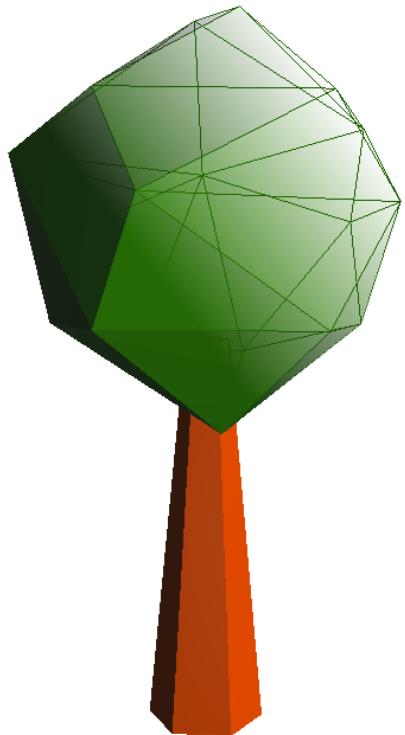


Einführung

Einführung in die Computergrafik (für Augmented Reality)

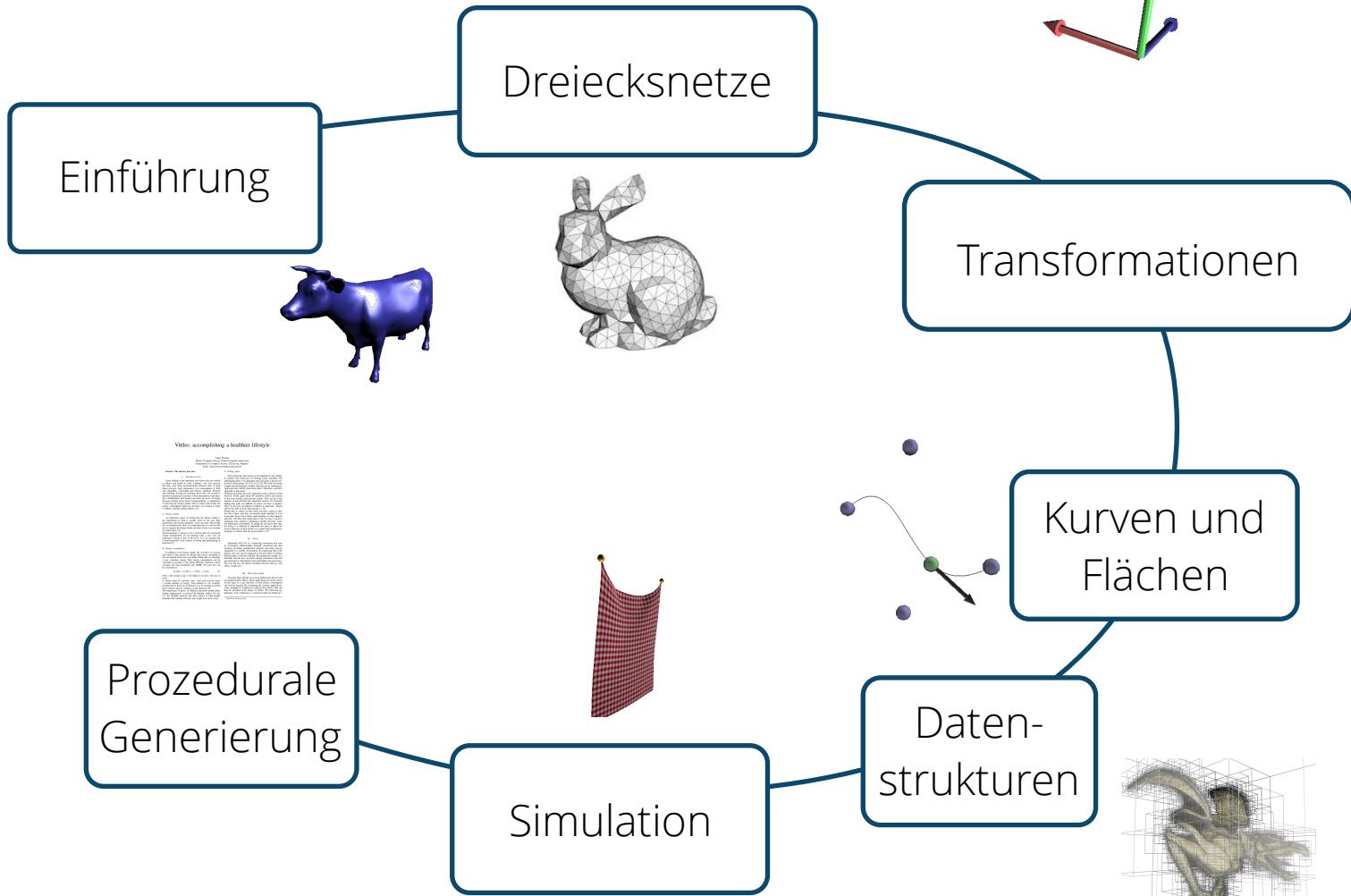
Agenda

- Organisation
- Grundlagen: Vektoren
- 3D-Raum
- Dreiecksnetze
- OpenGL ES
- Rendering in OpenGL
- Szenengraph
- Augmented Reality



Organisation

Inhalt



Erforderliche Vorkenntnisse

- Software
 - JDK 8+
 - Git
 - IntelliJ und/oder Android Studio
- Interesse an mathematischen Grundlagen
- Motivation
- präzises und strukturiertes Arbeiten
- Bereitschaft, Fachliteratur zu lesen

Literatur

- Virag, Gerhard: Grundlagen der 3D-Programmierung : Mathematik und Praxis mit OpenGL. München : Open Source Press, 2012
- Klawonn, Frank: Grundkurs Computergrafik mit Java : Die Grundlagen verstehen und einfach umsetzen mit Java3D. 3. Auflage, Wiesbaden : Vieweg und Teubner, 2010
- Akenine-Moeller, Tomas; Haines, Eric: Real-Time Rendering. 3. Auflage, Wellesley : A.K. Peters, 2008
- Hughes, John F.; van Dam, Andries; McGuire, Morgan; Sklar, David F.; Foley, James D.; Feiner, Steven K.; Akeley, Kurt: Computer Graphics: Principles and Practice, 2. Auflage, Amsterdam, Addison-Wesley Longman, 1996 (3. Auflage erscheint 2013)

Online-Quellen zu OpenGL

- OpenGL Programming Guide (Red Book):
 - <http://www.openglprogramming.com/red/>
- OpenGL API:
 - <http://docs.gl/>
 - <https://www.khronos.org/opengles/>

Material

- Folien
- Skript
- Praktikumsaufgaben
- Merkblatt zur Klausur
- Merkblatt zum Abschlussprojekt
- Latex-Vorlage zum Abschlussprojekt

EMIL

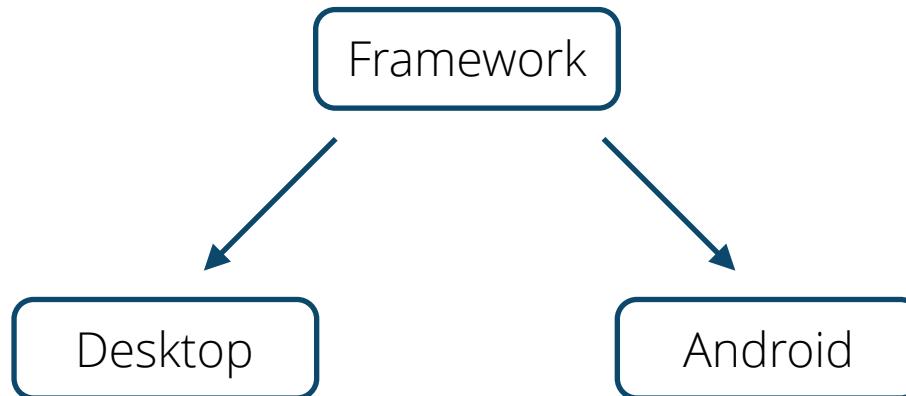
- <https://www.elearning.haw-hamburg.de/course/view.php?id=30963>
- Schlüssel: WPCG1819

Praktikum

- 7 Aufgabenblätter
- bearbeitet in 2er-Teams
- Abgabe
 - Vorstellung im Praktikum
 - Theorie + Quellcode
 - Aufgabe muss vollständig bearbeitet sein
 - Details können im Praktikum nachgearbeitet werden
 - jedes Team muss eigenständige Lösung haben

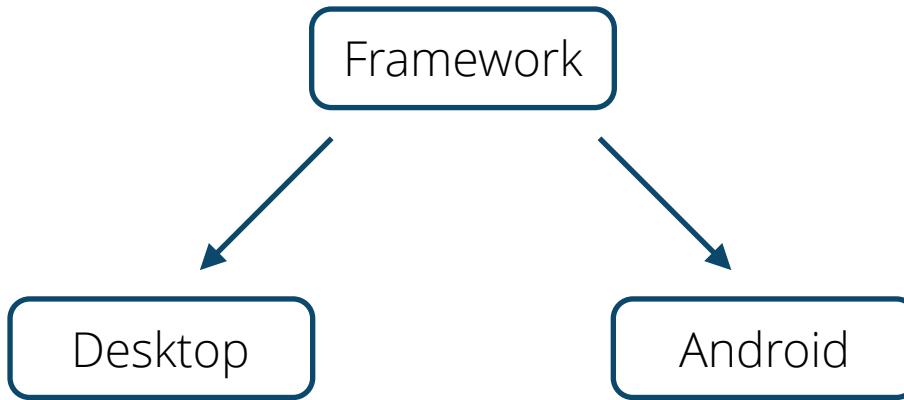
Praktikum

- PVL-Bedingungen
 - Teilnahme an alle Praktikumsterminen
 - erfolgreiche Bearbeitung aller Aufgaben



