Einführung

Einführung in die Computergrafik

Ausblick

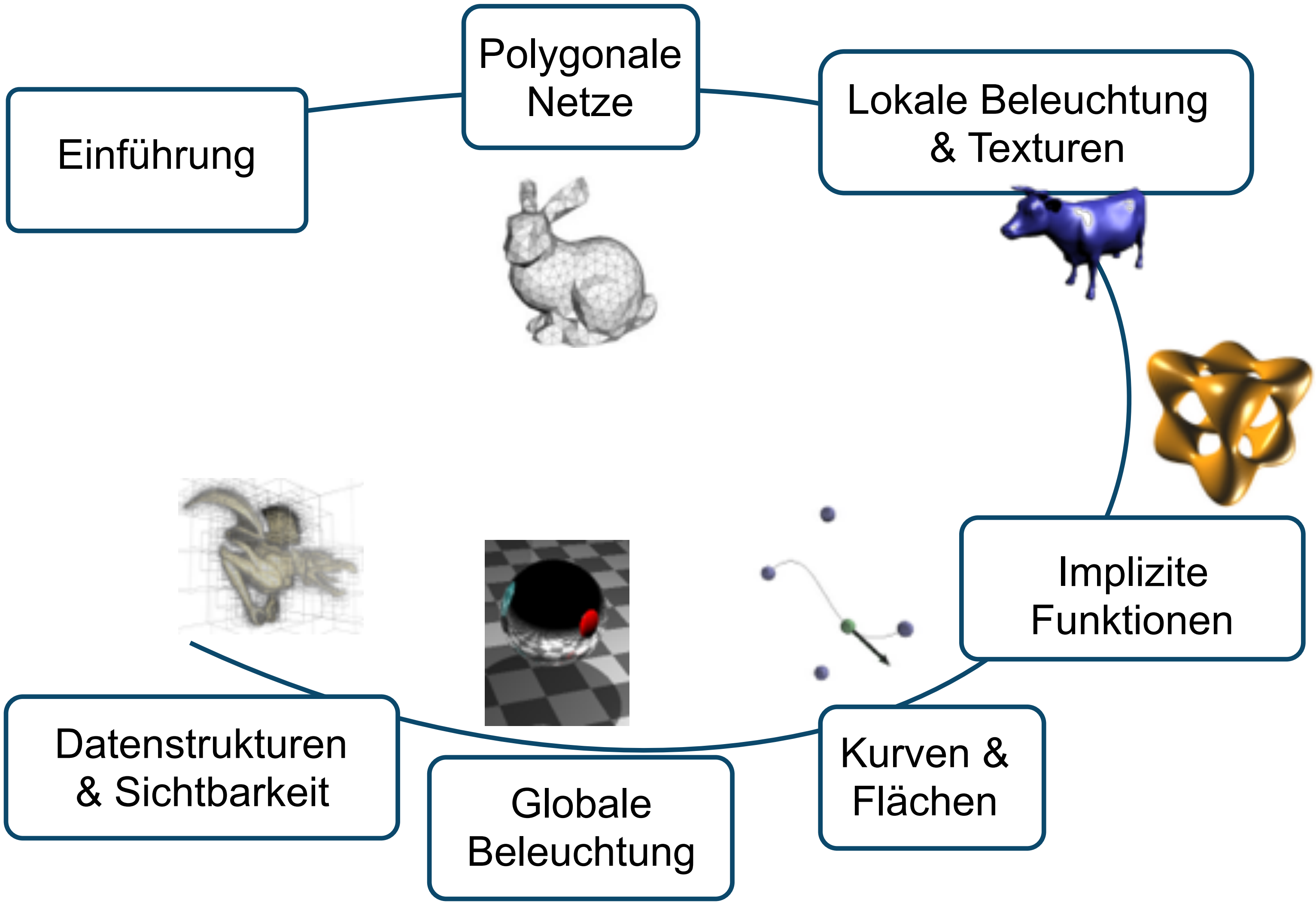
Worum gehts?

Agenda

- Organisation
- Der 3D Raum
- Dreiecksnetze
- OpenGL
- Framework für das Praktikum
- Szenengraph

Organisation

Inhalt


Einführung
Polygonale Netze
Lokale Beleuchtung & Texturen
Implizite Funktionen
Kurven & Flächen
Globale Beleuchtung
Datenstrukturen & Sichtbarkeit

Voraussetzungen

- Programmierkenntnisse in Java, Grundlagen der Mathematik
- Interesse an den notwendigen mathematischen Grundlagen
- Technologien
    - Java 8
    - Eclipse Mars
- Hohe Motivation
- Fähigkeit, systematisch und gewissenhaft zu arbeiten
- Bereitschaft, ein Buch oder Online-Dokumentation 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)
- Folien und ggf. eigene Mitschrift

Online-Quellen

- OpenGL Programming Guide (Red Book):

  http://www.glprogramming.com/red/
- OpenGL API:

  http://docs.gl/
- JOGL:

  http://jogamp.org/jogl/www/

EMIL

- *http://www.elearning.haw-hamburg.de/course/view.php?id=**671553***
- alle Informationen und Materialien zu Vorlesung und Praktikum
- Schlüssel zur Selbsteinschreibung: ***WPCGWS1617***

Vorlesungen

- es wird (außer heute) keine üblichen Vorlesungen geben
- Vortrag als Video auf EMIL (spätestens 2 Wochen vorher)
- zum Vorlesungstermin:
    - Fragen zum Vortrag
    - Übungen
    - Vertiefungen
    - Herleitungen
    - …
- Vortrag muss vorher als Eigenstudium angesehen werden
    - Notizen machen
    - Fragen notieren

Praktikum

- 7 Aufgabenblätter
- Bearbeitung in 2er-Teams
- Abgabe
  - Aufgabe muss vollständig bearbeitet sein – nur noch punktuelle Anpassungen im Praktikumstermin
  - Vorstellung/Finalisierung im Praktikum
  - offensichtliche Plagiate werden geahndet – jedes Team muss eine eigene Lösung entwickeln

Prüfung

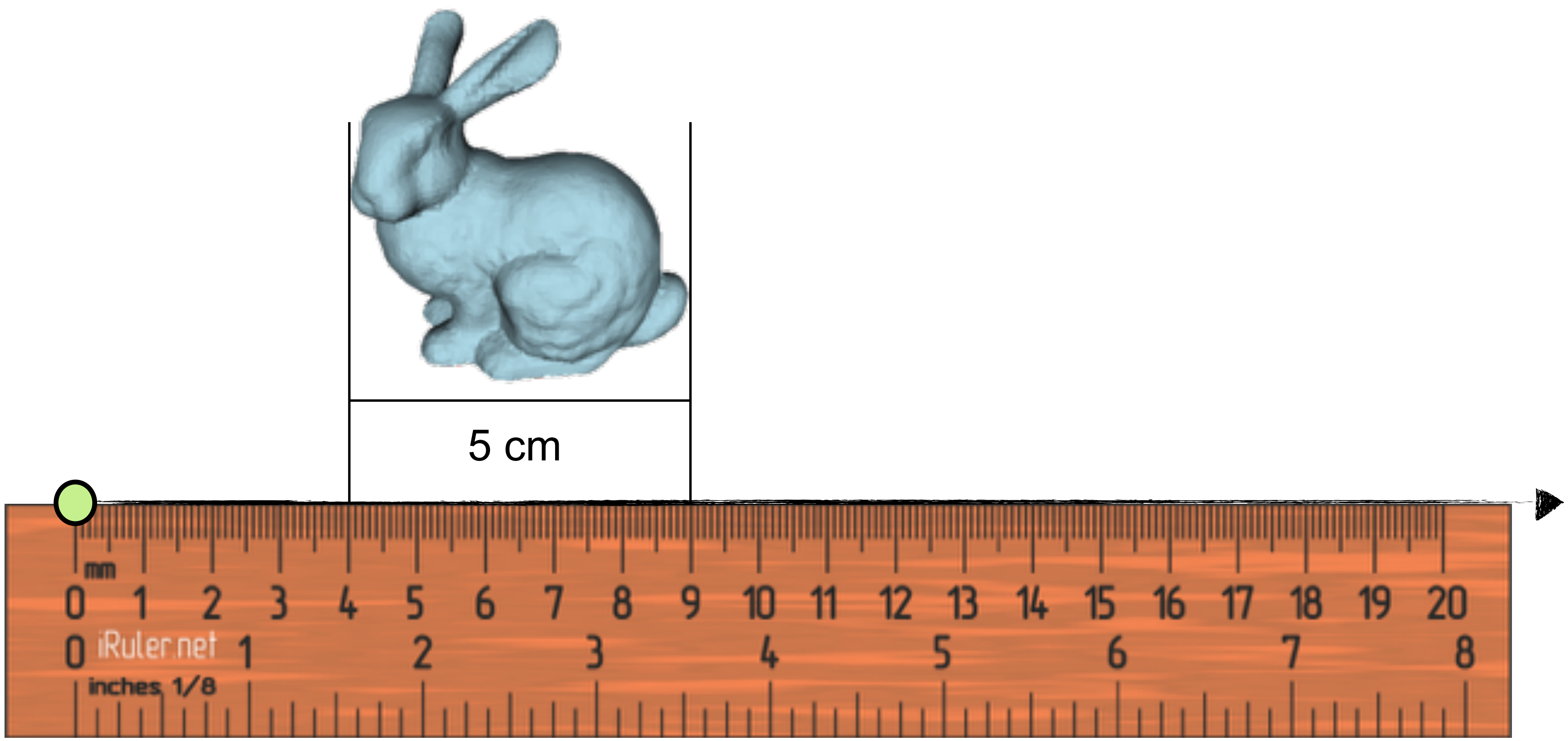
- Zwei Prüfungsmöglichkeiten
- verbindlicher Anmeldungstermin für eine der Möglichkeiten am Vorlesungsende
    - genaues Datum wir bekannt gegeben

