

—
Computergraphik
WS 2020
—

Prof. Dr. R. Dörner, HS RheinMain

1

—
Teil A:
Einführung
—

2



Beispiel für Computergrafik

Augmented MPM for phase-change
and varied materials



Alexey Stomakhin Craig Schroeder Chenfanfu Jiang
Lawrence Chai Joseph Teran Andrew Selle

©Disney

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [3]



3

Beispiel für Computergraphik



Quelle:
Y. Maekawa, Osaka
Sangyo University
F.S. Lai, University of
Massachusetts Lowell

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [4]



© R. Dörner

4



Beispiel für Computergraphik

Quelle:
Sony



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [5]



© R. Dörner

5

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 20

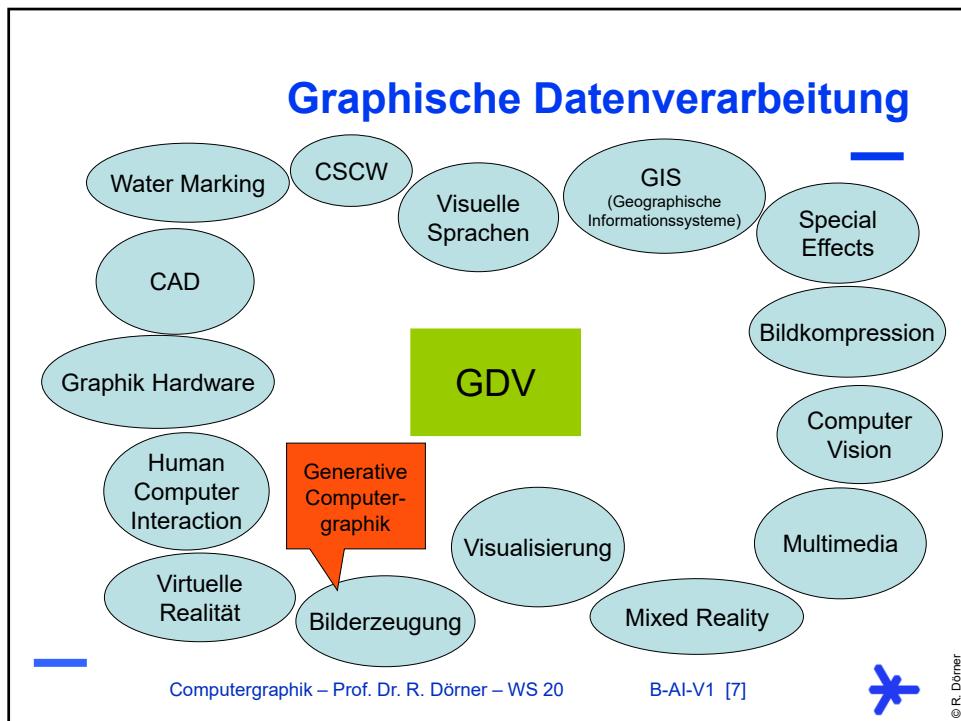
B-AI-V1 [6]



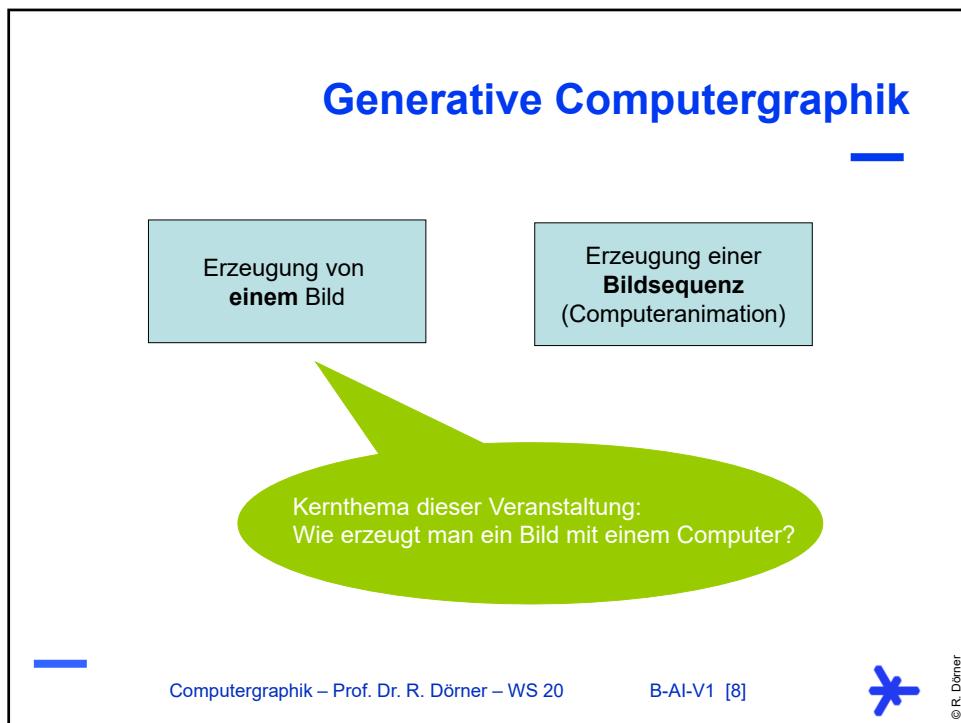
© R. Dörner

6





7



8



Computergenerierte Bilder



photorealistic

Quelle:
Bob Hoffman,
Digital Domain

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [9]



© R. Dörner

9

Computergenerierte Bilder



non - photorealistic

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [10]

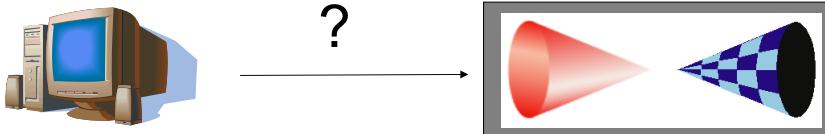


© R. Dörner

10



Kernfrage

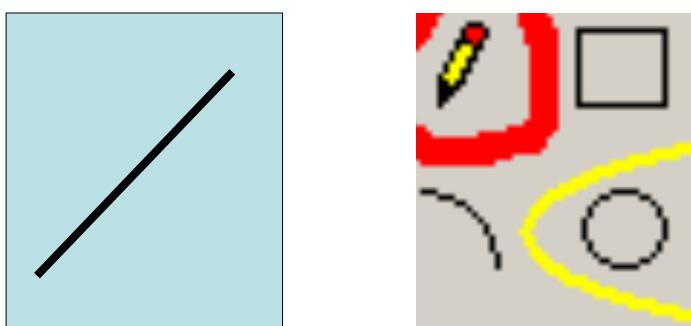


Wie Bild mit dem Computer erzeugen?
Was ist überhaupt ein Bild?
Wie wird ein Bild im Computer repräsentiert?

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [11]  © R. Dörner

11

Bildrepräsentation



Vektorgraphik
Speichern von:
Koordinaten, Primitive

Pixelbild
Speichern von:
Matrix von Farbwerten

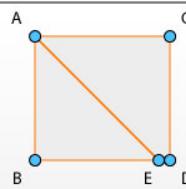
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [12]  © R. Dörner

12

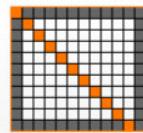


Bildrepräsentation

Vektorgrafik



Rastergrafik



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [13]



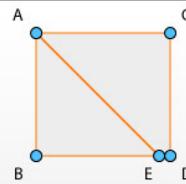
© R. Dörner

13

Bildrepräsentation

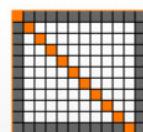
Vektorgrafik

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



Rastergrafik

Zu speichern ist der Wert von 24 x 24 Pixel



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [14]



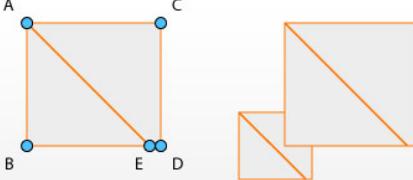
© R. Dörner

14



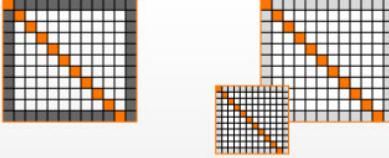
Bildrepräsentation

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 24×24 Pixel

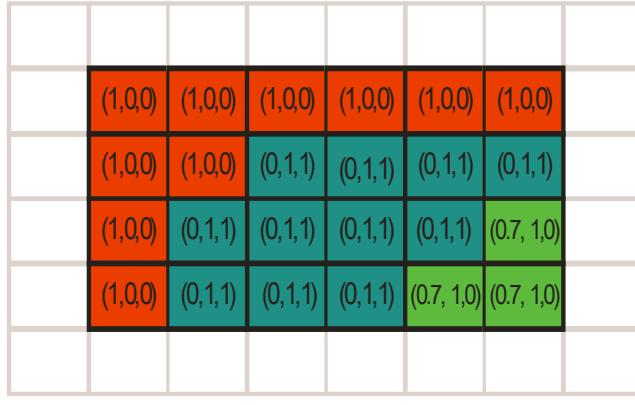


Bei Skalierung können Artefakte auftreten (z.B. Treppenstufen an Kanten)

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [15]  © R. Dörner

15

Bildrepräsentation



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [16]  © R. Dörner

16



Prozess der Bildgenerierung

Ausgangspunkt:

?

Prozess der Bildgenerierung

ausgeführt von: Computer

Ziel:

Matrix von
Farbwerten
(Bild)

Prozess der Bildgenerierung

Ausgangspunkt:

Spezifikation, was
auf dem Bild zu
sehen sein soll

ausgeführt von:
menschlicher Autor
(oder
Computerprogramm)

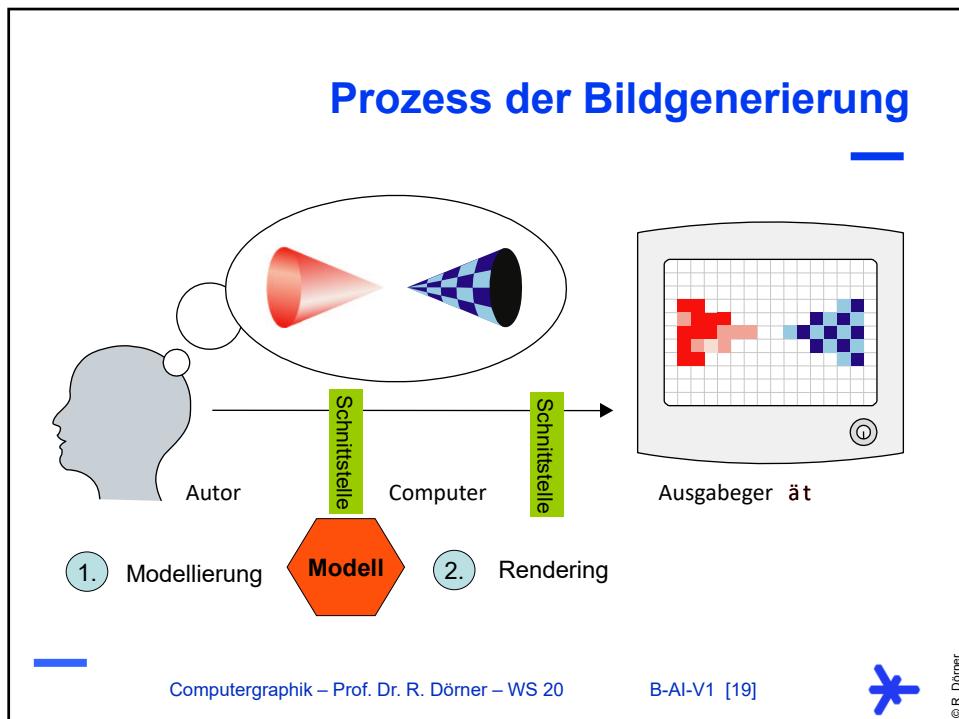
Prozess der Bildgenerierung

ausgeführt von: Computer

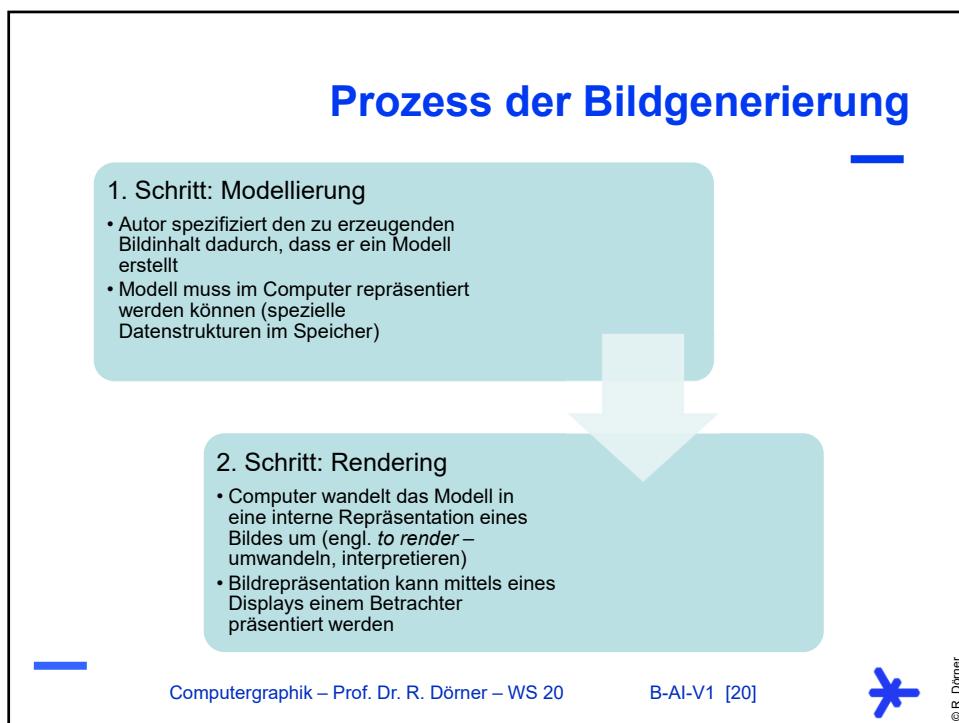
Ziel:

Matrix von
Farbwerten
(Bild)



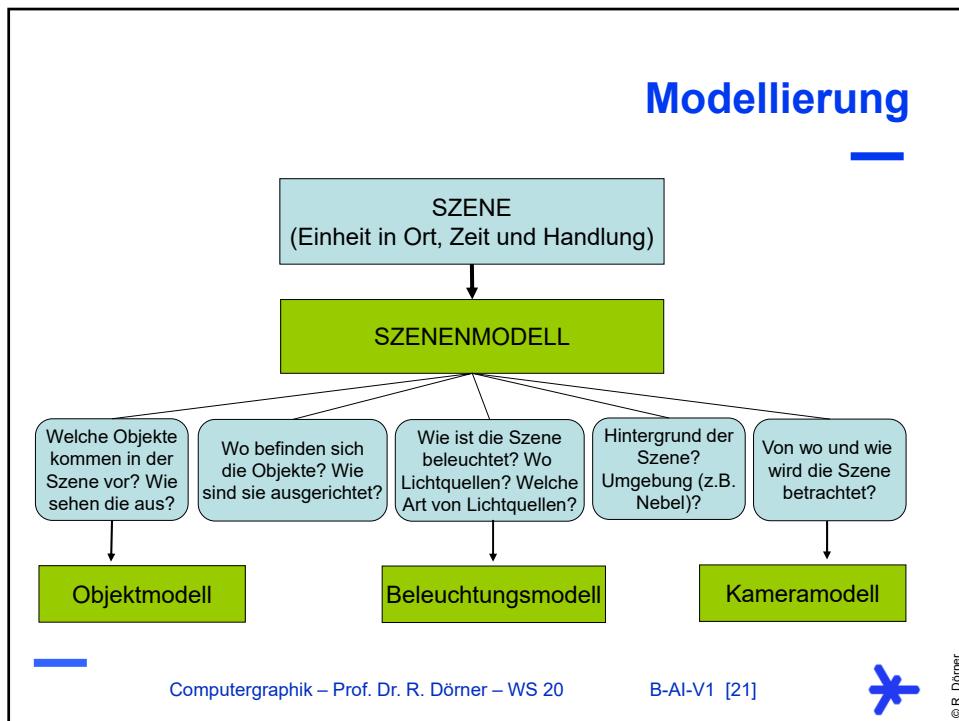


19

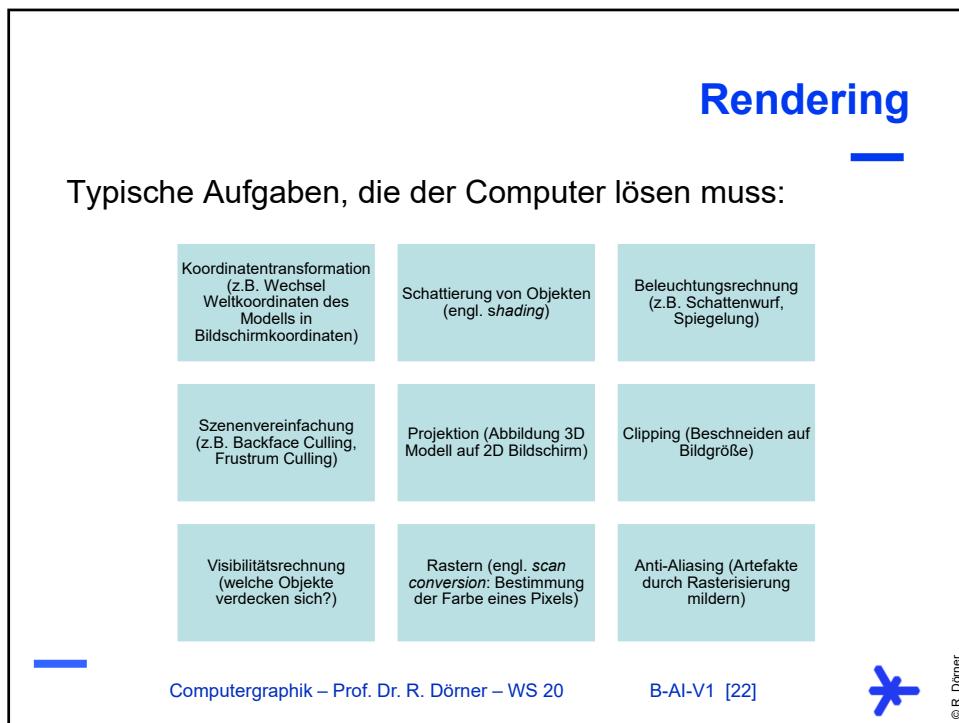


20





21



22



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**



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [23]



