

—  
**Computergraphik**  
**WS 2015**  
—

Prof. Dr. R. Dörner, HS RheinMain

—  
**Teil A:**  
**Einführung**  
—



## Beispiel für Computergraphik

Quelle:  
Nintendo



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [3]



© R. Dörner

## Graphische Datenverarbeitung

GDV  
Computergraphik  
Computer Graphics  
CG

Graphische Datenverarbeitung ist die Technologie, mit der **Bilder** im allgemeinsten Sinn des Wortes (Graphiken, Grau- und Farbbilder) mit Hilfe von Prozessoren (Rechnern) **erfaßt** bzw. **erzeugt**, **veraltet**, **dargestellt**, **manipuliert**, in für die jeweilige Anwendung geeignete Form **verarbeitet** und mit sonstigen, auch nichtgraphischen Anwendungsdaten **in Wechselbeziehungen gebracht** werden.

Auch die rechnergestützte Integration und Handhabung dieser Bilder mit anderen Datentypen wie Audio, Sprache und Video (Multimediale Systeme) sowie die zugehörigen, fortgeschrittenen Dialogtechniken gehören dazu.

J. L. Encarnacao

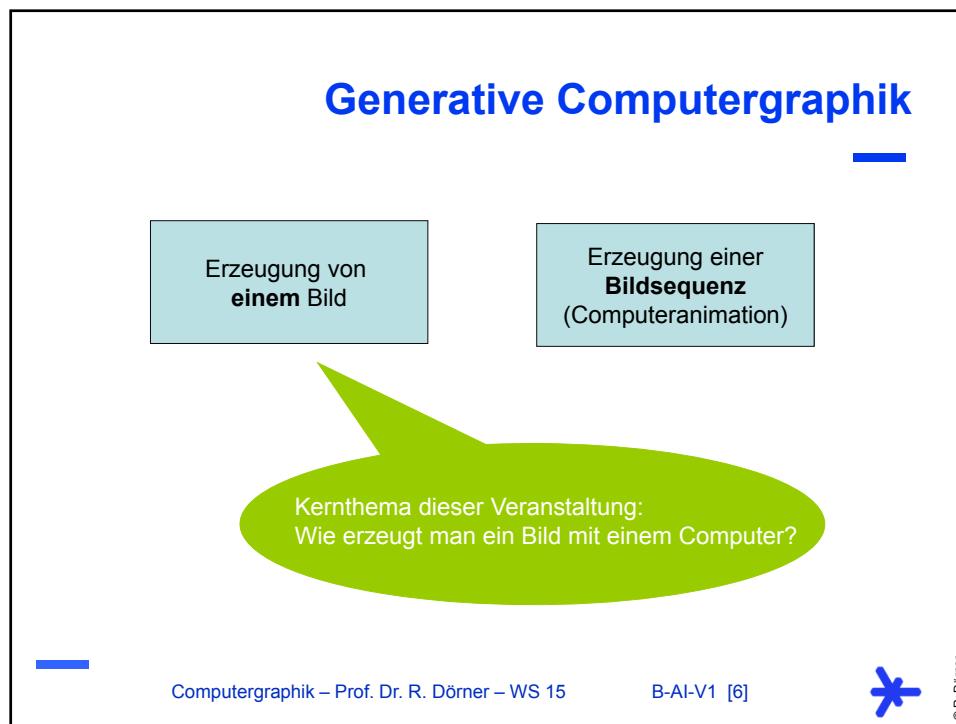
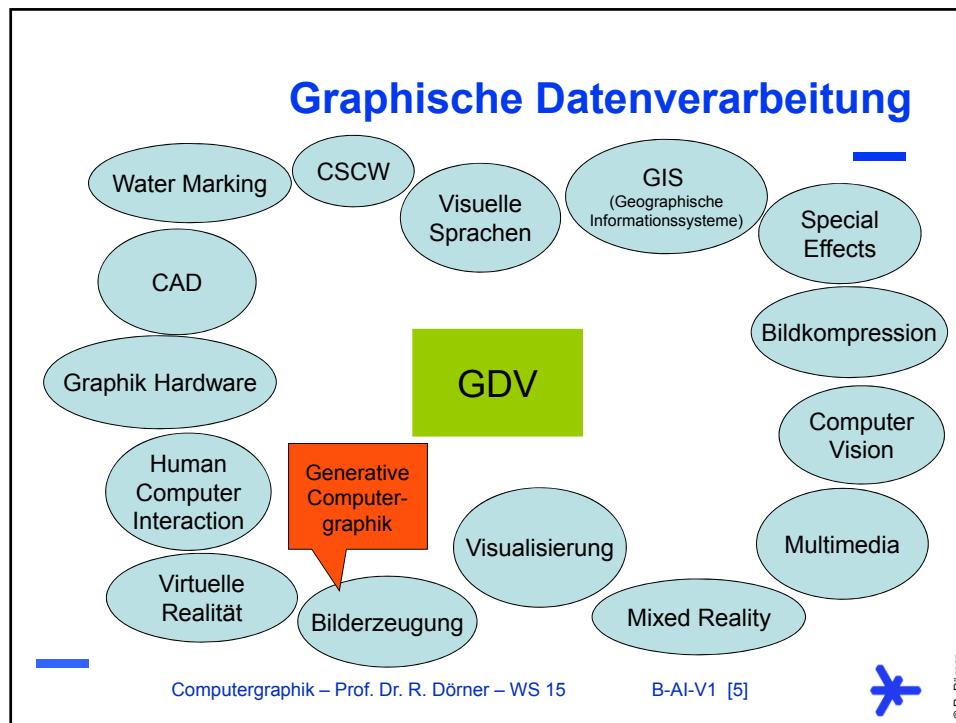
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [4]



© R. Dörner





## Computergenerierte Bilder



photorealistic

Quelle:  
Bob Hoffman,  
Digital Domain

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [7]



© R. Dörner

## Computergenerierte Bilder



non - photorealistic

Quelle:  
Bert Freudenberg,  
Maic Masuch,  
Thomas Strothotte,  
Uni Magdeburg

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [8]



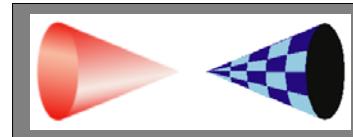
© R. Dörner



## Kernfrage



?



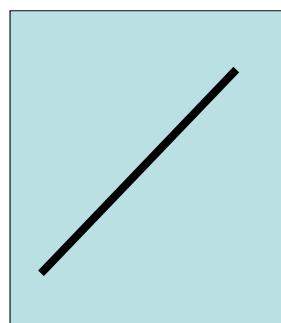
Wie Bild mit dem Computer erzeugen?

Was ist überhaupt ein Bild?

Wie wird ein Bild im Computer repräsentiert?

## Bildrepräsentation

Vektorgraphik



Speichern von:  
Koordinaten, Primitive

Pixelbild

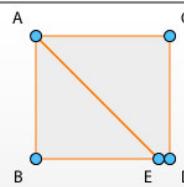


Speichern von:  
Matrix von Farbwerten

## Bildrepräsentation

### Vektorgrafik

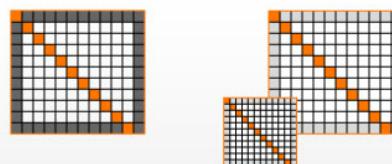
Zu speichern sind die Koordinaten von 5 Punkten und wie Punkte miteinander verbunden sind



Skalierung ist problemlos möglich

### Rastergrafik

Zu speichern ist der Wert von  $24 \times 24$  Pixel



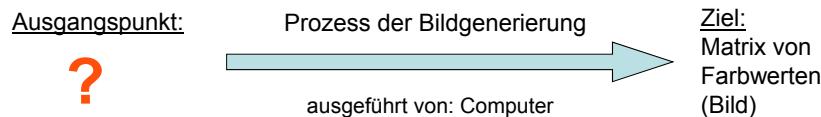
Bei Skalierung können Artefakte auftreten (z.B. Treppenstufen an Kanten)

## Bildrepräsentation

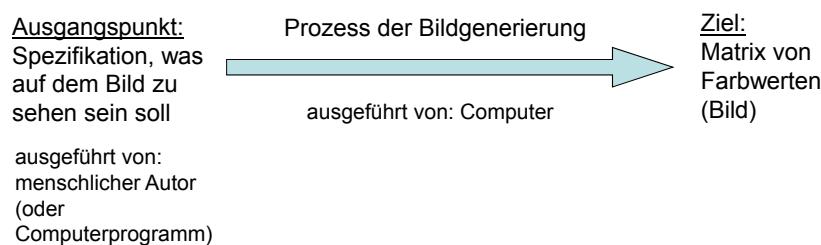
	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	(1,0,0)	
	(1,0,0)	(1,0,0)	(0,1,1)	(0,1,1)	(0,1,1)	(0,1,1)	
	(1,0,0)	(0,1,1)	(0,1,1)	(0,1,1)	(0,1,1)	(0,7,1,0)	
	(1,0,0)	(0,1,1)	(0,1,1)	(0,1,1)	(0,7,1,0)	(0,7,1,0)	

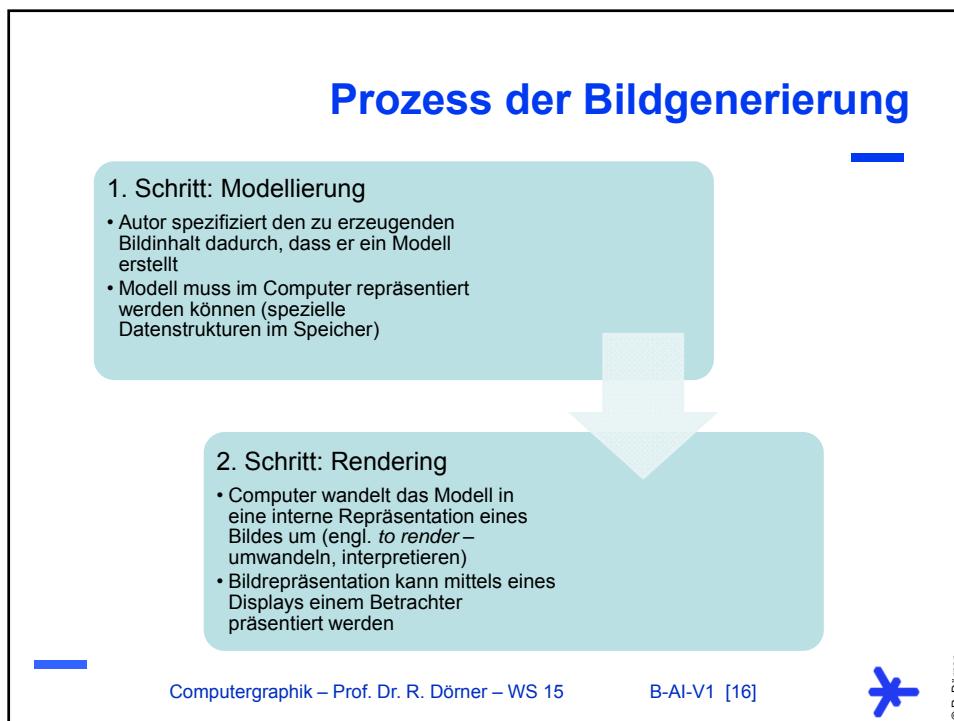
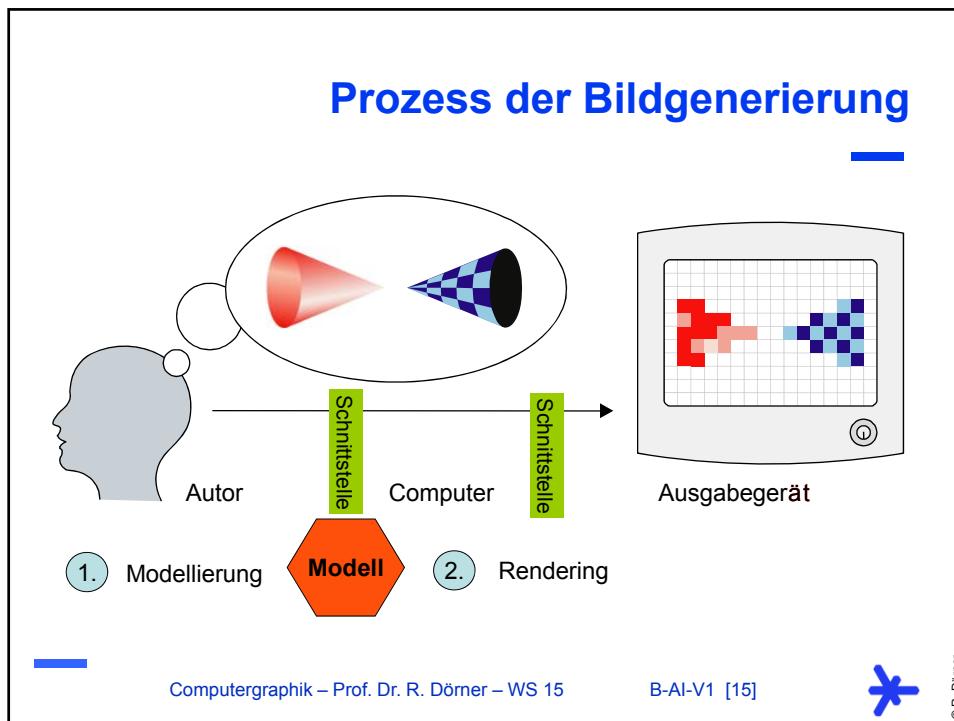
Repräsentation eines Bildes als eine Matrix von Farbwerten (z.B. RGB-Tripel)

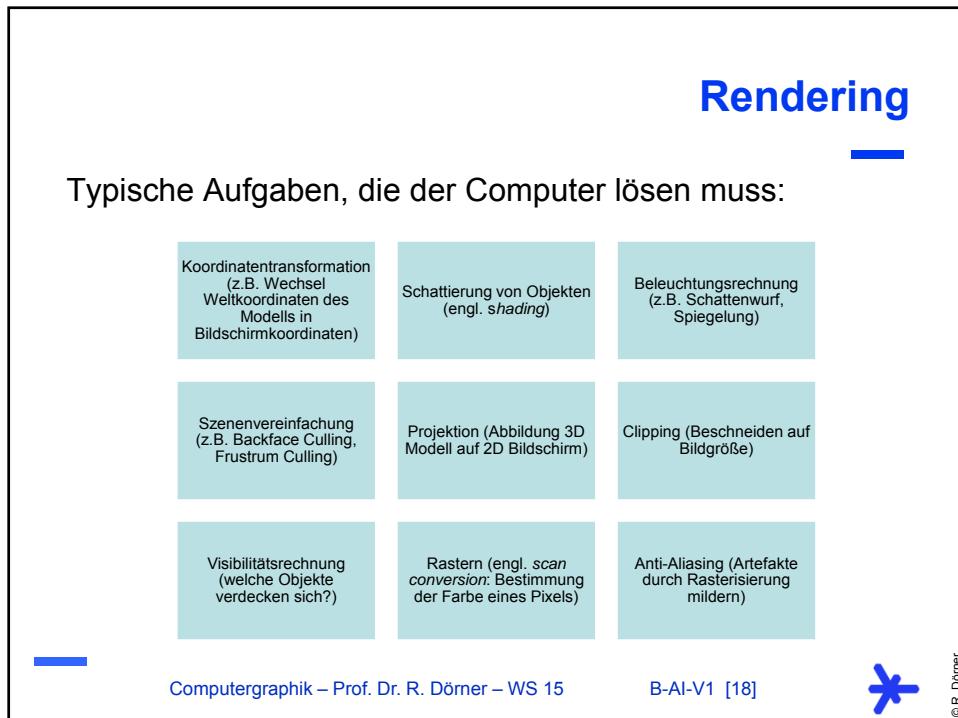
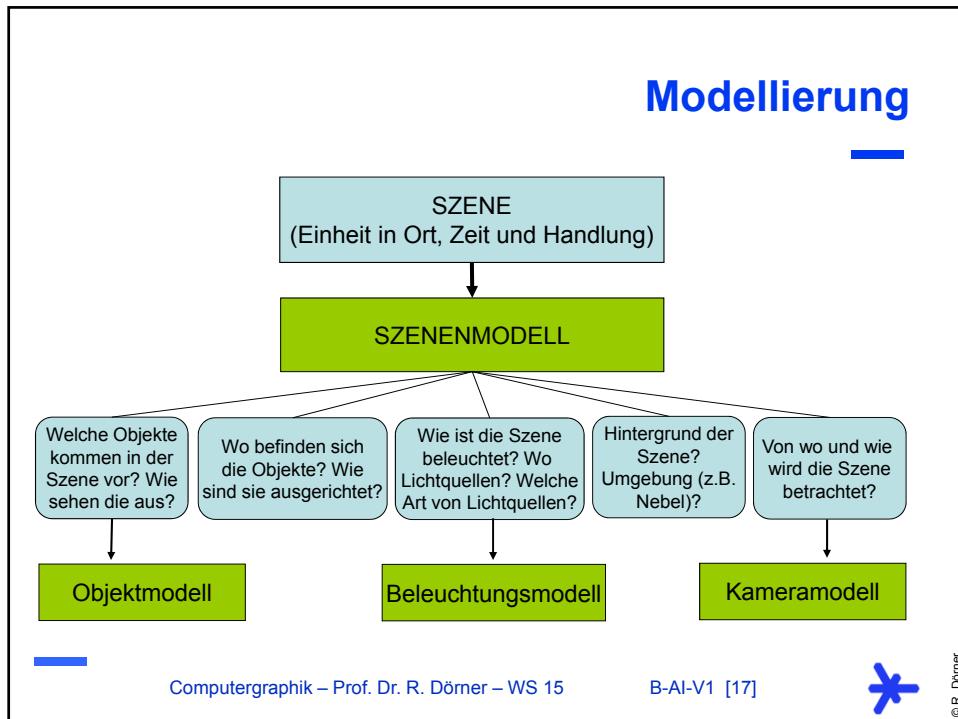
## Prozess der Bildgenerierung



## Prozess der Bildgenerierung







## Rendering

- In der GDV wurden eine ganze Reihe von Verfahren und Algorithmen entwickelt
  - für das Rendering als Ganzes
  - für einzelne Teilaufgaben des Renderings
- Häufig werden die Teilaufgaben nacheinander durchgeführt als sogenannte **Rendering-Pipeline**

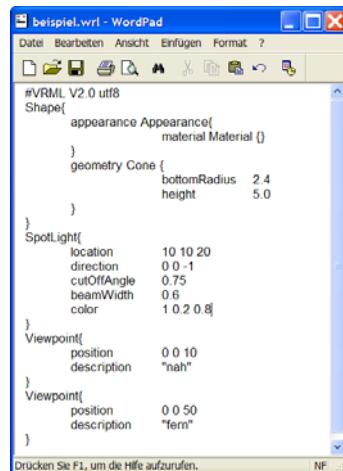


## Beispiel: VRML

- VRML (Virtual Reality Modeling Language)
- ISO Standard (ISO/IEC DIS 14772-1) seit 1997
- Darauf aufbauend: X3D ([www.web3d.org](http://www.web3d.org))
- Idee:
  - 3D Szenen werden als Text im VRML Format beschrieben
  - VRML Browser stellt 3D Szene dar



## Beispiel: VRML



```
#VRML V2.0 utf8
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry Cone {
        bottomRadius 2.4
        height 5.0
    }
}
SpotLight{
    location 10 10 20
    direction 0 0 -1
    cutOffAngle 0.75
    beamWidth 0.6
    color 1 0.2 0.8
}
Viewpoint{
    position 0 0 10
    description "nah"
}
Viewpoint{
    position 0 0 50
    description "fern"
}
```

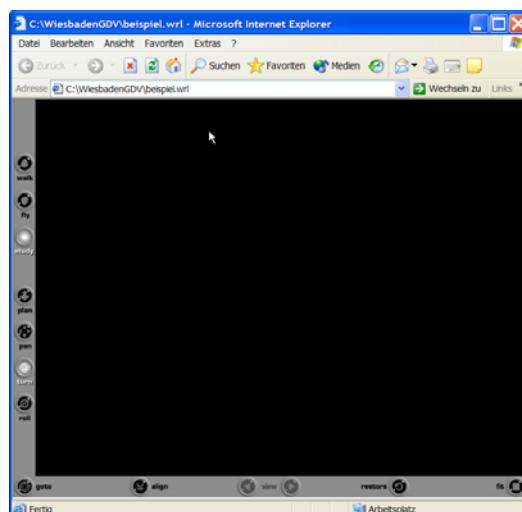
Szenenmodell als  
Textdatei im  
VRML Format

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [21]

© R. Dörner

## Beispiel: VRML



Darstellung des Szenen-  
modells in einem VRML  
Browser  
(hier: Cortona als IE-Explorer  
Plugin, siehe [www.cortona.com](http://www.cortona.com))

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [22]

© R. Dörner



## Beispiel: VRML

```
#VRML V2.0 utf8
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry Cone {
        bottomRadius 2.4
        height      5.0
    }
}
SpotLight{
    location     10 10 20
    direction    0 0 -1
    cutOffAngle 0.75
    beamWidth   0.6
    color        1 0.2 0.8
}
Viewpoint{
    position     0 0 10
    description  "nah"
}
Viewpoint{
    position     0 0 50
    description  "fern"
}
```

### Header

Objektmodell „Kegel“  
mit Standarderscheinung (Appearance)  
und Standardmaterial (Material)

Beleuchtungsmodell  
als Spotlight („Taschenlampenkegel“)

Kameramodell  
durch Definition zweier Blickpunkte  
(``Viewpoints``) bezeichnet als ``nah`` und  
``fern``

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [23]



© R. Dörner

## Beispiel: VRML

```
#VRML V2.0 utf8
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry Cone {
        bottomRadius 2.4
        height      5.0
    }
}
SpotLight{
    location     10 10 20
    direction    0 0 -1
    cutOffAngle 0.75
    beamWidth   0.6
    color        1 0.2 0.8
}
Viewpoint{
    position     0 0 10
    description  "nah"
}
Viewpoint{
    position     0 0 50
    description  "fern"
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [24]



© R. Dörner



## Beispiel: VRML

```
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry Cone {
        bottomRadius 2.4
        height 5.0
    }
}
```

The diagram illustrates the VRML scene graph structure. A central **Shape** node contains fields for **appearance** and **geometry**. The **appearance** field points to an **Appearance** node, which in turn points to a **Material** node. The **geometry** field points to a **Cone** node, which has attributes for **bottomRadius** (2.4) and **height** (5.0).

**Graphische Darstellung als Szenengraph**

Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [25]

© R. Dörner

## Beispiel: VRML

- In VRML wird eine Szene als Szenengraph modelliert
  - Knoten des Szenengraph heißen *Nodes*
  - Attribute in den Nodes heißen *Fields*
  - Kanten zwischen den Knoten heißen *Edges*
- Das Modell des Szenengraphen bedient sich selbst wiederum anderer Modelle, z.B. Beleuchtungsmodelle wie das Spotlight-Modell

The diagram shows the same VRML scene graph as above, but with labels for its components: **Node** (Shape), **Edge** (connections between nodes), and **Field** (the fields within the Shape node).

The diagram shows the same VRML scene graph as above, but without labels.

Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [26]

© R. Dörner



## Beispiel: VRML

```
#VRML V2.0 utf8
Transform{
    rotation      1 1 1 0.785
    children [
        Transform{
            rotation      0 0 1 0.785
            translation   1 0 -10
            children [
                Shape{
                    appearance Appearance{
                        material Material {}
                    }
                    geometry Sphere {
                        radius      2.4
                    }
                }
                Shape{
                    appearance Appearance{
                        material Material {}
                    }
                    geometry Box {
                        size       1 4 2
                    }
                }
            ]
        }
    ]
}
```

Aufgabe:  
Zeichnen Sie den  
Szenengraphen!

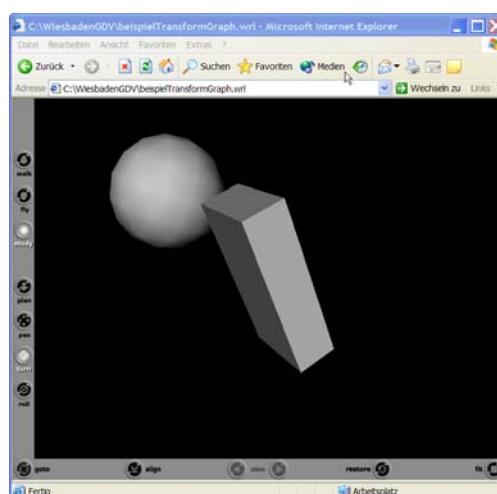
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [27]



© R. Dörner

## Beispiel: VRML



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [28]



© R. Dörner



## Beispiel: VRML

Prozess der Bildgenerierung: Teil 1

Im Kopf des Autors entsteht die Idee und die Vorstellung, was auf dem Bild zu sehen sein soll



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [29]

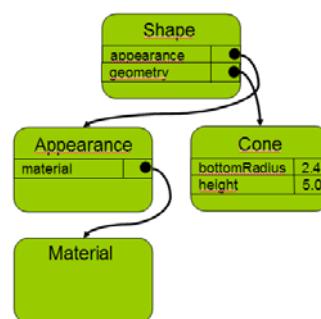


© R. Dörner

## Beispiel: VRML

Prozess der Bildgenerierung: Teil 2

Der Autor beschreibt seine Vorstellung explizit mit Hilfe von Modellen. Ziel ist es, eine Szene zu modellieren. Dazu bietet VRML das Modell des Szenen-graphen an und auch andere Modelle, wie z.B. das Modell eines Lichtkegels (Spotlight) als Beleuchtungsmodell.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [30]



© R. Dörner



## Beispiel: VRML

### Prozess der Bildgenerierung: Teil 3

Der Autor notiert das Szenenmodell als Textdatei, die einem bestimmten Format genügen muss, das der VRML Standard festlegt (formale Sprache wie eine Programmiersprache).



```
#VRML V2.0 utf8
Shape{ appearance Appearance{
material Material []
}
geometry Cone {
bottomRadius 2.4
height 5.0
}
}
SpotLight{
location 10 10 20
direction 0 0 -1
cutoffAngle 0.75
beamWidth 0.6
color 1 0.2 0.8
}
Viewpoint{
position 0 0 10
description "nah"
}
Viewpoint{
position 0 0 50
description "fern"
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [31]

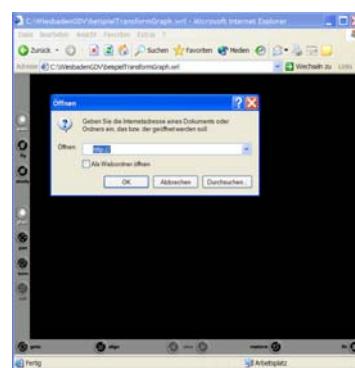


© R. Dörner

## Beispiel: VRML

### Prozess der Bildgenerierung: Teil 4

In dem VRML-Browser (d.h. GDV Softwaressystem) wird die Textdatei eingelesen (Parsing). Dabei wird eine interne Repräsentation des Szenenmodells im Speicher des Computers aufgebaut, z.B. könnten Nodes als Objekte im Speicher vorliegen und Fields als Attribute dieser Objekte.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [32]



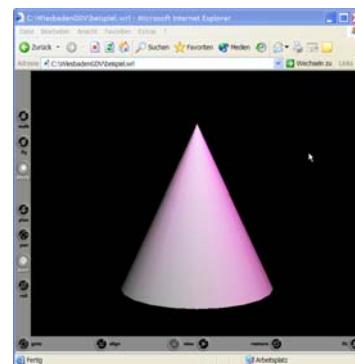
© R. Dörner



## Beispiel: VRML

### Prozess der Bildgenerierung: Teil 5

In dem VRML-Browser (d.h. GDV Softwaresystem) wird die interne Repräsentation des Szenenmodells im Rendering in die interne Repräsentation eines Bildes umgewandelt (z.B. als zweidimensionales Array). Diese interne Repräsentation des Bildes wird dann entsprechend in den Bildspeicher der Graphikkarte geladen und auf einem Display (z.B. Computermonitor) ausgegeben.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [33]



© R. Dörner

## Vokabeln

- GDV
- CG
- GIS
- Human Computer Interaction
- Generative CG
- Computeranimation
- Vektorgraphik
- Pixelgraphik
- Non-photorealistic
- Modellierung
- Szene
- Szenenmodell, Objektmodell, Beleuchtungsmodell, Kameramodell
- Szenengraph
- Rendering
- Shading
- Clipping
- Scan Conversion
- Rendering Pipeline
- VRML

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [34]



© R. Dörner



## Ziele der Lehrveranstaltung

- Kennen lernen der Grundlagen der Computergraphik
  - Modellierung, geometrische Transformationen, Beleuchtung, Texturierung, Verdeckungen, ...
  - Erste Erfahrungen in der Erstellung von Computergraphik Szenen (VRML) und Software (OpenGL)
- aber auch über die Computergraphik hinaus:
  - mathematisch exakte Beschreibung von Modellen in der Informatik, Mensch-Maschine-Schnittstelle, u.v.m.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [35]



© R. Dörner

## Computergraphik

- A. Einführung
- B. Szenengraphen und Koordinatensysteme
- C. Kameramodell
- D. Beleuchtungsmodell
- E. Szenenmodell
- F. Objektmodelle
- G. Rendering und OpenGL
- H. Vertex Operationen
- I. Culling, Clipping und Rasterisierung
- J. Fragment Operationen
- K. GDV Anwendungen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [36]



© R. Dörner



## Organisatorisches

- Praktikum und Zeiten
- Pünktlichkeit und Ruhe
- Leistungsnachweis

Aktuelle Infos, Skript, etc. unter  
[~doerner/public\\_r/gdv2015](http://~doerner/public_r/gdv2015)



## Stellenwert Vorlesung und Praktikum



- Einfach nur in Vorlesung und Praktikum gehen ist NICHT genug (genauso hilfreich wie Handauflegen)
- Studieren heißt lernen (= arbeiten), auseinandersetzen (AKTIV!), ist leider mühsam und kostet Zeit
- Praktikum ist wichtig, Aufgaben vor dem Praktikumstermin (am besten in einer Lerngruppe) vorbereiten – in der Praktikumsstunde werden Aufgaben nur besprochen



## Zeitaufwand der Lehrveranstaltung



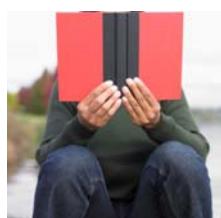
- Im Semester: 30 CP = 900 h Zeitaufwand
- bei 15 Vorlesungswochen: 60 h / Woche (ist sehr viel!)
- Computergraphik hat 5 CP also 150 h
- Zeit der Lehrveranstaltungen selbst (Vorlesung und Praktikumszeiten im Semester): 42 h
- Problem: keine 42 Stunden Leistung für 150 Stunden anerkennbar
- Notwendigkeit: Hausaufgaben (ist auch sehr sinnvoll), Zeit in Stundenplan eintragen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [39]



## Hausaufgaben: Vor- und Nachbereitung



- Jede Woche gibt es folgende Hausaufgabe:
  - Stoff der Vorlesung nachbereiten – allein oder in Lerngruppe (z.B. Nachlesen in Büchern)
  - Es sollte nichts unklar bleiben – man kommt sonst in der nächsten Vorlesung nicht gut mit und verschwendet Zeit
  - Praktikumsaufgaben machen und so Praktikumsstunde vorbereiten – ansonsten bringt Teilnahme am Praktikum nicht viel und ist Zeitverschwendug
- Zeit dafür einplanen!
- Selbstdisziplin – Sie sind Studenten, keine Schüler: keine Kontrolle durch Hochschule (außer ganz am Schluss in der Klausur)

Computergraphik – Prof. Dr. R. Dörner – WS 15

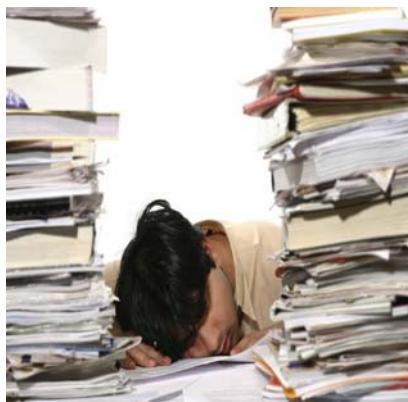
B-AI-V1 [40]



© R. Dörner



## Richtig Lernen!



- Lernen erst zur Klausurzeit ist zu spät – zuviel Stoff und Lernen braucht Zeit
- Strategie: Reduktion auf alte Klausuraufgaben funktioniert in der Computergraphik nicht – jedesmal völlig neue Aufgaben in der Klausur
- Ziel: Computergraphik lernen und nicht Klausuraufgaben lernen
- Kontinuierlich mitmachen, am Ball bleiben

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [41]



## Akademische Ehrlichkeit



- Alles, was zur Benotung abgegeben wird, muss selbst erstellt worden sein
  - alle Hilfen von anderen müssen angegeben werden
  - alle Texte, Bilder, Code-Teile, die man übernommen hat, müssen als solche gekennzeichnet und mit Quellenangabe versehen sein
- Arbeiten müssen eigenständig (allein) gemacht werden, außer es ist explizit Gruppenarbeit gefordert
- Anderen Einzelpersonen / Gruppen sollte der eigene Code / Text nicht gezeigt werden, niemals sollte für andere programmiert werden.

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [42]



© R. Dörner



## Leistungsnachweis



- Vorlesung
  - Klausur
- Praktikum
  - (Klausur)
  - Online-Tests
  - Projektabgabe
  - Mindestpunktzahl in den einzelnen Teilen

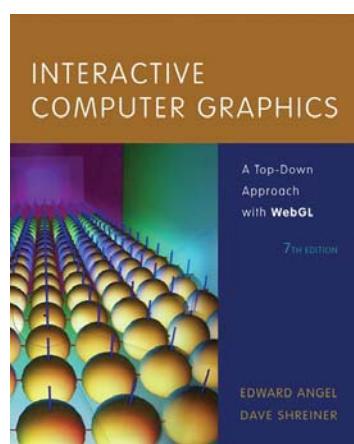
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [43]



© R. Dörner

## Empfohlene Literatur



Edward Angel, Dave Shreiner:  
Interactive Computer Graphics,  
7.te Auflage, Addison-Wesley,  
2014

*Buch, das detailliert und schrittweise vorgeht.  
Programmierung mit OpenGL / WebGL wird gleichzeitig mit den Konzepten eingeführt.  
Empfohlen zur Vorlesungsbegleitung.*

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [44]



© R. Dörner



## Empfohlene Literatur



A. Nischwitz et. al:  
Computergrafik und  
Bildverarbeitung,  
2.te Auflage, Vieweg, 2007

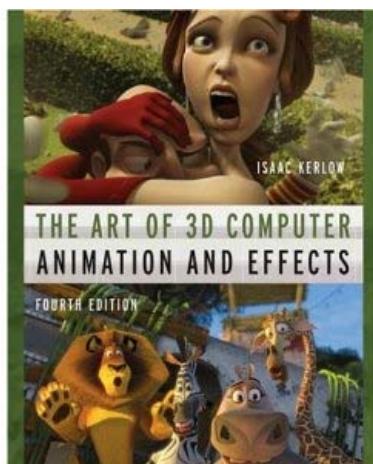
*Deutschsprachig. Illustriert  
theoretische Grundlagen an  
OpenGL. Eine Hälfte des  
Buches beschäftigt sich mit  
Bildverarbeitung*

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [45]



## Weitere Literatur



Isaac Kerlow:  
The Art of 3D Computer  
Animation and Effects,  
John Wiley & Sons, 2009

*Visuell sehr attraktives Buch,  
das eine interessante  
Einführung in die 3D  
Produktion und einen Blick  
hinter die Kulissen gibt. Sehr  
ansprechend und verständlich.*

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [46]

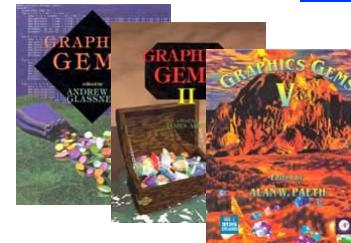


© R. Dörner

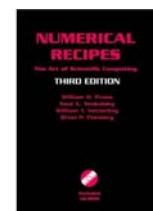


## Weitere nützliche Bücher

- Graphics Gems  
Sammlung von Algorithmen für verschiedene Einsatzzwecke



- Numerical Recipes in C / C++  
Sammlung von „Rezepten“ zur Implementierung mathematischer Operationen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [47]



© R. Dörner

## Forschungsrelevante Literatur



- Konferenzen
  - SIGGRAPH Konferenz und Workshops ([www.siggraph.org](http://www.siggraph.org))
  - Eurographics Konferenz und Workshops ([www.eg.org](http://www.eg.org))
- Zeitschriften
  - IEEE Transactions on Visualization and Computer Graphics
  - ACM Transactions on Graphics
- Online Ressourcen
  - [portal.acm.org](http://portal.acm.org)
  - [www.citeseer.org](http://www.citeseer.org)
  - [ieeexplore.ieee.org](http://ieeexplore.ieee.org)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [48]



© R. Dörner



## Acknowledgements

- Dank an Jan-Christoph Sievers für die Erstellung von Abbildungen
- Dank an Christoph Schulz für Anregungen und Feedback



## Teil B: Szenengraphen und Koordinatensysteme



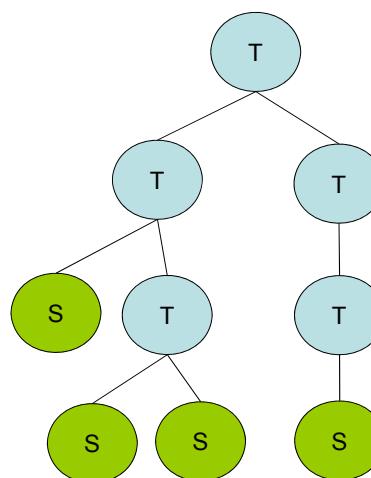
## Szenenraphen und Koordinatensysteme

- B.1 Shape-Nodes
- B.2 Transform-Nodes und Objekthierarchien
- B.3 Transformationsmatrizen
- B.4 Wechsel von Koordinatensystemen



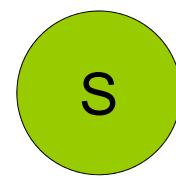
## Szenenraph

- Wichtige Datenstruktur in der Computergraphik
- Haben bisher Beispiel VRML gesehen
- Weitere Vertreter:
  - Open Inventor
  - OpenScenegraph
  - Coin3D



## Shape-Node

- Objektmodelle stehen in VRML unter dem Shape-Node
- Zwei Felder:
  - **geometry** (Beschreibung der Form)
  - **appearance** (Beschreibung des Aussehens)
- Hinweis: als Wert vom **appearance** Feld immer einen **Appearance – Node** (dessen **material** Feld mit einem **Material – Node** belegt ist) angeben, sonst wird das Objektmodell nicht richtig dargestellt



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [53]



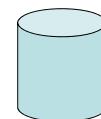
© R. Dörner

## Grundprimitive

- VRML kennt eine Reihe von sog. Grundprimitiven:
  - Kugel (**sphere**)
  - Kegel (**Cone**)
  - Zylinder (**Cylinder**)
  - Quader (**Box**)
- Jedes Grundprimitiv wird durch einen Node repräsentiert

```
Sphere{  
    radius  
}
```

4



```
Cylinder{  
    bottom  
    side  
    top  
    height  
    radius  
}  
    FALSE  
    TRUE  
    TRUE  
    4  
    2
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [54]

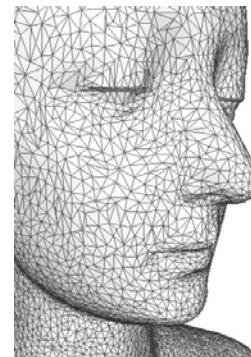


© R. Dörner



## Polygonnetze

- Frage: Komplexere Objektmodelle mit Grundprimitiven?
- In VRML zusätzlich: Polygonnetze
- Name des Nodes:  
**IndexedFaceSet**



Computergraphik – Prof. Dr. R. Dörner – WS 15

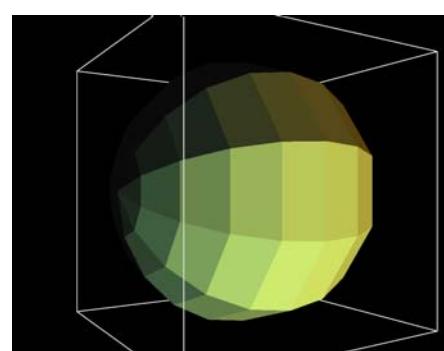
B-AI-V1 [55]



© R. Dörner

## Polygonnetze

- Grundlage: Polygonflächen
- Kombination von Polygonflächen zu einem dreidimensionalen Netz
- Mehrfachverwendung von Eckpunkten



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [56]

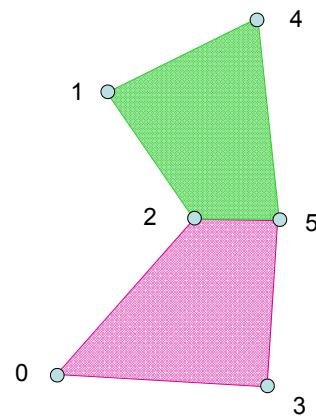


© R. Dörner



## IndexedFaceSet

- Angabe einer Punktliste:
  - Koordinaten von jedem Punkt
  - Koordinaten durch Komma getrennt
- Angabe einer Flächenliste:
  - Zu jeder Fläche werden die Eckpunkte (genauer: Index der Eckpunkte in der Punktliste) der Reihenfolge nach angegeben
  - Index beginnt bei 0
  - Abschluss mit -1: Verbindung zum ersten Punkt der Fläche



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [57]



© R. Dörner

## IndexedFaceSet

```
#VRML V2.0 utf8
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry IndexedFaceSet{
        coord Coordinate{
            point [
                0 0 1,
                1.5 0 0,
                0 3.5 0,
                0 0 0
            ]
        }
        coordIndex [
            0 1 2 -1 3 0 2 -1
            2 1 3 -1 3 1 0
        ]
    }
}
```

Punktliste

Flächenliste

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [58]

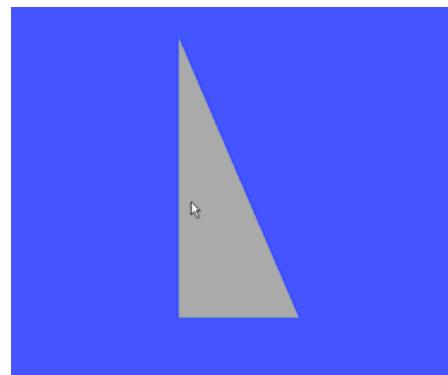


© R. Dörner



## Polygonnetze

Beispiel: Verwendung von **Polygonnetzen** in VRML



Computergraphik – Prof. Dr. R. Dörner – WS 15

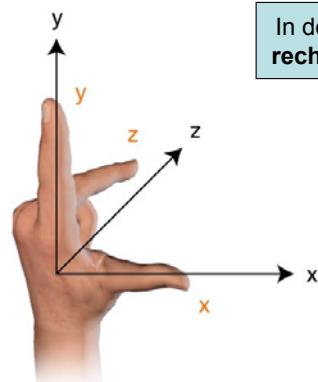
B-AI-V1 [59]



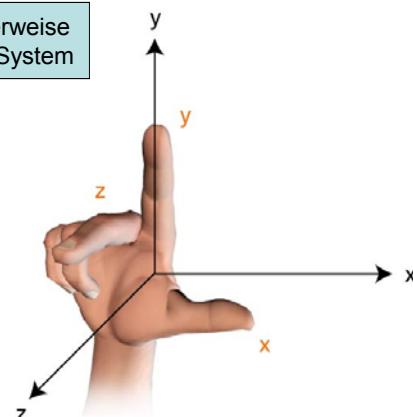
© R. Dörner

## Koordinatensysteme

In der CG üblicherweise  
**rechtshändiges** System



linkshändiges Koordinatensystem



rechtshändiges Koordinatensystem

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [60]



© R. Dörner



## IndexedFaceSet

- Vorsicht: Koordinaten in rechtshändigem Koordinatensystem angeben
- Vorsicht: Punkte in der richtigen Reihenfolge angeben
  - Jede Fläche hat eine Vorderseite und eine Rückseite (i.d.R. wird nur Vorderseite gezeichnet)
  - Unterscheidung durch die Flächennormale (Vektor, der senkrecht auf der Fläche steht)
  - In VRML: Angabe der Punkte gegen den Uhrzeigersinn („Fläche liegt links“)



Computergraphik – Prof. Dr. R. Dörner – WS 15

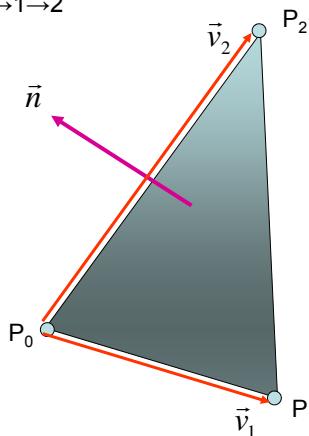
B-AI-V1 [61]



© R. Dörner

## Innenseite und Außenseite

außen:  
 $0 \rightarrow 1 \rightarrow 2$



$$\vec{v}_1 \times \vec{v}_2 = (\vec{p}_1 - \vec{p}_0) \times (\vec{p}_2 - \vec{p}_0) = \vec{n}$$

mit  $|\vec{n}| \geq 0$ , d.h. die Vektoren  $\vec{v}_1, \vec{v}_2, \vec{n}$  bilden ein Rechtssystem

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [62]

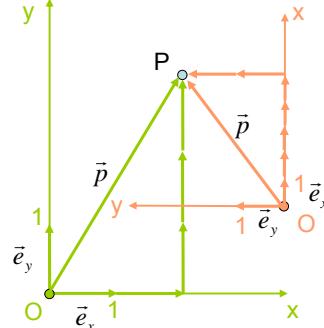


© R. Dörner



## Koordinaten von Punkten

- Zur Beschreibung eines Polygonnetzes müssen wir die Lage der Eckpunkte angeben
- Die Lage eines Eckpunktes wird durch Koordinaten angegeben
- Ein und derselbe Punkt kann mehrere Koordinaten haben – je nach Wahl des Koordinatensystems



$$\vec{p} = 2 \cdot \vec{e}_x + 3 \cdot \vec{e}_y \Rightarrow P(2/3)$$

$$\vec{p} = 5 \cdot \vec{e}_x + 2 \cdot \vec{e}_y \Rightarrow P(5/2)$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [63]



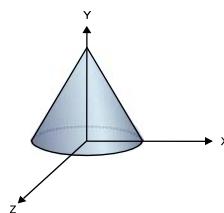
© R. Dörner

## Wahl des lokalen Koordinatensystems

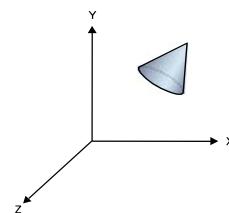
Frage: In welchem der vielen möglichen Koordinatensystemen geben wir die Koordinaten der Eckpunkte (als Zahlenwerte) an?

Antwort: Eigentlich egal, aber sinnvoll ist: In einem Koordinatensystem, in dem man die Werte günstig ermitteln kann

Objektkoordinaten  
Lokale Koordinaten



Sinnvolle Wahl



Unsinnige Wahl

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [64]



© R. Dörner



## Szenenraphen und Koordinatensysteme

B.1 Shape-Nodes

### B.2 Transform-Nodes und Objekthierarchien

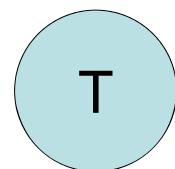
B.3 Transformationsmatrizen

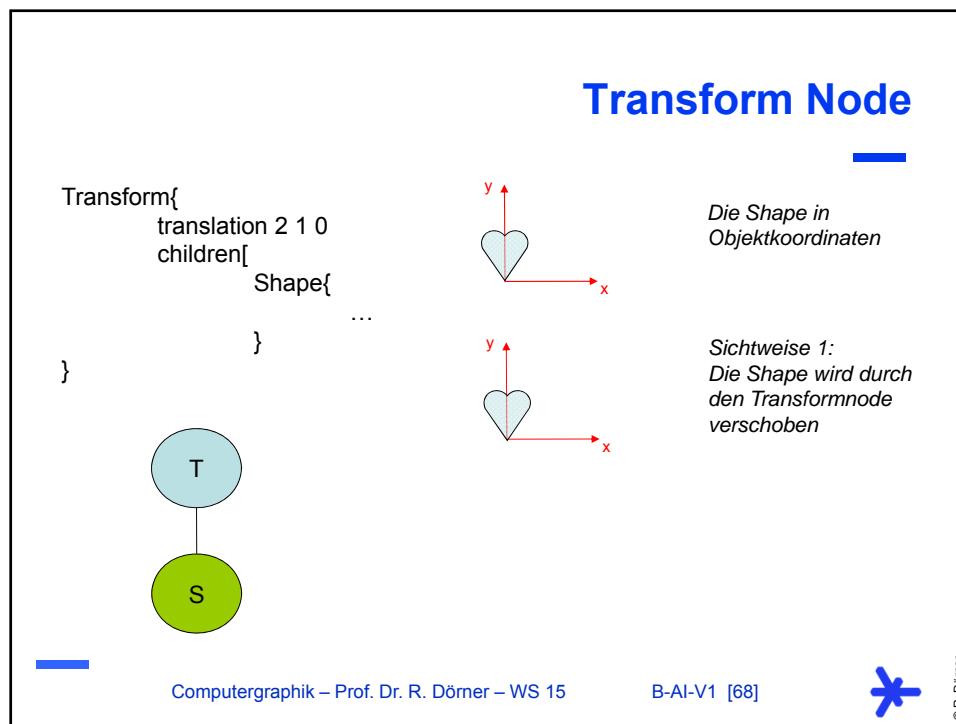
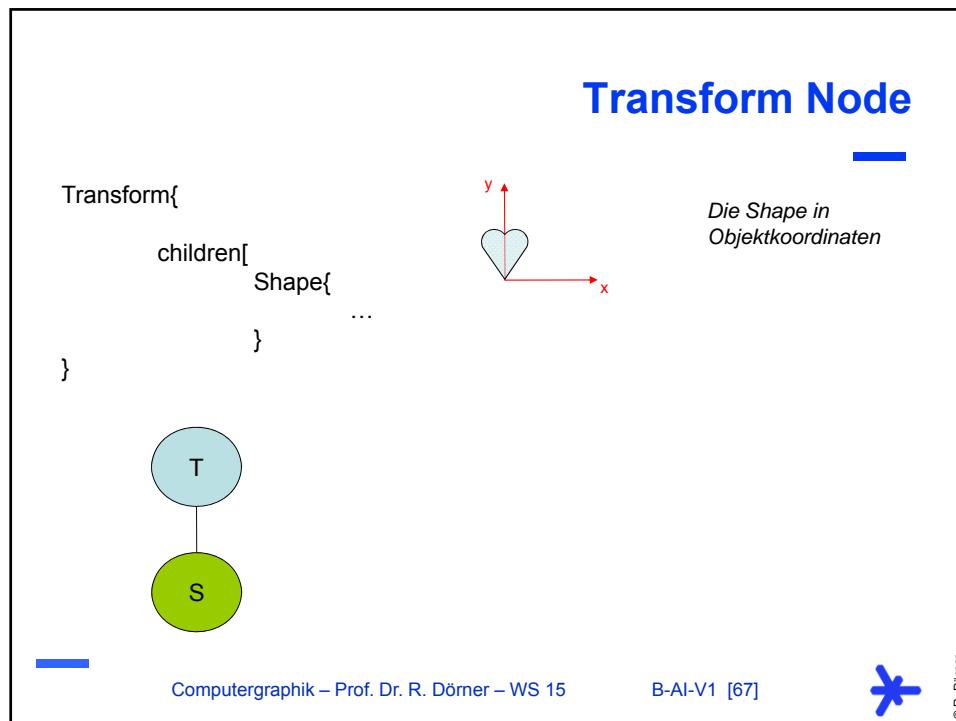
B.4 Wechsel von Koordinatensystemen

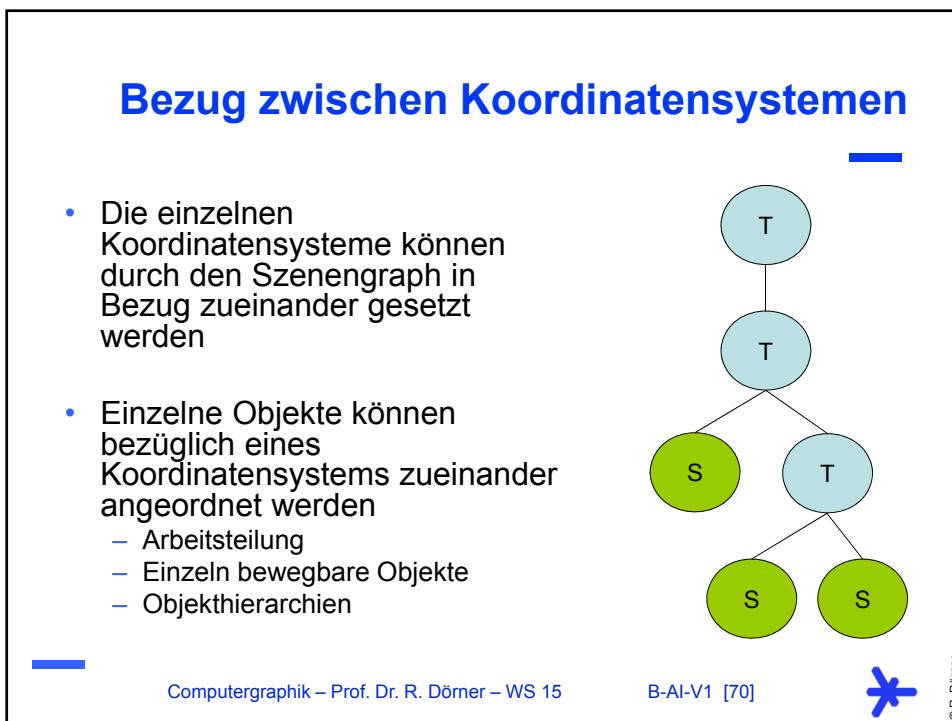
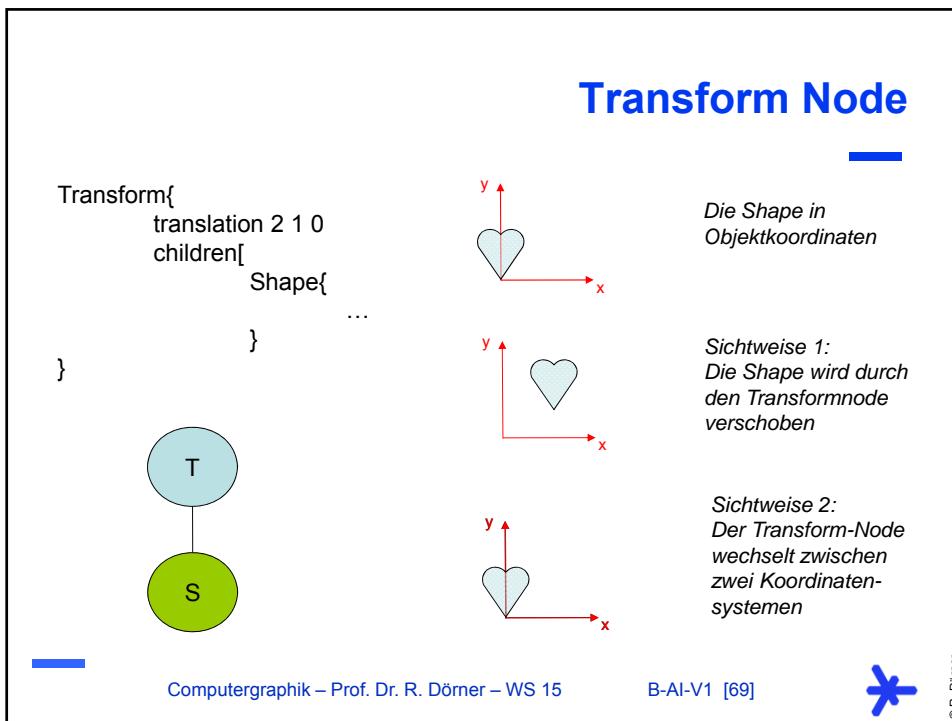


## Transform - Node

- Shape Nodes sind Blätter des Szenenraphen
- Transform Nodes sind innere Knoten des Szenenraphen
- Transform Nodes dienen dazu, Objekte zu platzieren, zu rotieren, ihre Größe zu ändern. Außerdem können Objekte kombiniert werden.







## Beispiel

Objektmodell „Gesicht“      Objektmodell „Körper“

Szenengraph

Jedes Objektmodell ist in eigenem Koordinatensystem definiert: lokales Koordinatensystem

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [71]

© R. Dörner

## Beispiel

Mit dem Transform-Node wird der Shape-Node „Gesicht“ im lokalen Koordinatensystem von Shape „Körper“ eingefügt – und zwar genau so, wie „Gesicht“ in seinem lokalen Koordinatensystem definiert wurde.

Szenengraph

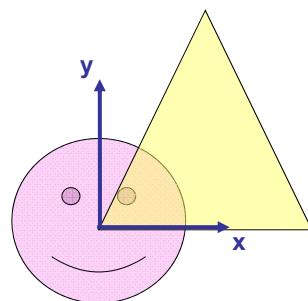
Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [72]

© R. Dörner



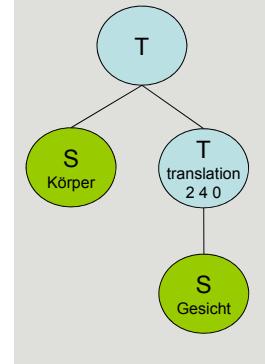
## Beispiel

Durch Veränderung des translation - Felds im Transform Node werden alle Shapes unterhalb des Transform Nodes verschoben.



Computergraphik – Prof. Dr. R. Dörner – WS 15

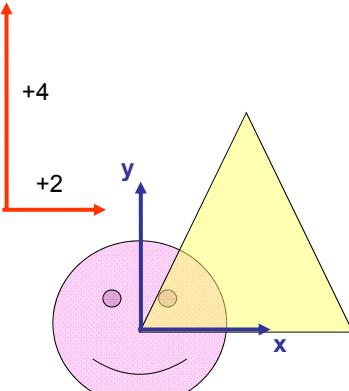
Szenengraph



B-AI-V1 [73]



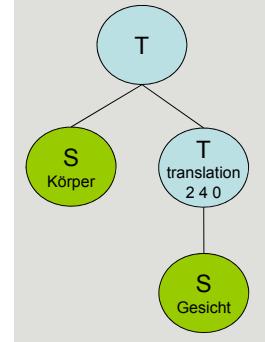
© R. Dörner



Computergraphik – Prof. Dr. R. Dörner – WS 15

## Beispiel

Szenengraph



B-AI-V1 [74]



© R. Dörner



## Beispiel

Szenengraph