- Möglichkeit 1: Klausur
- Möglichkeit 2: Projekt (Softwareentwicklung, Präsentation, Ausarbeiten (ca. 5-10 Seiten)

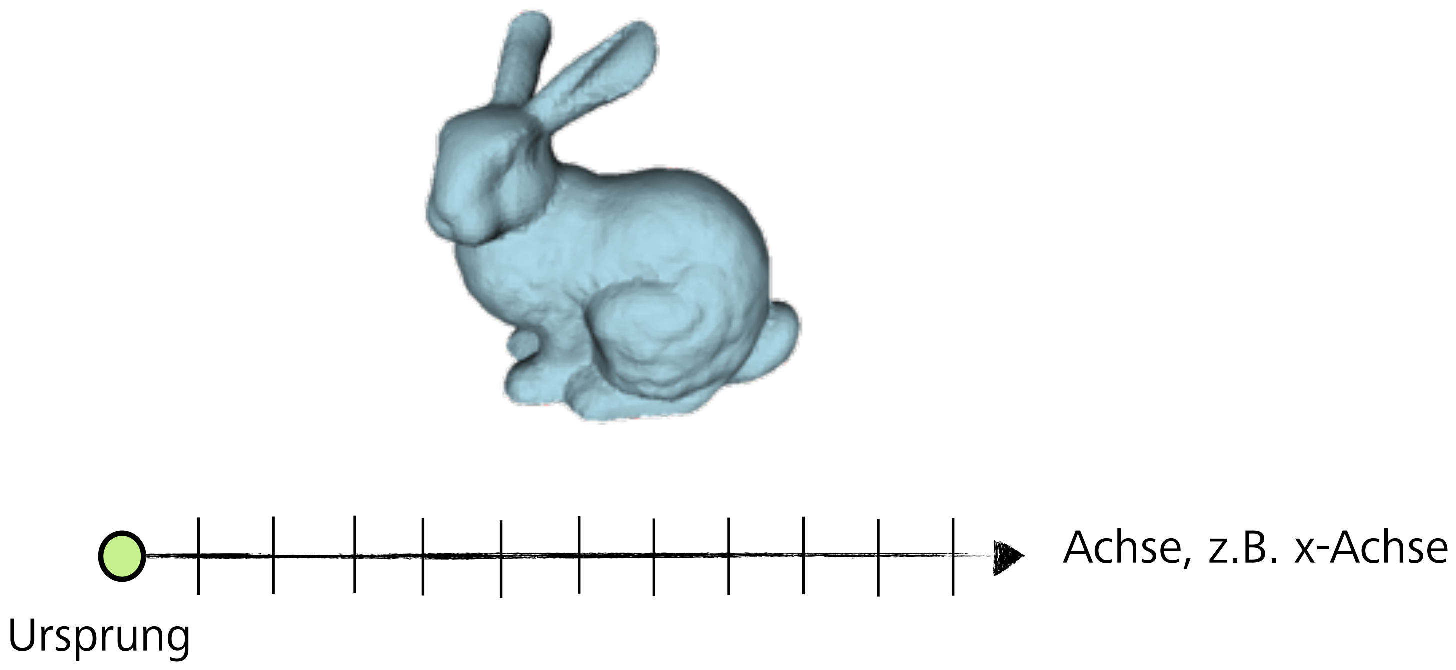
Der 3D Raum

Längen

- Längen messen: Skala benötigt

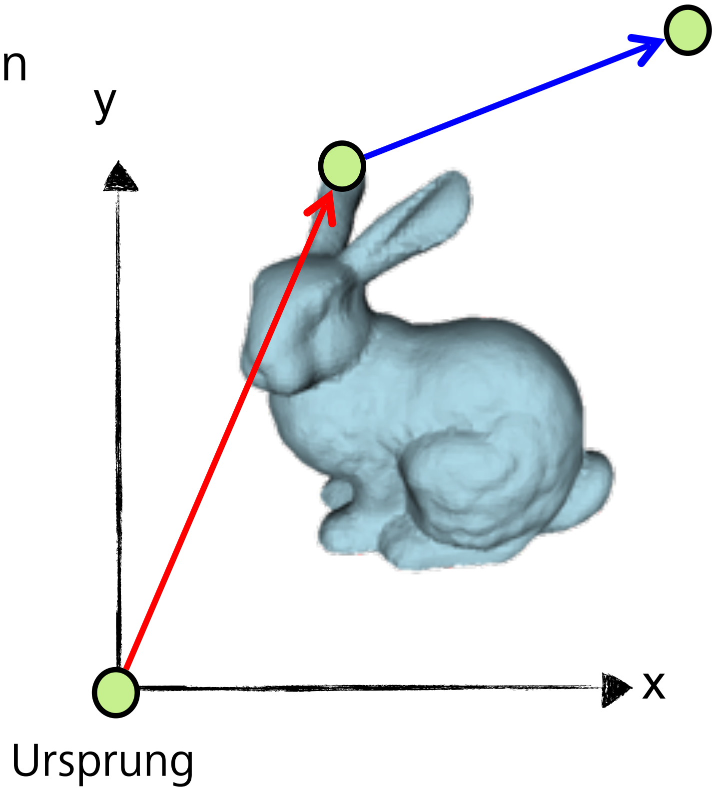


Länge entlang einer Achse



Koordinatensystem

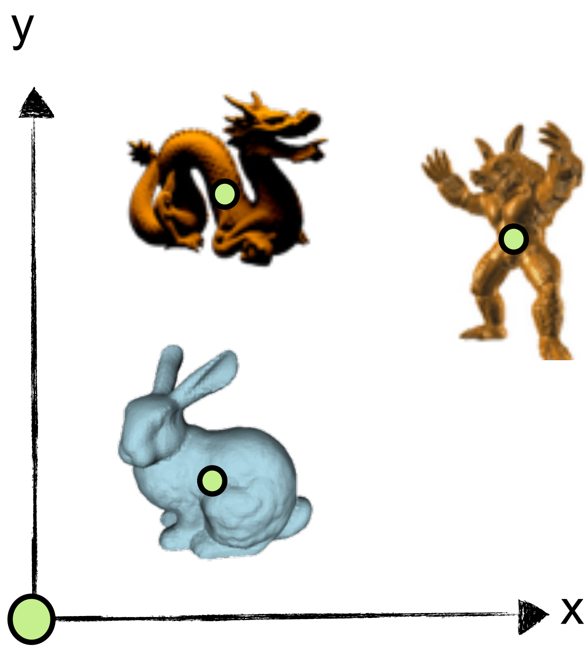
- Ursprung + Achsen = Koordinatensystem
- hier Karthesisches Koordinatensystem
  - Achsen senkrecht zueinander
  - Positionen im Raum sind Vektoren
    - Ortsvektor
  - Verschiebungen ebenso
    - Richtungsvektor