© R. Dörner

23

Beispiel: VRML

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [24]



© R. Dörner

24



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 20

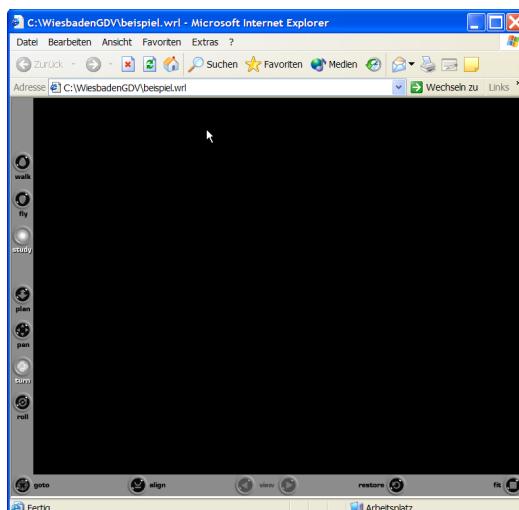
B-AI-V1 [25]



© R. Dörner

25

Beispiel: VRML



Darstellung des Szenenmodells in einem VRML Browser
(hier: Cortona als IE-Explorer Plugin, siehe www.cortona.com)

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [26]



© R. Dörner

26



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 20 B-AI-V1 [27]

© R. Dörner

27

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 20 B-AI-V1 [28]

© R. Dörner

28



Beispiel: VRML

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

Graphische Darstellung als **Szenengraph**

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [29]

© R. Dörner

29

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [30]

© R. Dörner

30



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
          }
        }
      ]
    }
  ]
}
```

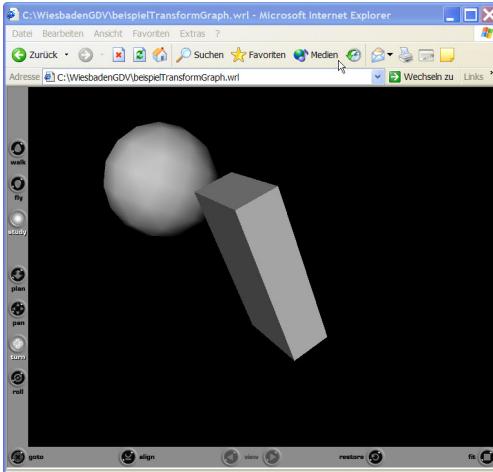
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [31]

Aufgabe:
Zeichnen Sie den Szenengraphen!

© R. Dörner

31

Beispiel: VRML



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [32]

© R. Dörner

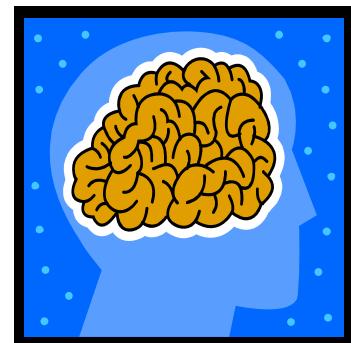
32



Beispiel: VRML

Prozess der Bildgenerierung: Teil 1

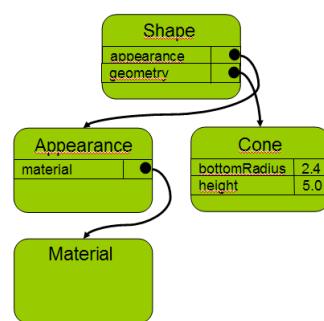
Im Kopf des Autors entsteht die Idee und die Vorstellung, was auf dem Bild zu sehen sein soll



Beispiel: VRML

Prozess der Bildgenerierung: Teil 2

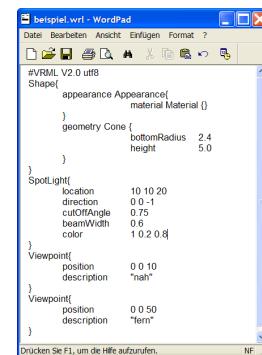
Der Autor beschreibt seine Vorstellung explizit mit Hilfe von Modellen. Ziel ist es, eine Szene zu modellieren. Dazu bietet VRML das Modell des Szenen-graphen an und auch andere Modelle, wie z.B. das Modell eines Lichtkegels (Spotlight) als Beleuchtungsmodell.



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).

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 20

B-AI-V1 [35]



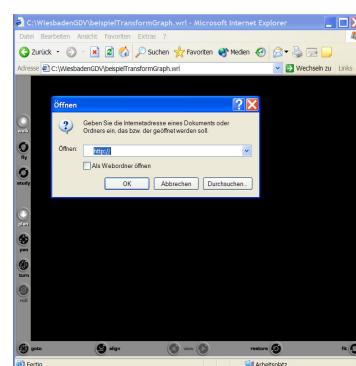
© R. Dörner

35

Prozess der Bildgenerierung: Teil 4

In dem VRML-Browser (d.h. GDV Software-System) 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.

Beispiel: VRML



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [36]



© R. Dörner

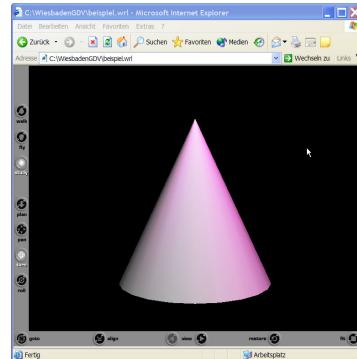
36



Beispiel: VRML

Prozess der Bildgenerierung: Teil 5

In dem VRML-Browser (d.h. GDV Software-System) 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 20 B-AI-V1 [37]  © R. Dörner

37

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 20 B-AI-V1 [38]  © R. Dörner

38



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

- 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



Organisatorisches

- Praktikum und Zeiten
- Pünktlichkeit und Ruhe
- Leistungsnachweis

Aktuelle Infos, Skript, etc. unter
[/staff/doerner/public_r/gdv2020](http://staff/doerner/public_r/gdv2020)



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 20

B-AI-V1 [43]



43

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 Zeitverschwendungen
- 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 20

B-AI-V1 [44]

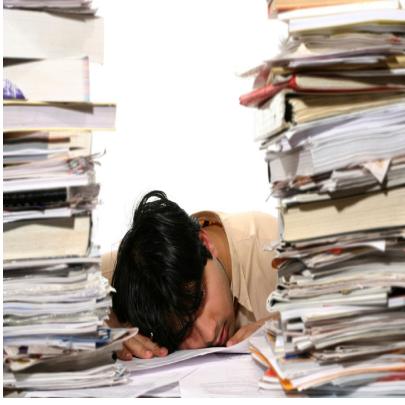


© R. Dörner

44



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 20 B-AI-V1 [45]



45

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 20 B-AI-V1 [46]



© R. Dörner

46



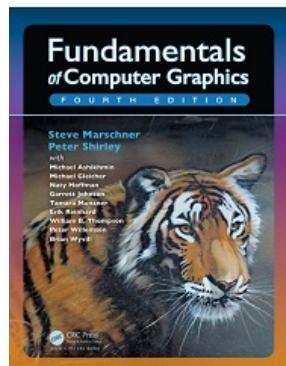
Leistungsnachweis



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



Empfohlene Literatur



Peter Shirley, Steve
Marschner:
Fundamentals of Computer
Graphics, 4.te Auflage,
Adisson-Wesley, 2016

*Buch, das Grundkonzepte
inklusive mathematischer
Grundlagen und Grundlagen
von Shadern bespricht.
Empfohlen zur
Vorlesungsbegleitung.*



Empfohlene Literatur



A. Nischwitz et. al:
Computergrafik und
Bildverarbeitung, Bd. I
4.te Auflage, Vieweg, 2019

*Deutschsprachig. Illustriert
theoretische Grundlagen an
OpenGL. Ein weiterer Band
des Buches beschäftigt sich
mit Bildverarbeitung*

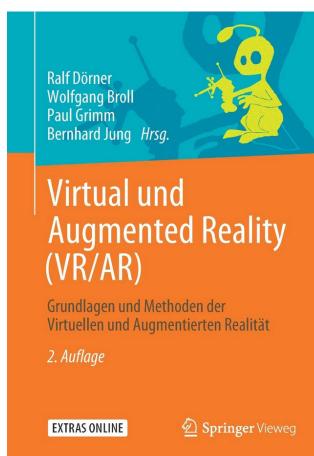
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [49]



49

Empfohlene Literatur



Ralf Dörner, Wolfgang Broll,
Paul Grimm, Bernhard Jung
(Hrsg.): Virtual und Augmented
Reality (VR/AR), Springer
Verlag, 2.te Auflage, 2019

*Buch, das eine Anwendung
von Computergraphik
beleuchtet. Enthält Erläuterung
zu mathematischen
Grundlagen der
Computergraphik (Kapitel 11).
Kostenlos als e-Book unter
springerlink.com*

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [50]



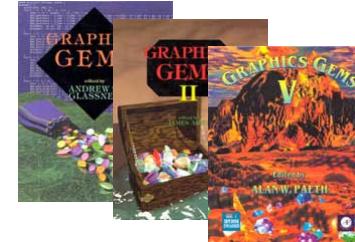
© R. Dörner

50

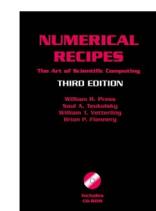


Weitere nützliche Bücher

- 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 20

B-AI-V1 [51]



© R. Dörner

51

Forschungsrelevante Literatur



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [52]



© R. Dörner

52



Acknowledgements

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



Teil B: Szenengraphen und Koordinatensysteme



Szenengraphen und Koordinatensysteme

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



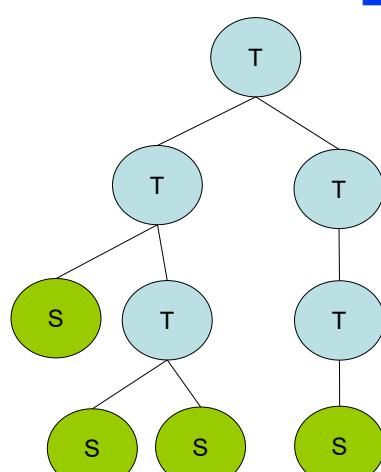
Szenengraphen und Koordinatensysteme

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



Szenengraph

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



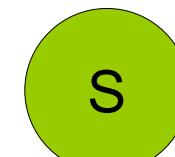
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [57]

© R. Dörner

57

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 20 B-AI-V1 [58]

© R. Dörner

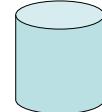
58



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  
}
```

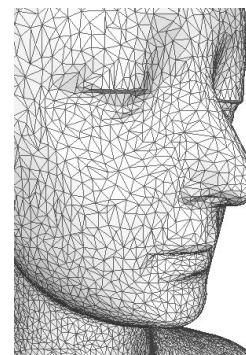


```
Cylinder{  
    bottom   FALSE  
    side     TRUE  
    top      TRUE  
    height   4  
    radius   2  
}
```



Polygonnetze

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

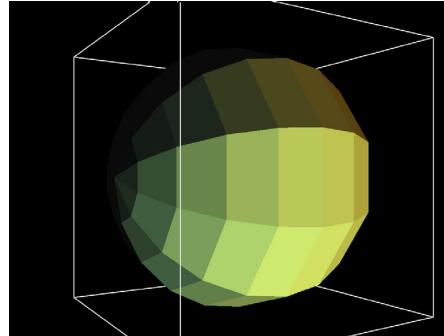


Quelle: <http://glasnost.itcarlow.ie>



Polygonnetze

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

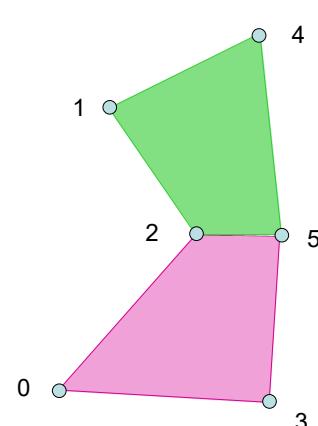


Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [61]  © R. Dörner

61

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 20 B-AI-V1 [62]  © R. Dörner

62



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 20

B-AI-V1 [63]

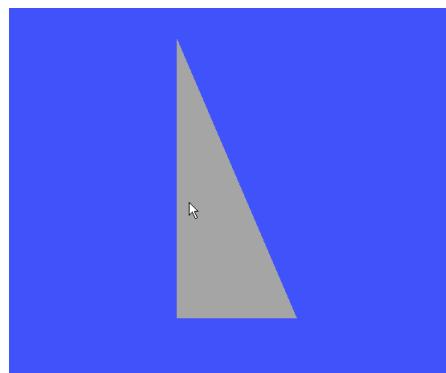


© R. Dörner

63

Polygonnetze

Beispiel: Verwendung von **Polygonnetzen** in VRML



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [64]



© R. Dörner

64

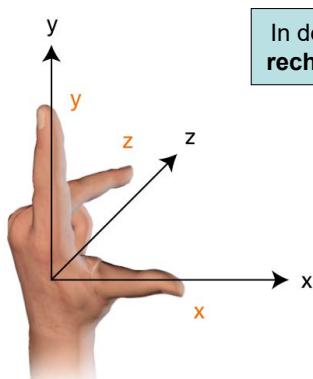


IndexedFaceSet

- Vorsicht: Koordinaten in rechtshändigem Koordinatensystem angeben

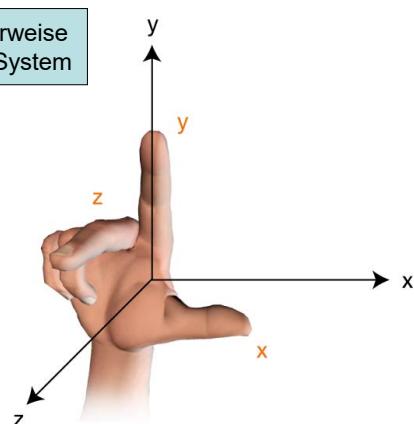


Koordinatensysteme



linkshändiges Koordinatensystem

In der CG üblicherweise
rechtshändiges System



rechtshändiges Koordinatensystem



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 20

B-AI-V1 [67]

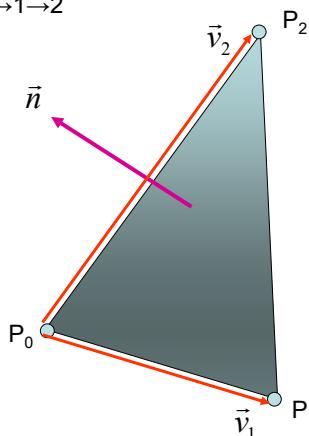


© R. Dörner

67

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 20

B-AI-V1 [68]



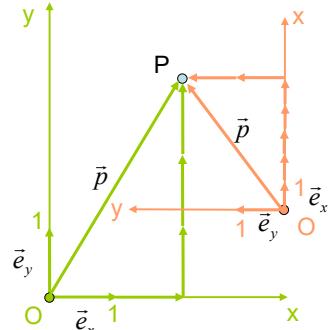
© R. Dörner

68



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



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

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

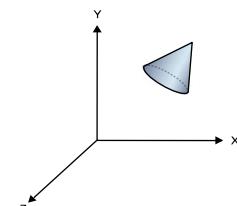
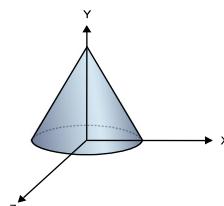


Wahl des lokalen Koordinatensystems

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



Szenenraphen und Koordinatensysteme

B.1 Shape-Nodes

B.2 Transform-Nodes und Objekthierarchien

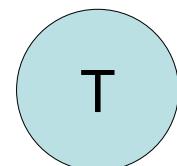
B.3 Transformationsmatrizen

B.4 Wechsel von Koordinatensystemen



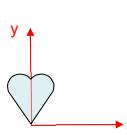
Transform - Node

- 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.



