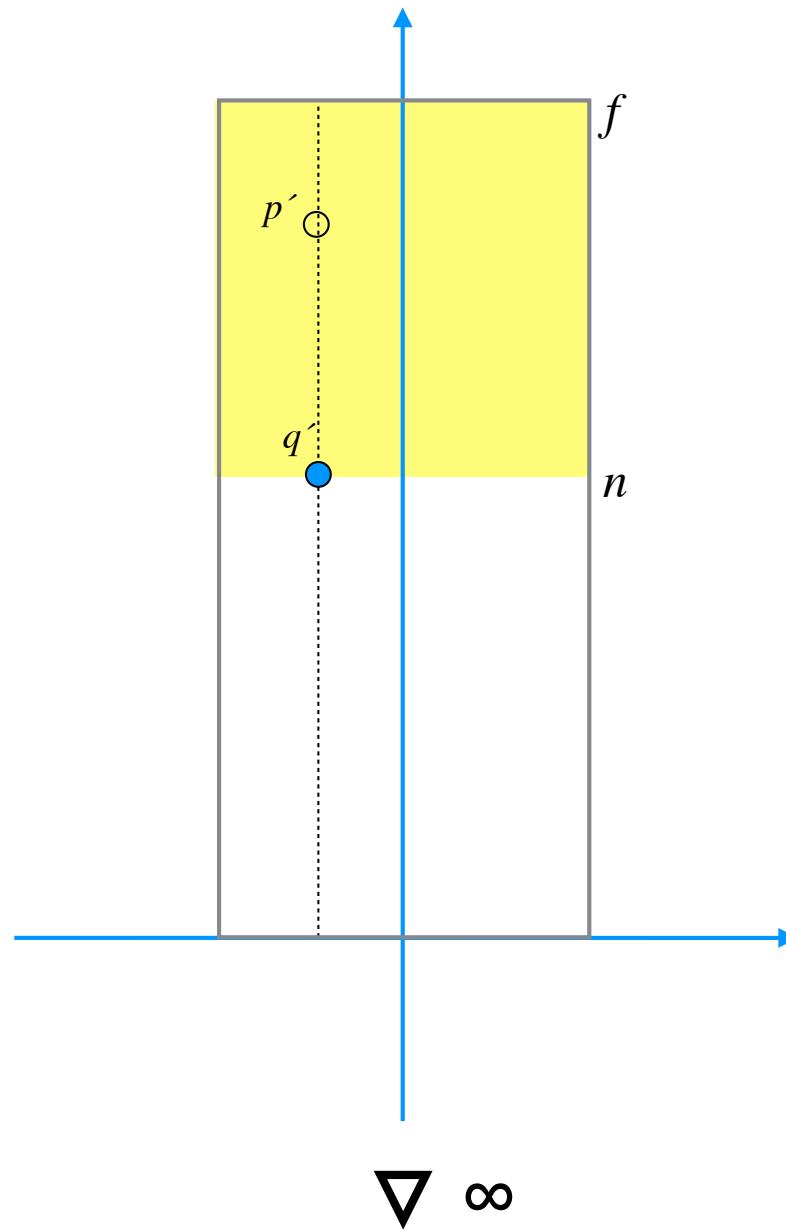
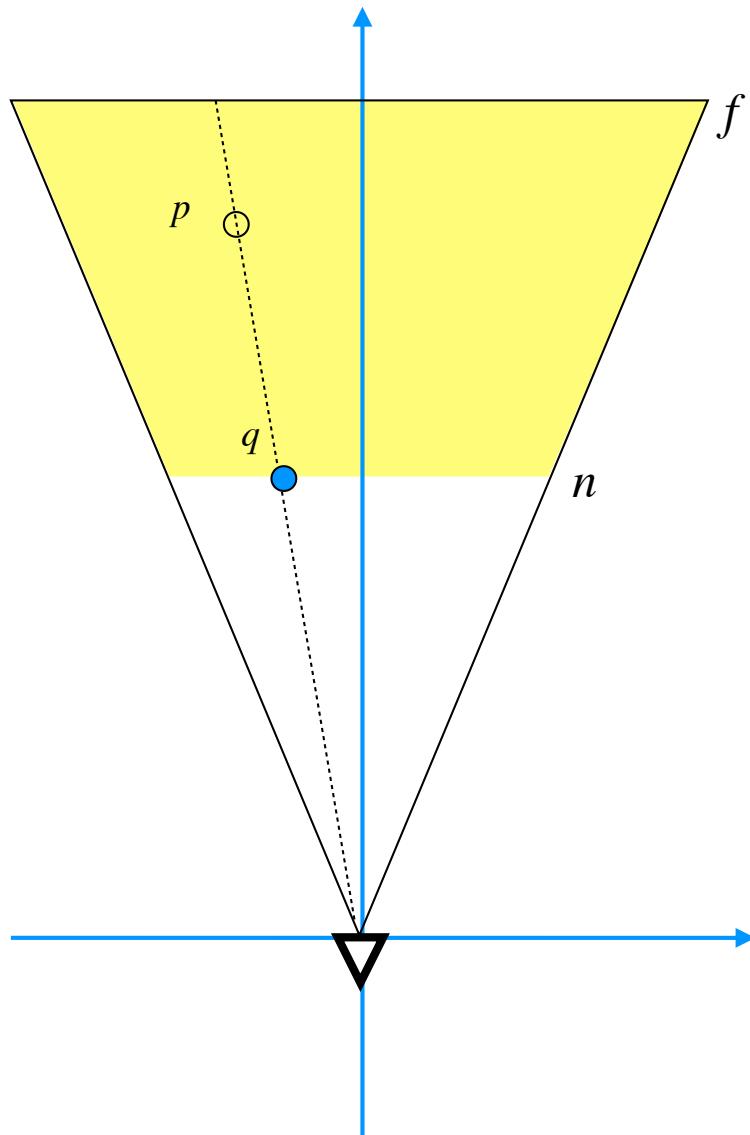
Z Spiegelung