```
graph TD; T1[T] --- S1[S Körper]; T1 --- T2["T translation 2 4 0"]; T2 --- S2[S Gesicht]
```

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [75]

© R. Dörner

## Weltkoordinaten

- Koordinatensystem an der Wurzel heißt Weltkoordinatensystem oder Model-Koordinatensystem
- Alle anderen Koordinatensysteme haben einen spezifizierten Bezug zu den Weltkoordinaten
- Umwandlung: Welt in Objektkoordinaten durch Pfad Blatt -> Wurzel gegeben

Szenengraph

```
graph TD; T1[T] --- S1[S Körper]; T1 --- T2["T translation 2 4 0"]; T2 --- S2[S Gesicht]
```

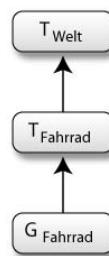
Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [76]

© R. Dörner

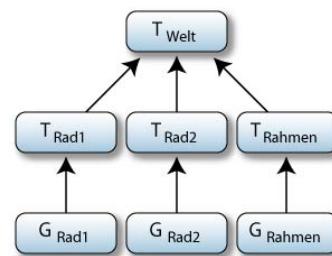


## Objekthierarchie im Szenengraphen

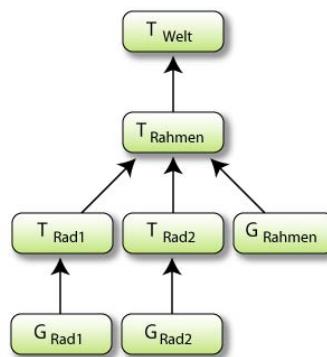
Variante ①



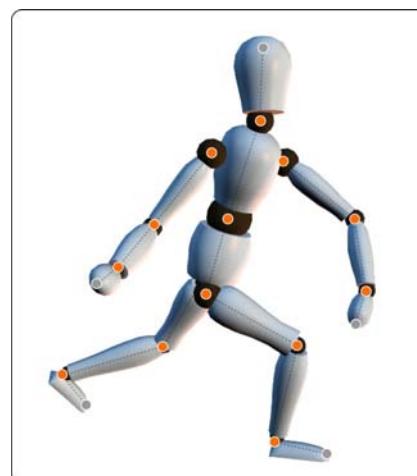
Variante ②



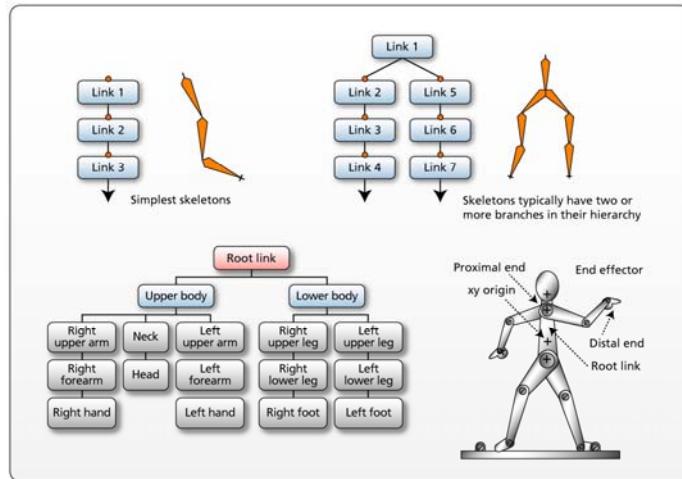
Variante ③



## Objekthierarchie im Szenengraphen



## Objekthierarchie im Szenengraphen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [79]

© R. Dörner

## Geometrische Operationen im Transform Node

- Verschiebung / Translation
  - Schlüsselwort: translation
  - Parameter: Verschiebungsvektor ( $t_x, t_y, t_z$ )
- Drehung / Rotation
  - Schlüsselwort: rotation
  - Parameter: Drehwinkel, Drehachse, Drehpunkt (center)
- Skalierung
  - Schlüsselwort: scale
  - Parameter: Skalierungswerte ( $s_x, s_y, s_z$ ), Skalierungsorientierung

scaleOrientation  $R_S$  wird benötigt, falls man entlang einer Achse A skalieren möchte, die nicht die x-Achse, y-Achse oder z-Achse ist: durch  $R_S$  wird A auf die x-Achse, y-Achse oder z-Achse gedreht

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [80]

© R. Dörner



## Beispiel: VRML

```
Transform{  
    children[  
        Shape{      hier den „Körper“      }  
        Transform{  
            translation      2   -1.3   5  
            rotation         0   0   1   3.14  
            center          2   3   -5.1  
            scale            0.5   1   3  
            scaleOrientation 1   2   1   1.57  
            children[  
                Shape{      hier das „Gesicht“      }  
            ]  
        ]  
    ]  
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [81]

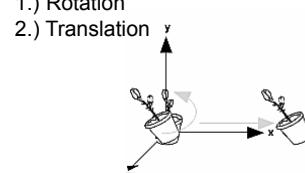


© R. Dörner

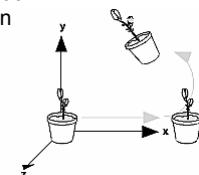
## Beispiel: VRML

- Vorsicht: Reihenfolge der Transformationen ist nicht gleichgültig
- In VRML ist die Reihenfolge im Transform-Node festgelegt:
  1. Skalierung
  2. Rotation
  3. Translation
- Will man andere Reihenfolge: mehrere Transform-Nodes direkt hintereinander

Quelle: giprogramming.com



1.) Translation  
2.) Rotation



1.) Rotation  
2.) Translation

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [82]

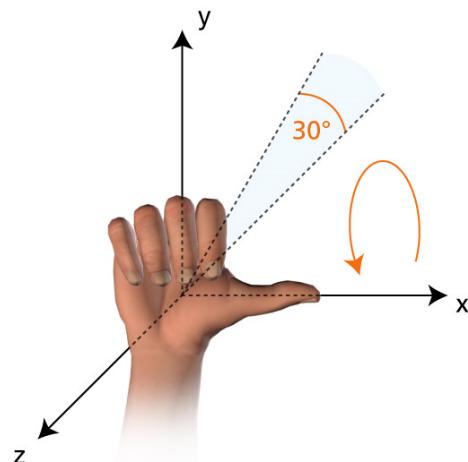


© R. Dörner



## Vorzeichen des Drehwinkels

- Rechte Hand nehmen (bei rechtshändigem Koordinatensystem)
- Daumen in Richtung der Drehachse
- Gekrümmte Finger geben positive Drehrichtung an



## Szenengraphen und Koordinatensysteme

- B.1 Shape-Nodes
- B.2 Transform-Nodes und Objekthierarchien
- B.3 Transformationsmatrizen**
- B.4 Wechsel von Koordinatensystemen

## Motivation

- Wollen berechnen, wohin Punkte nach Rotationen, Translationen etc. abgebildet werden
- Wollen die Koordinaten desselben Punktes in verschiedenen Koordinatensystemen ineinander umrechnen
- In der CG ist raffinierte Mathematik basierend auf Matrizen entwickelt worden
  - Umsetzung in den Grafik APIs
  - Umsetzung direkt in Grafik Hardware
- Kenntnis dieser Mathematik ist wichtig, u.a.
  - Berechnungen selbst durchführen
  - CG Bücher und Algorithmen verstehen



© R. Dörner

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [85]



## Eine einzige Formel

Im Kern steht die Verwendung von Transformationsmatrizen:

$$\vec{p}' = M \cdot \vec{p}$$

$\vec{p}$  Koordinaten des Punktes P vor der Transformation

$M$  Transformationsmatrix

$\vec{p}'$  Koordinaten des Punktes P', der durch Transformation von P entsteht

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [86]



© R. Dörner



## Transformationsmatrizen

- Jede Transformation (z.B. Rotation, Skalierung, Verschiebung, Spiegelung) wird durch eine Matrix dargestellt
- Wie sehen diese Matrizen aus?
- Idee: Formeln für Matrizen für einige einfache Standardfälle ermitteln, komplexere Transformationen auf Einfachere zurückführen

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

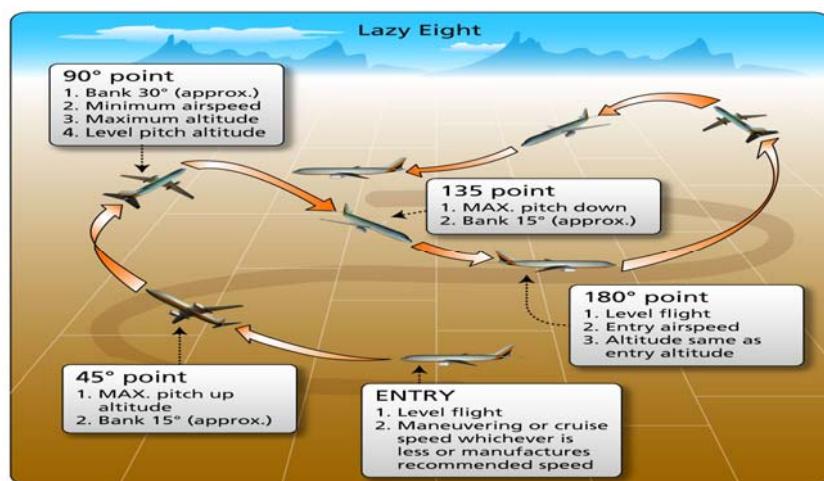
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [87]



© R. Dörner

## Zerlegung von komplexen Transformationen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [88]



© R. Dörner



## Zerlegung von komplexen Transformationen

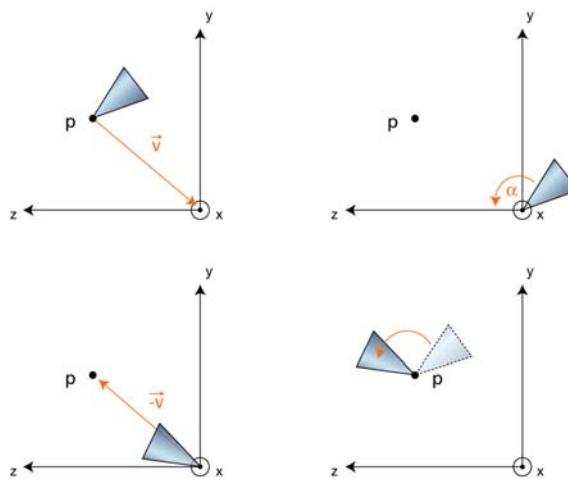
- Komplexe Transformation:  
Objekt drehen und dabei verschieben und verkleinern
- Zerlegung:
  1. Skalierung (in x-, y- oder z-Richtung)
  2. Drehung (um x-, y- oder z-Achse um den Ursprung)
  3. Verschiebung

Jede affine Transformation lässt sich in Skalierung,  
Verschiebung und Rotation zerlegen



## Zerlegung von komplexen Transformationen

- Komplexe Transformation:  
Drehung um einen beliebigen Punkt P
- Zerlegung:
  1. Verschiebung in Punkt P
  2. Drehung um den Ursprung
  3. Rückverschiebung



## Zerlegung von komplexen Transformationen

- Komplexe Transformation:  
Drehung um eine beliebige Achse um Winkel  $\phi$
- Zerlegung:
  1. Drehung um die x-Achse um Winkel  $\alpha$
  2. Drehung um die y-Achse um Winkel  $\beta$
  3. Drehung um die z-Achse um Winkel  $\gamma$
- Die Winkel  $(\alpha, \beta, \gamma)$  heißen Euler-Winkel



## Zerlegung von komplexen Transformationen

- Komplexe Transformation:  
Skalierung entlang einer beliebigen Achse
- Zerlegung:
  1. Transformiere so, dass beliebige Achse auf x-Achse liegt
  2. Skaliere entlang der x-Achse
  3. Mache Transformation von 1. wieder rückgängig



## Formel für Hintereinanderausführung von Transformationen

Wird eine Transformation  $M$  in drei Transformationen  $M_1$ ,  $M_2$  und  $M_3$  zerlegt, so lautet die Formel für die Hintereinanderausführung (Konkatenation) wie folgt:

$$\vec{p}' = M \cdot \vec{p} = M_3 \cdot M_2 \cdot M_1 \cdot \vec{p}$$

Allgemein:  $\vec{p}' = M_n \cdot \dots \cdot M_2 \cdot M_1 \cdot \vec{p}$



## Konkatenation von Transformationen

$$\vec{p}' = \underbrace{M_n \cdot \dots \cdot M_2 \cdot M_1}_M \cdot \vec{p}$$

Beobachtung 1:  
Matrix-Multiplikation ist nicht kommutativ, die Reihenfolge ist also wichtig

Beobachtung 2:  
Die Transformation, die ganz rechts steht, wird zuerst ausgeführt (entgegen Leserichtung)

Beobachtung 3:  
Die Matrizen können unabhängig vom Punkt aufmultipliziert werden und das Resultat dann für beliebig viele gleich zu transformierende Punkte genutzt werden (enorme Rechenzeiteinsparnis)



## Standardtransformationen

- Jede affine Transformation in 3D auf folgende **fünf** Standardtransformationen zurückführen:
  - Translation um einen Verschiebungsvektor
  - Rotation um die x-Achse um den Ursprung
  - Rotation um die y-Achse um den Ursprung
  - Rotation um die z-Achse um den Ursprung
  - Skalierung in x-Richtung, y-Richtung und z-Richtung
- Kennen wir die **fünf** Matrizen für die Standardtransformationen, können wir also jede beliebige affine Transformation berechnen.

wir benötigen  
**fünf** Formeln

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$



## Standardtransformationen

- Für 3D liegt nahe, dass diese Matrizen 3x3 Matrizen sind
- Problem: es gibt keine 3x3 Matrizen, die das Gewünschte leisten
- Geniale Idee: wir nehmen 4x4 Matrizen und verwenden statt 3D Koordinaten sogenannte homogene Koordinaten

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

4D

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$



## Homogene Koordinaten

- Umrechnung Homogene Koordinate – 3D Koordinate:

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \cong \frac{1}{w} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \cong \begin{pmatrix} w \cdot x \\ w \cdot y \\ w \cdot z \\ w \end{pmatrix}$$

- Üblicherweise wird  $w=1$  gewählt
- Für  $w=0$  geben homogene Koordinaten keine Punkte, sondern Richtungen (Vektoren) an



## Transformation mit homogenen Koordinaten

- Unsere Formel lautet nun in homogenen Koordinaten

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

- Können jetzt die Transformationsmatrizen für die **fünf** Standardfälle angeben



## Translation

- Verschiebung um den Vektor  $(t_x, t_y, t_z)$

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## Rotation

- Rotation um x-Achse (Drehpunkt: Ursprung)

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



## Rotation

- Rotation um y-Achse (Drehpunkt: Ursprung)

$$R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## Rotation

- Rotation um z-Achse (Drehpunkt: Ursprung)

$$R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 & 0 \\ \sin \gamma & \cos \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## Skalierung

- Skalierung um  $s_x$  in x-Richtung,  $s_y$  in y-Richtung und  $s_z$  in z-Richtung

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## Beispiel

- Wohin wird Punkt  $P(3,1,0)$  nach Rotation um  $30^\circ$  um die y-Achse mit Drehpunkt  $(2,0,0)$  abgebildet?

$$\begin{aligned}\vec{p}' &= T(2,0,0) \cdot R_y(30^\circ) \cdot T(-2,0,0) \cdot \vec{p} \\ &= \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos 30^\circ & 0 & \sin 30^\circ & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 30^\circ & 0 & \cos 30^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} 3 \\ 1 \\ 0 \\ 1 \end{pmatrix}\end{aligned}$$



## Inverse Transformation

- Eine Transformation  $M$  kann durch die Transformation  $M^{-1}$  wieder rückgängig gemacht werden (müssen also die Transformationsmatrix invertieren)
- Rotationsmatrizen sind orthonormal: die inverse Matrix ist identisch mit der transponierten Matrix
- Für unsere Standardmatrizen gilt:
  - $T(t_x, t_y, t_z)^{-1} = T(-t_x, -t_y, -t_z)$
  - $R_x(\alpha)^{-1} = R_x(-\alpha)$
  - $S(s_x, s_y, s_z)^{-1} = S(s_x^{-1}, s_y^{-1}, s_z^{-1})$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [105]



© R. Dörner

## Transformation von Objekten

- Bei Polygonen: jeden Eckpunkt transformieren, Objekt neu zeichnen
- Transformation von Vektoren (z.B. Flächen-normalen): nicht mit  $M$  transformieren, sondern mit  $(M^{-1})^T$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [106]



© R. Dörner



## Transformationsmatrizen in 2D

- Ebenfalls Verwendung von homogenen Koordinaten (hier: 3 dimensional)
- Verwendung von 3x3 Matrizen
- Translations- und Skalierungsmatrix analog zu 3D; es gibt nur eine Rotationsmatrix:

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



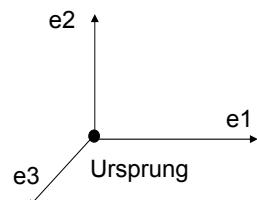
## Szenengraphen und Koordinatensysteme

- B.1 Shape-Nodes
- B.2 Transform-Nodes und Objekthierarchien
- B.3 Transformationsmatrizen
- B.4 Wechsel von Koordinatensystemen**

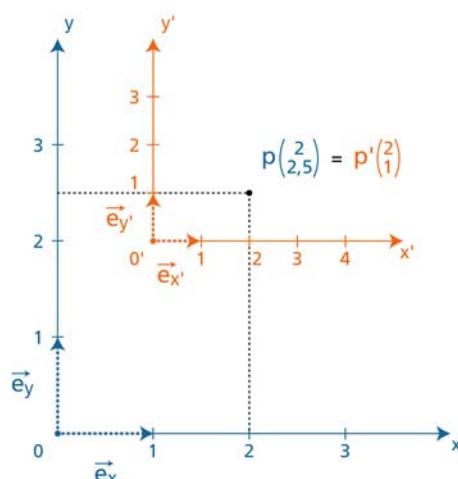


## Koordinatensysteme

- In 3D ein Koordinatensystem besteht aus 3 linear unabhängigen Richtungen (Einheitsvektoren) und einem Referenzpunkt (Ursprung)
- In homogenen Koordinaten haben die Einheitsvektoren die w-Komponente 0, der Ursprungspunkte eine w-Komponente ungleich 0



## Verschiedene Koordinatensysteme



$P = O + 2 \cdot e_x + 1 \cdot e_y$   
Koordinaten von P: (2; 1)

$P = O + 2 \cdot e_x' + 2,5 \cdot e_y'$   
Koordinaten von P: (2; 2,5)

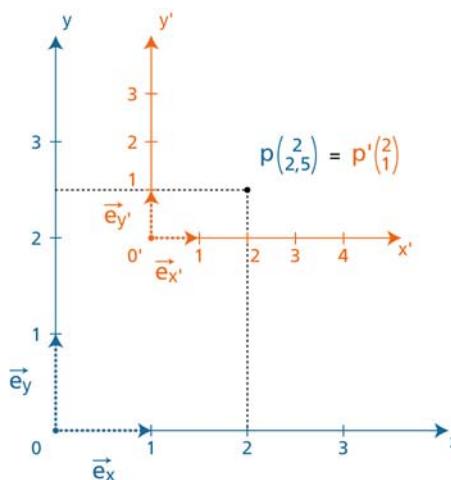
**Gesucht:** Matrix, die Koordinaten zwischen verschiedenen Koordinatensystem umrechnet

## Wechsel von Koordinatensystemen

1. Bestimme Koordinaten der Richtungsvektoren des neuen Koordinatensystems bezüglich des alten Koordinatensystems in homogenen Koordinaten
2. Bestimme Koordinaten des neuen Ursprungspunkts bezüglich des alten Koordinatensystems in homogenen Koordinaten
3. Baue eine Matrix T auf: erste Spalte sind Koordinaten des ersten Richtungsvektors (wie unter 1.), zweite Spalte sind Koordinaten des zweiten Richtungsvektors (wie unter 1.), usw., letzte Spalte sind die Koordinaten des Ursprungspunktes (wie unter 2.)
4. Multipliziere die Koordinaten eines Punktes P bezüglich des alten Koordinatensystems mit  $T^{-1}$ : man erhält die Koordinaten von P bezüglich des neuen Koordinatensystems



## Beispiel



Richtung von  $\vec{e}_{x'}$  bezüglich  $\vec{e}_x$  und  $\vec{e}_y$  :

$$\begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix} := \vec{a}$$

Richtung von  $\vec{e}_{y'}$  bezüglich  $\vec{e}_x$  und  $\vec{e}_y$  :

$$\begin{pmatrix} 0 \\ \frac{1}{2} \\ 0 \end{pmatrix} := \vec{b}$$

Richtung von  $0'$  bezüglich  $\vec{e}_x$  und  $\vec{e}_y$  :

$$\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} := \vec{c}$$

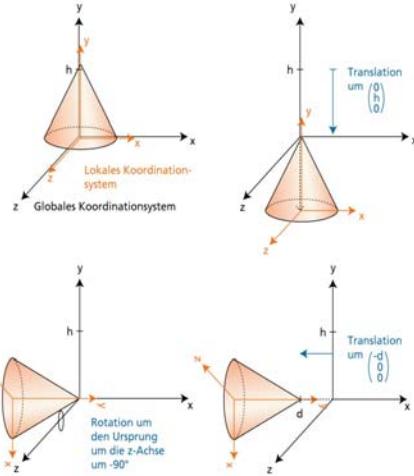
$\Rightarrow T = (\vec{a} \ \vec{b} \ \vec{c}) = \begin{pmatrix} \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 2 \\ 0 & 0 & 1 \end{pmatrix}$

$\bar{p} = T \cdot \bar{p}' \Leftrightarrow \bar{p}' = T^{-1} \cdot \bar{p}$



## Transformation und Wechsel des Koordinatensystems

- Unsere Transformationsmatrizen für Rotation, Translation und Skalierung können auch als Wechselmatrizen zwischen Koordinatensystemen aufgefasst werden
- Jede Transformation wechselt also zwischen zwei Koordinatensystemen



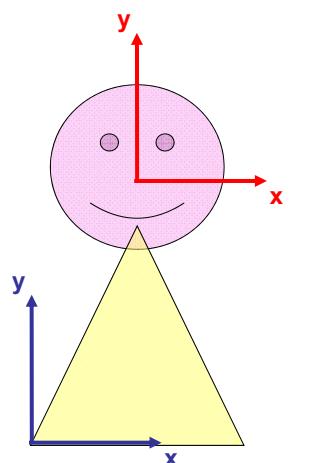
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [113]



© R. Dörner

## Beispiel



Umrechnung:

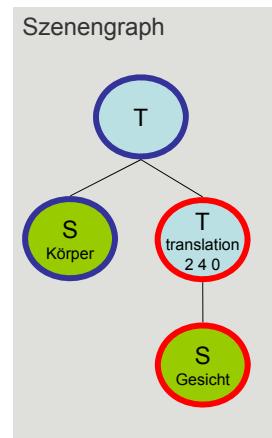
1. Transform-Node entspricht  $T(2,4,0)$

2. Umrechnung rot  $\rightarrow$  blau

$$\vec{p} = T(2,4,0) \cdot \vec{p}$$

3. Umrechnung blau  $\rightarrow$  rot

$$\vec{p}' = (T(2,4,0))^{-1} \cdot \vec{p}'$$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [114]



© R. Dörner



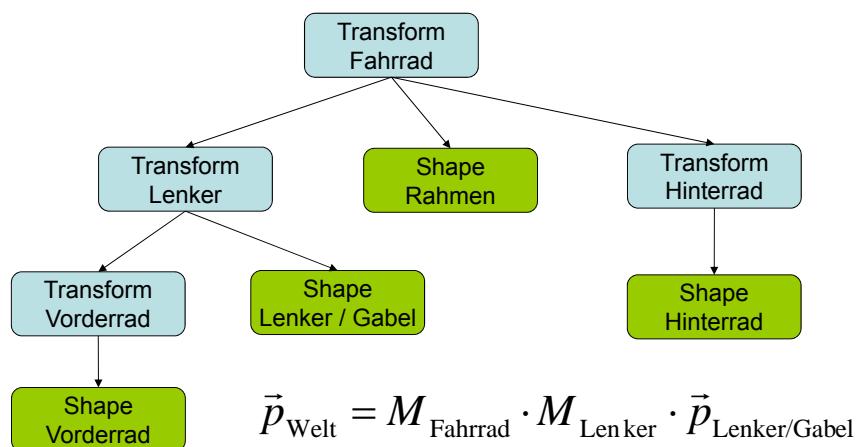
## Transform Node in VRML

Jeder Transform-Node in VRML bestimmt eine Transformationsmatrix

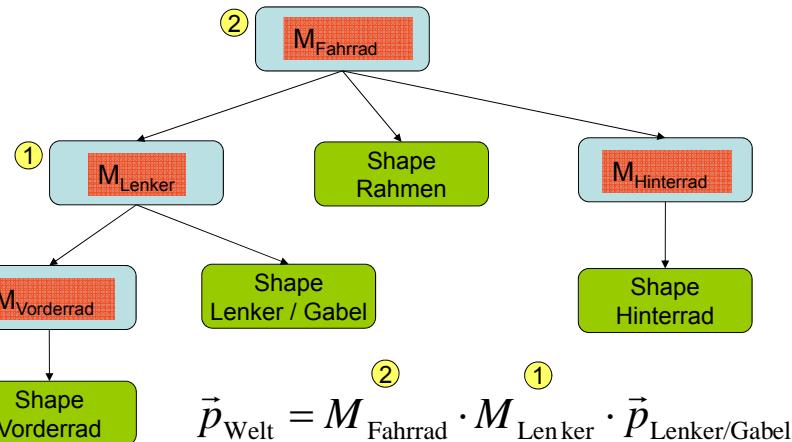
$$M = T(t) \cdot T(c) \cdot R \cdot R_s \cdot S \cdot R_s^T \cdot T(-c)$$

1. Skalierung **S** mit ScaleOrientation **R<sub>s</sub>**
2. Rotation **R** um center point **c**
3. Translation **T(t)**

## Szenengraph und Transformationsmatrizen



## Szenengraph und Transformationsmatrizen



## Konkatenation in VRML

- Reihenfolge in verschachtelten Transformnodes: der innere Node wird zuerst angewendet
- Beispiel:

```
Transform {
    rotation 1 0 0 1.58 ②
    children Transform {
        translation 1 2 0 ①
        children Shape { ....
    }
}
```

Hier wird zuerst die Translation durchgeführt, danach die Drehung



## Wechsel von Koordinatensystemen

Es gibt zwei Alternativen:

Alternative 1:

Wechselmatrix direkt aufstellen

Alternative 2:  
Transformationen ermitteln, die schrittweise das eine Koordinatensystem in das andere Koordinatensystem überführen - entsprechende Matrizen (in richtiger Reihenfolge) aufmultiplizieren



## Teil C: Kameramodell



## Die virtuelle Kamera

- Metapher 1: Virtueller Betrachter / virtuelles Auge befindet sich in der Szene
- Metapher 2: Bild wird aus Sicht einer virtuellen Kamera erstellt, die in der Szene angebracht wird
- Müssen spezifizieren:
  - Äußere Parameter
    - Position
    - Orientierung
  - Innere Parameter
    - Objektiv / Brennweite
    - Seitenverhältnis des Bildes (Aspect Ratio)



Computergraphik – Prof. Dr. R. Dörner – WS 15

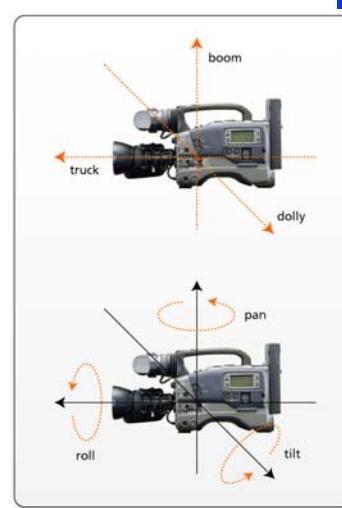
B-AI-V1 [121]



© R. Dörner

## Äußere Parameter

- Spezielle Terminologie aus dem Film
- Position:
  - truck, dolly, boom
- Orientierung
  - pan, roll, tilt



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [122]

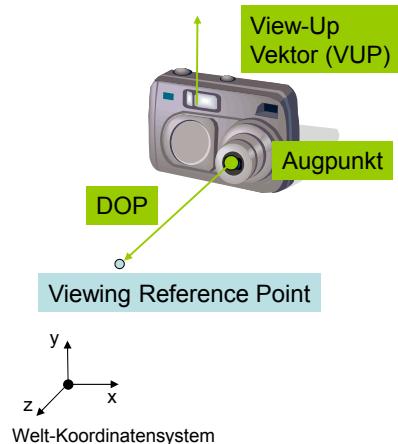


© R. Dörner



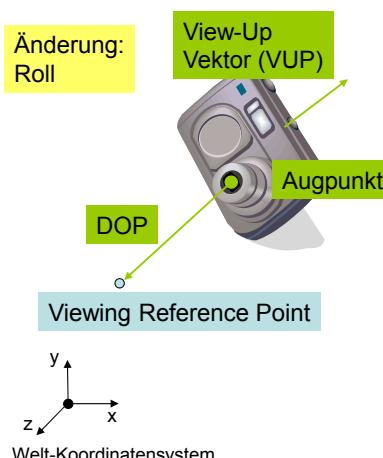
## Spezifikation von Position und Orientierung

- Position der Kamera:
  - Angabe eines Punktes (Augpunkt) in Weltkoordinaten
- Ausrichtung der Kamera:
  - Angabe eines Punktes auf den die Kamera zentral blickt (Viewing Reference Point) in Weltkoordinaten
  - Vektor vom Augpunkt zum Viewing Reference Point heisst DOP (Direction of Projection)
  - Angabe eines Vektors, der nach oben zeigt (View Up Vektor) in Weltkoordinaten



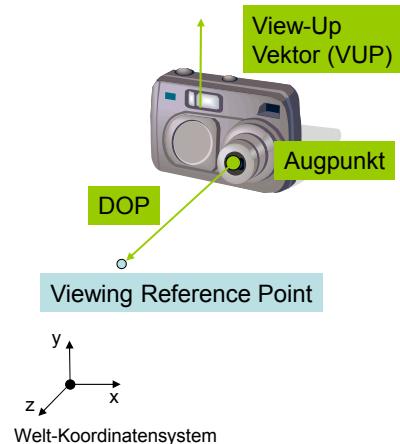
## Spezifikation von Position und Orientierung

- Position der Kamera:
  - Angabe eines Punktes (Augpunkt) in Weltkoordinaten
- Ausrichtung der Kamera:
  - Angabe eines Punktes auf den die Kamera zentral blickt (Viewing Reference Point) in Weltkoordinaten
  - Vektor vom Augpunkt zum Viewing Reference Point heisst DOP (Direction of Projection)
  - Angabe eines Vektors, der nach oben zeigt (View Up Vektor) in Weltkoordinaten

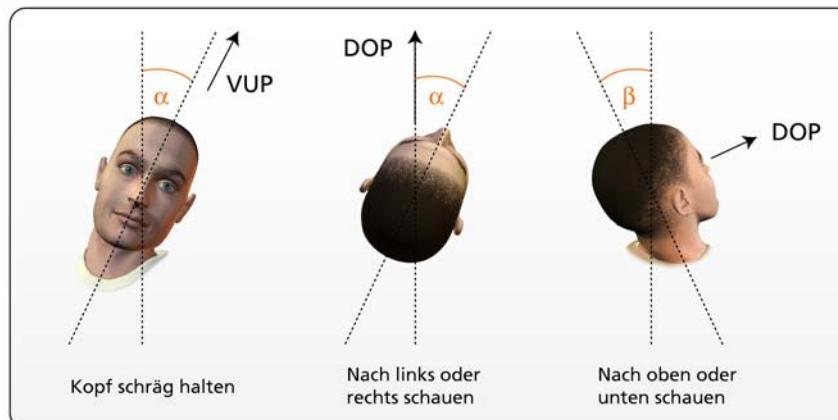


## Spezifikation von Position und Orientierung

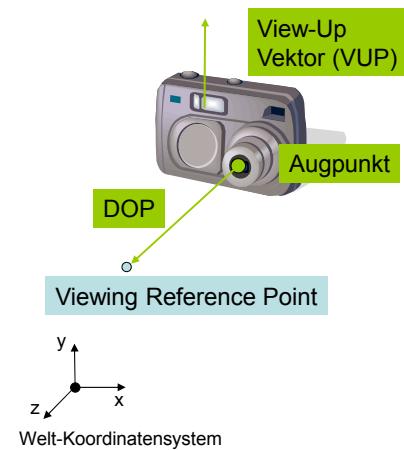
- Position der Kamera:
  - Angabe eines Punktes (Augpunkt) in Weltkoordinaten
- Ausrichtung der Kamera:
  - Angabe eines Punktes auf den die Kamera zentral blickt (Viewing Reference Point) in Weltkoordinaten
  - Vektor vom Augpunkt zum Viewing Reference Point heisst DOP (Direction of Projection)
  - Angabe eines Vektors, der nach oben zeigt (View Up Vektor) in Weltkoordinaten



## Wahl von VUP und DOP

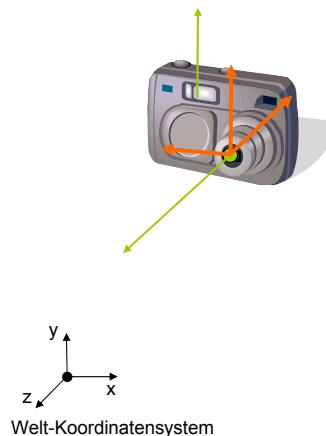


## Kamerakoordinaten



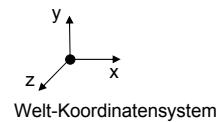
## Kamerakoordinaten

- Wir finden einen Vektor UP der auf DOP senkrecht steht und wie VUP nach oben zeigt mittels Projektion
- Wir finden durch Kreuzprodukt von DOP und UP einen Vektor, der auf beiden senkrecht steht
- Dieser Vektor gemeinsam mit -DOP und UP und dem Augpunkt ergibt ein Koordinatensystem



## Kamerakoordinaten

- Wir finden einen Vektor UP der auf DOP senkrecht steht und wie VUP nach oben zeigt mittels Projektion
- Wir finden durch Kreuzprodukt von DOP und UP einen Vektor, der auf beiden senkrecht steht
- Dieser Vektor gemeinsam mit -DOP und UP und dem Augpunkt ergibt ein Koordinatensystem

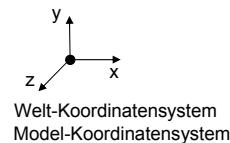


## Kamerakoordinaten

- Per Konvention werden die x-, y- und z-Richtungen festgelegt
- Die Koordinaten heißen Kamerakoordinaten oder Viewkoordinaten
- Angabe des View-Koordinatensystems = Angabe der äußeren Parameter

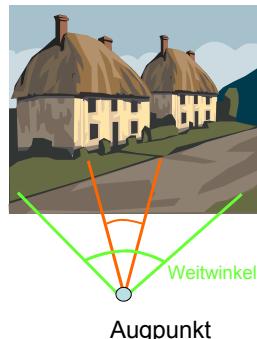


x ist auf dem Bild rechts  
y ist auf dem Bild oben  
Kamera schaut in Richtung negativer z-Achse.

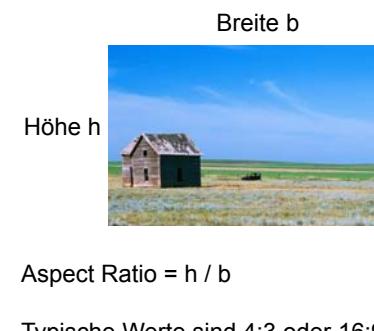


## Innere Parameter

- Blickwinkel / Sehwinkel  
(Field of View)

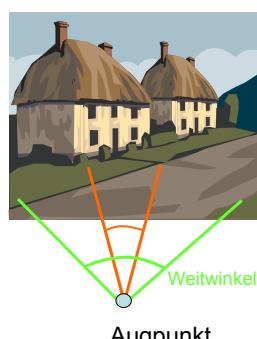


- Seitenverhältnis  
(Aspect Ratio)



## Innere Parameter

- Blickwinkel / Sehwinkel  
(Field of View)

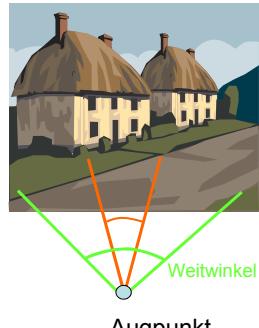


- Seitenverhältnis  
(Aspect Ratio)



## Innere Parameter

- Blickwinkel / Sehwinkel  
(Field of View)



- Seitenverhältnis  
(Aspect Ratio)



$$\text{Aspect Ratio} = h / b$$

Typische Werte sind 4:3 oder 16:9



## Kamera Modell in VRML

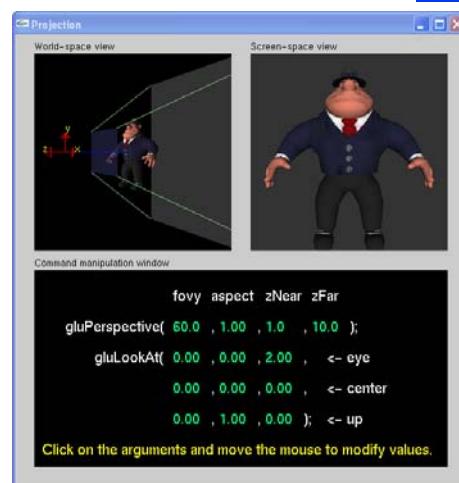
- fieldOfView: Sehwinkel in Radian
- position: Koordinaten des EyePoint
- orientation: Rotation aus einer Standardposition (schauen entlang negativer z-Achse)
- description: Unterscheidung mehrerer Viewpoints

```
ViewPoint{  
    fieldOfView      0.3455  
    position        0 0 10  
    orientation     0 0 1 0  
    description     "vorn"  
}
```



## Andere Spezifikationsformen

- In VRML: Autor kann innere Parameter nicht ändern, sind festgelegt
- Kamera kann auch anders als in VRML spezifiziert werden
- Beispiel:
  - LookAt-Funktion für äußere Parameter
    - Augpunkt
    - Viewing Reference Point
    - Up Vector
  - Perspective-Funktion für innere Parameter
    - Field of View
    - Aspect Ratio
    - Near Plane
    - Far Plane



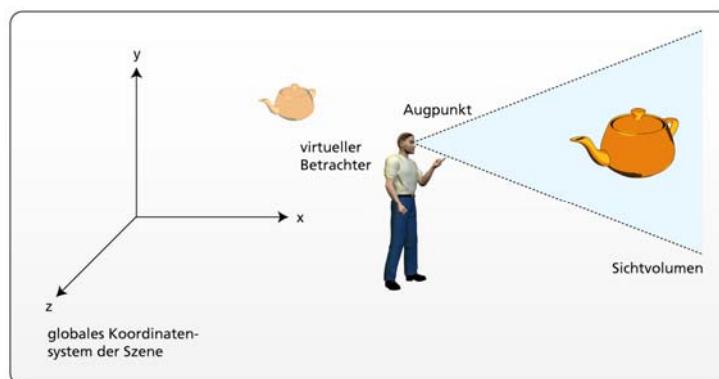
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [135]



© R. Dörner

## Sichtvolumen



Nur ein bestimmter Bereich der Szene  
kann im Bild erscheinen: Sichtvolumen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [136]

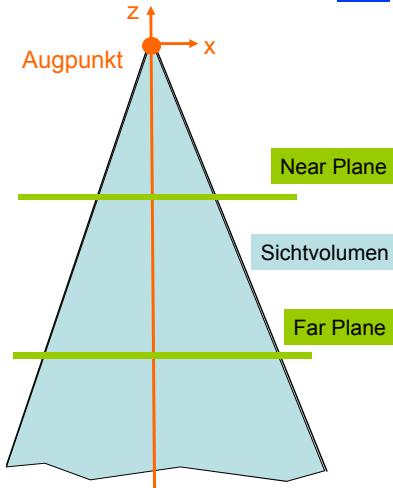


© R. Dörner



## Sichtvolumen

- Alle Objekte außerhalb des Sichtvolumens (Viewing Volume) sind garantiert nicht auf dem Bild zu sehen
- Das Sichtvolumen wird durch einfügen von zwei Ebenen (Near Plane und Far Plane), die parallel zur xy-Ebene liegen, endlich gemacht



Computergraphik – Prof. Dr. R. Dörner – WS 15

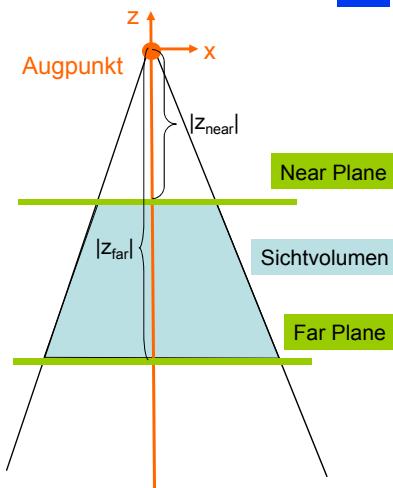
B-AI-V1 [137]



© R. Dörner

## Sichtvolumen

- Alle Objekte außerhalb des Sichtvolumens (Viewing Volume) sind garantiert nicht auf dem Bild zu sehen
- Das Sichtvolumen wird durch einfügen von zwei Ebenen (Near Plane und Far Plane), die parallel zur xy-Ebene liegen, endlich gemacht



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [138]

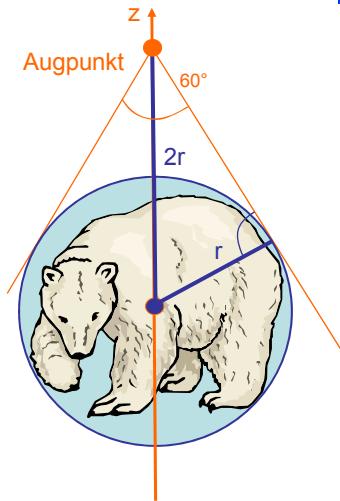


© R. Dörner



## Sichtkugelverfahren

- Anleitung, wie man eine gute Kameraposition zur Betrachtung eines Objekts findet
- Schritte:
  - Kugel um das Objekt legen
  - Augpunkt im Abstand des Kugeldurchmessers wählen
- Durch das Verfahren wird automatisch ein FOV von  $60^\circ$  erreicht, was positiv wahrgenommen wird



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [139]

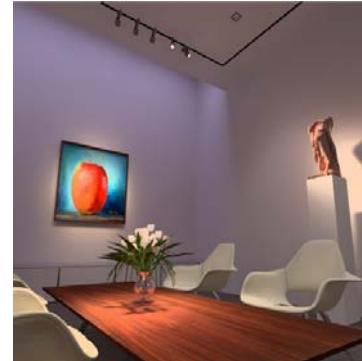
© R. Dörner

## Teil D: Beleuchtungsmodell



## Licht und Materie

- Motivation: im Szenenmodell muss Beleuchtung spezifiziert werden
  - Position und Art der Lichtquellen, z.B. Sonnenlicht, roter Lichtkegel eines Scheinwerfers
  - Materialeigenschaften (Teil des Objektmodells), z.B. glänzend oder matt
  - Modell: wie interagiert Licht und Material?
- Physikalische Modelle aus der Optik sind ungeeignet, da i.d.R. zu aufwändig zu berechnen
  - Spezielle Beleuchtungsmodelle in der CG
  - Wichtig ist nicht physikalische Korrektheit sondern gutes Aussehen (visuelle Qualität)



Quelle: snipview.com

Computergraphik – Prof. Dr. R. Dörner – WS 15

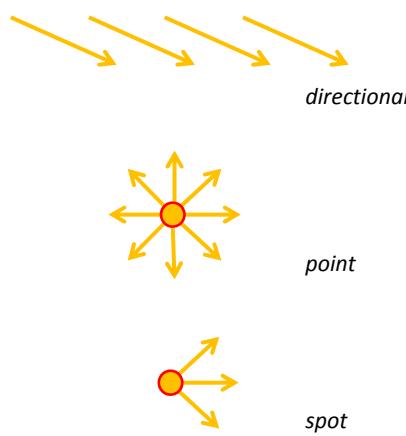
B-AI-V1 [141]



© R. Dörner

## Modelle für Lichtquellen

- Wichtige Parameter:
  - Position und Richtung
  - Intensität (Helligkeit), unterschieden nach Rot-, Grün- und Blau-Komponente der Lichtquelle
  - Art der Lichtquelle
- Lichtquellenarten:
  - Directional Light
  - Point Light
  - Spot Light
  - Area Light



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [142]

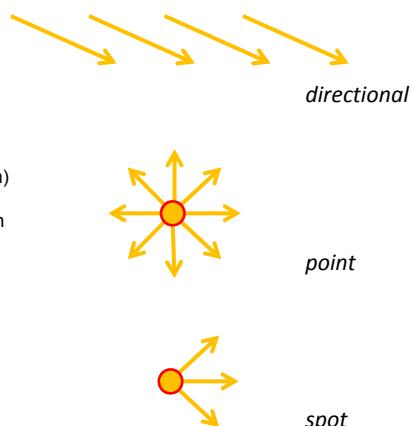


© R. Dörner



## Lichtquellen-Arten

- Directional Light:
  - Licht hat keine Position („im Unendlichen“), sondern nur Richtung
  - Lichtstrahlen sind parallel
  - Beispiel: Sonnenlicht
- Point Light und Spot Light:
  - Licht kommt von einem (unendlich kleinen) Punkt
  - Strahlt in alle Richtungen (Point) / in einen bestimmten Bereich (Spot)
  - Beispiel: Leuchtdiode, Glühbirne (Point), Taschenlampe (Spot)
- Area Light:
  - Lichtquelle ist flächig
  - Annäherung durch ein Feld von Point Lights
  - Beispiel: Leuchtstoffröhre



Computergraphik – Prof. Dr. R. Dörner – WS 15

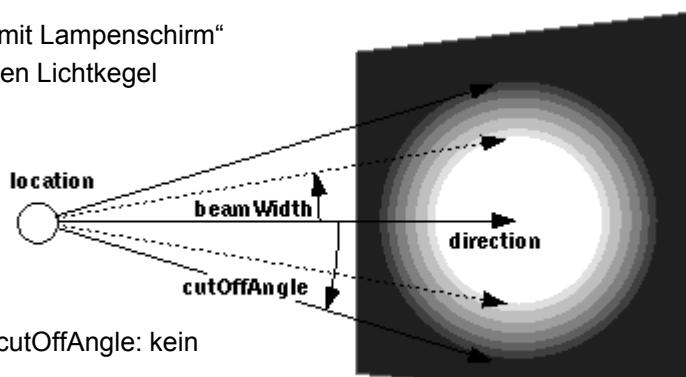
B-AI-V1 [143]



© R. Dörner

## Spot Light

- Punktlicht „mit Lampenschirm“
- Erzeugt einen Lichtkegel



- Außerhalb cutOffAngle: kein Licht
- Innerhalb beamWidth: Licht gleicher Intensität

Computergraphik – Prof. Dr. R. Dörner – WS 15

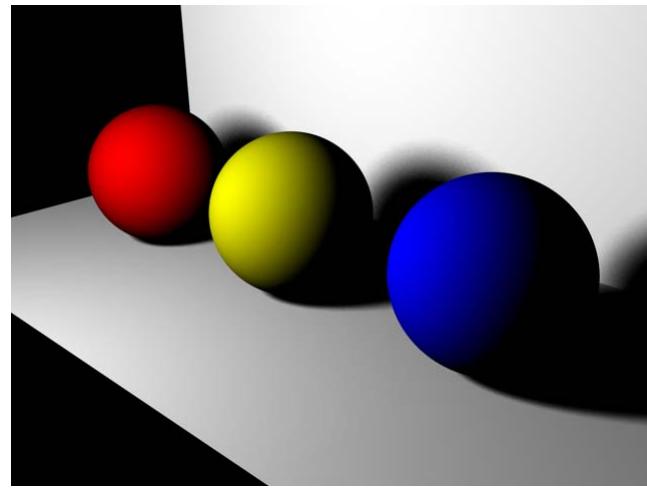
B-AI-V1 [144]



© C. Schulz



## Gegenseitige Beleuchtung von Objekten



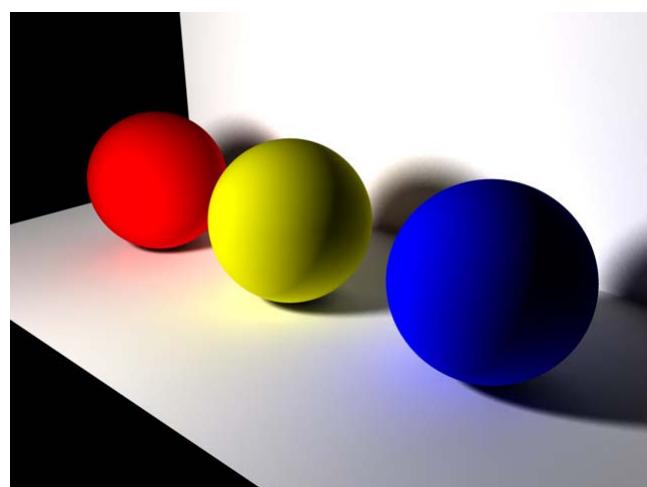
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [145]



© R. Dörner

## Gegenseitige Beleuchtung von Objekten



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [146]



© R. Dörner



## Eine „uneigentliche“ Lichtquelle

- Licht kommt nicht nur von den Lichtquellen, die Licht emittieren, sondern auch von Körpern, die nicht selbst leuchten durch gegenseitige Beleuchtung (Umgebungslicht, ambientes Licht)
- Sehr schwierig zu berechnen
- Idee: Annäherung des Umgebungslicht dadurch, dass alle Objekte angeleuchtet werden an jedem Punkt der Szene unabhängig von der Position und Richtung einer Lichtquelle



unter dem Tisch ist es nicht komplett dunkel, obwohl kein Licht direkt die Fläche anstrahlt