Loslegen

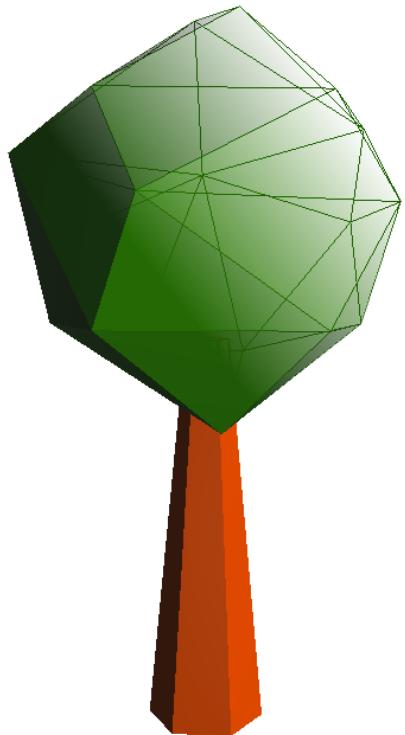
- E-Mail mit Login (a...-number) → *philipp.jenke@haw-hamburg.de*
- Bestätigung erhalten
- Clone *https://gitlab.informatik.haw-hamburg.de/wp_cg/wpcgar.git*



- Installationshinweise in */doc/wp_cg_ar.pdf*

Prüfungsleistung

- Auswahl zwischen zwei Prüfungen
- entweder: Klausur
 - wie üblich in den Prüfungswochen
- oder: Abschlussprojekt
 - Softwareentwicklung, Präsentation, Kurzausarbeitung
 - siehe Merkblatt
 - Ende der Semesterferien



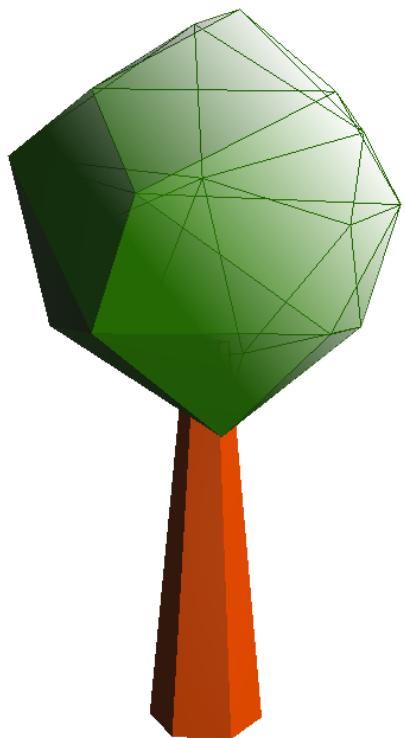
Grundlagen: Vektoren

Grundlagen: Vektoren

- Einführung
- Skalierung
- Skalarprodukt
- Orthogonalität
- Kreuzprodukt

Übung: Vektoren

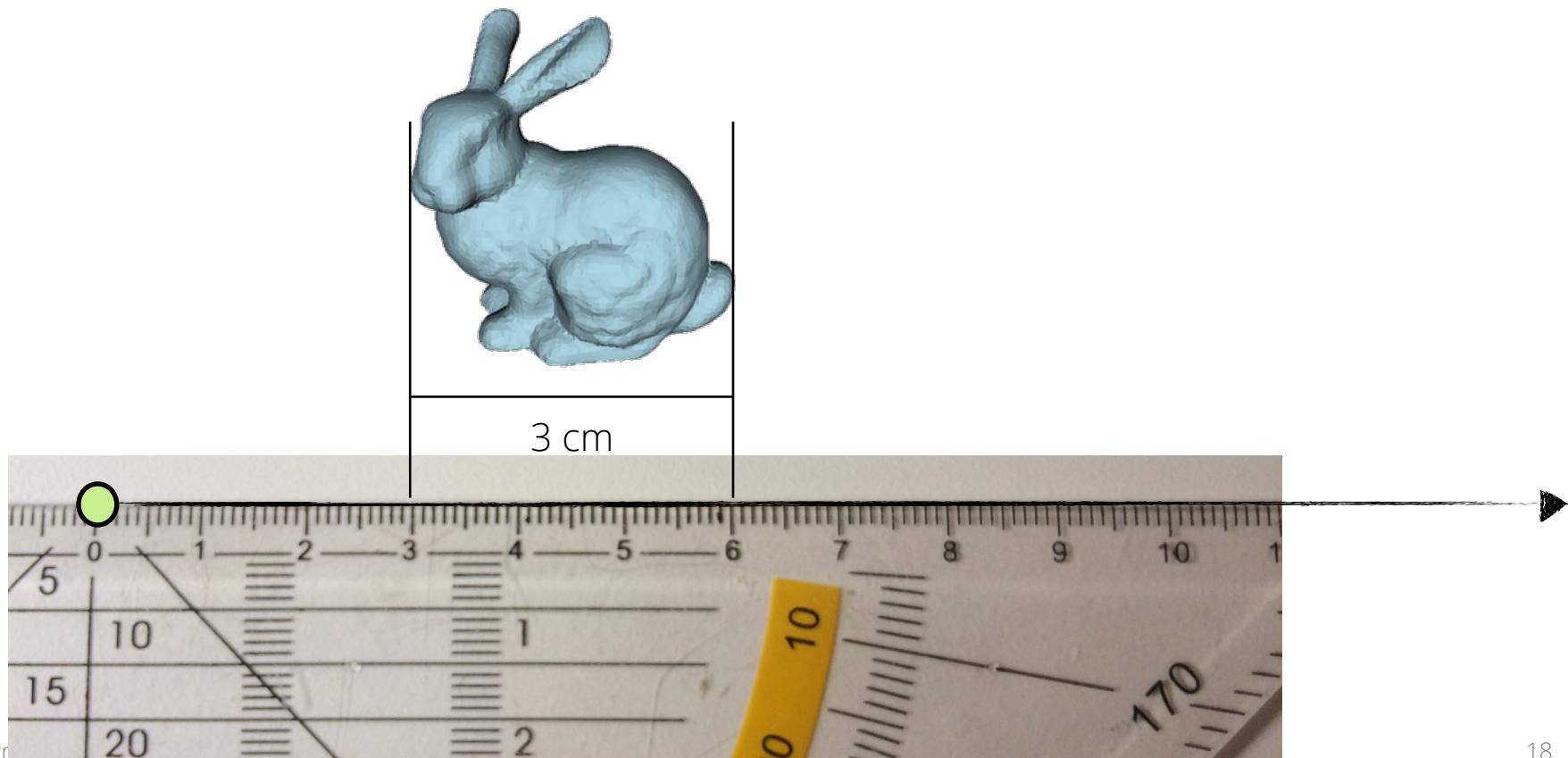
- Wir beschreiben die Geschwindigkeit eines Objektes in 2D als eine Funktion über die Zeit: $v(t) = (1 - 0.5 * t, 0.25 * t)$.
- Die initiale Position des Objektes ist (0,0).
- Berechnen Sie die Position des Objektes zu den Zeitpunkten
 - $t = 1\text{s}$, $t = 2\text{s}$, $t = 3\text{s}$
- Die Bewegung wird nur einmal pro Sekunde ausgewertet



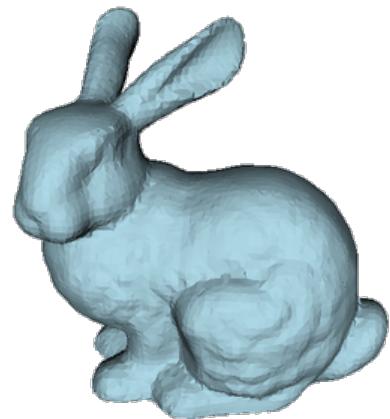
3D-Raum

Entfernungen

- Entfernungsmessung: Skalierung notwendig

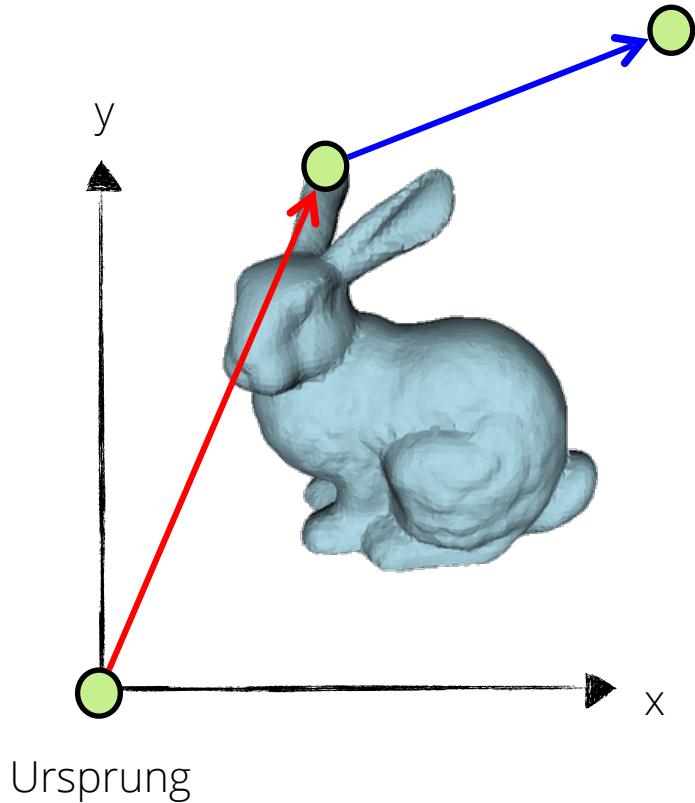


Achsen



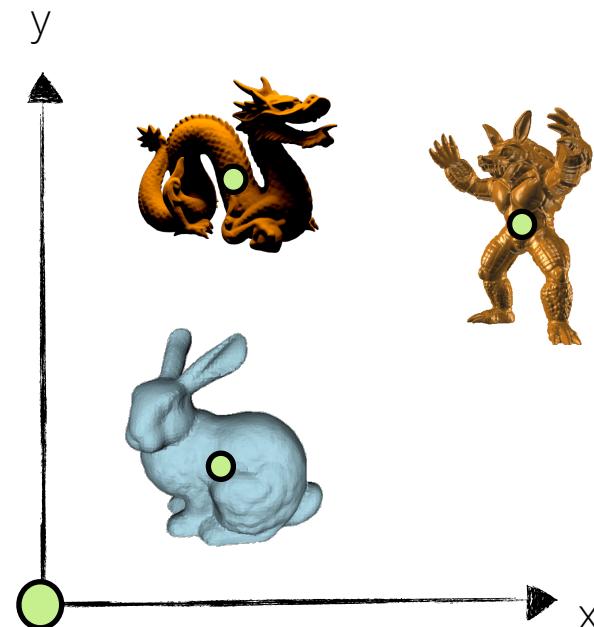
Koordinatensystem

- Ursprung + Achsen = Koordinatensystem
- hier: Karthesisches Koordinatensystem
 - Achsen paarweise senkrecht
 - Position im Raum: Vektor
 - Ortsvektor
 - Verschiebung im Raum: Vektor
 - Richtungsvektor
- Beispiel:
 - 2D Koordinatensystem
 - x- und y-Achse