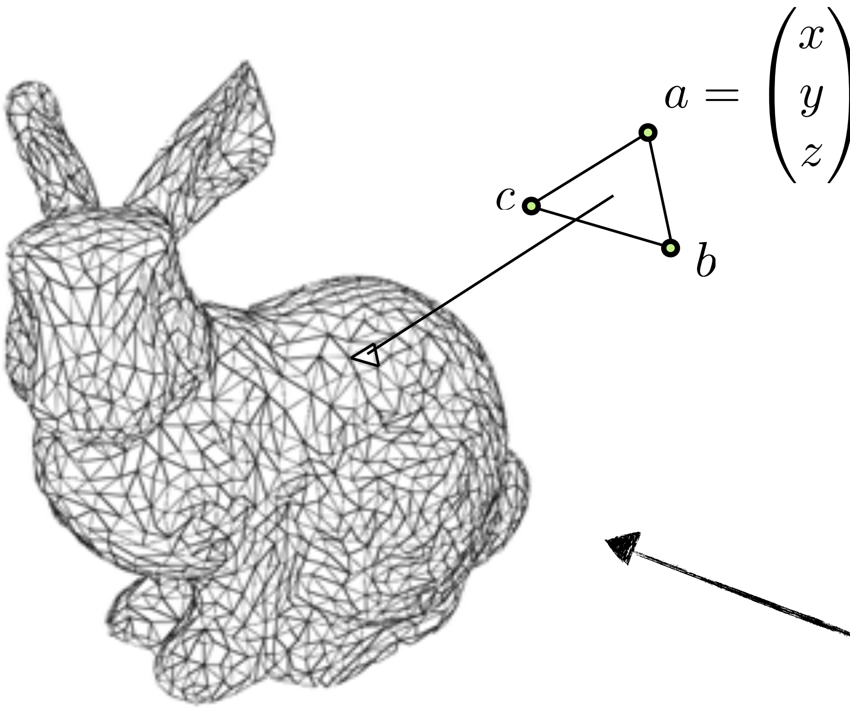
- Beispiel:
  - 2D Koordinatensystem
  - x- und y-Achse

Szenen

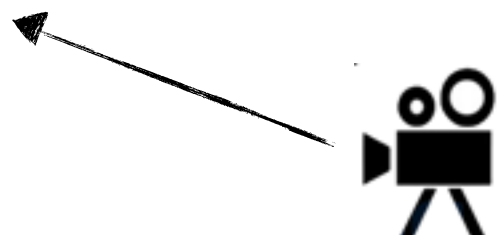
- Wir betrachten in der Computergrafik Szenen mit gemeinsamem Koordinatensystem (meist 3D, also x-, y- und z-Achse)
    - "Weltkoordinatensystem"
    - Position von Szenenobjekten im Weltkoordinatensystem



Ausgangssituation



Ziel: 2D Bild auf dem Bildschirm (= Farbwerte für Bildpunkte im Bildspeicher/ Framebuffer)



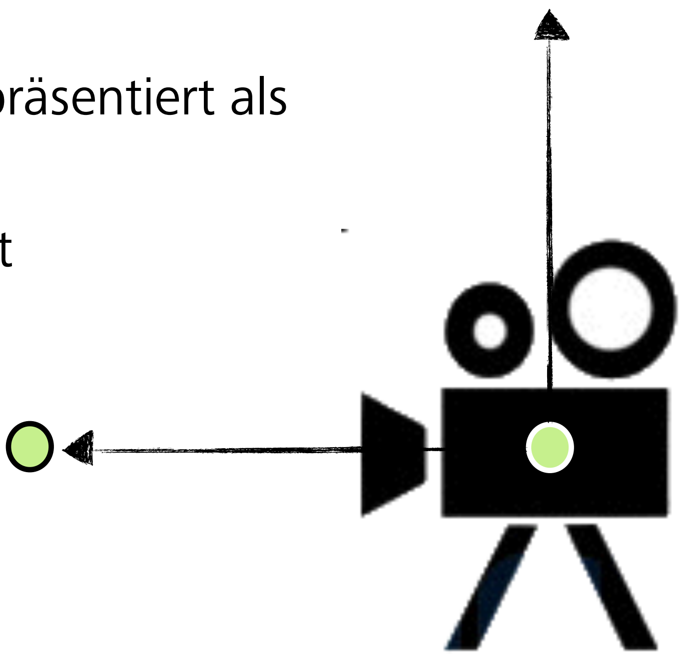
dreidimensionale Objekte (hier: Dreiecke mit 3D-Eckpunkten)

virtuelle Kamera

(virtuelle) Kamera

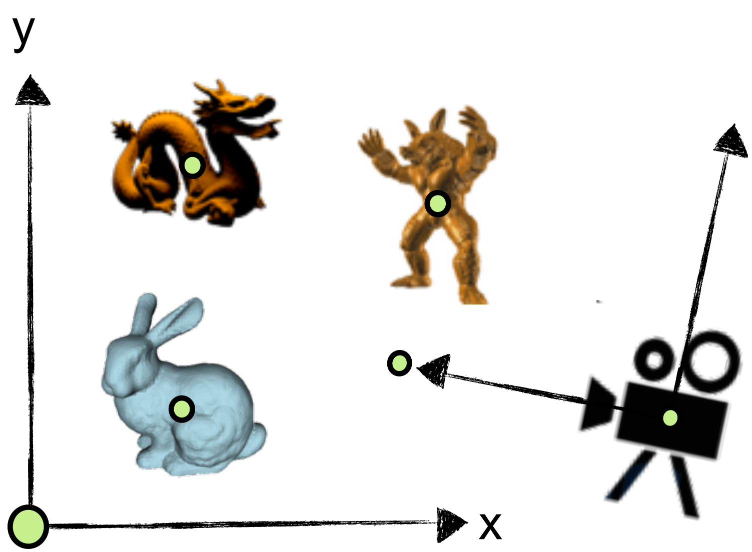
- Kamera bildet eigenes Koordinatensystem
  - Position
  - Blickrichtung
  - Oben-Vektor (ansonsten Rotation der Kamera nicht eindeutig)

- häufig alternativ repräsentiert als
  - Position
  - Referenzpunkt
  - Oben-Vektor



Koordinatensysteme

- Kamera hat eigenes Koordinatensystem
- bewegt sich durch das Weltkoordinatensystem
- Umrechnung: Transformation zwischen Koordinatensystemen
  - siehe Model & View-Transformation (Rendering Pipeline)
  - siehe Skript "Mathematische Grundlagen"



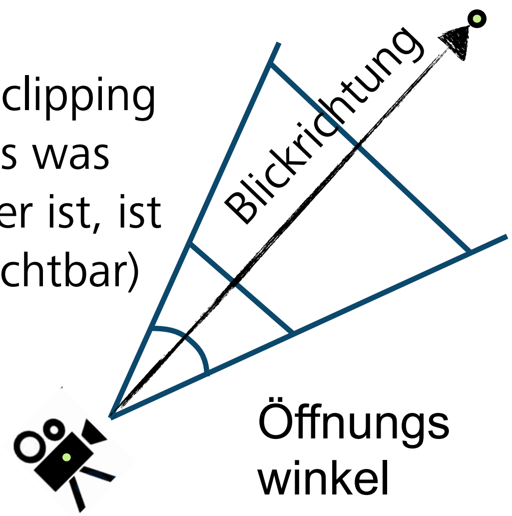
Virtuelle Kamera

- Kamera beschreibt Sichtvolumen

Öffnungswinkel im 3D ergibt sich aus Breite-Höhe-Verhältnis (aspect) und fovy

Referenz-
punkt

Nahclipping (alles was näher ist, ist unsichtbar)