## Paralleles Licht in VRML

- Verwendung des Nodes `DirectionalLight`

```
DirectionalLight {  
    direction          1 0 0  
    intensity         1  
    color             1 0 0  
    ambientIntensity 1  
}
```

Beitrag der Lichtquelle  
zum ambienten Licht

Farbe = color · intensity



## Paralleles Licht in VRML



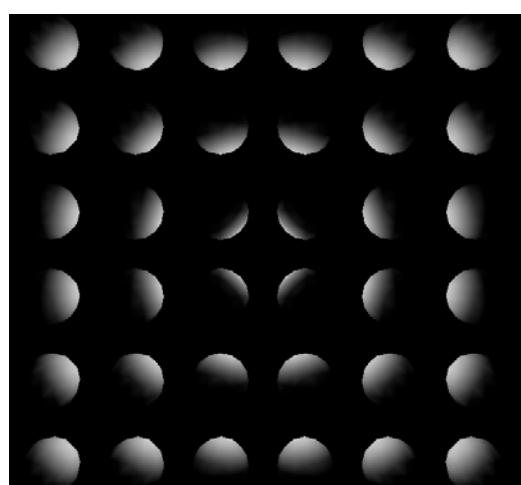
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [149]



© R. Dörner

## Punktlicht in VRML



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [150]

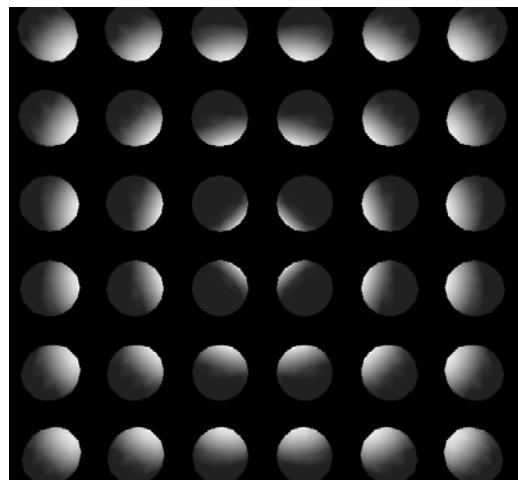


© R. Dörner



## Punktlicht in VRML

Mit stärkerem  
ambienten Licht



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [151]



© R. Dörner

## Punktlicht in VRML

- Verwendung des Nodes PointLight

```
PointLight {  
    location          0 0 1  
    intensity         1  
    color             1 0 0  
    ambientIntensity 0.3  
    radius            100  
    attenuation       0 0 1  
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [152]



© R. Dörner



## Attenuation (Lichtdämpfung)

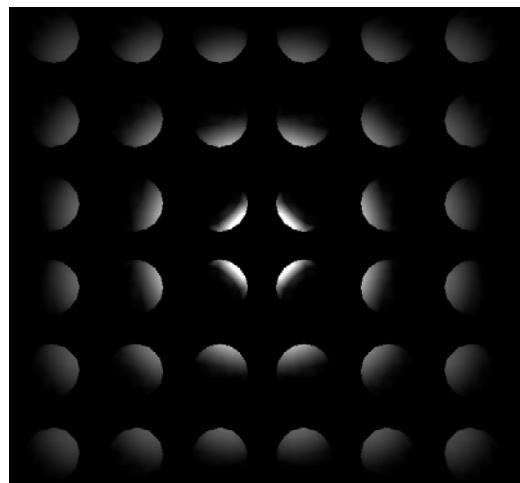
- Die Intensität  $I$  des Lichts nimmt mit der Entfernung  $d$  von der Lichtquelle ab
- Die Werte  $a_1, a_2, a_3$  des Feldes **attenuation** werden so berücksichtigt:

$$I' = \frac{I}{a_1 + a_2 \cdot d + a_3 \cdot d^2}$$

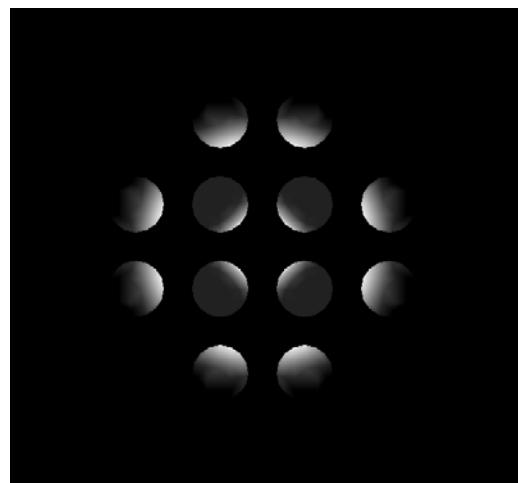
- VRML hat außerdem/alternativ noch einen Wirkungskreis mit Radius **radius**. Außerhalb des Wirkungskreises hat die Lichtquelle keine Auswirkungen.



## Punktlicht in VMRL mit Lichtdämpfung



## Punktlicht in VMRL mit Wirkungsradius



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [155]



© R. Dörner

## Spotlicht in VRML

- Verwendung des Nodes SpotLight

```
SpotLight {  
    location          0 0 1  
    intensity         1  
    color             1 0 0  
    ambientIntensity 0  
    radius            100  
    attenuation       0 0 1  
    beamWidth         0.7  
    cutOffAngle       0.9  
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

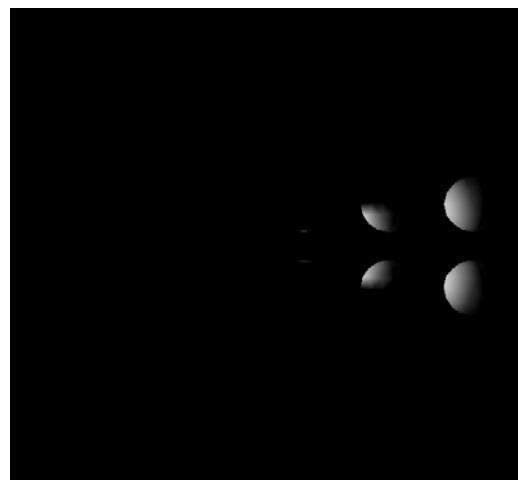
B-AI-V1 [156]



© R. Dörner



## Spotlicht in VRML



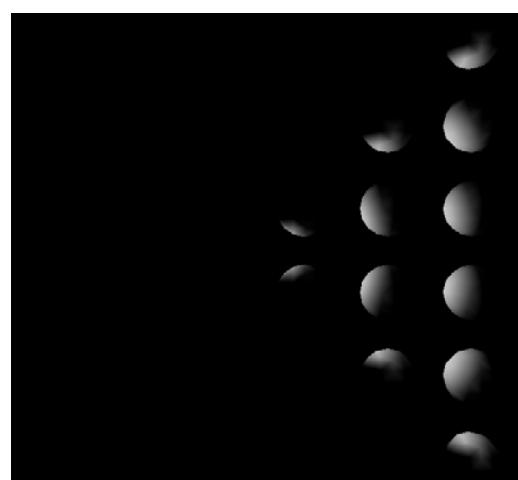
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [157]



© R. Dörner

## Spotlicht in VRML



Mit weiterem  
Öffnungswinkel

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [158]

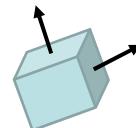
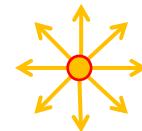


© R. Dörner



## Beleuchtungsrechnung

- Der User muss spezifizieren:
  - Welche Lichtquellen gibt es in der Szene, welche Eigenschaften haben sie
  - Welche Objekte gibt es in der Szene, welche Eigenschaften haben deren Oberfläche (Appearance)
- Benötigen nun ein Modell, das angibt, wie aus diesen Informationen der Farbwert für einen Punkt auf der Objektoberfläche berechnet wird (um letztendlich den Farbwert eines Pixels im Bild zu bestimmen)
- Ein häufig verwendetes Modell, das auch VRML nutzt, ist das Phong Beleuchtungsmodell



Computergraphik – Prof. Dr. R. Dörner – WS 15

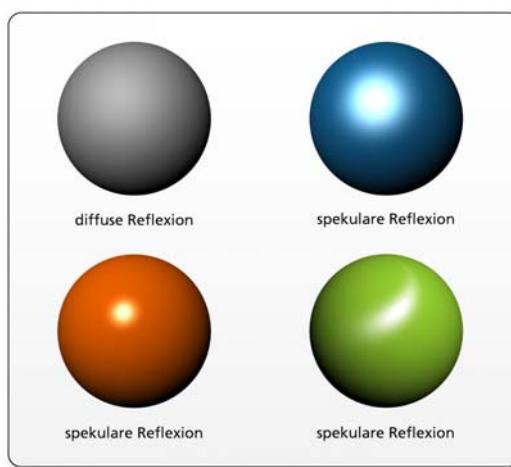
B-AI-V1 [159]



© R. Dörner

## Beleuchtungsmodell von Phong

- Vereinfachung der physikalischen Beleuchtung und Reflexion
- Drei Beleuchtungsanteile (werden aufaddiert)
  - Diffus
  - Spekular
  - Ambient



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [160]

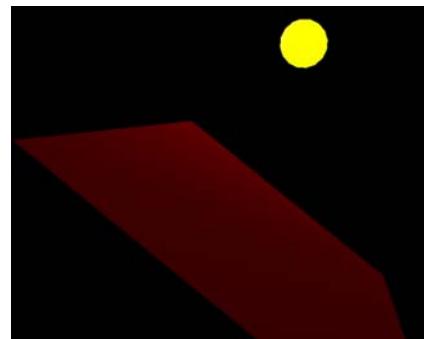


© R. Dörner



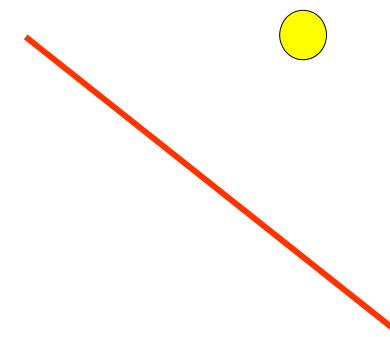
## Diffuse Reflexion nach Phong

- Streulicht: Licht trifft auf rauhe Oberfläche und wird in alle Richtungen gestreut
- Position des Betrachters ist unwichtig
- Je steiler das Licht einfällt, desto mehr wird reflektiert



## Diffuse Reflexion nach Phong

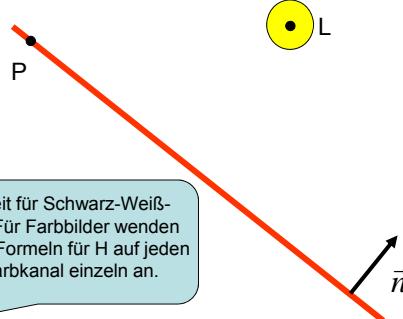
- Streulicht: Licht trifft auf rauhe Oberfläche und wird in alle Richtungen gestreut
- Position des Betrachters ist unwichtig
- Je steiler das Licht einfällt, desto mehr wird reflektiert



## Diffuse Reflexion nach Phong

- L ist Lichtquellenposition
- I ist die Intensität der Lichtquelle
- Reflektionseigenschaft der Fläche ist beschrieben durch Reflektionskoeffizient R
- Ausrichtung der Fläche ist beschrieben durch deren Normalenvektor
- Gesucht ist Helligkeit H an Punkt P

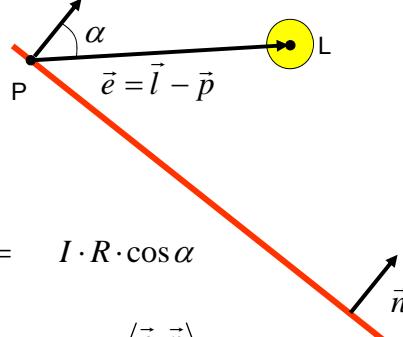
Helligkeit für Schwarz-Weiß-Bilder. Für Farbbilder wenden wir die Formeln für H auf jeden RGB Farbkanal einzeln an.



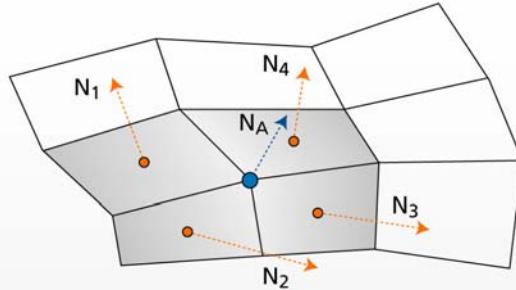
## Diffuse Reflexion nach Phong

- L ist Lichtquellenposition
- I ist die Intensität der Lichtquelle
- Reflektionseigenschaft der Fläche ist beschrieben durch Reflektionskoeffizient R
- Ausrichtung der Fläche ist beschrieben durch deren Normalenvektor
- Gesucht ist Helligkeit H an Punkt P

$$\begin{aligned} H &= I \cdot R \cdot \cos \alpha \\ &= I \cdot R \cdot \frac{\langle \vec{e}, \vec{n} \rangle}{| \vec{e} | \cdot | \vec{n} |} \end{aligned}$$



## Woher kommt die Normale?



$$N_A = \frac{N_1 + N_2 + N_3 + N_4}{4}$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

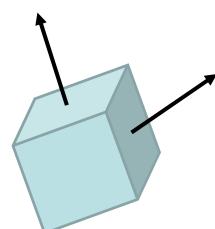
B-AI-V1 [165]



© R. Dörner

## Woher kommt die Normale?

- VRML berechnet die Normalen nicht automatisch – dies muss der Anwender selbst tun
- Müssen an allen Eckpunkten ausgerechnet und dann angegeben werden
- In VRML gibt es dafür entsprechende Sprach-Konstrukte: Node namens **Normal** (entsprechend gibt es ein Feld **normal** im **IndexedFaceSet** Node)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [166]

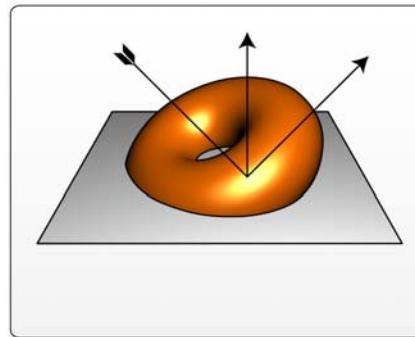


© R. Dörner



## Spekulare Reflexion nach Phong

- Glanzlicht (engl. Highlight): Licht trifft auf glatte Oberfläche und wird nach dem Reflexionsgesetz „Einfallsinkel = Ausfallswinkel“ reflektiert
- Lichtquelle spiegelt sich im Objekt, es entstehen Highlights in der Farbe der Lichtquelle
- Je glatter das Objekt, desto schärfer begrenzt und kleiner sind die Highlights
- Position des Betrachters ist wichtig



Computergraphik – Prof. Dr. R. Dörner – WS 15

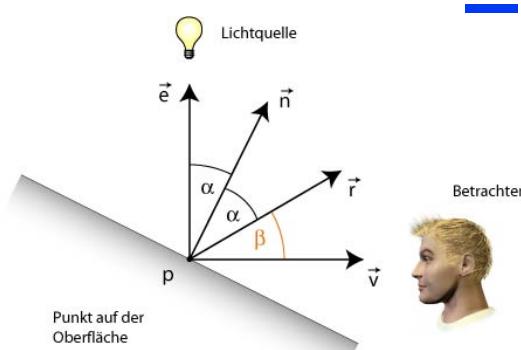
B-AI-V1 [167]

© R. Dörner

## Spekulare Reflexion nach Phong

- $I$  ist die Intensität der Lichtquelle
- $R$  ist der Koeffizient, der angibt, wieviel spekulares Licht reflektiert wird
- Die Shininess  $s$  ist ein Maß für die Glattheit der Oberfläche

$$H = I \cdot R \cdot \cos^s \beta$$



$$= I \cdot R \cdot \left( \frac{\langle \vec{r}, \vec{v} \rangle}{|\vec{r}| \cdot |\vec{v}|} \right)^s$$

$$\vec{r} = 2 \cdot \frac{\vec{n}}{|\vec{n}|} \cdot \left\langle \frac{\vec{n}}{|\vec{n}|}, \frac{\vec{e}}{|\vec{e}|} \right\rangle - \frac{\vec{e}}{|\vec{e}|}$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [168]

© R. Dörner

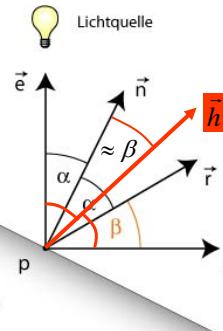


## Phong-Blinn Beleuchtungsmodell

- Reflektionsrichtung ist aufwendig zu berechnen, muss für jeden Punkt P neu durchgeführt werden
- Idee: Annäherung mit dem Halbvektor

$$H = I \cdot R \cdot \cos^s \beta$$

$$\approx I \cdot R \cdot \left( \frac{\langle \vec{n}, \vec{h} \rangle}{|\vec{n}| \cdot |\vec{h}|} \right)^s$$



Punkt auf der Oberfläche



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [169]



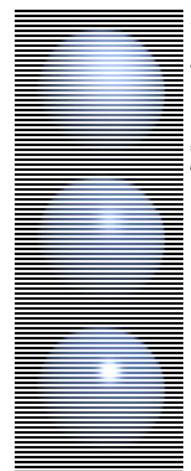
© R. Dörner

## Shininess

- je höher der Wert von s, desto scharf umrissener das Highlight, desto glatter wirkt das Material
- Wert hat keine physikalische Bedeutung
- Ausprobieren (z.B. Aufgabe von Artists in der Entwicklung von Medien mit Computergrafik)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [170]



Quelle: cgru.sourceforge.net



## Beleuchtungsmodell von Phong

- Ambientes Licht ist bei Phong einfach eine Konstante:  
 $H = R \cdot I$
- Resultat für eine Lichtquelle:  
$$H = H_{diff}(I_{diff}, R_{diff}) + H_{spec}(I_{spec}, R_{spec}, s) + H_{amb}(I_{amb}, R_{amb})$$
- Die Resultate für alle Lichtquellen werden aufsummiert
- Intensitäten werden vom User bei der Spezifikation der Lichtquelle angegeben
- Reflexionskoeffizienten  $R_{diff}$ ,  $R_{spec}$ ,  $s$  und  $R_{amb}$  müssen im Objektmodell (im Rahmen der Appearance Beschreibung) angegeben werden



## Reflexionskoeffizienten in VRML

- Verwendung des Nodes Material

```
material Material {  
    diffuseColor  
    specularColor  
    shininess  
    ambientIntensity  
    emissiveColor  
    transparency  
}
```

Objekt soll so aussehen, als ob es grün leuchtet

Durchsichtigkeit: man sieht den Hintergrund zu 70%

$s$

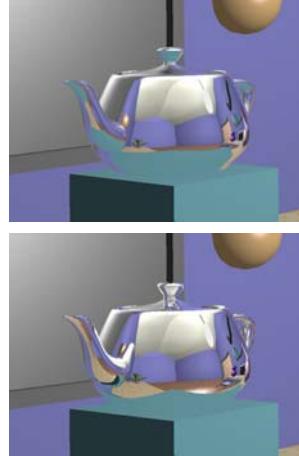
$R_{diff}$ für den Rot-Kanal
1.0 0.0 0.4
0.5 0.5 0.5
1.0
0.2
0.0 0.3 0.0
0.3

$R_{spec}$  für den Grün-Kanal



## Lokale Beleuchtungsmodelle

- Im Phong-Beleuchtungsmodell wird jeder Punkt eines Objektes isoliert betrachtet
  - Kein Schattenwurf
  - Keine Spiegelung
  - Keine Lichtbrechung
  - Starke Vereinfachung des ambienten Lichts
- Vorteile
  - Schnelle Berechnung durch Vereinfachung
  - Gute Parallelisierbarkeit
- Nachteile
  - Abweichungen von der Realität
  - Fehlende Effekte müssen durch „Tricks“ realisiert werden



Quelle: Watt / Polycarpo 3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 15

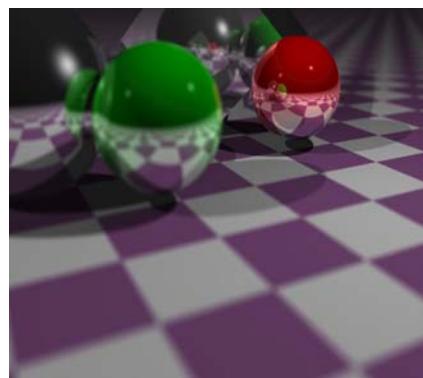
B-AI-V1 [173]



© R. Dörner

## Globale Beleuchtungsmodelle

- Globale Beleuchtungsmodelle berücksichtigen die Szene als Ganzes, können Wechselwirkungen zwischen Objekten erfassen
- Wichtige Vertreter
  - Strahlverfolgung (Raytracing)
  - Lichteinflussrechnung (Radiosity)

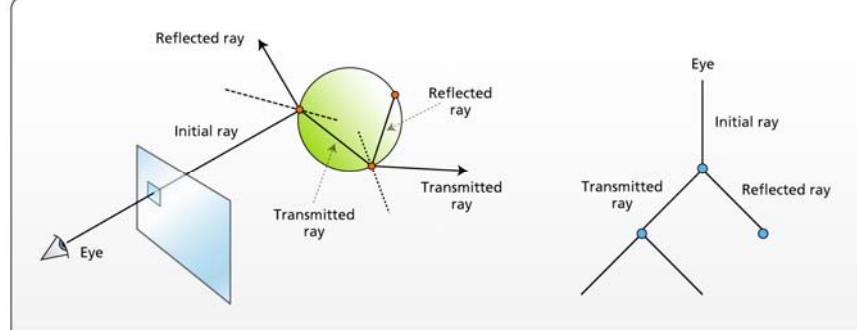


Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [174]



## Raytracing



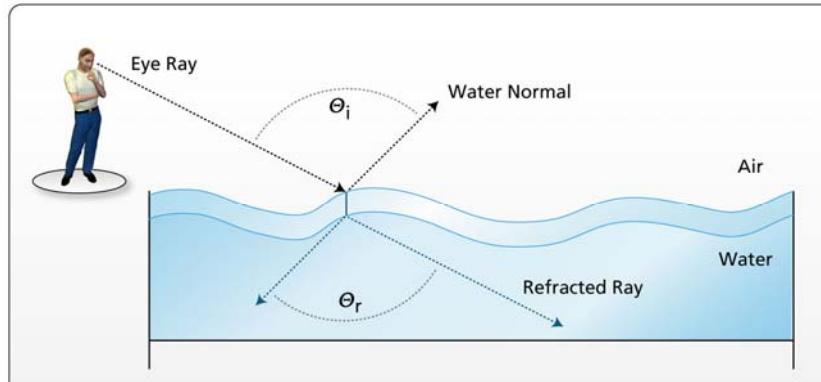
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [175]



© R. Dörner

## Raytracing mit Refraktion (Lichtbrechung)



Computergraphik – Prof. Dr. R. Dörner – WS 15

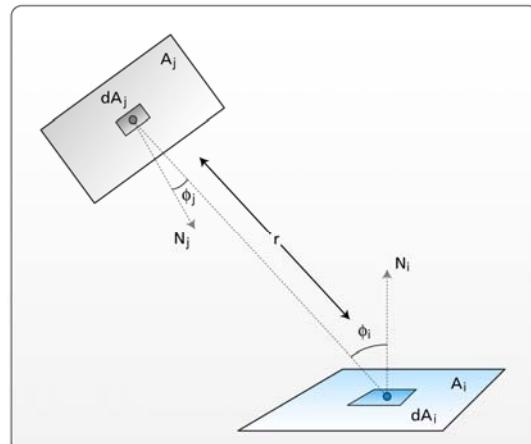
B-AI-V1 [176]



© R. Dörner



## Radiosity



Lichtausgleichsrechnung:

Tesselierung der Szene in Mikrofacetten

Mathematische Beschreibung der Lichtinteraktion zwischen Mikrofacetten

Lösung eines Gleichungssystems (Anzahl Gleichungen = Anzahl Mikrofacetten)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [177]



© R. Dörner

## Teil E: Szenenmodell

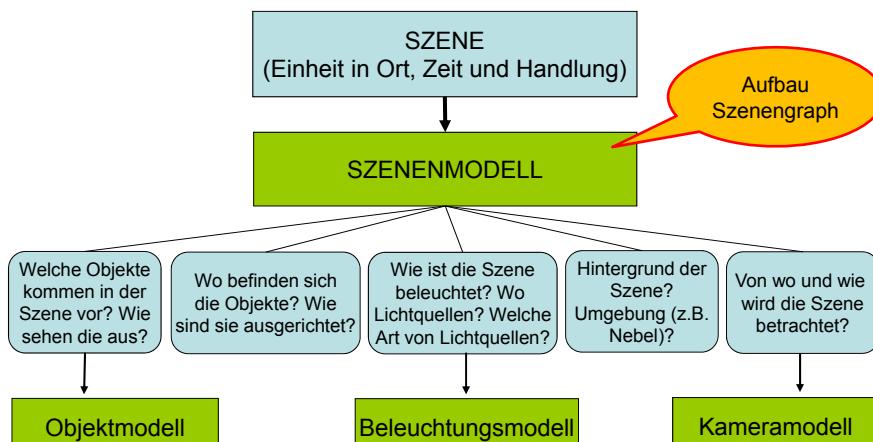


## Szenenmodell

- E.1 Zusammensetzen der Szene**
- E.2 Shading**
- E.3 Hintergrund und Umgebung**
- E.4 Animation und Interaktion**



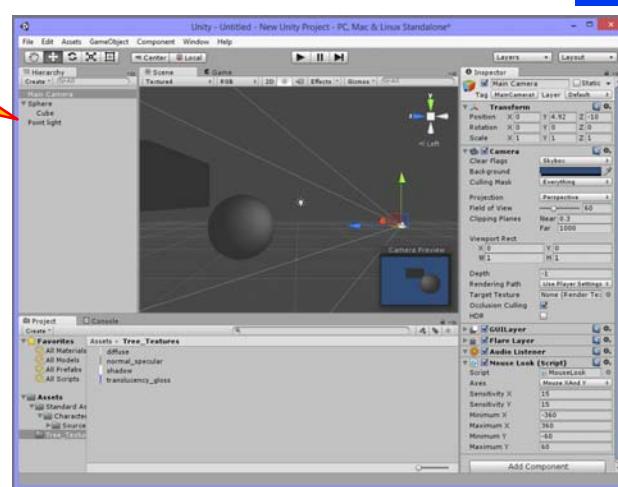
## Szenenmodell



## Tools für den Aufbau der Szene

Darstellung  
Szenengraph

Beispiel eines  
Editors zur  
Erstellung  
einer Szene:  
Unity3D  
([www.unity3d.com](http://www.unity3d.com))



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [181]



## Szenenmodell

- E.1 Zusammensetzen der Szene**
- E.2 Shading**
- E.3 Hintergrund und Umgebung**
- E.4 Animation und Interaktion**

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [182]

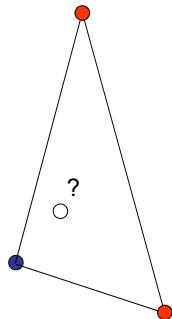


© R. Dörner



## Shading

- Geben Materialeigenschaften (z.B. Farben) nur für Eckpunkte (Vertices) an
- Beleuchtungsrechnung wird nur für die Eckpunkte durchgeführt
- Welche Farben haben denn dann die Punkte, die keine Eckpunkte sind?
- Dies wird durch das Shading bestimmt: Auswahl des Shadings ist Teil des Szenenmodells



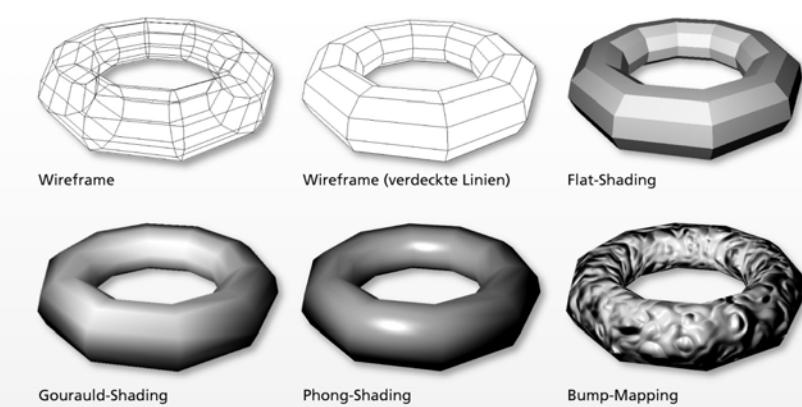
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [183]



© R. Dörner

## Shading: Beispiele



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [184]

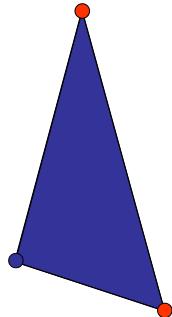


© R. Dörner



## Flat Shading

- Jede Fläche bekommt genau eine Farbe zugewiesen
- Welche Farbe? Verschiedene Möglichkeiten der Auswahl
  - z.B.: es wird die Farbe des Eckpunktes gewählt, der als Letzter in der Flächenliste steht
- Problem: Flat-Shading hebt den Fehler hervor, dass runde Objekte durch eckige Polygonnetze angenähert werden



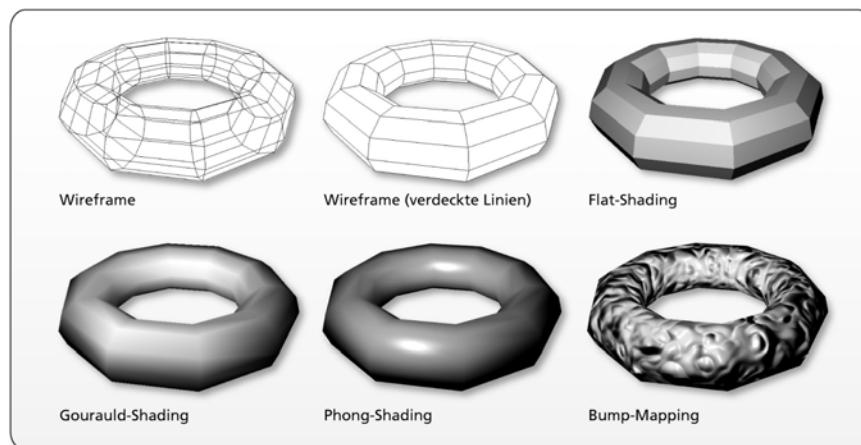
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [185]



© R. Dörner

## Shading: Beispiele



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [186]

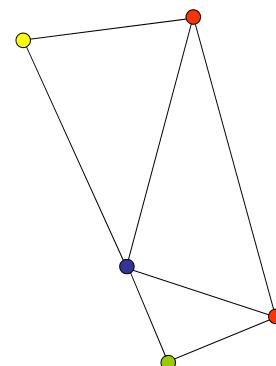


© R. Dörner



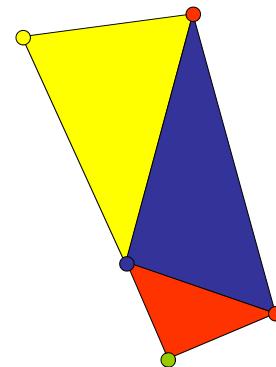
## Gouraud-Shading

- Schritt 1: Flat-Shading durchführen
  - Variante 1: Eckpunktfarben sind vorgegeben: Auswahl einer Eckpunktfarbe für Fläche
  - Variante 2: Phong-Beleuchtungsrechnung für jede Fläche (unter Annahme, dass es eine Normale pro Fläche gibt)



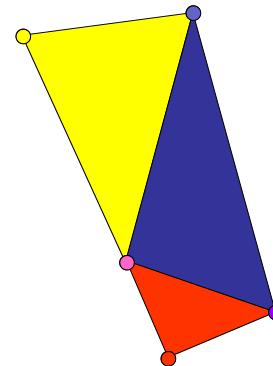
## Gouraud-Shading

- Schritt 1: Flat-Shading durchführen
- Schritt 2: Farben an den Eckpunkten ändern
  - Neue Farbe des Eckpunktes ist Mittelwert (arithmetisches Mittel oder nach Flächeninhalt gewichtetes Mittel) der Farben der angrenzenden Flächen
  - Alternativ: Verzicht auf Schritt 1 und Beleuchtungsrechnung für jeden Eckpunkt mit gemittelter Normale



## Gouraud-Shading

- Schritt 1: Flat-Shading durchführen
- Schritt 2: Farben an den Eckpunkten ändern
  - Neue Farbe des Eckpunktes ist Mittelwert (arithmetisches Mittel oder nach Flächeninhalt gewichtetes Mittel) der Farben der angrenzenden Flächen
  - Alternativ: Verzicht auf Schritt 1 und Beleuchtungsrechnung für jeden Eckpunkt mit gemittelter Normale



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [189]

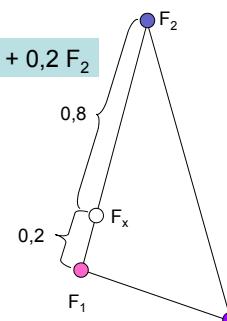


© R. Dörner

## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren

$$F_x = 0,8 F_1 + 0,2 F_2$$



Lineare Interpolation:  
$$F_x = a F_1 + b F_2$$
  
$$a + b = 1$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [190]



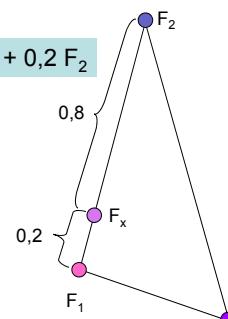
© R. Dörner



## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren

$$F_x = 0,8 F_1 + 0,2 F_2$$

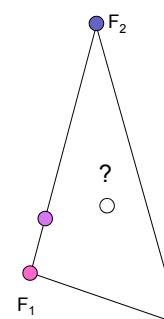


Lineare Interpolation:  
 $F_x = a F_1 + b F_2$   
 $a + b = 1$



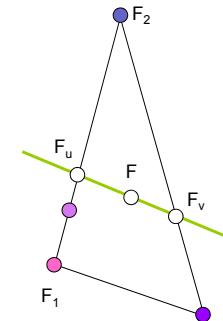
## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren
  - Beliebige Gerade g durch inneren Punkt P führen



## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren
  - Beliebige Gerade g durch inneren Punkt P führen
  - Schnittpunkte mit den Kanten finden
  - Farbe an P durch lineare Interpolation der Farben an den Schnittpunkten ermitteln

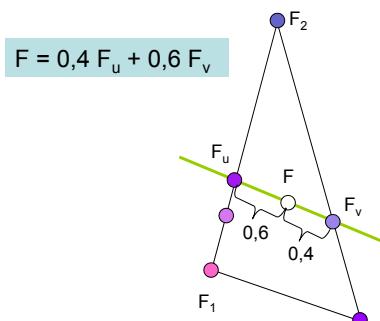


Lineare Interpolation:  
 $F = a F_u + b F_v$   
 $a + b = 1$



## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren
  - Beliebige Gerade g durch inneren Punkt P führen
  - Schnittpunkte mit den Kanten finden
  - Farbe an P durch lineare Interpolation der Farben an den Schnittpunkten ermitteln



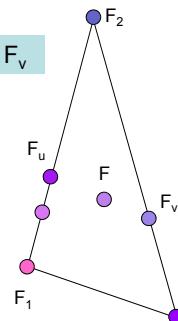
Lineare Interpolation:  
 $F = a F_u + b F_v$   
 $a + b = 1$



## Gouraud-Shading

- Schritt 3: bilineare Interpolation ins Innere durchführen
  - Farben auf den Kanten linear interpolieren
  - Beliebige Gerade g durch inneren Punkt P führen
  - Schnittpunkte mit den Kanten finden
  - Farbe an P durch lineare Interpolation der Farben an den Schnittpunkten ermitteln

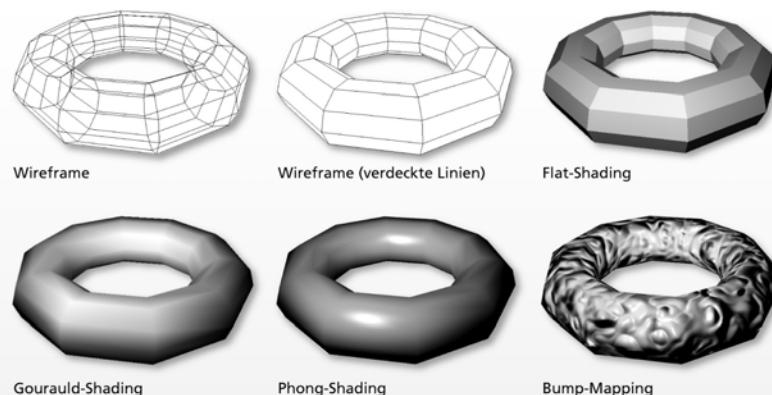
$$F = 0,4 F_u + 0,6 F_v$$



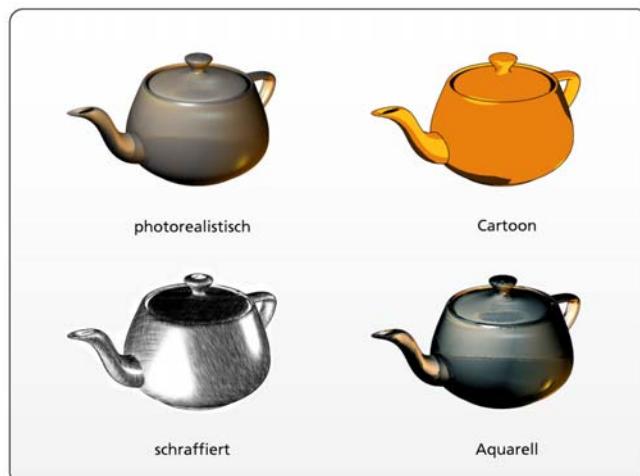
Lineare Interpolation:  
 $F = a F_u + b F_v$   
 $a + b = 1$



## Shading: Beispiele



## Weitere Shading Verfahren



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [197]

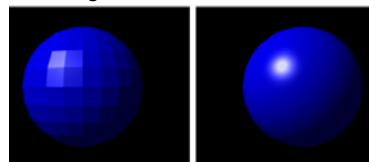


© R. Dörner

## Reduzierung des Shadings

- Glätten durch Shading kann auch unerwünscht sein
- Konturlinien sollen erhalten bleiben (z.B. Nase nicht ins Gesicht hinein glätten)
- Lösung: kein glättendes Shading, wenn Winkel zwischen zwei benachbarten Flächen groß ist
- in VRML: `creaseAngle`

Shading erwünscht:



Quelle: wikipedia.org

Shading unerwünscht:



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [198]



© R. Dörner



## Szenenmodell

- E.1** Zusammensetzen der Szene
- E.2** Shading
- E.3 Hintergrund und Umgebung**
- E.4** Animation und Interaktion



## Hintergrund und Umgebung

- Default – Farbe, mit der Bildspeicher initialisiert wird
- Muss nicht immer schwarz sein: z.B. Farbverlauf oder Textur verwenden: ganze Szene in Kugel oder Würfel packen, Hintergrundbild aufbringen
- Weitere Umgebungseffekte: z.B. Nebel, atmosphärische Effekte
- Auch: Terrain (für große Szenen)

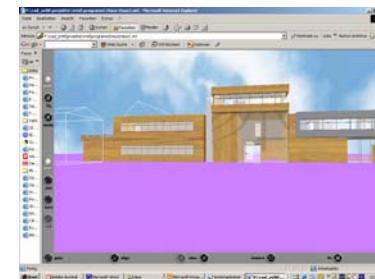


Quelle: johnstejskal.com



## Hintergrundtextur: Wolken

- Würfel der die Szene enthält und von innen mit Bildern (Texturen) beklebt werden kann
- Würfel kann vom User nicht erreicht werden
- Sieht an den Ecken etwas seltsam aus
- Statt große Bilder (ggf. Performance Problem) kann auch ein Farbverlauf gewählt werden



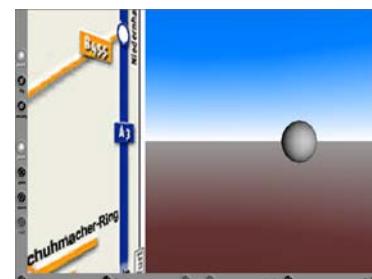
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [201]



## Background in VRML

```
Background{  
    skyColor [ 0.0 0.2 0.7,  
              0.0 0.5 1.0,  
              1.0 1.0 1.0 ]  
    skyAngle [ 1.31, 1.57 ]  
    groundColor [0.1 0.1 0.0,  
                 0.4 0.2 0.2,  
                 0.6 0.6 0.6 ]  
    groundAngle [ 1.31, 1.57 ]  
    leftUrl "image.gif"  
}
```



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [202]

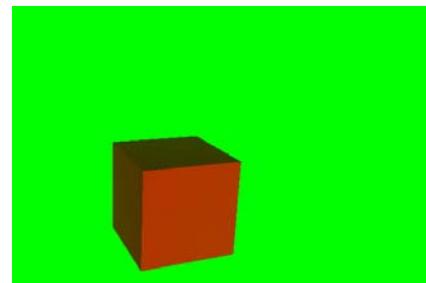


© R. Dörner



## Nebel

- Nebel durch Beimischung einer Nebelfarbe abhängig von der Entfernung
- Wichtig für Realismus
  - Atmosphärische Effekte werden nachgebildet
  - Kaschierung von Darstellungsfehlern in der Ferne
  - Vorziehen der Far-Clipping Plane möglich (Performanzverbesserung)



```
Fog{  
    color      0.8 0.8 0.8  
    fogType   "EXPONENTIAL"  
    visibilityRange 30.0  
}
```



## Szenenmodell

- E.1** Zusammensetzen der Szene
- E.2** Shading
- E.3** Hintergrund und Umgebung
- E.4** Animation und Interaktion



## Animation und Interaktion

- Animation: Veränderung des Bildes über die Zeit
  - Folge von einzelnen Bildern (Frames)
  - Mindestanzahl von Frames pro Sekunde damit Eindruck kontinuierlicher Bewegung entsteht
  - Bilder ändern sich (z.B. Position und Farbe von Objekten)
- Interaktion
  - User kann mit Bild interagieren (z.B. mit Maus anklicken, Kamera-Position verändern)
  - Bild ändert sich als Reaktion auf Benutzeraktion



Quelle: Nintendo

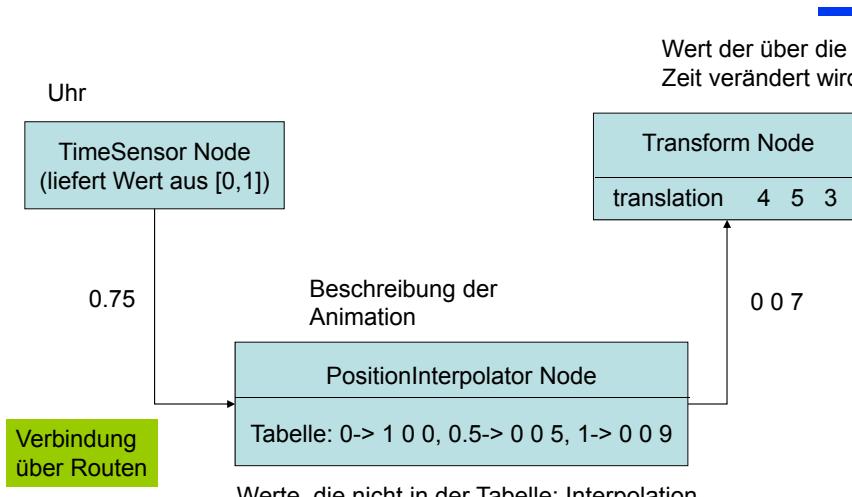
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [205]



© R. Dörner

## Animation in VRML



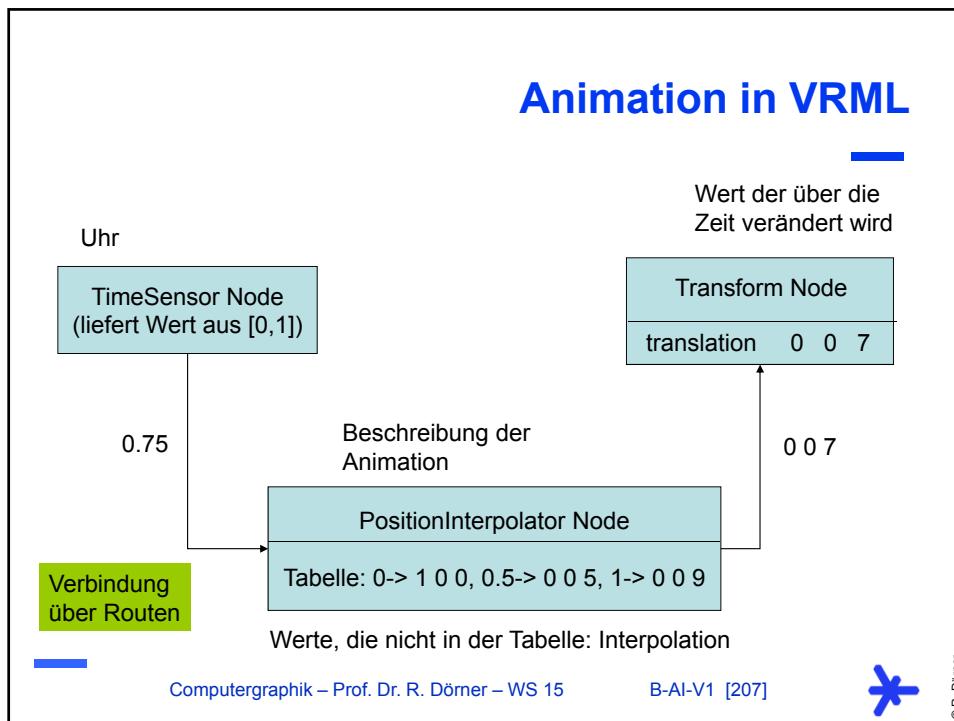
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [206]



© R. Dörner





## Animation in VRML

```
DEF Objekt Transform{ translation 0 0 1
                      children[ bla bla ] }
DEF Uhr TimeSensor{ loop          TRUE
                     cycleInterval 8      }
DEF Anim PositionInterpolator{
    key [ 0.0, 0.5, 1.0]
    keyValue[ 1 0 0, 0 0 5, 0 0 9 ] }
ROUTE Uhr.fraction_changed TO
    Anim.set_fraction
ROUTE Anim.value_changed TO
    Objekt.set_translation
```

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [208]

© R. Dörner



## Interaktion in VRML

*Transform Nodes, Shape Nodes, Interpolator Nodes*

```
DEF Uhr TimeSensor{ enabled FALSE
                      loop TRUE
                      cycleInterval 8 }

DEF Maus TouchSensor{ }

ROUTE Maus.isOver TO Uhr.set_enabled
```

*Restliche Routen für Animation*

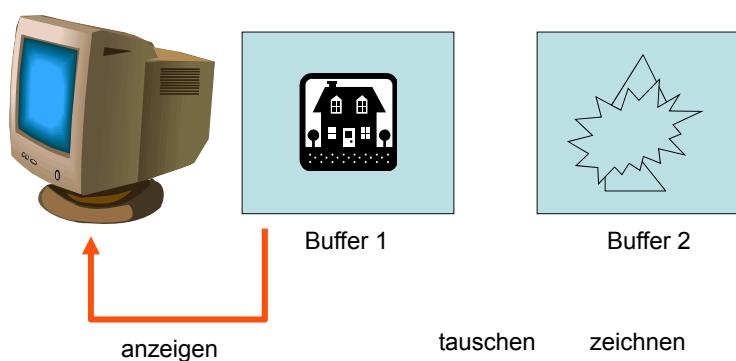
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [209]



© R. Dörner

## Double Buffering



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [210]



© R. Dörner



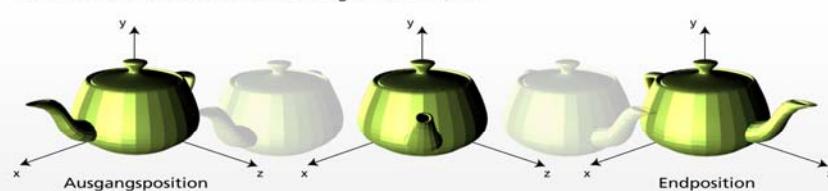
## Probleme mit Eulerwinkel

- Eulerwinkel bergen zwei Probleme bei Animationen
  - Mehrdeutigkeit, Schwierigkeit bei Interpolation für Animationen
  - Gimbal Lock

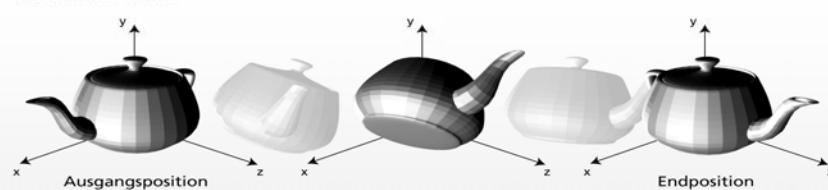


## Mehrdeutigkeit

Erwarteter Ablauf bei einer 90°-Drehung um die Y-Achse



Tatsächlicher Ablauf



## Gimbal Lock

The diagram illustrates Gimbal Lock through two cases involving a teapot's rotation.

**1. Fall:** Shows a yellow teapot's rotation sequence. It starts at "Ausgangsposition" (initial position), rotates 20° around the X-axis ("20° Drehung um die X-Achse") to a intermediate position, and then rotates 90° around the Y-axis ("90° Drehung um die Y-Achse") to the "Endposition" (final position). The intermediate position shows the teapot's body rotated 90° relative to its initial orientation, which can lead to a loss of one degree of freedom.

**2. Fall:** Shows a blue teapot's rotation sequence. It starts at "Ausgangsposition", rotates 90° around the Y-axis ("90° Drehung um die Y-Achse") to an intermediate position, and then rotates 20° around the Z-axis ("20° Drehung um die Z-Achse") to the "Endposition". Similar to the first case, it highlights how specific rotation orders can result in Gimbal Lock.

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [213]

© R. Dörner

## Probleme mit Eulerwinkel

- Eulerwinkel bergen zwei Probleme bei Animationen
  - Mehrdeutigkeit, Schwierigkeit bei Interpolation für Animationen
  - Gimbal Lock
- Lösung: Statt Spezifikation einer Rotation durch 3 Winkel ist es besser einen Winkel und eine Rotationsachse zu verwenden
- So macht es auch VRML, z.B.  
`rotation 2.0 2.0 0.0 3.14`

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [214]

© R. Dörner



## Winkel und Rotationsachse

- Wie kann man mit Winkel  $\alpha$  und Rotationsachse  $(u,v,w)^T$  rechnen?  
Welche Transformationsmatrix gehört dazu?
- Lösung 1
  - Zerlegung der Transformation in Rotationen um die x-Achse, y-Achse und z-Achse
  - Problem: Winkel der Teilrotationen müssen bestimmt werden
  - Problem: Haben dann wieder Eulerwinkel
- Lösung 2
  - Verwendung einer allgemeinen Formel für die Transformationsmatrix



## Winkel und Rotationsachse

$$R(\alpha, u, v, w) =$$

$$\begin{bmatrix} (1 - \cos \alpha)u^2 + \cos \alpha & (1 - \cos \alpha)uv + w \sin \alpha & (1 - \cos \alpha)uw + v \sin \alpha & 0 \\ (1 - \cos \alpha)uv + w \sin \alpha & (1 - \cos \alpha)v^2 + \cos \alpha & (1 - \cos \alpha)vw + u \sin \alpha & 0 \\ (1 - \cos \alpha)uw + v \sin \alpha & (1 - \cos \alpha)vw + u \sin \alpha & (1 - \cos \alpha)w^2 + \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



## Winkel und Rotationsachse

- Wie kann man mit Winkel  $\alpha$  und Rotationsachse  $(u,v,w)^T$  rechnen?  
Welche Transformationsmatrix gehört dazu?
- Lösung 1
  - Zerlegung der Transformation in Rotationen um die x-Achse, y-Achse und z-Achse
  - Problem: Winkel der Teilrotationen müssen bestimmt werden
  - Problem: Haben dann wieder Eulerwinkel
- Lösung 2
  - Verwendung einer allgemeinen Formel für die Transformationsmatrix
- Lösung 3
  - Verwendung von Quaternionen, d.h. einem Tupel, das aus einem Skalar und einem 3D Vektor besteht

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [217]



© R. Dörner

## Rechenregeln für Quaternionen

Schreibweise eines Quaternionen:

$$\hat{q} = (\alpha, u, v, w) = (\alpha, \begin{pmatrix} u \\ v \\ w \end{pmatrix}) = (\alpha, \vec{r})$$

Addition von zwei Quaternionen:

$$\hat{q}_1 + \hat{q}_2 = (\alpha, \vec{r}) + (\beta, \vec{s}) = (\alpha + \beta, \vec{r} + \vec{s})$$

Betrag eines Quaternionen:

$$|\hat{q}| = \sqrt{\alpha^2 + \langle \vec{r}, \vec{r} \rangle}$$

Multiplikation von zwei Quaternionen:

$$\begin{aligned} \hat{q}_1 \cdot \hat{q}_2 &= (\alpha, \vec{r}) \cdot (\beta, \vec{s}) \\ &= (\alpha \cdot \beta - \langle \vec{r}, \vec{s} \rangle, \alpha \cdot \vec{s} + \beta \cdot \vec{r} + \vec{r} \times \vec{s}) \end{aligned}$$

Inverses eines Quaternionen:

$$\hat{q}^{-1} = \frac{1}{|\hat{q}|^2} \cdot (\alpha, -\vec{r})$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [218]



© R. Dörner



## Rotieren mit Quaternione

- Wir wollen Punkt  $P(x,y,z)$  rotieren um Achse  $(u,v,w)$  um Winkel  $\alpha$ . Gesucht ist  $P'(x',y',z')$
- Wir definieren folgende Quaternionen:

$$\hat{a} = (0, x, y, z) = \left(0, \begin{pmatrix} x \\ y \\ z \end{pmatrix}\right) \quad \hat{b} = \left(\cos \frac{\alpha}{2}, \frac{1}{\sqrt{u^2 + v^2 + w^2}} \cdot \sin \frac{\alpha}{2} \begin{pmatrix} u \\ v \\ w \end{pmatrix}\right)$$

- Wir berechnen

$$\hat{c} = \hat{b} \cdot \hat{a} \cdot \hat{b}^{-1} = \left(0, \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}\right)$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

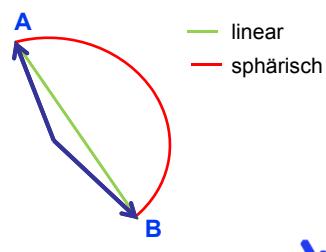
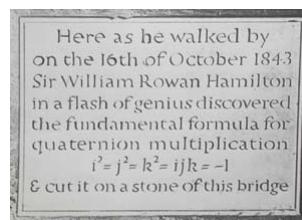
B-AI-V1 [219]



© R. Dörner

## Vorteile von Quaternionen

- Effizienz: Weniger Additionen und Multiplikationen werden benötigt als bei Transformationsmatrizen
- Konkatenation ist unabhängig vom transformierten Punkt realisierbar (wie bei Transformationsmatrizen auch)
- Vorteile bei der Hardware-Implementierung
- Sphärische lineare Interpolation (zwischen zwei Quaternionen) kann mit dem slerp-Algorithmus gut realisiert werden: weiche, mit gleichmäßiger Geschwindigkeit ablaufende Animation auf Großkreisen einer 4D Hyperkugel (nicht für Kamera-Animation geeignet, da Roll eingeführt wird)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [220]



© R. Dörner



—  
**Teil F:  
Objektmodelle**  
—

**Objektmodelle**

- F.1** Erstellung von Objektmodellen
- F.2** Farbe und Textur
- F.3** Polygonnetze
- F.4** Kurven und Patches
- F.5** Weitere Methoden der Objektmodellierung



## Zweck von Objektmodellen

- Für den Autor
  - Spezifikation der einzelnen Objekte, die in der Szene vorhanden sind
  - Ausgangspunkt für den Aufbau eines Szenenmodells
- Für den Computer
  - Formale Repräsentation von Daten über Objekte der Szene
  - Überführung von einem Objektmodell in ein anderes ist möglich
  - Ausgangspunkt für das Rendering

```
#VRML V2.0 utf8
Shape{
    appearance Appearance{
        material Material {}
    }
    geometry IndexedFaceSet{
        coord Coordinate{
            point [
                0 0 1,
                1.5 0 0,
                0 3.5 0,
                0 0 0
            ]
        }
        coordIndex [ 0 1 2 -1 3 0 2 -1
                    2 1 3 -1 3 1 0 ]
    }
}
```



## Erstellung von Objektmodellen

Erzeugen (z.B. Eintippen) eines Textes in einer formalen Sprache (Beispiel: VRML), häufig Berechnung von Koordinaten oder Parametern

Verwendung von speziellen Werkzeugen (Modeller)

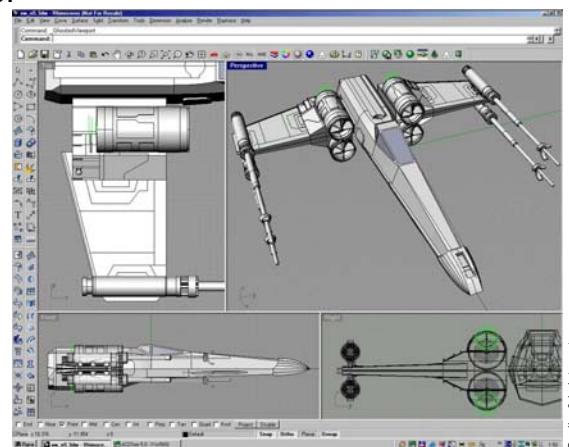


## Erstellung von Objektmodellen

Modellierwerkzeuge:

Beispiele:

- 3D Studio Max
- Rhinoceros
- Alias Maya
- AC3D
- Wings3D
- SoftCAD 3D
- Blender
- ...