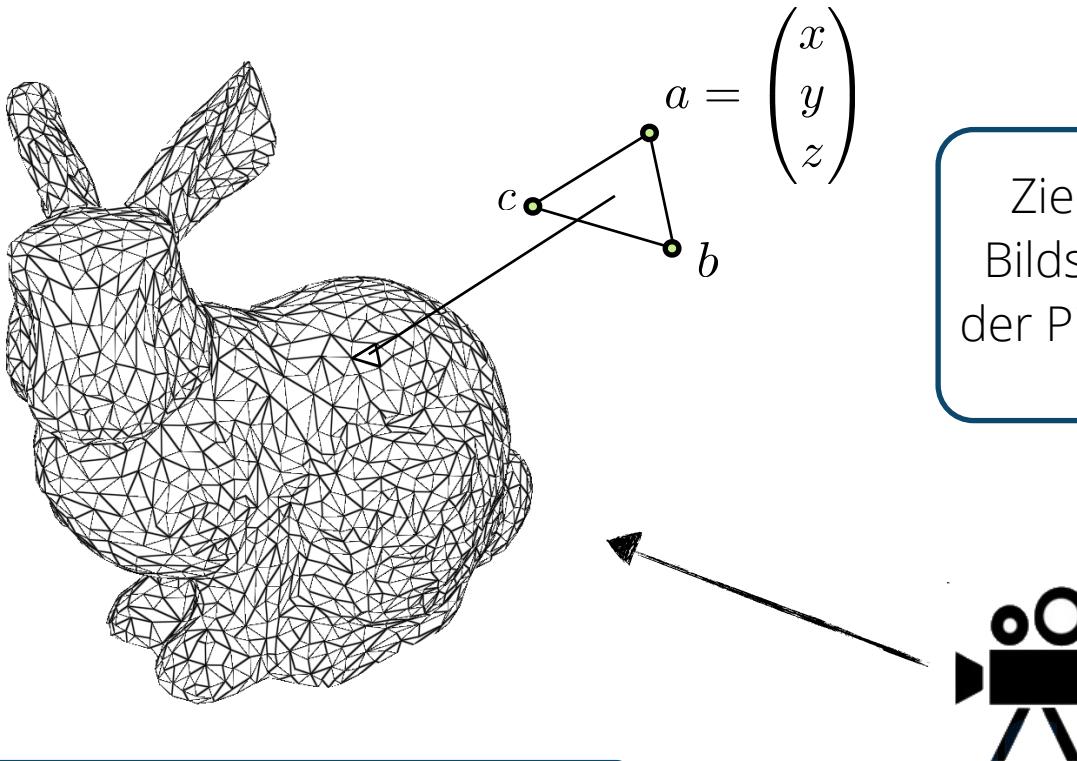


Szenen

- Wir betrachten Szenen mit einem gemeinsamen Koordinatensystem
 - üblicherweise 3D: x-, y- and z-Achse
 - „Weltkoordinatensystem“
 - Positionen der Objekte im Raum



Unsere Situation

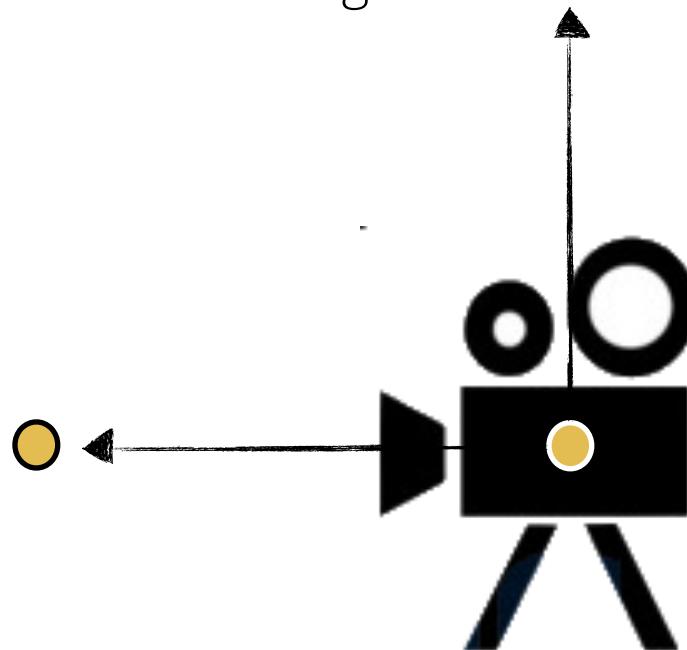


3D-Objekte
(hier: Dreiecke mit 3D Vertices)

virtuelle Kamera

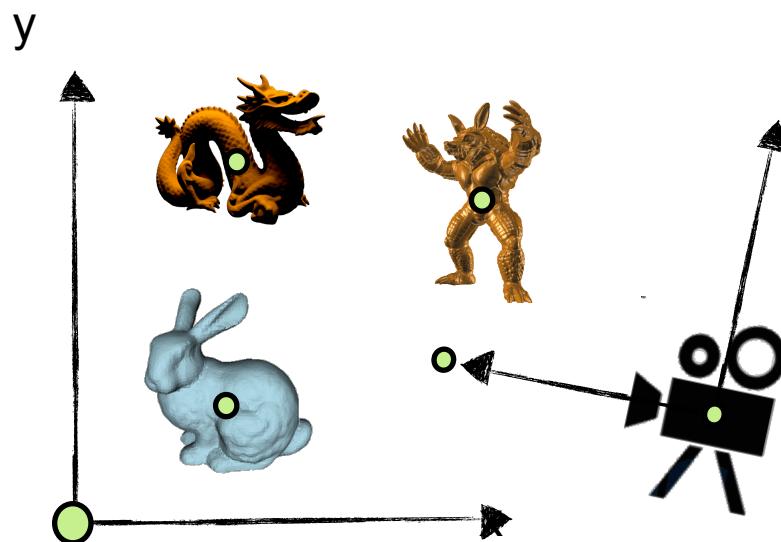
(Virtuelle) Kamera

- Kamera hat eigenes Koordinatensystem
 - Position
 - Blickrichtung
 - Oben-Vektor
 - ohne diese ist die Orientierung nicht eindeutig
- Alternative Repräsentation
 - Position
 - Referenzpunkt
 - Oben-Vektor



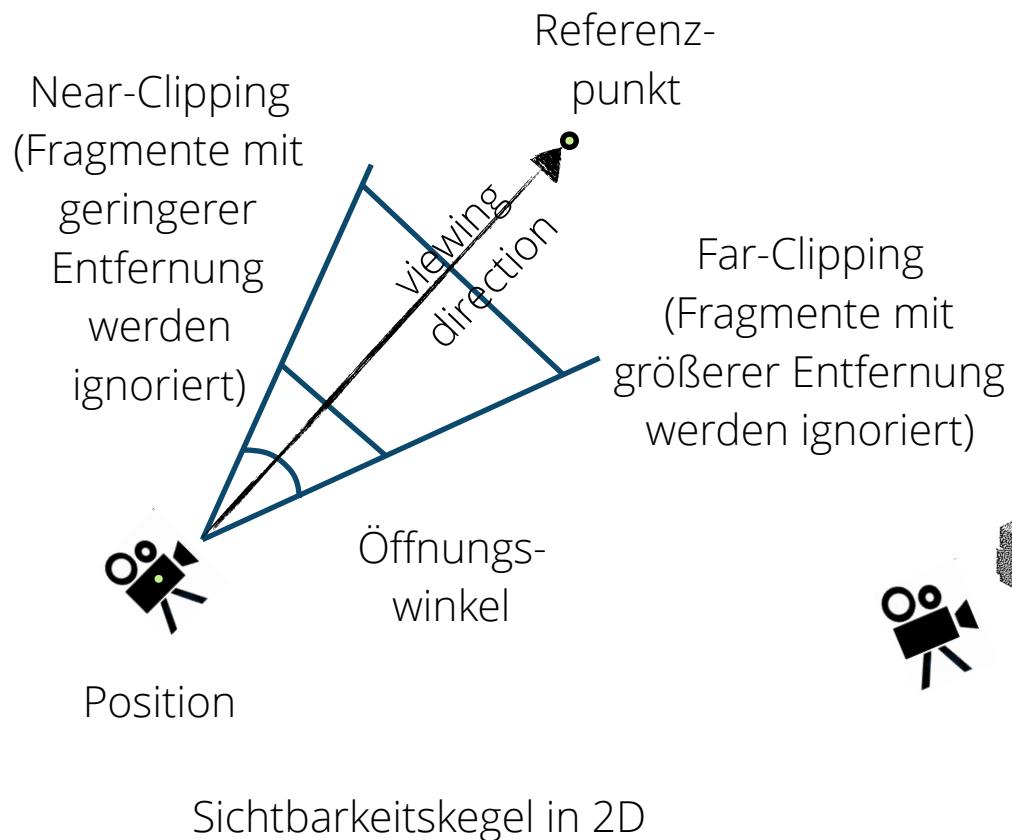
Koordinatensysteme

- Kamera bewegt sich durch das Weltkoordinatensystem
- Transformation zwischen Koordinatensystemen notwendig
 - siehe Model-View-Transformation (OpenGL pipeline)
 - siehe Skript