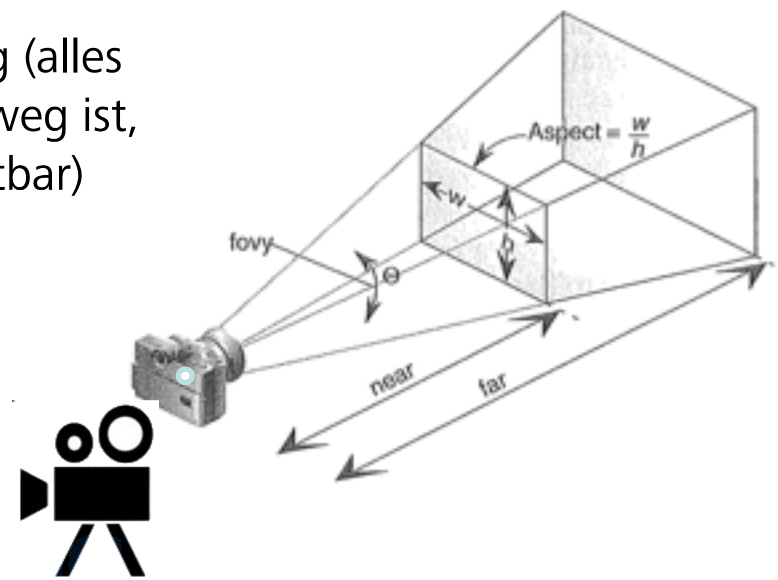


Fernclipping (alles was weiter weg ist, ist unsichtbar)

Öffnungs
winkel

Kamera-
position

Sichtvolumen im 2D



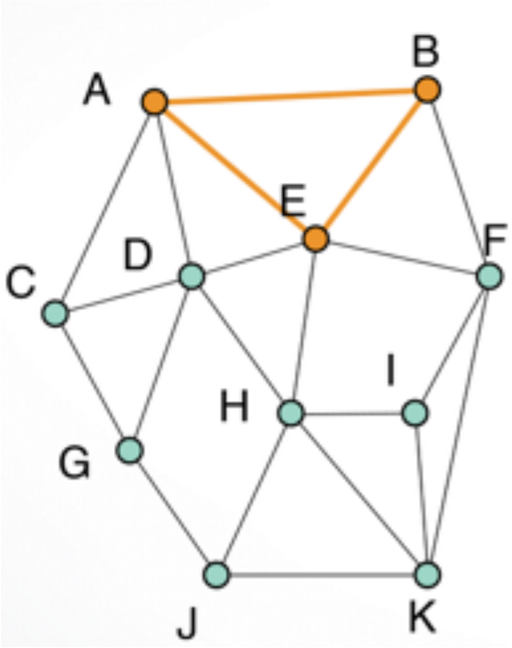
Sichtvolumen im 3D

Dreiecksnetze

Polygonales Netz

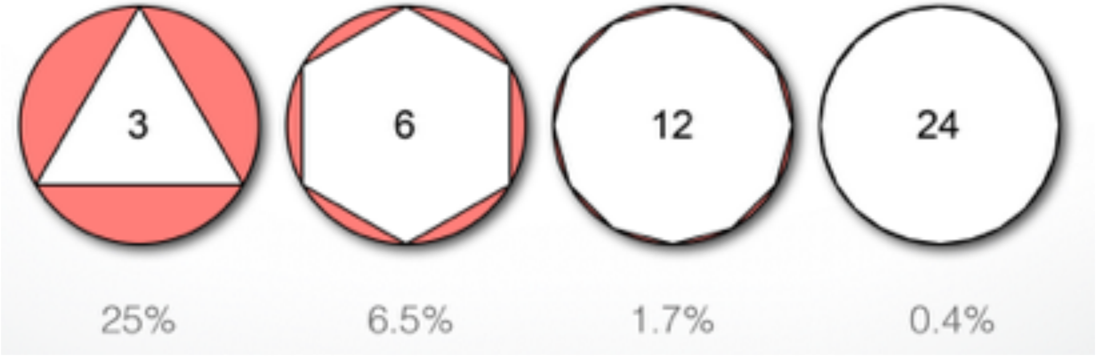
- Graph-Struktur
  - Graph {V,E}
  - Knoten: Vertices V = {A, B, C, D, … }
  - Kanten E = {(AB), (AE), …}
  - Facetten F = {(AEB), (EFB), …}

- Grad oder Valenz eines Knotens
  - Anzahl anliegenden Kanten
  - Beispiele
    - deg(A) = 4
    - deg(E) = 5

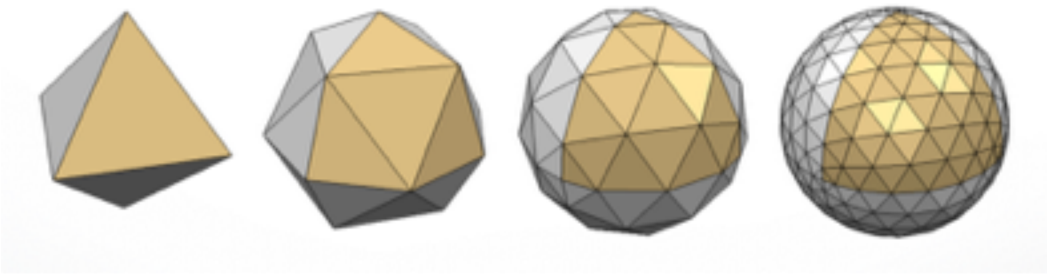


Warum Polygone?

- Gute Approximationseigenschaften (O(h2))



- Fehler invers proportional zur Anzahl der Facetten



Quelle: [1]

Repräsentation

- Definition
- ein Polygonales Netz ist definiert durch eine Knotenliste V und eine Facettenliste F
- jeder Knoten (vertex) $v \in V$ ist durch einen Punkt $v.p \in R3$ definiert
- jede Facette $f \in F$ besteht aus einer Liste von Knoten
- die Punkte einer Facette sollten dabei in einer Ebene liegen
    - nicht immer garantiert
- häufigste Facettentypen sind Dreiecke und Vierecke
    - meist auch Beschränkung auf konvexe Polygone

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
x_1 y_1 z_1
...
x_V y_V z_V

Triangles
i_{11} i_{12} i_{13}
...
...
...
...
i_{F1} i_{F2} i_{F3}

Index-Facettenliste (Verwendung im OBJ-Datenformat)

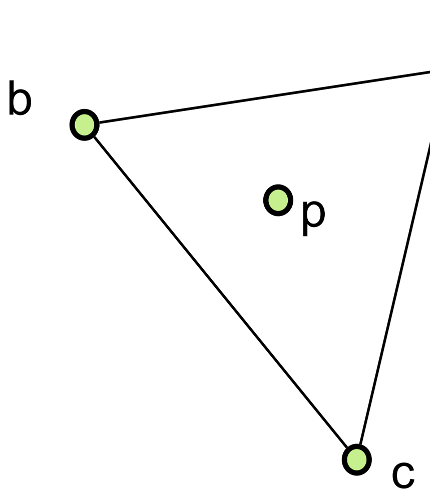
Einführung

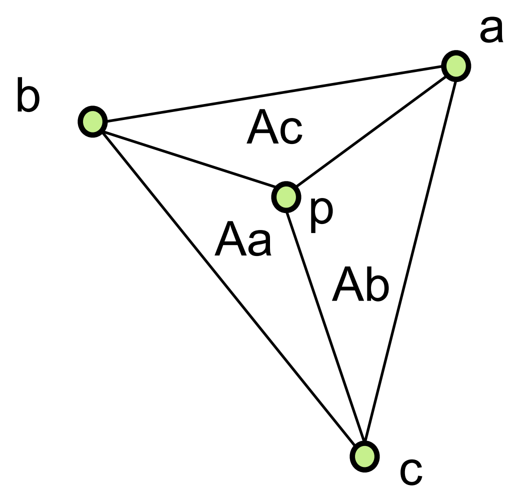
- wohlgeformte polygonale Netze:
  - an jeder Kante liegen 1 oder 2 Facetten an
  - an jeder Kante liegen zwei Knoten an
  - jede Facette ist durch Kanten begrenzt
  - Kanten und Facetten bilden Fan oder Scheibe

Einführung

- eine gängige Vereinfachung ist die Beschränkung auf Dreiecksnetze
    - Triangulieren der Polygone beim Einlesen
    - jedes polygonale Netz ist triangulierbar
    - alle Facetten sind Dreiecke und somit automatisch eben
    - für Dreiecksnetze ohne Rand gilt $2e = 3f$
    - typischerweise gilt: $f \approx 2v$ und $e \approx 3v$
    - die mittlere Valenz ist 6 [$\Sigma V\ v(V) = 6v$]
- Erweiterung um weitere Attribute
    - gleiche Indizierung wie Knoten oder Facetten
        - z.B. Farbe
    - eigene Indizierung
        - z.B. Texturkoordinaten