Quelle: 3dprinter.net

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [225]



© R. Dörner

## Erstellung von Objektmodellen

Erzeugen (z.B. Eintippen)  
eines Textes in einer  
formalen Sprache (Beispiel:  
VRML), häufig Berechnung  
von Koordinaten oder  
Parametern

Verwendung von speziellen  
Werkzeugen (Modelle)

Rekonstruktion eines  
Objektmodells aus Bildern  
aus unterschiedlicher  
Perspektive (⇒  
Bildverarbeitung, Computer  
Vision)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [226]

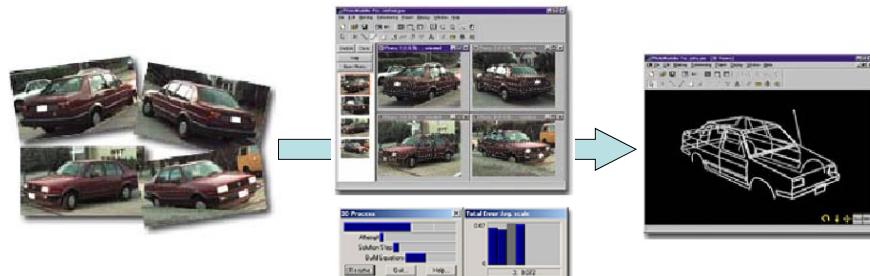


© R. Dörner



## Erstellung von Objektmodellen

3D Rekonstruktion:



Quelle:  
[www.photomodeler.com](http://www.photomodeler.com)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [227]



© R. Dörner

## Erstellung von Objektmodellen

Erzeugen (z.B. Eintippen) eines Textes in einer formalen Sprache (Beispiel: VRML), häufig Berechnung von Koordinaten oder Parametern

Verwendung von speziellen Werkzeugen (Modeller)

Rekonstruktion eines Objektmodells aus Bildern aus unterschiedlicher Perspektive (⇒ Bildverarbeitung, Computer Vision)

Einsatz von speziellen Scan – Geräten und Algorithmen, die Objektmodell aus der Scan – Information erzeugen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [228]



© R. Dörner



## Erstellung von Objektmodellen

3D Scanner:



Quelle:  
Konica Minolta  
Europe GmbH

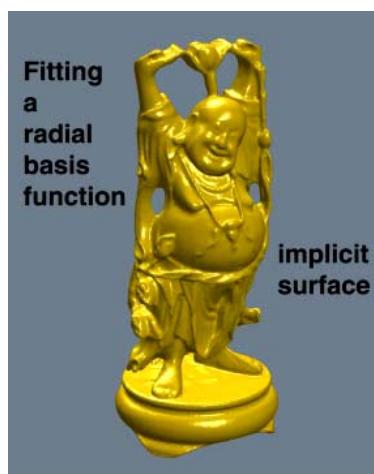
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [229]



© R. Dörner

## Erstellung von Objektmodellen



Quelle:  
R.K. Beatson, T.J. Mitchell,  
ARANZ

Computergraphik – Prof. Dr. R. Dörner – WS 15

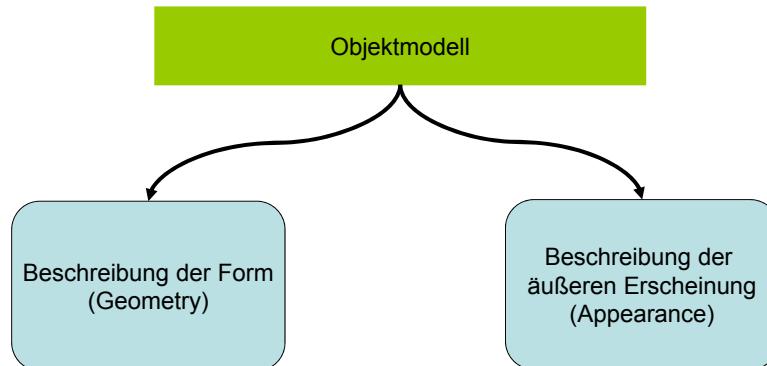
B-AI-V1 [230]



© R. Dörner



## Umfang von Objektmodellen



## Geometry

- Beschreibung der Form des Objekts
- Häufig:
  - Verwendung von **Parametern** (z.B. Radius einer Kugel)
  - Angabe von Koordinaten in einem **lokalen Koordinatensystem** (**Objektkoordinaten**)



## Appearance

- Beschreibung der äußereren Erscheinung eines Objekts:
    - Farbe
    - Oberflächenstruktur (rauh? glatt?)
    - Matt oder glänzend?
    - Spiegelnd?
    - Durchsichtig (transparent)?
    - Durchscheinend (transluzent)?
    - Oberflächeneffekte (z.B. metallisch)?
- } ⇒ Parameter von Beleuchtungsmodellen, erweiterte Farbmodelle



## Objektmodelle

- F.1 Erstellung von Objektmodellen**
- F.2 Farbe und Textur**
- F.3 Polygonnetze**
- F.4 Kurven und Patches**
- F.5 Weitere Methoden der Objektmodellierung**



## Farbmodelle

- RGB-Modell mit Grundfarben Rot, Grün und Blau
- Farbe wird als Tripel  $(r, g, b)$  spezifiziert
  - Wert von  $r, g, b$  liegt zwischen 0 und 1
  - Gibt einen Anteil an (auch 0 bis 255 üblich, mit  $255 \cong 1$ )
- Beispiel:
  - $(0,0,0) \cong$  schwarz
  - $(1,1,1) \cong$  weiß
  - $r = b = g \cong$  grau

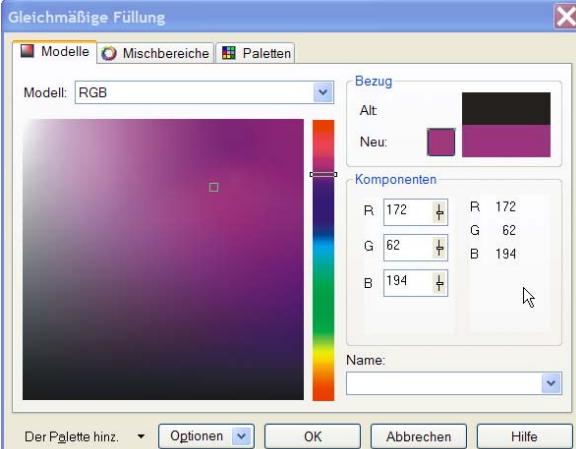


## Farbmodelle

- Problem bei dem RGB Farbmodell:
- 
- Die Illustration zeigt einen stilisierten menschlichen Kopf in profile. Von seinem Mund führt ein Pfeil zu einer lila Quadratfläche, die in einem ovalen Rahmen dargestellt ist. Eine Sprachblase neben dem Kopf enthält die Farbwerte '(204, 51, 153)'.
- RGB ist ein Geräte-Farbmodell
  - Es gibt auch andere Farbmodelle, z.B. Wahrnehmungs-Farbmodelle, mit denen Menschen einfacher Farbe spezifizieren können



## Farbmodelle



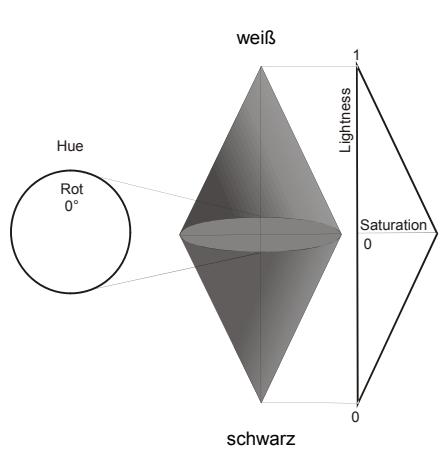
Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [237]

© R. Dörner

## Farbmodelle

HLS System ist Wahrnehmungs-Farbmodell

- Hue: Farbton, spezifiziert als Winkel im Farbkreis
- Lightness: Helligkeit, spezifiziert als Wert aus  $[0,1]$
- Saturation: Sättigung, spezifiziert als Wert aus  $[0,1]$



Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [238]

© R. Dörner

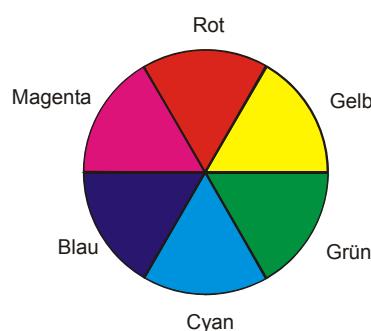


## Farbmodelle

Farbkreise:



Farbkreis von Goethe



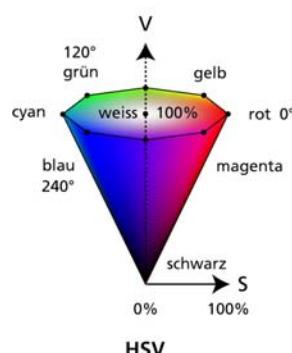
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [239]

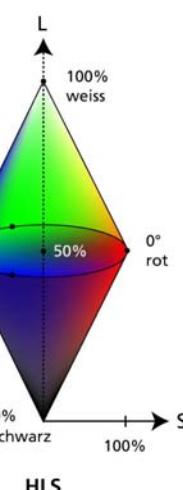


© R. Dörner

## HSV und HLS



Computergraphik – Prof. Dr. R. Dörner – WS 15



B-AI-V1 [240]



© R. Dörner



## Farbmodelle

Umrechnung **RGB** nach **HLS**:

```
a=max(R,G,B); b=min(R,G,B); c=a-b; L=0,5*(a+b);
if (c==0)
    S=0; H="undefined"; exit;
if (L<0.5)
    S=c/(a+b);
else
    S=c/(2-a-b);
r=(a-R)/c; g=(a-G)/c; b=(a-B)/c;
if (R==a)
    H=b-g;
else if (G==a)
    H=2+r-b;
else
    H=4+g-r;
H=60*H;
if (H<0)
    H=H+360;
exit;
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [241]

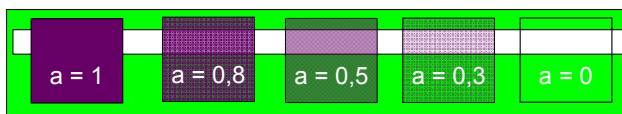


© R. Dörner

## Erweiterte Farbmodelle

- RGB – Modell wird um einen Wert *Alpha* (geschrieben *a* oder  $\alpha$ ) erweitert: RGBA
- Alpha-Kanal gibt Transparenz an:  $a=1$  bedeutet keine Transparenz,  $a=0$  bedeutet volle Transparenz

- Beispiel:



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [242]



© R. Dörner



## Erweiterte Farbmodelle

- Gegeben: RGBA – Farbe  $(r, g, b, a)$  und Hintergrundfarbe als RGB-Wert  $(r^*, g^*, b^*)$
- Es ergibt sich folgender Farbwert mittels Farbmischung (durch lineare Interpolation):

$$a \cdot (r, g, b) + (1 - a) \cdot (r^*, g^*, b^*)$$



addiert sich zu 100%



## Beispiel: VRML

- Angabe der Farbe in Feldern des Material Nodes
- Material Node ist Wert eines Feldes im Appearance Node

```
Material{  
    diffuseColor    1 0 0  
    transparency   0.3  
}
```



## Texturen

- Idee: als Eingabe für die Erzeugung von Bildern schon bestehende Bilder verwenden
- Motivation:
  - Objektmodell wird einfacher (Bsp. Koordinaten aller Grashalme einer Wiese eingeben vs. Foto einer Wiese machen)
  - Anzahl der Eckpunkte wird reduziert (wichtig für die Geschwindigkeit des Renderings)
  - Visuelle Qualität wird erhöht



Quelle: www.vstep.nl

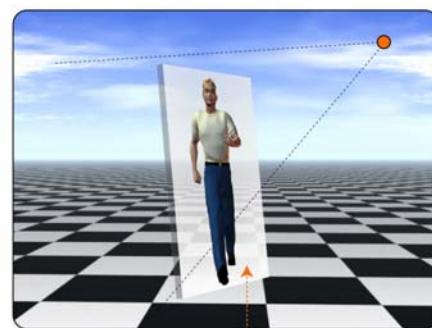
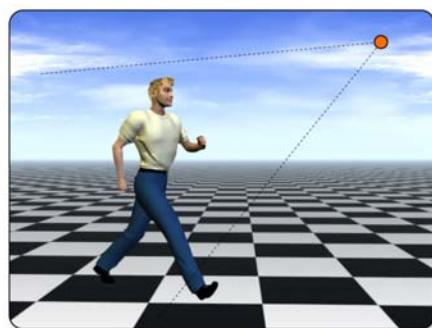
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [245]



© R. Dörner

## Texturen: Beispiel Billboard



Projection plane through object

Billboard wird automatisch so gedreht,  
dass es senkrecht zum Betrachter steht

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [246]



© R. Dörner



## Textur - Quellen

- Fotoapparat
- Computer-generiertes Bild
  - Insbesondere: Multi-Pass-Rendering
- Allgemeine Berechnung
  - Textur ist nichts anderes als ein mehrdimensionales Feld von Werten (müssen keine Farbwerte sein, vgl. Bump Mapping)
  - Prozedurale Texturen



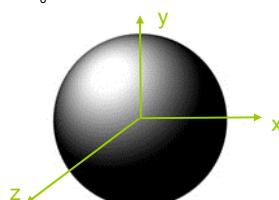
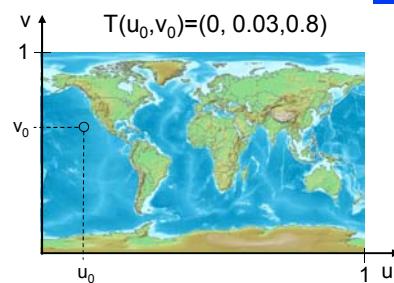
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [247]

© R. Dörner

## Texturmapping

- Gegeben
  - Textur (im Texturkoordinatensystem, üblich normiert auf  $[0, 1]$ ): jedem Wert in  $[0, 1] \times [0, 1]$  wird ein Wert aus dem Wertebereich (hier: RGB-Farbraum) zugeordnet
  - 3D Objekt (in Raumkoordinaten), das texturiert werden soll
- Problem: wie kann man angeben, wie die Textur auf das Objekt aufgebracht werden soll?

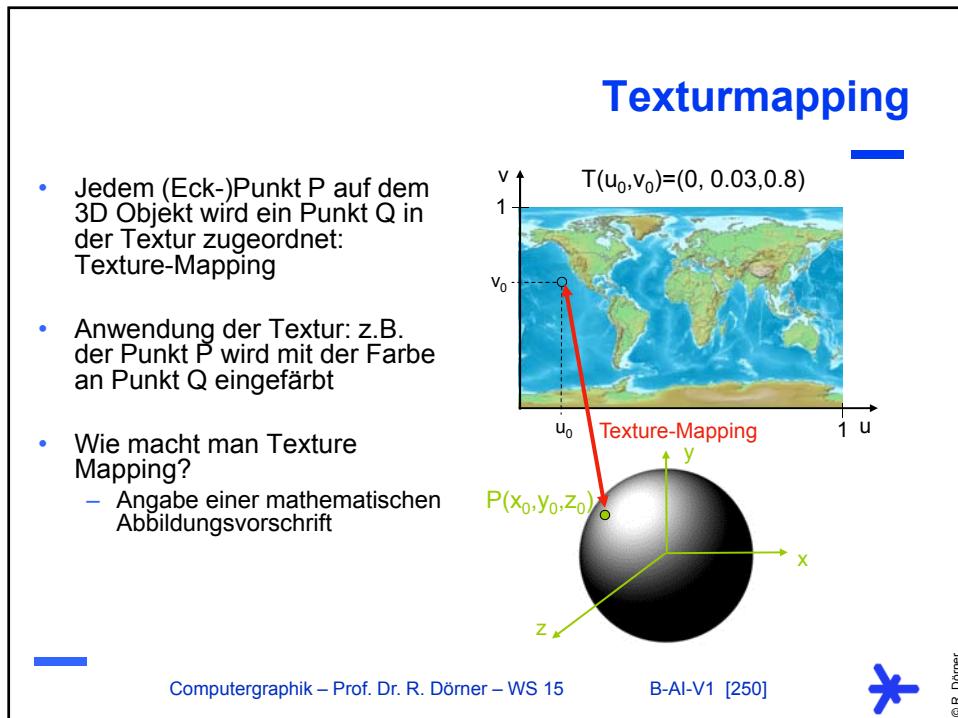
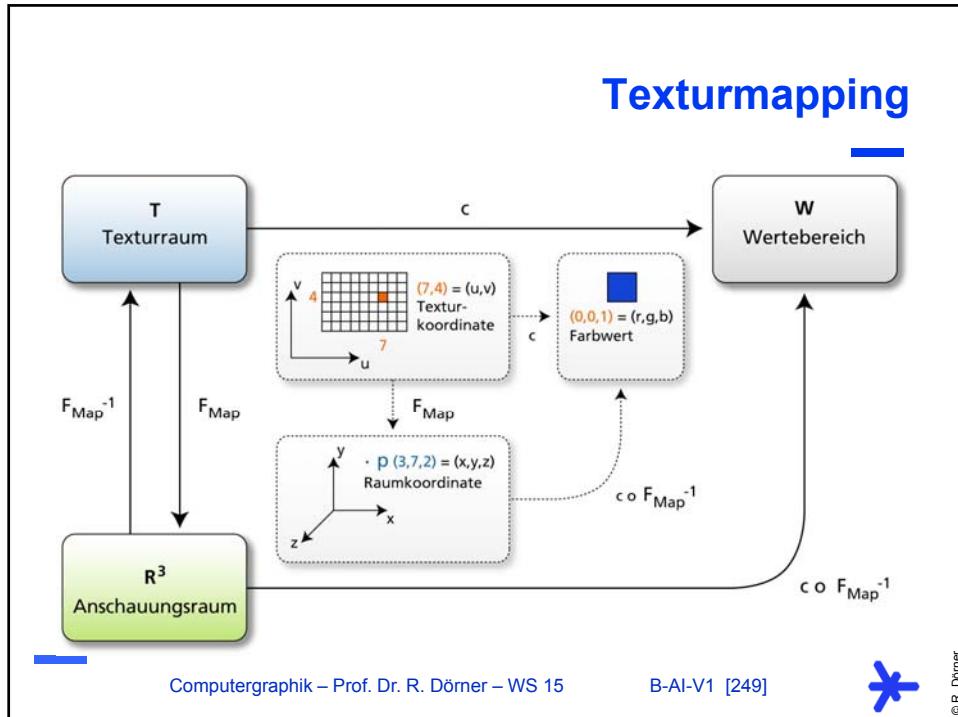


Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [248]

© R. Dörner





## Texturmapping

$$F(u, v) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = r \cdot \begin{pmatrix} \sin(2\pi \cdot u) \cdot \cos(2\pi \cdot v) \\ \sin(2\pi \cdot u) \cdot \sin(2\pi \cdot v) \\ \cos(2\pi \cdot u) \end{pmatrix},$$

$(u, v) \in [0,1] \times [0,1]$

$T(u_0, v_0) = (0, 0.03, 0.8)$

$P(x_0, y_0, z_0)$

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [251]

\* © R. Dörner

## Texturmapping

- Jedem (Eck-)Punkt P auf dem 3D Objekt wird ein Punkt Q in der Textur zugeordnet: Texture-Mapping
- Anwendung der Textur: z.B. der Punkt P wird mit der Farbe an Punkt Q eingefärbt
- Wie macht man Texture Mapping?
  - Angabe einer mathematischen Abbildungsvorschrift
  - Verwendung Hüllkörper

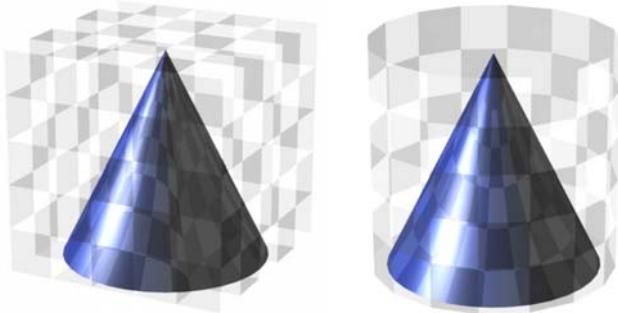
$T(u_0, v_0) = (0, 0.03, 0.8)$

$P(x_0, y_0, z_0)$

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [252]

\* © R. Dörner

## Hüllkörper beim Texturmapping

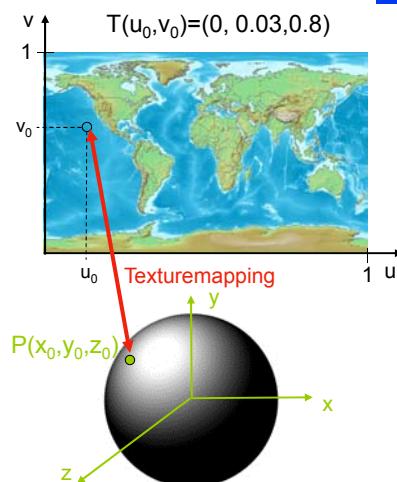


$$F(u, v) = \begin{pmatrix} r \cdot \cos 2\pi \cdot u \\ r \cdot \sin 2\pi \cdot u \\ h \cdot v \end{pmatrix},$$
$$(u, v) \in [0,1] \times [0,1]$$

1. Hüllkörper so wählen, dass mathematische Formel bekannt
2. Projektion vom texturierten Hüllkörper in das Innere

## Texturmapping

- Jedem (Eck-)Punkt P auf dem 3D Objekt wird ein Punkt Q in der Textur zugeordnet:  
Texturmapping
- Anwendung der Textur: z.B.  
der Punkt P wird mit der Farbe  
an Punkt Q eingefärbt
- Wie macht man Textur-  
mapping?
  - Angabe einer mathematischen  
Abbildungsvorschrift
  - Verwendung Hüllkörper
  - Angabe der Texturkoordinaten  
direkt pro (Eck-) Punkt im  
Sinne einer Wertetabelle



## Angabe von Texturkoordinaten

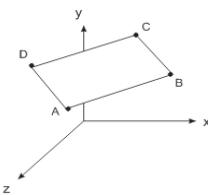
- Sind ein Koordinatensystem
  - in der Textur (in der Regel im Einheitsintervall)
  - auf der Oberfläche des zu texturierenden Objektes
- 3D Objekt hat 2D Oberfläche, Texturkoordinaten daher in der Regel 2D
- Geben an welcher Punkt der Textur an welchem Punkt der Oberfläche abgebildet wird

```
IndexedFaceSet{  
    ... # Definition Viereck  
  
    texCoord TextureCoordinate{  
        point[ 0.0 0.0,  
               0.0 0.8,  
               1.0 0.0,  
               0.8 0.7 ]  
    }  
  
    texCoordIndex [0 1 3 2]  
}
```



## Beispiel

3D-Welt:



Punkt A: 0.5, 0.75  
Punkt B: 0.5, 1.0  
Punkt C: 0.0, 1.0  
Punkt D: 0.0, 0.75

Texturraum:

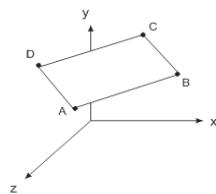


Resultierende Texturierung:



## Beispiel

3D-Welt:

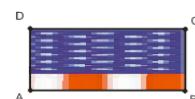


Texturraum:



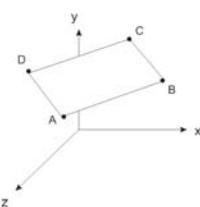
Punkt A: 0.5, 0.75  
Punkt B: 0.5, 1.0  
Punkt C: 0.0, 1.0  
Punkt D: 0.0, 0.75

Resultierende Texturierung:



## Vergrößerung des Texturraums

3D-Welt:



Texturraum:



Punkt A: 1.5, 2.0  
Punkt B: 0.0, 2.0  
Punkt C: 0.0, 0.0  
Punkt D: 1.5, 0.0

Resultierende Texturierung:



## Vergrößerung des Texturraums

3D-Welt:

Punkt A: 1.5, 2.0  
Punkt B: 0.0, 2.0  
Punkt C: 0.0, 0.0  
Punkt D: 1.5, 0.0

Texturraum:

Resultierende Texturierung:

clamp repeat

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [259]

© R. Dörner

## Vergrößerung des Texturraums

3D-Welt:

Punkt A: 1.5, 2.0  
Punkt B: 0.0, 2.0  
Punkt C: 0.0, 0.0  
Punkt D: 1.5, 0.0

Texturraum:

Resultierende Texturierung:

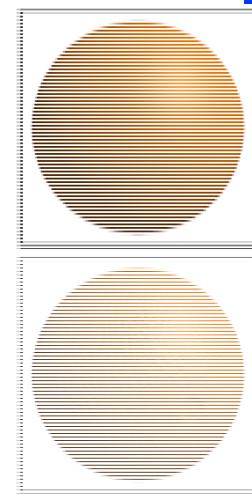
clamp repeat

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [260]

© R. Dörner

## Bump Mapping

- Beobachtung: Position der Normale bestimmt durch das Shading das Aussehen der Oberfläche
- Idee: Normale über eine Textur verändern (die Textur bestimmt also nicht die Farbe an einem Punkt der Oberfläche, sondern wie die Normale verändert wird)
- Vorteil: Oberflächenstruktur (z.B. kleine Dellen) müssen nicht modelliert werden. Im Gegensatz zu einer Textur unterliegen sie der Beleuchtungsrechnung (sehen also z.B. beim Ändern der Lichtposition anders aus).



Computergraphik – Prof. Dr. R. Dörner – WS 15

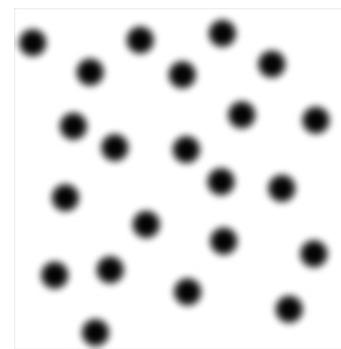
B-AI-V1 [261]



© R. Dörner

## Bump Mapping

Quelle:  
Watt / Pollicarpo  
3D Games



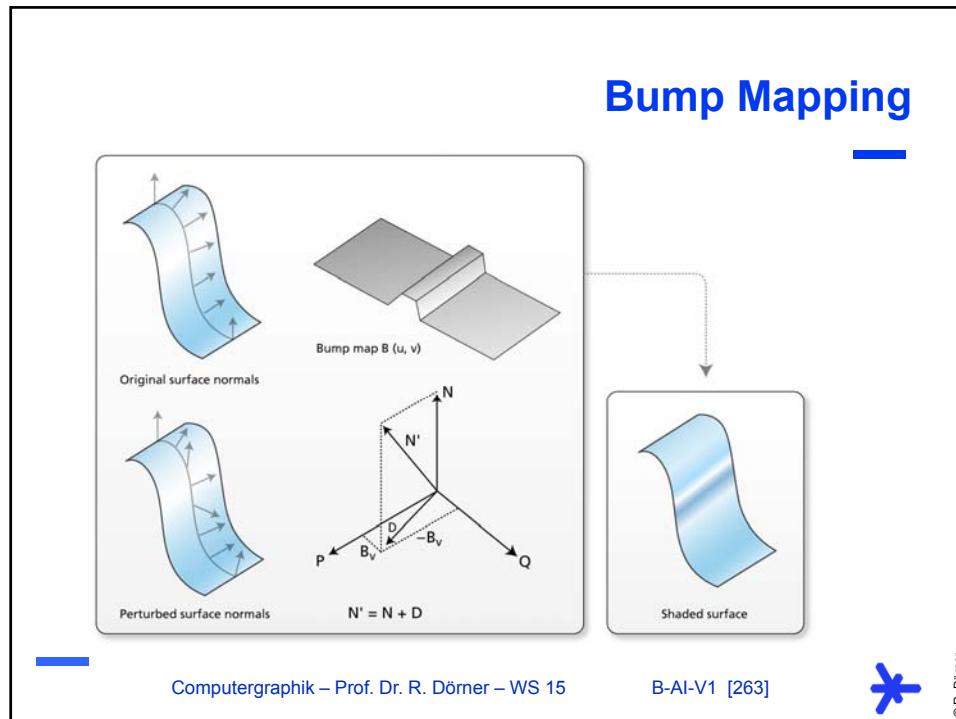
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [262]



© R. Dörner





## Texturarten

- 1D Texturen
  - z.B. Isolinien, Strichlierungen
- 2D Texturen
- 3D Texturen (Bildstapel)
  - z.B. Computertomographie
- Environment Maps
  - Trick, um Reflektionen zu erzeugen
  - Kubisch oder Sphärisch
- Bump Texturen
  - Veränderung der Normalen

Quelle: nVIDIA Corp., siehe auch [http://developer.nvidia.com/object/cube\\_map\\_ogl\\_tutorial.html](http://developer.nvidia.com/object/cube_map_ogl_tutorial.html)

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [264]

© R. Dörner



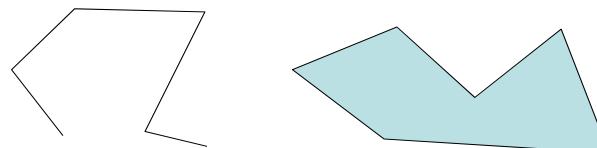
## Objektmodelle

- F.1 Erstellung von Objektmodellen
- F.2 Farbe und Textur
- F.3 Polygonnetze**
- F.4 Kurven und Patches
- F.5 Weitere Methoden der Objektmodellierung



## Polygone und Polygonzüge

- Spezifiziert durch
  - Koordinaten der Eckpunkte
  - Angabe, welcher Punkt mit welchem verbunden ist
  - Beschreibung von Linien und Flächen



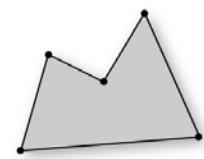
- Offene vs. geschlossene Polygone



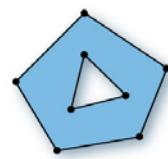
## Polygonflächen

- Geschlossene Polygonzüge bilden Polygonflächen

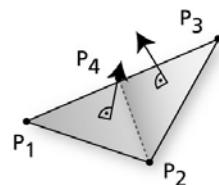
- Sonderfälle:



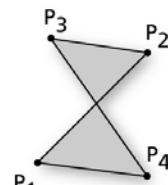
Fläche mit  
"Einbuchtung"  
(konkav Fläche)



Fläche mit "Loch"



Polygon, das  
nicht in einer  
Ebene liegt



selbst-  
schneidendes  
Polygon

Computergraphik – Prof. Dr. R. Dörner – WS 15

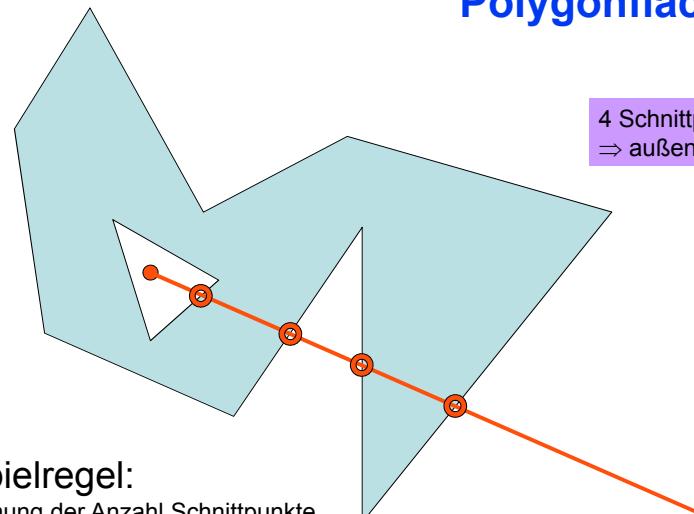
B-AI-V1 [267]



© R. Dörner

## Polygonflächen

4 Schnittpunkte  
⇒ außen



### Beispielregel:

Bestimmung der Anzahl Schnittpunkte  
mit dem Polygonzug

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [268]



© R. Dörner



## Polygonflächen

Beispielregel:  
Bestimmung der Anzahl Schnittpunkte mit dem Polygonzug

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [269]

© R. Dörner

## Polygonflächen

- Sonderfälle können das Rendering erschweren
- Lösung: Ausschließliche Verwendung von Polygonzügen mit nur 3 Eckpunkten (Dreiecke)
  - Dreiecke sind immer konvex
  - Dreiecke sind immer eben
  - Dreiecke haben keine Löcher
  - Alle Polygone sind gleich aufgebaut (weitere Vereinfachung im Rendering)

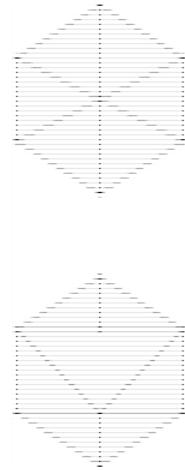
Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [270]

© R. Dörner



## Polygonflächen

- Jede Polygonfläche kann in Dreiecke zerlegt werden (Triangulierung)
- Die Triangulierung ist nicht eindeutig
- Es gibt Algorithmen, um Triangulierung automatisiert vorzunehmen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [271]



© R. Dörner

## Polygonflächen

### Algorithmen zur Triangulierung

- Beispiel: DeLaunay
  - Basiert auf Voronoi Zerlegung: Verbindung von Punktenpaaren mit benachbarten Voronoi - Gebieten
  - Umkreis jeden Dreiecks enthält keine anderen Punkte
  - Varianz der Seitenlänge ist minimiert
  - Optimal bzgl. Vermeidung von spitzen Winkeln
  - Bei  $n$  Punkten:  $O(n \log n)$ , für spezielle Punktverteilungen  $O(n)$
  - Verschiedene Realisierungen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [272]

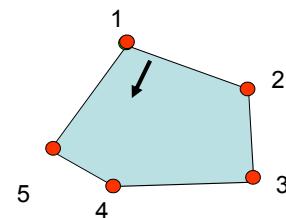


© R. Dörner



## Polygonflächen

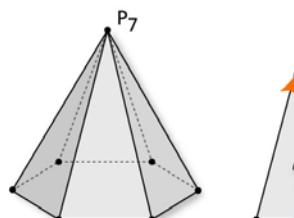
- Man kann bei einer Polygonfläche Innenseite und Außenseite unterscheiden
- Festlegung durch
  - Konvention / Regel (z.B. Angabe des Umlaufsinns)
  - Angabe der Flächennormalen
- Häufiger Fehler: Objekte haben „Löcher“



Beispielregel:  
Umlaufsinn „rechts“



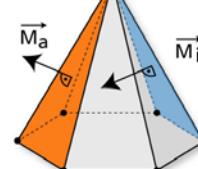
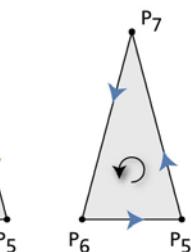
## Polygonflächen: Innenseite und Außenseite



6 → 7 → 5



6 → 5 → 7



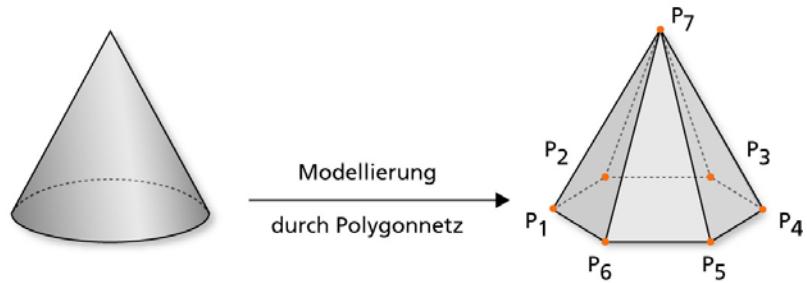
$\vec{M}_a$ : Normale auf der Außenseite  
 $\vec{M}_i$ : Normale auf der Innenseite

In diesem Beispiel ist die Konvention (anders als bei VRML):  
Im Uhrzeigersinn ist außen



## Näherungen

- Polygonnetze können manche Objekte nur Näherungsweise beschreiben



Computergraphik – Prof. Dr. R. Dörner – WS 15

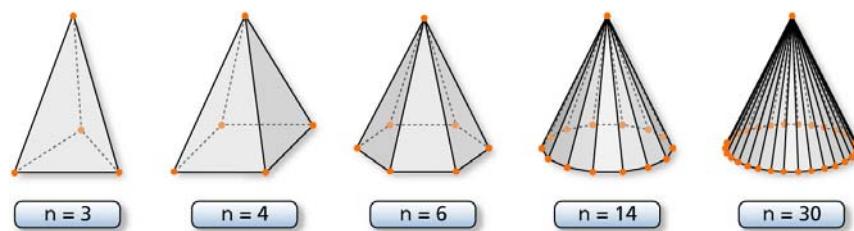
B-AI-V1 [275]



© R. Dörner

## Näherungen

- Näherungen können beliebig gut durchgeführt werden, ABER: hohes Datenvolumen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [276]

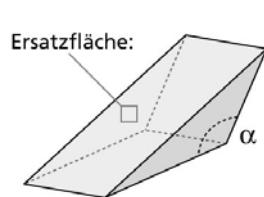


© R. Dörner

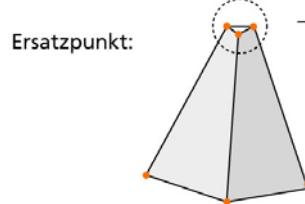


## Näherungen

- Polygonnetze können mit speziellen Algorithmen simplifiziert werden



Kriterium: kleiner Winkel  $\alpha$



Kriterium: Eckpunkt weniger als  $\alpha$  voneinander entfernt

Computergraphik – Prof. Dr. R. Dörner – WS 15

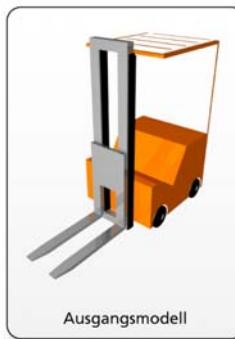
B-AI-V1 [277]



© R. Dörner

## Näherungen

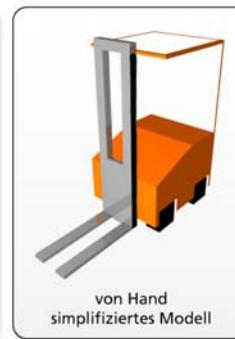
- In manchen Fällen ist Simplifizierung von Hand notwendig (Bewahrung von Semantik)
- Idee: für ein Objekt *mehrere* Objektmodelle anfertigen mit unterschiedlicher Näherung / Detaillierung (Levels-of-Detail, LOD)



Ausgangsmodell



automatisch simplifiziertes Modell



von Hand simplifiziertes Modell

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [278]



© R. Dörner



## LOD

The diagram illustrates the concept of Level of Detail (LOD) in computer graphics. On the left, a full 3D model of a forklift truck is shown on a road, labeled with circled numbers 1, 2, and 3. To the right, six smaller boxes show progressively simplified versions of the forklift at each level: 1 shows the full truck; 2 shows the truck without the forks; 3 shows only the forks; 4 shows a simple cube; 5 shows a wireframe version; and 6 shows a very low-polygon version. A blue asterisk icon is in the bottom right corner.

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [279]

© R. Dörner

## Näherungen

A 3D rendering of a man in a dark suit, white shirt, and red tie, standing with his arms slightly out. The model is shown against a black background. To the left, text explains that visual quality can be improved by shading polygon meshes. A blue asterisk icon is in the bottom right corner.

Visuelle Qualität der Näherung mit Polygonnetzen kann mittels Shading erheblich verbessert werden

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [280]

© R. Dörner



## Vorteile und Nachteile

- ☺ Effiziente Repräsentation im Rechner
- ☺ Existenz effizienter Renderingalgorithmen
- ☺ Umsetzung von Algorithmen in Hardware verfügbar
- ☺ Durch Triangulierung keine Sonderfälle, einfache Grundeinheit
- ☹ Bei einigen Objekten nur Näherungen möglich
- ☹ Hohes Datenvolumen
- ☹ Polygonnetze sind vom Autor schwer und mühselig zu beschreiben
- ☹ Schwer zu verändern: Manipulation vieler Punkte, Einfügen neuer Punkte



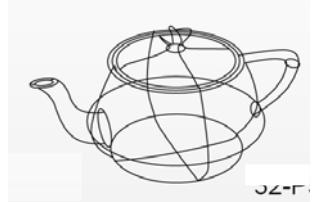
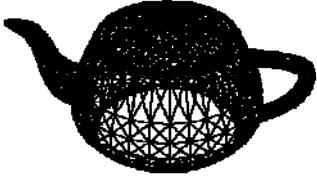
## Objektmodelle

- F.1 Erstellung von Objektmodellen**
- F.2 Farbe und Textur**
- F.3 Polygonnetze**
- F.4 Kurven und Patches**
- F.5 Weitere Methoden der Objektmodellierung**



## Motivation

- Polygonnetze können manche Objekte nur nähern, wir wollen aber Objekte exakt beschreiben
- Wir wollen in der Objektmodellierung auch mit gekrümmten Flächen und Kurven arbeiten
- Objektmodelle können dadurch genauer sein und weniger Speicher benötigen



Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [283]



## Kurven

Mathematische Beschreibung von Kurven		
Bsp.: $y = \sqrt{x^3} - \sqrt{x}$	Bsp.: $x^3 - 2x^2 + x - y^2 = 0$	Bsp.: $Q(t) = (t^2, t^3 - t)$
$f(x) = y$	$f(x,y) = 0$	$Q(t) = (x(t), y(t))$
Explizite Darstellung	Implizite Darstellung	Parameter - Darstellung
<ul style="list-style-type: none"><li>• Nur ein y-Wert für jedes x</li><li>• Keine vertikalen Tangenten</li></ul>	<ul style="list-style-type: none"><li>• Gleichung kann mehr Lösungen als gewollt haben</li><li>• Richtung der Tangente ist schwer zu ermitteln</li></ul>	<ul style="list-style-type: none"><li>• Keine Mehrdeutigkeiten</li><li>• Tangentenvektoren (keine unendliche Steigung)</li><li>• Parameter als Zeit interpretierbar</li></ul>

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [284]



© R. Dörner



## Parameterkurven

Parameter:  $t$

Kurve:  $Q(t) = (x \ y)^T$

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [285]

© R. Dörner

## Parameterkurven

Das Bild eines reellen Intervalls I (Parameterintervall) unter einer stetigen, lokal injektiven Abbildung in den IR2 oder IR3 heißt **Parameterkurve**.

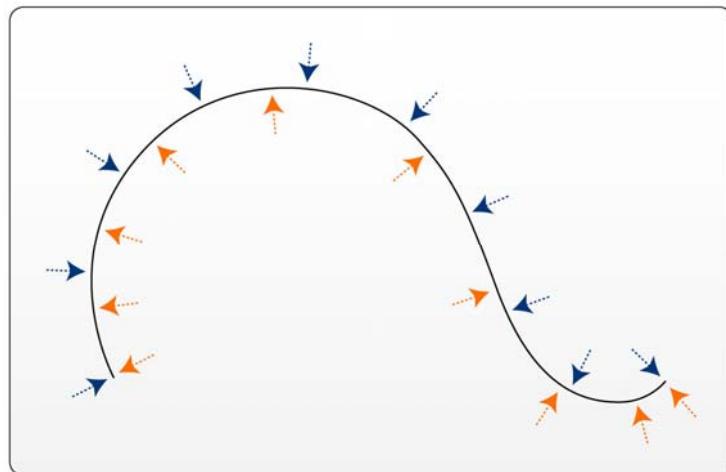
- Kurven mit demselben Bild können unterschiedlich parametrisiert sein
  - Beispiel:  $Q(t) = (t, t)^T$  und  $R(t) = (t^2, t^2)^T$
- Durch  $t = t(t^*)$  kann aus  $Q(t)$  durch eine Parametertransformation  $Q^*(t^*)$  erhalten werden

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [286]

© R. Dörner



## Unterschiedliche Parametrisierung



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [287]

© R. Dörner

## Beispiel Kreis

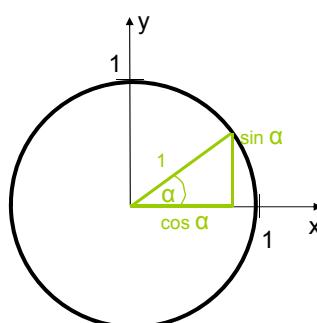
$$Q(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}, \quad t \in [0, 2\pi]$$

Unterschiedliches Parameterintervall:

$$Q(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos 2\pi \cdot t \\ \sin 2\pi \cdot t \end{pmatrix}, \quad t \in [0, 1]$$

Unterschiedliche Parametrisierung:

$$Q(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos 2\pi \cdot t^2 \\ \sin 2\pi \cdot t^2 \end{pmatrix}, \quad t \in [0, 1]$$