Transform Node

```
Transform{
    children[
        Shape{
            ...
        }
    ]
}
```



Die Shape in Objektkoordinaten

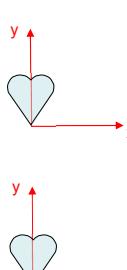
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [73]

© R. Dörner

73

Transform Node

```
Transform{
    translation 2 1 0
    children[
        Shape{
            ...
        }
    ]
}
```



Die Shape in Objektkoordinaten

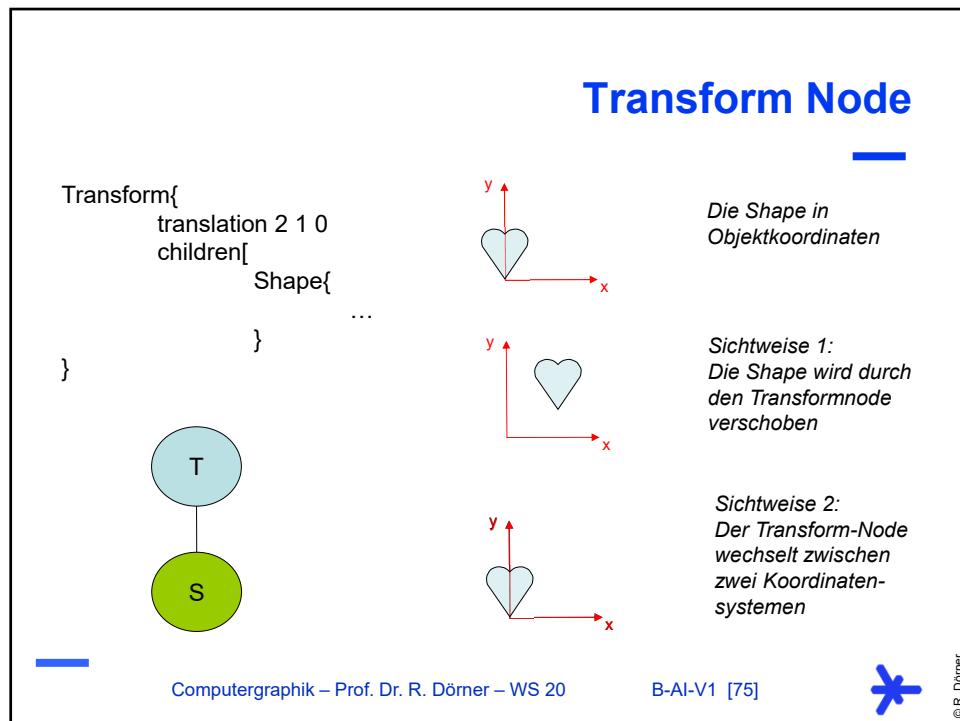
Sichtweise 1:
Die Shape wird durch
den Transformnode
verschoben

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [74]

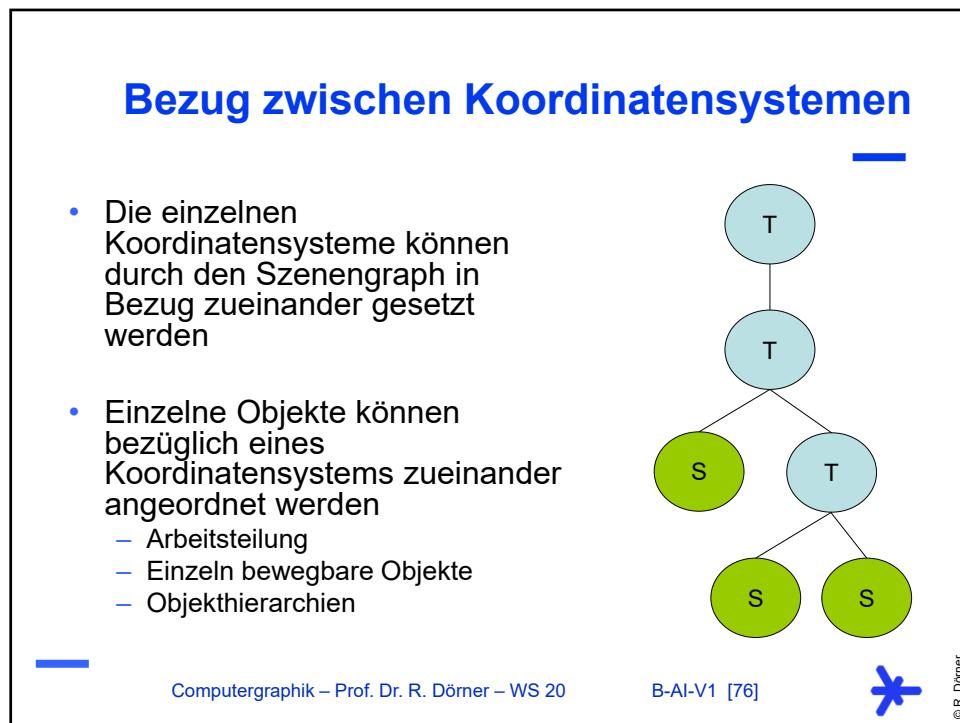
© R. Dörner

74





75

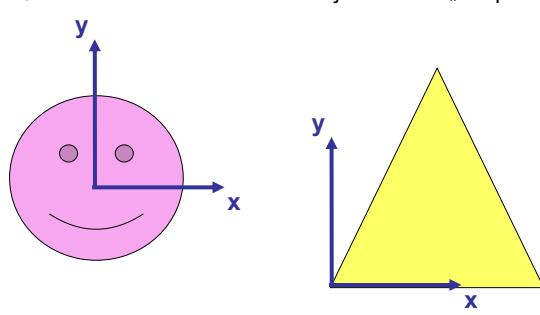


76

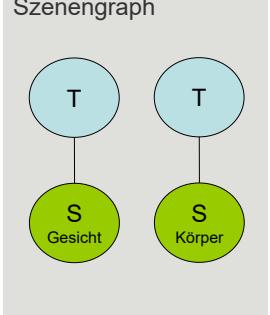


Beispiel

Objektmodell „Gesicht“ Objektmodell „Körper“



Szenengraph



Jedes Objektmodell ist in eigenem Koordinatensystem definiert: lokales Koordinatensystem

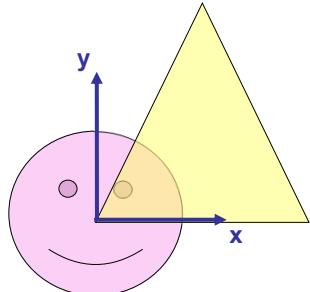
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [77]

© R. Dörner

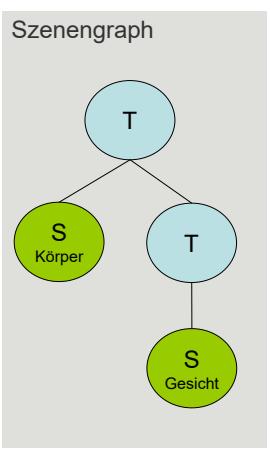
77

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 20 B-AI-V1 [78]

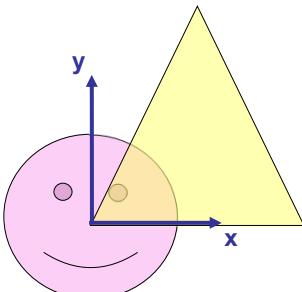
© R. Dörner

78



Beispiel

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



Szenengraph

```
graph TD; T1((T)) --- S1((S  
Körper)); T1 --- T2((T  
translation  
2 4 0)); T2 --- S2((S  
Gesicht))
```

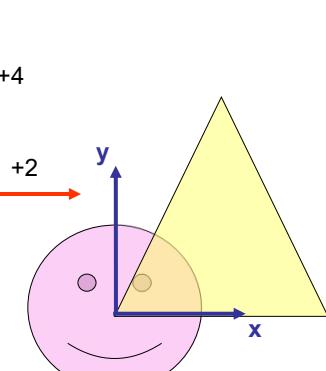
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [79]

© R. Dörner

79

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 20

B-AI-V1 [80]

© R. Dörner

80



Beispiel

Szenengraph

```
graph TD; T1[T] --- S1((S  
Körper)); T1 --- T2((T  
translation  
240)); T2 --- S2((S  
Gesicht))
```

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [81]

