HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Computergrafik 1 (MDI)

4. Koordinatensysteme und Transformationen

Prof. Dr. Dennis Allerkamp
Sommersemester 2025

Koordinatensysteme im Überblick

Euklidisches Koordinatensystem

- OpenGL benutzt ein *3-dimensionales Euklidisches Koordinatensystem*
- Drei *senkrecht aufeinander stehende Achsen* (x, y und z)
  - *x-Achse* zeigt nach rechts
  - *y-Achse* zeigt nach oben
  - *z-Achse* in Richtung des Beobachters
- Es ist ein „*rechtshändiges*“ Koordinatensystem

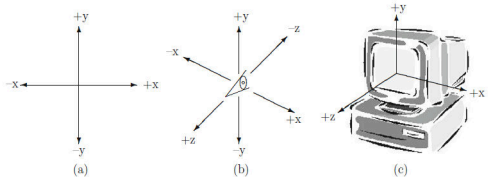
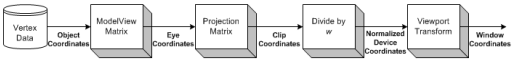


Bild 7.1: Die Definition des Koordinatensystems in OpenGL aus verschiedenen Perspektiven: (a) aus der Perspektive des Augenpunkts, (b) aus der Perspektive eines dritten Beobachters, der nach rechts oben versetzt ist und auf den Ursprung des Koordinatensystems blickt, in dem der Augenpunkt standardmäßig sitzt, (c) aus einer ähnlichen Perspektive wie in der Mitte, aber jetzt mit Blick auf den Bildschirm

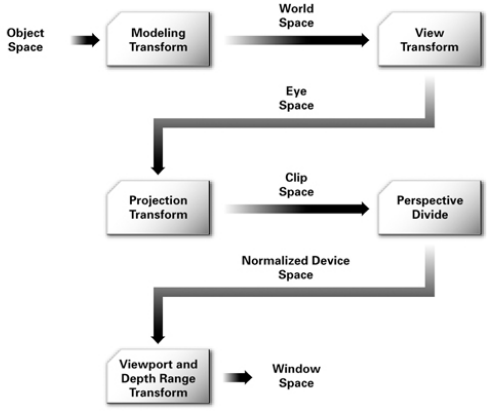
Computergrafik und Bildverarbeitung Nischwitz 2011 S.122

Koordinatensysteme im Überblick

- Während der Pipeline durchläuft OpenGL mehrere Koordinatensysteme und Transformationen



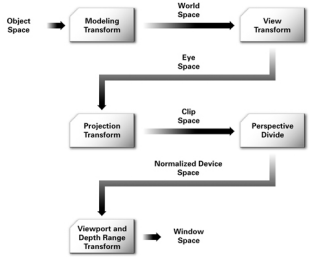
http://www.songho.ca/opengl/gl_transform.html



Computergrafik und Bildverarbeitung Nischwitz 2011 S.123

Kurzer Überblick der Koordinatensysteme

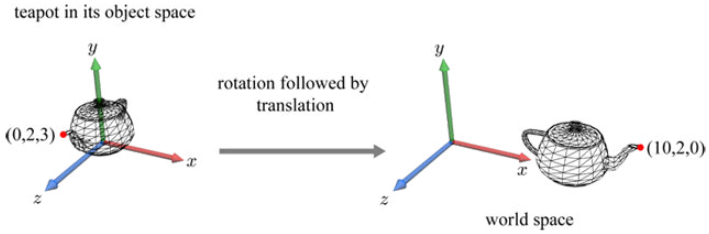
- Object space / Objektkoordinaten / Lokale Koordinaten
  - Koordinaten, in denen die 3D-Objekte *lokal definiert* sind
- World space / Weltkoordinaten / Globale Koordinaten
  - *Gemeinsames* Koordinatensystem aller Objekte nach der Model-Transformation
- Eye space / Kamerakoordinaten / Augenpunktkoordinaten
  - Koordinatensystem nach der View-/Kameratransformation
  - enthält Ort und Richtung der Objekte *relativ zur Kamera*
- Clip space / Projektionskoordinaten / Kamerakoordinaten
  - Koordinatensystem nach der Projektionstransformation
  - enthält *Perspektive* der Kamera
- Normalized device space / Normalized device coordinates (NDC) / Normierte Koordinaten
  - Koordinaten nach der Division (perspective divide) der Projektionskoordinaten durch w in einen Wertebereich -1..1
- Window space / Bildschirmkoordinaten
  - Koordinaten, die die Szene nach der Viewport-Transformation in der gewählten *Viewport-Fenstergröße* darstellt



Computergrafik und Bildverarbeitung Nischwitz 2011 S.123

Model Transformation

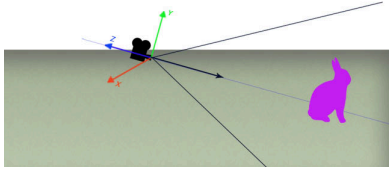
- 3D-Objekte besitzen ein *eigenes lokales Koordinatensystem (Object Space)*
- Mit Modelltransformationen werden sie *verschoben/transliert*, *rotiert* oder *skaliert*
- Nach der Transformation gelangen einzelne Objekte in gemeinsamen *World Space*



https://developer.tizen.org/development/guides/native-application/graphics/opengl-es/vertex-shader

Viewing Transformation

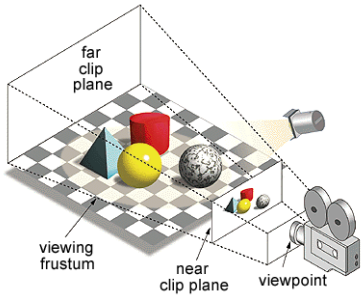
- Die „*Kamera*“ der Szene
- Legt fest von *welchem Punkt* die Szene „fotografiert“ werden soll
- Dazu transformiert die Matrix alle Vertices der Szene in *Eye-/View-/Camera Coordinates*
- Kamera im Ursprung (0,0,0), guckt in -z-Richtung
- Es spielt keine Rolle ob die Kamera sich bewegt oder die Szene sich verschiebt



http://www.siliconjazz.net/computer-graphics/opengl/generic-3d-rendering-part3.html