```
// Näherung Kreis durch Polygon mit 100 Eckpunkten
glBegin(GL_LINE_LOOP);
for( float t=0.0; t < 1.0; t += 0.01 ) {
    glVertex2f( cos(2.0 * PI * t), sin(2.0 * PI * t) );
}
glEnd();
```

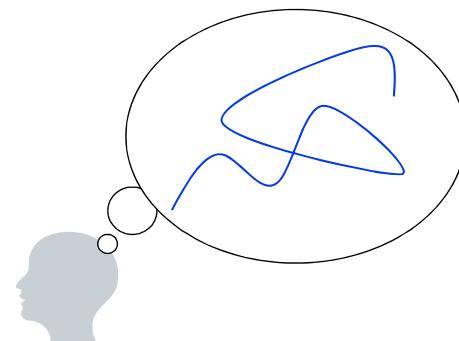
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [288]

© R. Dörner



## Spezifikation von Kurven



Problem:

Wie ermittelt man die  
mathematische Formeln  
für die Parameterdarstellung  
einer Kurve?

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [289]



© R. Dörner

## Spezifikation von Kurven

Möglichkeit A:

### Exakte Darstellung

- Jeder Punkt ist durch eine Formel definiert
- Problem: Formel ist meist nicht bekannt oder zu komplex

Möglichkeit B:

### Interpolatorische Darstellung

- Kurve ist durch Stützstellenbedingungen beschrieben
- Kurve ist an den Stützstellen determiniert

Möglichkeit C:

### Approximative Darstellung

- Kurve ist durch Stützstellenbedingungen beschrieben
- Kurve ist an den Stützstellen *nicht* determiniert

Computergraphik – Prof. Dr. R. Dörner – WS 15

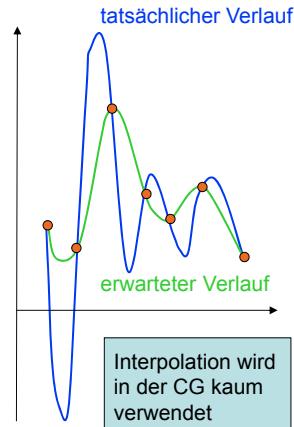
B-AI-V1 [290]



© R. Dörner



## Interpolationspolynome



- Gegeben sind Stützpunkte
- Gesucht ist ein Polynom, das durch alle Stützpunkte geht
- Für  $n+1$  paarweise verschiedene Stützpunkte gibt es genau ein Polynom vom Grad  $n$
- Nachteil: Berechnung ist aufwendig
- Nachteil: Oszillationsproblem



## Spezifikation von Kurven

### Möglichkeit A: **Exakte Darstellung**

- Jeder Punkt ist durch eine Formel definiert
- Problem: Formel ist meist nicht bekannt oder zu komplex

### Möglichkeit B: **Interpolatorische Darstellung**

- Kurve ist durch Stützstellenbedingungen beschrieben
- Kurve ist an den Stützstellen *nicht* determiniert

### Möglichkeit C:

#### **Approximative Darstellung**

- Kurve ist durch Stützstellenbedingungen beschrieben
- Kurve ist an den Stützstellen *nicht* determiniert



## Approximation mit Polynomen

- Grad  $k = 1$ :  
Polygonzug, unstetige Steigungen an den Eckpunkten
- Grad  $k = 2$ :  
In 3D können nur planare Kurven erhalten werden (d.h.  
Kurve liegt immer in einer Ebene)
- Grad  $k = 3$ :  
→ Die vier Koeffizienten können durch vier Stützstellen od.  
zwei Stützstellen und zwei Tangenten gegeben werden
- Grad  $k > 3$ :  
Rechenaufwendig, nur in speziellen Anwendungen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [293]



© R. Dörner

## Approximation mit kubischen Polynomen

$$Q(t) = [x(t) \ y(t) \ z(t)] = [t^3 \ t^2 \ t \ 1] \cdot M \cdot$$

$$\begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Geometrievektor  
enthält Stützpunkte  
oder Tangenten-  
vektoren

Parametervektor

Basismatrix

Geometrievektor

4 x 4 - Matrix

Blendingfunktionen

Gewichtung der geometrischen Nebenbedingungen  $G_i$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [294]



© R. Dörner



## Beispiel: Hermite Kurve

$$Q(t) = [t^3 \quad t^2 \quad t \quad 1] \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

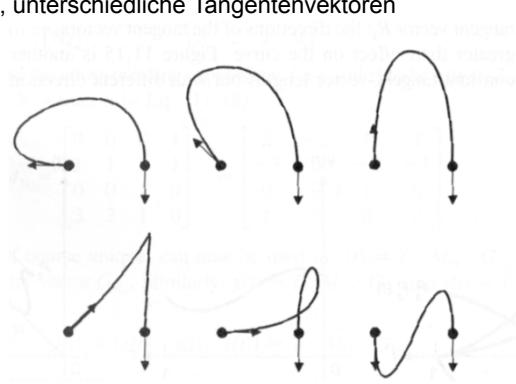
zwei Stützpunkte

zwei Tangenten  
vektoren

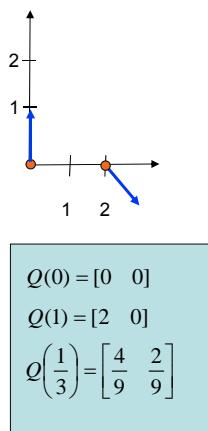


## Beispiel: Hermite Kurve

6 Beispiele:  
Gleiche Stützpunkte, unterschiedliche Tangentenvektoren



## Beispiel: Hermite Kurve



$$Q(t) = [t^3 \ t^2 \ t \ 1] \cdot \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\begin{bmatrix} -3 & 0 \\ 5 & -1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}} \cdot \begin{bmatrix} 0 & 0 \\ 2 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}$$

$$Q(t) = [-3t^3 + 5t^2 \quad -t^2 + t]$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [297]



© R. Dörner

## Bézier - Kurven

$$Q(t) = \vec{p}^T = [t^3 \ t^2 \ t \ 1] \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

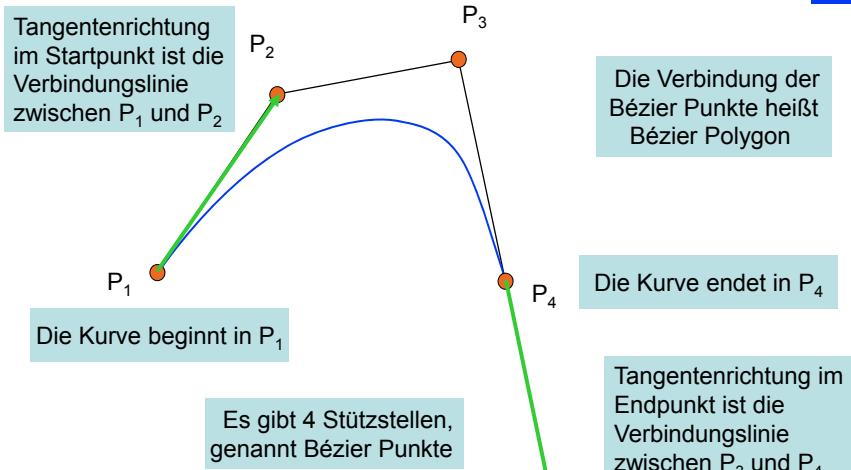
B-AI-V1 [298]



© R. Dörner



## Eigenschaften einer Bézier – Kurve (1)



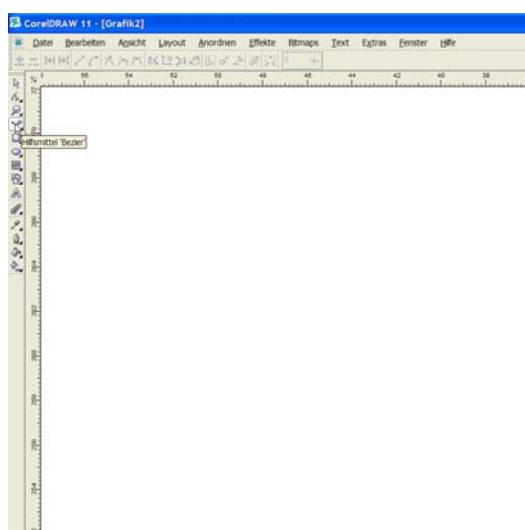
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [299]



© R. Dörner

## Beispiel: Bézier-Kurve in CorelDraw



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [300]



© R. Dörner



## Bernstein - Polynome

- Die Blendingfunktionen sind die Bernstein-Polynome, das  $i$ -te Bernstein-Polynom von Grad  $n$  hat die Form

$$B_i^n = \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i$$

- Die Werte der Bernstein Polynome sind stets nicht negativ
- Für jedes  $t$  gilt: die Summe der Werte aller Bernstein Polynome an der Stelle  $t$  beträgt 1 („sum of unity“)

Beweis:

$$\sum_{i=0}^n \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i = [(1-t) + t]^n = 1^n = 1$$

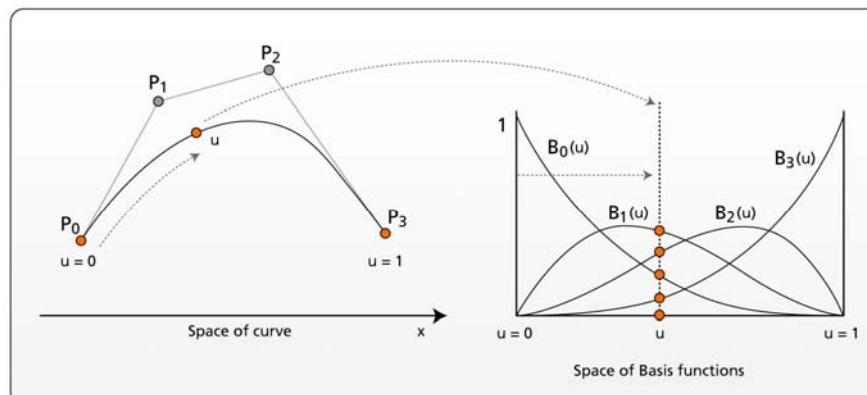
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [301]



© R. Dörner

## Bernstein Polynome



Computergraphik – Prof. Dr. R. Dörner – WS 15

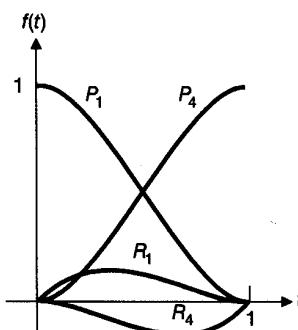
B-AI-V1 [302]



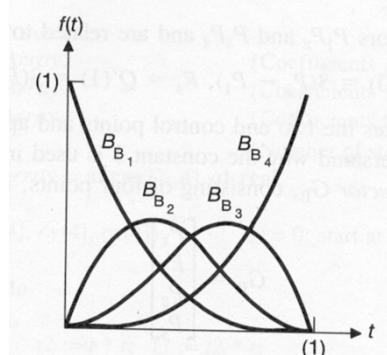
© R. Dörner



## Bernstein - Polynome



Blendingfunktionen von  
Hermite - Kurven



Blendingfunktionen von  
Bézier – Kurven  
(Bernstein-Polynome)

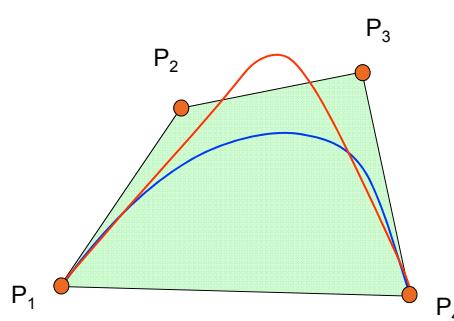
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [303]



© R. Dörner

## Eigenschaften einer Bézier – Kurve (2)



Aus der Sum – of –Unity folgt:

Bézier – Kurven haben die **Convex-Hull-Property**, d.h. die Kurve liegt immer in der konvexen Hülle der Bézier Punkte

- Wichtig für Sichtbarkeits – berechnung
- Wichtig für Clipping
- Wichtig für eine intuitive Vorhersage, wo Kurve liegen wird
- Wichtig, da Oszillation beschränkt wird

Computergraphik – Prof. Dr. R. Dörner – WS 15

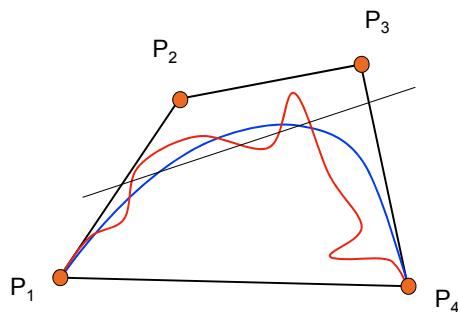
B-AI-V1 [304]



© R. Dörner



## Eigenschaften einer Bézier – Kurve (3)



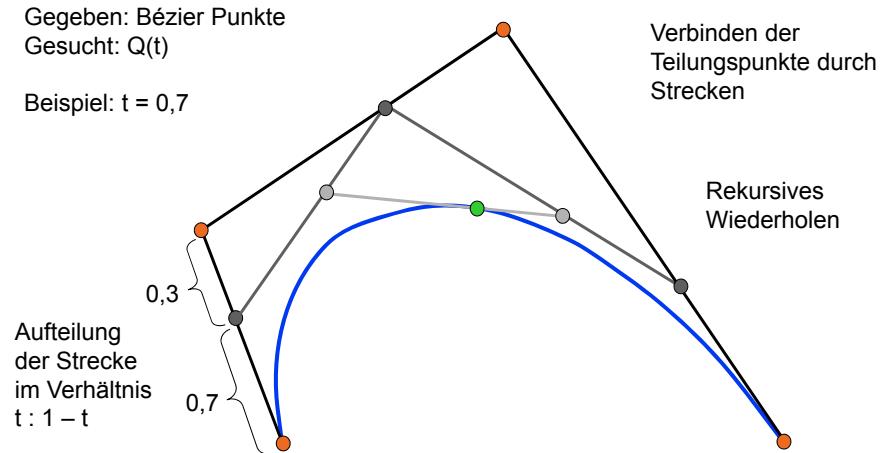
Bézier – Kurven haben die **Variation-Diminishing-Property**, d.h. die Kurve hat höchstens so viele Wendepunkte wie ihr Kontrollpolygon

- Geometrische Bedeutung: eine beliebige Gerade schneidet die Kurve nicht häufiger als das Kontrollpolygon
- Praktische Bedeutung: Kurve oszilliert nicht, „schöne, glatte“ Kurven

## deCasteljau Algorithmus

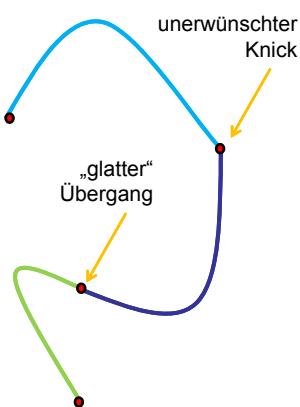
Gegeben: Bézier Punkte  
Gesucht:  $Q(t)$

Beispiel:  $t = 0,7$



## Anschluß von Kurven

- Eine Kurve, die mit kubischen Polynomen approximiert wird, ist nur durch 4 Geometriebedingungen festgelegt  
=> Problem bei „langen“ Kurven
- Idee: Mehrere Kurven als Kurvensegmente betrachten und durch Anschluss aneinander fügen
- An den Anschlüssen wird die Einhaltung von Bedingungen gefordert, die eine „glatten“ Anschluss erlauben



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [307]

© R. Dörner

## Anschlußbedingungen

Geometrische Stetigkeit:

Der Anschluß zweier Kurven  $Q(t)$  und  $R(t)$  heißt  $G^0$  stetig, wenn ein gleicher Segmentrandpunkt der Kurven existiert.

Der Anschluß zweier Kurven  $Q(t)$  und  $R(t)$  heißt  $G^1$  stetig, wenn die Richtung der Tangenten am Segmentrandpunkt übereinstimmt. Die Übereinstimmung in der Länge der Tangentenvektoren (wie bei  $C^1$  Stetigkeit) ist nicht gefordert.

$G^1$  Stetigkeit impliziert nicht  $C^1$  Stetigkeit und umgekehrt

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [308]

© R. Dörner

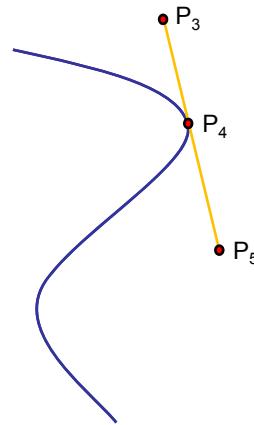


## Anschluß bei Bézier Kurven

- Sei  $Q(t)$  eine Bézier Kurve mit Bézier Punkten  $P_1, \dots, P_4$   
Sei  $R(t)$  eine Bézier Kurve mit Bézier Punkten  $P_4, \dots, P_7$   
Der Segmentrandpunkt ist  $P_4$
- $G^1$  stetiger Anschluß ist erreicht, wenn gilt:

$P_3 - P_4 = k (P_4 - P_5)$ ,  $k > 0$   
d.h.  $P_3, P_4$  und  $P_5$  sind kollinear (liegen auf einer Geraden)

Für  $k=1$  ergibt sich ein  $C^1$  stetiger Übergang



## Uniforme B-Splines

- Motivation: Wollen Kurven, die „länger“ sind, d.h. die mehr als 4 Stützpunkte haben
- Idee: Lange Kurve  $Q$  aus lauter Segmenten  $Q_i$  zusammensetzen, jedes Segment ist wieder ein kubisches Polynom (d.h. hat 4 Stützpunkte)
- Also:

$P_0 P_1 P_2 P_3$  bestimmen Segment  $Q_3$

$P_1 P_2 P_3 P_4$  bestimmen Segment  $Q_4$

$P_2 P_3 P_4 P_5$  bestimmen Segment  $Q_5$

...

## Uniforme B-Splines: Ein Beispiel

Gegeben:  $m+2$  deBoor Punkte  $P_0, \dots, P_{m+1}$

$m = 5$

Gegeben: Knotenvektor  $T = [t_3, \dots, t_{m+2}]$

$t^* = [0, 1, 2, 3, 4]$

Knotenwerte haben gleichen Abstand:  
uniformer B – Spline

Es gibt  $m$  Knoten

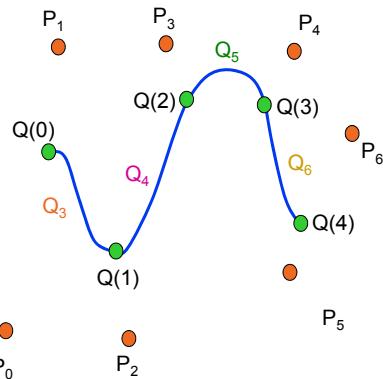
Knoten:  $Q(0), Q(1), Q(2), Q(3), Q(4)$

Anzahl Kurvensegmente:  $m - 1$

4 Kurvensegmente:  $Q_3, Q_4, Q_5, Q_6$

$Q_i(t)$  ist definiert für  $t_i \leq t < t_{i+1}$

$Q_5(t)$  ist definiert für  $2 \leq t < 3$



## Uniforme B-Splines

- Gegeben sind  $m+2$  Punkte  $P_0, P_1, \dots, P_{m+1}$  (mit  $m>2$ ), genannt Kontrollpunkte oder deBoor Punkte
- Der B-Spline  $Q(t)$  besteht aus  $m-1$  Kurvensegmenten  $Q_3, Q_4, \dots, Q_{m+1}$ . Jedes  $Q_i$  ist durch ein kubisches Polynom beschrieben
- Durch Parametertransformation wird erreicht, daß jedes  $Q_i$  definiert ist in einem Bereich  $t_i \leq t < t_{i+1}, 3 \leq i \leq m+1$
- Die Punkte  $Q(t_i)$  heißen Knoten

## Uniforme B-Splines

- Die Werte  $t_i$  heißen Knotenwerte
- Ein B-Spline wird definiert durch deBoor Punkte und Knotenwerte
- Haben zwei aufeinanderfolgende Knotenwerte  $t_i$  und  $t_{i+1}$  den gleichen Abstand für alle  $3 \leq i \leq m+1$  heißt der B-Spline uniform (o.B.d.A.  $t_3 = 0$  und  $t_{i+1} - t_i = 1$ )
- Der B-Spline ist eine gewichtete Summe von polynomiellen Basisfunktionen (daher „B“)



## Kurvensegmente uniformer B-Splines

- Jedes Kurvensegment  $Q_i$  wird durch die 4 Kontrollpunkte  $P_{i-3}, P_{i-2}, P_{i-1}$  und  $P_i$  beschrieben. Der Geometrievektor  $G_i$  für  $Q_i$  lautet daher

$$G_i = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

- Der Parametervektor  $T_i$  für Kurvensegment  $Q_i$  lautet

$$T_i = \left[ \left( \frac{t - t_i}{t_{i+1} - t_i} \right)^3 \quad \left( \frac{t - t_i}{t_{i+1} - t_i} \right)^2 \quad \left( \frac{t - t_i}{t_{i+1} - t_i} \right) \quad 1 \right]$$



## Kurvensegmente uniformer B-Splines

- Damit lässt sich  $Q_i$  in der allgemeinen Form darstellen als

$$Q_i(t) = \vec{p}^T = T_i \cdot \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot G_i$$



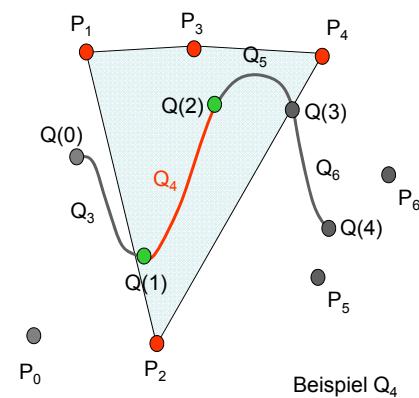
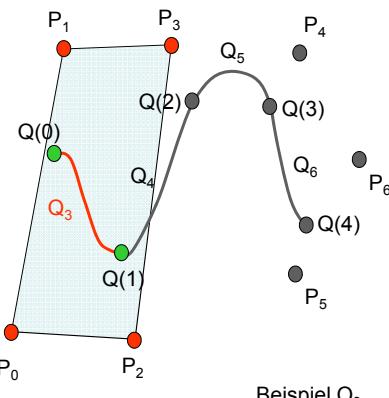
## Eigenschaften uniformer B-Splines

- Für jedes Kurvensegment  $Q_i$  gilt die Convex Hull Property hinsichtlich der Kontrollpunkte  $P_{i-3}, P_{i-2}, P_{i-1}$  und  $P_i$
- Die Änderung des Kontrollpunktes  $P_i$  bewirkt nur eine Veränderung in  $Q_i, Q_{i+1}, Q_{i+2}$  und  $Q_{i+3}$  (Local Control Property)
- Ein geschlossener B-Spline kann durch folgende Kontrollpunktsequenz erreicht werden:

$P_0, P_1, P_2, \dots, P_m, P_0, P_1, P_2$



## B-Splines: Convex Hull Property



Computergraphik – Prof. Dr. R. Dörner – WS 15

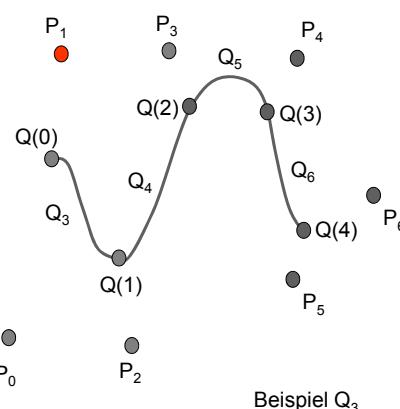
B-AI-V1 [317]



© R. Dörner

## B-Splines: Local Control Property

Änderung von P<sub>1</sub>:  
Nur Q<sub>3</sub> und Q<sub>4</sub>  
müssen neu  
berechnet werden  
Rest des B – Splines  
bleibt gleich.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [318]



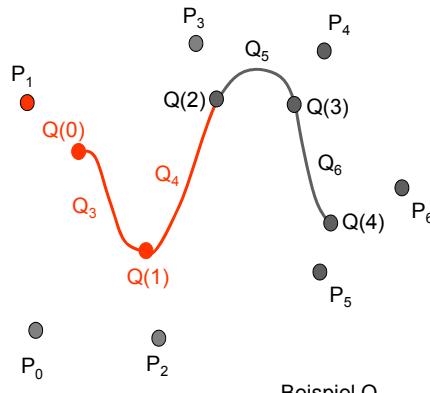
© R. Dörner



## B-Splines: Local Control Property

Änderung von  $P_1$ :

Nur  $Q_3$  und  $Q_4$   
müssen neu  
berechnet werden  
Rest des B – Splines  
bleibt gleich.



Beispiel  $Q_3$

## Nicht-uniforme B-Splines

- B-Splines mit mehrfachen oder ungleichabständigen Knotenwerten sind nicht-uniform
- Nicht-uniforme B-Splines haben noch zusätzlich Knotenwerte  $t_0, t_1, t_2, t_{m+2}, t_{m+3}, t_{m+4}$
- Spezialfall: Für den Knotenvektor  $T = [0,0,0,0,1,1,1,1]$  ist der nicht-uniforme B-Spline eine Bézier-Kurve
- Nicht-uniforme B-Splines haben für jedes Kurvensegment unterschiedliche Blendingfunktionen. Diese können nicht durch eine Matrix, sondern müssen durch Rekurrenzgleichungen angegeben werden

## Nicht-uniforme B-Splines

Allgemeine Gleichung für die B-Spline-Funktionen  $N_{i,k}$  ( $i=0, \dots, n$ ) bei Verwendung von Polynomen des Grades  $k-1$  (üblich:  $k=4$ ) und Trägervektor der Knotenwerte  $T = [t_0, t_1, \dots, t_{n+k}]$ :

$$N_{i,1}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & \text{sonst} \end{cases}$$
$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} \cdot N_{i,k-1}(t) + \frac{-t + t_{i+k}}{t_{i+k} - t_{i+1}} \cdot N_{i+1,k-1}(t)$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [321]



© R. Dörner

## Nicht-uniforme B-Splines

- Mit den Kontrollpunkten  $P_i$  ( $i=1, \dots, n$ ), dem Trägervektor  $T = [t_0, t_1, \dots, t_{n+k}]$  und den B-Spline-Funktionen  $N_{i,k}$  ergibt sich folgende Parameterdarstellung der B-Spline Kurve  $Q(t)$

$$Q(t) = \vec{p} = \sum_{i=0}^n N_{i,k}(t) \cdot \vec{p}_i \quad , t \in [t_0, t_{n+k}]$$

- Berechnung von Punkten auf dem B-Spline mit dem deBoor Algorithmus (verallgemeinerter deCasteljau-Algorithmus)

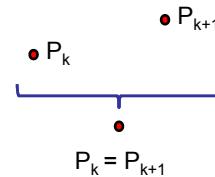
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [322]



## Mehrfache Kontroll- und Knotenwerte

- Problem:  
Anfangs- und Endpunkt der Kurve kann nicht durch Wahl der geometrischen Nebenbedingungen einfach bestimmt werden



- Lösungen:
  - Identifikation von Kontrollpunkten
  - Identifikation von Knotenwerten

$$T = [t_0, t_1, \dots, t_k, \underbrace{t_{k+1}, t_{k+2}, \dots, t_n}]$$

$$T = [t_0, t_1, \dots, t_k, t_k, t_{k+2}, \dots, t_n]$$



## Mehrfache Kontrollpunkte bei nicht-uniformen B-Splines

2-fach	$C^2, G^1$ Konvexe Hülle wird kleiner
3-fach	$C^2, G^0$ Kurve interpoliert den 3-fach Kontrollpunkt, Kurvensegmente sind linear
4-fach	$C^2, G^0$ Kurve interpoliert den 4-fach Kontrollpunkt und die beiden benachbarten Kontrollpunkte, Kurvensegmente sind linear



## Mehrfache Knotenwerte bei nicht-uniformen B-Splines

2-fach	$C^1, G^1$ Knoten in einer schmäleren konvexen Hülle
3-fach	$C^0, G^0$ Kurve interpoliert Kontrollpunkt
4-fach	Unstetigkeit Kurve endet an einem Kontrollpunkt und geht am nächsten Kontrollpunkt weiter

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [325]



© R. Dörner

## NURBS

- Idee: Angabe der Kontrollpunkte in homogenen Koordinaten (w-Wert wird als Gewicht für die „Anziehungskraft“ eines Kontrollpunkts verwendet: zusätzlicher Freiheitsgrad)
- Blending Funktionen sind nicht ganzrationale Funktionen, sondern gebrochenrationale Funktionen (da durch den homogenen Teil dividiert werden muss, Parameter t tritt im Nenner auf)
- Wir erhalten:  
**Nicht-Uniforme Rationale B-Splines (NURBS)**

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [326]



© R. Dörner



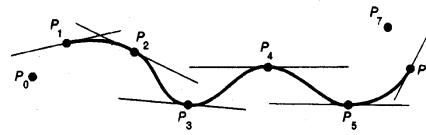
## Eigenschaften von NURBS

- Haben Eigenschaften der B-Splines (Stetigkeit, lokale Kontrolle, konvexe Hülle für die einzelnen Kurvensegmente)
- Invarianz gegenüber geometrischen Transformationen, wie z.B. Rotation (d.h. nur die Kontrollpunkte müssen transformiert werden und nicht jeder Punkt der Kurve)
- NURBS können Kegelschnitte (z.B. Kreise, Ellipsen, Parabeln) exakt beschreiben

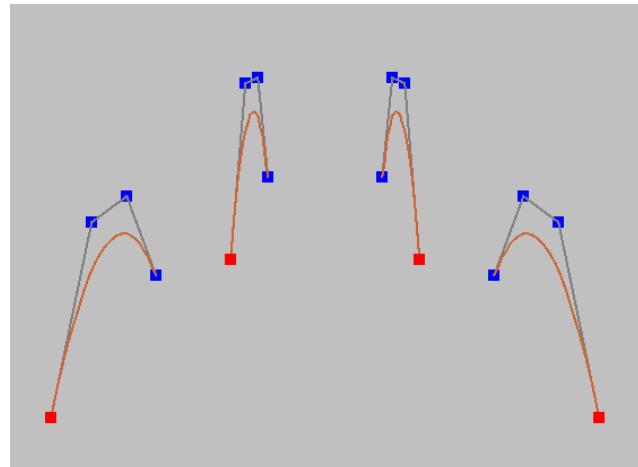


## Definition von Kurven

- Es gibt eine große Spannbreite weiterer Kurvendefinitionen
  - $\beta$ -Splines
  - Splines in Tension
  - Exponentialsplines
  - Wilson-Fowler Splines
  - Catmull-Rom Splines
  - ...
- Es gibt keine „beste“ Kurvenrepräsentation, diese muß anwendungsabhängig gewählt werden (heute werden meist NURBS verwendet)



## Von Kurven zu Flächen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [329]

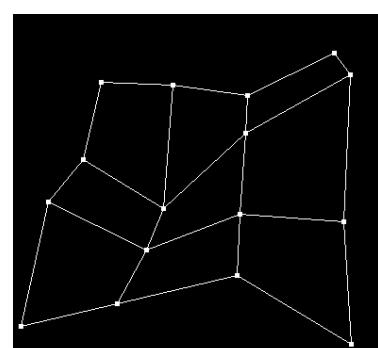


© R. Dörner

## Definition von Flächen

- Erweiterung von Kurven auf Flächen: Kurven beschreiben Schnitte von Flächen
- Verwendung von zwei Parametern:  $F(u,v)$
- Freiformflächen (Erweiterung der allgemeinen Darstellung):

$$\begin{aligned} F(u,v) &= T(u) M G(v) \\ &= T(u) M G M^T S(v)^T \end{aligned}$$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [330]

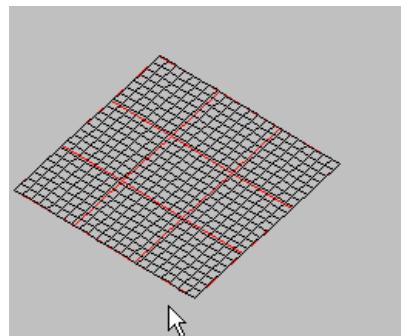


© R. Dörner



## Definition von Flächen

- Beispiel Bézier Flächen:
  - M ist die Basismatrix der Bézier – Kurven
  - G besteht aus 16 Kontrollpunkten
  - T und S sind die Parametervektoren
- Anschluß von Flächensegmenten führt zu Patches
- Große Spannbreite von Flächendefinitionen:
  - B – Spline Flächen
  - Gordon – Coons – Flächen
  - Dreiecks – Bézier – Flächen
  - ...



Bézier-Patches

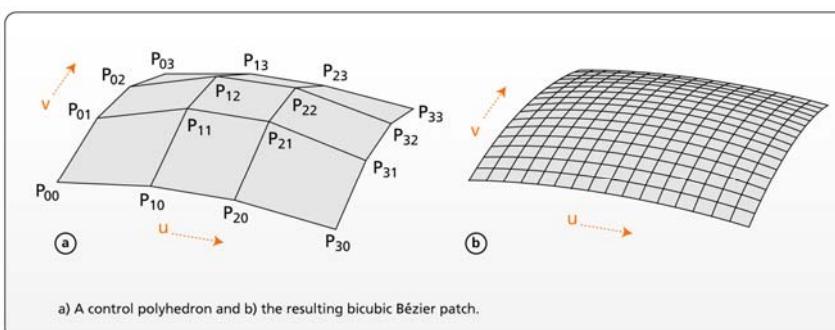
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [331]



© R. Dörner

## Beispiel: Bézier-Patches



a) A control polyhedron and b) the resulting bicubic Bézier patch.

Computergraphik – Prof. Dr. R. Dörner – WS 15

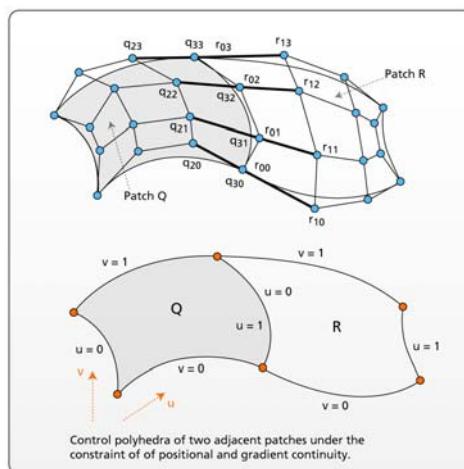
B-AI-V1 [332]



© R. Dörner



## Zusammensetzen von Patches



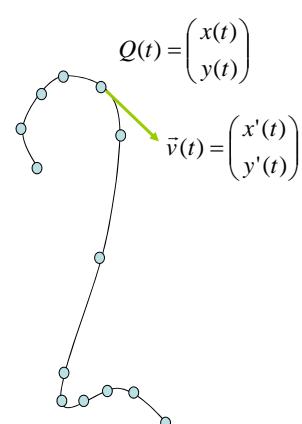
Flächenstücke (Patches)  
können zu größeren  
Flächen zusam-  
mengesetzt werden, ähnlich  
wie ein B-Spline aus lauter  
Kurvensegmenten besteht.

Bestimmte Stetigkeits-  
bedingungen müssen dabei  
eingehalten werden.



## Vorteile von Parameterkurven und Parameterflächen

- Einfach (ohne Fallunterscheidung) und mit einfach wählbarer Genauigkeit zeichenbar
- Parameterisierung kann für Texturmapping genutzt werden
- Formel für Tangentenvektor durch Ableitung nach dem Parameter einfach zu ermitteln
- Parameter passt sich in der Regel an die Krümmung an
- Durchlaufen mit verschiedenen Geschwindigkeiten machbar



## Objektmodelle

- F.1 Erstellung von Objektmodellen
- F.2 Farbe und Textur
- F.3 Polygonnetze
- F.4 Kurven und Patches
- F.5 Weitere Methoden der Objektmodellierung**



## B-Rep

- B-Rep ist Abkürzung für Boundary Representation
- Begrenzung von Objekten wird beschrieben
- Verallgemeinerung von Polygonnetzen
  - Kurven
  - Splines

Punktliste

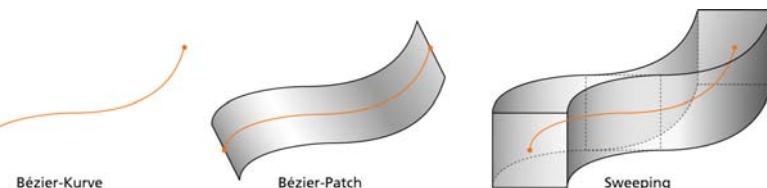
Kantenliste

Flächenliste



## Oberflächenbeschreibungen

- Flächenbeschreibungen wie z.B. Bézier-Patches
- Sweeping und Extrusion:
  - 1. Querschnitt eines Objektes modellieren
  - 2. Querschnitt entlang einer Kurve im Raum bewegen



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [337]

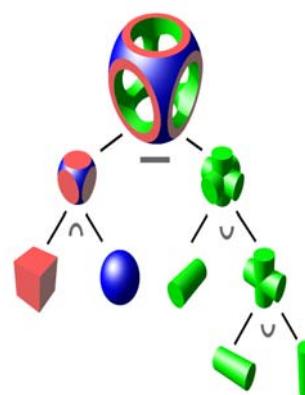


© R. Dörner

## Volumenbeschreibungen

- Volumen mit impliziten Ungleichungen beschreiben, z.B.  $x^2 + y^2 + z^2 < r^2$
- CSG (Constructive Solid Geometry)
- Voxel: Lauter kleine Würfel
- Quadtree und Octree: Volumenelemente unterschiedlicher Größe
- Partikelsysteme

Quelle: Wikipedia



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [338]

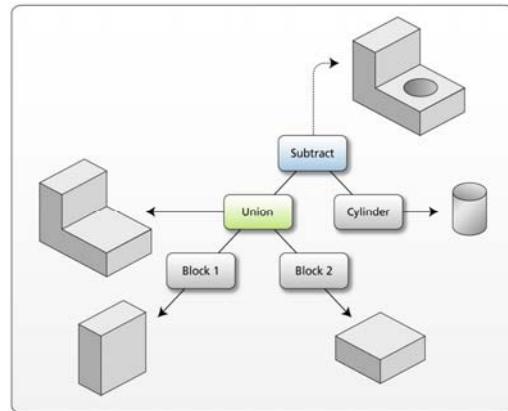


© R. Dörner



## CSG (Constructive Solid Geometry)

- Grundprimitive (Quader, Zylinder, Kegel, Kugel)
- Mengentheoretische / boolsche Operationen
- Aus den Grundprimitiven werden schrittweise komplexere Objekte zusammengesetzt (wie mit Lego-Steinen)
- Verbreitet in CAD – Anwendungen



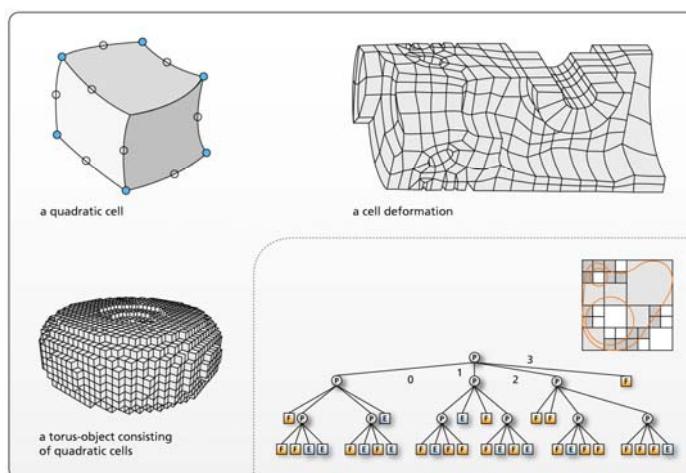
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [339]



© R. Dörner

## Volumenbeschreibungen



Computergraphik – Prof. Dr. R. Dörner – WS 15

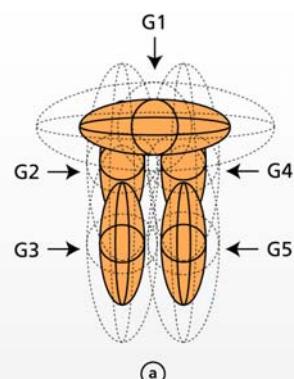
B-AI-V1 [340]



© R. Dörner



## Volumenbeschreibungen



Group	Blends with
G1	G2 G4
G2	G1 G3
G3	G2
G4	G1 G5
G5	G4

(a)



(b)

(c)

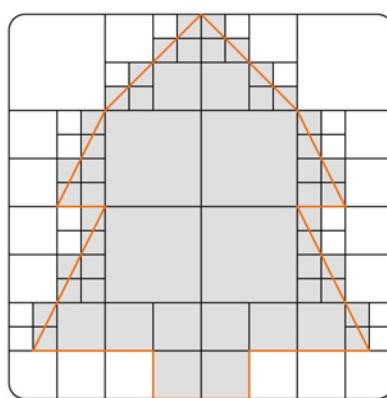
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [341]

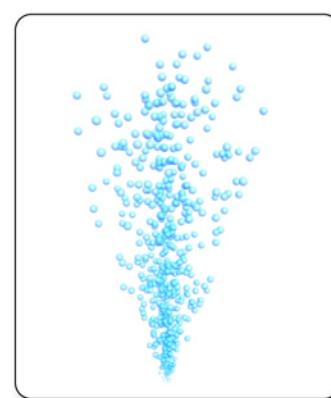


© R. Dörner

## Quadtree und Partikelsystem



Quadtree



Partikelsystem

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [342]

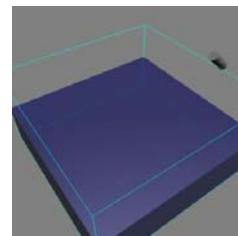
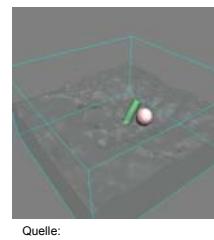
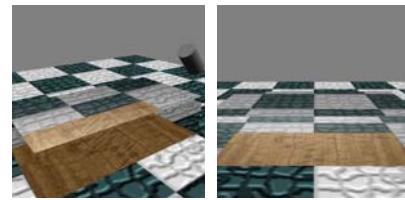


© R. Dörner



## Sonderfälle

- Spezielle Modellierungs-methoden für Objekte, die nicht fest sind (non-rigid) und Aussehen über die Zeit verändern
- Beispiel:
  - Physikalische Modellierung (z.B. Navier – Stokes bei Flüssigkeiten)
  - Partikelsysteme
  - Spring – Feather Modelle



Quelle:  
T. Takahashi et.al.,  
The simulation of fluid-rigid body  
interaction, Siggraph

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [343]



© R. Dörner

## Sonderfall: Gase

### Cloud Simulation

Quelle:  
Dobashi et.al.  
Cloud Simulation,  
Siggraph 2000

Computergraphik – Prof. Dr. R. Dörner – WS 15

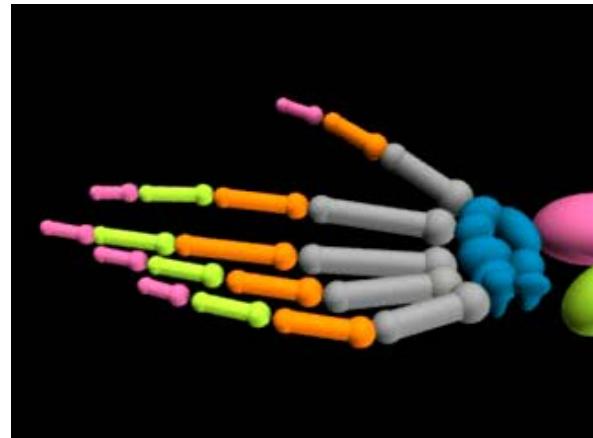
B-AI-V1 [344]



© R. Dörner



## Sonderfall: Bewegliche Modelle



Quelle:  
F. Scheepers et.al.,  
Anatomy-Based Modeling  
of the Human Musculature,  
Siggraph 1997

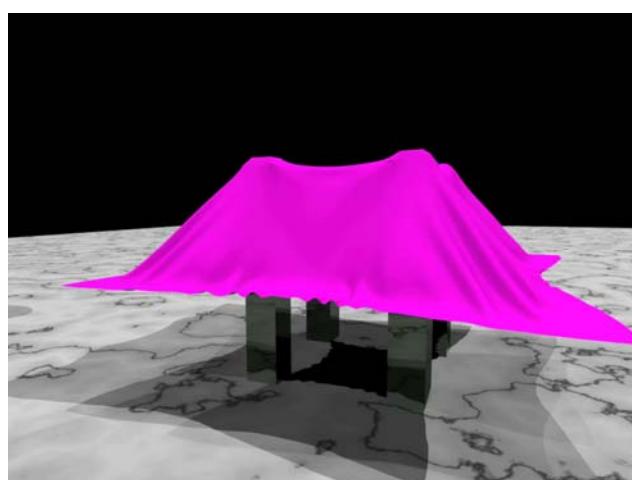
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [345]



© R. Dörner

## Sonderfall: Stoff



Quelle:  
R. Bridson et.al.,  
Robust Treatment of Collisions,  
Contact and Friction for Cloth  
Animation,  
Siggraph 2002

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [346]



© R. Dörner



## Kriterien für die Auswahl von Objektmodellen

Genaugigkeit

Mächtigkeit

Renderbarkeit

Validierbarkeit

Animierbarkeit

Repräsentierbarkeit

Erstellbarkeit

Kein Verfahren der Objektmodellierung ist optimal für alle Kriterien  
⇒ Kombination von Verfahren, Konversion



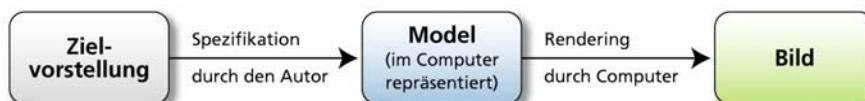
## Teil G: Rendering und OpenGL



## Rendering und OpenGL

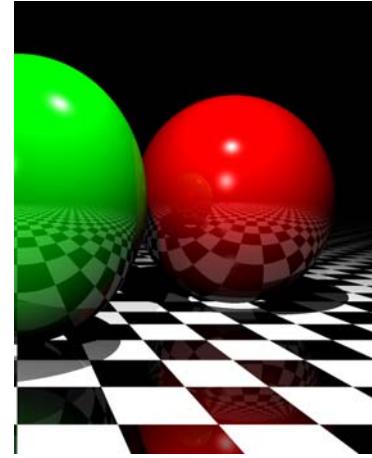
- G.1 Pipeline-Rendering
- G.2 Shader-Programmierung
- G.3 CPU-Seite und GPU-Seite

## Rendering



## Wie funktioniert das Rendering?

- Mehrere Verfahren für das Rendering sind in der Computergrafik bekannt:
  - Raytracing
  - Radiosity
  - Rasterisierung
  - ...
- Zunächst: Umsetzung dieser Verfahren in Form eines (Software-) Programms



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [351]



## Hardware-Beschleunigung

- Problem: Bildberechnung mit Software kann lange dauern
- Benötigen in bei interaktiver Graphik mindestens 60 fps (Frames pro Sekunde)
- Idee: Realisierung in Hardware, um Rendering zu beschleunigen
- Zunächst: Auswahl eines Renderingverfahrens, das sich besonders gut in Hardware realisieren lässt (sog. Renderpipeline).



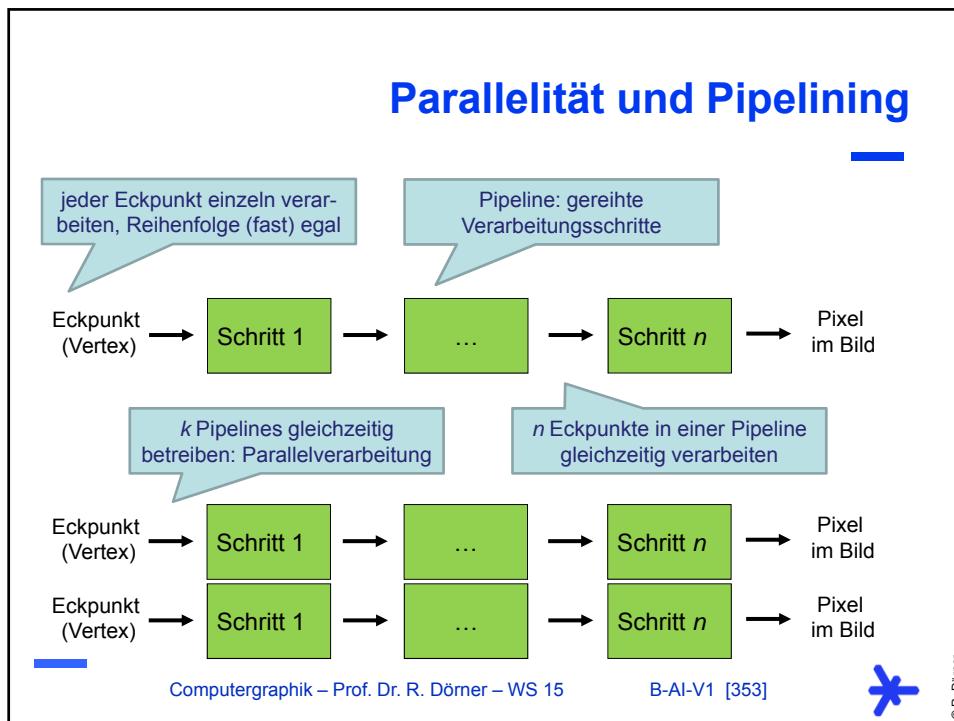
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [352]



© R. Dörner





## Varianten beim Rendering



photorealistisch

Cartoon



schräffiert



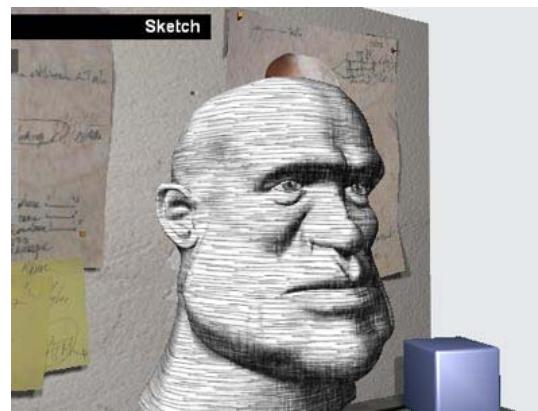
Aquarell

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [355]

© R. Dörner

## Beispiele von Shadern



Quelle: Virtools ([www.3ds.com](http://www.3ds.com))

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [356]



## Flexibilität beim Rendering

- Es gibt viele Fälle, in denen man vom Standard Rendering von OpenGL abweichen möchte
- Problem: Möchte man vom in Hardware „fest verdrahtetem“ Rendering abweichen, bleibt nur das Software-Rendering übrig
- Software-Rendering ist langsam (akzeptabel nur bei Offline-Rendering)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [357]



© R. Dörner

## Programmierbare Grafik-Hardware

- Idee: Hardware-Beschleunigung flexibler machen
- Grafikchip-Hersteller haben GPUs (Graphical Processing Units) programmierbar gemacht ähnlich den CPUs
- Resultat: Ganze Welt an neuen Möglichkeiten das Rendering für interaktive 3D Computergrafik zu gestalten



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [358]



© R. Dörner



## Shader Programmierung

- GPU (anstatt der CPU) programmieren
- Diese Programme nennt man **Shader**
- Wichtige Grundlage
  - Neue Visualisierungstechniken
  - Hohe visuelle Qualität
  - Interaktivität (für VR, Visualisierung, ...)
- GPU arbeitet parallel, hohe dedizierte Rechenleistung, die auch in anderen Gebieten außerhalb der GDV genutzt werden kann



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [359]



© R. Dörner

## Shader und Standard Rendering Pipeline

- OpenGL realisierte bislang eine Standard Rendering Pipeline („Fixed Functionality“) für GPUs
- Nur ein Teil dieser Standard Pipeline ist flexibel programmierbar
- Für bestimmte Schritte in der Pipeline gibt es so gute Algorithmen, dass man diese „fest verdrahtet“ lassen kann



Quelle: bearingdrift.com

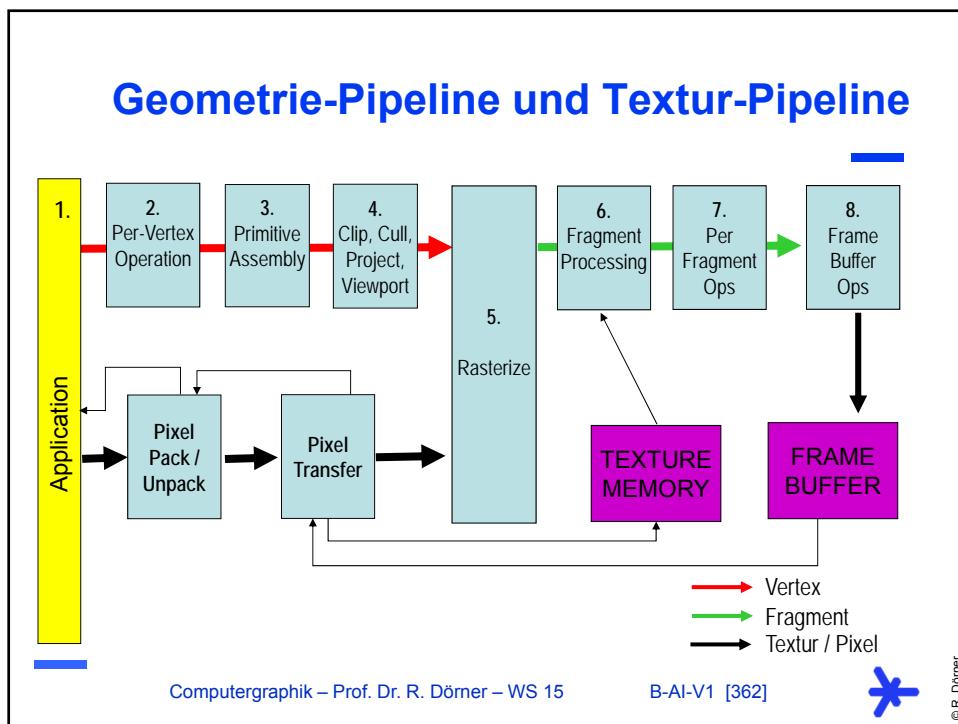
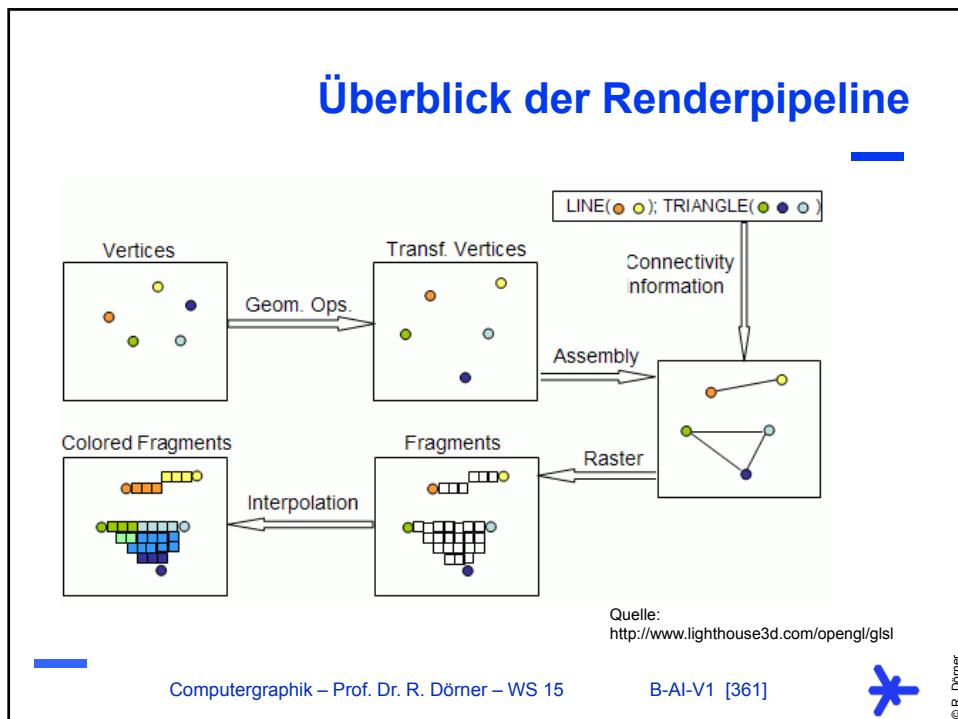
Computergraphik – Prof. Dr. R. Dörner – WS 15

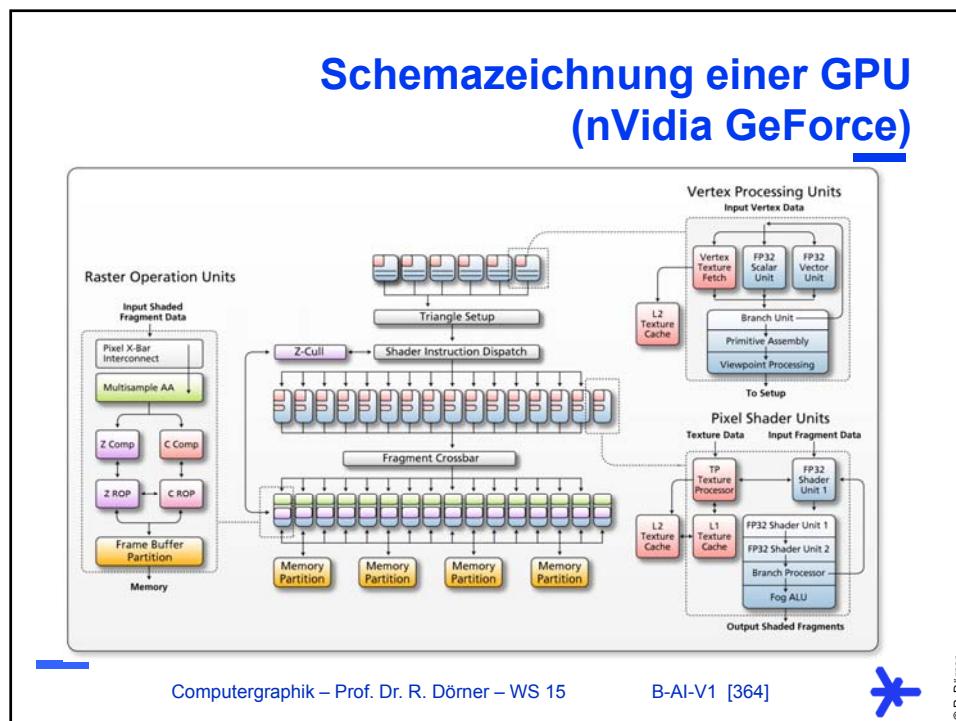
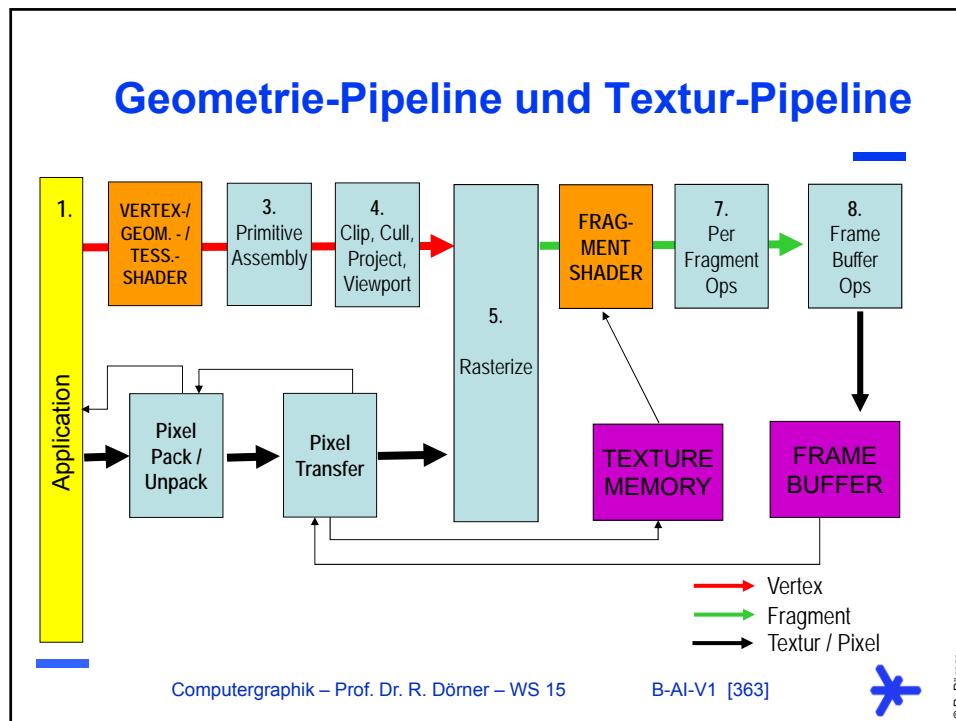
B-AI-V1 [360]



© R. Dörner

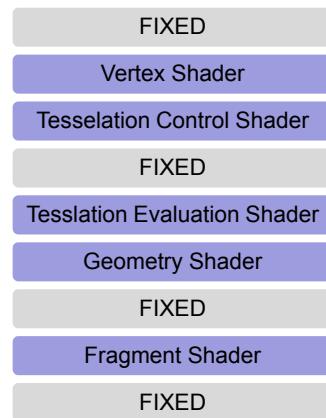






## Shader Typen

- Die Renderpipeline in einer GPU kann an mehreren Stellen durch Programmierung flexibel gestaltet werden:
  - Bearbeitung der Eckpunkte (Vertex-Shader) in 3D
  - Bearbeitung der Fragmente (Fragment-Shader, auch Pixel-Shader genannt) in 2D
  - Hinzufügen von Eckpunkten (Geometry-Shader), z.B. Bezier-Kurvenpunkte aus Stützpunkten berechnen
  - Spezifikation der Tesselierung (Tesselation Control und Tesselation Evaluation Shader)
- Das gleiche Shader-Programm wird in mehrere besondere Prozessoren in der GPU geladen (Parallelverarbeitung)



## Rendering und OpenGL

- G.1 Pipeline-Rendering
- G.2 Shader-Programmierung
- G.3 CPU-Seite und GPU-Seite



## Programmiersprachen für Shader

- Zunächst wurden programmierbare GPUs in Assembler programmiert
- Nachteile:
  - Starke Abhängigkeit von dem individuellen Graphik-Chip
  - Schwierig und ungewohnt zu programmieren
- Idee: Verwendung einer höheren Programmiersprache