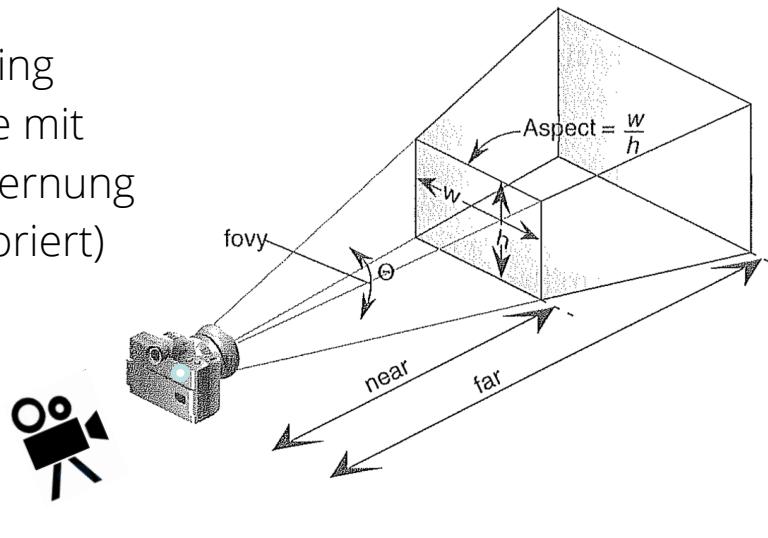


Kamera

- Sichtbarkeitsvolumen



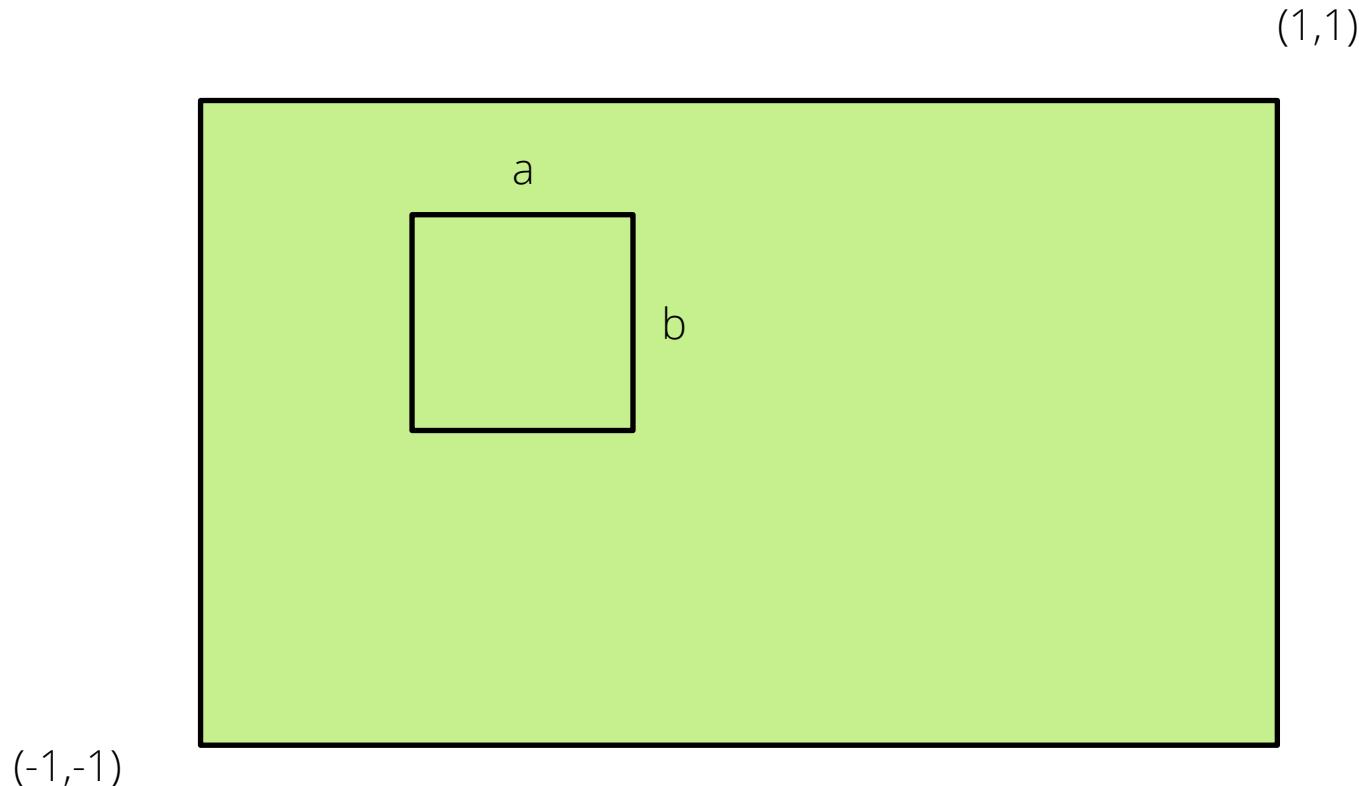
Öffnungswinkel in 3D:
Verhältnis zwischen Breite und Höhe (aspect) und Öffnungswinkel (fovy)

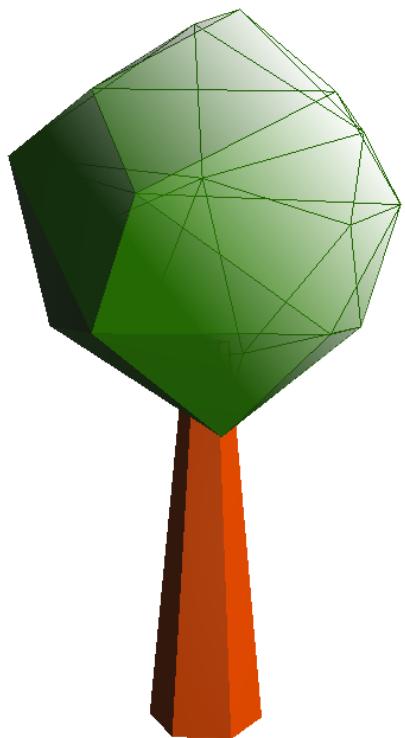


Sichtbarkeitsvolumen 3D

Übung: Quadrat in Bildkoordinaten

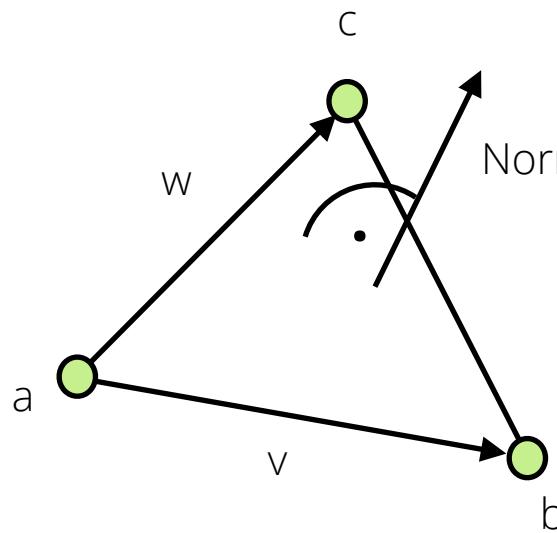
- Sie wollen ein Quadrat mit der Ausdehnung $w \times h$ zeichnen. Das Quadrat hat in Bildkoordinaten die Breite a . Wie groß ist b ?





Dreiecksnetze

Dreieck



$$\text{Normale } n = \frac{v \times w}{\|v \times w\|} = \frac{(b - a) \times (c - a)}{\|(b - a) \times (c - a)\|}$$

Kreuzprodukt

$$\vec{v} \times \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{pmatrix}$$

Dreiecksnetz - Datenstruktur

Triangles		
$x_{11} \ y_{11} \ z_{11}$	$x_{12} \ y_{12} \ z_{12}$	$x_{13} \ y_{13} \ z_{13}$
$x_{21} \ y_{21} \ z_{21}$	$x_{22} \ y_{22} \ z_{22}$	$x_{23} \ y_{23} \ z_{23}$
...
...
...
$x_{F1} \ y_{F1} \ z_{F1}$	$x_{F2} \ y_{F2} \ z_{F2}$	$x_{F3} \ y_{F3} \ z_{F3}$

Facettenliste

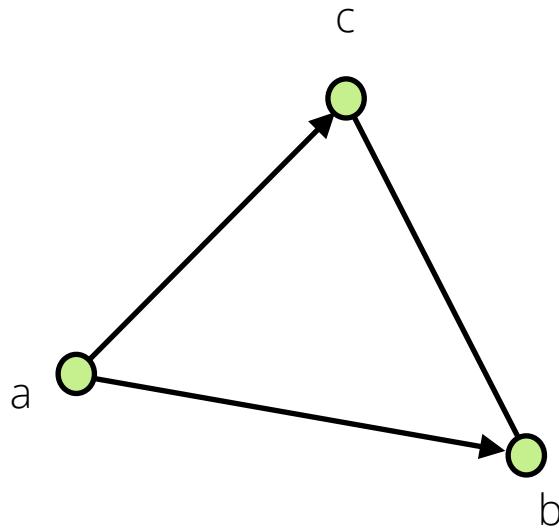
Vertices	Triangles
$x_1 \ y_1 \ z_1$	$i_{11} \ i_{12} \ i_{13}$
...	...
$x_v \ y_v \ z_v$...
...	...
...	...
...	...
$i_{F1} \ i_{F2} \ i_{F3}$	

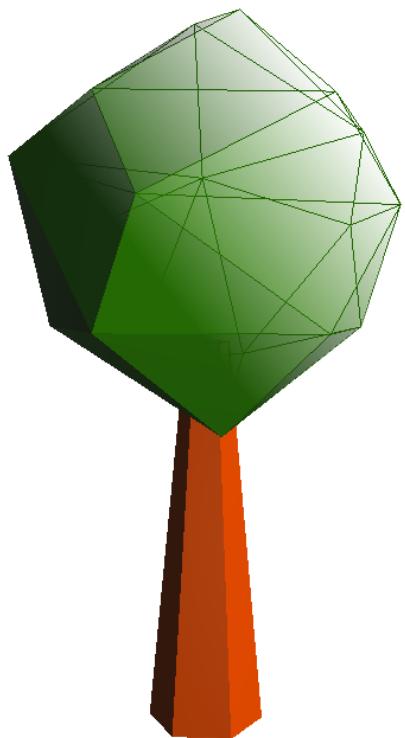
Indizierte Facettenliste (z.B. im Wavefront-Obj Datenformat verwendet)

Beispiel: Normale

- Berechnen Sie die Dreiecksnormale für das Dreieck (abc)