Baryzentrische Koordinaten





$$\alpha = \frac{A_a}{A}$$

$$\beta = \frac{A_b}{A}$$

$$\gamma = \frac{A_c}{A}$$

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

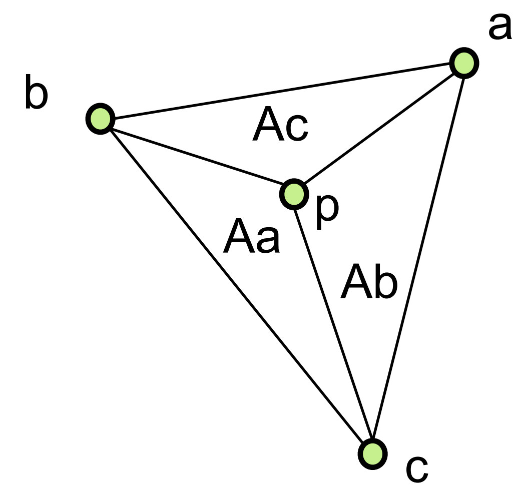
Punkt innerhalb des Dreiecks:

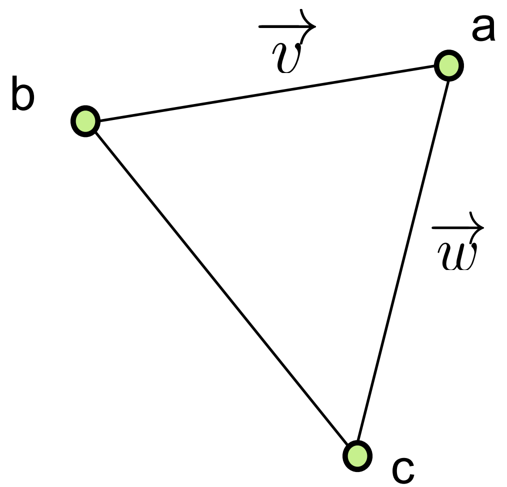
$$\alpha + \beta + \gamma = 1$$

$$0 \leq \alpha, \beta, \gamma \leq 1$$

Flächeninhalt

- Berechnen von Aa, Ab und Ac
- Kreuzprodukt!

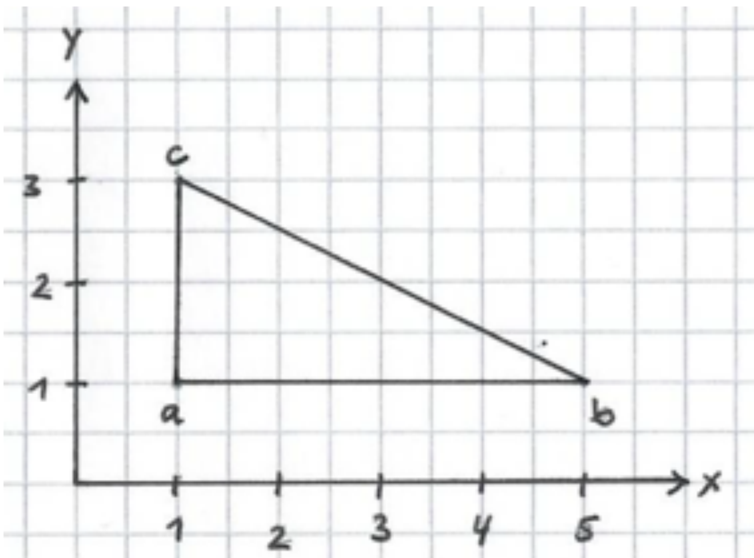




$$|\overrightarrow{v} \times \overrightarrow{w}| = A_{Parallelogramm} = 2A_{Dreieck}$$

Übung: Flächeninhalt

- Berechnen Sie den Flächeninhalt des folgenden Dreiecks



$$a = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$b = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix}$$

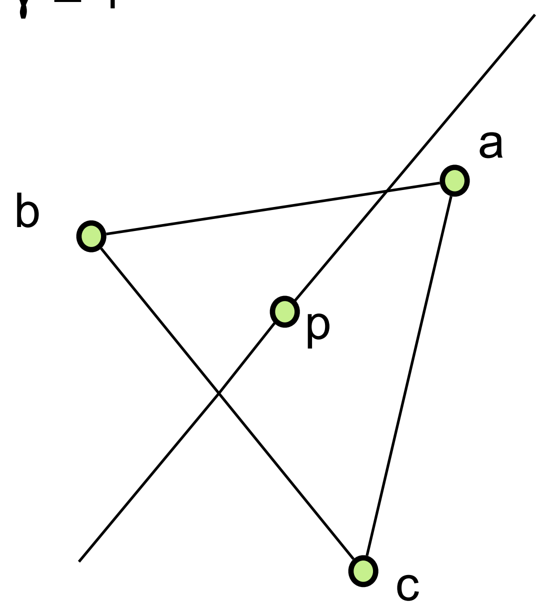
$$c = \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix}$$

Erinnerung:
Kreuzprodukt

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

Anwendungsbeispiel

- Schnitt Strahl-Dreiecks
  - Ebene aus a, b, c (z.B. mit Hesse-Normalform)
  - Schnitt: p = Ebene – Strahl
  - Berechnen der Baryzentrischen Koordinaten von p
    - falls $0 <= \alpha, \beta, \gamma <= 1$ und $\alpha + \beta + \gamma = 1$

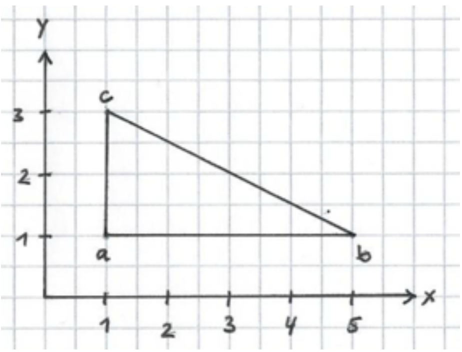


Hausaufgabe

- Prüfen Sie, ob der Strahl

$$s = \begin{pmatrix} 4 \\ 1.5 \\ -2 \end{pmatrix} + \lambda \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

- das folgende Dreieck schneidet:



$$a = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad b = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} \quad c = \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix}$$

OpenGL

Rendering Pipeline

3D Modell

Texturen, Materialien

Kamera

Grafikkarte
(Hardware)