Projektionstransformation

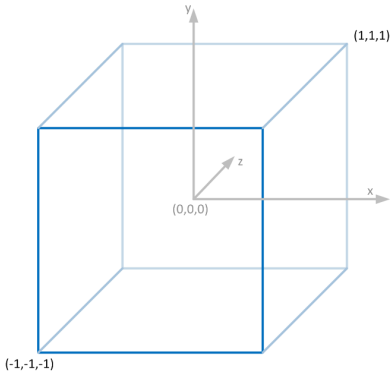
- Eigenschaften wie *Blickwinkel*
  - Entspricht quasi der Auswahl eines *Objektivs einer Kamera*
- Wendet Projektionstransformation auf Eye-Koordinaten an
- Definiert das Sichtbarkeitsvolumen (*view frustrum*)
- Vertices außerhalb des Volumens werden entfernt (*clipping*)



http://encyclopedia2.thefreedictionary.com/View+frustum

Perspective Division und Normierung

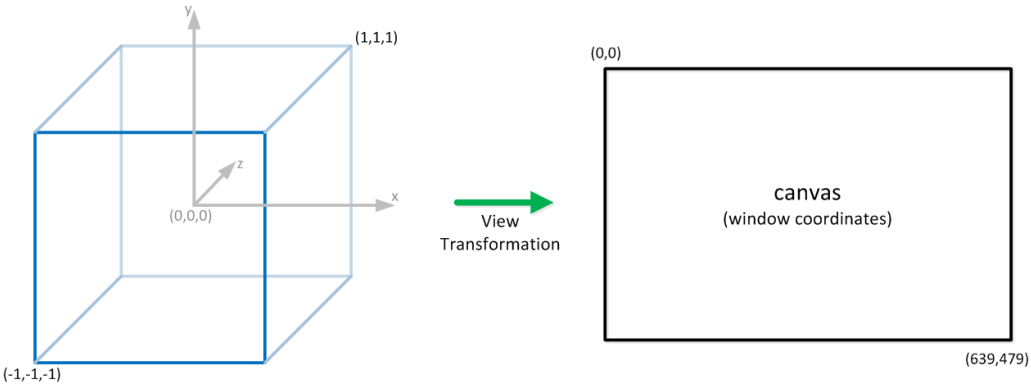
- Besteht aus *zwei Schritten*
  1. x,y und z Koordinaten werden auf [-w,+w] transformiert
  2. x,y und z Koordinaten durch w dividiert
- Die Schritte erzeugen *Normalized Device Coordinates* [-1.0,1.0]
  - NDC im linkshändigen Koordinatensystem



http://www.martinchristen.ch/webgl/tutorial02

Viewport Transformation

- Abhängig von den in Pixel definierten Bildschirmfensters
- Transformiert *NDC zu Bildschirmkoordinaten*



http://www.martinchristen.ch/webgl/tutorial02

Transformationsmatrizen

Mathematische Grundlagen – Homogene Koordinaten

- Punkt im *Euklidschen Raum* kann durch $(x, y, z)T$ definiert werden
- Beliebige Transformation der Punkte kann mit einer 3x3 Matrix beschrieben werden
- Problem: Translationen können durch eine 3x3 Matrix nicht beschrieben werden

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \text{mit} \quad \begin{aligned} x' &= m_{11} \cdot x + m_{12} \cdot y + m_{13} \cdot z \\ y' &= m_{21} \cdot x + m_{22} \cdot y + m_{23} \cdot z \\ z' &= m_{31} \cdot x + m_{32} \cdot y + m_{33} \cdot z \end{aligned}$$

Computergrafik und Bildverarbeitung Nischwitz 2011 S.124

Vierte Komponente w – Homogene Koordinaten

- Für Translationen fehlt hier die *additive Komponente*
- $\rightarrow$ Darstellung durch vier Komponenten x_h, y_h, z_h, w
- w ist *inverser Streckungsfaktor*
- Abbildung einer Position in den Euklidischen Raum:

$$x = \frac{x_h}{w} \qquad y = \frac{y_h}{w} \qquad z = \frac{z_h}{w}$$

Vierte Komponente w

$$x = \frac{x_h}{w} \qquad y = \frac{y_h}{w} \qquad z = \frac{z_h}{w}$$

- *Standardmäßig ist* $w = 1$
- Bei $w = 0.5$ werden die Koordinaten um Faktor 2 gestreckt
  - (2, 4, -3, 1) und (1, 2, -1.5, 0.5) beschreiben denselben euklidischen Ortspunkt (2, 4, -3)
- Eine Division durch $w = 0$ *ist nicht definiert*
  - $=>$ $w = 0$ bildet ins Unendliche ab
- (x, y, z, 0) gelten deshalb als *Richtungsvektoren*
  - z.B. bei Lichtberechnung wichtig (parallele Lichtstrahlen, Sonnenlicht)

Allgemeine Transformationsmatrizen

- Transformation eines Ortsvektors $v = (x, y, z, w)$
- Erreicht durch eine 4x4 Matrix

$$\mathbf{v}' = \mathbf{M}\mathbf{v} \qquad \Leftrightarrow \qquad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Computergrafik und Bildverarbeitung Nischwitz 2011 S.125

Optimierung – Zusammenfassen von Matrizen

- Jeder Vertex v einer Szene muss alle Transformationsstufen durchlaufen,

$$v' = (V \cdot N \cdot P \cdot (S \cdot R \cdot T)*) \cdot v$$

  - Model-, View- bis hin zur Projektionstransformation (P),
  - Normalisierung (N)
  - und der Viewporttransformation (V)
- $(S \cdot R \cdot T)*$ ist eine *beliebige Kombination* von Skalierungen, Rotationen und Translationen
  - (Reihenfolge der affinen Transformationen beispielhaft, variiert je nach Anwendungsfall)