# Perspektivische Projektion

Perspektivische Verzerrung der  $x$ - und  $y$  Koordinaten

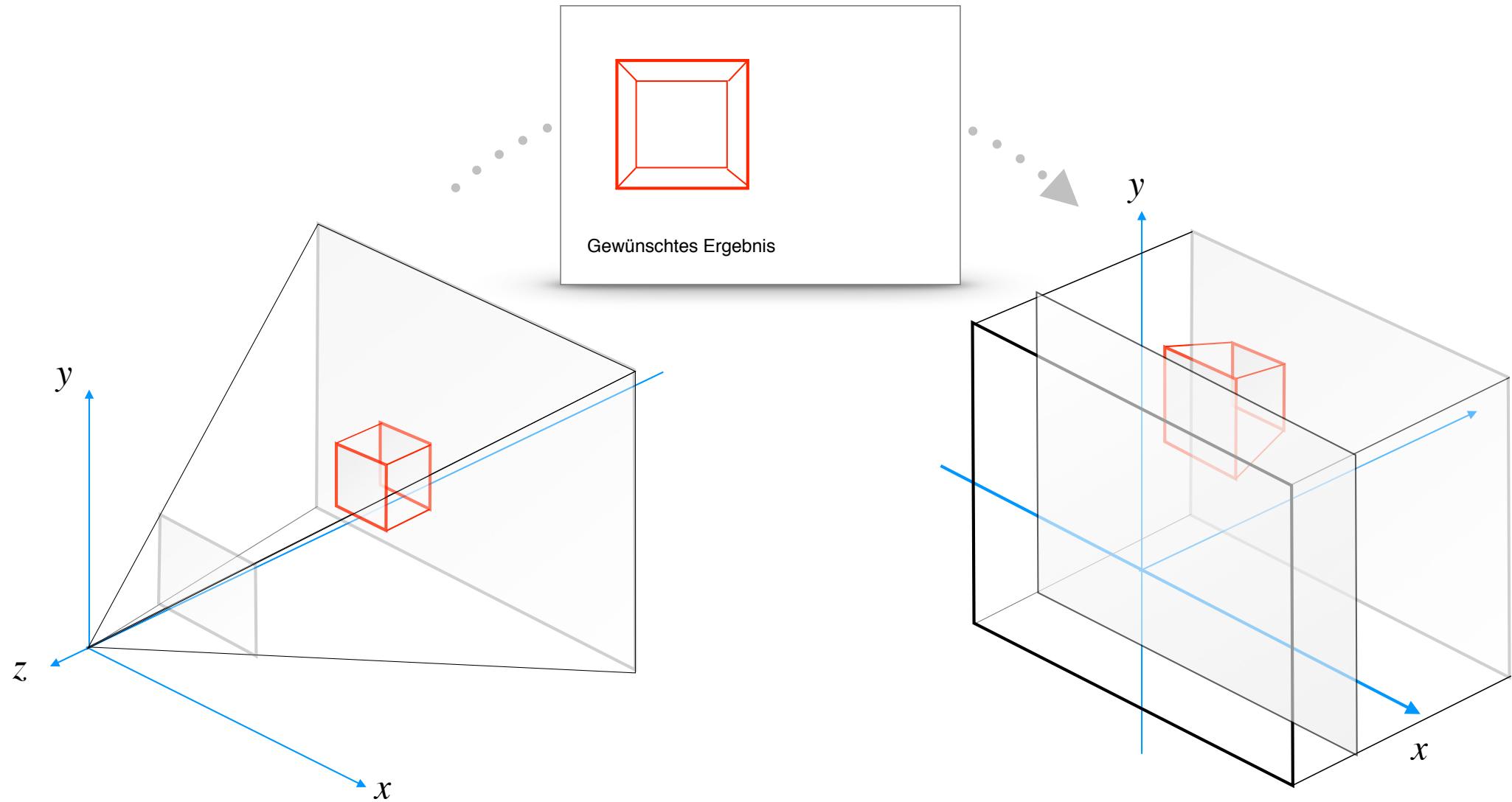
$z$ -abhängige Skalierung von  $x,y$



# Perspective Projection

Perspektivische Verzerrung der  $x$ - und  $y$  Koordinaten

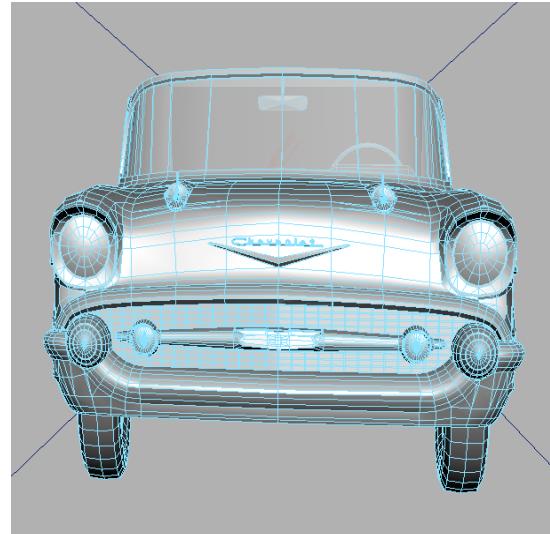
$z$ -abhängige Skalierung von  $x, y$



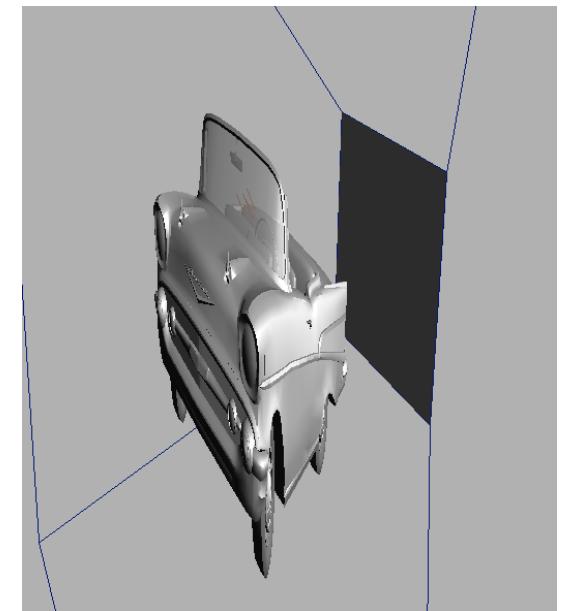
- was passiert mit den Objekten im frustum ?