Bibliothek, API

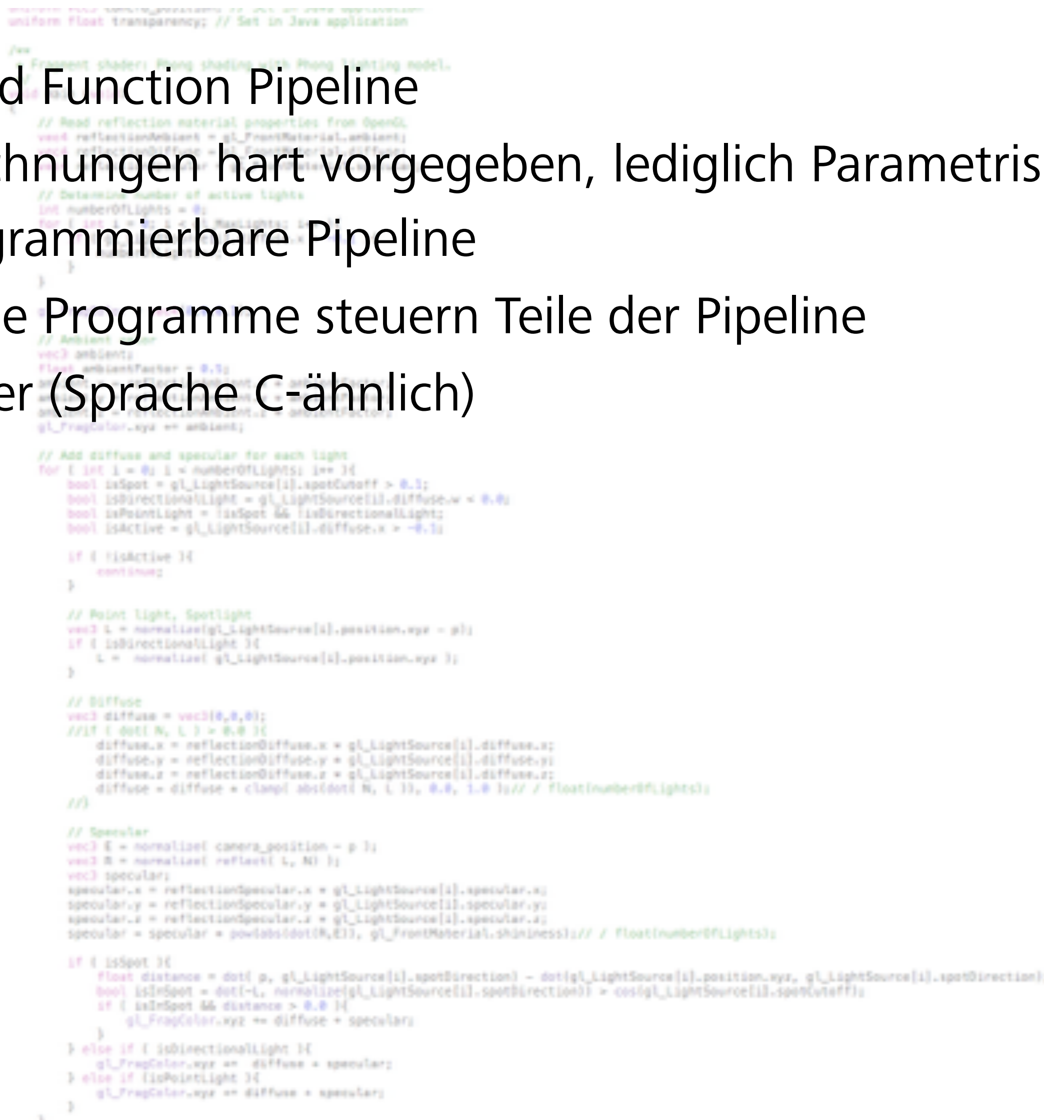
Rendering Pipeline

- **Model-Transformation:** Transformationen auf die Primitive der Szene (siehe Transformationsknoten im Szenengraph)
- **View-Transformation:** Ausrichten der Szene = Transformation aller Objekte so, dass Kamera in z-Richtung blickt
- **Projektion:** Abbildung vom 3D auf Bildebene
- **Clipping:** Abschneiden der Teile, die nicht auf Renderfläche sichtbar sind
- **Screen Mapping:** Verschieben entsprechend der Renderfläche auf dem Bildschirm
- **Rasterisierung:** Primitive → Pixel

Jeder Schritt = Multiplikation mit einer Matrix!

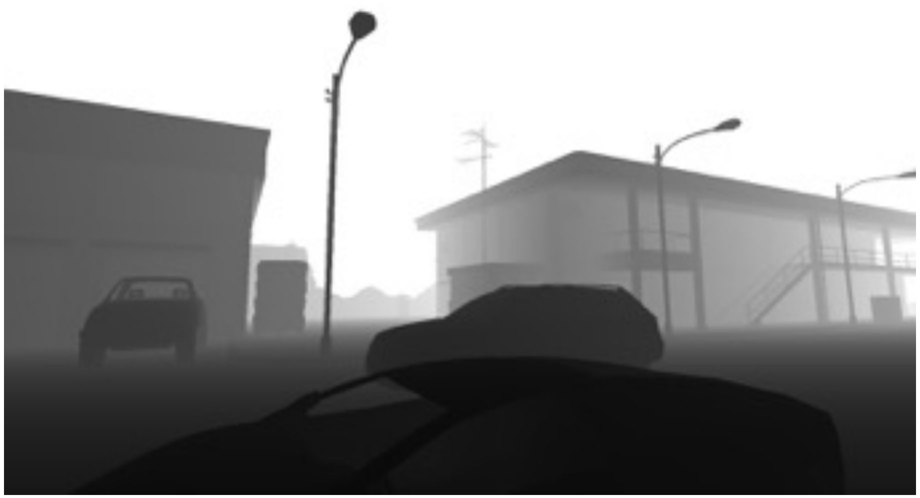
Rendering Pipeline

- früher: Fixed Function Pipeline
  - Berechnungen hart vorgegeben, lediglich Parametrisierung möglich
- heute: programmierbare Pipeline
  - eigene Programme steuern Teile der Pipeline
  - Shader (Sprache C-ähnlich)

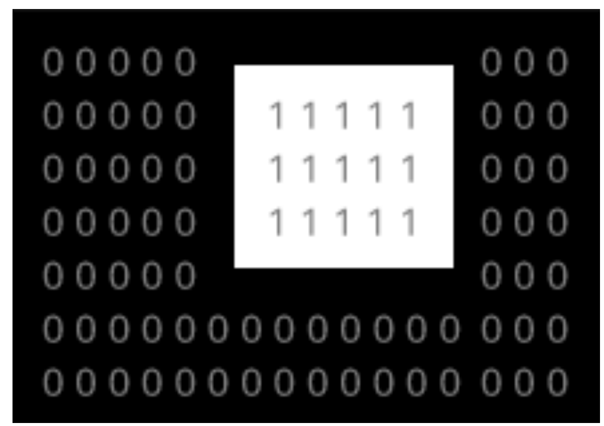


Framebuffer

- Ergebnis der Rendering Pipeline: Matrix von Farbwerten
  - wird im Farbpuffer abgelegt (Auflösung identisch zu Renderfläche)
  - Farben im RGB-Format
- Framebuffer beinhaltet zusätzliche Puffer (neben Farbpuffer)
  - Tiefenpuffer (Depth Buffer, z-Buffer): Tiefstwert pro Pixel
  - Stencil Buffer: programmierbarer Zähler pro Pixel



Tiefenpuffer

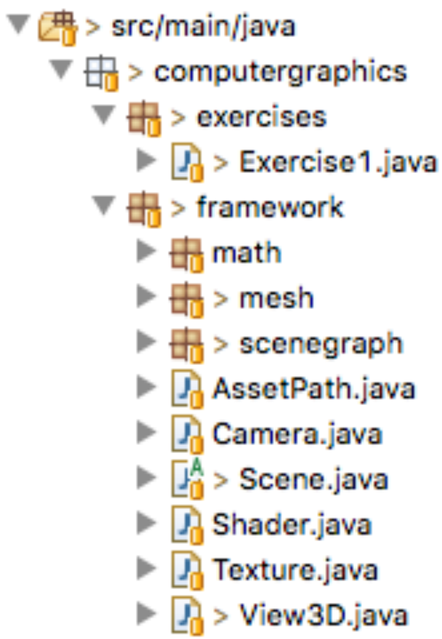


Stencil Buffer (Schema)

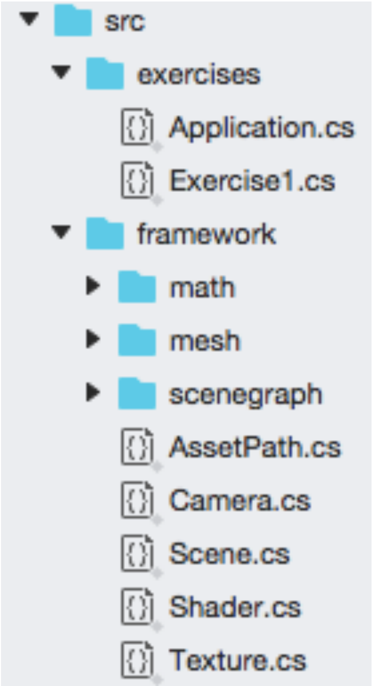
Framework für das Praktikum

Aufbau Framework

- Framework wird in zwei Varianten bereitgestellt: Java, C#
- Empfohlene Entwicklungsumgebungen:
  - Java -> Eclipse
  - C# -> Xamarin Studio



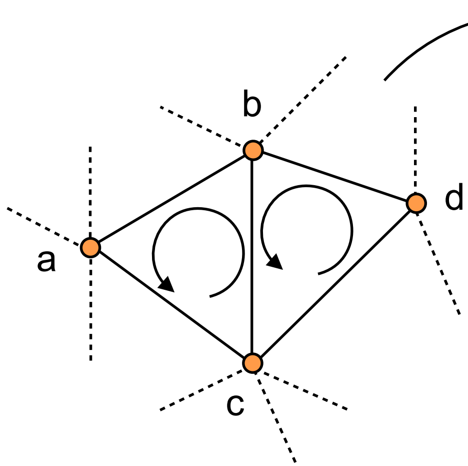
Java



C#

Zeichnen von Primitiven

- OpenGL kann mit unterschiedlichen Primitiven umgehen (z.B. Dreiecke, Quads …)
- alle relevanten Informationen werden zusammengefasst an Grafikkarte übertragen (Vertex Buffer Object)
- Abstraktion im Framework: Klasse