```
push    ebp
mov     ebp,esp
movzx  ecx,[ebp+arg_0]
pop    ebp
movzx  dx,cl
lea    eax,[edx+edx]
add    eax,edx
shl    eax,2
add    eax,edx
shr    eax,8
sub    cl,al
shr    cl,1
shr    al,5
```



## Programmiersprachen für Shader

- Höhere Programmiersprachen für CPUs (wie z.B. Java oder C++) können schlecht verwendet werden
  - GPUs benötigen keine Chars und Strings
  - Parallelverarbeitung
  - Spezielle Funktionen sind in Hardware realisiert
- Daher: Entwicklung neuer, spezieller Programmiersprachen auf Basis von C/C++ (bei vielen Programmierern bekannt)

```
layout (std140) uniform
Matrices{
    mat4 pmvMat; };

in vec3 position;
in vec2 aTexCoord;
out vec2 texCoord;

void main(){
    texCoord = aTexCoord;
    gl_Position =
        pmvMat *
        vec4(position, 1);
}
```



## Beispiele von Shader Sprachen

- RenderMan SL (Pixar): für off-line Renderer
- Cg (NVIDIA), für DirectX und OpenGL, auch Karten anderer Hersteller
- HLSL (Microsoft), nur DirectX, Direct3D Effects System
- GLSL: OpenGL Shading Language
- CUDA, OpenCL u.a.: Ausnutzung der GPU für Scientific Computing



## Die GLSL Programmiersprache

- GLSL kann für alle Shadertypen genutzt werden
- Jeder Shader hat eine eigene *main* Funktion der Form **void main() { }**
- Kommentare mit // oder /\* \*/

```
layout (std140) uniform
Matrices{
    mat4 pmvMat; };

in vec3 position;
in vec2 aTexCoord;
out vec2 texCoord;

void main(){
    texCoord = aTexCoord;
    gl_Position =
        pmvMat *
        vec4(position, 1);
}
```



## Unterschiede zu C / Java

- Keine Chars und Strings
- Kein goto – Statement, kein switch – Statement
- Präprozessor (z.B. #define) erlaubt, aber kein #include und keine Header-Dateien
- Kein sizeof – Operator
- Keine Pointertypen und Dereferenzierung
- Funktionen sind nur call-by-value-return
- Keine static Variablen, dafür uniform
- Besondere Schlüsselworte: layout, discard
- Spezielle Datentypen



## Datentypen

- Texturen
  - Datentypen:  
`sampler1D, sampler2D, sampler3D, samplerCube,  
sampler1DShadow, sampler2DShadow`
  - Beispiel:  

```
vec2 coord = vec2(0.7, 0.9); // Texturkoordinate
uniform sampler2D sp; // Textur (kommt von außen)
vec4 color = texture2D(sp, coord); // Auslesen
```
- Structs und Arrays ähnlich wie in C
  - Bsp.: `vec4 points[ ]; points[2] = vec4(1.0);`



## Datentypen

- Skalare
  - `float, int, bool`
- Vektoren
  - `vec2, vec3, vec4, ivec2, ivec3, ivec4, bvec2, bvec3, bvec4`
- Matrizen
  - `mat2, mat3, mat4`
- Strenges Überprüfen der Datentypen, keine Konversion
  - `int i = 7;`
  - `float f = i; // Falsch: float vs. Integer`
  - `float f = float(i); // Richtig: Verwenden eines // Konstruktors`

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [373]



© R. Dörner

## Swizzling

- Zugriff auf die zwei-, drei- oder vierdimensionalen Datentypen möglich als
  - x y z w
  - r g b a
  - s t p q
- Beispiel:

```
vec3 v3 = (1.0, 2.0, 3.0, 4.0); // okay, 4.0 ignoriert
vec4 v4 = v3.xxzz; // v4 = (1.0, 1.0, 3.0, 3.0)
v3.r = v4.p; // v3 = (3.0, 2.0, 3.0)
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [374]



© R. Dörner



## Spezielle Operatoren

- vec – Datentypen verfügen über komponentenweise Operatoren

- Beispiel:

```
vec3 u, v;
float f;
v = u + f;    // in jeder Komponente wird f addiert
v = u + v;    // komponentenweise Addition
mat3 m;
v = u + m[1]; // erste Spalte der Matrix wird aufaddiert
v = 3 * u;    // jede Komponente mal 3
v = u * v;    // komponentenweise Multiplikation
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [375]



© R. Dörner

## Spezielle Operatoren

- Multiplikation von Vektoren und Matrizen

- Beispiel:

```
vec4 u, v; mat4 m;
u = m * v; u = v * m; m = m * m;
```

- Skalarprodukt und Kreuzprodukt

- Beispiel:

```
vec3 u, v, w; float f;
f = dot(u,v);      // f = <u,v>
w = cross(u,v);   // w = u x v
w = normalize(w); // bringt w auf Länge 1
f = length(w);    // bestimmt die Länge. Hier: f=1
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [376]



© R. Dörner



## Eingebaute Funktionen

- GLSL hat eine Vielzahl eingebauter Funktionen
- Im Zweifelsfall sollte immer die eingebaute Funktion verwendet werden, da diese ggf. direkt und sehr effizient in Hardware realisiert ist
- Beispiel:

```
vec2 v; float f;  
f = sqrt(v.x * v.x + v.y * v.y); // schlecht  
f = length(v); // gut
```



## Eingebaute Funktionen

- Trigonometrische Funktionen  
`sin, cos, tan, asin, acos, atan, radians, degrees`
- Exponential-Funktionen  
`pow, exp2, log2, sqrt, inversesqrt`
- Arithmetische Funktionen  
`abs, sign, floor, ceil, fract, mod, min, max, clamp`



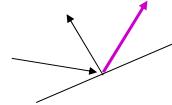
## Eingebaute Funktionen

- Vektorfunktionen  
`length, distance, dot, cross, normalize`
- Matrixfunktionen  
`matrixcompmult`
- Funktionen für das Verhältnis von Vektoren  
`lessThan, lessThanEqual, greaterThan, greaterThanEqual, equal, notEqual, any, all, not`



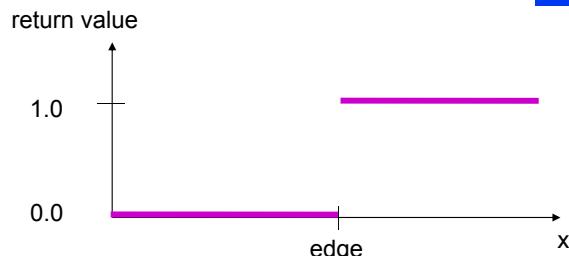
## Eingebaute Funktionen

- Die Funktion `clamp(x, min, max)` sorgt dafür, dass der Wert  $x$  im Intervall  $[min, max]$  liegt
- Die Funktion `mix(x, y, a)` berechnet
$$\text{mix}(x, y, a) = (1 - a) \cdot x + a \cdot y$$
- Die Funktion `faceforward(N, I, Nref)` liefert  $N$  falls  $\langle N, I \rangle$  kleiner 0, ansonsten  $-N$
- Die Funktion `reflect(I, N)` liefert die Reflektion des Eingangsvektors  $I$  an der Normalen  $N$

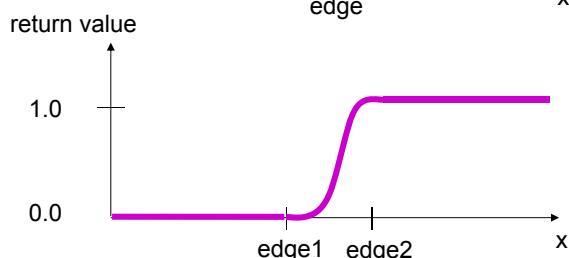


## Eingebaute Funktionen

- `step(  
edge,  
x)`



- `smoothstep(  
edge1,  
edge2,  
x)`



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [381]



© R. Dörner

## Eingebaute Funktionen

- Funktionen für den Zugriff auf Texturen  
`texture1D, texture1DProj, texture1DLod,`  
`texture1DProjLod, ..., texture3DProjLod,`  
`textureCube, textureCubeLod, shadow1D,`  
`shadow1DProj, shadow1DLod,`  
`shadow1DProjLod, shadow2D, ... ,`  
`shadow2DProjLod`
- Noise Funktionen  
`noise1, noise2, noise3, noise4`

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [382]



© R. Dörner



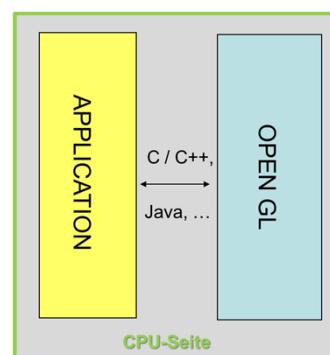
## Rendering und OpenGL

- G.1 Pipeline-Rendering
- G.2 Shader-Programmierung
- G.3 CPU-Seite und GPU-Seite



## OpenGL

- Graphics Library
  - unabhängig vom Betriebssystem / Windows System
  - unabhängig von einer bestimmten Programmiersprache
- Für konkrete Anwendung muss Auswahl getroffen werden
  - Umsetzung in einer Programmiersprache (z.B. Java, C++, Javascript, Python, ...)
  - Anbindung an die Grafikfunktionen des Betriebssystems (z.B. durch spezielle Libraries wie GLUT, WGL oder GLX oder durch Nutzung eines Webbrowsers)
- Weitere Hilfs-Libraries, z.B. GLU oder three.js, die aber nicht standardisiert sind



# OpenGL und WebGL

- OpenGL wird durch ein Industriekonsortium weiter entwickelt: die Khronos Group ([www.khronos.org](http://www.khronos.org))
  - Neben OpenGL wird auch standardisiert:
    - OpenGL|ES für Embedded Systems
    - WebGL für Nutzung im Webbrowser
    - OpenGL ES und WebGL enthalten einen „abgespeckten“ Funktionsumfang, leichte Unterschiede in Syntax



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-Al-V1 [385]



20

## WebGL und HTML

- Nutzen des Canvas – Elements aus HTML, in das WebGL zeichnet
  - Javascript mit WebGL-Befehlen, die WebGL Realisierung im Browser aufrufen
  - WebGL ist dazu in vielen Browsern (z.B. Chrome, Firefox) implementiert – kein Plugin ist nötig

```
<canvas id="glc"
        width="600" height="600">
    Canvas not supported
</canvas>

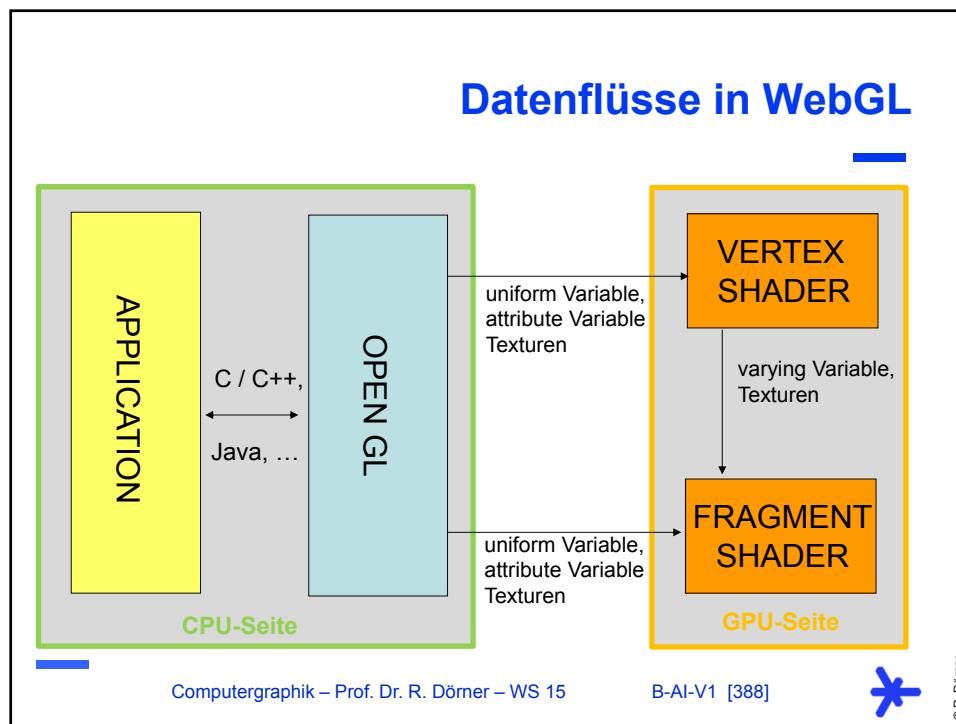
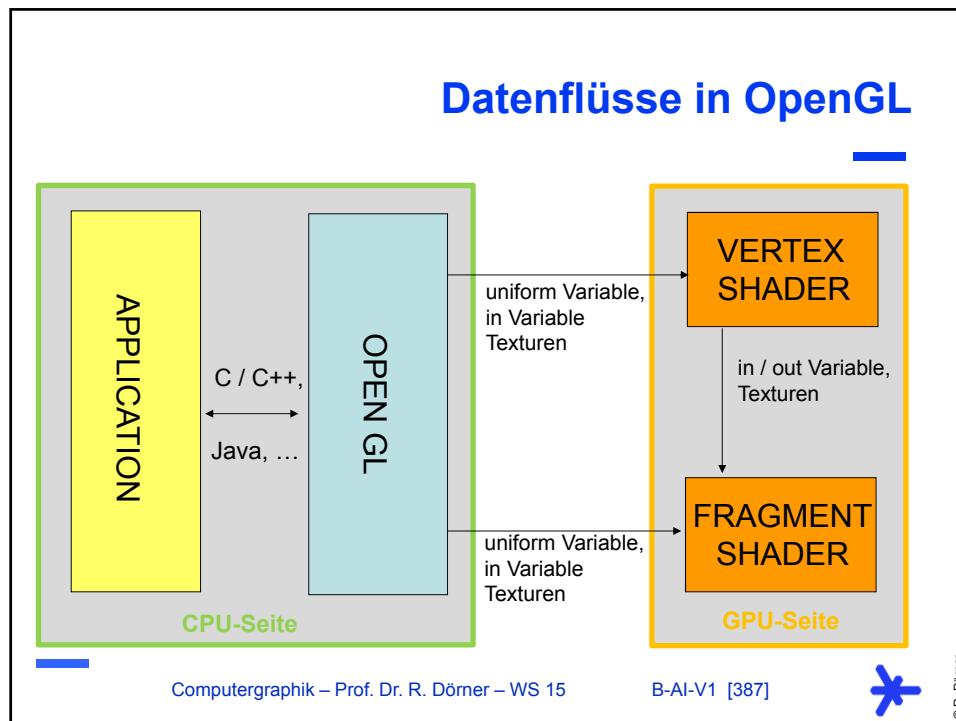
<script type="text/javascript"
        src=". /webgl-utils.js"></script>
<script id="vertex-shader"
        type="x-shader/x-vertex">
Code des Vertex-Shaders
</script>

window.onload = function init() {
    canvas =
        document.getElementById("glc" );
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) {
        alert( "WebGL isn't available" );
    }
    gl.clearColor( 0.9, 0.9, 1.0, 1.0 );
    USW.
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

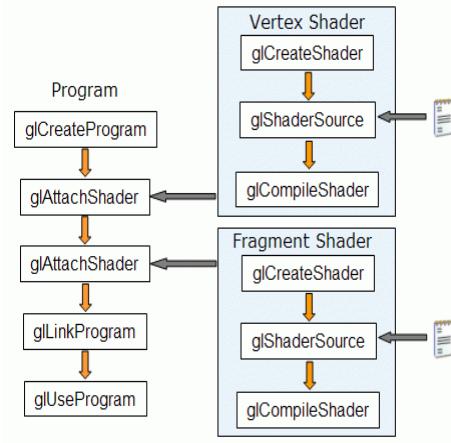
B-AI-V1 [386]





## Einbindung von Shadern

- GLSL-Sourcecode
  - aus Datei einlesen
  - Zeichenreihe im Quellcode (z.B. Javascript)
  - Tag in HTML
- Compiler ist ein Teil des Grafikkarten-Treibers
- OpenGL-Befehle, mit denen man sich die Compiler-Meldungen ausgeben lassen kann



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [389]



© R. Dörner

## Uniform Variablen

- In GLSL
  - `uniform float Variablenname;`
- In OpenGL
  - Herstellen der Verbindung
    - `GLint glGetUniformLocation(GLhandle program, const char* name)`
  - Setzen des Wertes
    - `void glUniform[1,2,3,4]f(GLint location, GLfloat value)`
    - `GLint glUniform[1,2,3,4]fv(GLint location, GLsizei count, GLfloat *v)`

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [390]



© R. Dörner



## Beispiel Uniform Variable in WebGL

```
gl.useProgram(program);

var aFlag = true;
var aNumber = 0.4;
uniform float Variablenname;
gl.uniform1i(
  gl.getUniformLocation(program,
  „flag”),aFlag);
gl.uniform1f(
  gl.getUniformLocation(program,
  „alphaValue”),aNumber);
```

CPU- Seite

```
uniform bool flag;
uniform float alphaValue;

vec4 myColor =
(1.0, 0.0, 0.0, alphaValue);
```

GPU-Seite

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [391]



## Teil H: Vertex Operationen

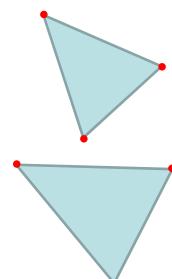


## Vertex Operationen

- H.1 Übergabe der Vertex-Informationen
- H.2 Projektion
- H.3 Umrechnung Koordinatensysteme
- H.4 Aufgaben des Vertex-Shader



## Vertex-Informationen



im Beispiel: 6 Eckpunkte (Vertices)

Jedem Eckpunkt sind Informationen zugeordnet:

- Position (x-Koordinate, y-Koordinate, z-Koordinate)
- Koordinate der Normalen am Eckpunkt
- Texturkoordinate am Eckpunkt
- ...

Informationen in Arrays zusammenstellen (auf CPU-Seite) und an GPU-Seite übergeben

im Beispiel: Vertex-Shader wird 6x aufgerufen

jeder Vertex (mit zugehörigen Daten) wird einzeln (evtl. parallel zu anderen Vertices) verarbeitet

	V1	V2	V3	V4	V5	V6
Position						
Normal						
TexCoord						



## Attribute-Variable

Attribute-Variable für  
Normal wird beim  
Aufruf für Vertex V3  
mit den Werten aus  
diesem Feld belegt

	V1	V2	V3	V4	V5	V6
Position						
Normal						
TexCoord						

- auf GPU-Seite Variable mit dem Schlüsselwort **attribute** deklarieren
- diese Variablen werden jeweils für jeden Aufruf des Vertex-Shaders mit den passenden Daten aus den Arrays gefüllt
- Daten, die für alle Vertices gleich sind (z.B. Position einer Lichtquelle), sollten nur einmal übergeben werden: keine Attribute-Variable, sondern Uniform-Variable nutzen



## OpenGL-Arrays

```
// Array füllen
var pArray = [];
pArray.push(1.0, 2.3, -1.5);
usw.

// Übergabe an Shader
var pBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, pBuffer);
gl.bufferData(gl.ARRAY_BUFFER,
              flatten(pArray),
              gl.STATIC_DRAW);
var vPos = gl.getAttribLocation(program,
                                  "vPosition");
gl.VertexAttribPointer(vPos, 4, gl.FLOAT,
                      false, 0, 0);
gl.enableVertexAttribArray(vPos);
```

CPU-Seite

```
attribute vec3
vPosition;
```

GPU-Seite



## OpenGL-Arrays

- Die Funktion `flatten` ist selbst zu schreiben und muss die Daten auf eine bestimmte Weise strukturieren und im Speicher ablegen
- Im Shader kann angegeben werden, wie diese Struktur aussieht, damit die Daten richtig eingelesen werden
- OpenGL arbeitet als Zustandsautomat, z.B.
  - `glEnable` gilt solange bis `glDisable` aufgerufen wird
  - `glBind` bindet einen Buffer, bis ein anderer Buffer gebunden wird, solange beziehen sich alle Befehle auf den aktuell gebundenen Buffer

```
layout (std140) uniform  
Matrices{  
    mat4 pmvMat; };
```



## Zeichnen mit OpenGL

- Befehl zum Zeichnen sorgt für das Aufrufen der Renderpipeline, die Vertices werden dabei auf die verfügbaren Renderpipelines verteilt
- Mit `glDrawArrays` werden die Daten aus allen „aktivierten“ Vertex-Attrib-Arrays übergeben
- Alternativ kann auch wie beim IndexedFaceSet in VRML ein Index-Array verwendet werden:  
`glDrawElements`

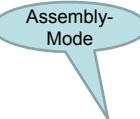
Assembly-  
Mode

```
gl.drawArrays( gl.TRIANGLES,  
               0, numVertices );
```



## Assembly Mode

- Assembly Mode gibt an, wie die Vertices zu Grundprimitiven zusammengesetzt werden sollen
- WebGL kennt folgende Primitive:
  - POINTS
  - LINES
  - LINE\_STRIP
  - LINE\_LOOP
  - TRIANGLES
  - TRIANGLE\_STRIP
  - TRIANGLE\_FAN



```
gl.drawArrays( gl.TRIANGLES,  
                0, numVertices );
```

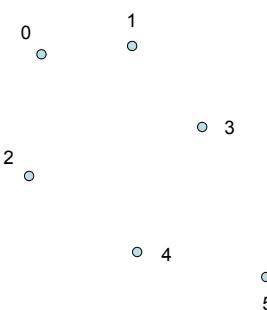
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [399]



## Assembly Mode

POINTS



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [400]

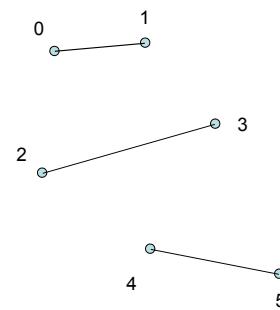


©R. Dörner



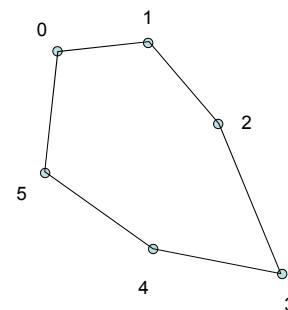
## Assembly Mode

LINES



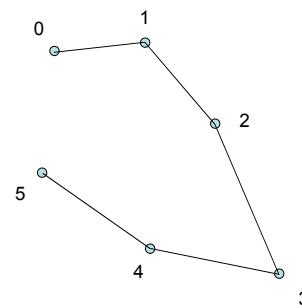
## Assembly Mode

LINE\_LOOP



## Assembly Mode

LINE\_STRIP



Computergraphik – Prof. Dr. R. Dörner – WS 15

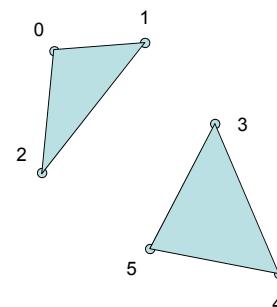
B-AI-V1 [403]



© R. Dörner

## Assembly Mode

TRIANGLES



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [404]

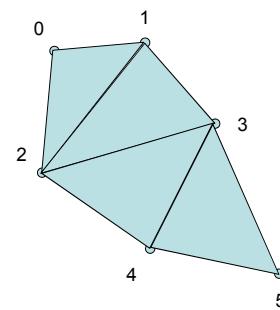


© R. Dörner



## Assembly Mode

TRIANGLE\_STRIP



Computergraphik – Prof. Dr. R. Dörner – WS 15

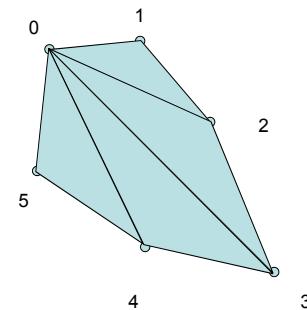
B-AI-V1 [405]



© R. Dörner

## Assembly Mode

TRIANGLE\_FAN



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [406]

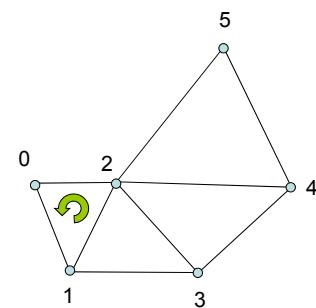


© R. Dörner



## TriangleStrips

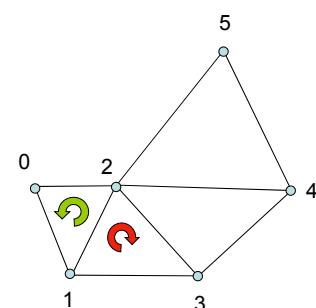
- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



Strip: 0, 1, 2, 3, 4, 5

## TriangleStrips

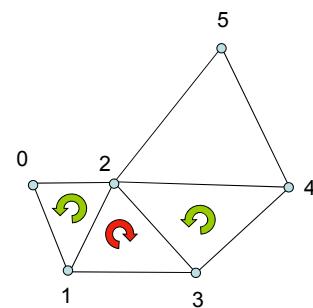
- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



Strip: 0, 1, 2, 3, 4, 5

## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem

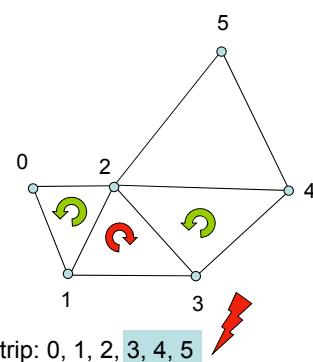


Strip: 0, 1, 2, 3, 4, 5



## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem

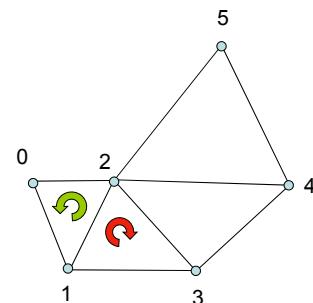


Strip: 0, 1, 2, 3, 4, 5



## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



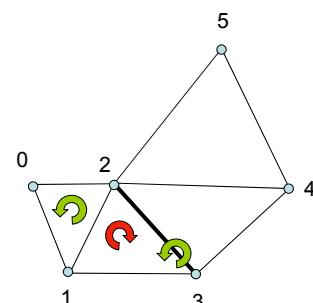
Strip: 0, 1, 2, 3, 4, 5

Strip: 0, 1, 2, 3, 2, 4, 5



## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



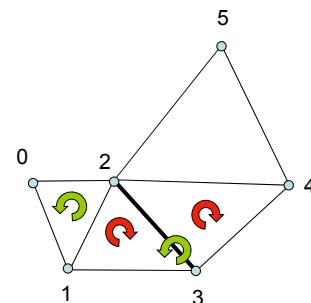
Strip: 0, 1, 2, 3, 4, 5

Strip: 0, 1, 2, 3, 2, 4, 5



## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



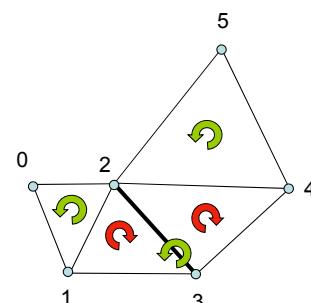
Strip: 0, 1, 2, 3, 4, 5

Strip: 0, 1, 2, 3, 2, 4, 5



## TriangleStrips

- Umlaufsinn beachten:  
„Richtungswechsel“ nur  
durch entartete Dreiecke  
möglich
- Finden eines optimalen  
Trianglestrips ist NP-  
vollständiges Problem



Strip: 0, 1, 2, 3, 4, 5

Strip: 0, 1, 2, 3, 2, 4, 5



## Vertex Operationen

H.1 Übergabe der Vertex-Informationen

**H.2 Projektion**

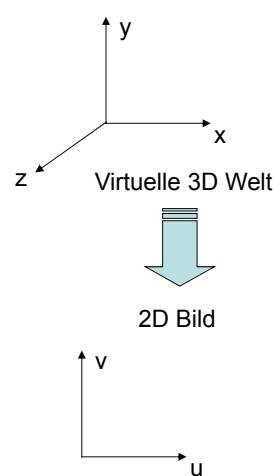
H.3 Umrechnung Koordinatensysteme

H.4 Aufgaben des Vertex-Shader

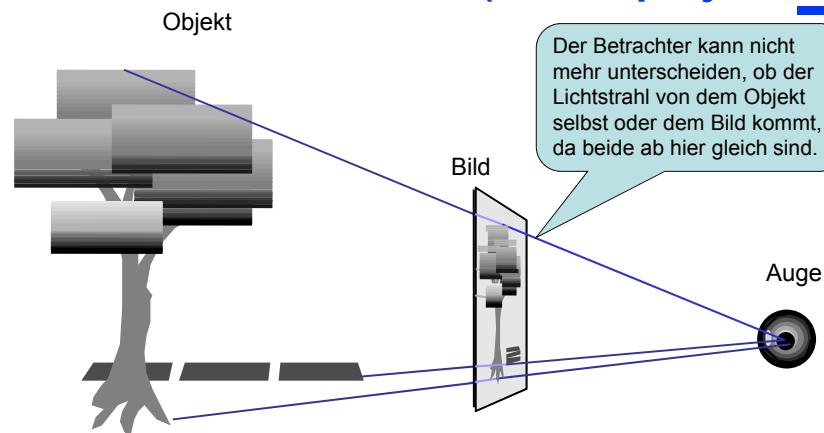


## Projektion

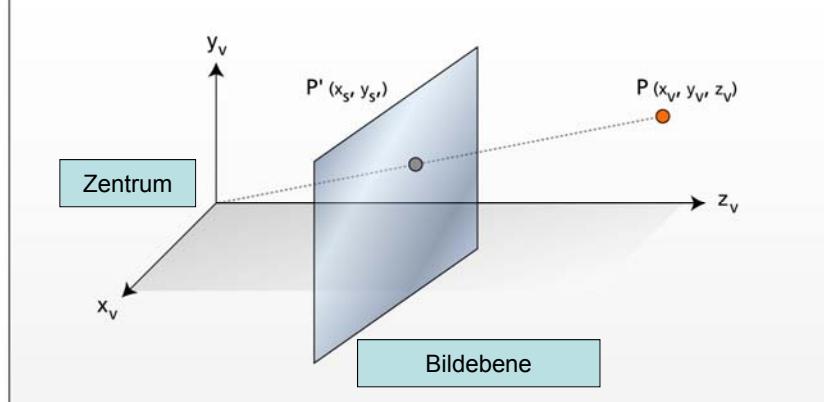
- Geometrische Transformationen wie Skalierung, Rotation, Translation bleiben in der virtuellen Welt
- Im Rendering müssen wir von der 3D Welt auf ein 2D Bild kommen
- Reduktion der Dimensionalität nennt man Projektion



## Perspektivische Projektion (Zentralprojektion)

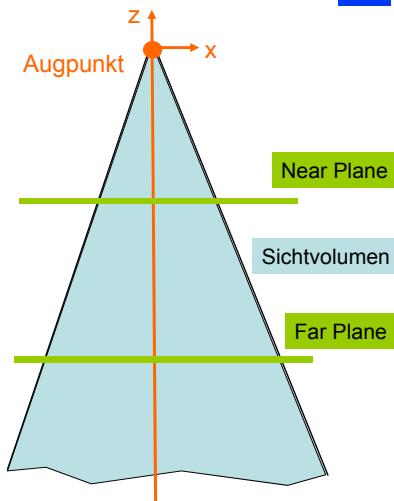


## Ausgangssituation



## Perspektivische Projektion

- Wir gehen von dem Standard View-Koordinatensystem aus
- Die Bildebene (Image Plane) ist identisch mit der Near Plane



Computergraphik – Prof. Dr. R. Dörner – WS 15

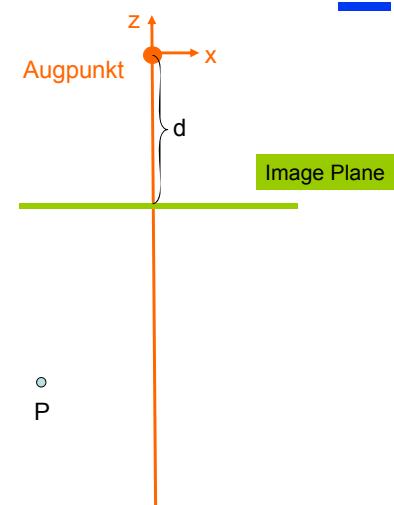
B-AI-V1 [419]



© R. Dörner

## Perspektivische Projektion

- Wir gehen von dem Standard View-Koordinatensystem aus
- Die Bildebene (Image Plane) ist identisch mit der Near Plane
- Die Bildebene befindet sich parallel zur xy-Ebene im Abstand  $d$  zum Augpunkt



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [420]

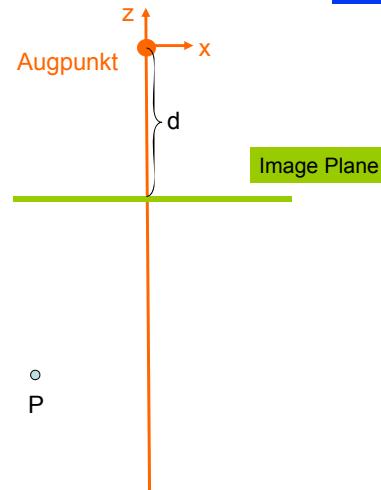


© R. Dörner



## Perspektivische Projektion

- Wir verschieben die Bildebene in die  $xy$ -Ebene



Computergraphik – Prof. Dr. R. Dörner – WS 15

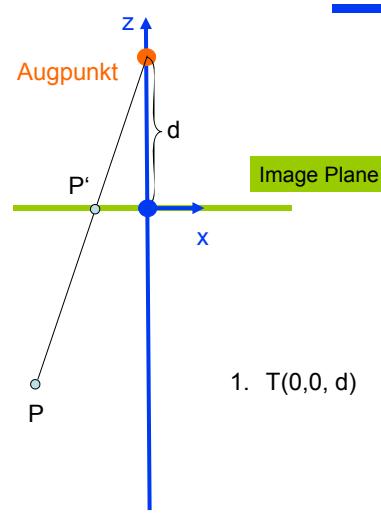
B-AI-V1 [421]



© R. Dörner

## Perspektivische Projektion

- Wir verschieben die Bildebene in die  $xy$ -Ebene
- Die Koordinatenachsen  $u$  und  $v$  des Bildes fallen mit der  $x$ -Achse bzw.  $y$ -Achse zusammen
- Durch Projektion erhalten wir dann die Koordinaten des Bildpunktes  $P'$



1.  $T(0,0, d)$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [422]

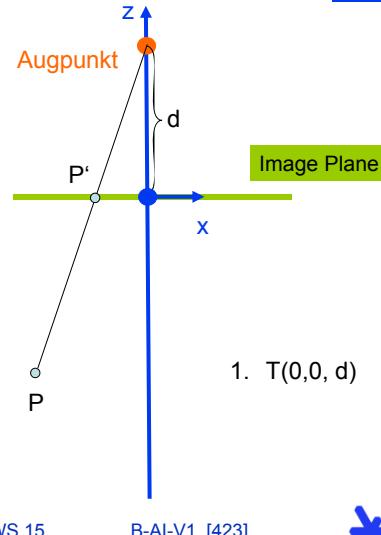


© R. Dörner



## Perspektivische Projektion

- Wir verschieben die Bildebene in die  $xy$ -Ebene
- Die Koordinatenachsen  $u$  und  $v$  des Bildes fallen mit der  $x$ -Achse bzw.  $y$ -Achse zusammen
- Durch Projektion erhalten wir dann die Koordinaten des Bildpunktes  $P'$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [423]

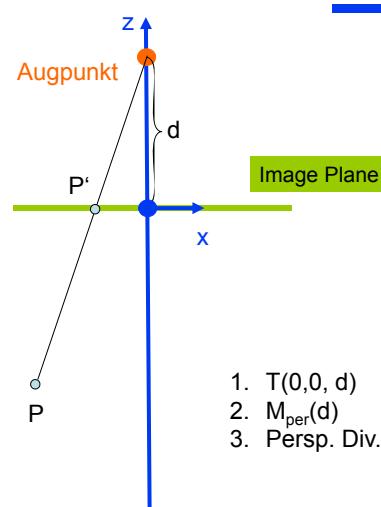
© R. Dörner

## Perspektivische Projektion

- Mathematisch wird die Projektion durch Multiplikation mit der Matrix  $M_{per}(d)$  dargestellt

$$M_{per}(d) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}$$

- Der Wert in homogenen Koordinaten muss noch durch den  $w$ -Anteil dividiert werden (perspektivische Division)



Computergraphik – Prof. Dr. R. Dörner – WS 15

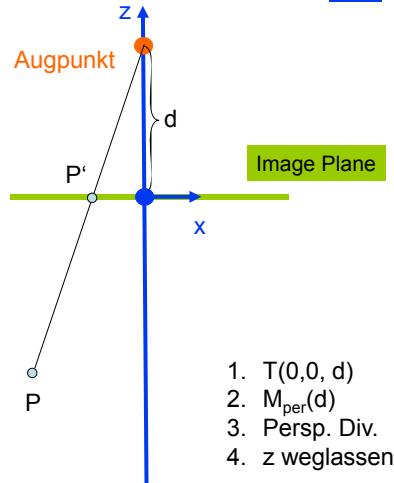
B-AI-V1 [424]

© R. Dörner



## Perspektivische Projektion

- Die  $uv$ -Koordinaten des Bildpunktes  $P'$  entsprechen dann den  $x$  und  $y$ -Koordinaten des Resultats ( $z$ -Koordinate wird weggelassen)



- $T(0,0, d)$
- $M_{per}(d)$
- Persp. Div.
- $z$  weglassen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [425]



© R. Dörner

## Beispiel

Gegeben ist der Punkt  $P(2,1,-2)$  in Viewkoordinaten. Die Bildebene befindet sich auf der negativen  $z$ -Achse in Entfernung 5 vom Augpunkt.

Welche Koordinaten  $P'(u,v)$  wird der  $P$  nach perspektivischer Projektion im Bild haben?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 3 \\ 2/5 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 2 \\ 1 \\ 3 \\ 2/5 \end{pmatrix} \approx \frac{5}{2} \cdot \begin{pmatrix} 2 \\ 1 \\ 3 \\ 7,5 \end{pmatrix} = \begin{pmatrix} 5 \\ 2,5 \\ 7,5 \end{pmatrix} \Rightarrow P'(5/2, 2,5, 7,5)$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [426]



© R. Dörner



## Funktionsweise von $M_{\text{per}}(d)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1-z/d \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \frac{d-z}{d} \end{pmatrix}$$

$$\Rightarrow \vec{p}' = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{d}{d-z} \cdot x \\ \frac{d}{d-z} \cdot y \end{pmatrix}$$

Nach dem Strahlensatz gilt  
(hier für die x-Koordinate):

$$\frac{u}{x} = \frac{d}{d-z}$$

Definitionsfläche  
für  $z = d$

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [427]

© R. Dörner

## Perspektivische Verzerrung

$M_{\text{per}}(d)$  führt eigentlich keine Projektion direkt durch (4D Koordinaten werden ja wieder in 4D Koordinaten verwandelt – und nicht in 2D).

Man kann sich die Wirkung so vorstellen, dass  $M_{\text{per}}$  die Welt verzerrt.

Im Gegensatz zu den anderen Transformationen bleiben Parallelen nicht parallel: keine affine Abbildung mehr.

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [428]

© R. Dörner

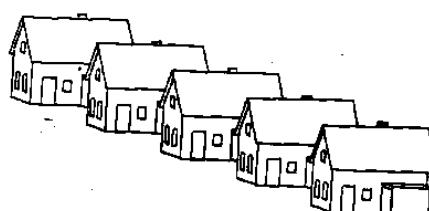
## Weitere Projektionsarten

- Haben bislang nur perspektivische Projektion betrachtet



## Weitere Projektionsarten

- Haben bislang nur perspektivische Projektion betrachtet
- Lassen wir den Abstand  $d$  zwischen Zentrum und Bildebene ins unendliche wachsen, dann sind die Projektionsstrahlen parallel: Parallelprojektion
- Parallelprojektion verkleinert entfernte Objekte nicht



## Parallelprojektion

The diagram illustrates two types of projection. In the top part, labeled 'Parallelprojektion', two points P<sub>1</sub> and P<sub>2</sub> are projected onto a 'Projektionsfläche' (projection plane) as P'<sub>1</sub> and P'<sub>2</sub>. The text 'Projektorien sind parallel' (Projectors are parallel) indicates that the projection rays are parallel to each other. In the bottom part, labeled 'Perspektivische Projektion', the same points P<sub>1</sub> and P<sub>2</sub> are projected onto the same plane, but the projection rays converge to a single point 'Zentrum der Projektion' (Center of projection), which is represented by a small orange dot.

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [431]

© R. Dörner

## Parallelprojektion: Standardfall

The diagram shows a 3D scene with a 3D coordinate system (x, y, z axes). A 2D projection plane is shown at z=0, labeled 'Projektionsebene z=0'. An orange letter 'F' is projected onto this plane. The text 'Sichtvolumen ist ein Quader.' (View volume is a cube) is displayed in a green box. To the right, the transformation matrix for orthographic projection is given as:

$$M_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

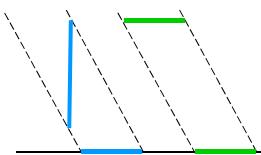
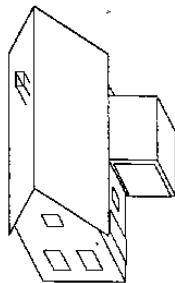
`ortho( l, r, b, t, z, f )`

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [432]

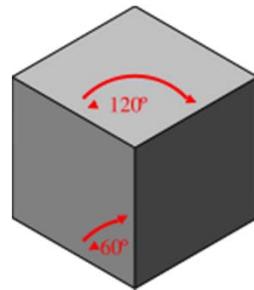
© R. Dörner



## Erhaltung von Längen und Winkeln



Würfel in isometrischer Projektion: Verkürzungsfaktoren in allen Richtungen sind gleich, Seitenlängen und Flächeninhalte im Bild sind gleich



Bei Parallelprojektion werden Längenmaße und Winkel z.T. erhalten: Man kann Werte aus den Bildern ablesen

Computergraphik – Prof. Dr. R. Dörner – WS 15

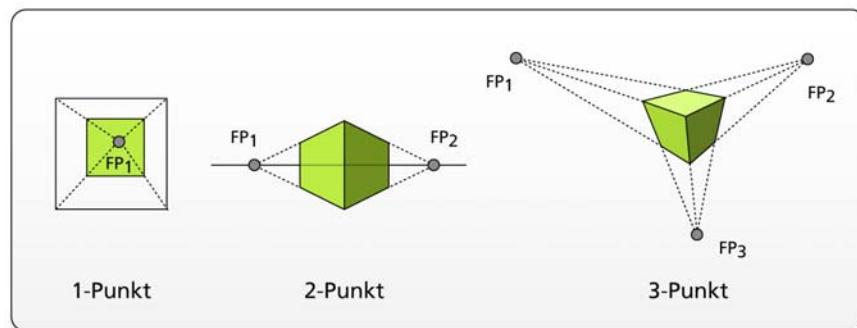
B-AI-V1 [433]



© R. Dörner

## Weitere Projektionsarten

- Verschiedene Projektionsarten bei Zentralprojektion:



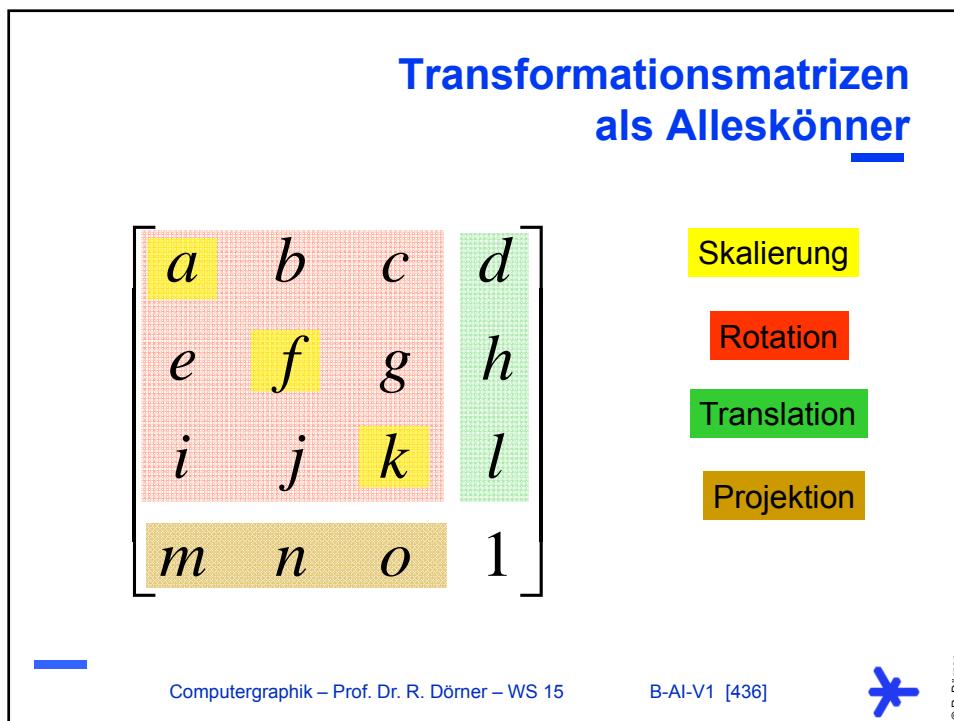
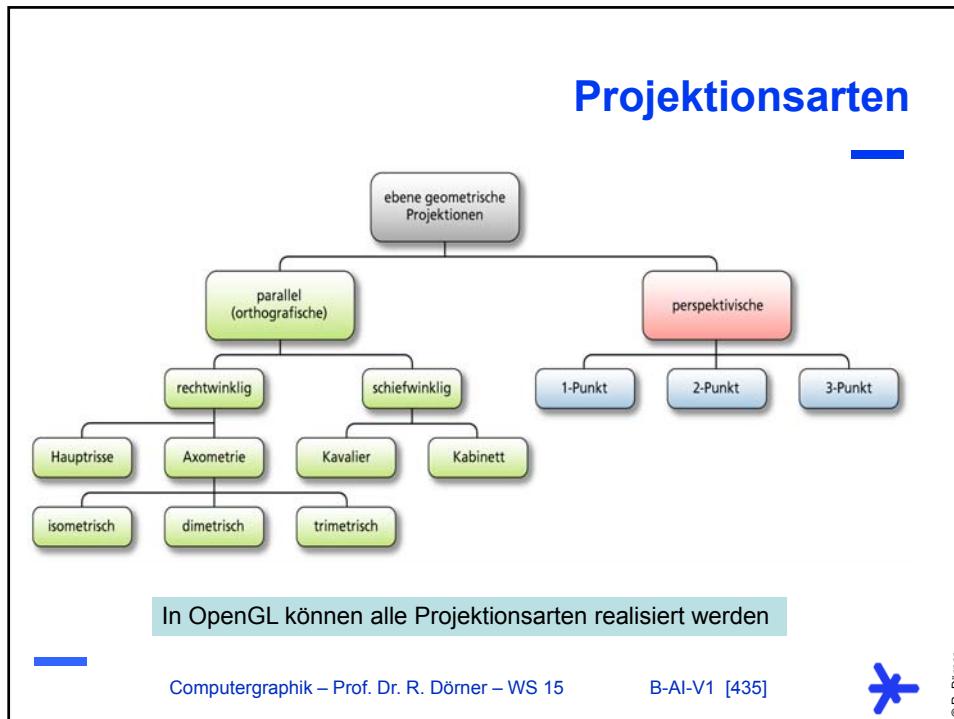
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [434]



© R. Dörner





## Vertex Operationen

H.1 Übergabe der Vertex-Informationen

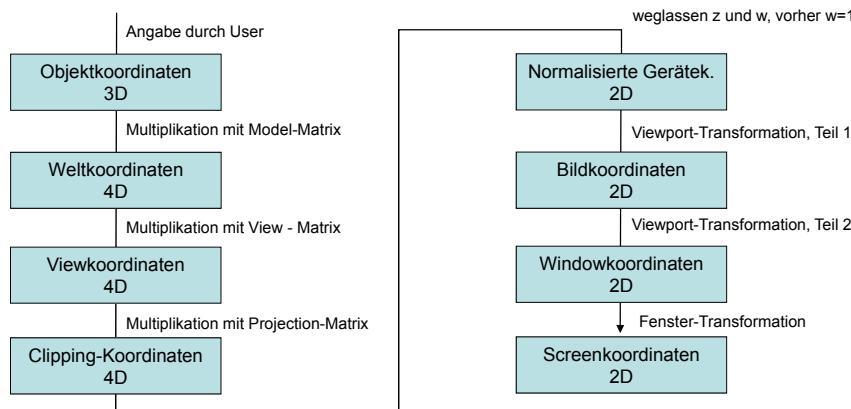
H.2 Projektion

**H.3 Umrechnung Koordinatensysteme**

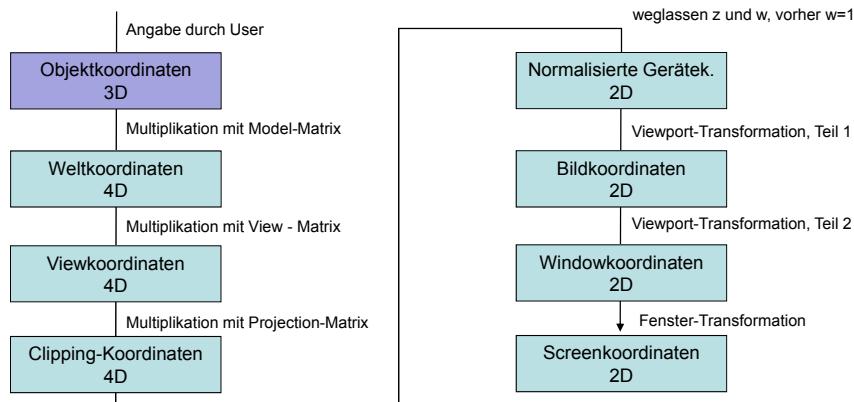
H.4 Aufgaben des Vertex-Shader



## Koordinatensysteme in der GDV



## Koordinatensysteme in der GDV



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [439]



© R. Dörner

## Transformationen in Weltkoordinaten

- Auf CPU Seite muss eine 4x4 Matrix erstellt werden, die beschreibt, wie Vertices transformiert (Skalierung, Rotation, Translation) werden
- Variante 1: Matrix auf Papier ausrechnen und Werte als Konstante ins Programm eintragen
- Variante 2: Hilfsfunktionen auf CPU-Seite schreiben (Erzeugung von Transformationsmatrizen, Matrixoperationen)

mat4():  
erzeugt 4x4 Einheitsmatrix

mult(A, B):  
liefert die 4 x 4 Matrix A · B

rotate(phi, [x, y, z]):  
erzeugt 4x4 Matrix einer Rotation um den Winkel phi um die Achse [x, y, z]

translate(tx, ty, tz):  
erzeugt 4x4 Matrix einer Translation

scalem(sx, sy, sz):  
erzeugt 4x4 Matrix einer Skalierung

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [440]



## Erstellen einer Model-Matrix

```
// Model-Matrix rechnet Objektkoordinaten in Weltkoordinaten um
var model = mat4();

// Model-Matrix als Uniform Variable namens mMat an den Vertex-Shader
gl.uniformMatrix4fv(gl.getUniformLocation(program,"mMat",
    false,flatten(model));

// Körper malen
Daten für Körper laden, drawArrays aufrufen

// Zuletzt Verschieben, davor Rotieren
model = mult(model, translate(2.0, 4.0, 0.0));
model = mult(model, rotate(180.0, [0.0, 0.0, 1.0])); (2) (1)

// Model-Matrix als Uniform Variable namens mMat an den Vertex-Shader
gl.uniformMatrix4fv(gl.getUniformLocation(program,"mMat",
    false,flatten(model));

// Gesicht malen
Daten für Gesicht laden, drawArrays aufrufen
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