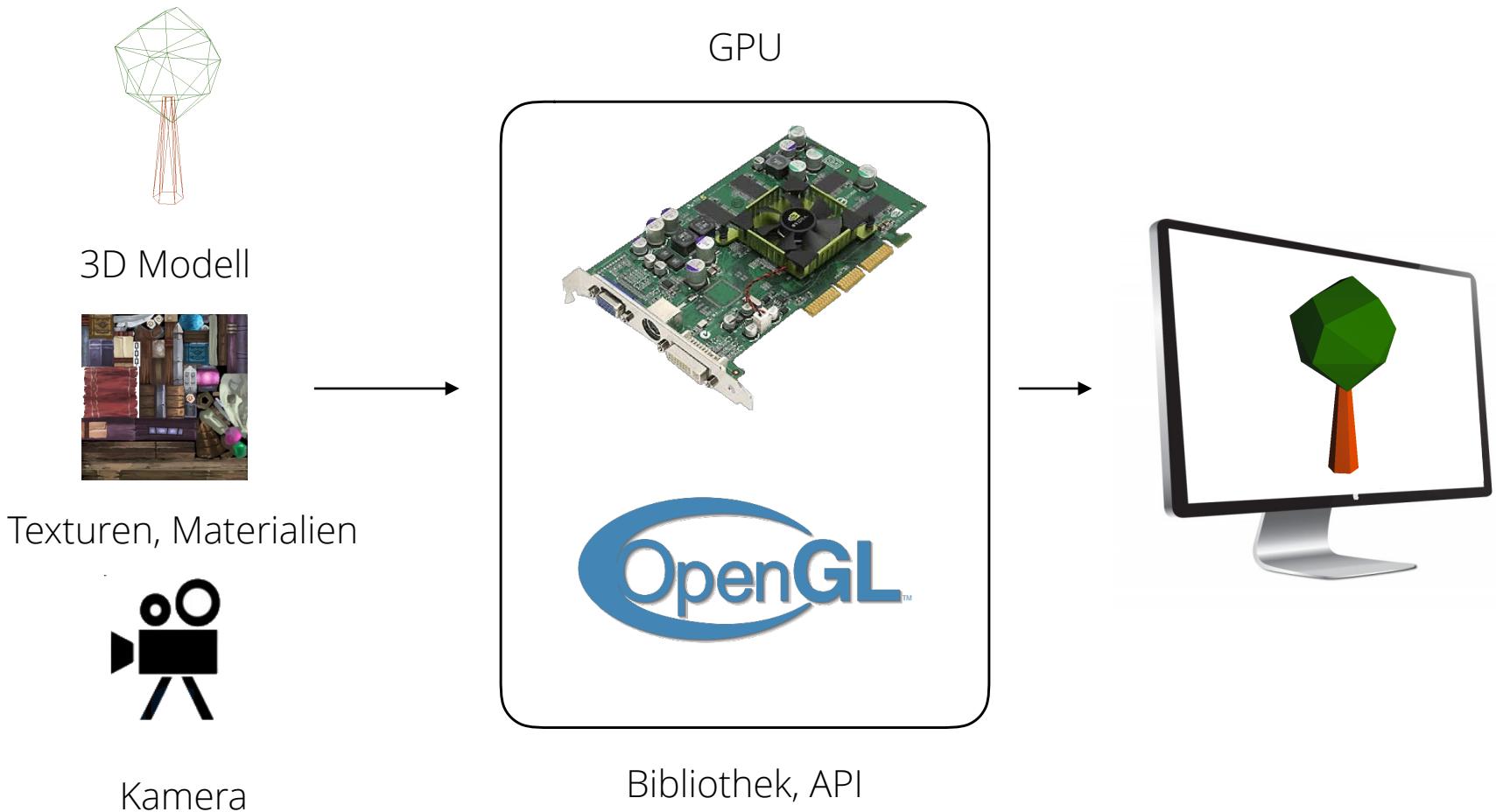
$$\vec{v} \times \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{pmatrix}$$





OpenGL (ES)

Rendering Pipeline



Rendering Pipeline

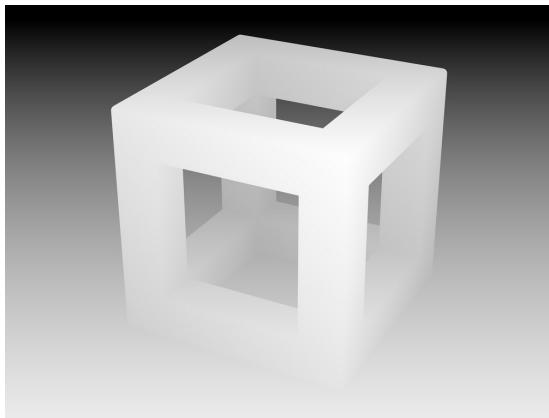
- **Modell-Transformation:** Transformation der Szenen-Primitive (siehe Transformationsknoten im Szenengraphen)
- **View-Transformation:** Ausrichtung der Szene = Kamera blickt entlang der (negativen) z-Achse
- **Projection:** Abbildung aus dem 3D-Raum auf die Bildebene
- **Clipping:** Nicht sichtbare Teile der Szenen ignorieren (Außerhalb des Bildrahmens)
- **Screen Mapping:** Transformation des Bildes auf den Bildschirm
- **Rasterisierung:** Primitive → Pixel
- jeder Schritt = Matrix-Multiplikation

Rendering Pipeline

- ursprünglich: fixed-function Pipeline
 - vordefinierte Berechnungen, nur Parameter-Einstellungen möglich
- heute: programmierbare Pipeline
 - eigene Programme steuern die Kontrollfluss
 - Shader (C-ähnliche Programmiersprache)

Framebuffer

- Ergebnis nach der Rendering Pipeline: Matrix der Farbwerte (und weiterer Werte)
 - abgelegt im Farbpuffer (Auflösung identisch zur Bildauflösung)
 - Farben im RGBA-Format
- Framebuffer beinhaltet weitere Puffer
 - Tiefen-Puffer, z-Puffer: nahester Tiefenwert pro Pixel
 - Stencil-Puffer: programmierbarer Zäher pro Pixel



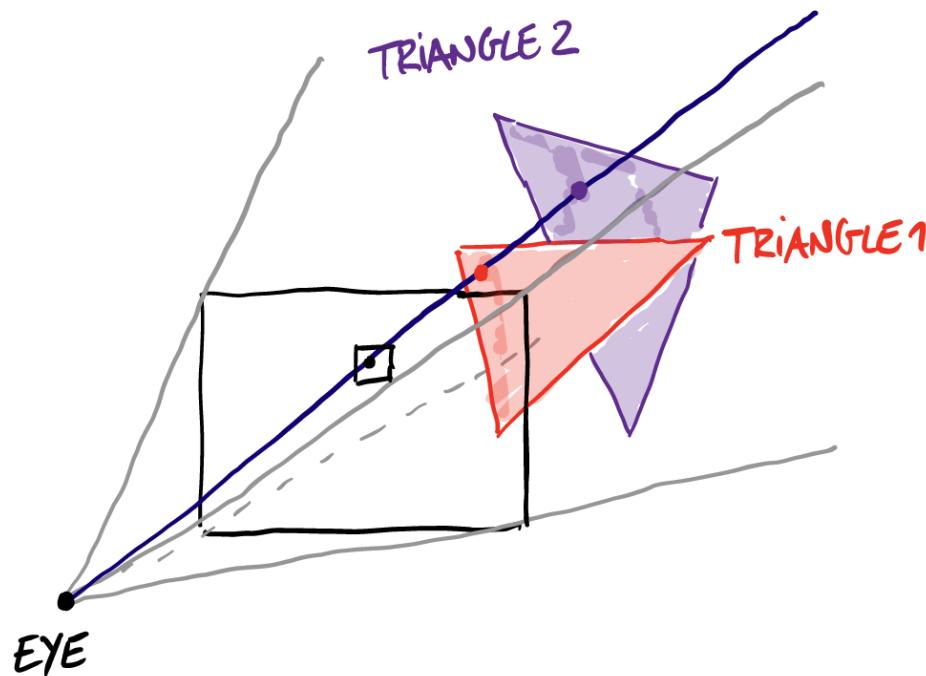
Tiefen-Puffer [1]

00000	000
00000	000
00000	000
00000	000
00000	000
0000000000000000	000
0000000000000000	000

Stencil-Puffer (Schema)

Sichtbarkeit: Depth Buffer

- Problem: Pixel von unterschiedlichen Dreiecken werden auf gleichen Pixel im Bild projiziert

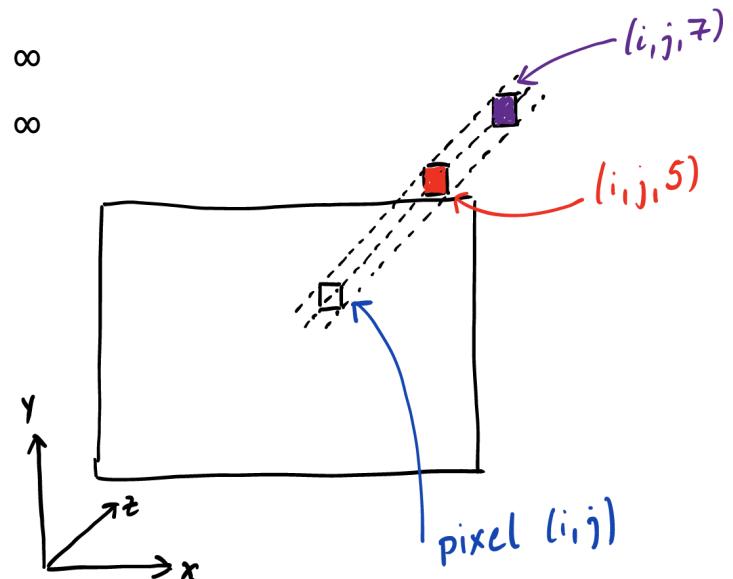


Sichtbarkeit: Tiefen-Puffer

- Idee: wähle Pixel mit geringster Tiefe

- Algorithmus:

```
init all depth buffer d pixels with ∞  
init all color buffer c pixels with ∞  
for each triangle  
    for each triangle pixel p  
        if (f[p.x][p.y] > p.z )  
            d[p.x][p.y] = p.z  
            c[p.x][p.y] = p.color  
        endif  
    endfor  
endfor
```