Vertex Buffer:

	…	a_x	a_y	a_z	c_x	c_y	c_z	b_x	b_y	b_z	c_x	c_y	c_z	d_x	d_y	d_z	b_x	b_y	b_z	…	

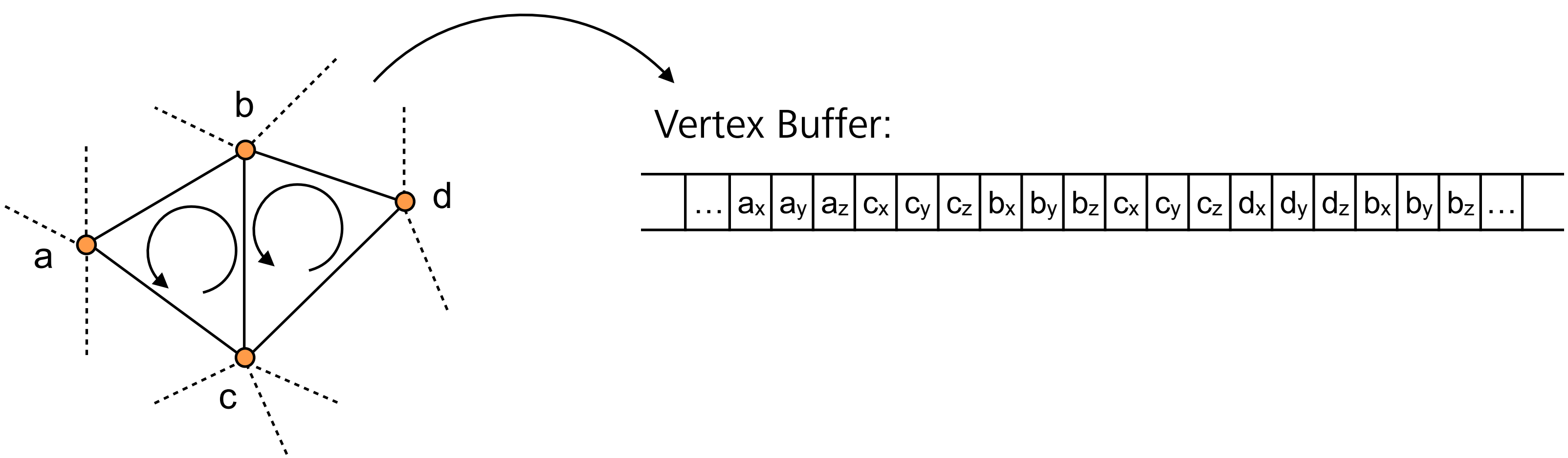
Weitere Puffer für:
- Normalen
- Farben
- Texturkoordinaten

Verwenden von VertexBufferObject

- Methode

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

  aufrufen:
    - erhält als Argument eine Liste von Vertices
        - bei Dreiecken: 3 pro Dreieck
    - und den Primitivtyp, z.B. Dreiecke:
        - GL2.GL_TRIANGLES in Java
        - PrimitiveType.Triangles in C#



Szenengraph

Szenenbeschreibung

- Datenstruktur zur Speicherung und Strukturierung von Szenendaten
- Der Szenengraph ist ein gerichteter azyklischer Graph von Knoten (Objekten) Die Knoten enthalten Informationen über
    - Geometrie
    - Materialeigenschaften
    - Gruppen
- Bearbeitet wird eine Szene, indem Aktionen auf den Graph oder Teilgraph angewendet werden:
    - Darstellen (Rendering)
    - Auswählen (Picking)
    - Berechnen von Bounding Boxen
        - Schreiben in ein File

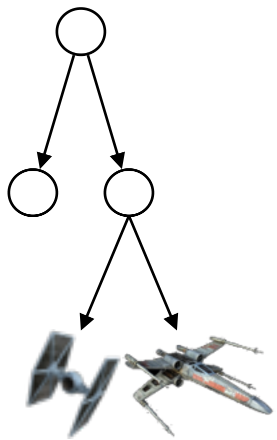
Knoten (Framwork)

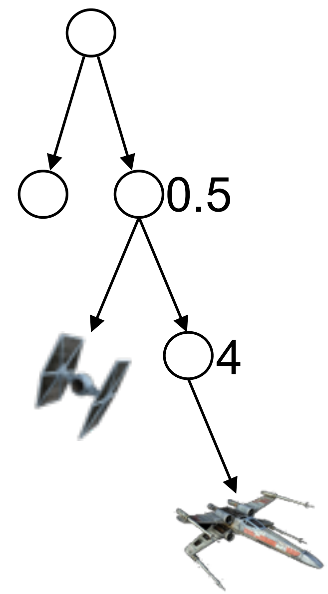
- INode: abstrakte Basisklasse
- InnerNode: beinhaltet mehrere Kindknoten
- LeafNode: keine Kindknoten aber Inhalt, der gezeichnet werden soll
- Transformations-Knoten
    - ScaleNode: Skalierung der Kindknoten
    - TranslationNode: Verschiebung der Kindknoten
- RootNote: Wurzelknoten mit Szeneninformation
- Geometrie-Knoten
    - CubeNode: Repräsentiert einen Würfel
    - SphereNode: Repräsentiert eine Kugel

Szenengraph

- Eigenschaften eines Knotens gelten für alle Kindknoten und deren Kinder (rekursiv)
- Beispiel: Skalierungsknoten

Szenengraph:





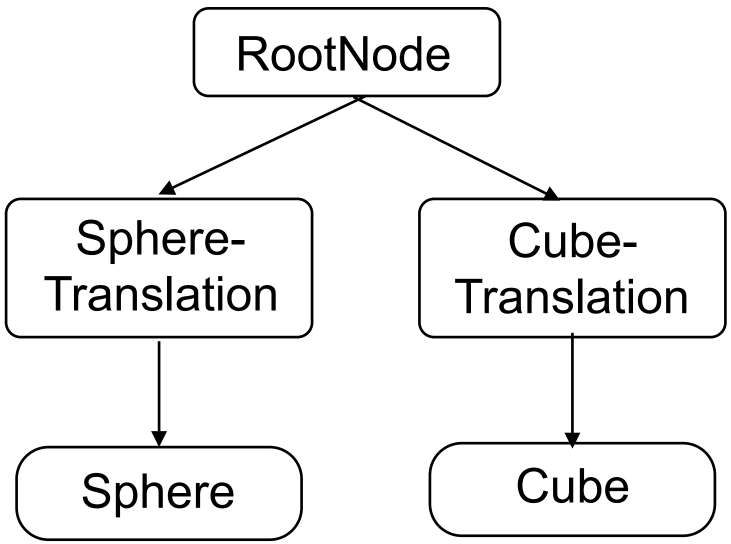
Ergebnis:

Szenengraph im Praktikum

- Beispiel (siehe computergraphics.applications.CGFrame):