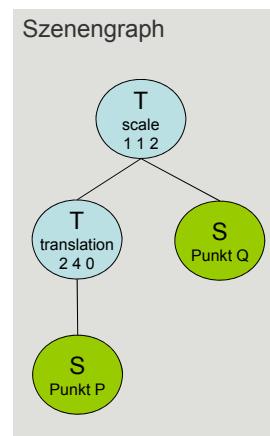
B-AI-V1 [441]



© R. Dörner

## Szenengraphen und OpenGL

- OpenGL kennt keinen Szenengraph, man muss diesen selbst traversieren
- Typische Vorgehensweise: Nutzung eines Stacks für 4x4 Matrizen mit den Methoden `pushMatrix()` und `popMatrix()`
- Stack + Methoden muss man selbst auf CPU-Seite implementieren



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [442]

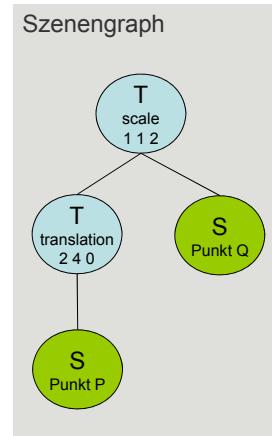


© R. Dörner



## Szenengraphen und OpenGL

```
var model = mat4();
model = mult(model,
             scale(1.0, 1.0, 2.0));
pushMatrix();
model = mult(model,
              translate(2.0, 4.0, 0.0));
zeichne Punkt P
popMatrix();
zeichne Punkt Q
```



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [443]



© R. Dörner

## Szenengraphen und OpenGL

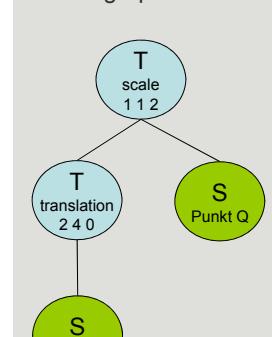
```
var model = mat4();
model = mult(model,
             scale(1.0, 1.0, 2.0));
pushMatrix();
model = mult(model,
              translate(2.0, 4.0, 0.0));
zeichne Punkt P
popMatrix();
zeichne Punkt Q
```

Aktuelle  $M_M$ :

Aktueller Stack:

|||||

Szenengraph



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [444]



© R. Dörner



## Szenengraphen und OpenGL

```

var model = mat4();
model = mult(model,
             scale(1.0, 1.0, 2.0));
pushMatrix();

model = mult( model,
              translate(2.0, 4.0, 0.0));
zeichne Punkt P
popMatrix();

zeichne Punkt Q

```

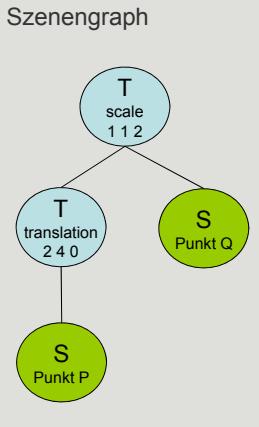
Aktuelle  $M_M$ :

$I * S(1,1,2)$

Aktueller Stack:



Szenengraph



Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [445]
© R. Dörner

## Szenengraphen und OpenGL

```

var model = mat4();
model = mult(model,
             scale(1.0, 1.0, 2.0));
pushMatrix();

model = mult( model,
              translate(2.0, 4.0, 0.0));
zeichne Punkt P
popMatrix();

zeichne Punkt Q

```

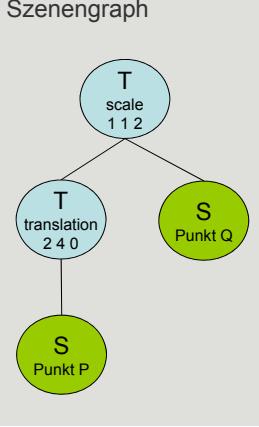
Aktuelle  $M_M$ :

$I * S(1,1,2)$

Aktueller Stack:



Szenengraph



Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [446]
© R. Dörner

## Szenengraphen und OpenGL

```

var model = mat4();
model = mult(model,
    scale(1.0, 1.0, 2.0));
pushMatrix();

model = mult( model,
    translate(2.0, 4.0, 0.0));

zeichne Punkt P
popMatrix();

zeichne Punkt Q

```

Aktuelle  $M_M$ :

$$I * S(1,1,2) * T(2,4,0)$$

Aktueller Stack:

$$I * S(1,1,2)$$


---

Szenengraph

```

graph TD
    T1[T "T scale 1 1 2"] --> T2[T "T translation 2 4 0"]
    T1 --> S1[S "S Punkt Q"]
    T2 --> S2[S "S Punkt P"]

```

Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [447]
© R. Dörner

## Szenengraphen und OpenGL

```

var model = mat4();
model = mult(model,
    scale(1.0, 1.0, 2.0));
pushMatrix();

model = mult( model,
    translate(2.0, 4.0, 0.0));

zeichne Punkt P
popMatrix();

zeichne Punkt Q

```

Aktuelle  $M_M$ :

$$I * S(1,1,2) * T(2,4,0)$$

Aktueller Stack:

$$I * S(1,1,2)$$


---

Szenengraph

```

graph TD
    T1[T "T scale 1 1 2"] --> T2[T "T translation 2 4 0"]
    T1 --> S1[S "S Punkt Q"]
    T2 --> S2[S "S Punkt P"]

```

Computergraphik – Prof. Dr. R. Dörner – WS 15
B-AI-V1 [448]
© R. Dörner

## Szenengraphen und OpenGL

```

var model = mat4();
model = mult(model,
             scale(1.0, 1.0, 2.0));
pushMatrix();

model = mult(model,
             translate(2.0, 4.0, 0.0));
zeichne Punkt P

popMatrix();

zeichne Punkt Q

```

Aktuelle  $M_M$ :

$I * S(1,1,2)$

Aktueller Stack:

Szenengraph

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [449]

© R. Dörner

## Koordinatensysteme in der GDV

Angabe durch User

Objektkoordinaten 3D	Multiplikation mit Model-Matrix
Weltkoordinaten 4D	Multiplikation mit View - Matrix
Viewkoordinaten 4D	Multiplikation mit Projection-Matrix
Clipping-Koordinaten 4D	

weglassen z und w, vorher w=1

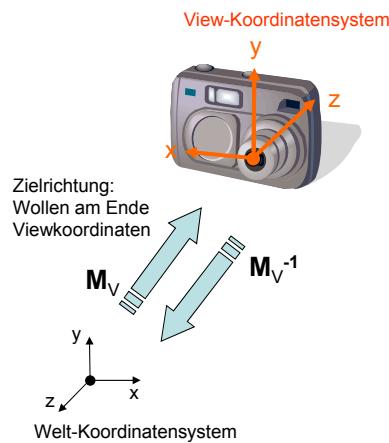
Normalisierte Gerätek. 2D	Viewport-Transformation, Teil 1
Bildkoordinaten 2D	Viewport-Transformation, Teil 2
Windowkoordinaten 2D	Fenster-Transformation
Screenkoordinaten 2D	

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [450]

© R. Dörner

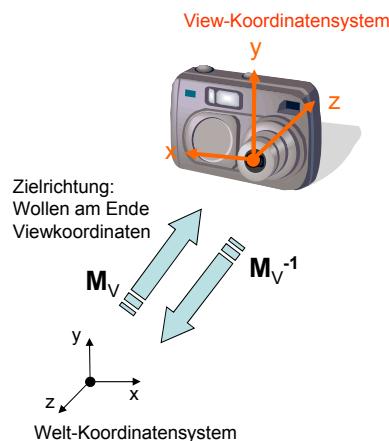
## Weltkoordinaten vs. Viewkoordinaten

- Es ist einfacher die Position der Kamera in der Welt zu beschreiben als umgekehrt:  
Verwendung von Weltkoordinaten zur Spezifikation der äußeren Orientierung
- Mathematisch betrachtet:  
Wechsel von Koordinatensystemen, die mit View-Matrix  $M_V$  beschrieben werden kann



## Weltkoordinaten vs. Viewkoordinaten

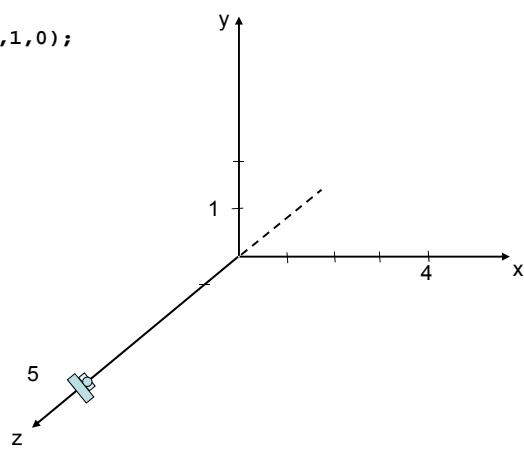
- Die View-Matrix kann durch folgende Werte spezifiziert werden:
  - Punkt: Koordinaten der Kameraposition
  - Punkt: Koordinaten des Viewing Reference Point
  - Vektor: Up-Vektor
- Eine Funktion `lookAt` selbst schreiben, die aus den Werten die 4x4 View-Matrix aufstellt
- Alternativ: Die View-Matrix kann durch geometrische Transformationen (Skalierung, Rotation, Translation) aufgebaut werden



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(0,0,5, 0,0,1, 0,1,0);
```

T(0, 0, -5)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [453]

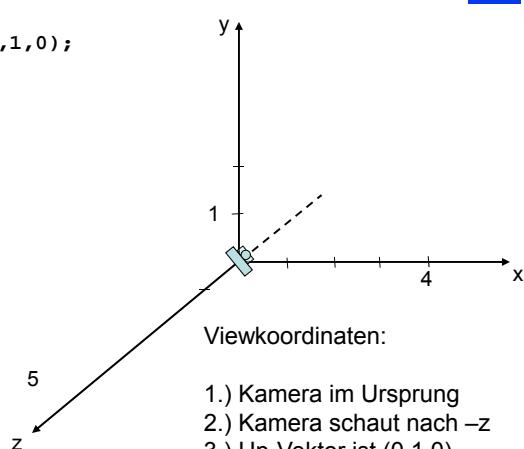


© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(0,0,5, 0,0,1, 0,1,0);
```

T(0, 0, -5)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [454]



© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(0,0,5, 0,0,1, 0,1,0);
```

$$M_v = T(0, 0, -5)$$

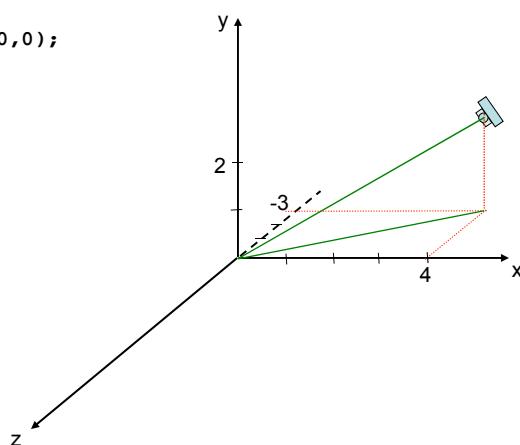
```
var view = mat4();  
view = mult(view, translate(0,0,-5));
```

Diese Befehle  
bewirken  
das Gleiche



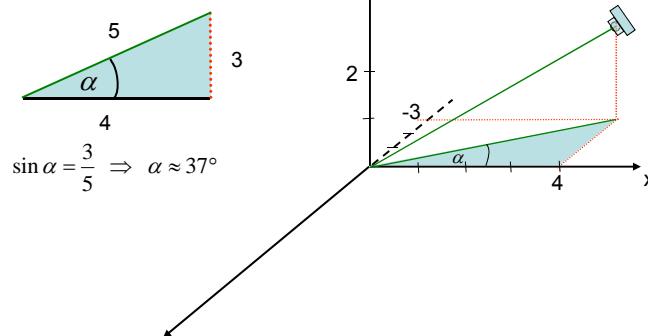
## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

`lookAt(4,2,-3,0,0,0,1,0,0);`



Computergraphik – Prof. Dr. R. Dörner – WS 15

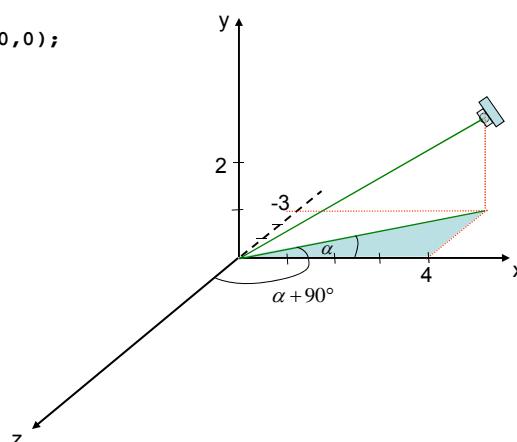
B-AI-V1 [457]



© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

`lookAt(4,2,-3,0,0,0,1,0,0);`



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [458]

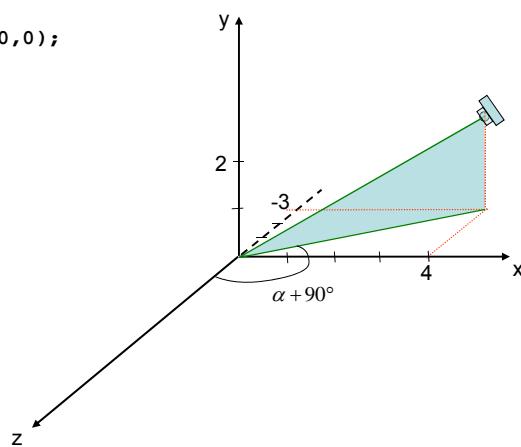


© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

`lookAt(4,2,-3,0,0,0,1,0,0);`



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [459]

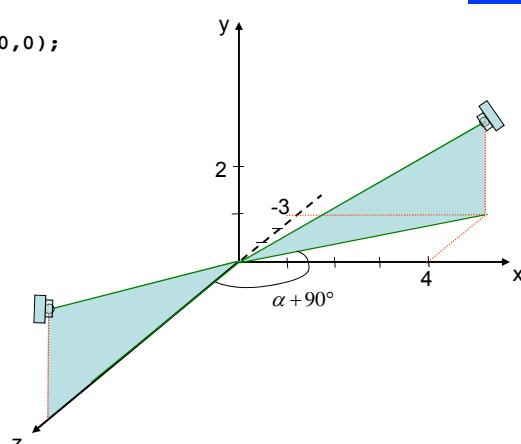


© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

`lookAt(4,2,-3,0,0,0,1,0,0);`

1.  $R_y(-(\alpha + 90^\circ))$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [460]



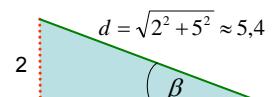
© R. Dörner



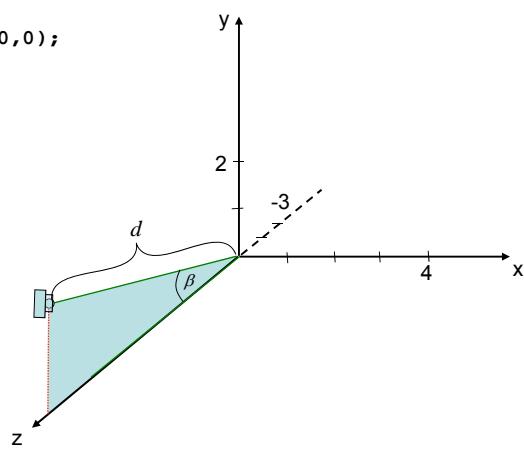
## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$



$$\sin \beta = \frac{2}{\sqrt{2^2 + 5^2}} \Rightarrow \beta \approx 22^\circ$$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [461]

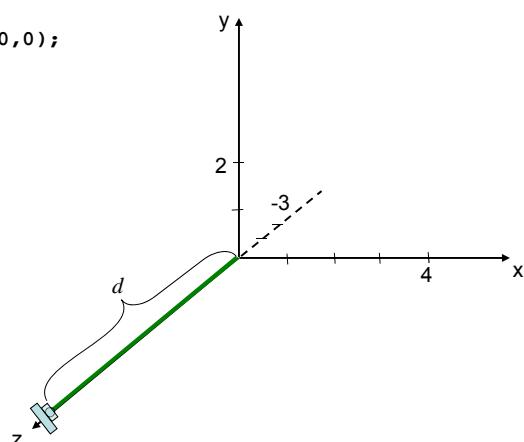


© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [462]



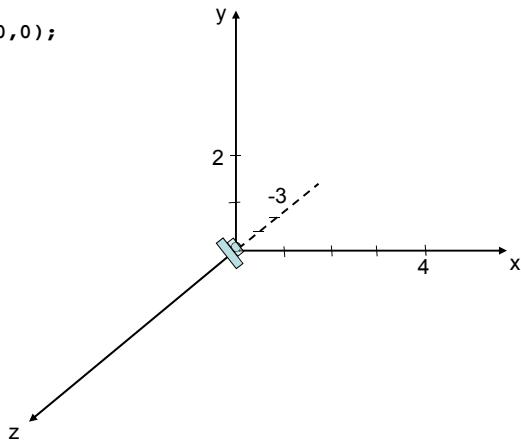
© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [463]



© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

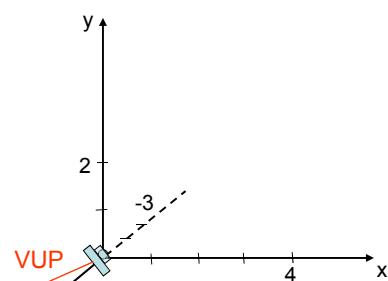
```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$

Der Vektor VUP hat auch alle bisherigen Transformationen mitgemacht.

$$\vec{v}_{up} = T(0,0,-d) \cdot R_x(\beta) \cdot R_y(-\alpha - 90^\circ) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\vec{v}_{up} = \begin{pmatrix} -0,6 \\ -0,3 \\ -0,74 \end{pmatrix}$$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [464]



© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

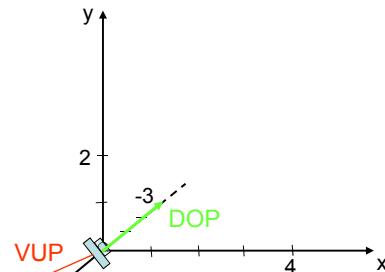
`lookAt(4, 2, -3, 0, 0, 0, 1, 0, 0);`

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$

Wir müssen nun dafür sorgen, dass VUP in y-Richtung zeigt.

Dabei muss aber die Blickrichtung (DOP) nach wie vor in Richtung negativer z-Achse zeigen.

Das klappt nur dann problemlos, wenn VUP und DOP senkrecht aufeinander stehen.



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [465]



© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

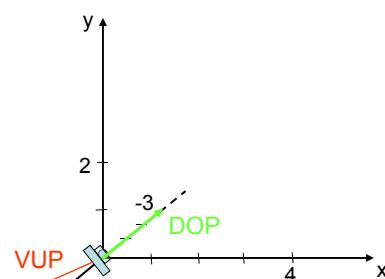
`lookAt(4, 2, -3, 0, 0, 0, 1, 0, 0);`

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$

In unserem Beispiel ist das nicht der Fall. Wir ersetzen daher den Vektor VUP durch einen Vektor, den wir durch Projektion von VUP auf die Ebene erhalten, von der DOP der Normalenvektor ist (Ebene parallel zur Bildebene):

$$\vec{v} = \vec{v}_{up} - \frac{\langle \vec{v}_{up}, \vec{n} \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n}$$

$$\vec{v} = \begin{pmatrix} -0,6 \\ -0,3 \\ 0 \end{pmatrix}$$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [466]



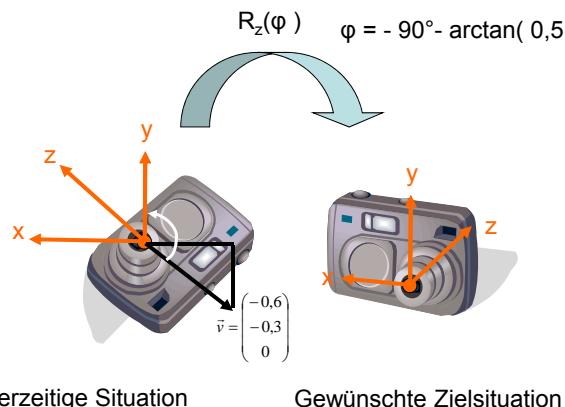
© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$
4.  $R_z(\varphi)$



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [467]

© R. Dörner

## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$
4.  $R_z(\varphi)$

$$M = R_z(\varphi) \cdot T(0, 0, -d) \cdot R_x(\beta) \cdot R_y(-(\alpha + 90^\circ))$$

M rechnet Weltkoordinaten in Viewkoordinaten um

```
glRotatef(phi,0,0,1);
glTranslatef(0,0,-d);
glRotatef(beta,1,0,0);
glRotatef(-(alpha+90),0,1,0);
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [468]

© R. Dörner



## Beispiel: Umrechnung Welt- zu Viewkoordinaten

```
lookAt(4,2,-3,0,0,0,1,0,0);
```

1.  $R_y(-(\alpha + 90^\circ))$
2.  $R_x(\beta)$
3.  $T(0, 0, -d)$
4.  $R_z(\varphi)$

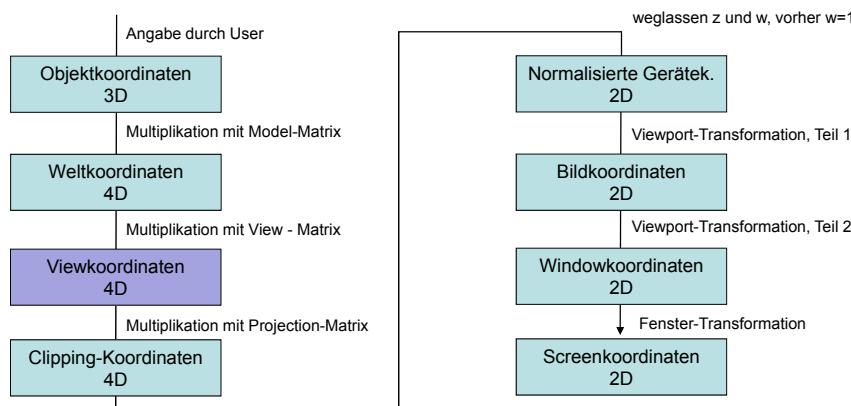
$$M = R_z(\varphi) \cdot T(0, 0, -d) \cdot R_x(\beta) \cdot R_y(-(\alpha + 90^\circ))$$

```
var view = mat4();
view = mult(view, rotate(phi,[0,0,1]));
view = mult(view, translate(0,0,-d));
view = mult(view, rotate(beta,[1,0,0]));
view = mult(view, rotate(-(alpha+90),[0,1,0]));
```

Diese Befehle bewirken das Gleiche



## Koordinatensysteme in der GDV



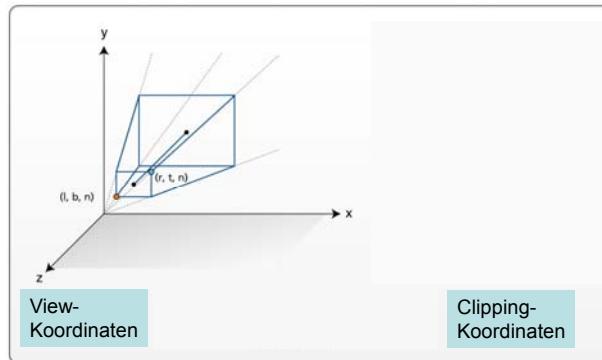
## Clipping Koordinaten

Sichtvolumen hat die Form eines Pyramidenstumpfs (Frustum)

Das Frustum wird spezifiziert durch den linken unteren Punkt  $(l, b)$ , den rechten oberen Punkt  $(r, t)$  sowie den Beträgen der Abstände der beiden Clipping-Planes  $n$  und  $f$ .

Alternativ auch Angabe mit *Sehwinkel, Aspect Ratio, n und f*

Die Matrix  $M_{per\_clip}$  verzerrt das Frustum zu einem Würfel (für perspektivische Projektion)



`frustum(l, r, b, t, n, f)`

`perspective(sehWinkel, aspectRatio, n, f)`

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [471]

© R. Dörner

## Clipping Koordinaten

- Spezifikation des Sichtvolumens:  
`left, right, bottom, top, | near |, | far |`
- Matrix für Wechsel von View- zu Clipping Koordinaten:

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

Computergraphik – Prof. Dr. R. Dörner – WS 15

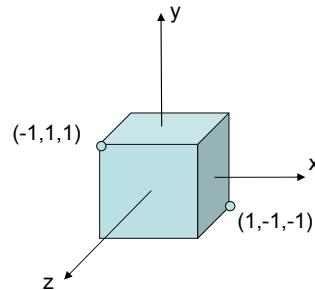
B-AI-V1 [472]

© R. Dörner



## Clipping Koordinaten

- $M_{per\_clip}$  bildet nicht auf irgendeinen Würfel ab, sondern auf einen Einheitswürfel
- Ursprung im Mittelpunkt des Würfels, Seitenlänge 2
- Man kann durch einfache Vergleichoperationen herausfinden, ob ein Punkt im Sichtvolumen liegt oder nicht: alle Koordinatenwerte müssen in  $[-1,1]$  liegen



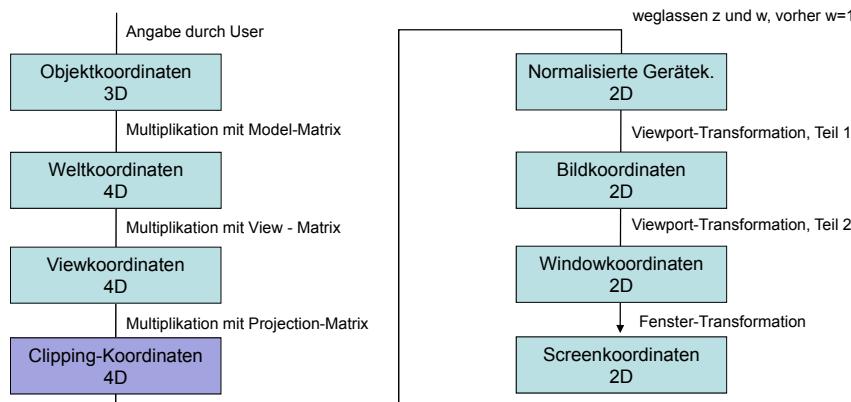
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [473]



© R. Dörner

## Koordinatensysteme in der GDV



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [474]

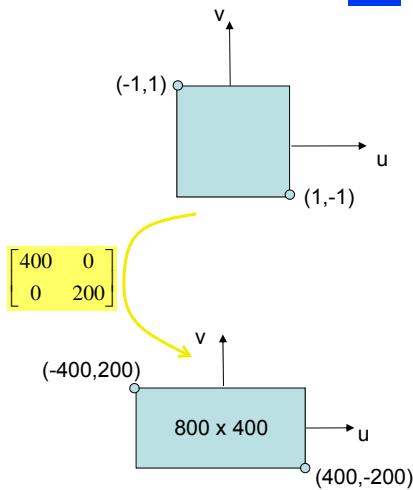


© R. Dörner



## Normalisierte Gerätekordinaten

- Führt man die Projektion zu Ende (perspektivische Division, weglassen der z-Koordinate) erhält man normalisierte Gerätekordinaten
- Das Bild ist dann ein Quadrat der Seitenlänge 2
- Das Bild kann einfach auf eine beliebige Größe gebracht werden



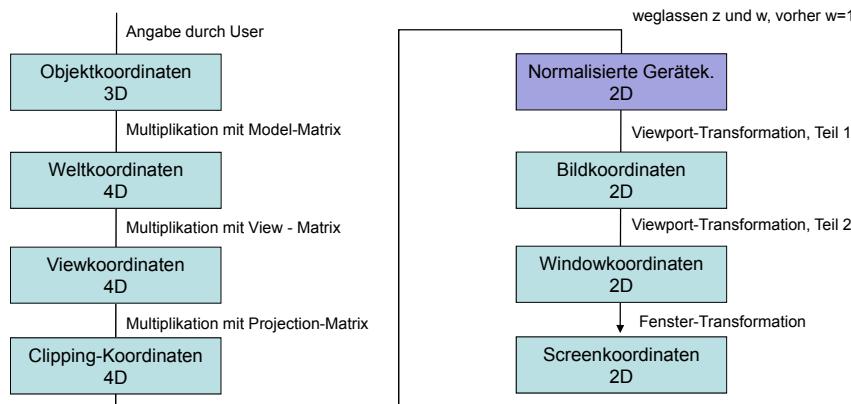
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [475]



© R. Dörner

## Koordinatensysteme in der GDV



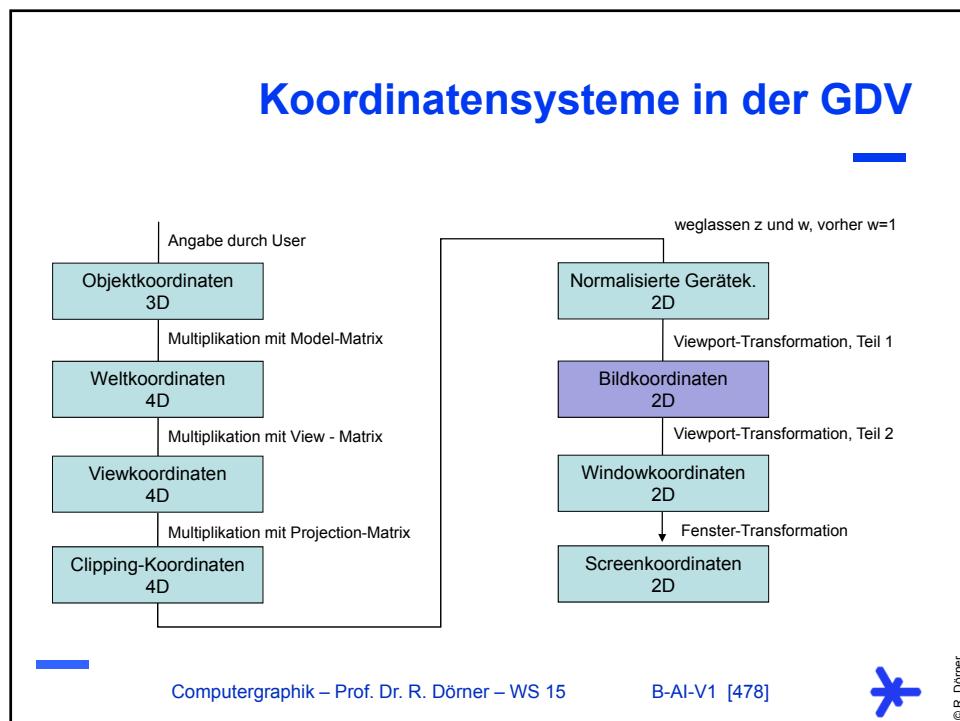
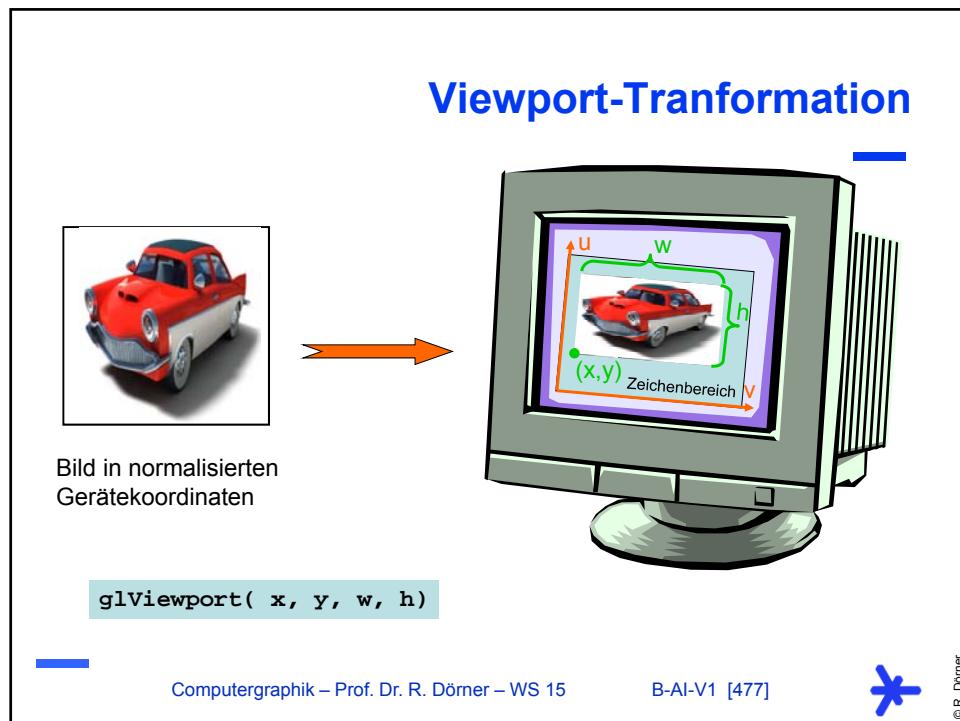
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [476]

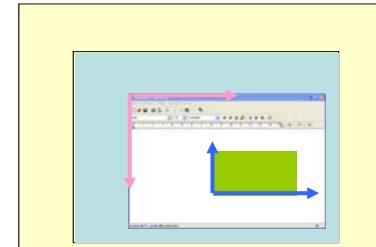


© R. Dörner

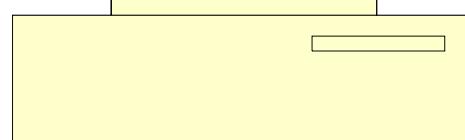




## Viewport Koordinaten, Window Koordinaten und Screen Koordinaten



Viewport =  
Bereich im Window



Window Koordinaten

Viewport Koordinaten

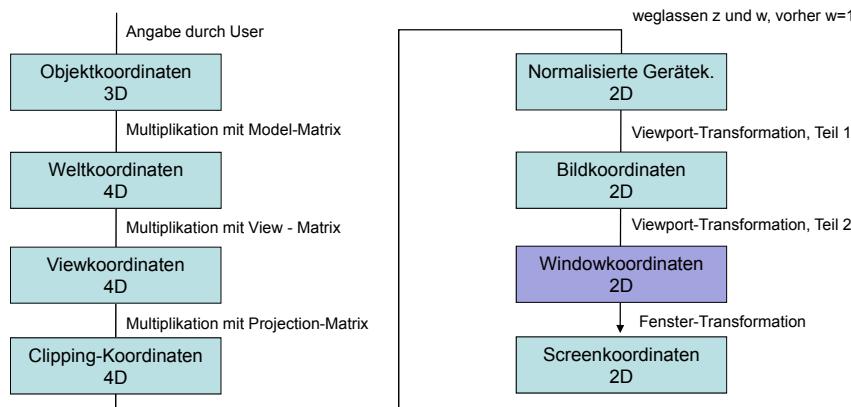
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [479]



© R. Dörner

## Koordinatensysteme in der GDV



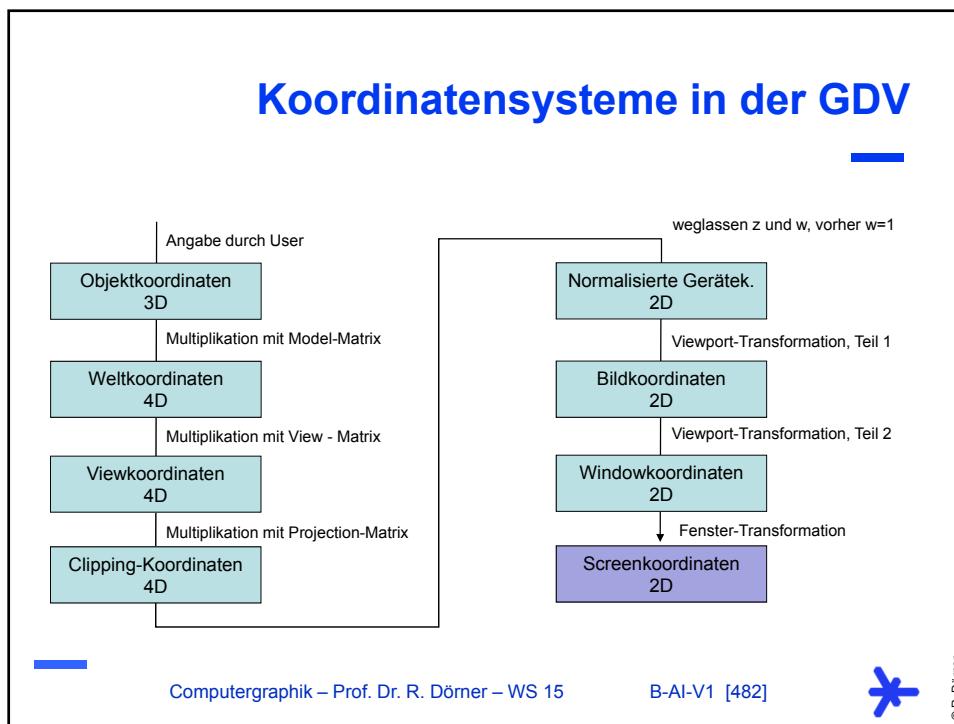
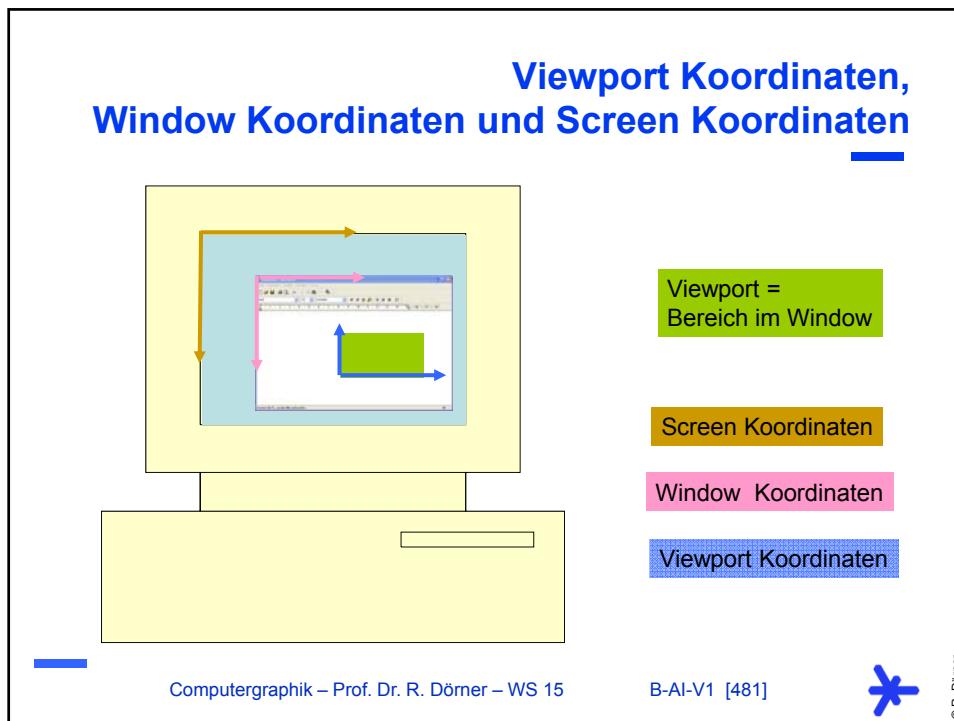
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [480]



© R. Dörner





## Vertex Operationen

- H.1 Übergabe der Vertex-Informationen
- H.2 Projektion
- H.3 Umrechnung Koordinatensysteme
- H.4 Aufgaben des Vertex-Shader**



## Vertex Shader

- Wird ausgeführt für jeden Eckpunkt
- Dabei kann auf Daten von anderen Eckpunkten nicht zugegriffen werden
- Minimale Aufgabe: Berechnung der Position des Eckpunktes im Bild (ohne perspektivische Division)
  - in OpenGL: Realisierung durch Schreiben der eingebauten Variable `gl_Position` (vom Typ `vec4`)
- Weitere Aufgaben: Beleuchtungsrechnung, Modifikation Texturkoordinaten, Modifikation Normalen, Skinning, Morphing, Animation



## GLSL Beispiel

```
Übergabe der Matrizen
uniform mat4 modelMat;
uniform mat4 viewMat;
uniform mat4 projectionMat;

für diesen Eckpunkt: Position und Farbe
attribute vec3 position;
attribute vec4 vColor;
varying vec4 fColor;

void main(){
    fColor = vColor;
    gl_Position =
        projectionMat * viewMat * modelMat *
        vec4(position, 1);
}
```

Übergabe des Wertes an Fragment-Shader

keine Beleuchtungsrechnung: der Farbwert wird einfach weitergereicht

Ergebnis des Vertex Shaders wird in vordefinierte Variable geschrieben

Konstruktor für 4D-Vektor, da 4x4 Matrix

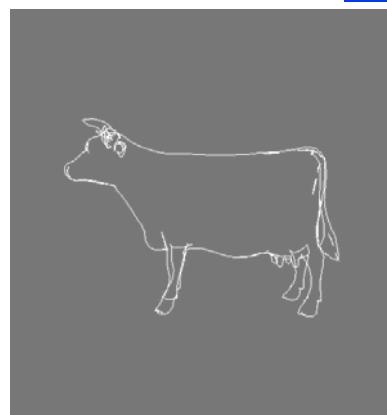
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [485]



## Geometry Shader

- Ausführung nach Vertex-Shader
- Input: komplettes Primitiv (z.B. eine Linie, ein Dreieck)
- Output: verändertes Primitiv oder zusätzliche Primitive
- Einsatzgebiet - Beispiele:
  - Erstellung zusätzlicher Linien, um eine Kurve zu nähern
  - Änderung der Primitiven (z.B. Verwandlung Dreiecke in Linien für NPR)



Quelle:  
NVIDIA

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [486]

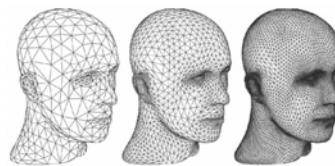


© R. Dörner



## Tesselation Shader

- Motivation
  - neu berechnen statt speichern
  - LODs
  - Anpassung an Kontext (Bildschirmauflösung, Systemleistung)
- Tesselation Control Shader
  - Umfang der Tesselation bestimmen
  - Transformation der Eingangsdaten (z.B. um Lücken zwischen Patches zu vermeiden)
- Tesselation Evaluation Shader
  - Fixed Function Tesselation ist allgemeiner Algorithmus, der auf einem abstrakten Patch arbeitet
  - Nimmt abstrakte Koordinaten und berechnet tatsächliche Werte für die generierten Vertices



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [487]

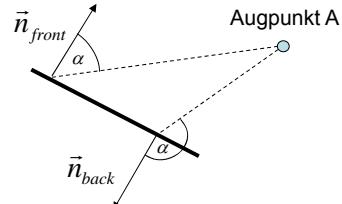


## Teil I: Culling, Clipping & Rasterisierung



## Culling

- Entfernen von Primitiven, die nicht auf dem Bild erscheinen können
- Grund: Rechenzeitzersparnis
- View-Frustum-Culling
  - Alles außerhalb des Sichtvolumens entfernen
  - Einfach in Clipping Koordinaten durchzuführen
- Backface-Culling
  - Entfernen der abgewandten Flächen



Computergraphik – Prof. Dr. R. Dörner – WS 15

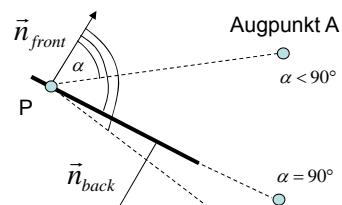
B-AI-V1 [489]



© R. Dörner

## Culling

- Entfernen von Primitiven, die nicht auf dem Bild erscheinen können
- Grund: Rechenzeitzersparnis
- View-Frustum-Culling
  - Alles außerhalb des Sichtvolumens entfernen
  - Einfach in Clipping Koordinaten durchzuführen
- Backface-Culling
  - Entfernen der abgewandten Flächen



$$\text{Vorderseite sichtbar} \Leftrightarrow |\alpha| < 90^\circ$$

$$-90^\circ < \alpha < 90^\circ \Leftrightarrow \frac{\langle \vec{n}_{front}, \vec{a} - \vec{p} \rangle}{|\vec{n}_{front}| \cdot |\vec{a} - \vec{p}|} > 0$$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [490]



© R. Dörner



## Backface-Culling

- In Clipping-Koordinaten ist der View orthographisch
- Die Verbindungsline von einem Punkt zum Augpunkt ist parallel zur z-Richtung
- OpenGL Befehle
  - Zum Einschalten
  - Zur Festlegung, ob Vorder- oder Rückseite entfernt werden soll

Vorderseite sichtbar  $\Leftrightarrow |\alpha| < 90^\circ$

$$-90^\circ < \alpha < 90^\circ \Leftrightarrow \frac{\langle \vec{n}_{front}, \vec{a} - \vec{p} \rangle}{|\vec{n}_{front}| \cdot |\vec{a} - \vec{p}|} > 0$$

$$\text{Clipping - Koordinaten} \Rightarrow \vec{a} - \vec{p} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Vorderseite sichtbar  $\Leftrightarrow n_z > 0$

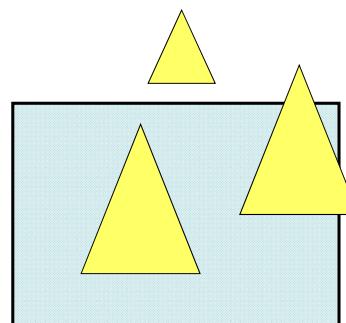
```
glEnable(GL_CULL_FACE)
glCullFace(GL_FRONT)
glCullFace(GL_BACK)
```



## Clipping

- Aus dem Bildbereich / Window sollen keine Objekte herausragen
- Überstehendes abschneiden (Clipping)
- Verschiedene Algorithmen
  - Cohen-Sutherland
  - Liang-Barsky
- Durchführung in Clipping-Koordinaten
- Problem: aus einfachen Primitiven können komplexere werden

Objekte der Szene



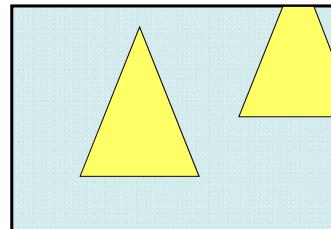
Window



## Clipping

- Aus dem Bildbereich / Window sollen keine Objekte herausragen
- Überstehendes abschneiden (Clipping)
- Verschiedene Algorithmen
  - Cohen-Sutherland
  - Liang-Barsky
- Durchführung in Clipping-Koordinaten
- Problem: aus einfachen Primitiven können komplexere werden

Geklippte Objekte der Szene



Window

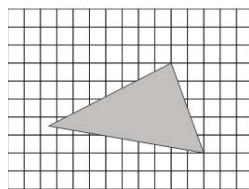
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [493]

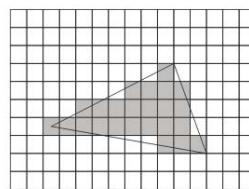


© R. Dörner

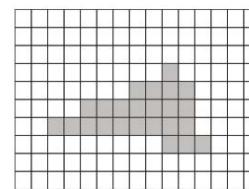
## Rasterisierung



Ein Dreieck vor einem Pixelraster.



Rasterisierung - Pixel können entweder ganz oder gar nicht eingefärbt werden.



Das Ergebnis der Rasterisierung:  
Das Dreieck wurde in 24 Fragmente zerlegt (graue Quadrate von der Größe eines Pixels).

Ein Fragment ist ein Flächenstück von der Größe eines Pixels. Fragmente sind das Ergebnis der Rasterisierung.

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [494]

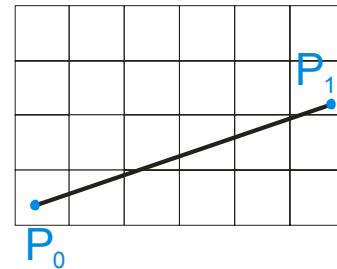


© R. Dörner



## Rasterisieren einer Linie

- Gegeben ist Linie von  $P_0(x_0, y_0)$  nach  $P_1(x_1, y_1)$  mit Steigung  $m$  ( $0 < m < 1$ ) sowie Pixelraster
- Gesucht: Fragmente der Linie
- Mehrere Algorithmen sind dafür bekannt, z.B.
  - DDA (Digital Differential Analyser) Algorithmus
  - Bresenham's Algorithmus (nur Integer-Arithmetik), ist heute Standard



$$m = \frac{\Delta y}{\Delta x} = \frac{y_1 - y_0}{x_1 - x_0}$$

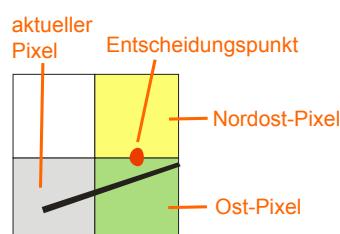
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [495]

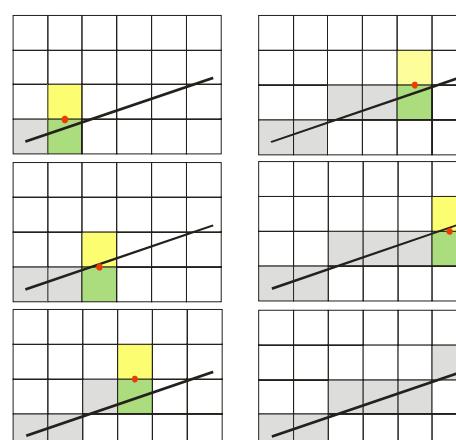


© R. Dörner

## Bresenham's Algorithmus



Entscheidungsregel:  
Liegt die Linie über dem Entscheidungspunkt, so wird der Nordost-Pixel gewählt, ansonsten der Ost-Pixel



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [496]



© R. Dörner



## Bresenham's Algorithmus

```
void zeichneLinie(int x0, int y0, int x1, int y1) {  
    int dy = y1 - y0; int dx = x1 - x0;  
    int entscheidung = 2*dy + dx;  
    int ost = 2*dy; int nordost = 2*(dx + dy);  
    int y = y0;  
    for(int x = x0; x <= x1; x++) {  
        setzePixel(x, y);  
        if ( entscheidung <= 0 ) {  
            entscheidung += ost; } // Ost-Pixel wählen  
        else {  
            entscheidung += nordost;  
            y++; } // Nordost-Pixel wählen  
    }  
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

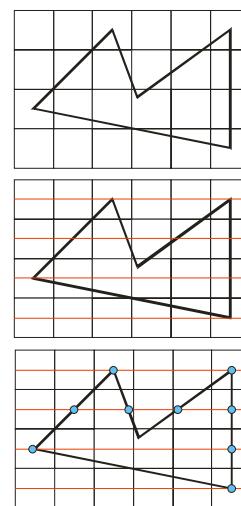
B-AI-V1 [497]



© R. Dörner

## Rasterisierung von Polygonen

- Häufig verwendet: Scanline-Algorithmus
- Bild mit Scanlines durchgehen:
  - Schnittpunkte Polygon mit Scanline finden
  - Sortieren der Schnittpunkte nach wachsender x-Koordinate



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [498]

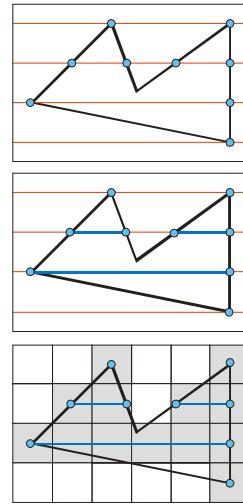


© R. Dörner



## Rasterisierung von Polygonen

- Häufig verwendet: Scanline-Algorithmus
- Bild mit Scanlines durchgehen:
  - Schnittpunkte Polygon mit Scanline finden
  - Sortieren der Schnittpunkte nach wachsender x-Koordinate
  - Spans (Gruppen zusammenhängender Pixel) zwischen den Schnittpunkten setzen: Ausnutzung von Kohärenz



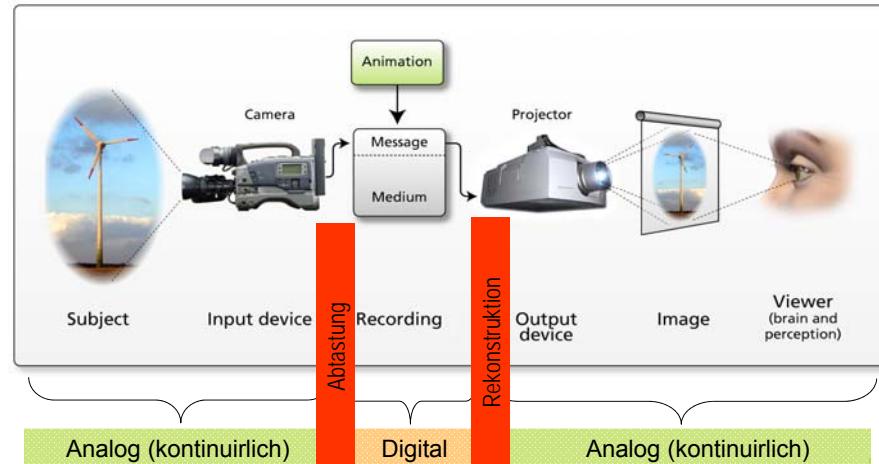
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [499]



© R. Dörner

## Abtastung und Rekonstruktion



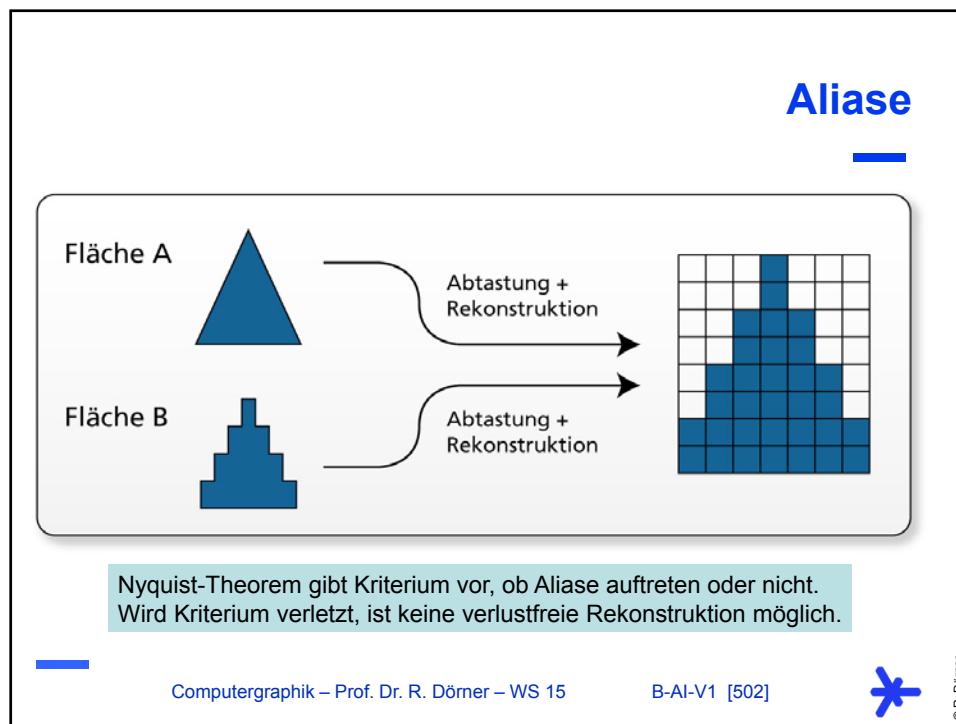
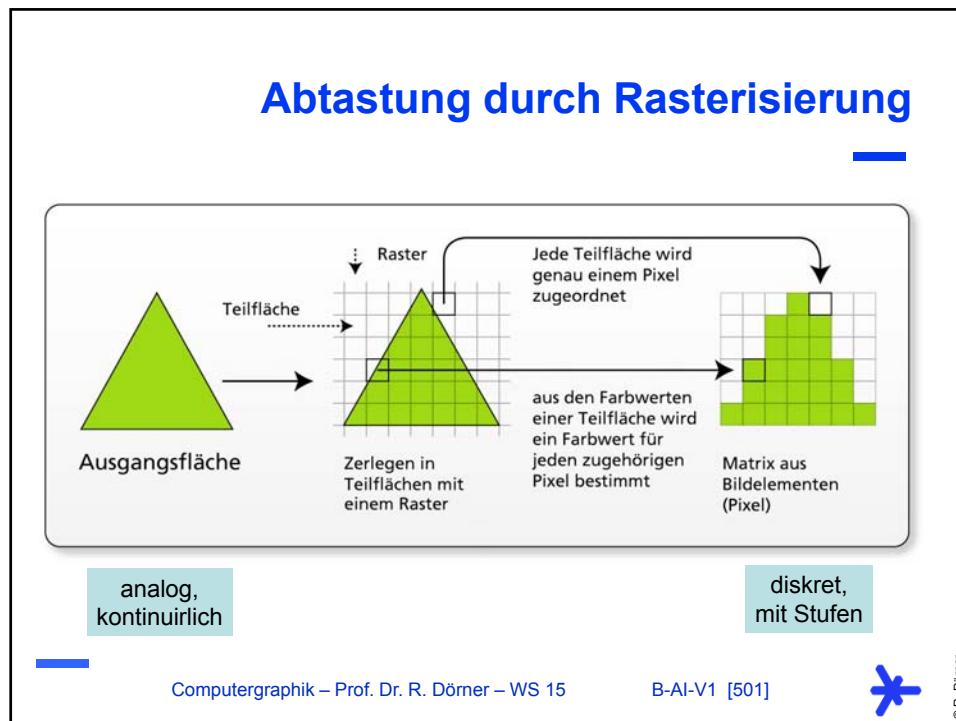
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [500]

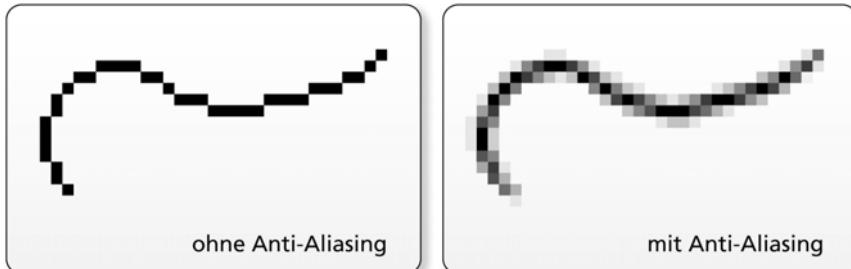


© R. Dörner





## Anti-Aliasing



The figure shows two versions of a jagged circle. The left version, labeled "ohne Anti-Aliasing", has sharp, jagged edges. The right version, labeled "mit Anti-Aliasing", has smooth, rounded edges.