© R. Dörner

81

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  
240)); T2 --- S2((S  
Gesicht))
```

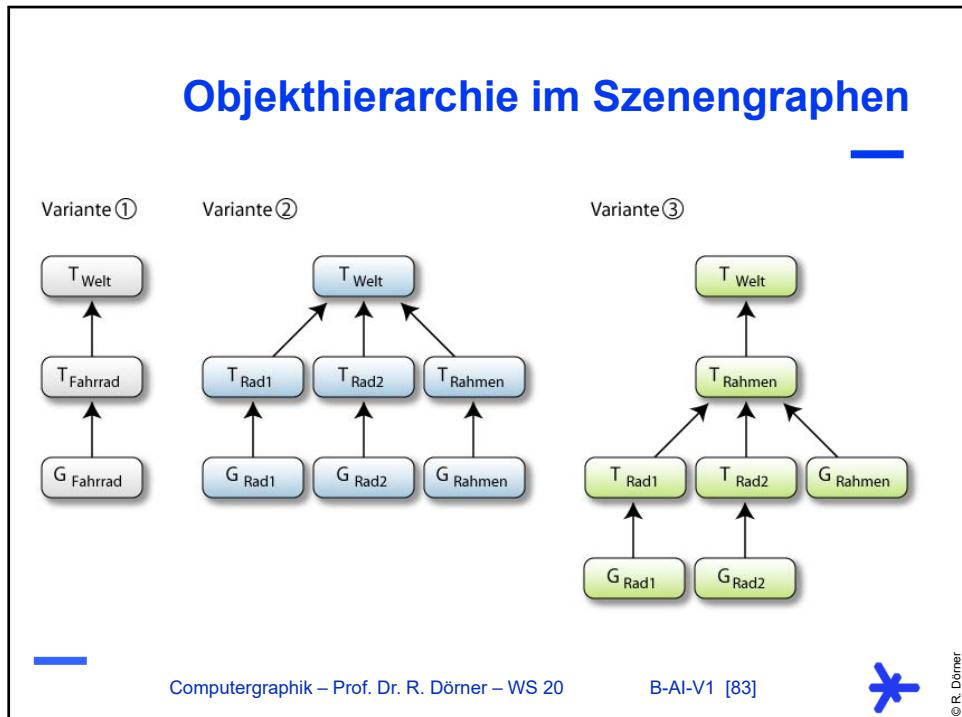
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [82]

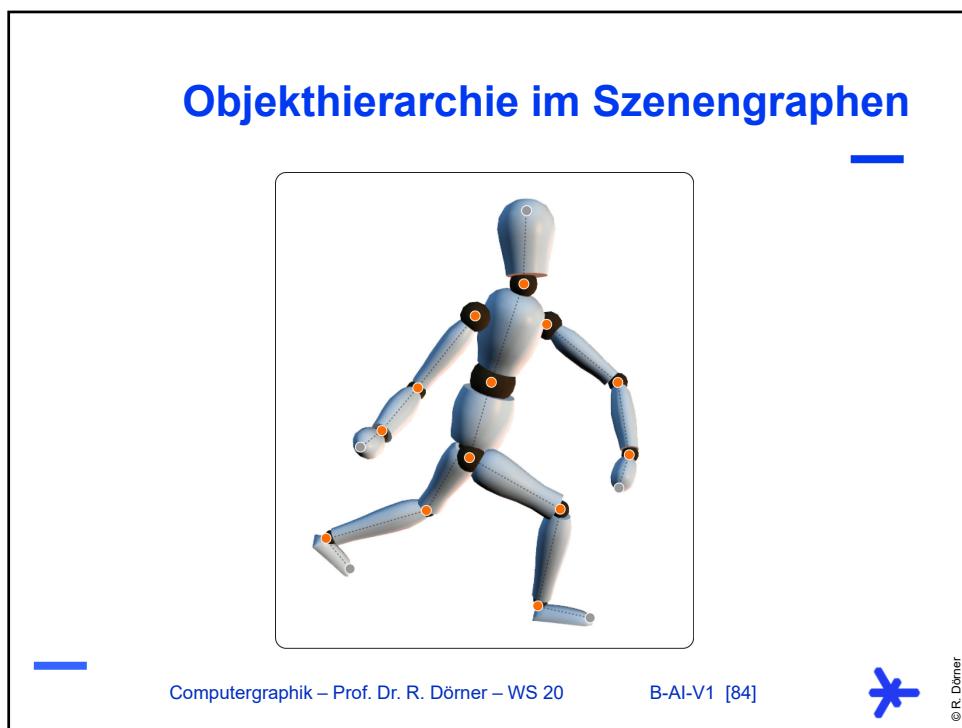
© R. Dörner

82





83



84



Objekthierarchie im Szenengraphen

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [85]

© R. Dörner

85

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 20 B-AI-V1 [86]

© R. Dörner

86



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 20

B-AI-V1 [87]

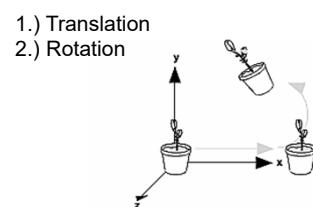
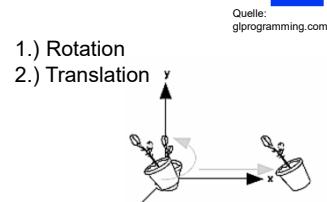


© R. Dörner

87

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [88]



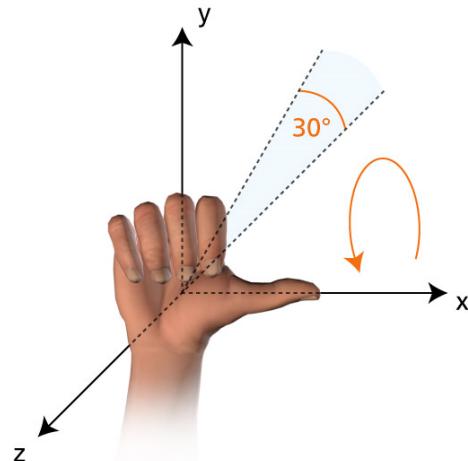
© R. Dörner

88



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



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



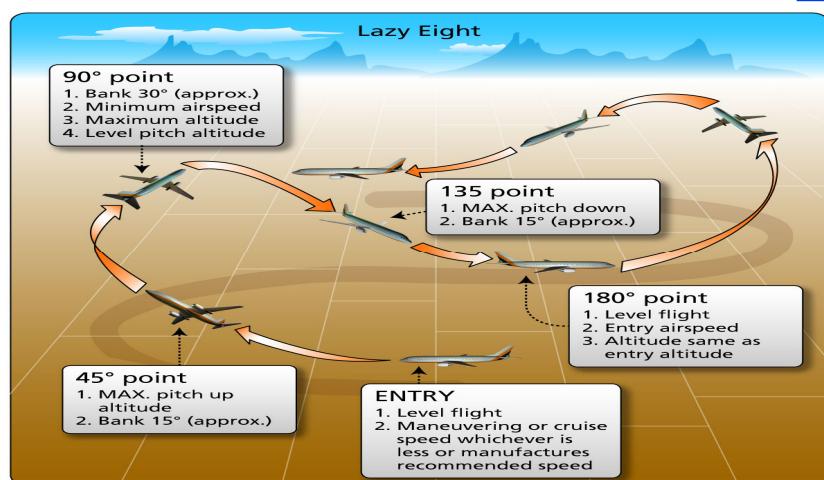
Transformationsmatrizen

- 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

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



Zerlegung von komplexen Transformationen



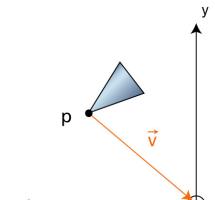
Zerlegung von komplexen Transformationen

- 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

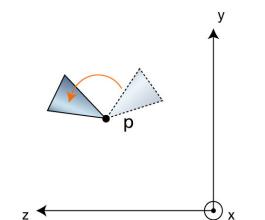
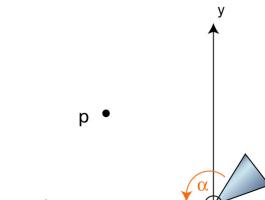
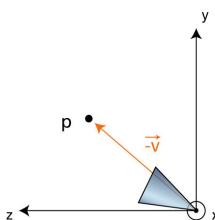
Jede affine Transformation lässt sich in Skalierung,
Verschiebung und Rotation zerlegen

Zerlegung von komplexen Transformationen

- Komplexe Transformation:
Drehung um
einen
beliebigen
Punkt P



- Zerlegung:
 1. Verschiebung
in Punkt P
 2. Drehung um
den Ursprung
 3. Rückverschie-
bung



Zerlegung von komplexen Transformationen

- Komplexe Transformation:
Drehung um eine beliebige Achse um Winkel ϕ
- Zerlegung:
 1. Drehung um die x-Achse um Winkel α
 2. Drehung um die y-Achse um Winkel β
 3. Drehung um die z-Achse um Winkel γ
- Die Winkel (α, β, γ) 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° um die y-Achse mit Drehpunkt $(2,0,0)$ abgebildet?

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



Beispiel

- Wohin wird der Vektor $\vec{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ durch eine um Translation $T(5,2,-1)$ abgebildet?

$$\vec{v}' = T(5,2,-1) \cdot \vec{v}$$

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



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:
 - $T(t_x, t_y, t_z)^{-1} = T(-t_x, -t_y, -t_z)$
 - $R_x(\alpha)^{-1} = R_x(-\alpha)$
 - $S(s_x, s_y, s_z)^{-1} = S(s_x^{-1}, s_y^{-1}, s_z^{-1})$



Transformation von Objekten

- Bei Polygonen: jeden Eckpunkt transformieren, Objekt neu zeichnen
- Transformation von Normalen: nicht mit M transformieren, sondern mit $(M^{-1})^T$



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

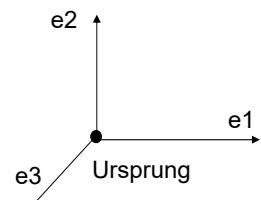
B.3 Transformationsmatrizen

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [117]

© R. Dörner

117

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [118]

© R. Dörner

118



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 e_x und e_y :

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

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

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [119] © R. Dörner

119

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 20 B-AI-V1 [120] © R. Dörner

120



Beispiel

Umrechnung:

1. Transform-Node entspricht $T(2,4,0)$
2. Umrechnung rot -> blau
 $\vec{p}' = T(2,4,0) \cdot \vec{p}$
3. Umrechnung blau -> rot
 $\vec{p}' = (T(2,4,0))^{-1} \cdot \vec{p}'$

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 20 B-AI-V1 [121] © R. Dörner

121

Transform Node in VRML

Jeder Transform-Node in VRML bestimmt eine Transformationsmatrix

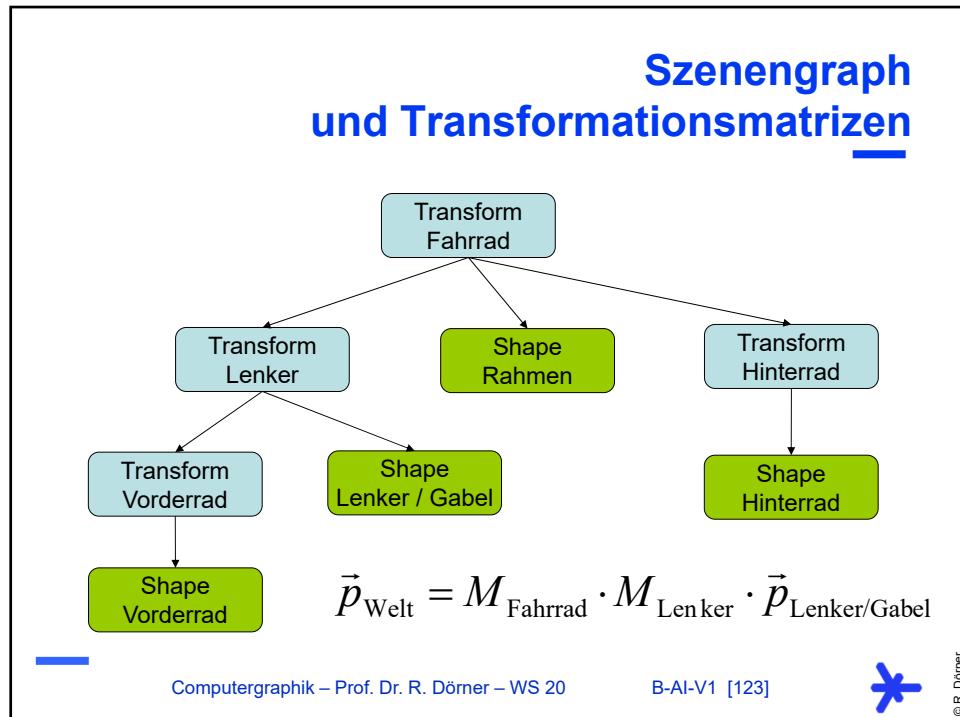
$$M = T(t) \cdot T(c) \cdot R \cdot R_s \cdot S \cdot R_s^T \cdot T(-c)$$

1. Skalierung **S** mit ScaleOrientation **R_s**
2. Rotation **R** um center point **c**
3. Translation **T(t)**

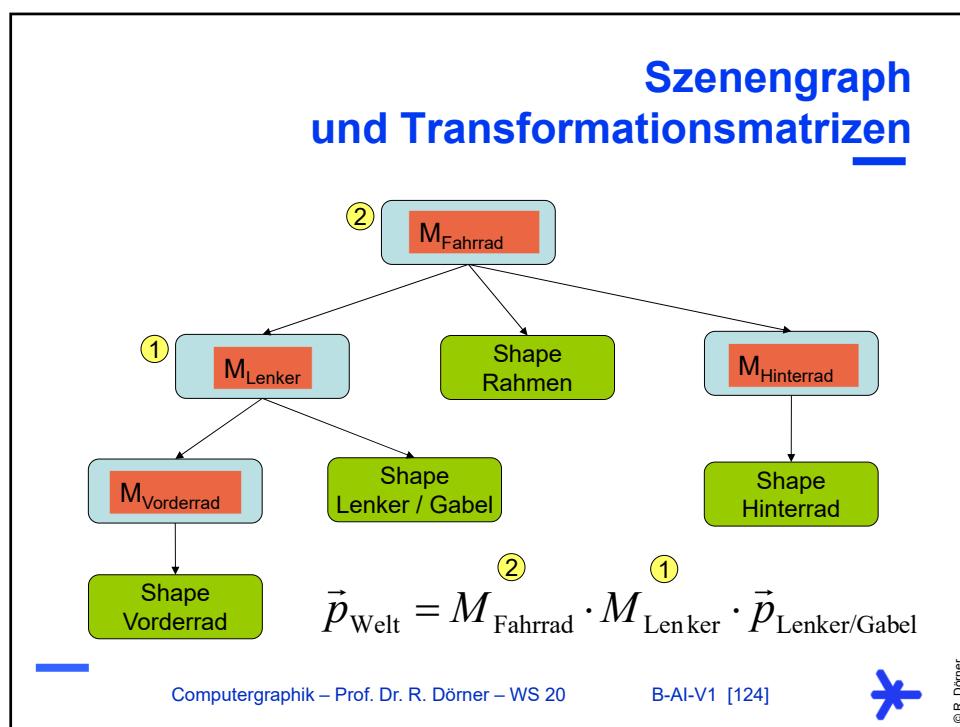
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [122] © R. Dörner

122





123



124



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

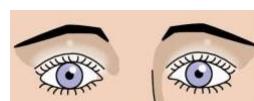


Teil C: Kameramodell

127

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)

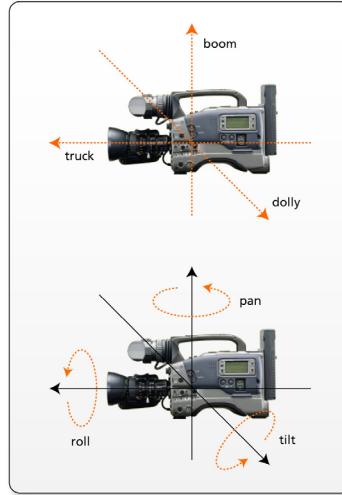


128



Äußere Parameter

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

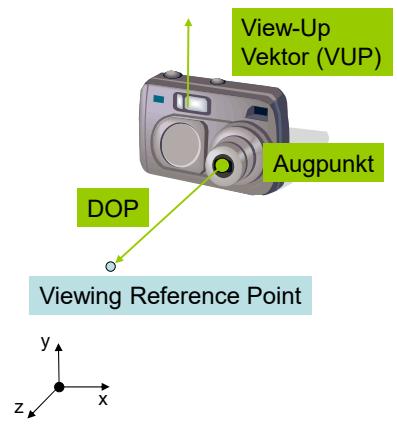


Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [129]  © R. Dörner

129

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



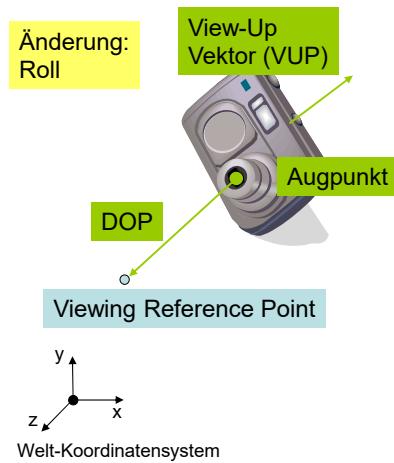
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [130]  © R. Dörner

130



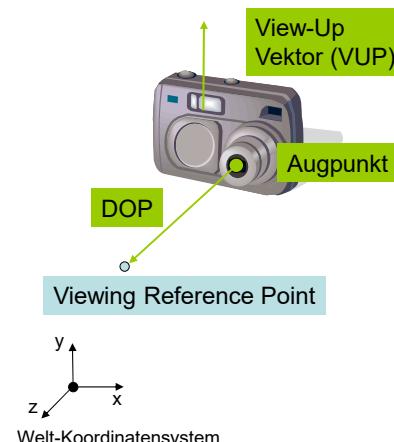
Spezifikation von Position und Orientierung

- 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

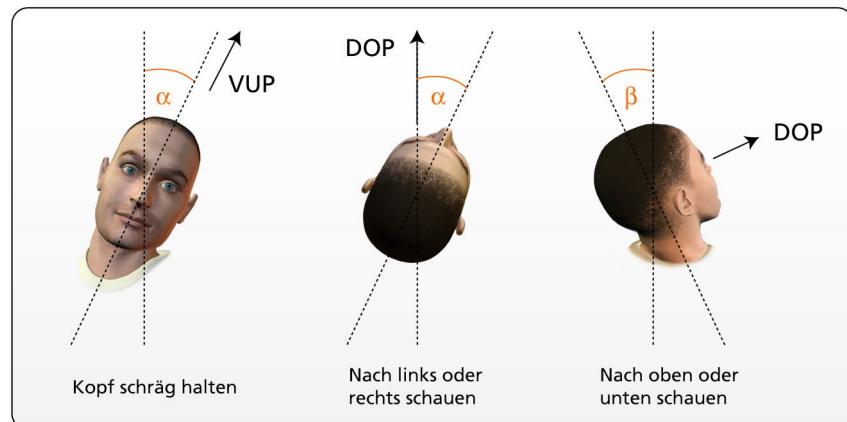


Spezifikation von Position und Orientierung

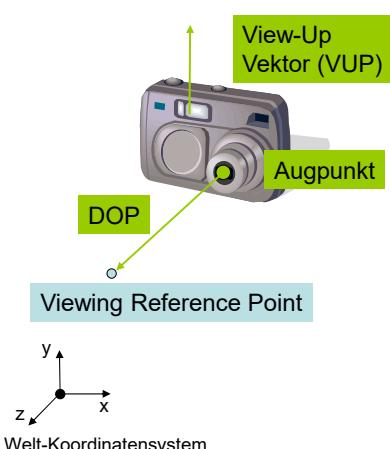
- Position der Kamera:
 - Angabe eines Punktes (Augpunkt) in Weltkoordinaten
- Ausrichtung der Kamera:
 - Angabe eines Punktes auf den die Kamera zentral blickt (Viewing Reference Point) in Weltkoordinaten
 - Vektor vom Augpunkt zum Viewing Reference Point heisst DOP (Direction of Projection)
 - Angabe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



Wahl von VUP und DOP

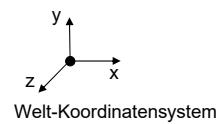
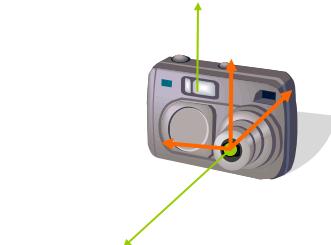


Kamerakoordinaten



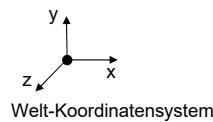
Kamerakoordinaten

- 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



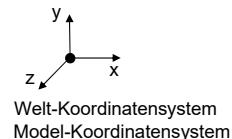
Kamerakoordinaten

- 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 äußereren Parameter



Innere Parameter

- Blickwinkel / Sehwinkel (Field of View)
- Seitenverhältnis (Aspect Ratio)



Augpunkt

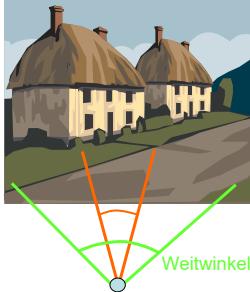


$$\text{Aspect Ratio} = h / b$$

Typische Werte sind 4:3 oder 16:9



Innere Parameter

- Blickwinkel / Sehwinkel (Field of View)


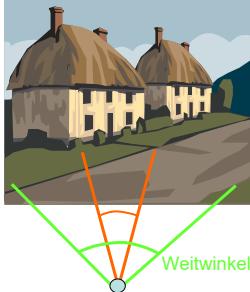
Augpunkt
Weitwinkel
- Seitenverhältnis (Aspect Ratio)


Breite b
Höhe h
Aspect Ratio = h / b
Typische Werte sind 4:3 oder 16:9

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [139]  © R. Dörner

139

Innere Parameter

- Blickwinkel / Sehwinkel (Field of View)


Augpunkt
Weitwinkel
- Seitenverhältnis (Aspect Ratio)


Breite b
Höhe h
Aspect Ratio = h / b
Typische Werte sind 4:3 oder 16:9

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [140]  © R. Dörner

140



Kamera Modell in VRML

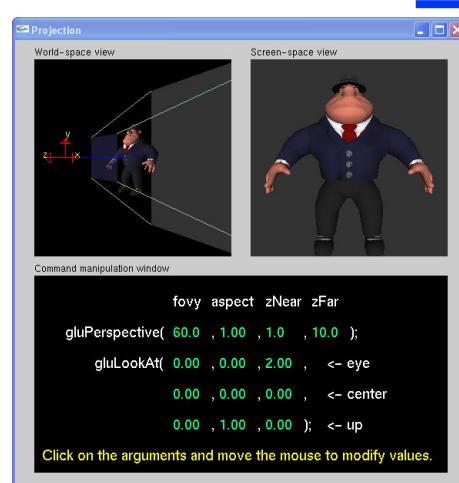
- fieldOfView: Sehwinkel in Radian
- position: Koordinaten des EyePoint
- orientation: Rotation aus einer Standardposition (schauen entlang negativer z-Achse)
- description: Unterscheidung mehrerer Viewpoints

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

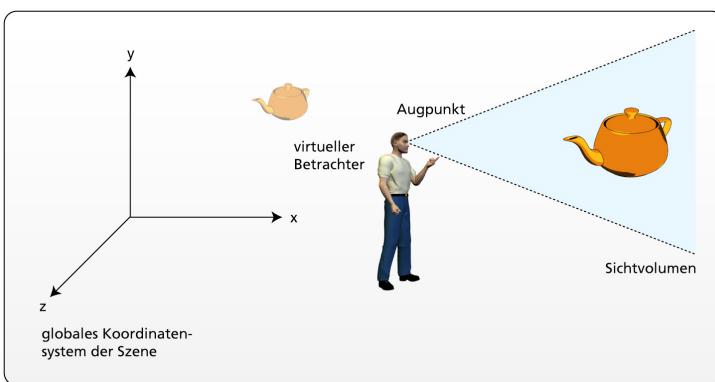


Andere Spezifikationsformen

- 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



Sichtvolumen



The diagram illustrates the viewing volume in a 3D coordinate system. A global coordinate system is shown with axes x, y, and z. A 'virtueller Betrachter' (virtual viewer) is positioned at a specific point, with an 'Augpunkt' (eye point) indicated. Two teapots are in the scene: one yellow one on the left and one orange one on the right. A light blue cone, representing the viewing volume, extends from the Augpunkt through the orange teapot. The text 'Sichtvolumen' is written at the bottom right of the cone. A callout box at the bottom states: 'Nur ein bestimmter Bereich der Szene kann im Bild erscheinen: Sichtvolumen' (Only a certain area of the scene can appear in the image: Viewing Volume).

Computergraphik – Prof. Dr. R. Dörner – WS 20

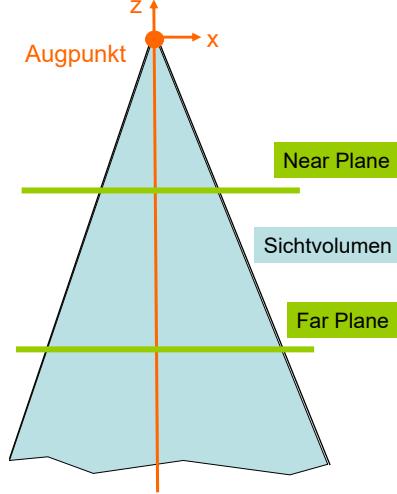
B-AI-V1 [143]

© R. Dörner

143

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



The diagram shows a 3D coordinate system with axes x, y, and z. A 'Near Plane' and a 'Far Plane' are represented as two parallel green horizontal lines. The region between these two planes is shaded light blue and labeled 'Sichtvolumen' (Viewing Volume). An 'Augpunkt' (eye point) is marked on the z-axis. The text 'Near Plane' and 'Far Plane' are placed near their respective lines.

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [144]

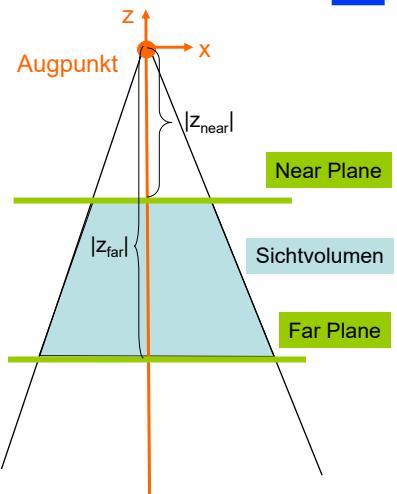
© R. Dörner

144



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

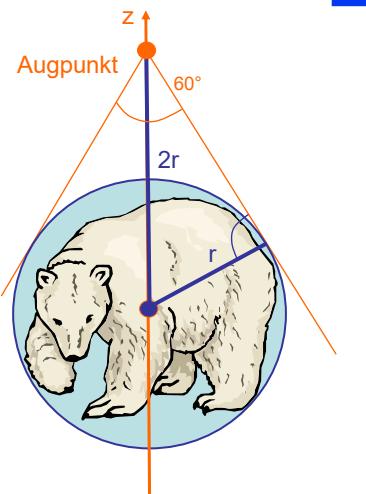


Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [145] © R. Dörner

145

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° erreicht, was positiv wahrgenommen wird



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [146] © R. Dörner

146

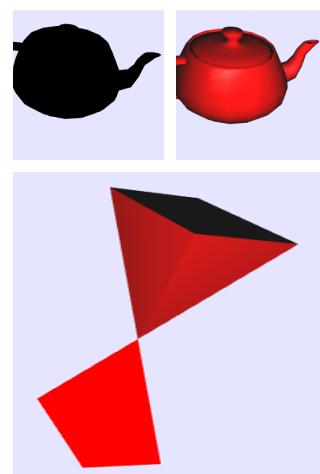


Teil D: Beleuchtungsmodell

147

Bedeutung der Beleuchtung

- Nicht nur Geometrie, sondern auch Beleuchtung muss modelliert werden
- Wichtig für menschliche Wahrnehmung u.a. von Formen und räumlicher Anordnung
- Interaktion von Licht (elektromagnetischen Wellen bzw. Lichtteilchen / Photonen) mit Objekten erlaubt Rückschlüsse auf die Welt



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [148]



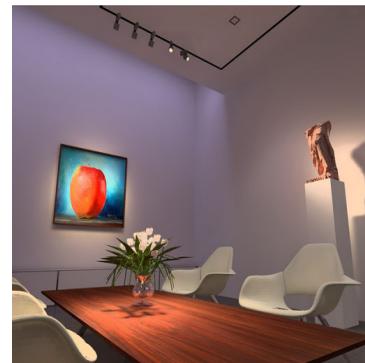
© R. Dörner

148



Beleuchtungsrechnung

- Beleuchtungsmodelle:
 - Modelle für Lichtquellen
 - Modelle für Materialien
 - Modelle für Interaktion Licht-Objekt
- Licht berechnen, das in virtuelle Kamera fällt, um Farbe von Bildpixeln zu ermitteln



Quelle: snipview.com

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [149]

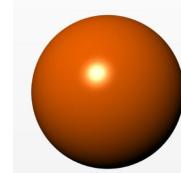