perspective view



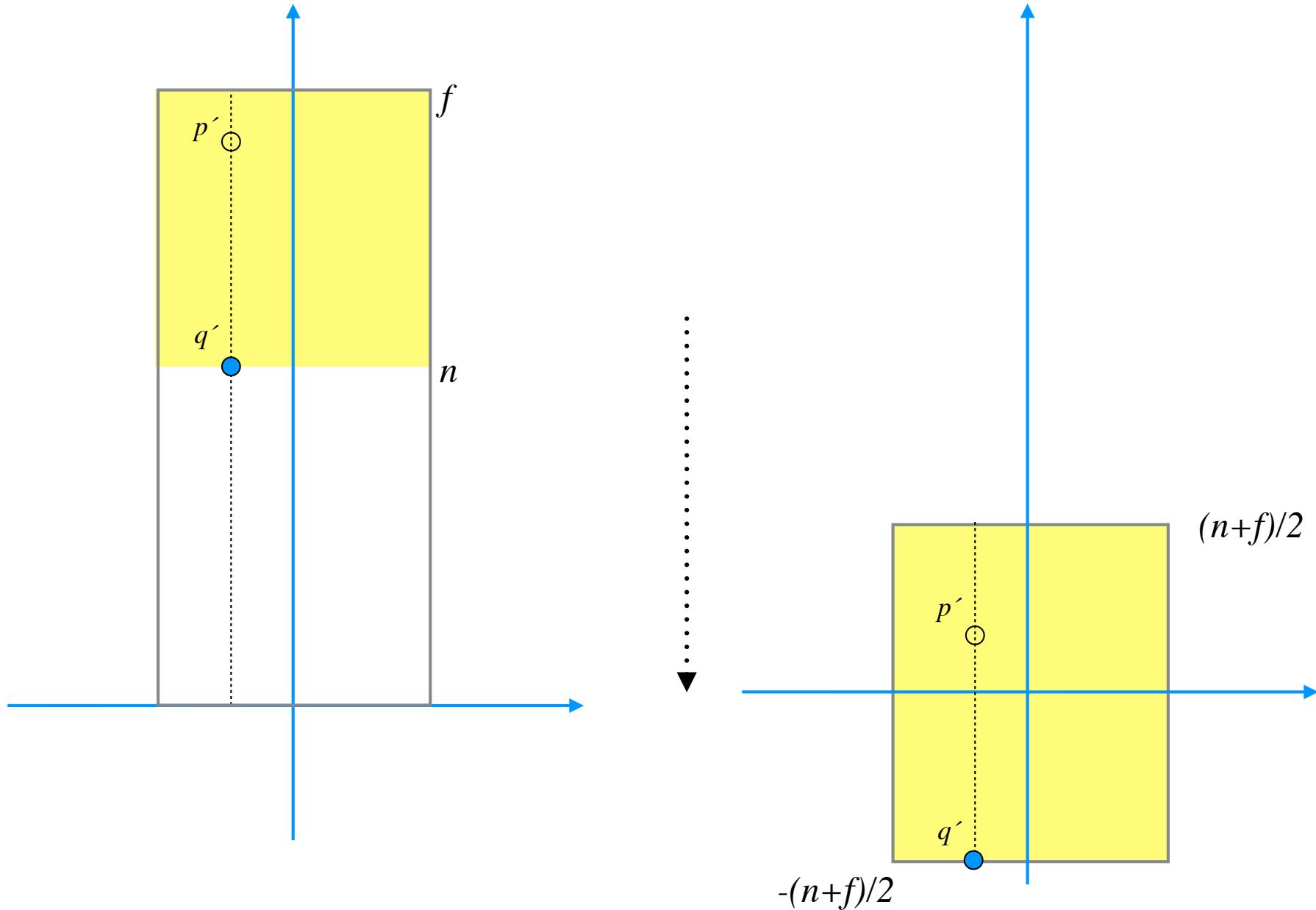
front perspective view



Transformation to unit cube

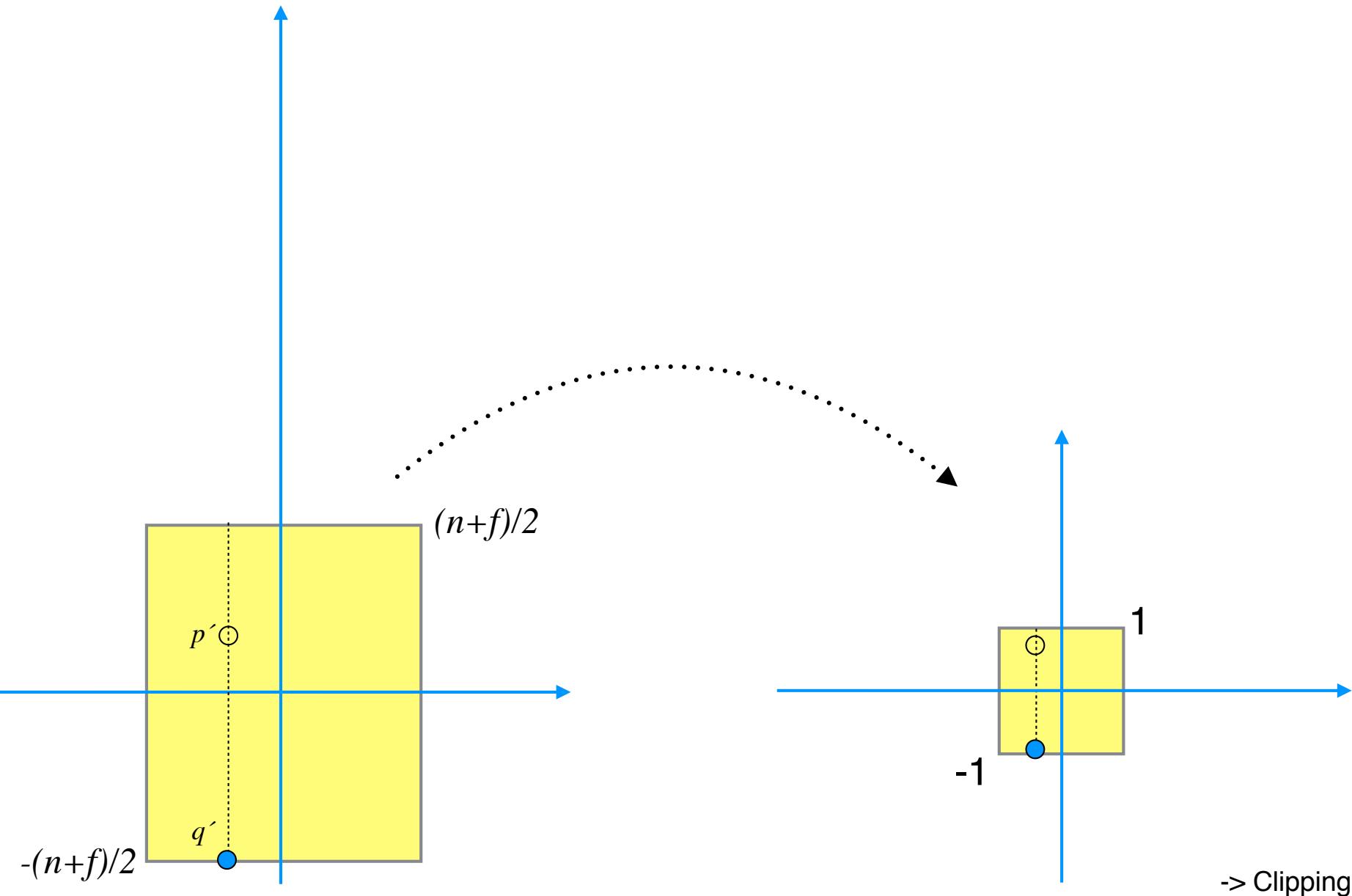
# Perspektivische Projektion

Analog zur orthogonalen Transformation, zentrieren: Translation (nur  $z$ -Werte)