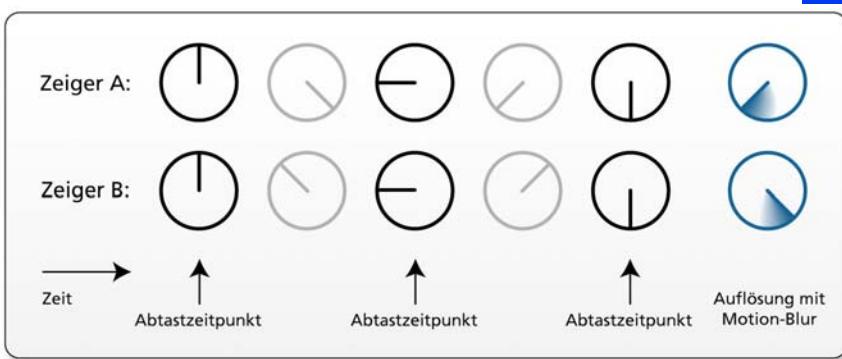
Verschiedene Anti-Aliasing Verfahren

- Filterung mit Tiefpass-Filter (Bild wird unschärfer)
- Intern mit höherer Auflösung arbeiten (Super-Sampling)

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [503]

© R. Dörner

## Zeitliches Anti-Aliasing



The diagram illustrates motion blur in the time domain. It shows two sets of circles representing the state of objects over time. The top row is labeled "Zeiger A" and the bottom row "Zeiger B". Arrows indicate the progression of time from left to right. At each time step, an arrow points to a circle labeled "Abtastzeitpunkt". The final result is labeled "Auflösung mit Motion-Blur".

Aliasing-Effekte treten auch durch diskrete Frames auf

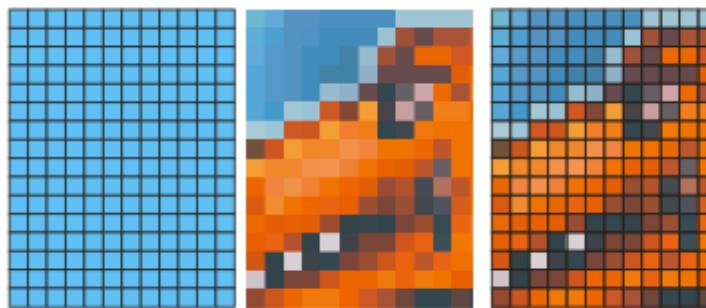
- Objekte bewegen sich scheinbar rückwärts
- Kleine Bewegungen können zu Pixel – Flackern führen

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [504]

© R. Dörner

## Filterung von Texturen

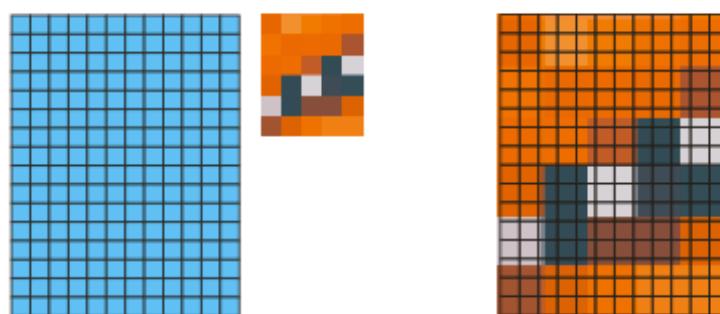
Fall 1: Bild und Textur sind gleich groß:



*Zu einem Bildpixel gehört ein Texturpixel*

## Filterung von Texturen

Fall 2: Bild ist größer als Textur:



*Ein Texturpixel wird auf mehrere Bildpixel vergrößert*

## Filterung von Texturen

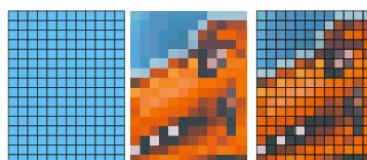
Fall 3: Bild ist kleiner als Textur:



Mehrere Texturpixel werden verkleinert,  
um in ein Bildpixel zu passen.

## Filterung von Texturen

Fall 1: Bild und Textur sind gleich groß:



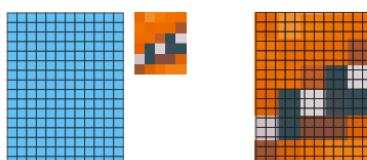
Zu einem Bildpixel gehört ein Texturpixel

Fall 3: Bild ist kleiner als Textur:



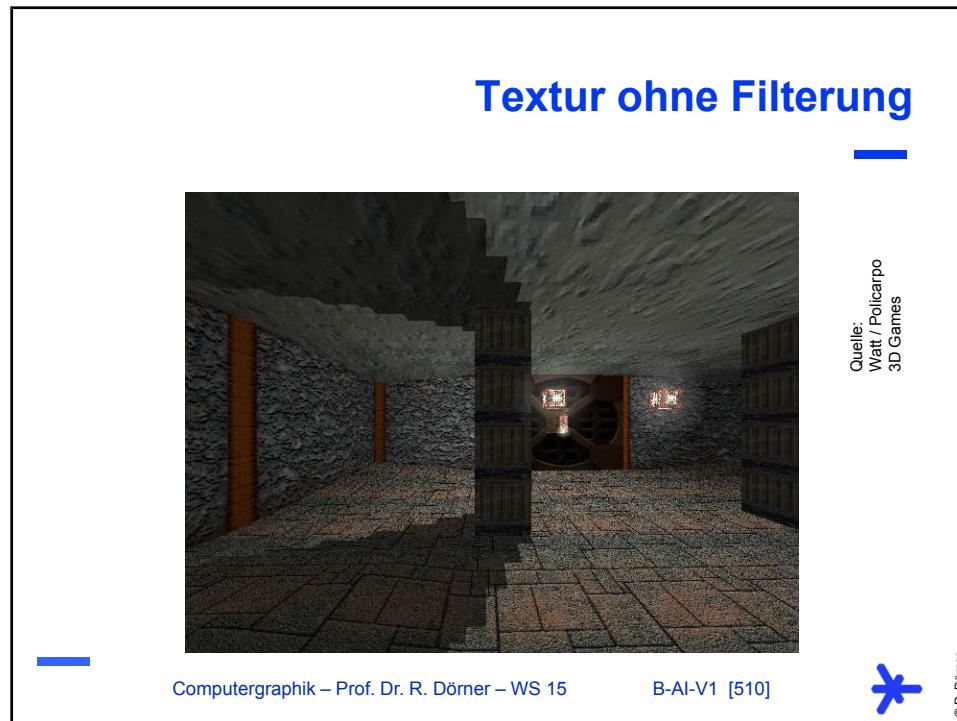
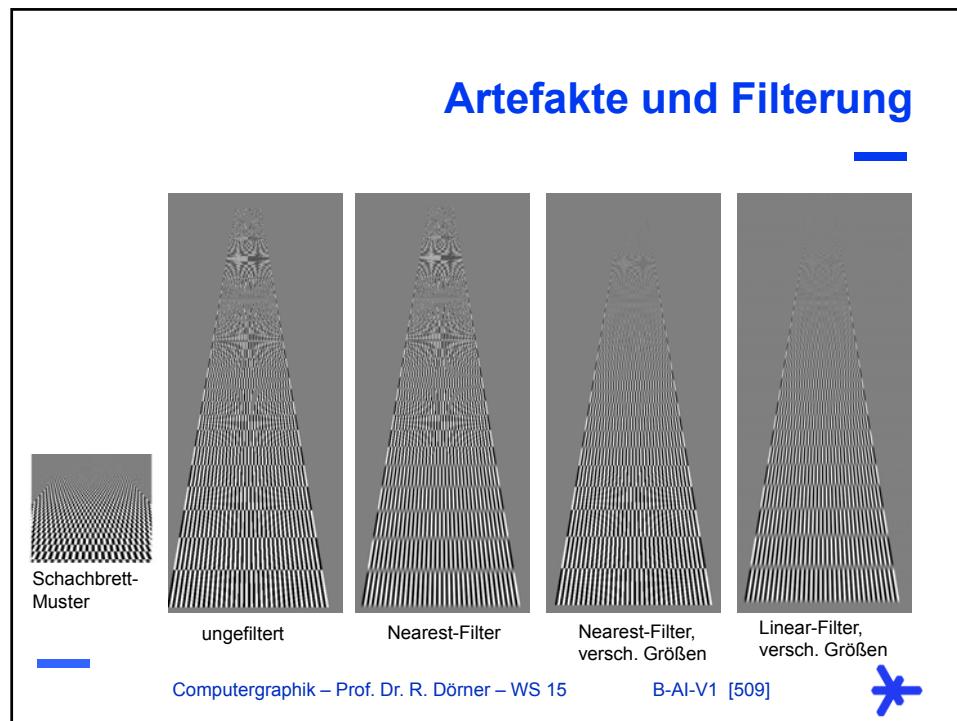
Mehrere Texturpixel werden verkleinert,  
um in ein Bildpixel zu passen.

Fall 2: Bild ist größer als Textur:



Ein Texturpixel wird auf mehrere Bildpixel vergrößert

Vermeidung von „pixelig“ wirkenden  
Texturen durch „glätten“ mit einem  
sogenannten „Filter“.



## Textur mit Filterung



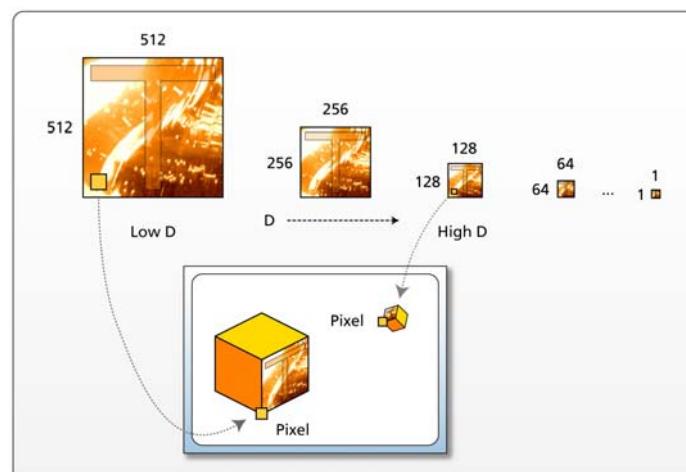
Quelle:  
Watt / Pollicapo  
3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [511]



© R. Dörner

## Textur-Filterung und MipMaps



Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [512]



© R. Dörner



## Speichern von MipMaps



Quelle:  
Watt / Polycarpo  
3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [513]



© R. Dörner

## Teil J: Fragment Operationen



## Fragment Operationen

- J.1 Vom Fragment zum Pixel
- J.2 Fragment Shader
- J.3 Verdeckungsrechnung



## Fragment Processing

Rasterisierung erzeugt Fragmente

Fragmente sind Flächen von der Größe eines Pixels, haben aber nicht nur Farbwert, sondern auch Tiefenwert (Entfernung von Kamera). Ein Fragment wird als Pixel im Bild eingetragen („Pixelkandidat“), wenn es nicht von anderen Fragmenten verdeckt wird.

Fragment Shader bestimmt Farbe und manipuliert ggf. Wert der Tiefenwert

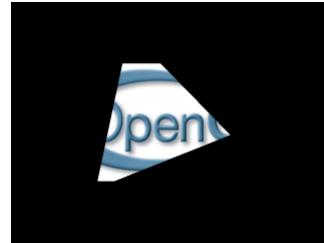
Per Fragment Ops (z.B. Verdeckungsrechnung)

Framebuffer - Operationen



## Per Fragment Ops

- Typische Operationen sind u.a.:
  - Entscheiden, welches Fragment andere Fragmente verdeckt (Depth Test, Verwendung des z-Buffers)
  - Berechnen, ob ein Fragment durch ein anderes Fragment durchschimmert (Alpha-Test)
  - Testen, ob Bildbereich durch anderes Fenster verdeckt wird (Pixel Ownership Test)
  - Löschen aller Fragmente, die in einer bestimmten Maske liegen (Scissor Test)
- Resultat: endgültige Farbe für einen Pixel



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [517]



© R. Dörner

## Framebuffer Operationen

- Die endgültigen Farben für die Pixel werden im Framebuffer eingetragen
- Auf dem ganzen Framebuffer können noch Operationen durchgeführt, z.B. Initialisieren des gesamten Buffers mit der Hintergrundfarbe (Befehl:  
`glClearColor`,  
`glClearBuffer`)
- Für Multipass-Rendering werden Rendertargets verwendet



Quelle: nVIDIA

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [518]



© R. Dörner



## Fragment Operationen

J.1 Vom Fragment zum Pixel

J.2 **Fragment Shader**

J.3 Verdeckungsrechnung



## Fragment Shader

- Wird ausgeführt für jedes Fragment. Dabei keine Daten von anderen Fragmenten zugreifen.
- Minimale Aufgabe:
  - Bestimmung, ob Fragment berücksichtigt werden soll (in OpenGL `discard` verwenden oder nicht)
  - Berechnung der Farbe des Fragment (in OpenGL schreiben der eingebauten Variable `vec4 gl_FragColor`)
  - Berechnung des z-Wertes für die Verdeckungsrechnung (in OpenGL schreiben der eingebauten Variable `float gl_FragDepth`), in WebGL ist das nicht nötig, wenn auf CPU-Seite die Verdeckungsrechnung (`GL_DEPTH_TEST`) aktiviert wurde
- Weitere Aufgaben: Anwendung Texturen, Beleuchtungsverfahren im Bild, Shading, Nebel, Bildeffekte (z.B. Blur)



Quelle: cgw.com



## GLSL Beispiel

```
varying vec4 fColor;  
varying float f;
```

Übergabe vom  
Vertex-  
Processing

```
void main(){
```

```
    glFragColor = fColor;  
}
```

Ergebnis des Fragment  
Shaders wird in  
vordefinierte Variable  
geschrieben

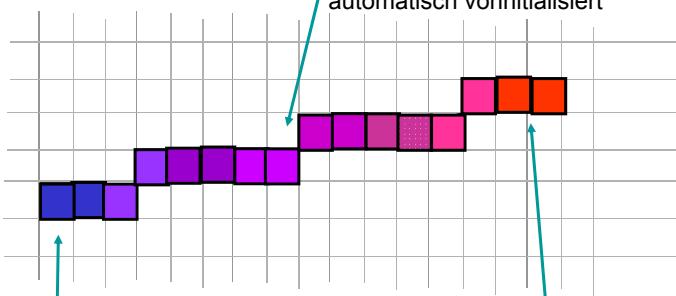
keine weiteren  
Modifikationen,  
aber Gouraud  
Shading der  
Farbe durch  
automatische  
Interpolation in  
der varying  
Variablen



## Interpolation bei Rasterisierung

```
varying float f;
```

Im Fragment Shader:  
Für das mittlere Fragment wird  $f$  auf 0.5  
automatisch vorinitialisiert

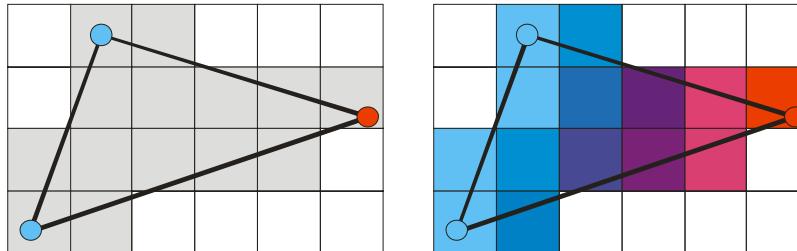


Im Vertex Shader:  
Dieser Vertex setzt  $f = 0.0$ ;

Im Vertex Shader:  
Dieser Vertex setzt  $f = 1.0$ ;



## Durchführung des Shadings



Basierend auf den Farben der Eckpunkte wird den Fragmenten durch ein Shading-Verfahren (z.B. Gouraud-Shading) eine Farbe zugeordnet.

## Veränderung der Fragment-Farbe

- Die Farbe, die aus den Eckpunktdata der 3D Szene für ein Fragment bestimmt wird, kann noch verändert werden
- Anwendung von Nebel ändert die Farbe
- Anwendung von Texturen ändert die Farbe
  - Texturkoordinaten des Fragments bestimmen
  - Schauen, welche Farbe in der Textur vorliegt
  - Diese Farbe mit der eigenen Farbe zu neuer Farbe mischen

```
uniform sampler2D src;
varying vec2 fTexCoord;

void main(){
    glFragColor =
        texture2D(src,
                  fTexCoord
        );
}
```

## Texturen und Shader in WebGL

```
in HTML-Seite: 
```

```
var img = document.getElementById("texSrc");
var t = gl.createTexture(); gl.bindTexture( gl.TEXTURE_2D, t );
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, img );
gl.generateMipmap( gl.TEXTURE_2D );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
                  gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );
gl.uniform1i(gl.getUniformLocation(program, "src"), 0);
```

für jeden Eckpunkt noch Texturkoordinaten bestimmen und beim Zeichnen als Attribute-Variable an Vertex-Shader übergeben (ähnlich wie bei Positionsdaten)

```
im Vertex-Shader: Texturkoordinaten an Fragment-Shader einfach als varying Variable übergeben
im Fragment-Shader: Textur auswerten und gl_FragColor entsprechend setzen
```



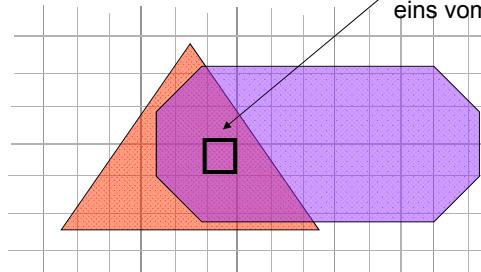
## Fragment Operationen

- J.1 Vom Fragment zum Pixel
- J.2 Fragment Shader
- J.3 Verdeckungsrechnung**



## Verdeckungsrechnung

In diesem Pixel liegen zwei Fragmente:  
eins vom Dreieck, eins vom Sechseck



Beobachtung: Pro Pixel  
im Bild kann es mehrere  
Fragmente geben

Objekte verdecken sich: bei undurchsichtigen Objekten sollte die Farbe des Fragments gewählt werden, das von dem Objekt stammt, das am nächsten zum Betrachter ist.



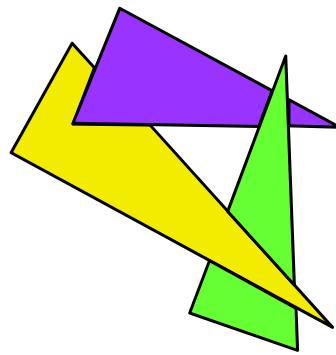
## Painter-Algorithmus

- Die Objekte werden in Entfernung zur Kamera sortiert
- Das am weitesten hintere Objekt wird zuerst gezeichnet
- Der Reihe nach arbeitet man sich nach vorn durch: bisherige Farben werden überschrieben



## Painter-Algorithmus

- Verdeckungsrechnung ist ein Sortierproblem
- Allerdings gibt es nicht immer eine Lösung bei gegenseitiger Verdeckung
- Aufsplitten von Objekten notwendig, ggf. auf Ebene der Fragmente



Computergraphik – Prof. Dr. R. Dörner – WS 15

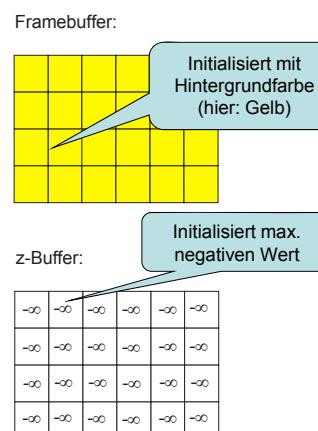
B-AI-V1 [529]



© R. Dörner

## z-Buffer Algorithmus

- Wir speichern zu jedem Fragment noch den z-Wert (in Clipping-Koordinaten)
- z-Wert ist Maß für die Entfernung zur Kamera
- Zusätzlicher Speicherbereich: z-Buffer
  - Größe des Bildes
  - Typischerweise 24 Bit Tiefe
  - Bsp. 800 x 600 x 24 Bit ergibt ca. 1,4 MB



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [530]

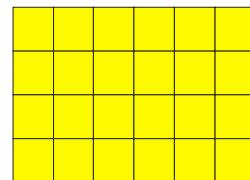


© R. Dörner



## z-Buffer Algorithmus

Framebuffer:



z-Buffer:

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Computergraphik – Prof. Dr. R. Dörner – WS 15

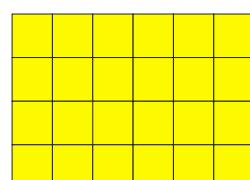
B-AI-V1 [531]



© R. Dörner

## z-Buffer Algorithmus

Framebuffer:



Wollen diese 4 Fragmente eintragen.  
Jedes Fragment hat eine Farbe, eine Position und einen z-Wert

-0.2	-0.4
-0.2	-0.5

z-Buffer:

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [532]

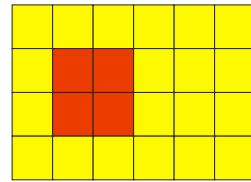


© R. Dörner



## z-Buffer Algorithmus

Framebuffer:



Der Hintergrund wird immer  
überschrieben.

-0.2	-0.4
-0.2	-0.5

z-Buffer:

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

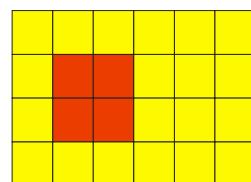
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [533]

© R. Dörner

## z-Buffer Algorithmus

Framebuffer:



Wollen nun diese 3 Fragmente  
eintragen.

-0.3	
-0.3	-0.3

z-Wert des Fragments ist größer.  
Also wird es eingetragen.

Je größer der z-Wert desto näher  
an der Kamera desto eher ist es  
sichtbar, weil es andere verdeckt.

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Computergraphik – Prof. Dr. R. Dörner – WS 15

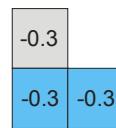
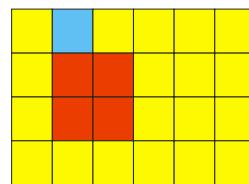
B-AI-V1 [534]

© R. Dörner



## z-Buffer Algorithmus

Framebuffer:



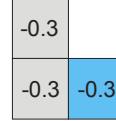
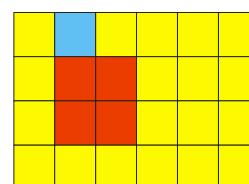
z-Wert des Fragments ist kleiner.  
Also wird es nicht beachtet.

z-Buffer:

$-\infty$	-0.3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

## z-Buffer Algorithmus

Framebuffer:



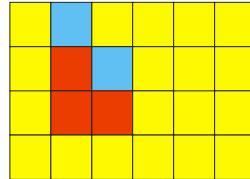
z-Wert des Fragments ist größer.  
Also wird es eingetragen.

z-Buffer:

$-\infty$	-0.3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

## z-Buffer Algorithmus

Framebuffer:



z-Buffer:

$-\infty$	-0.3	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.3	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	$-\infty$	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [537]



© R. Dörner

## z-Buffer Algorithmus

Initialisiere Framebuffer  $fb[x,y]$  mit der Hintergrundfarbe

Initialisiere z-Buffer  $zb[x,y]$  mit  $-\text{Unendlich}$

Für alle Polygone in der Szene:

    Für alle Fragmente eines Polygons:

        Bestimme Farbe  $F$  des Fragments

        Bestimme Position  $x, y$  des Fragments

        Bestimme Wert  $z$  des Fragments

        if ( $z > zb[x,y]$ )

$zb[x,y]=z$

$fb[x,y]=F$

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [538]



© R. Dörner



## z-Buffer Algorithmus in WebGL

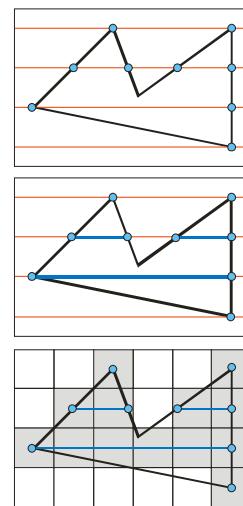
- z-Buffer wird automatisch angelegt, `gl_FragDepth` wird automatisch mit Wert belegt
- Spezielle GL-Befehle für den z-Buffer
- Near-Plane und Far-Plane möglichst dicht aneinander, damit keine Genauigkeitsprobleme im z-Buffer auftreten

```
glEnable(GL_DEPTH_TEST);  
  
glClear(GL_DEPTH_BUFFER_BIT);
```



## Scan-Line Algorithmen

- Scan-Line Algorithmen kann für Rasterisierung als auch für Shading und Verdeckungsrechnung verwendet werden
- Rechenzeitzersparnis durch Behandeln von Spans als Ganzes (z.B. sind Anfangs- und Endpunkt eines Spans verdeckt, dann brauchen alle dazwischen liegenden Fragmente nicht mehr untersucht werden)
- Untere Schranke für die Komplexität der Verdeckungsrechnung:  $O(n \log n)$



## Teil K: GDV Anwendungen

### GDV Anwendungen

- K.1 Grafik-APIs
- K.2 Autorenwerkzeuge
- K.3 Anwendungsbeispiele



## GDV Anwendungen

- K.1 Grafik-APIs
- K.2 Autorenwerkzeuge
- K.3 Anwendungsbeispiele



## OpenGL und Direct3D

- Schnittstellen zu low-level Funktionalität von Graphikhardware
- OpenGL unabhängig vom Betriebssystem, verschiedene Bindings für Programmiersprachen (z.B. jogl für Java)
- Direct3D nur für Microsoft Produkte, Teil von DirectX

```
// Initialize the world matrix
g_World = XMMatrixIdentity();

// Initialize the view matrix
XMVECTOR Eye
= XMVectorSet( 0.0f, 1.0f, -5.0f, 0.0f );
XMVECTOR At
= XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
XMVECTOR Up
= XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
g_View = XMMatrixLookAtLH( Eye, At, Up );

// Initialize the projection matrix
g_Projection
= XMMatrixPerspectiveFovLH( XM_PIDIV2, width
/ (FLOAT)height, 0.01f, 100.0f );
```



## VRML und X3D

- VRML hat höhere Abstraktionsebene als OpenGL oder DirectX:  
Szenengraph
- Nachfolger von VRML: X3D  
([www.web3d.org](http://www.web3d.org))
  - XML-basierte Notation: statt „Transform{ }“ nun „<transform>“
  - Mehr Node-Types
  - Unterschiedliche Ausbaustufen
- Skriptsprache: ECMA-Script  
(über Script-Node einbinden)



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [545]



© R. Dörner

## Szenengraph für HTML

- Szenengraph-APIs um mit HTML ohne Plug-ins einen Szenengraph zu nutzen
- Beispiel: JavaScript Framework SceneGraph.js
- Ergänzung der Funktionalität um Grundprimitive, Import von 3D Daten etc.

### CGSSceneGraph Class

Extends Object  
Defined in: [src/classes.szenengraph.js#126](#)  
Module: Szenen

Represent the scene graph it self.

#### Constructor

`CGSSceneGraph ( canvas , context )`

Defined in [src/classes.szenengraph.js#126](#)

#### Parameters:

- `canvas`: `HTMLElement`  
a handler to the canvas HTML element
- `context`: `CanvasRenderingContext2D`  
context to render on

[Index](#) [Methods](#) [Properties](#)

#### Item Index

##### Methods

- `addNode`
- `deselectAll`
- `deselectNode`
- `invalidateTheme`
- `pickNode`
- `pickNodes`
- `removeNode`
- `render`
- `selectNode`
- `setCanvasDimension`

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [546]



## Open SG

- Weiteres Beispiel einer Szenengraph-API
- Fokus auf Multi-threading, Cluster-Fähigkeit und Rendering-performanz
- Open Source unterstützt von DaimlerChrysler, Audi, VW, BMW, Siemens, HP u.a.

```
Quaternion q;
Matrix m;
m.setIdentity();
q = Quaternion( Vec3f(0,1,0), PI /60 );
m.setRotation( q );
NodePtr grandpa = Node::create();
NodePtr aunt = Node::create();
NodePtr mother = Node::create();
NodePtr me = Node::create();
// now we create the hierarchy
beginEditCP(grandpa);
    grandpa->addChild(aunt);
    grandpa->addChild(mother);
endEditCP(grandpa);
beginEditCP(mother);
    mother->addChild(me);
endEditCP(mother);
```

Quelle:  
[www.opensg.org](http://www.opensg.org)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [547]



© R. Dörner

## jMonkey

- Szenengraph-API für Java, weitere Funktionen (3D Games)
- [jmonkeyengine.org](http://jmonkeyengine.org)
- Nodes des Szenengraphs sind als Java-Klassen realisiert
- Open Source

```
// Create Box
Box box
= new Box("my box", new Vector3f(0,
0, 0), 2, 2, 2);
box.setModelBound(
    new BoundingSphere());
box.updateModelBound();
box.updateRenderState();
s.getRootNode().attachChild(box);
s.getRootNode().updateRenderState();
GameStateManager.getInstance().
attachChild(state);
```

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [548]



## GDV Anwendungen

K.1 Grafik-APIs

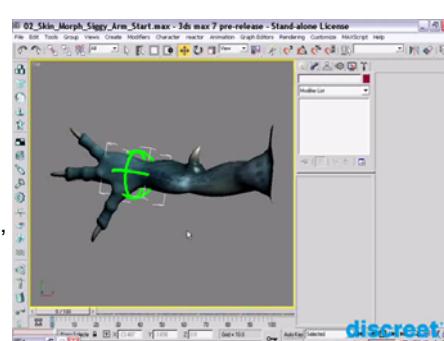
K.2 Autorenwerkzeuge

K.3 Anwendungsbeispiele



## Grafik Werkzeuge und Skriptsprachen

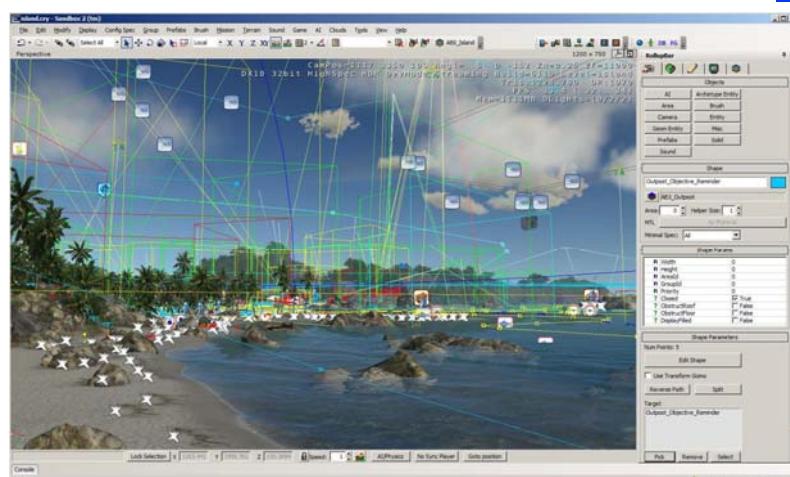
- Tools für Bilderstellung und Bildbearbeitung
  - CorelDraw, PhotoShop, ...
- Tools für Modellierung und Animation
  - 3DSMax (MaxScript), Maya (MELScript), Blender (Python), Flash (ActionScript), ...
- Tools für Shader-Programmierung
  - Povray, RenderMonkey, ...



Quelle: discreet



## Autorenwerkzeuge von Game Engines



Quelle: crytek.com

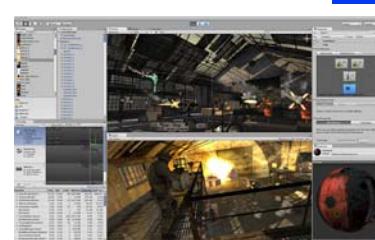
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [551]



## Autorenwerkzeuge von Game Engines

- Einige Game Engines (z.B. Unity3D, CryEngine) verfügen über WYSIWYG-Editor
- Aufgabe ist nicht primär die Erstellung von Assets, sondern deren Verknüpfung und Erzeugung der Simulation der virtuellen Welt



Quelle: unity3d.com

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [552]



## GDV Anwendungen

- K.1 Grafik-APIs
- K.2 Autorenwerkzeuge
- K.3 Anwendungsbeispiele**

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [553]



© R. Dörner

## Anwendungen

- Kennen Grundlagen, wie man mit Computer Bilder erstellen kann
- Wozu wird dies genutzt?
- Was kann man damit machen?  
Anwendungen?
- Was gibt es noch zu lernen?



Luc\_Bianco - Terragen V0.8.44 - 07/2002

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [554]



© R. Dörner



## Anwendungen

- Wichtigste Anwendung:
  - Verbesserung der Mensch-Maschine Schnittstelle
  - User Interfaces und Usability
- Ebenfalls wichtig:
  - Visualisierung
  - Daten visuell begreifbar und analysierbar machen
- Daneben:
  - Computer Animation
  - Special Effects
  - Virtual Reality
  - Computer Spiele (hohe wirtschaftliche Bedeutung)
  - ...

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [555]



© R. Dörner

## Visualisierung



Quelle:  
Deutscher Wetterdienst

Computergraphik – Prof. Dr. R. Dörner – WS 15

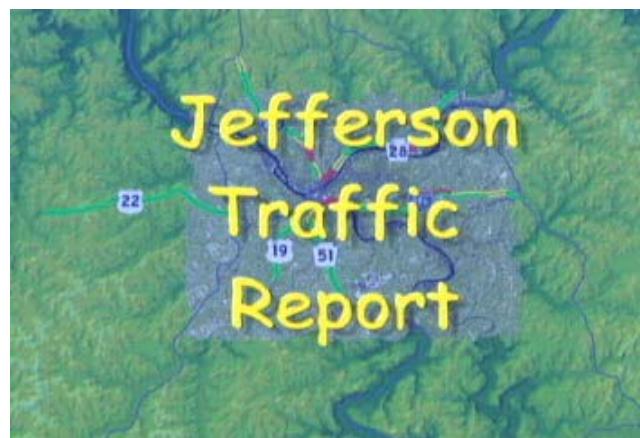
B-AI-V1 [556]



© R. Dörner



## Visualisierung



Quelle:  
Andre Guezic:  
3D Traffic Visualization  
in Real Time  
[www.trianglesoftware.com](http://www.trianglesoftware.com)

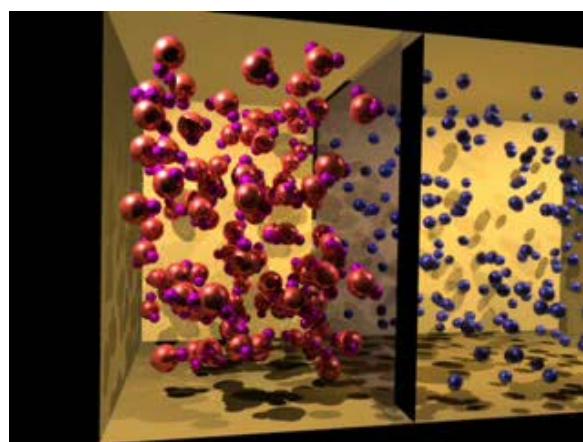
Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [557]



© R. Dörner

## Visualisierung



Quelle:  
Brian Mirtich  
Timewarp Rigid Body Simulation  
Siggraph 2000

Computergraphik – Prof. Dr. R. Dörner – WS 15

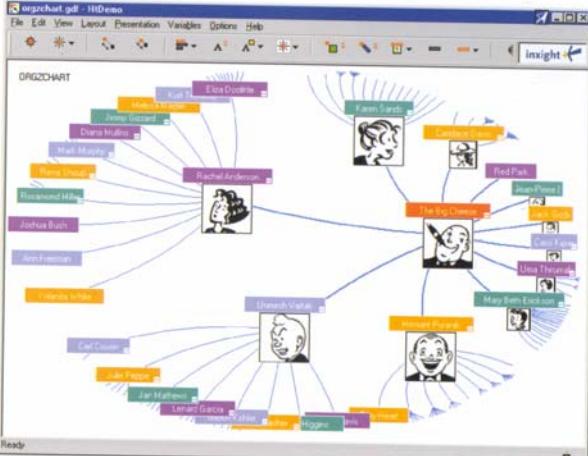
B-AI-V1 [558]



© R. Dörner



## Visualisierung

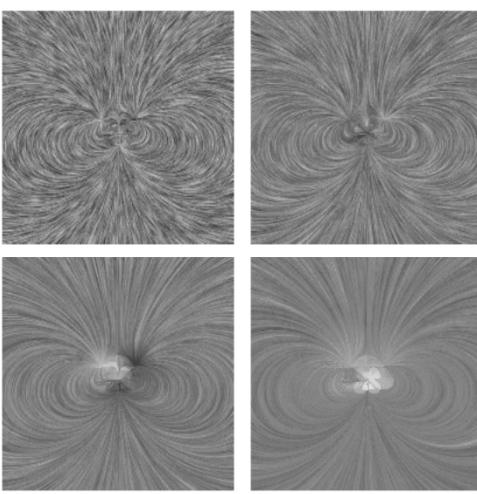


Quelle:  
Robert Spence,  
Information Visualization

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [559]

© R. Dörner

## Visualisierung

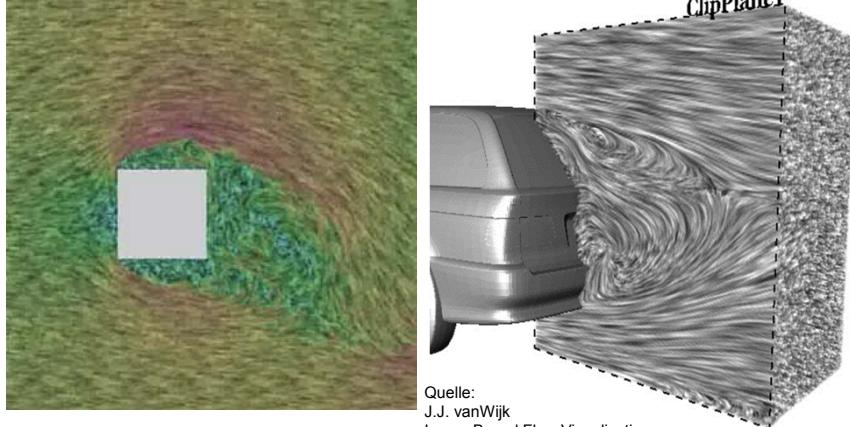


Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [560]

© R. Dörner



## Visualisierung



Quelle:  
J.J. vanWijk  
Image Based Flow Visualization  
Siggraph 2002

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [561]

© R. Dörner

## Visualisierung



Quelle:  
Fraunhofer IGD

Computergraphik – Prof. Dr. R. Dörner – WS 15      B-AI-V1 [562]

© R. Dörner



## Visualisierung



Quelle:  
GCF

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [563]



© R. Dörner

## Visualisierung



Quelle:  
Fraunhofer IGD

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [564]



© R. Dörner



## Visualisierung

The slide displays two examples of medical visualization. On the left, a screenshot of a virtual colonoscopy interface shows a 3D rendering of a colon with a camera path and an artificial horizon. To the right are three grayscale axial slices of a CT scan, with a yellow arrow pointing to the 'current position' on one of the slices. Below these images is the text: 'Quelle: Dirk Bartz, Universität Tübingen'. On the right side of the slide is a 3D reconstruction of a human skull in blue wireframe, labeled 'Quelle: Visible Human Project, National Library of Medicine, Maryland, USA'. At the bottom left is the text 'Computergraphik – Prof. Dr. R. Dörner – WS 15' and at the bottom right is 'B-AI-V1 [565]'. A small blue asterisk logo is in the bottom right corner.

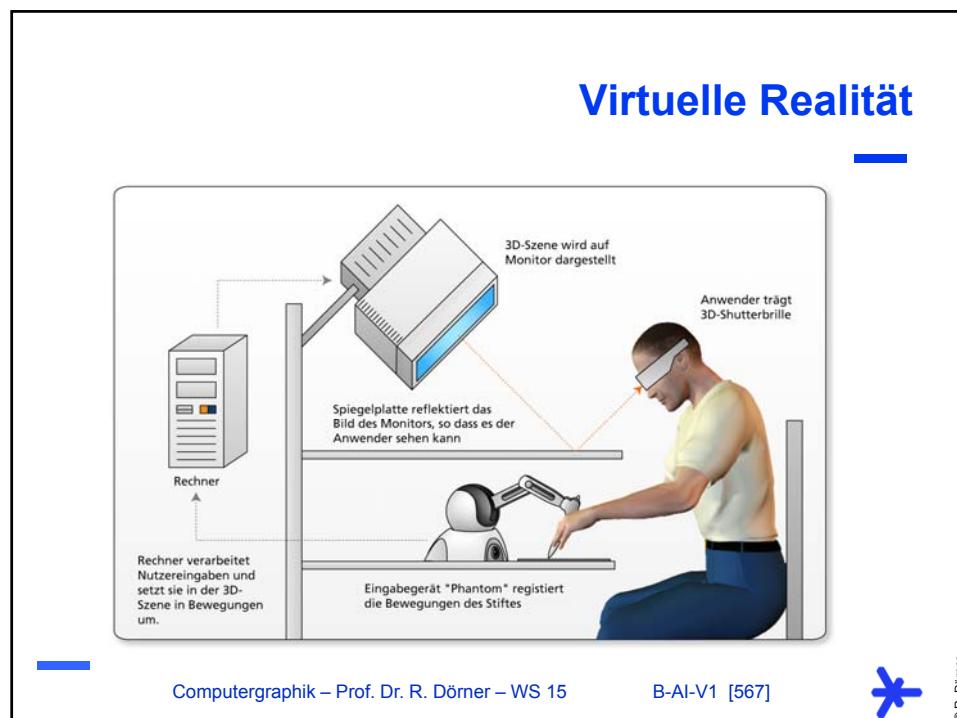
## Visualisierung und GIS

The slide shows a Geographic Information System (GIS) application interface. The main window displays a map of Lake Nipigon with various sampling stations marked by red dots. A circular selection tool is overlaid on the map. To the right of the map is a bar chart titled 'Results of Sampling' comparing Nitrate and Fluoride levels across three samples. Below the map are several windows: 'water Resources' showing a legend; 'water Resources' showing a detailed view of the lake area; 'Lake Nipigon' showing a photograph of the lake; and a table titled 'SAMPLE LATITUDE LONGITUDE NITRATE FLUORIDE' with the following data:

SAMPLE	LATITUDE	LONGITUDE	NITRATE	FLUORIDE
1	49.9191	100.0442	6.2	5.3
42	49.9191	100.0442	6.3	5.2
43	49.9191	100.0442	6.3	5.2
1	49.9117	102.0096	6.5	3.2
2	49.9036	102.0212	6.1	3.2
3	49.9036	102.0212	6.3	3.3
4	49.9141	102.0254	6.1	3.5

Below the table is a 'Viewers distance' window showing a photograph of a yellow flower. The text 'Quelle: www.gis.com' is located at the bottom right. A small blue asterisk logo is in the bottom right corner.





## Tiled Wall / Stereodarstellung



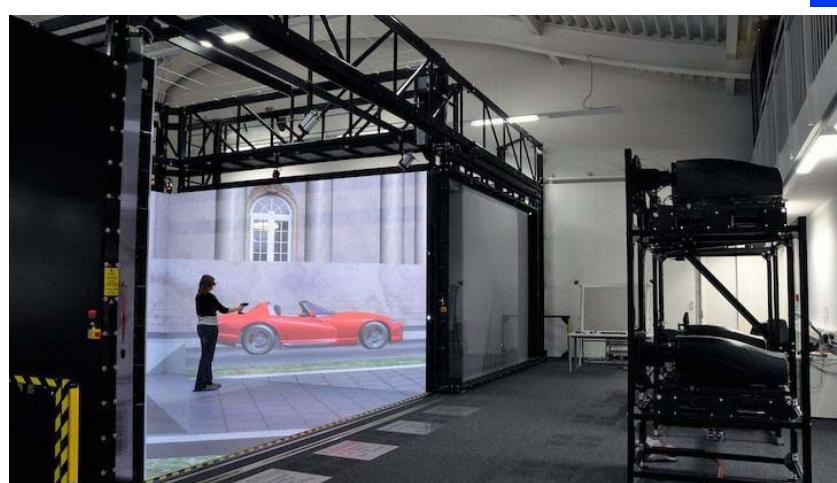
Abb. 5.7 Tiled Wall aus Monitoren am Beispiel des Stony Brook's Reality Deck

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [569]



## CAVE



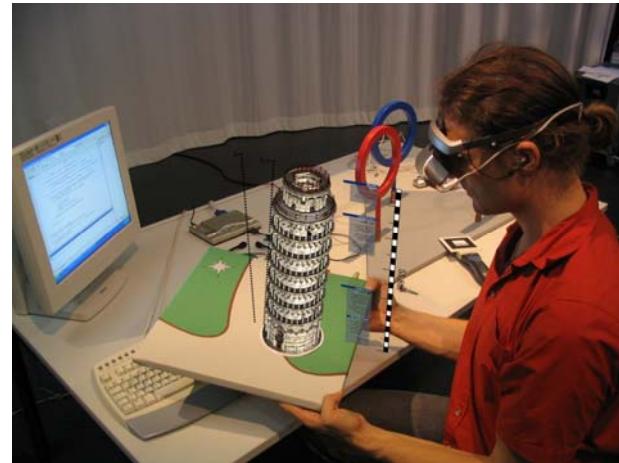
Quelle: RWTH Aachen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [570]



## Mixed Reality



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [571]

© R. Dörner

## Spatial AR



Abb. 9.13 Benutzung von SAR im Interieur Design-Prozess: a) Setup mit zwei Projektoren und realem Modell; b)-d) Projektion verschiedener Passat Varianten auf das Modell. Nach Menk (2012)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [572]

★



## Mobile AR



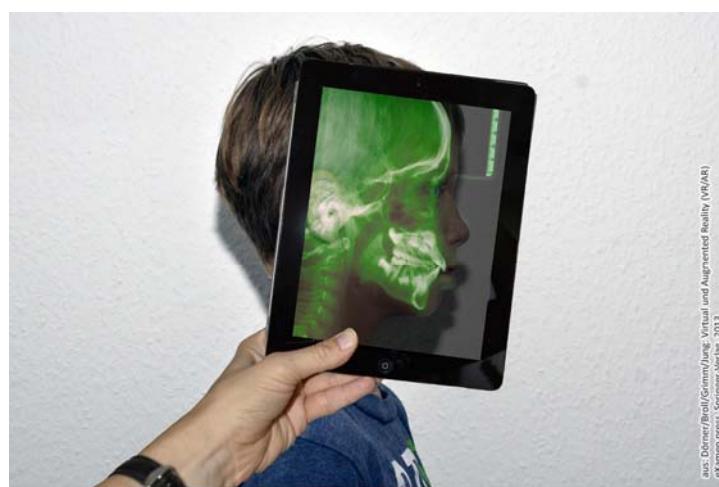
**Abb. 9.15** Einsatz des entwickelten mobilen AR-Systems zur Untersuchung eines Hallenneubaus  
(Quelle: Volkswagen AG)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [573]



## Magic Lens



**Abb. 8.6** Beispiel einer Magic Lens

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [574]



## Mobile AR



Abb. 9.17 AR-System auf der Seite des entfernten Teilnehmers mit eingeblendeten Informationen (Quelle: Fraunhofer FKIE)

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [575]



## Mixed Reality

### Face Tracking Live Demo

L. Vacchetti, V. Lepetit, P. Fua



Computergraphik – Prof. Dr. R. Dörner – WS 15

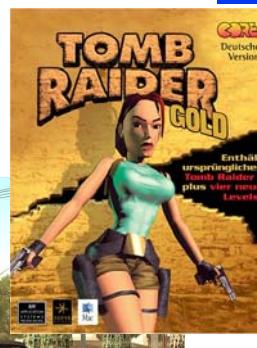
B-AI-V1 [576]



© R. Dörner



## Computer Games



Quelle:  
Vivendi, Rockstar Games, EIDOS

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [577]



© R. Dörner

## Learning & Training



Quelle:  
SPG Media Group



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [578]



© R. Dörner



## Visualisierungslabor

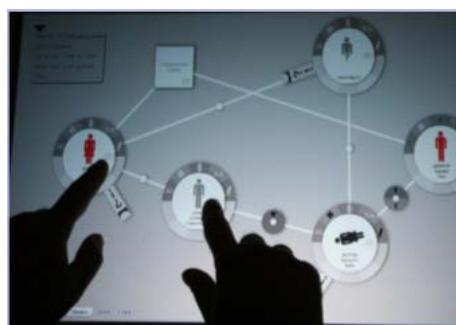


Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [579]



## Multi-Touch Displays



Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [580]



## Was gibt es noch zu lernen?

- Interaktive Computergraphik
  - Wie auf User-Interaktion reagieren? Bsp. Worauf hat User geklickt?
  - Navigation durch 3D
- Echtzeit Computergraphik
  - Welche Tricks gibt es, damit 60 Bilder pro Sekunde gerendert werden?
  - Computer Animation und Optimierungen



## Was gibt es noch zu lernen?

- Usability und Visualisierung
  - Wahrnehmung von Bildern und Raum
  - Visualisierungstechniken und Usability Engineering
  - Visualisierungssysteme
- Virtuelle Realität
  - Ein- und Ausgabegeräte, Tracking
  - Terrain und GIS
  - Kombination mit Bildverarbeitung



## Was gibt es noch zu lernen?

- Computer Games
  - Game Design
  - Game Physics
  - Game AI
- Spezielle Tools und APIs
  - Animationssysteme
  - Modellierungswerkzeuge
  - Game Engines

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [583]



© R. Dörner

## Was gibt es noch zu lernen?

- Visuelle Qualität
  - Tricks mit Beleuchtung und Texturen
  - Non-Photorealismus
  - Shading Languages
- ... und vieles mehr

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [584]



© R. Dörner



## Computergraphik

- A. Einführung
- B. Szenengraphen und Koordinatensysteme
- C. Kameramodell
- D. Beleuchtungsmodell
- E. Szenenmodell
- F. Objektmodelle
- G. Rendering und OpenGL
- H. Vertex Operationen
- I. Culling, Clipping und Rasterisierung
- J. Fragment Operationen
- K. GDV Anwendungen

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [585]



© R. Dörner

# ENDE

Computergraphik – Prof. Dr. R. Dörner – WS 15

B-AI-V1 [586]



© R. Dörner