Optimierung – Zusammenfassen von Matrizen

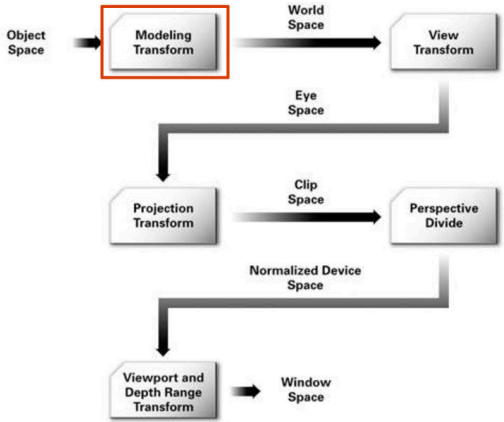
- Effizienzsteigerung erreicht man durch *Zusammenfassung der Transformationsmatrizen* zu einer Gesamtmatrix

$$v' = (V(N(P(S(R(T \cdot v)))))) = (V \cdot N \cdot P \cdot S \cdot R \cdot T) \cdot v$$

- Bei Millionen Vertices eines Objekts werden deshalb *n Vertices* mit nur *einer Gesamttransformationsmatrix* multipliziert.
- Bei einem *Objekt mit einer Hierarche* (z.B. Auto mit Räder) braucht *jeder Knoten* eine eigene Gesamttransformationsmatrix

Model Transformationen

Model Transformationen



Model Transformationen mit Matrizen

- Matrix transformiert *Objekt/Vektor* in ein anderes *Koordinatensystem*

$$\mathbf{v}' = \mathbf{M}\mathbf{v} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

x-Achse, y-Achse, z-Achse, Ursprung

- Mögliche Operationen/ affine Transformationen:
  - Translation,
  - Rotation,
  - Skalierung
  - affine Transformationen = Kollinearität und Parallelität bleiben erhalten
- *Positioniert, orientiert und skaliert* also das Objekt *in der Szene*

Model Transformationen – Translation

- Verschiebt das Koordinatensystem des Objekts
- Dadurch verschiebt sich das Objekt

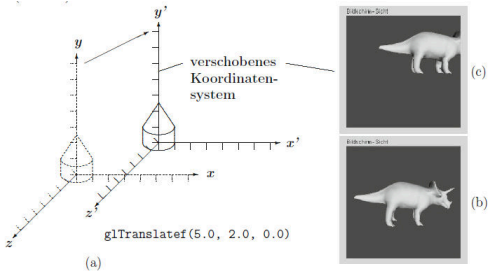


Bild 7.3: Translation des Koordinatensystems in OpenGL: (a) gestrichelt: nicht verschobenes Koordinatensystem (x, y, z), durchgezogen: verschobenes Koordinatensystem (x', y', z'). 3D-Objekt im Ursprung: Zylinder mit aufgestülptem Kegelmantel. (b) Bildschirm-Sicht eines nicht verschobenen Objekts (Triceratops). (c) Bildschirm-Sicht des mit `glTranslatef(5.0, 2.0, 0.0)` verschobenen Objekts.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.128

Model Transformationen – Translation

- *Translation* eines Vertex $v = (x, y, z)$ um einen Richtungsvektor (T_x, T_y, T_z) in kartesischen Koordinaten

$$x' = x + T_x \qquad y' = y + T_y \qquad z' = z + T_z$$

- In Homogenen Koordinaten mit einer 4x4 Matrix

$$\mathbf{v}' = \mathbf{M}\mathbf{v} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

x-Achse, y-Achse, z-Achse, Ursprung

Model Transformationen – Rotation

- Rotiert/dreht das Koordinatensystem *um eine Achse*
- Verläuft *gegen den Uhrzeigersinn*
- α in Grad

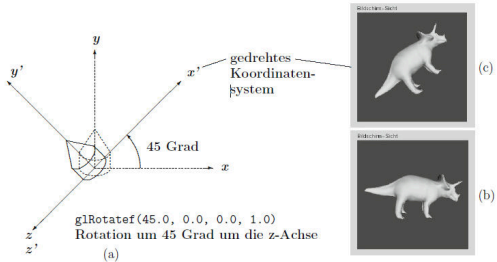


Bild 7.4: Rotation des Koordinatensystems in OpenGL: (a) gestrichelt: nicht gedrehtes Koordinatensystem (x, y, z); durchgezogen: um 45° bzgl. der z–Achse gedrehtes Koordinatensystem (x', y', z'). (b) Bildschirm-Sicht eines nicht gedrehten Objekts (Triceratops). (c) Bildschirm-Sicht des mit `glRotatef(45.0, 0.0, 0.0, 1.0)` gedrehten Objekts.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.130

Model Transformationen – Rotation in 3D

- Rotation um die *jeweiligen euklidischen Achsen* durch jeweilige Rotationsmatrizen:

Rotation um die x–Achse:

$$\mathbf{R^x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.9}$$

Rotation um die y–Achse:

$$\mathbf{R^y} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.10}$$

Rotation um die z–Achse:

$$\mathbf{R^z} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.11}$$

Computergrafik und Bildverarbeitung Nischwitz 2011 S.131

Model Transformationen – Rotation in 3D

- Verkettung in gemeinsamer Matrix:
  - Rotation um Θ um Achse (R_x, R_y, R_z)

$$\begin{bmatrix} \cos\theta + {R_x}^2(1-\cos\theta) & R_xR_y(1-\cos\theta) - R_z\sin\theta & R_xR_z(1-\cos\theta) + R_y\sin\theta & 0 \\ R_yR_x(1-\cos\theta) + R_z\sin\theta & \cos\theta + {R_y}^2(1-\cos\theta) & R_yR_z(1-\cos\theta) - R_x\sin\theta & 0 \\ R_zR_x(1-\cos\theta) - R_y\sin\theta & R_zR_y(1-\cos\theta) + R_x\sin\theta & \cos\theta + {R_z}^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

https://learnopengl.com/Getting-started/Transformations

- Verknüpfung mehrerer Rotationen bzw. Rotation um beliebige Achse ist nicht trivial
  - Gimbal Lock = Rotation um 90° einer Achse kann weitere Rotationen unmöglich machen
  - Gut erklärt: https://youtu.be/zc8b2Jo7mno
- Reale Lösung: Quaternionen $\mathbb{H}$
  - CG3