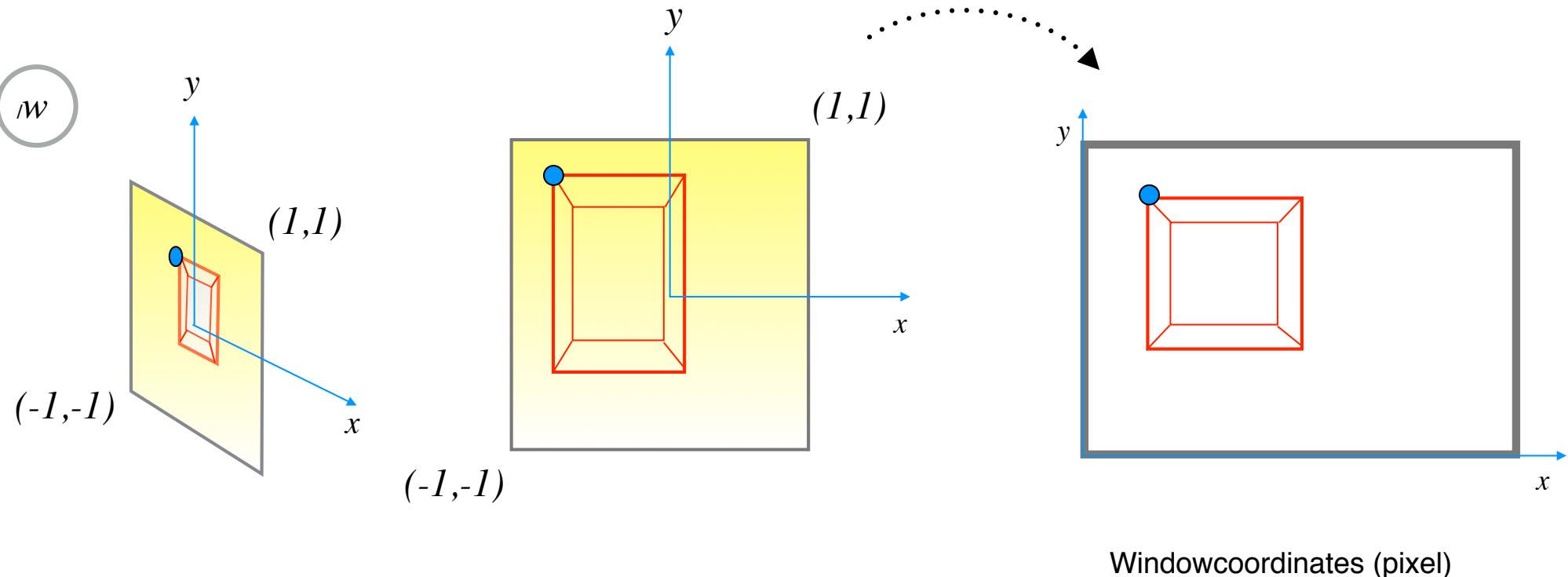
# Perspektivische Projektion

Analog zur orthogonalen Transformation, Skalierung auf Einheitswürfel



# Perspective Projection

Analog zur orth. Transformation : Viewport Transformation



Analog zur Orthogonalen Transformation,  $z$  Werte spiegeln

$z$  Achse spiegeln : 
$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

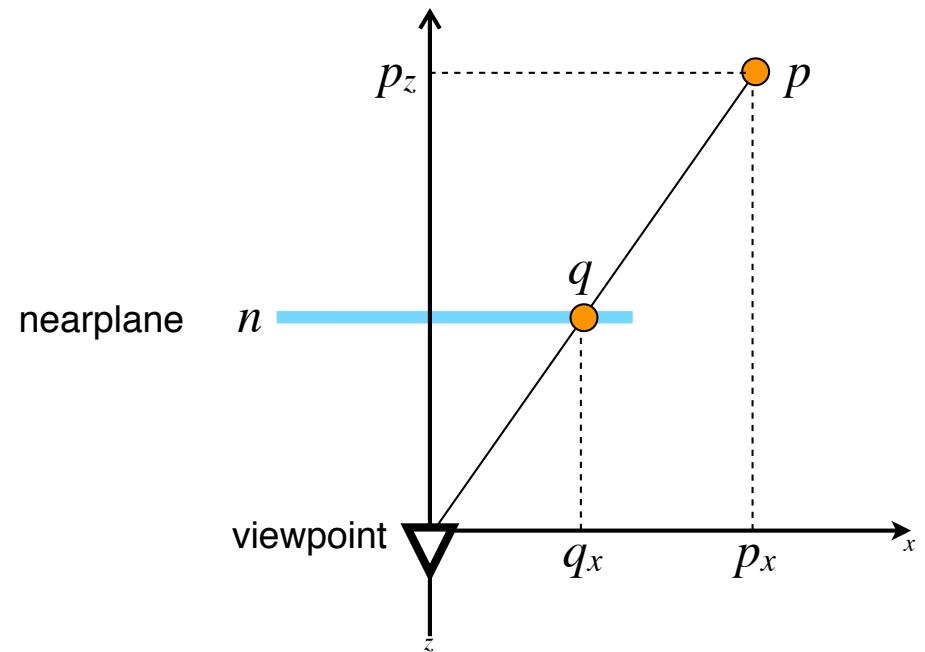
# Perspective Projection

- Punkt  $p=(p_x, p_y, p_z)$  : abbilden auf  $q=(q_x, q_y, n)$

1 Strahlensatz

$$\frac{q_x}{p_x} = \frac{n}{p_z} \rightarrow q_x = n \cdot \frac{p_x}{p_z}$$

$$\frac{q_y}{p_y} = \frac{n}{p_z} \rightarrow q_y = n \cdot \frac{p_y}{p_z}$$



2 Wie kriegen wir das in einer 4X4 Matrix rein

# Perspective Projection

$$\frac{q_x}{p_x} = \frac{n}{p_z} \rightarrow q_x = n \cdot \frac{p_x}{p_z}$$

$$\frac{q_y}{p_y} = \frac{n}{p_z} \rightarrow q_y = n \cdot \frac{p_y}{p_z}$$

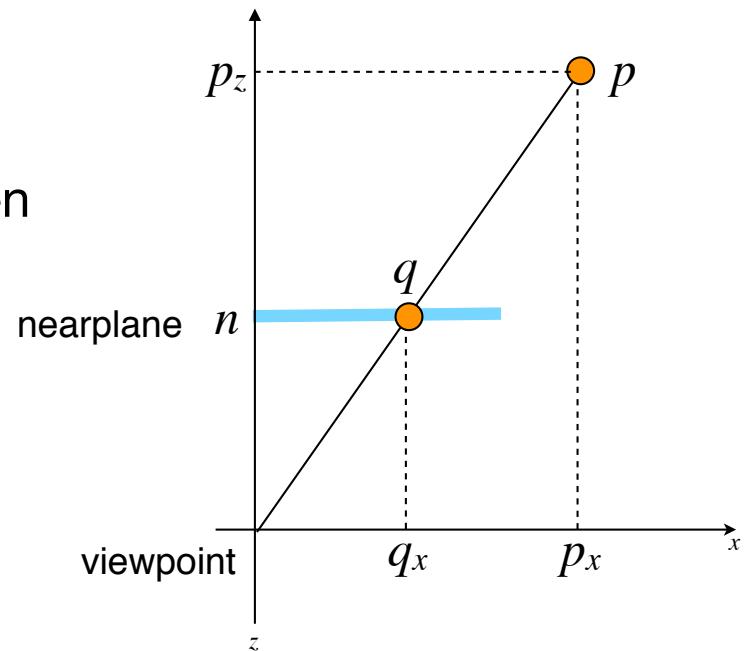
$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} n \cdot p_x \\ n \cdot p_y \\ n \cdot p_z \\ p_z \end{pmatrix} \xrightarrow{\text{homog.}} \frac{1}{p_z} \rightarrow \begin{pmatrix} (n \cdot p_x)/p_z \\ (n \cdot p_y)/p_z \\ n \\ 1 \end{pmatrix}$$