Quellen

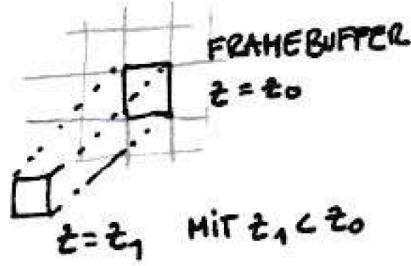
- Edwin Catmull: A Subdivision Algorithm for Computer Display of Curved Surfaces. Dissertation, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City 1974
- Wolfgang Straßer: Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten. Dissertation, TU Berlin 1974

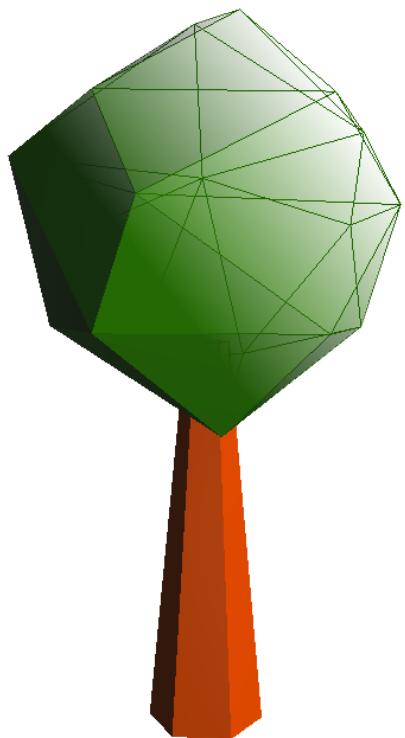
Übung: Tiefen-Puffer

- Welche Farbe hat der Farbpuffer am Pixel (23, 42) im Ergebnisbild, wenn die folgenden Pixel gezeichnet werden?
 - (x: 23, y: 42, z: 23, color: rot)
 - (x: 23, y: 42, z: 42, color: grün)
 - (x: 42, y: 23, z: 12, color: blau)
 - (x: 23, y: 42, z: -12, color: orange)

Transparenz

- a-Wert für Vertices, Format: RGBA
 - a = 0: transparent
 - a = 1: opaque
- glEnable(GL_LIGHTING)
- glBlendFunc(source, destination)
 - source: neue Farbe
 - destination: aktuelle Farbe im Farb-Puffer
- z.B.: glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

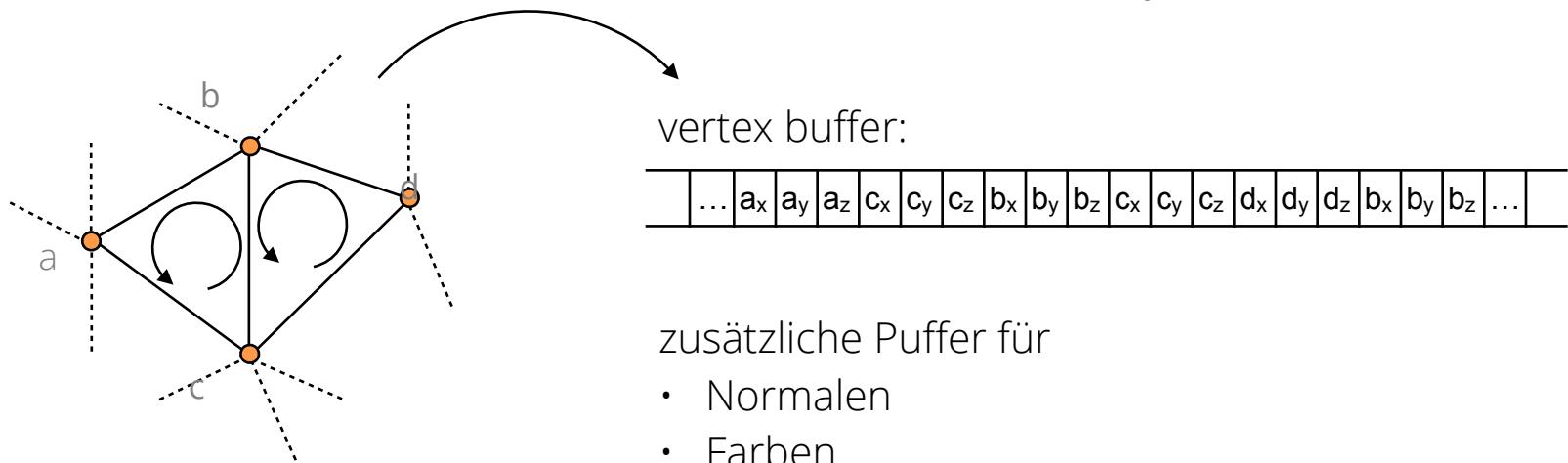




Rendering in OpenGL

Primitive Zeichnen

- OpenGL kann mit unterschiedlichen Primitiven umgehen (z.B. Dreiecke, Quads ...)
- Information wird gebündelt an die GPU übertragen (in einem Vertex-Buffer-Object)
- Abstraction im Framework: class VertexBufferObject



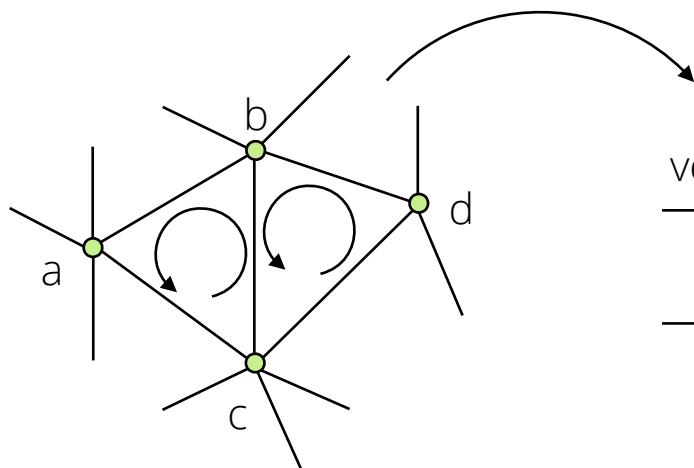
Verwendung des VertexBufferObjects

- Methode

```
public void Setup(List<RenderVertex> renderVertices, int primitiveType)
```

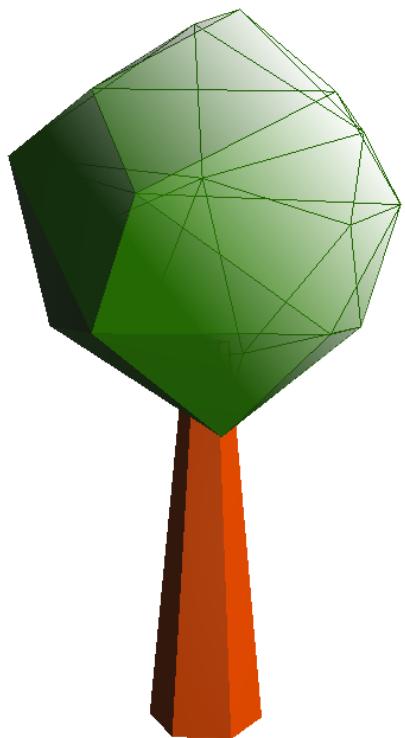
- Argumente:

- Liste von Vertices
 - für Dreiecke: 3 Vertices
- und Primitiv-Typ (z.B.: Dreiecke)
 - GL2.GL_TRIANGLES in Java



vertex buffer:

...	a	a	a	c	c	c	b	b	b	c	c	c	d	d	d	b	b	b	b	...
...	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	...



Szenengraph

Szenenbeschreibung

- Datenstruktur zur Szenenbeschreibung
- orientierter, azyklischer Graph von Knoten
- Knoten beinhalten Informationen zu:
 - Geometrie
 - Materialien
 - Gruppen
 - ...
- Szeneneigenschaften können durch Knoteneigenschaften gesetzt werden (angewendet auf die Kindknoten)
- Beispiele:
 - Rendering
 - Selektion
 - Hüllkörperberechnungen
 - Datei-I/O

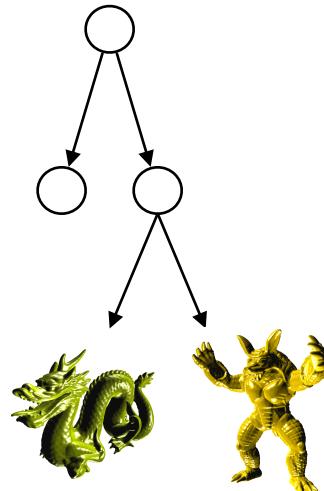
Node

- INode: abstrakte Basisklasse
- InnerNode: hat Kindknoten
- LeafNode: keine Kinder, kann gerendert werden
- Transformationsknoten:
 - ScaleNode: Skalierung
 - TranslationNode: Verschiebung
 - RotationNode: Drehung
- RootNote: Wurzelknoten mit generellen Szeneninformationen
- Geometrie-Knoten:
 - CubeNode: Darstellung eines Würfels
 - SphereNode: Darstellung einer Kugel

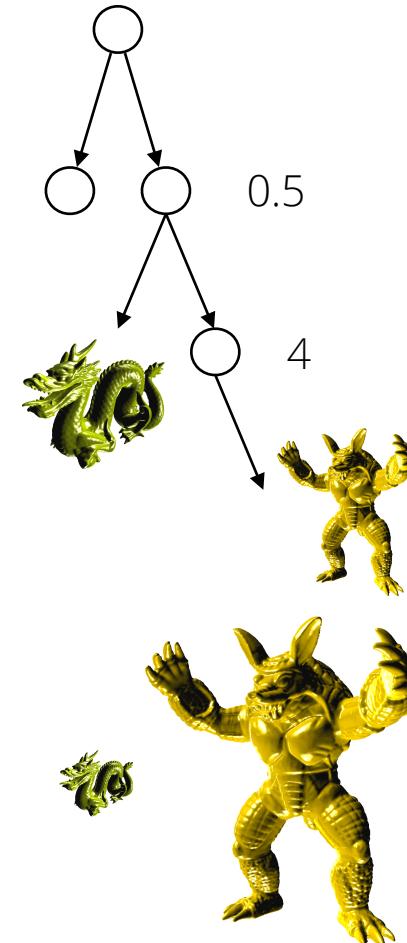
Szenengraph

- Eigenschaften gelten für Knoten und rekursiv für Kinder
- Beispiel: Skalierung

Szenengraph:



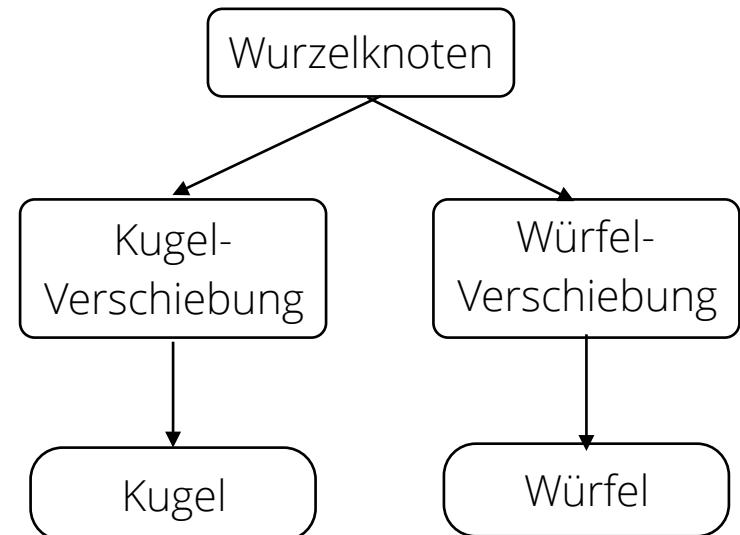
Ergebnis:



Szenengraph im Praktikums-Framework

- Beispiel:

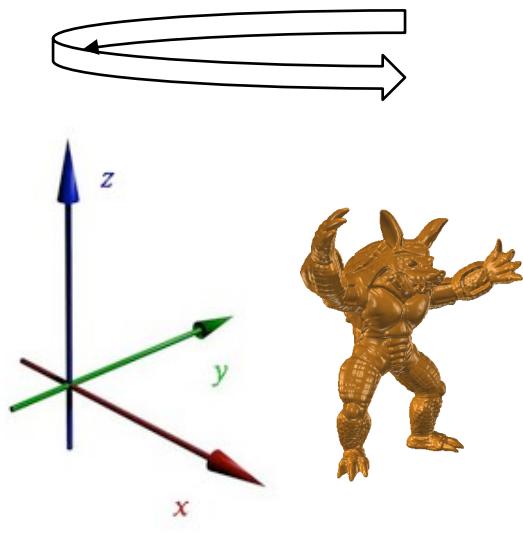
```
TranslationNode sphereTranslation =  
new TranslationNode(new Vector3(-2f, 0, 0));  
INode sphereNode = new SphereNode(0.5f, 20);  
sphereTranslation.AddChild(sphereNode);  
getRoot().AddChild(sphereTranslation);  
  
TranslationNode cubeTranslation =  
new TranslationNode(new Vector3(0, 0, 2f));  
INode cubeNode = new CubeNode(0.5f);  
cubeTranslation.AddChild(cubeNode);  
getRoot().AddChild(cubeTranslation);
```



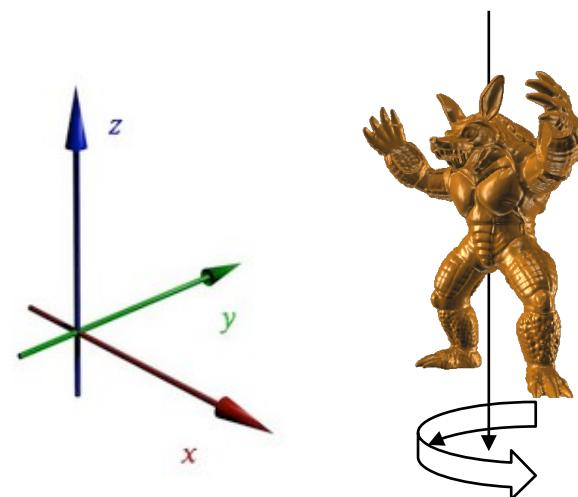
Rotationen

- Rotation um 90° entgegen dem Uhrzeigersinn (counter-clockwise, ccw)

1) Rotation um Hauptachse
(hier: z-Achse)



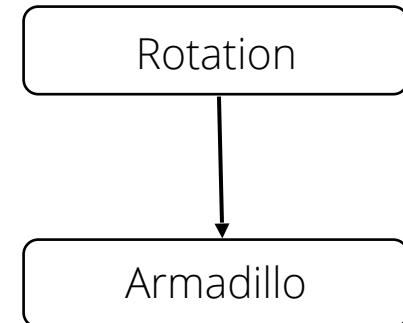
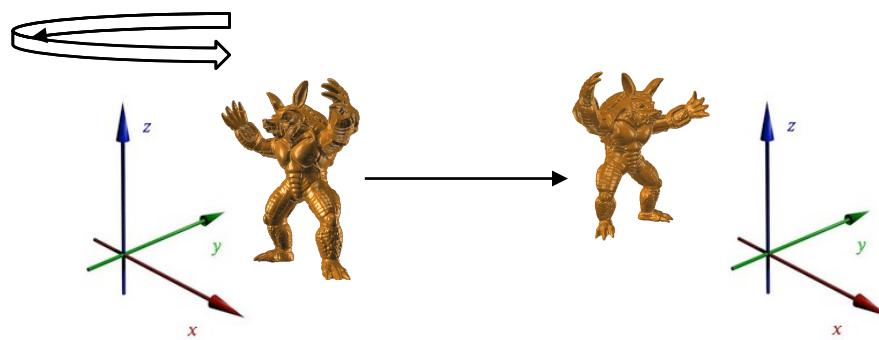
2) Rotation um
eigene Achse



Rotation um Hauptachse

- Rotationsmatrix um z-Achse:

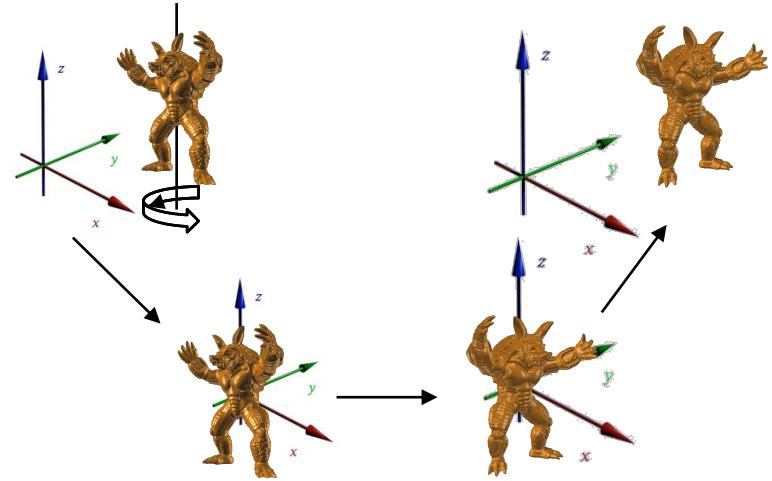
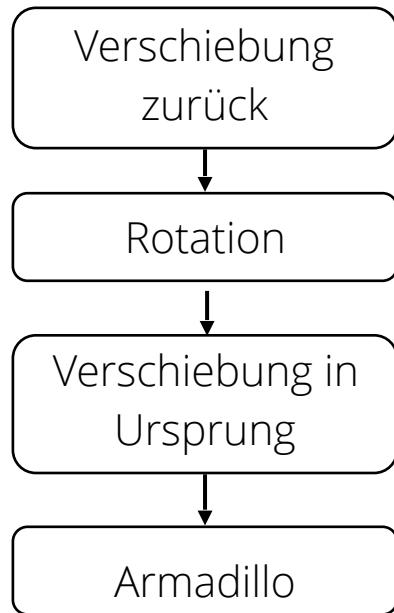
$$T_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Rotation um
Hauptachse

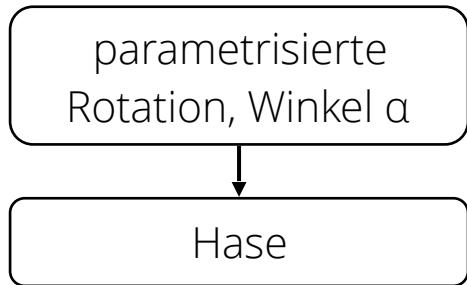
Rotation um eigene Achse

- Idee
 - Verschiebung in Ursprung
 - Rotation (z.B. um z-Achse)
 - Verschiebung zurück
- Szenengraph:



Animationen

- Beispiel: kontinuierliche Rotation
- Szenengraph



- in jedem Zeitschritt
 - Winkel a erhöhen
 - Rotationsmatrix anpassen

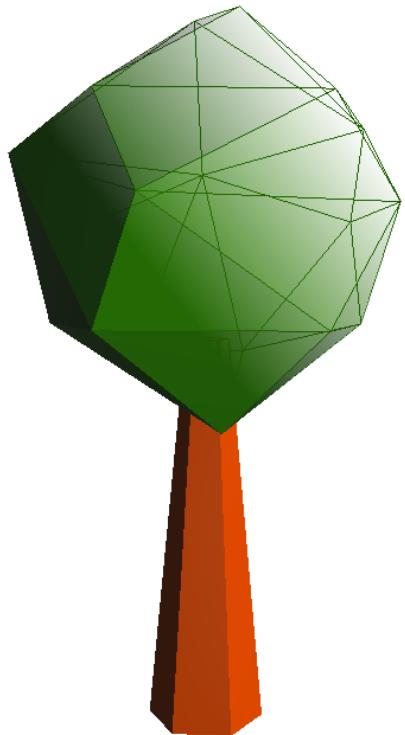


Übung: Szenengraph

- Schreiben Sie einen Algorithmus in Pseudocode, der 12 Einhörner auf einem 3D Gitter (3×4) positioniert. Der Abstand zwischen den Zellen ist Δ . Das erste Einhorn wird an Stelle (x_0, y_0) positioniert. Die übrigen Einhörner folgen in den weiteren Zellen. Der Szenengraph hat die folgenden Knotentypen:
 - Gruppenknoten
 - Verschiebungsknoten
 - Einhornknoten



Source: Deviantart: wallakitty, Creative Commons 3.0, <http://fav.me/d8l29c9>



Augmented Reality

Was ist Augmented Reality?