Model Transformationen – Skalierung

- Getrennte Skalierung wird mit *drei Faktoren* gewährleistet

$$x' = S_x \cdot x \qquad y' = S_y \cdot y \qquad z' = S_z \cdot z$$

- Allgemeine Skalierung in Homogenen Koordinaten:

$$\mathbf{v}' = \mathbf{S}\mathbf{v} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Computergrafik und Bildverarbeitung Nischwitz 2011 S.133

Model Transformationen – Skalierung

- Streckt das Koordinatensystem in der jeweiligen Achse mit s>1
- Oder staucht falls s<1
- Negative Werte *bewirken Spiegelung*
- s=0 ist nicht definiert

Skalierungsfaktor (s)	Effekt
$\|s\| > 1.0$	Streckung / Dimensionen vergrößern
$\|s\| = 1.0$	Dimensionen unverändert
$0.0 < \|s\| < 1.0$	Stauchung / Dimensionen verkleinern
$s = 0.0$	unzulässiger Wert

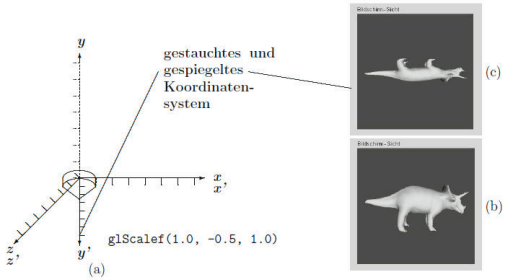


Bild 7.5: Skalierung des Koordinatensystems in OpenGL: (a) gestrichelt: original Koordinatensystem (x, y, z); durchgezogen: bzgl. der y–Achse um den Faktor $|s| = 0.5$ gestauchtes und gespiegeltes Koordinatensystem (x', y', z'). (b) Bildschirm-Sicht eines nicht skalierten Objekts (Triceratops). (c) Bildschirm-Sicht des mit `glScalef(1.0, -0.5, 1.0)` skalierten Objekts.

Reihenfolge der Transformationen – Weltkoordinaten

- Endgültige Lage hängt stark von der *Reihenfolge der Transformationen* ab
- Transformationen sind *nicht unabhängig voneinander*

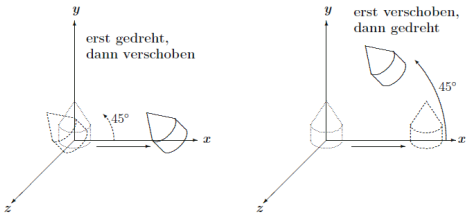


Bild 7.6: Die Reihenfolge der Transformationen in der Denkweise eines festen Weltkoordinatensystems: (a) gepunktet: original Objekt, gestrichelt: erst gedreht bzgl. der z–Achse, durchgezogen: dann verschoben bzgl. der x–Achse. (b) gepunktet: original Objekt, gestrichelt: erst verschoben bzgl. der x–Achse, durchgezogen: dann gedreht bzgl. der z–Achse.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.134

Reihenfolge der Transformationen

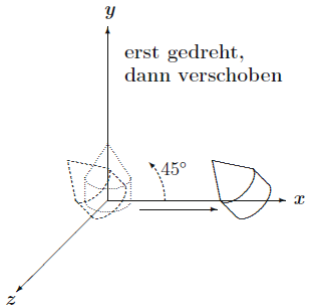
- Verkettung von Transformationen ist durch *Multiplikation der Matrizen* definiert
- Neu hinzukommende Matrizen werden *von links* aufmultipliziert: $v' = M_3 \cdot M_2 \cdot M_1 \cdot v$
  - Grund: Transformationen erfolgen im Programmcode in richtiger Reihenfolge.
- Matrizenmultiplikationen sind *nicht kommutativ*
  - *Die Reihenfolge der Matrizen ist entscheidend*
- Matrizenmultiplikationen sind *assoziativ*
  - $(M_3 \cdot M_2) \cdot M_1 \cdot v = M_3 \cdot (M_2 \cdot M_1) \cdot v$ (*jeweils von rechts aufmultipliziert*)

Beispiel: Erst Rotation, danach Translation

- Initialisiert wird mit einer Einheitsmatrix: $M = I$
- Dann wird die Translationsmatrix von rechts auf die Einheitsmatrix multipliziert: $M = M \cdot T$
- Danach wird die Rotationsmatrix von rechts auf die Transformationsmatrix multipliziert: $M = M \cdot R$
- Anschließend die Vertices von rechts auf die Gesamtmatrix: $Mv = ITRv$
- Zuerst wird also rotiert und anschließend verschoben: $T(Rv)$

Beispiel mit Matrizen in OpenGL

- Matrizen mithilfe von GLM Library generieren
- Gleiches Beispiel:
  - Erst Rotation, dann Translation
- Auch hier gilt: Matrizen von Rechts multiplizieren, d.h. $I \cdot T \cdot R$

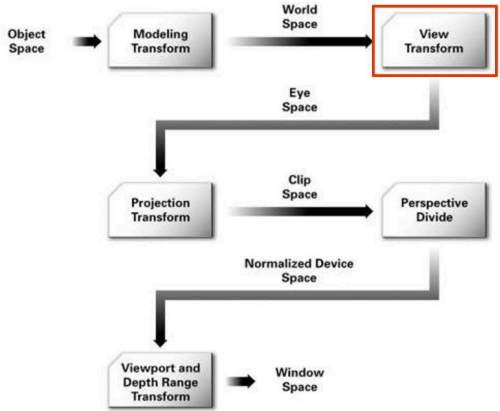


- GLM nimmt im ersten Parameter die zu multiplizierende Matrix an

```
glm::mat4 Model = glm::translate(glm::mat4(1.0f), glm::vec3(1.0f, 0.0f, 0.0f));
Model = glm::rotate(Model, 40.0f, glm::vec3(0.0f, 1.0f, 0.0f));
// Model = Translation * Rotation
```