! PROBLEM ?

# Perspective Projection

- Was passiert mit den  $z$  Werten
- Integrierbar in einer  $4 \times 4$  Matrix
- Tiefeninformation (Sichtbarkeit) soll nicht verlorengehen
- Ansatz:

$$\begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} n \cdot p_x / p_z \\ n \cdot p_y / p_z \\ ? \\ 1 \end{pmatrix}$$

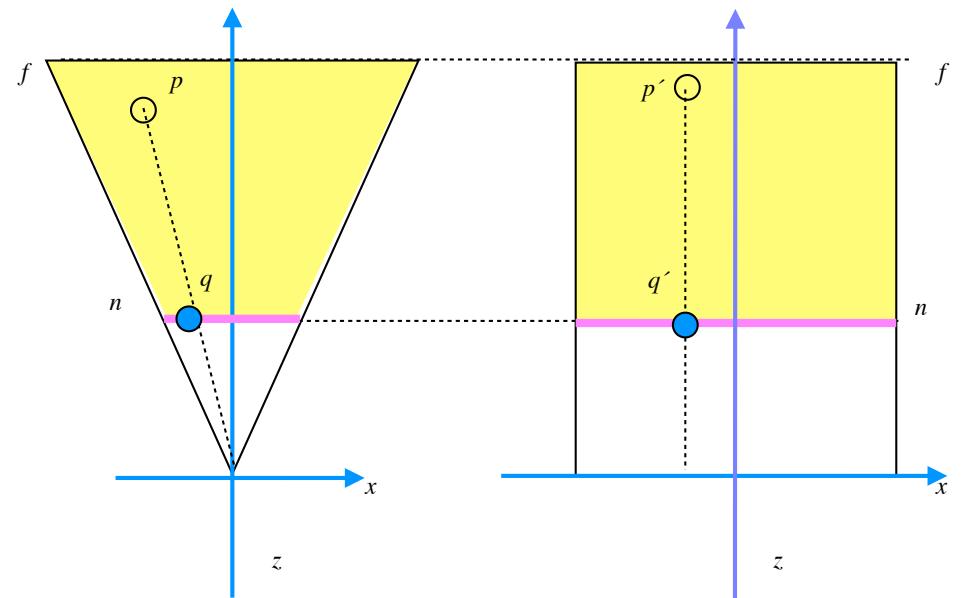


$$\begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} np_x / p_z \\ np_y / p_z \\ \sim p_z \\ 1 \end{pmatrix}$$

Ein Wert welcher Information über die Tiefe enthält.

- $x$  und  $y$  Werte gemäß Projektion verzerren
- Die  $z$  Werte müssen folgende Bedingungen erfüllen
  - Die near Werte ( $n$ ) werden auf near ( $n$ ) abgebildet
  - Die far Werte ( $f$ ) werden auf far ( $f$ ) abgebildet
  - Alle  $z$  Werte dazwischen behalten ihre Ordnung (Monoton steigend) / Proportional  $p_z$
- Alles muss in einer 4X4 Matrix integrierbar sein

$$\begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix} = \begin{pmatrix} np_x/p_z \\ np_y/p_z \\ \sim p_z \\ 1 \end{pmatrix}$$