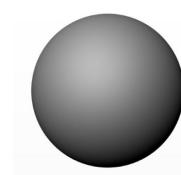


© R. Dörner

149

Beispiel: Phong - Beleuchtungsmodell

- Benannt nach Bùi Tường Phong (1942 - 1975)
- Ziel: Beleuchtungsrechnung in Echtzeit
- Nutzung von Vereinfachungen
 - Keine Berücksichtigung bestimmter Effekte
 - Superpositionsprinzip
 - Lokale Berechnung
 - Abstraktion von Lichtquellen



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [150]



© R. Dörner

150



Beispiel: Phong - Beleuchtungsmodell

- Benannt nach Bùi Tường Phong (1942 - 1975)
- Ziel: Beleuchtungsrechnung in Echtzeit
- Nutzung von Vereinfachungen
 - Keine Berücksichtigung bestimmter Effekte
 - Superpositionsprinzip
 - Lokale Berechnung
 - Abstraktion von Lichtquellen



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [151] © R. Dörner

151

Beispiel: Phong - Beleuchtungsmodell

- Benannt nach Bùi Tường Phong (1942 - 1975)
- Ziel: Beleuchtungsrechnung in Echtzeit
- Nutzung von Vereinfachungen
 - Keine Berücksichtigung bestimmter Effekte
 - Superpositionsprinzip
 - Lokale Berechnung
 - Abstraktion von Lichtquellen



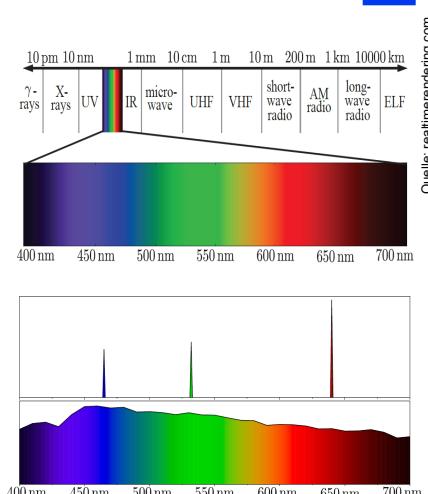
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [152] © R. Dörner

152



Phong: Superpositionsprinzip

- Beleuchtungsrechnung wird für jede Wellenlänge von Licht unabhängig durchgeführt
- Gesamtergebnis durch einfaches Überlagern (Superposition)
- Im Visual Computing: Berechnung für drei Wellenlängen (R, G, B)



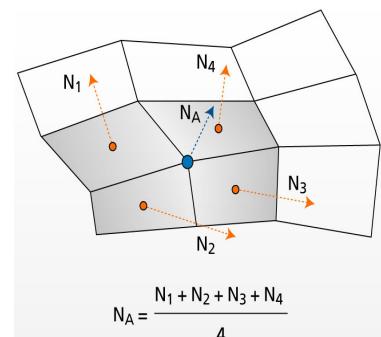
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [153]

© R. Dörner

153

Phong: Lokale Berechnung

- Beleuchtungsrechnung für einzelne Punkte P (meist Vertices in einem Polygonnetz)
- Normalenvektor beschreibt geometrische Situation an P
 - Normale für einen Punkt nicht definiert
 - Lokale Umgebung des Punktes wird beachtet
 - Bestimmung der Normale ist Aufgabe des Autoren – es wird erwartet, dass jedem Punkt eine Normale zugeordnet ist
- Shading ist notwendig


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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [154]

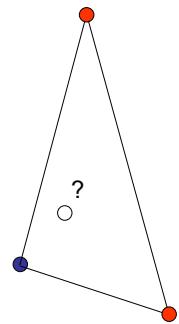
© R. Dörner

154

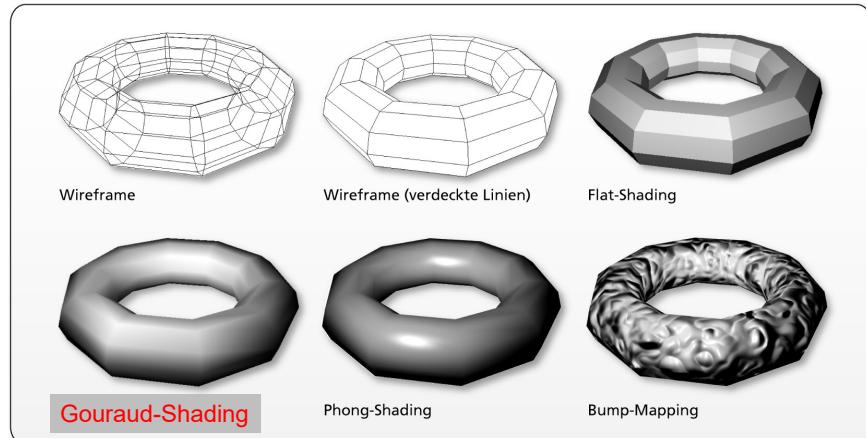


Shading

- Beleuchtungsrechnung wird nur für die Eckpunkte durchgeführt
- Welche Farben haben denn dann die Punkte, die keine Eckpunkte sind?
- Dies wird durch das Shading bestimmt
- Shadingverfahren \neq Beleuchtungsmodell

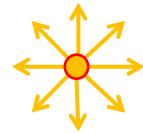


Shading: Beispiele



Phong: Abstraktion von Lichtquellen

- Lichtquellen ohne Ausdehnung
- Beispiel: Punktlichtquelle
 - Unendlich kleiner Punkt
 - Strahlt Licht mit Intensität $I^{(\lambda)}$ in alle Richtungen aus
- Für jede Lichtquelle Berechnung einzeln durchführen und Ergebnisse addieren



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [157]

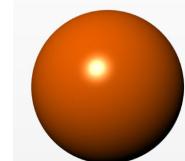
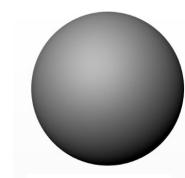


© R. Dörner

157

Beispiel: Phong - Beleuchtungsmodell

- Benannt nach Bùi Tường Phong (1942 - 1975)
- Ziel: Beleuchtungsrechnung in Echtzeit
- Nutzung von Vereinfachungen
 - Keine Berücksichtigung bestimmter Effekte
 - Superpositionsprinzip
 - Lokale Berechnung
 - Abstraktion von Lichtquellen



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [158]



© R. Dörner

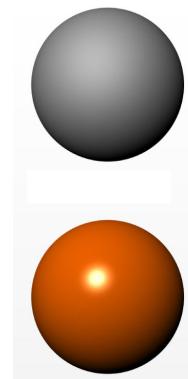
158



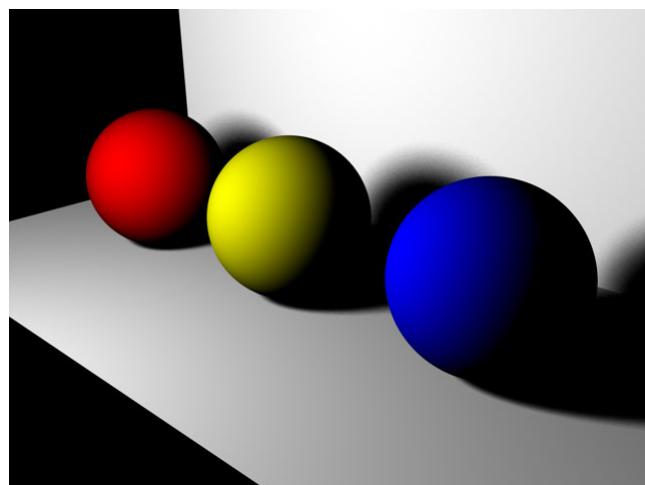
Beispiel: Phong - Beleuchtungsmodell

- Benannt nach Bùi Tường Phong (1942 - 1975)
- Ziel: Berechnung muss nicht physikalisch korrekt sein, sondern nur gut aussehende Ergebnisse liefern
- Nutzen: Lichtquellen
- Abstraktion von Lichtquellen

Grundidee:



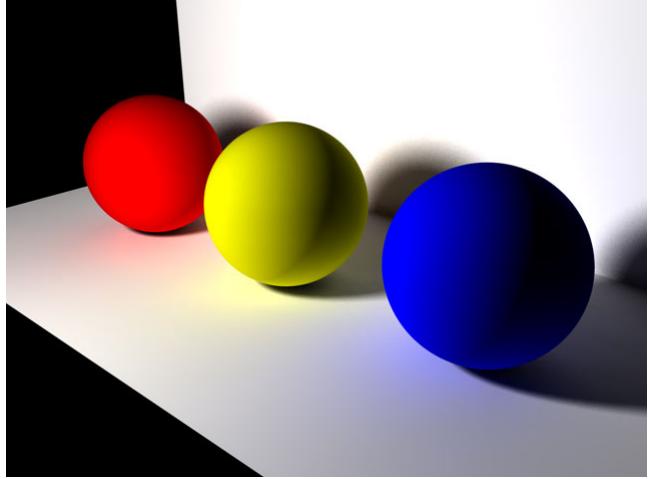
Plausibel aussehende Darstellung



Quelle:
Watt / Pollicardo 3D Games



Physikalisch korrekte Darstellung



Quelle:
Watt / Pollicapo 3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [161]

© R. Dörner

161

Ambientes Licht

- Licht nicht nur direkt von Lichtquellen
- Gegenseitige Beleuchtung von Objekten
- Idee von Phong: nicht berechnen, Szene gleichmäßig aufhellen



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [162]

© R. Dörner

162



Phong: Ambientes Licht

- Ambientes Licht (für Wellenlänge λ ist in der Szene konstant:

Intensität des
ambienten Lichts an
Punkt P

$$I_a^{(P,\lambda)} = I_a^{(\lambda)} \cdot R_a^{(P,\lambda)}$$

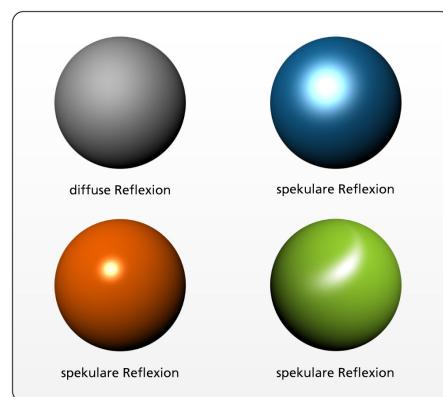
Intensität des
ambienten Lichts in
der Szene

Reflexionskoeffizient
(=Anteil reflektiertes
Licht, 0% - 100%) für
ambientes Licht an P,
Materialeigenschaft

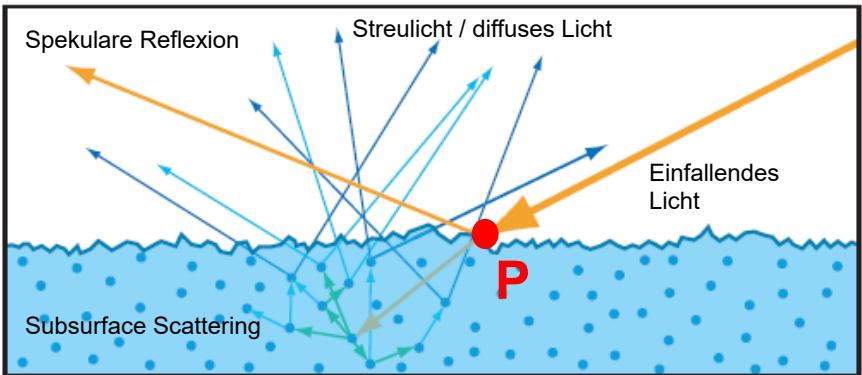


Phong Beleuchtungsanteile

- Drei Beleuchtungsanteile
 - Ambient
 - Diffus
 - Spekular
- Diffus: matte Oberflächen
- Spekular: glattes Objekt, Lichtquelle spiegelt sich



Diffuses und Spekulares Licht



The diagram shows a cross-section of a medium with a wavy surface. A point **P** is marked on the surface. An orange arrow labeled "Einfallendes Licht" (incident light) enters the medium at point **P**. Inside the medium, the light undergoes several interactions: "Spekulaire Reflexion" (specular reflection) is shown as an orange arrow reflecting off the surface; "Streulicht / diffuses Licht" (scattered/diffuse light) is shown as blue arrows scattering in various directions; and "Subsurface Scattering" is shown as green arrows scattering within the medium. The medium is depicted with a blue background and small blue dots representing scattering particles. A vertical line on the right side of the diagram is labeled "Quelle: realtime rendering.com".

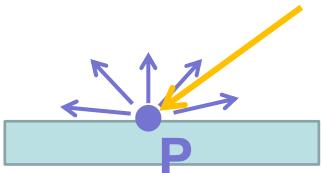
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [165]

© R. Dörner

165

Diffuses Licht: Lambert - Reflektor

- Einfachster Fall: Lambert-Reflektor
- Gleichmäßige Streuung



The diagram shows a light blue rectangular surface representing a Lambertian reflector. A point **P** is marked on the surface. From point **P**, several blue arrows point in different directions, representing the uniform scattering of light, which is characteristic of a Lambertian reflector. A vertical line on the right side of the diagram is labeled "Quelle: realtime rendering.com".

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [166]

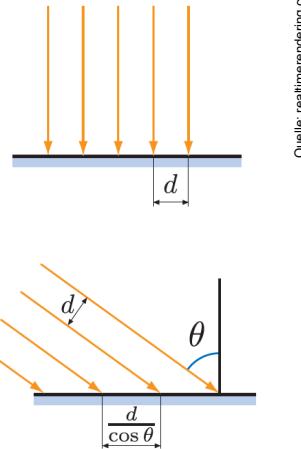
© R. Dörner

166



Diffuses Licht: Lambert - Reflektor

- Einfachster Fall: Lambert-Reflektor
- Gleichmäßige Streuung
- Abhängigkeit vom Winkel mit dem das Licht einfällt: je schräger, auf eine desto größere Fläche wird es verteilt



Quelle: realtime rendering.com

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [167]

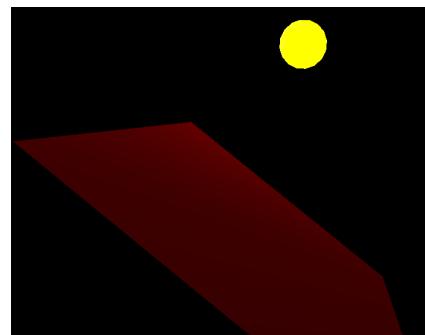


© R. Dörner

167

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [168]



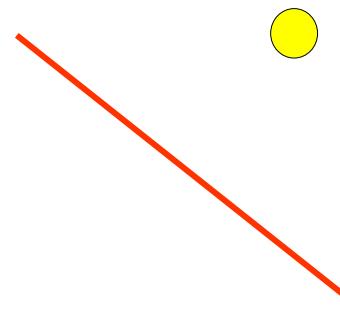
© R. Dörner

168



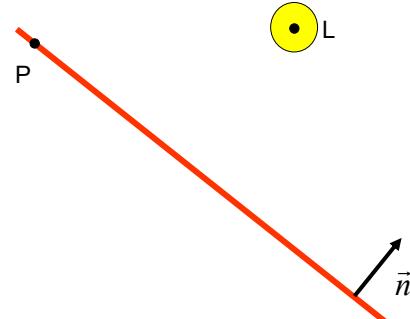
Diffuse Reflexion nach Phong

- 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



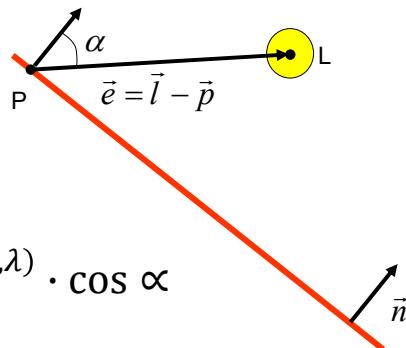
Diffuse Reflexion nach Phong

- λ : Betrachtete Wellenlänge
- $I_d^{(L, \lambda)}$: Intensität der Lichtquelle im Diffusen
- $R_d^{(P, \lambda)}$: Reflektionskoeffizient für diffuses Material am Punkt P
- \vec{n} : Normalenvektor an P



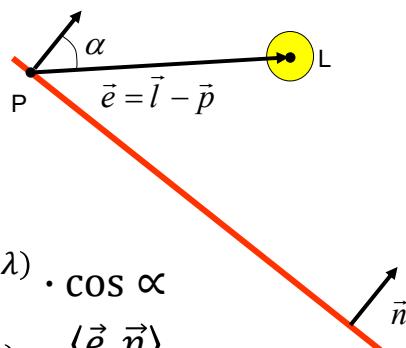
Gesucht: $I_d^{(P, \lambda, L)}$

Diffuse Reflexion nach Phong



$$I_d^{(P,\lambda,L)} = I_d^{(L,\lambda)} \cdot R_d^{(P,\lambda)} \cdot \cos \alpha$$

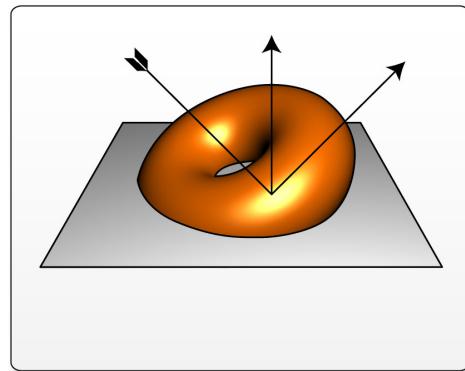
Diffuse Reflexion nach Phong



$$\begin{aligned} I_d^{(P,\lambda,L)} &= I_d^{(L,\lambda)} \cdot R_d^{(P,\lambda)} \cdot \cos \alpha \\ &= I_d^{(L,\lambda)} \cdot R_d^{(P,\lambda)} \cdot \frac{\langle \vec{e}, \vec{n} \rangle}{|\vec{e}| \cdot |\vec{n}|} \end{aligned}$$

Spekulare Reflexion nach Phong

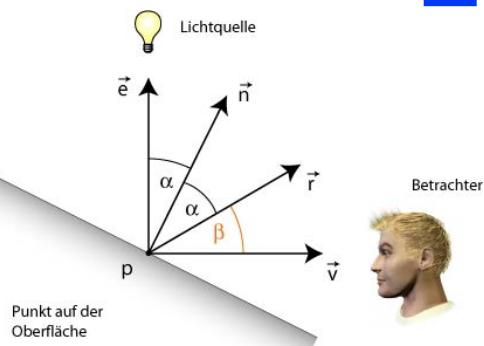
- Glanzlicht (engl. Highlight): Licht trifft auf glatte Oberfläche und wird nach dem Reflexionsgesetz „Einfallsinkel = Ausfallswinkel“ reflektiert
- Highlights haben Farbe der Lichtquelle
- Je glatter das Objekt, desto schärfer begrenzt und kleiner sind die Highlights
- Position des Betrachters ist wichtig



Spekulare Reflexion nach Phong

- Die Shininess σ ist ein Maß für die Glattheit der Oberfläche

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

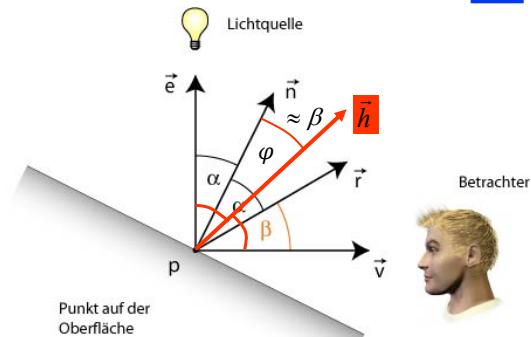


$$\cos \beta = \frac{\langle \vec{r}, \vec{v} \rangle}{|\vec{r}| \cdot |\vec{v}|}$$

$$I_s^{(P, \lambda, L)} = I_s^{(L, \lambda)} \cdot R_s^{(P, \lambda)} \cdot (\cos \beta)^\sigma$$

Phong-Blinn Beleuchtungsmodell

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



$$I_s^{(P, \lambda, L)} \approx I_s^{(L, \lambda)} \cdot R_s^{(P, \lambda)} \cdot (\cos \varphi)^{\sigma'}$$

$$\cos \varphi = \frac{\langle \vec{n}, \vec{h} \rangle}{|\vec{n}| \cdot |\vec{h}|}$$

$$\vec{h} = \frac{\vec{e} + \vec{v}}{|\vec{e} + \vec{v}|}$$

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [175]

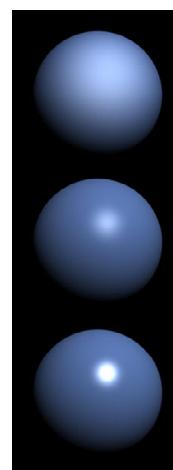


© R. Dörner

175

Shininess

- je höher der Wert von σ , desto scharf umrissener das Highlight, desto glatter wirkt das Material
- Wert hat keine physikalische Bedeutung
- Ausprobieren (z.B. Aufgabe von Artists)



Quelle: cgru.sourceforge.net

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [176]



© R. Dörner

176



Phong Beleuchtungsmodell

Für $\lambda \in \{\text{rot, grün, blau}\}$

$$I_{Phong}^{(P,\lambda)} = I_a^{(P,\lambda)} + \sum_{L \in \text{Lichtquellen}} I_d^{(P,\lambda,L)} + I_s^{(P,\lambda,L)}$$



Realisierung

- Beispiel: Vertex Shader schreiben für Beleuchtungsrechnung auf der GPU
- Beachten: Clamping auf $[0,1]$ und negative Skalarprodukte behandeln
- Benötigte Daten:
 - Geometriemodell (Inklusive Normale)
 - Lichtquellenmodellierung
 - Materialmodellierung