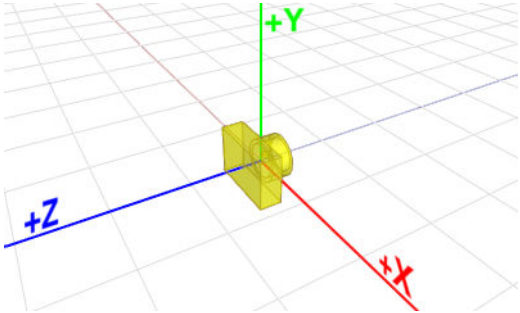
Viewing Transformation

Viewing Transformation



Viewing Transformation

- Ändert die *Position und Blickrichtung* der Kamera
- Einheitsmatrix bildet die Kamera *im Ursprung* ab
- *Blickrichtung ist in Richtung der negative z-Achse* und y-Achse zeigt nach oben



http://www.songho.ca/opengl/gl_camera.html

Viewing Transformation

- Die Viewmatrix *beschreibt die Kamera*
  - Position,
  - Blickrichtung
  - Up-Vektor
- Definiert zusammen mit der Modelmatrix die *Modelviewmatrix* (modelview = view*model)
- Hilfsfunktionen für die Generierung der Matrix hierfür werden meist *LookAt()* genannt
  - z.B. `gluLookAt()`

Viewing Transformation – `gluLookAt()`

- `gluLookAt( Eye.x,Eye.y,Eye.z, Look.x,Look.y,Look.z, Up.x,Up.y,Up.z );`
  - *Eye* = Position
  - *Look* = Blickrichtung
  - *Up* = Richtung, die nach oben zeigt

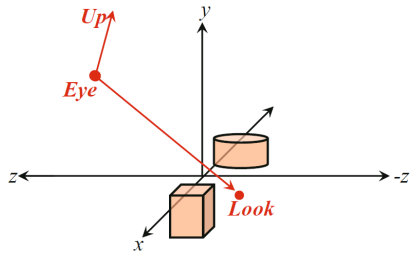


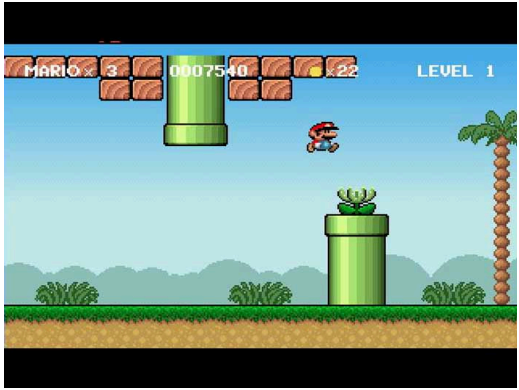
Bild 7.8: Das x, y, z-Koordinatensystem vor der Augenpunkttransformation `gluLookAt()`: Der Augenpunkt befindet sich am Ort **Eye** und der Blick geht von dort in Richtung des Punkts **Look**, in dessen Nähe sich die darzustellenden Objekte der Szene befinden. Die vertikale Ausrichtung des Augenpunkts (bzw. der Kamera) wird durch den Vektor **Up** angegeben.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.136

Viewing Transformation – `gluLookAt()`

- Die `gluLookAt()`-Matrix transformiert gesamte Szene,
  - so dass *Kamera im Ursprung* bleibt
- Technisch wird die Kamera gar nicht bewegt
  - sondern die *Objekte in der Szene*
- Optisch *nicht unterscheidbar*

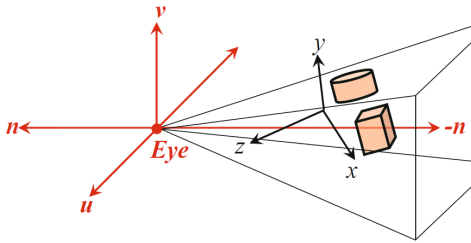
Die Welt bewegt sich nach links und *nicht* die Kamera nach rechts →



https://mozzastryl.wordpress.com/2013/01/20/types-of-game-perspectives/

Viewing Transformation - Herleitung

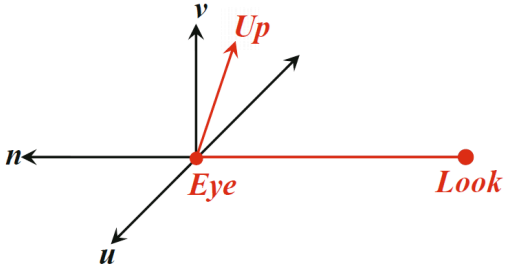
- Achsen der Kamera sind mit u, v, n bezeichnet
- Ursprung des Koordinatensystem liegt am Ort *Eye*
- Transformation besteht aus 2 Schritten:
  1. Drehung im Raum, sodass die Sichtachse n in z gedreht wird und der Vektor v parallel zu y ist
  2. Verschiebung des Eye in den Ursprung (0,0,0)
- Dadurch transformiert sich die Szene automatisch, weil die Viewmatrix mit der Modelmatrix multipliziert wird



Computergrafik und Bildverarbeitung
Nischwitz 2011 S.137

Viewing Transformation - Herleitung

- *Herleitung von n, u und v* durch einfache Operationen (Up-Vektor meist (0,1,0))
  $n = \text{Eye} - \text{Look}$
  $u = \text{Up} \times n$
  $v = n \times u$



- Schließlich *Normierung der Vektoren*

$$\mathbf{n}' = \frac{\mathbf{n}}{|\mathbf{n}|} = \frac{1}{\sqrt{\mathbf{n_x^2} + \mathbf{n_y^2} + \mathbf{n_z^2}}} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$\mathbf{u}' = \frac{\mathbf{u}}{|\mathbf{u}|} = \frac{1}{\sqrt{\mathbf{u_x^2} + \mathbf{u_y^2} + \mathbf{u_z^2}}} \cdot \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$$

$$\mathbf{v}' = \frac{\mathbf{v}}{|\mathbf{v}|} = \frac{1}{\sqrt{\mathbf{v_x^2} + \mathbf{v_y^2} + \mathbf{v_z^2}}} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