# Perspective Projection

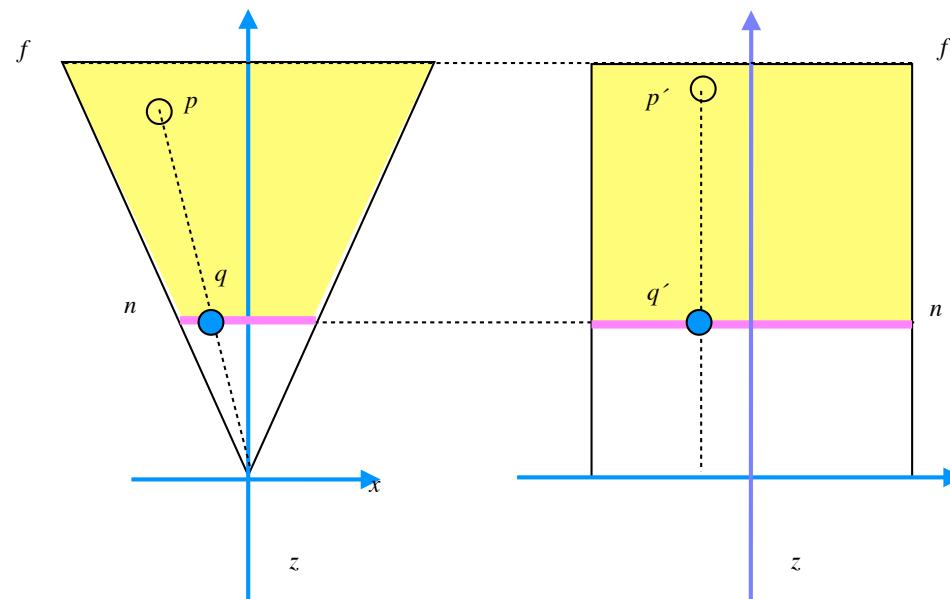
- Die  $z$  Werte müssen folgende Bedingungen erfüllen
  - Die near Werte ( $n$ ) werden auf near ( $n$ ) abgebildet
  - Die far Werte ( $f$ ) werden auf far ( $f$ ) abgebildet
  - Alle  $z$  Werte dazwischen behalten ihre Ordnung (Monoton steigend) / Proportional  $p_z$
- Alles muss in einer 4X4 Matrix integrierbar sein

$$\frac{n \cdot f}{p_z} \begin{cases} p_z = n \rightarrow f \\ p_z = f \rightarrow n \end{cases} \rightarrow (n + f) - \frac{n \cdot f}{p_z}$$

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -n \cdot f \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} n \cdot p_x \\ n \cdot p_y \\ (n + f) \cdot p_z - n \cdot f \\ p_z \end{pmatrix}$$

- 1 Projektion von x,y Koordinaten (z-abhängige Skalierung)

$$M = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -n \cdot f \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

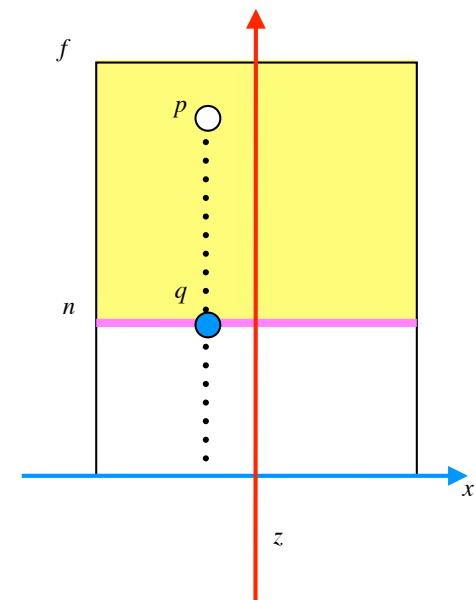


# Wichtiger Gedanke über die Z Werte

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -n \cdot f \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} n \cdot p_x \\ n \cdot p_y \\ (n+f) \cdot p_z - n \cdot f \\ p_z \end{pmatrix}$$

$$\begin{pmatrix} n \cdot p_x \\ n \cdot p_y \\ (n+f) \cdot p_z - n \cdot f \\ p_z \end{pmatrix} \Rightarrow \begin{pmatrix} n \cdot p_x / p_z \\ n \cdot p_y / p_z \\ n + f - n \cdot f / p_z \\ 1 \end{pmatrix}$$

homogenisiert



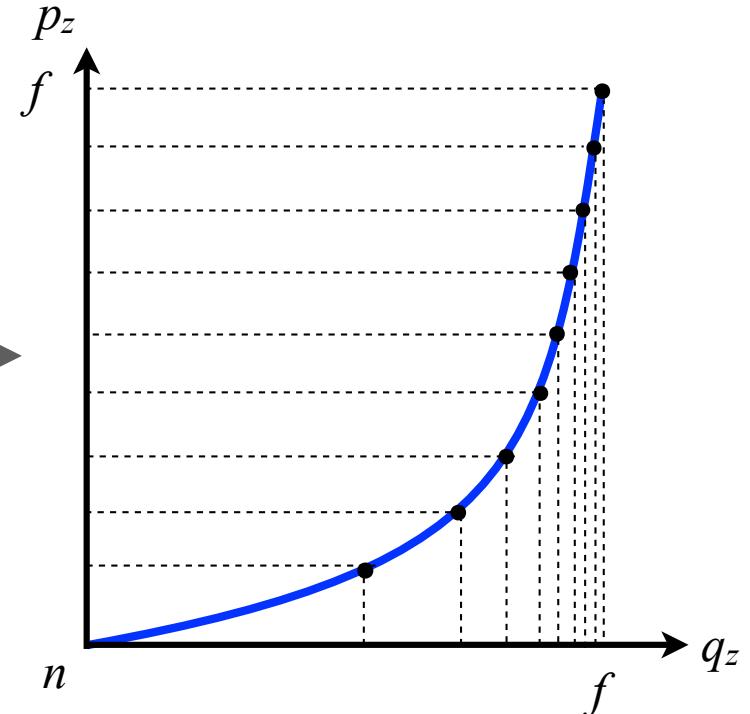
$$\begin{pmatrix} q_x \\ q_y \\ q_z \\ w \end{pmatrix} = \begin{pmatrix} np_x/p_z \\ np_y/p_z \\ n + f - nf/p_z \\ 1 \end{pmatrix}$$

- nicht lineare Skalierung der Z-Koordinaten
- $n$  wird auf  $n$  abgebildet
- $f$  wird auf  $f$  abgebildet



(?) Erkenntnis?