```
// Position Eckpunkt in Viewkoordinaten berechnen
vec3 pos = (viewMatrix * modelMatrix * vPosition).xyz;

// Position der Lichtquelle
vec3 light = (viewMatrix * lightPosition).xyz;

// Vektor der Laenge l vom Eckpunkt zur Lichtquelle
vec3 L = normalize(light - pos);

// Normale am Eckpunkt in Weltkoordinaten berechnen
vec3 N = (normalMatrix * vec4(vNormal, 0.0)).xyz;
N = normalize(N);

// Berechnung der diffusen Beleuchtung nach den Formeln
// von Phong
float Kd = max( dot(L, N), 0.0 );
vec3 diffuse = Kd*diffuseProduct.rgb;

// Spekular
vec3 E = normalize(-pos);
vec3 H = normalize(L + E);

float Ks = pow(max(dot(N, H), 0.0), shininess);
vec4 specular = Ks * specularProduct;

if (dot(L, N) < 0.0 ){
    specular = vec4(0.0, 0.0, 0.0, 1.0 );
}

// resultierende Farbe bestimmen
fColor = vec4(diffuse.xyz, 1.0) + ambientIntensity *
vec4(ambientProduct.xyz, 1.0) + vec4(specular.xyz, 1.0);
```



Modelle für Material

- Beispiel: Material Node in VRML

```

material Material {
    Albedo
    diffuseColor
    specularColor
    shininess
    ambientIntensity
    emissiveColor
    transparency
}

```

Objekt soll so aussehen, als ob es grün leuchtet

σ

$R_d^{(P, \lambda=rot)}$

0.3	0.0	0.4
0.2	0.3	0.5
1.0		
0.2		
0.0	0.3	0.0
0.3		

$R_s^{(P, \lambda=blau)}$

Durchsichtigkeit: man sieht den Hintergrund zu 70%

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [179]

© R. Dörner

179

Modelle für Material

- Bibliotheken mit Zahlenwerten für spezifische Materialien
 - Literatur
 - Asset-Stores
 - ...
- Verschiedene Materialien:
 - Anistrop: keine rotationssymmetrische Reflexion
 - Dielektrikum: nicht-metallische Substanzen
- Verhältnis Lichtwellenlänge zu Oberflächenstrukturen ist wichtig für Lichteffekte

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [180]

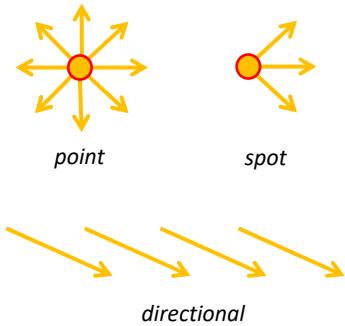
© R. Dörner

180



Modelle für Lichtquellen

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



Punktlicht in VRML

- Verwendung des Nodes PointLight

```
PointLight {  
    location          0 0 1  
    intensity         0.5  
    color             0.2 0 0.7  
    ambientIntensity 0.3  
    radius            100  
    attenuation       0 0 1  
}
```

Punktlicht in VRML

- Verwendung des Nodes PointLight

```
PointLight {  
    location 0 0 7  
    intensity 0.5  
    color 0.2 0 0.7  
    ambientIntensity 0.3  
    radius 100  
    attenuation 0 0 1  
}
```

$$I_d^{(L, \text{rot})} = 0.1$$



Punktlicht in VRML

- Verwendung des Nodes PointLight