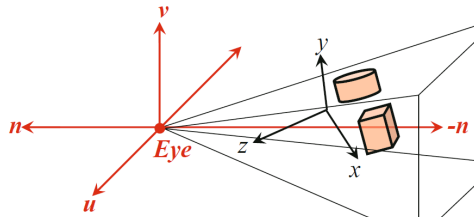
Viewing Transformation - Drehmatrix

- Gesuchte *Drehmatrix* lautet:
  (Die Zeilen definieren die Achsen)

$$\mathbf{M_R} = \begin{pmatrix} u'^T \\ v'^T \\ n'^T \end{pmatrix} = \begin{pmatrix} u'_x & u'_y & u'_z \\ v'_x & v'_y & v'_z \\ n'_x & n'_y & n'_z \end{pmatrix}$$

- Mit *Up = (0,1,0)* gilt:

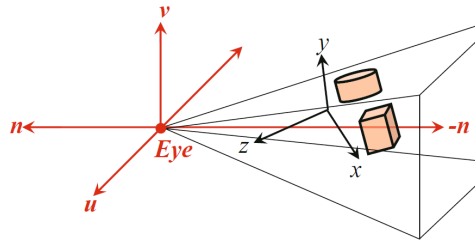
$$\begin{aligned} \mathbf{u}' &= (n'_z, 0, -n'_x) \\ \mathbf{v}' &= (-n'_x \cdot n'_y, n'^2_x + n'^2_y, -n'_y \cdot n'_z) \end{aligned}$$



Computergrafik und Bildverarbeitung
Nischwitz 2011 S.137

Viewing Transformation – Verschiebung zum Ursprung

- Translation zum Ursprung mit -eye *nicht möglich*, da das Koordinatensystem gedreht wurde
- *Lösung*: Gleichung mit Eye-Koordinaten aufstellen und Ergebnis gleich 0 setzen



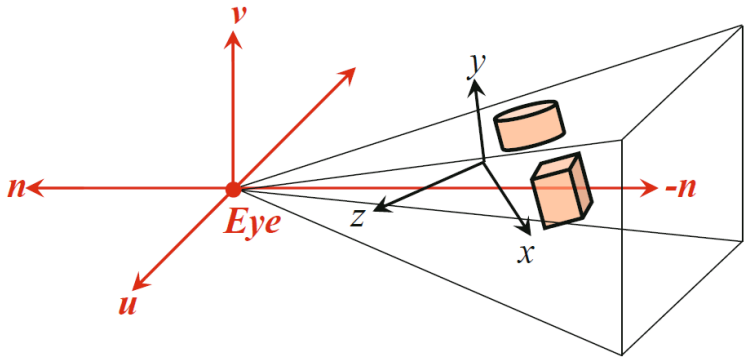
Computergrafik und Bildverarbeitung
Nischwitz 2011 S.137

$$\mathbf{M}_{\mathrm{RT}} \cdot \begin{pmatrix} Eye_x \\ Eye_y \\ Eye_z \\ 1 \end{pmatrix} = \begin{pmatrix} u'^T & t_x \\ v'^T & t_y \\ n'^T & t_z \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Eye_x \\ Eye_y \\ Eye_z \\ 1 \end{pmatrix} \quad \Leftrightarrow \begin{pmatrix} \mathbf{u}' \cdot \mathbf{Eye} + t_x = 0 \\ \mathbf{v}' \cdot \mathbf{Eye} + t_y = 0 \\ \mathbf{n}' \cdot \mathbf{Eye} + t_z = 0 \end{pmatrix}$$

$$= \begin{pmatrix} u'_x & u'_y & u'_z & t_x \\ v'_x & v'_y & v'_z & t_y \\ n'_x & n'_y & n'_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Eye_x \\ Eye_y \\ Eye_z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \Leftrightarrow \begin{pmatrix} t_x = -\mathbf{u}' \cdot \mathbf{Eye} \\ t_y = -\mathbf{v}' \cdot \mathbf{Eye} \\ t_z = -\mathbf{n}' \cdot \mathbf{Eye} \end{pmatrix}$$

Viewing Transformation – Hinweise

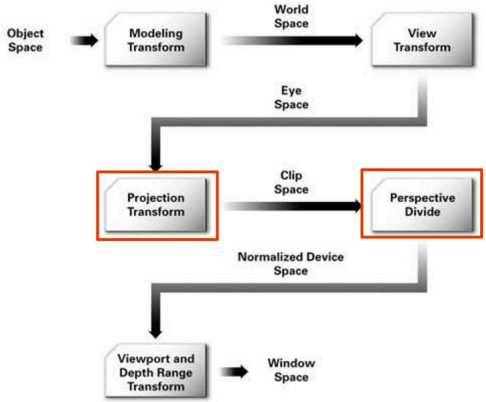
- Eye und Look *dürfen nicht identisch sein*, denn n = Eye – Look verschwindet
- Up-Vektor darf nicht parallel zum Richtungsvektor n sein, denn Up × n = u ergibt 0



Computergrafik und Bildverarbeitung Nischwitz 2011 S.137

Projektionstransformation

Projektionstransformation

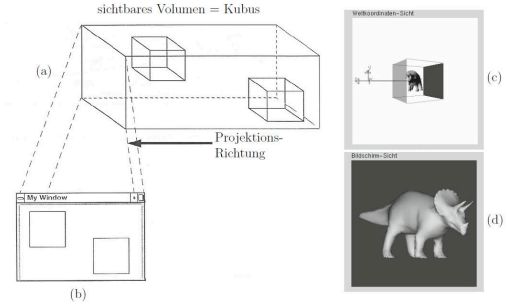


Projektionstransformation

- Model-Transformationen und Viewing Transformation werden zusammengefasst:
  - *ModelView-Matrix*
  - Nach der Modelview-Transformation bereits alle Objekte an gewünschter Position im 3D-Raum
  - Aus Sicht des Augenpunkts / der Kamera
  - Viewing Transformation = Position und Orientierung der Kamera
- *Ziel der Projektionstransformation*: Abbildung der 3D-Szene auf eine 2D-Fläche (x-y-Ebene)
  - Jedoch werden die z-Werte beibehalten (für Verdeckungsberechnungen in späterem Kapitel)
  - Projektionstransformation = Objektiv der Kamera
- Viele Projektionen möglich, in der Praxis zwei relevant
  - Orthographische Projektion
  - Perspektivische Projektion