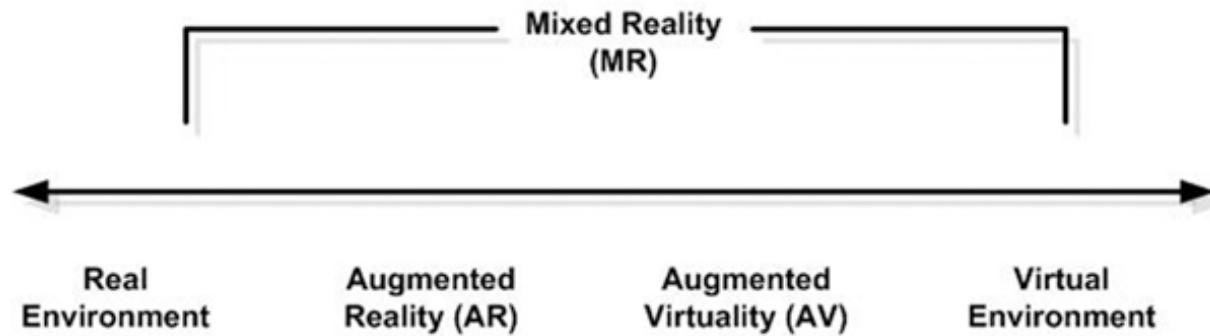
- Augmentierung (Erweiterung) der Realität mit virtuellen Objekten



- Phasen
 - Videoaufnahme (z.B. Smart Phone Livestream)
 - Tracking (Erkennen von Markern oder Fixpunkten)
 - Registrierung
 - Rendering (nahtlose Integration in Realbild)
 - Überlagerung durch die virtuellen Objekte

Definition

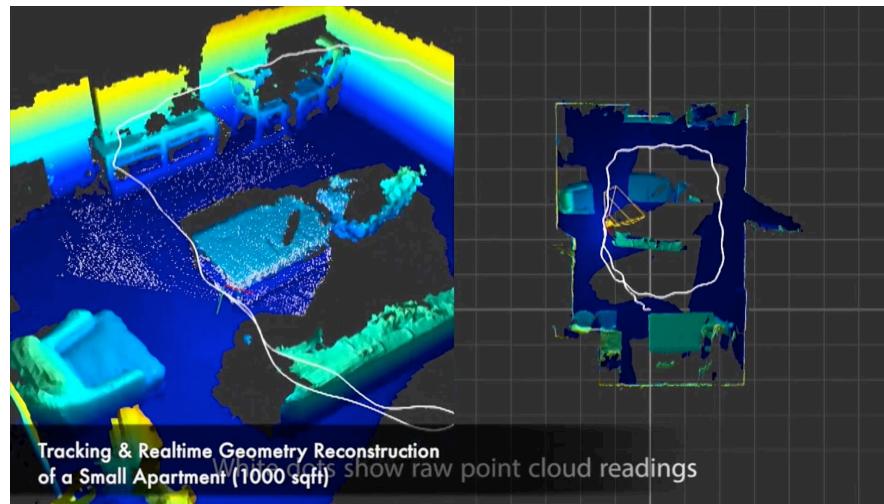
- Eigenschaften
 - Augmentierung (Erweiterung)
 - Interaktion
 - Echtzeit
- Glatte Transition zwischen AR und VR
 - Mixed Reality



Quelle: Milgram, Kishino, 1994

Herausforderungen

- Tracking
- glatte Integration von virtuellen Objekten
- Portabilität (mobil)
- Aufbau einer Karte der Umgebung (simultaneous localization and mapping, SLAM)
- Aufbau eines 3D-Modells der Umgebung (Scans + Rekonstruktion)



Quelle: Google:



Typen

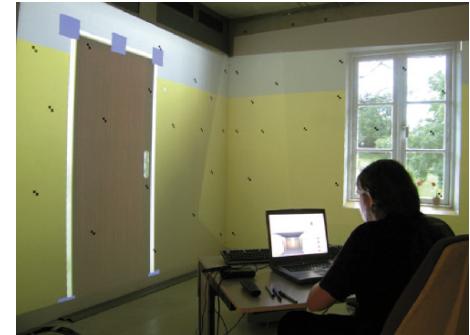
- video-see-thru AR
- optisches see-thru AR
 - semi-transparentes Display notwendig
- projektiionsbasiertes AR



video-see-thru AR



optisches see-thru AR
(e.g. Microsoft HoloLens)



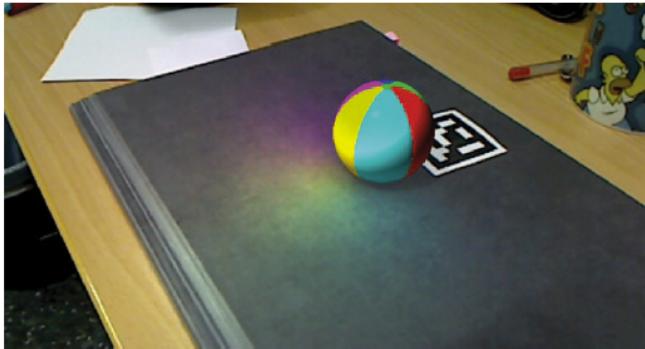
projektionsbasiertes
AR

Bausteine

- verschiedene Ansätze müssen integriert werden
 - location-based Services
 - Tracking
 - Computer Vision
 - Computergrafik

Registrierung

- geometriebasiert
 - Ergebnis des Trackings
 - Herausforderungen:
 - Latenz, Jittering
- Photometrische Registrierung
 - Anpassen an Umgebungslicht



photometrische Registrierung



geometrische Registrierung

Photometrische Registrierung

- schwieriges Problem
- benötigt Schätzung und Rekonstruktion in Echtzeit
- aktuell kaum zu leisten

Mobile Geräte

- Brillen



Microsoft HoloLens



Meta Vision Meta



Magic Leap One

- Smartphones



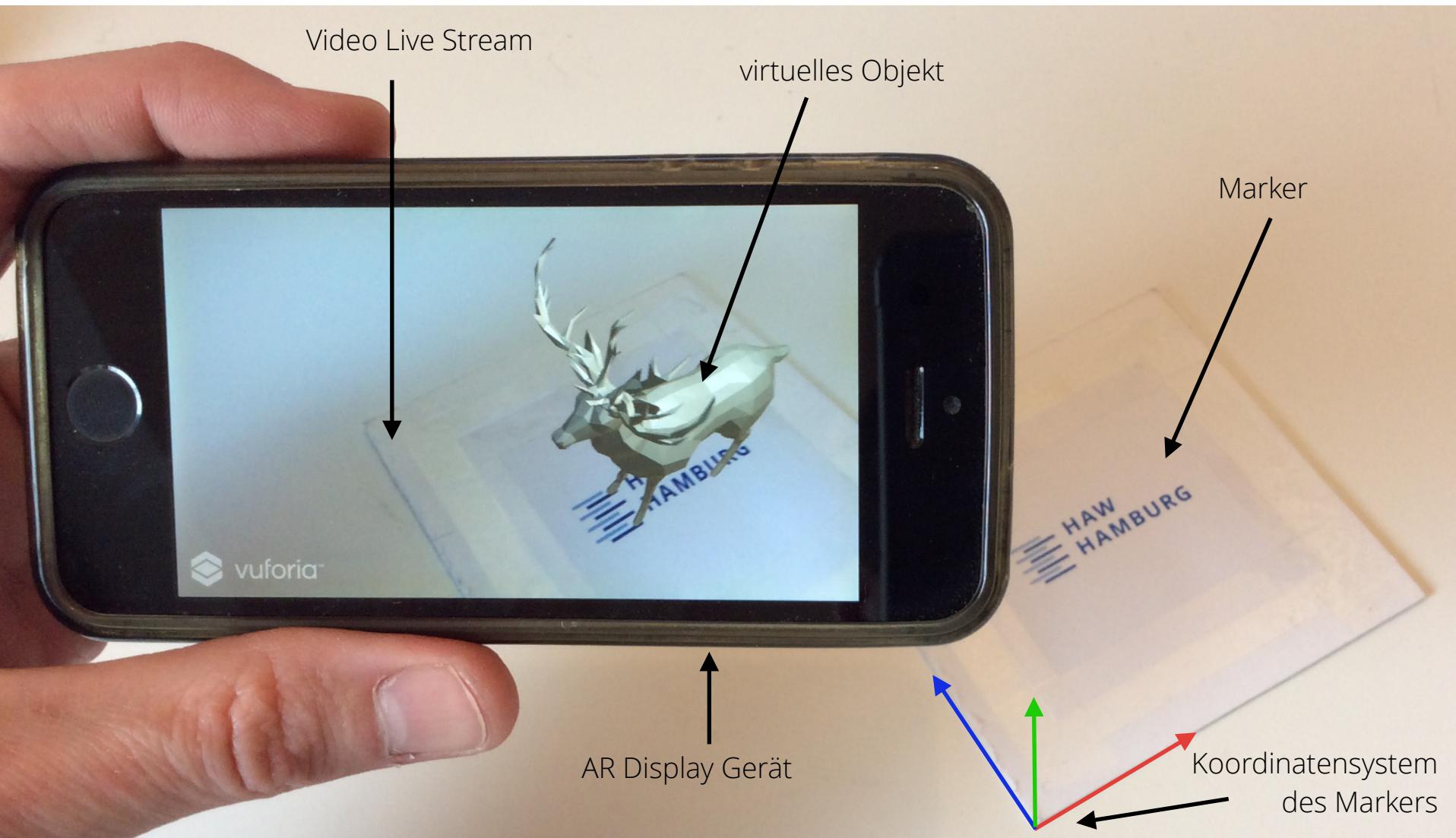
verschiedene

ARCore
G



maßgeschneiderte Technologie

Marker Tracking



Geräte Tracking



Zusammenfassung

- Organisation
- Grundlagen: Vektoren
- 3D-Raum
- Dreiecksnetze
- OpenGL ES
- Rendering in OpenGL
- Szenengraph
- Augmented Reality