```
PointLight {  
    location 0 0 7  
    intensity 0.5  
    color 0.2 0 0.7  
    ambientIntensity 0.3  
    radius 100  
    attenuation 0 0 1  
}
```

Wirkungskreis:
100 m



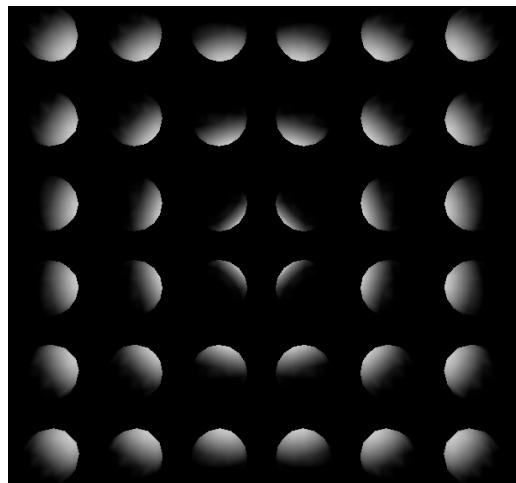
Punktlicht in VRML

- **Lichtdämpfung:** Intensität nimmt mit Entfernung d zur Lichtquelle ab.

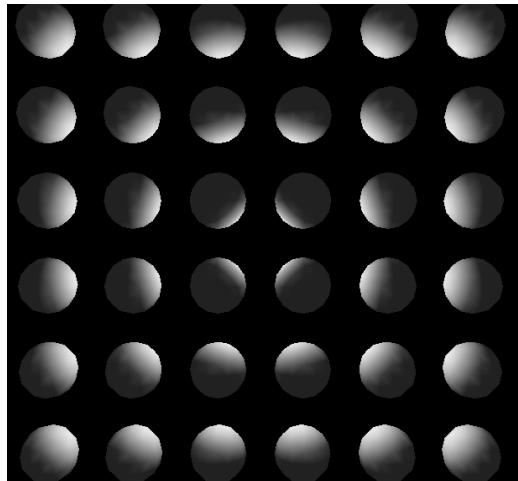
```
I' = 
$$\frac{I}{0.1 + 0.2 \cdot d + 0.3 \cdot d^2}$$
  
ambientIntensity 1  
radius 0.3  
attenuation 0.1 0.2 0.3  
}
```



Punktlicht



Punktlicht



Mit stärkerem
ambienten Licht

Computergraphik – Prof. Dr. R. Dörner – WS 20

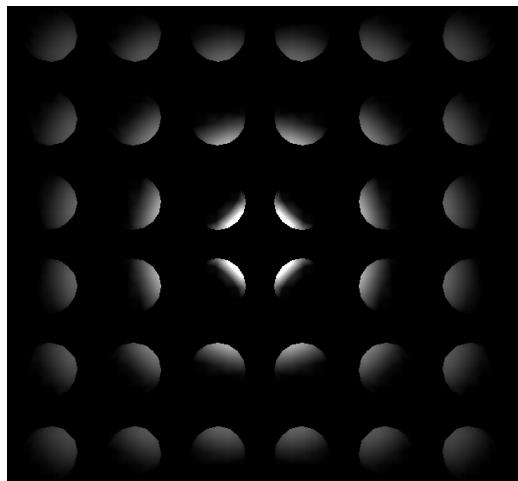
B-AI-V1 [187]



© R. Dörner

187

Punktlicht mit Lichtdämpfung



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [188]

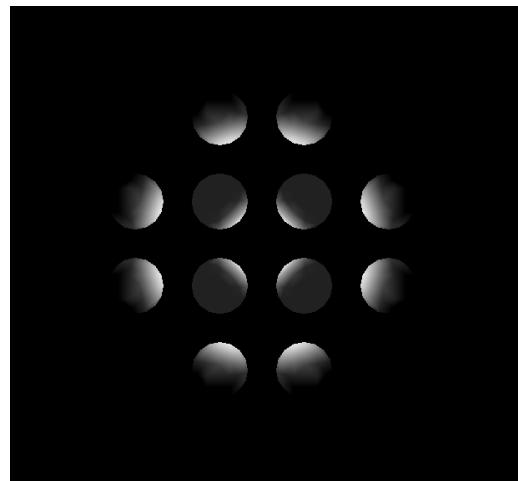


© R. Dörner

188



Punktlicht mit Wirkungsradius



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [189]



© R. Dörner

189

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 20

B-AI-V1 [190]

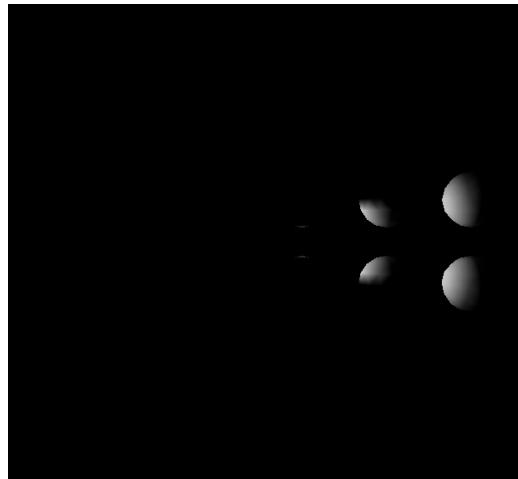


© R. Dörner

190



Spotlicht



Computergraphik – Prof. Dr. R. Dörner – WS 20

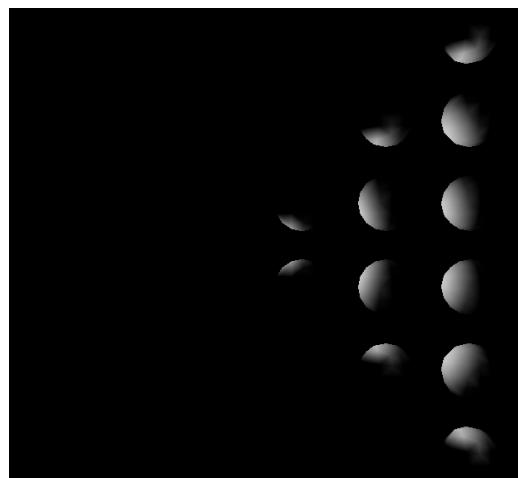
B-AI-V1 [191]



© R. Dörner

191

Spotlicht



Mit weiterem
Öffnungswinkel

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [192]



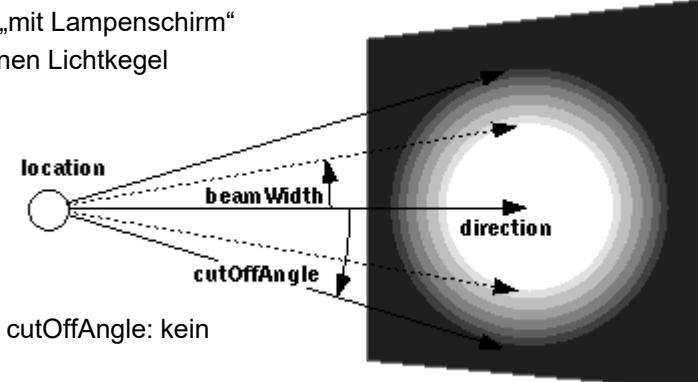
© R. Dörner

192



Spotlicht

- Punktlicht „mit Lampenschirm“
- Erzeugt einen Lichtkegel



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [193] © C. Schulz

193

Direktionales 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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [194] © R. Dörner

194



Direktionales Licht



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [195]



© R. Dörner

195

Vorteile Phong-Beleuchtungsmodell

- Punkt isoliert betrachtet: parallele Berechnung
- nicht ganze Information über Szene im Speicher
- gutes Verhältnis Aufwand / Resultat
- Hardware-Beschleunigung (früher noch wichtiger, da „fest verdrahtet“)



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [196]



© R. Dörner

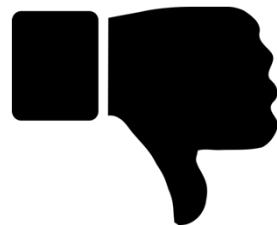
196



Grenzen Phong-Beleuchtungsmodell

Phong verwendet Näherungen (z.B. ambientes Licht) und kann **nicht** berechnen:

- Schatten
- Flächige Lichtquellen und Halbschatten
- Spiegelungen
- Transmission und Refraktion
- Kaustiken
- Subsurface-Scattering
- Anisotrope Materialien
- ...



Computergraphik – Prof. Dr. R. Dörner – WS 20

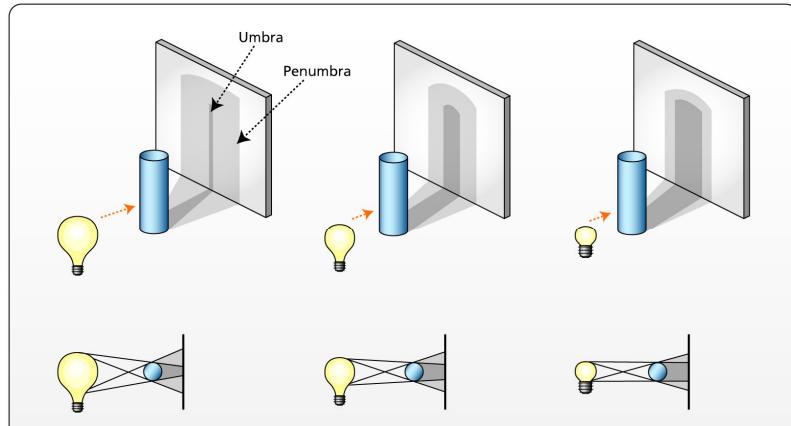
B-AI-V1 [197]



© R. Dörner

197

Flächige Lichtquellen und Halbschatten



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [198]



© R. Dörner

198



Transmission und Refraktion

TRANSMISSION

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [199]

Quelle: M. McGuire, M. Mars, A. Pharr
for Order-independent Transparency
Photorealistic Scatterabilg Model

© R. Dörner

199

Kaustiken

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [200]

© R. Dörner

200



Anistrophe Materialien



Quelle: realtimerendering.com

Computergraphik – Prof. Dr. R. Dörner – WS 20

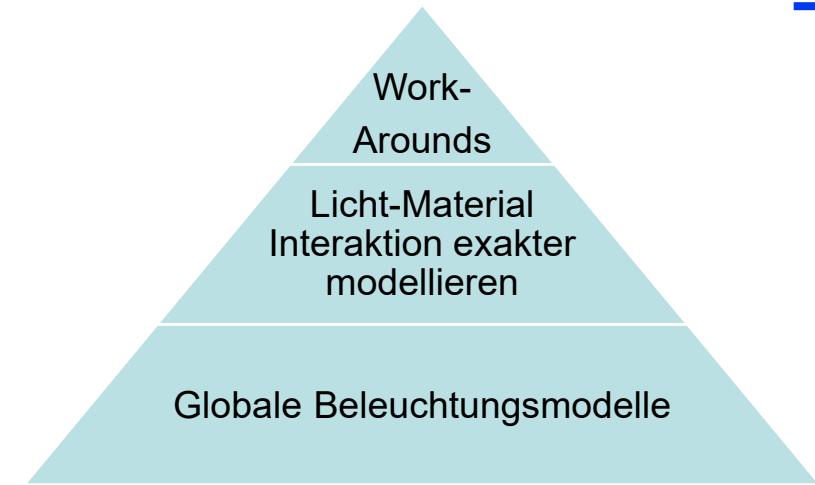
B-AI-V1 [201]



© R. Dörner

201

Grenzen von Phong überwinden



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [202]



© R. Dörner

202





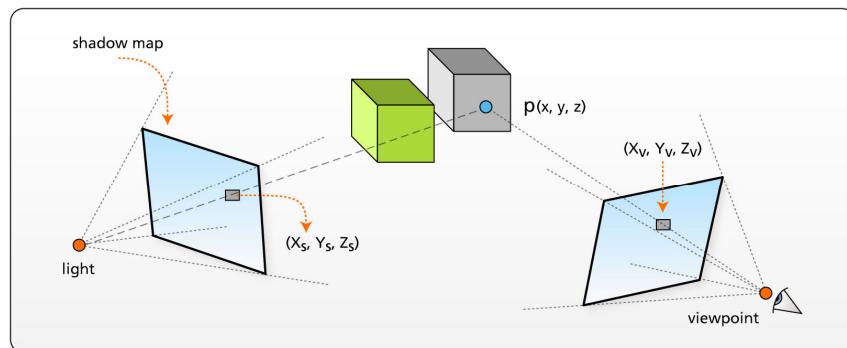
203



204



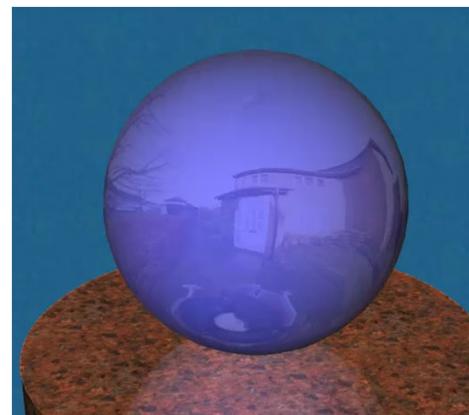
Beispiel: Shadow Maps



Schattenberechnung durch Visibilitysalgorithmen

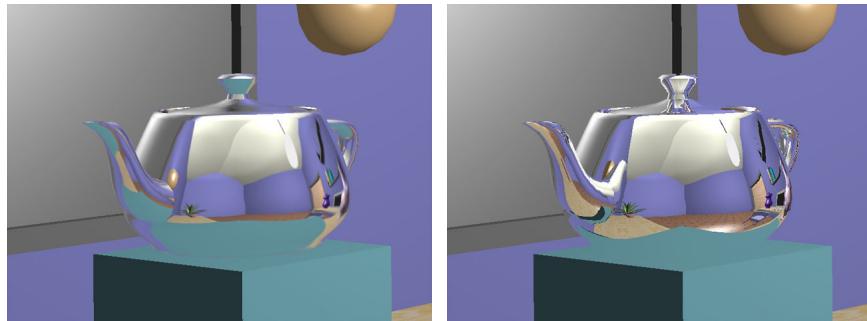
Beispiel: Environment Mapping

Idee: Spiegelung nicht berechnen, sondern Bild von der Umgebung erstellen und als Textur aufbringen



Grenzen des Environment Mappings

Quelle:
Watt / Policarpo
3D Games



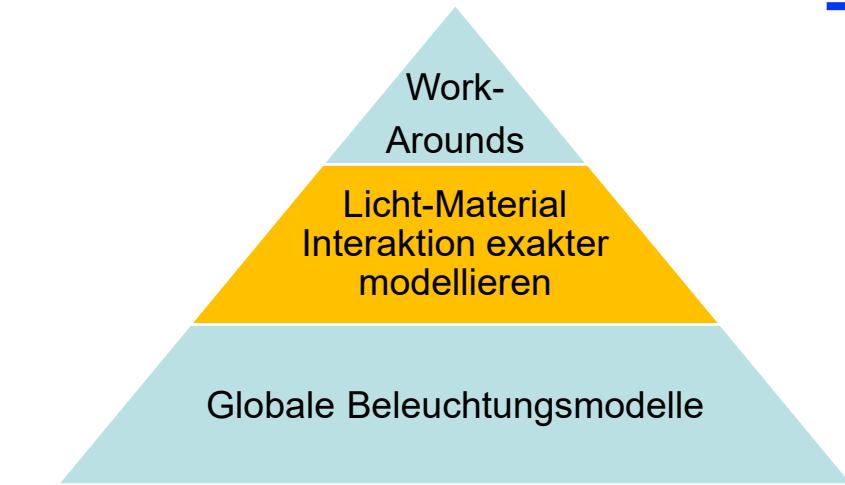
Computer Games – Prof. Dr. R. Dörner – SS 19 V1
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [207]



© R. Dörner

207

Grenzen von Phong überwinden



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [208]



© R. Dörner

208



Photometrie

- Strahlungsleistung Φ
 - Energie elektromagnetischer Strahlung [W]
 - engl. Radiant Flux
- Strahlungsintensität I
 - Strahlungsleistung pro Raumwinkel [W/sr]
 - engl. Radiant Intensity
- Strahldichte L
 - Strahlungsintensität pro Fläche [W/sr·m²]
 - engl. Radiance

Raumwinkel: A/r^2 mit r : Kugelradius,
 A : Flächeninhalt einer Teilfläche der
Kugeloberfläche. [sr (Steradian)]



Quelle: realtime-rendering.com

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [209] 

© R. Dörner

209

BRDF

Bidirectional Reflectance Distribution Function (BRDF) f_r



BRDF beschreibt Verhältnis von eingehender Lichtdichte aus Richtung eines Raumwinkels zu ausgehender Lichtdichte eines Raumwinkels

Quelle: J. Dupuy, W. Jakob: An Adaptive Parameterization for Material Acquisition and Rendering

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [210] 

© R. Dörner

210



BRDF

Bidirectional Reflectance Distribution Function (BRDF) f_r

$$f_r(\vec{l}, \vec{v}) = f_r(\theta_{in}, \phi_{in}, \theta_{out}, \phi_{out})$$

$$= \frac{dL_{out}(\vec{\omega}_{out})}{L_{in}(\vec{\omega}_{in}) \cdot \cos(\theta_{in}) d\vec{\omega}_{in}}$$

BRDF beschreibt Verhältnis von eingehender Lichtdichte aus Richtung eines Raumwinkels zu ausgehender Lichtdichte eines Raumwinkels

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [211]

© R. Dörner

211

BRDF

- BRDF berücksichtigt nur lokale Effekte (z.B. kein globales Subsurface Scattering)
- BRDF meist für RGB
- Falls Oberfläche nicht homogen: SVBRDF (d.h. eine BRDF pro Oberflächenpunkt)
- Erweiterung auf Subsurface Scattering: BSSRDF
- Erweiterung auf „untere Halbkugel“ (Transmission): BRTF

Positivität:

$$f_r(\vec{l}, \vec{v}) \geq 0$$

Helmholtz Umkehrbarkeit:

$$f_r(\vec{l}, \vec{v}) = f_r(\vec{v}, \vec{l})$$

Energieerhaltung:

$$\int_{\Omega} f_r(\vec{l}, \vec{v}) \cdot \cos(\theta_{out}) \cdot d\vec{\omega}_{out} \leq 1$$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [212]

© R. Dörner

212



BRDF Messung

Messung einer BRDF (hohe Datenmengen)



Quelle: J. Dupuy, W. Jakob: An Adaptive Parameterization for Material Acquisition and Rendering

Gonioreflectometer



Wikipedia CC BY-SA 4.0

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [213]

© R. Dörner

213

Reflektionsgleichung

$$L_{in}(\vec{c}, -\vec{v}) = L_{out}(\vec{p}, \vec{v})$$

Position der Kamera

Punkt auf Oberfläche

Blickrichtung zur Kamera

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [214]

© R. Dörner

214



Reflektionsgleichung

Position der
Kamera

Punkt auf
Oberfläche

Blickrichtung
zur Kamera

$$L_{in}(\vec{c}, -\vec{v}) = L_{out}(\vec{p}, \vec{v})$$

$$= \int_{\vec{l} \in \Omega} f_r(\vec{l}, \vec{v}) \cdot L_{in}(\vec{p}, \vec{l}) \cdot \langle \vec{n}, \vec{l} \rangle \cdot d\vec{l}$$

Alle Raumwinkel einer
Halbkugel um Punkt P

BRDF

Raumwinkel



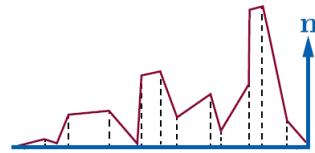
Vereinfachungen der Reflektionsgleichung mit BRDF

- Lambert
- Phong und Phong/Blinn
- Lafortune
 - Phong + mehrere Glanzlichter



Vereinfachungen der Reflektionsgleichung mit BRDF

- Lambert
- Phong und Phong/Blinn
- Lafortune
 - Phong + mehrere Glanzlichter
- Torrance-Sparrow
 - Basiert auf Mikrofacetten
 - Fresnel Spiegelung (genähert nach Schlick)
- Cook-Torrance
 - Berücksichtigung von Farbverschiebung
- Ward
 - Stochastische Modellierung Mikrofacetten auf Basis Gauß-Verteilung
- Oren-Nayar
 - Diffuse statt spiegelnde Mikrofacetten
 - Lichtinterferenzen
- Ashikhmin-Shirley
 - Berücksichtigt Subsurface Scattering
- ...



Microfacetten

Quelle: realtimerendering.com

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [217]

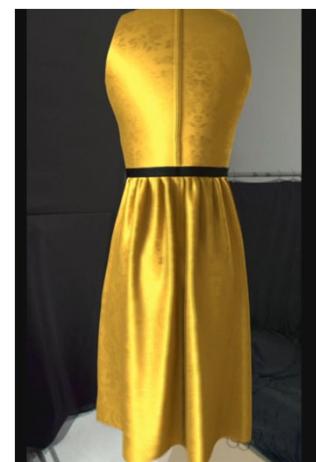


© R. Dörner

217

Vereinfachungen der Reflektionsgleichung mit BRDF

- Lambert
- Phong und Phong/Blinn
- Lafortune
 - Phong + mehrere Glanzlichter
- Torrance-Sparrow
 - Basiert auf Mikrofacetten
 - Fresnel Spiegelung (genähert nach Schlick)
- Cook-Torrance
 - Berücksichtigung von Farbverschiebung
- Ward
 - Stochastische Modellierung Mikrofacetten auf Basis Gauß-Verteilung
- Oren-Nayar
 - Diffuse statt spiegelnde Mikrofacetten
 - Lichtinterferenzen
- Ashikhmin-Shirley
 - Berücksichtigt Subsurface Scattering
- ...



Computergraphik – Prof. Dr. R. Dörner – WS 20

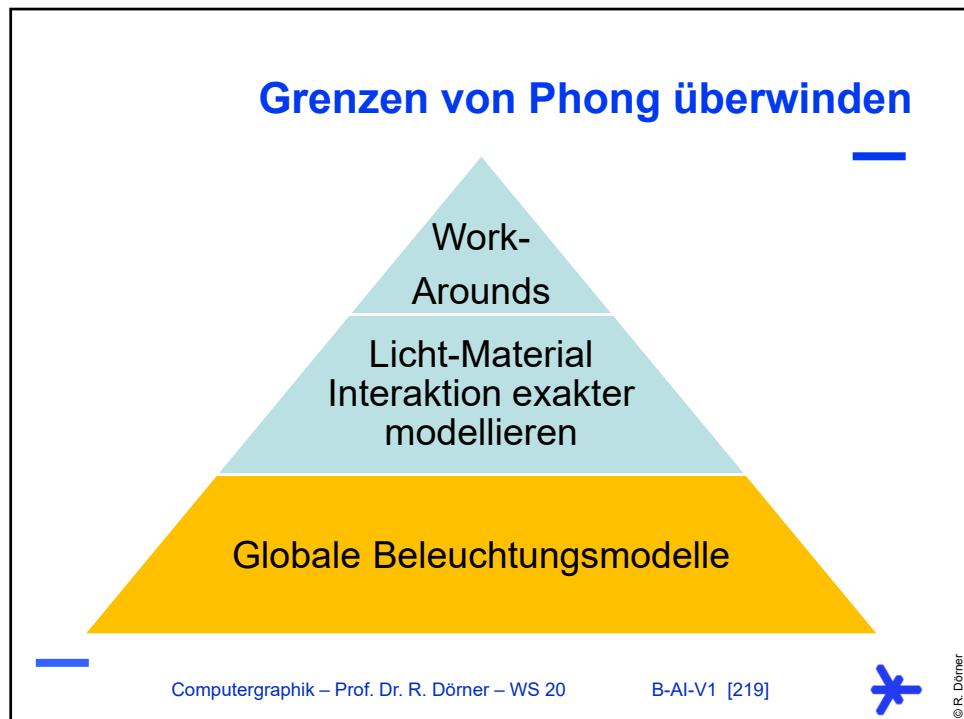
B-AI-V1 [218]



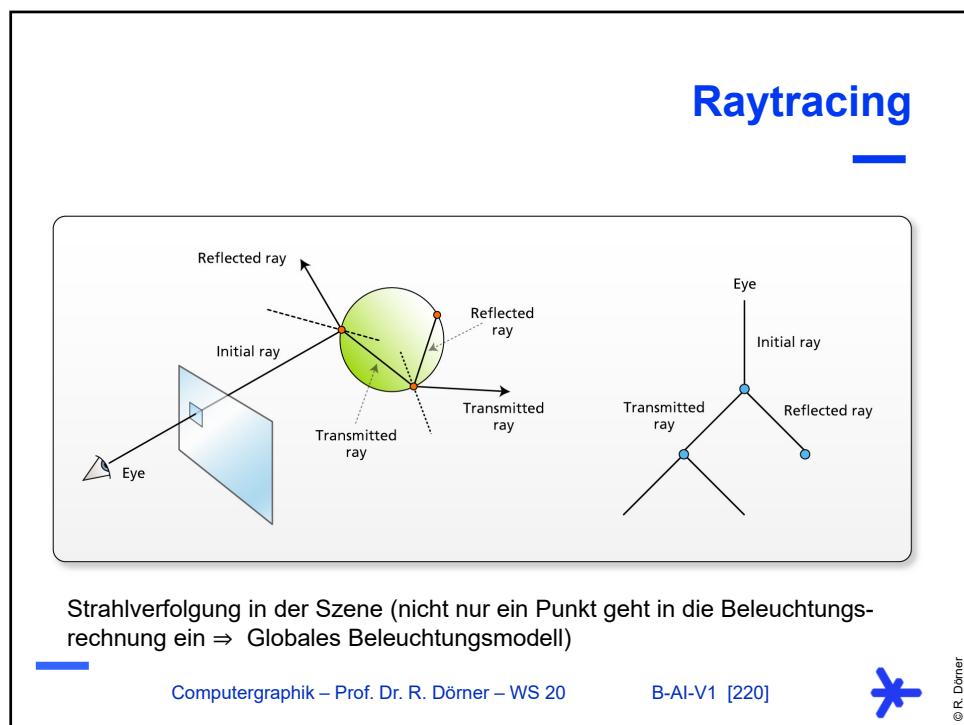
© R. Dörner

218





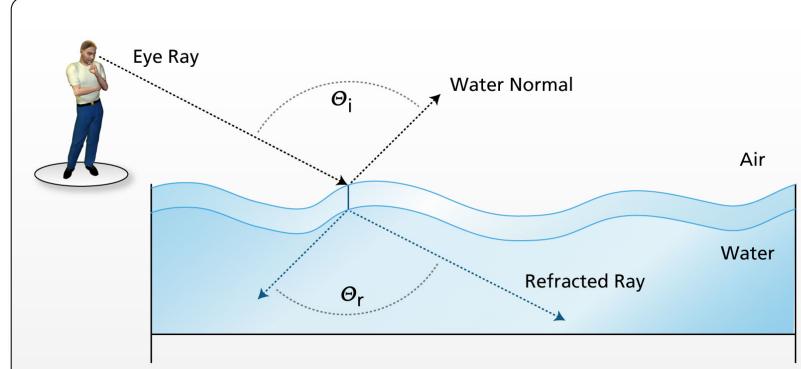
219



220



Raytracing mit Refraktion (Lichtbrechung)



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [221]



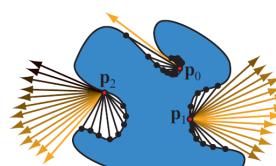
© R. Dörner

221

Ambient Occlusion

Verdeckung beeinflusst gegenseitige
Beleuchtung

Ambient Occlusion nutzt Information
über Sichtbarkeit via Raytracing (kann
vorberechnet werden)



Quelle: realtimerendering.com



Ambientes Licht berechnet mit: (a) Phong (b) Ambient Occlusion

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [222]



© R. Dörner

222



Renderinggleichung

- Traditionelles Raytracing hat Grenzen, z.B. bei Halbschatten
- Erweiterung der Reflektionsgleichung zur Renderinggleichung

$$L_{out}(\vec{p}, \vec{v}) = \int_{\vec{l} \in \Omega} f_r(\vec{l}, \vec{v}) \cdot L_{in}(\vec{p}, \vec{l}) \cdot \langle \vec{n}, \vec{l} \rangle \cdot d\vec{l}$$



$$L_{out}(\vec{p}, \vec{v}) = L_e(\vec{p}, \vec{v}) + \int_{\vec{l} \in \Omega} f_r(\vec{l}, \vec{v}) \cdot L_{out}(\vec{p}', -\vec{l}) \cdot \langle \vec{n}, -\vec{l} \rangle \cdot d\vec{l}$$

Emittiertes
Licht

Rekursion. \vec{p}' wird durch
Raytracing ermittelt

Renderinggleichung

- Traditionelles Raytracing hat Grenzen, z.B. bei Halbschatten
- Erweiterung der Reflektionsgleichung zur Renderinggleichung

$$L_{out}(\vec{p}, \vec{v}) = \int_{\vec{l} \in \Omega} f_r(\vec{l}, \vec{v}) \cdot L_{in}(\vec{p}, \vec{l}) \cdot \langle \vec{n}, \vec{l} \rangle \cdot d\vec{l}$$



$$L_{out}(\vec{p}, \vec{v}) = L_e(\vec{p}, \vec{v}) + \int_{\vec{l} \in \Omega} f_r(\vec{l}, \vec{v}) \cdot L_{out}(\vec{p}', -\vec{l}) \cdot \langle \vec{n}, -\vec{l} \rangle \cdot d\vec{l}$$

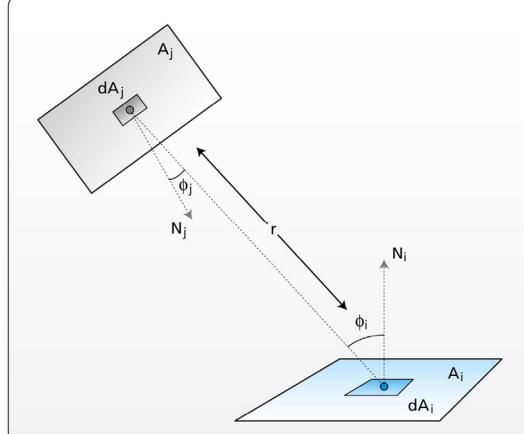
Zwei Punkte: Globale Beleuchtung



Lösung der Renderinggleichung

- Finite - Elemente - Methoden
 - Radiosity

Radiosity



Lichtausgleichsrechnung:

Tesselierung der Szene in
kleine Flächenelemente

Mathematische Beschreibung
der Lichtinteraktion zwischen
Flächenelementen

Lösung eines Gleichungs-
systems (Anzahl Gleichungen
= Anzahl Flächenelemente)



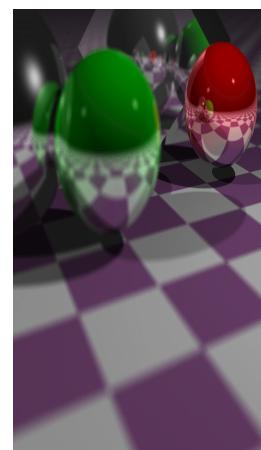
Lösung der Renderinggleichung

- Finite - Elemente - Methoden
 - Radiosity



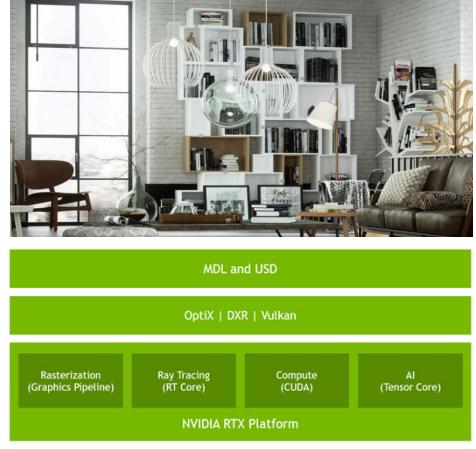
Lösung der Renderinggleichung (mit numerischen Methoden)

- Finite - Elemente - Methoden
 - Radiosity
- Monte – Carlo - Methoden
 - Path-Tracing
 - Viele Strahlen pro Pixel
 - An Schnittpunkten: stochastisches Sampling durch viele Strahlen, Nutzung BRDF
 - Photon Mapping
 - Photon Map: stochastisch Strahlen von Lichtquelle in Welt
 - Viele Strahlen vom Augpunkt, Nutzung Photon-Map
 - Distributed Raytracing
 - Bi-Direktionales Raytracing
 - ...



Trends

- Hardware Unterstützung beim Path-Tracing
 - Bsp. RTX von nVIDIA (Turing-Architektur)
- Schätzung von Beleuchtung aus Fotos mittels Bildverarbeitung und Nutzung für Beleuchtung (z.B. in AR)



Quelle: nVIDIA

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [229]

© R. Dörner

229

Überblick Beleuchtungsrechnung