Projektionstransformation: Orthographische Projektion

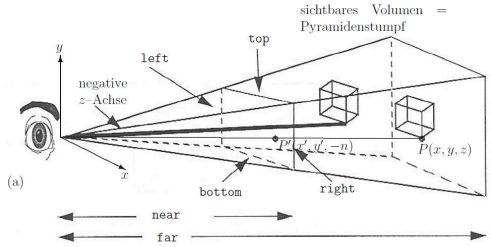
- Bildet durch *parallele Strahlen* die Projektionsfläche ab
- View frustrum ist ein *Kubus*
  - Alle Vertices außerhalb des Kubus werden weggeschnitten (*clipping*)
- Winkel und Größe aller Objekte *bleiben erhalten*
  - Unabhängig von Entfernung / Tiefenwert
- In CAD beliebt
  - Seitenansicht, Vorderansicht und Draufsicht
  - Für technische Zeichnungen optimal, denn *Maße bleiben erhalten*

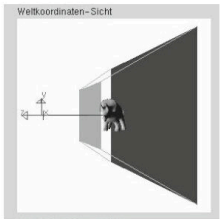


Computergrafik und Bildverarbeitung Nischwitz 2011 S.143

Projektionstransformation: Perspektivische Projektion

- *Konvergierende Strahlen* die im Eye-Punkt zusammenlaufen
- Objekte, die näher am Eye-Punkt sind, erscheinen *größer* als entfernte Objekte

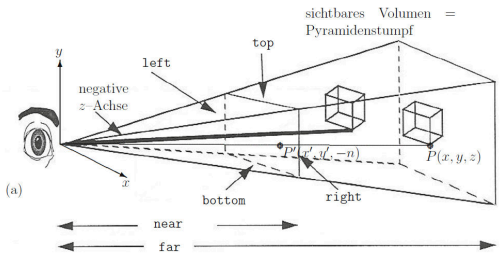




Computergrafik und Bildverarbeitung Nischwitz 2011 S.144

Projektionstransformation: Perspektivische Projektion

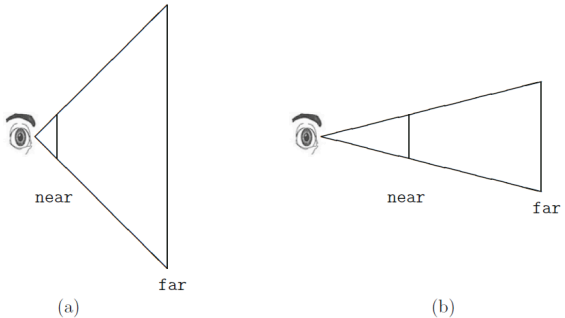
- Zentralperspektive = Approximiert unsere natürliche Wahrnehmung
  - Licht fällt durch Linse auf Netzhaut der Augen
- Häufigste *Anwendung in der 3D-Computergrafik*
- View frustrum ist ein *Pyramidenstumpf*
  - Alle Vertices außerhalb des Pyramidenstumpfs werden weggeschnitten (*clipping*)



Computergrafik und Bildverarbeitung Nischwitz 2011 S.144

Projektionstransformation: Perspektivische Projektion

- Besitzt sechs Begrenzungsebenen (*clipping-planes*)
- Der Boden des Pyramidenstumpfs ist die *far clipping plane*
- In der Spitze am Auge liegt die *near clipping plane*



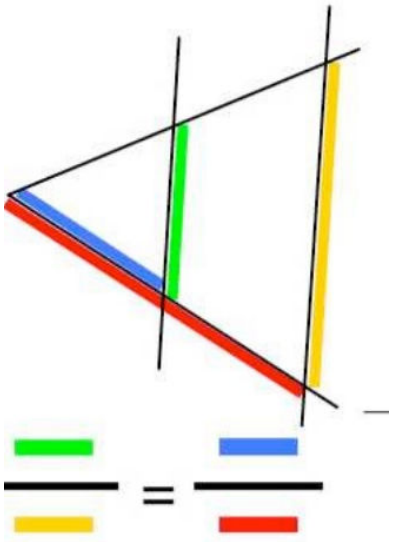
Computergrafik und Bildverarbeitung Nischwitz 2011 S.146

Projektionstransformation: Perspektivische Projektion

- Herleitung durch den Strahlensatz
  - Near clipping plane ist vom Augenpunkt entlang der negativen z-Achse um *n=near* verschoben (analog *f = far*)
  - Beide Ebenen liegen *parallel zur x-y-Ebene*
- Der Punkt $P = (x, y, z)$ wird in den Punkt $P' = (x', y', -n)$ wie folgt abgebildet:

$$\frac{x'}{-n} = \frac{x}{z} \quad \Leftrightarrow \quad x' = -\frac{n}{z} \cdot x$$

$$\frac{y'}{-n} = \frac{y}{z} \quad \Leftrightarrow \quad y' = -\frac{n}{z} \cdot y$$



Perspektivische Projektion – Abbildung der Punkte

- Die Transformationsmatrix P *erfüllt die Strahlensatzgleichungen*

$$\mathbf{v}' = \mathbf{P}\mathbf{v} \quad \Leftrightarrow \quad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1+\frac{f}{n} & f \\ 0 & 0 & -\frac{1}{n} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ \left(1+\frac{f}{n}\right) z + fw \\ -\frac{z}{n} \end{pmatrix}$$