- „Vorne“ gibt es eine höhere Auflösung als „hinten“
- Achtung: Flackern bei fernen Objekten
- Was bedeutet das praktisch?
- Es ist sehr wichtig near- und far- Clippingplane intelligent zu bestimmen.



$$q_z = n + f - n \cdot f / p_z$$

# Normalization for Perspective Projection 3D

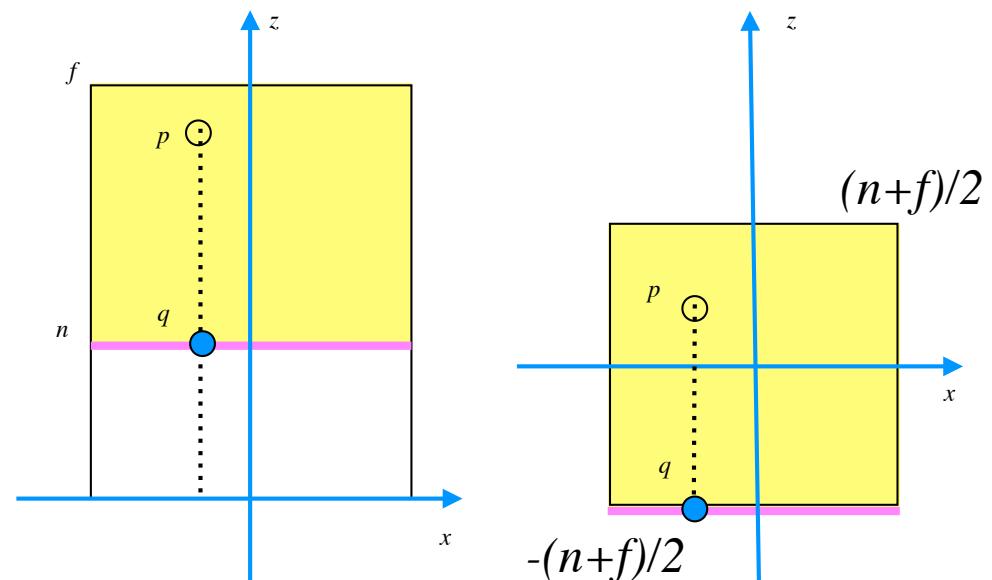
- 2 Da Quader schon bzgl.  $x, y$  zentriert nur noch um  $z$  verschieben

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

Translation bekannt aus orthogonaler Projektion

$$M = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & \frac{n+f}{2} & -n \cdot f \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Eingebaut in die bisherige Matrix  
Von rechts ran multiplizieren



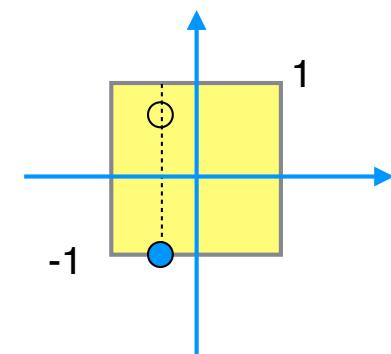
3 Skalierung der Werte zu einem Einheitswürfel

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

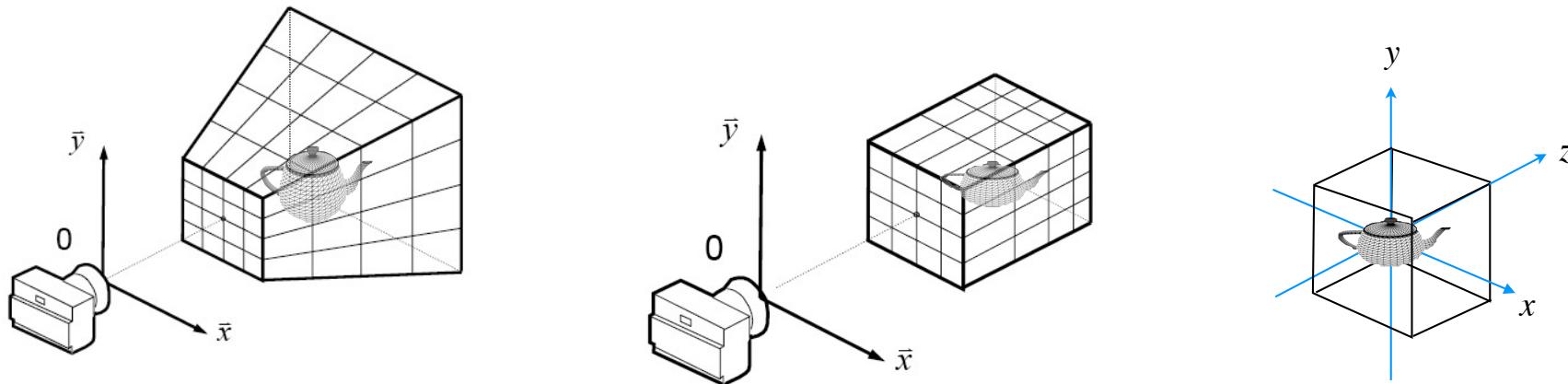
Skalierung bekannt aus orthogonaler Projektion

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

Eingebaut in die bisherige Matrix  
 Von rechts ran multiplizieren



# Perspective Projection

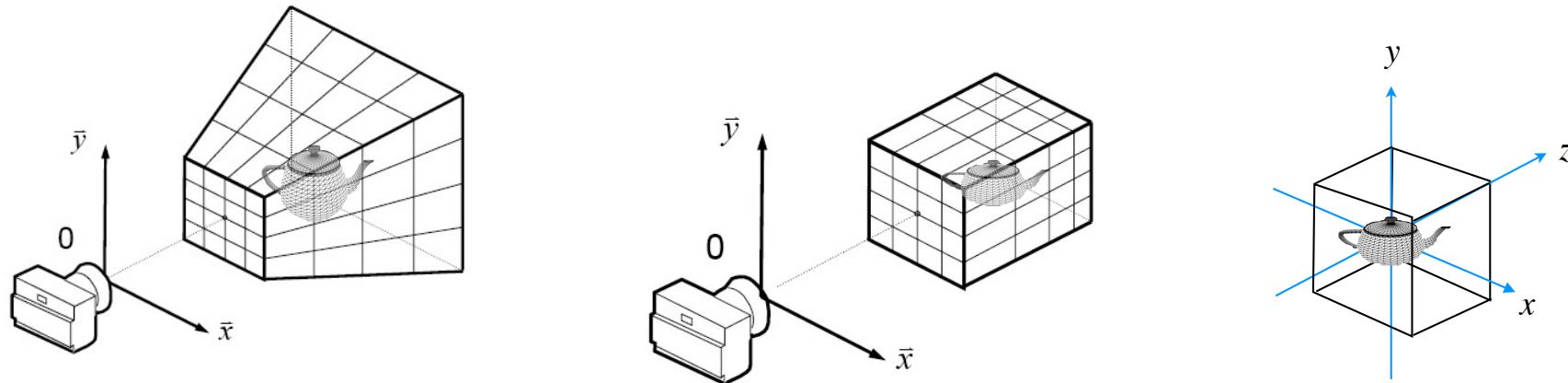


$$M = \begin{pmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -\frac{2n \cdot f}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$q = M_{Ortho} \cdot M_{Persp} \cdot M_{R \sim L} \cdot V \cdot T \cdot \dots \cdot S \cdot R \cdot p$$