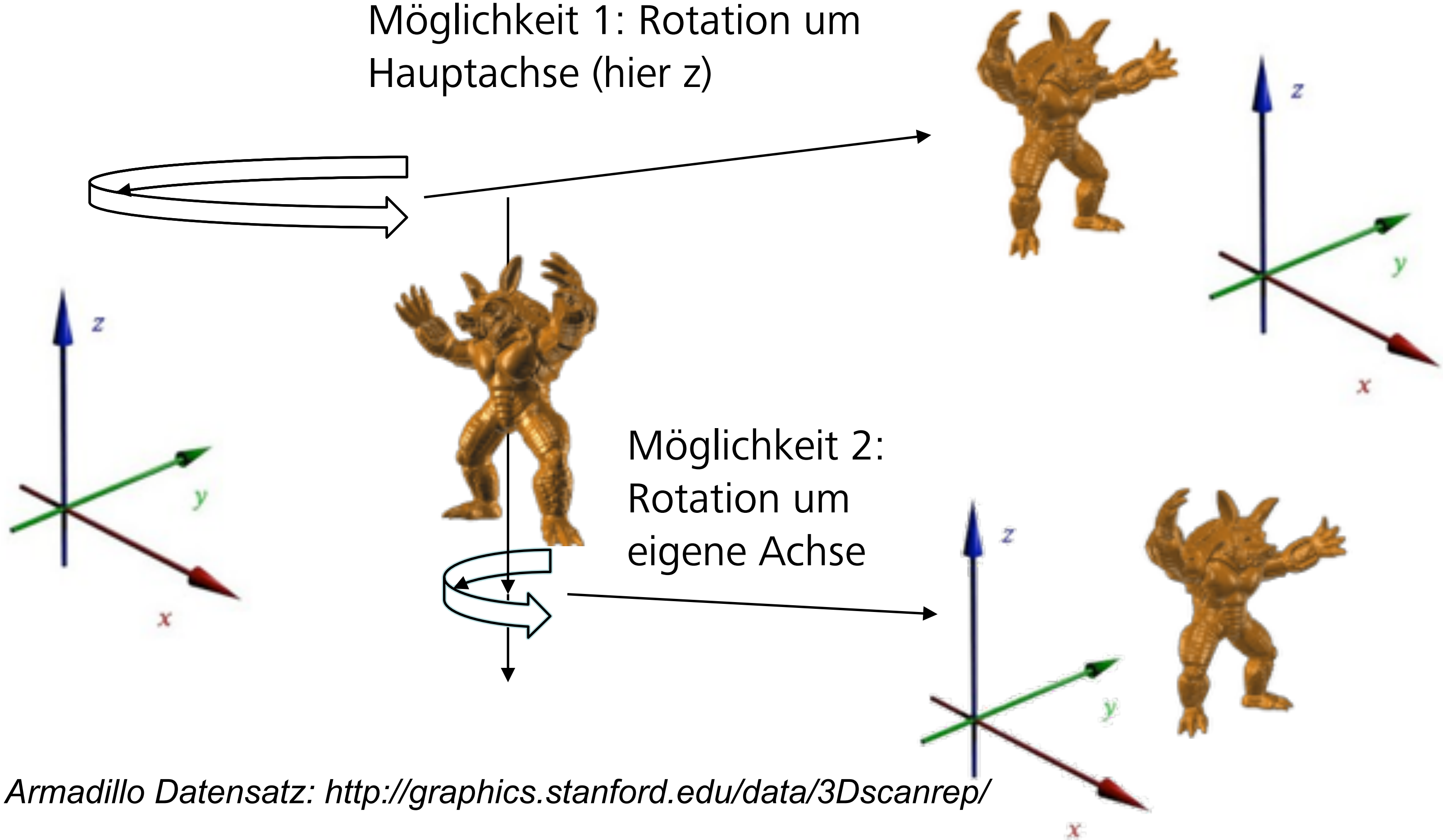
```
TranslationNode sphereTranslation =
new TranslationNode(new Vector3(-2f, 0, 0));
INode sphereNode = new SphereNode(0.5f, 20);
sphereTranslation.AddChild(sphereNode);
getRoot().AddChild(sphereTranslation);

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° gegen den Uhrzeigersinn

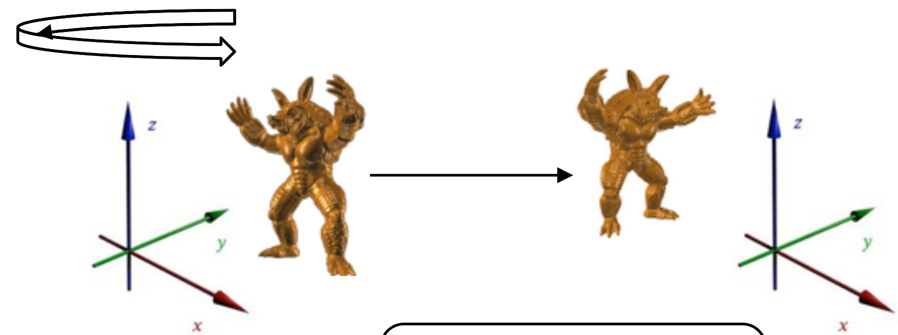


Armadillo Datensatz: http://graphics.stanford.edu/data/3Dscanrep/

Rotation um Hauptachse

- Rotationsmatrix um Z-Achse:

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



Armadillo

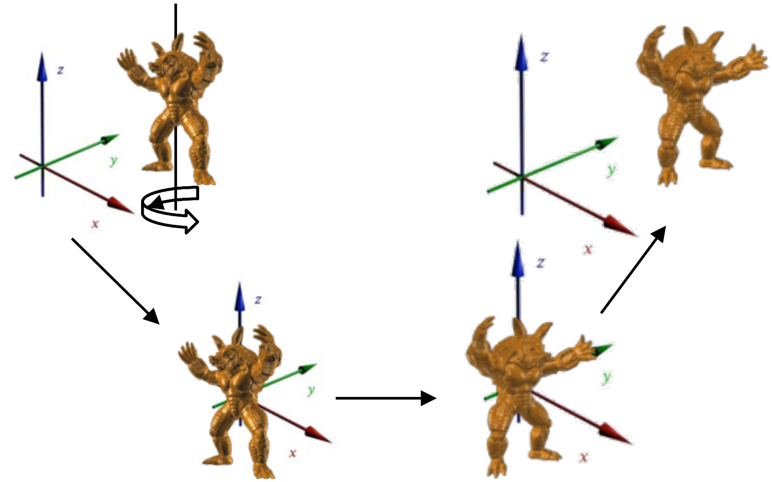
ohne Rotation um eigene Achse

Verschiebung zurück
↓
Rotation
↓
Verschiebung in Ursprung
↓
Armadillo

mit Rotation um eigene Achse

Rotation um eigene Achse

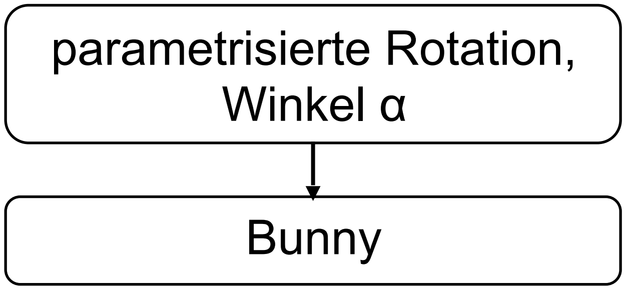
- Rotationsmatrix um eigene Achse
- Idee
  - Verschiebung in Ursprung
  - Rotation (z.B. um z-Achse)
  - Verschiebung zurück



- Umsetzung im Szenengraph
  - Einfügen eines Transformationsknotens T über dem Armadillo-Knoten
  - in T:
    - Rotationsmatrix an Kindknoten weitergeben

Animationen

- Beispiel: kontinuierliche Rotation
- Szenengraph:



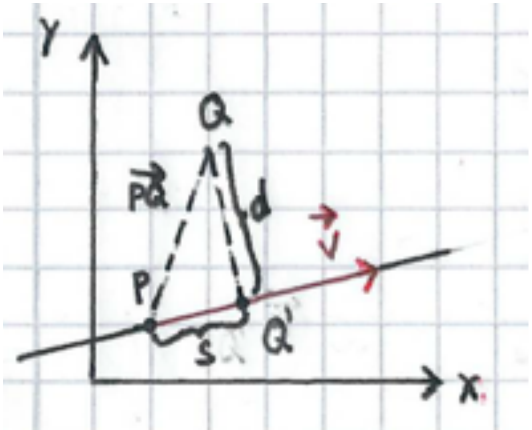
- in jedem Zeitschritt
  - Winkel α vergrößern
  - Rotationsmatrix neu setzen

Übung: Mathematische Grundlagen

- Berechnen Sie den Abstand zwischen dem Punkt Q und der Geraden g mit
- g = P + λv.

$$\mathbf{Q} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \vec{v} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

- Q', d und PQ sind Hilfskonstruktionen, die Sie verwenden können (aber nicht müssen)



Zusammenfassung

- Organisation
- Der 3D Raum
- Dreiecksnetze
- OpenGL
- Framework für das Praktikum
- Szenengraph

Quellen

- Die Folien sind inspiriert von Vorlesungsfolien von
- Prof. Dr. Wolfgang Straßer, Universität Tübingen, emeritiert
- Prof. Dr. Andreas Schilling, Universität Tübingen, emeritiert
- Prof. Dr. Stefan Gumhold, Technische Universität Dresden

- [1] Mario Botsch, Michael Spernat, Leif Kobbelt: Phong Splatting, Symposium on Point-Based Graphics 2004, 25-32