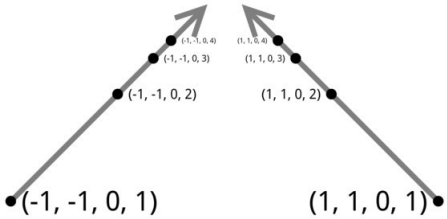
- Grund dafür ist die *nachliegende Division mit der w Komponente*
  (Umrechnung der homogenen in euklidische Koordinaten)

$$\begin{pmatrix} x'_E \\ y'_E \\ z'_E \end{pmatrix} = \begin{pmatrix} -\frac{n}{z} \cdot x \\ -\frac{n}{z} \cdot y \\ -\frac{n}{z} \cdot fw - (f+n) \end{pmatrix}$$

Perspektivische Projektion – w und die Perspektive

- Die Projektionsmatrix selbst lässt keine Objekte kleiner erscheinen die weiter entfernt sind
- Erst die spätere *perspective division* lässt Objekte kleiner erscheinen die weiter entfernt sind
- Grund: w ist abhängig von z (x, y nähern sich (0,0) an wenn z groß ist)
- Visuell gesehen kann man sich w als die Entfernung der Kamera zum Objekt vorstellen

$$\begin{pmatrix} x'_E \\ y'_E \\ z'_E \end{pmatrix} = \begin{pmatrix} -\frac{n}{z} \cdot x \\ -\frac{n}{z} \cdot y \\ -\frac{n}{z} \cdot fw - (f+n) \end{pmatrix}$$



http://www.learnopengles.com/understanding-opengls-matrices/dividebyw/

Normierung auf Normalized Device Coordinates (NDC)

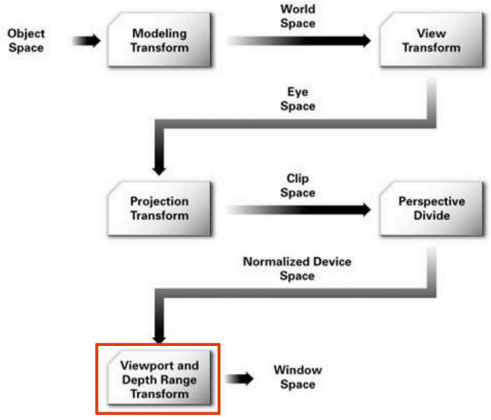
- Alle Vertices im *view frustrum* befinden sich nun auf der *near clipping plane*
- *Beliebig große* near clipping planes müssen auf eine festgelegte Anzahl von Pixeln gebracht werden
- Deshalb müssen sie *normiert* werden, d. h. die euklidischen Koordinaten müssen *zwischen -1 und 1* liegen (Normalized Device Coordinates)
- In OpenGL wird die Normierungsmatrix mit der Projektionsmatrix zusammengefasst, um *in einem Schritt* den Wertebereich [-1,1] zu erreichen

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

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

Viewport Transformation

Viewport Transformation



Viewport Transformation

- X-Y Ebene *durch Pixel definiert*
- Umrechnung der NDC in eine gewählte Viewportgröße

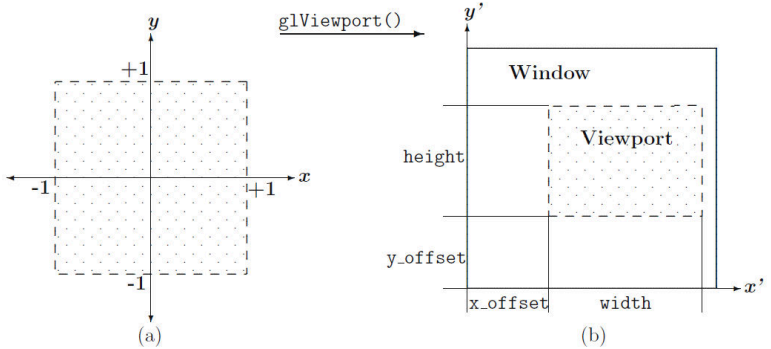


Bild 7.14: Abbildung des sichtbaren Volumens in einen *Viewport*: (a) Wertebereich der normierten Koordinaten des sichtbaren Volumens. (b) Wertebereich der Bildschirm–Koordinaten des Viewports nach der Viewport-Transformation. Der Ursprung der Bildschirm–Koordinaten liegt in der linken unteren Ecke des Windows.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.146

Viewport Transformation

- Abbildung auf das Window ist wie folgt definiert:
  - X auf Wertebereich [-width/2, width/2] setzen
  - Y auf Wertebereich [-height/2, height/2] setzen
  - Viewport um halbe Windowbreite/höhe verschieben
  - Mit *offset* zusätzlich verschiebbar

$$x' = \frac{width}{2} \cdot x + \left(x\_offset + \frac{width}{2} \right)$$

$$y' = \frac{height}{2} \cdot y + \left(y\_offset + \frac{height}{2} \right)$$

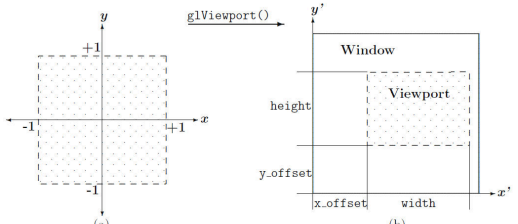


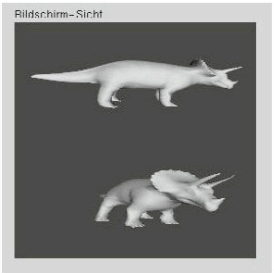
Bild 7.14: Abbildung des sichtbaren Volumens in einen *Viewport*: (a) Wertebereich der normierten Koordinaten des sichtbaren Volumens. (b) Wertebereich der Bildschirm–Koordinaten des Viewports nach der Viewport-Transformation. Der Ursprung der Bildschirm–Koordinaten liegt in der linken unteren Ecke des Windows.

Computergrafik und Bildverarbeitung Nischwitz 2011 S.146

Viewport Transformation

- In OpenGL: `glViewport(xoffset,yoffset,width,height)`

`glViewport(0,0,256,256)`
(a)



`glViewport(0,128,256,128)`
`glViewport(0,0,256,128)`
(b)

Computergrafik und Bildverarbeitung Nischwitz 2011 S.151