$M$

# Perspective Projection

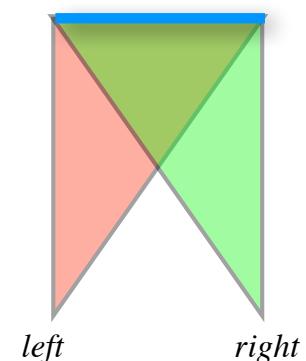
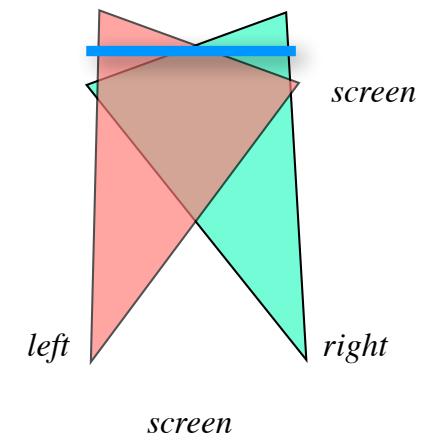
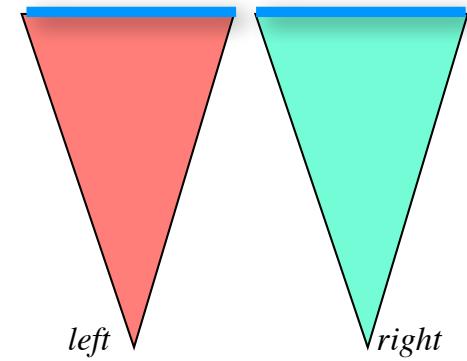


$$M = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -\frac{2n \cdot f}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$q = M_{Ortho} \cdot M_{Persp} \cdot M_{R \sim L} \cdot V \cdot T \cdot \dots \cdot S \cdot R \cdot p$$

$M$

- Stereoskopische Filme
- Zwei Kameras mit einem Abstand aufstellen
- Jedes Frame für das rechte und linke Auge rendern
- Frage ist: Wie erzeugen wir ein Bild daraus?
- Korrespondierende Pixel (Rechts / Links)
- Rotation der Kamera ist eine schlechte Lösung
- Lösung: Scherung des Sichtvolumens !
- Achtung: geht meistens nicht in den üblichen 3D Programmen
- Sichtvolumen muss selber erzeugt werden



# Stereoscopic Projection

Sei  $q_x = \frac{l+r}{2}$  Genau in der Mitte

Und wird damit auf die z-Achse abgebildet

Und  $p$  ein beliebiger Punkt

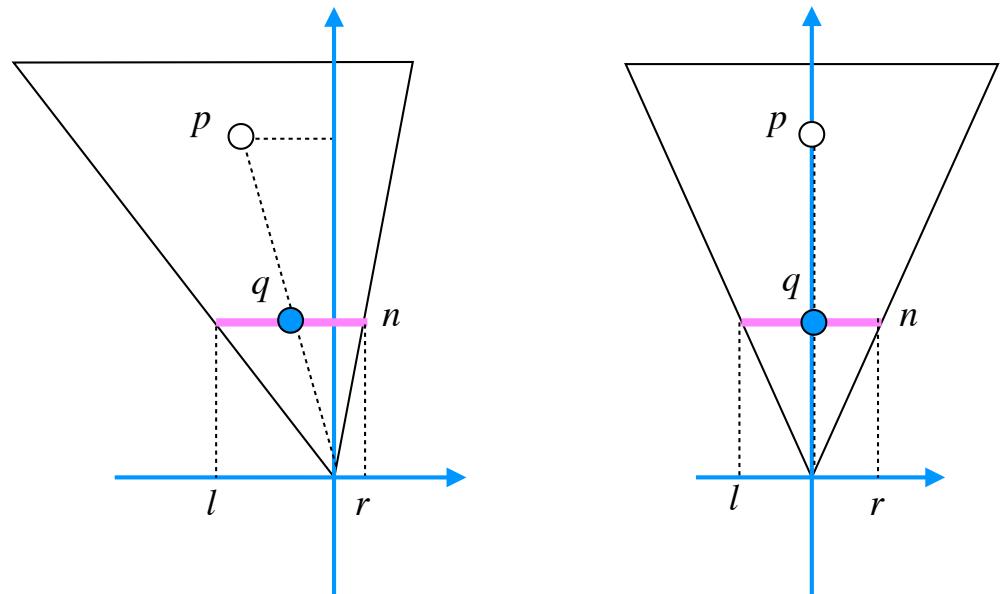
$$\frac{q_x}{p_x} = \frac{n}{p_z}$$

$$p_x = \frac{q_x}{n} \cdot p_z$$

$$p_x = \frac{l+r}{2n} \cdot p_z \quad \text{Benötigte Translation um } x$$

$$p_y = \frac{t+b}{2n} \cdot p_z \quad \text{Benötigte Translation um } y$$

$$M_{SH} = \begin{pmatrix} 1 & 0 & -\frac{r+l}{2n} & 0 \\ 0 & 1 & -\frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{Scherungsmatrix}$$



# Open GL Pipeline

- Lokales Koordinatensystem
- Vertex (x,y,z,w) / Normalen (x,y,z,w) / Farbe (r,g,b,a) / Material (amb. / diffuse / spec.) / Textur (u,v) /...
- Objekte in das globale Koordinatensystem transformieren
- Normalen mit der transponierten Inversen Matrix
- Kamera aufstellen (Position/ lookat/ up-vector)
- Transformation in das Kamerakoordinatensystem
- Auge im Ursprung / Blick entlang der negativen Z-Achse / Rechtssystem
- Beleuchtung, Illumination (Phong,...)
- Projektionstransformation
- Rechts ins Linkssystem Transformation vom Persp. Frustum in einem Quader
- Quader skalieren & zentrieren & auf -1/+1
- Culling / Clipping
- z-Werte für die Verdeckung benutzen (Z-Buffer)
- Projektion (Division durch w) (ab hier 2D )
- Viewport Transformation
- x,y Werte (-1...+1) werden auf das View Window (Pixel) abgebildet
- Rastering
- Shading
- Texturing

**3D**

Object Coordinates

Global Coordinates

Modelview Matrix

View Coordinates

Projection Matrix

Clip Coordinates

Perspective Division

**2D** Normalized device Coord.

Viewport Transformation

Window Coordinates