```
graph TD; A[Beleuchtungsmodell] --> B[Lichtmodell]; A --> C[Materialmodell]; A --> D[Modell Interaktion Licht - Material]; C --> E[Shadingverfahren]; D --> F[Lokal]; D --> G[Global]; B --> H[Point]; B --> I[Spot]; B --> J[Directional]; B --> K[Area]; F --> L[BRDF]; L --> M[Lambert]; L --> N[Phong, Phong/Blinn]; L --> O[Cook-Torrance]; L --> P[Ward]; L --> Q[...]; G --> R[Raytracing]; R --> S[Amb. Occlusion]; G --> T[Finite Elemente]; T --> U[Radiosity]; G --> V[Renderinggleichung]; V --> W[Monte Carlo]; W --> X[Path Tracing]; W --> Y[Photon Map.]; W --> Z[...];
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [230]

© R. Dörner

230



—
**Teil E:
Szenenmodell**
—

231

Szenenmodell
—

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

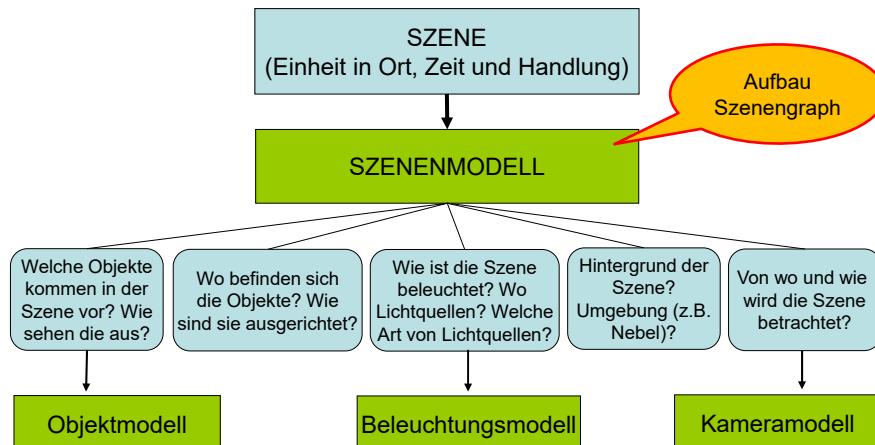
232



Szenenmodell

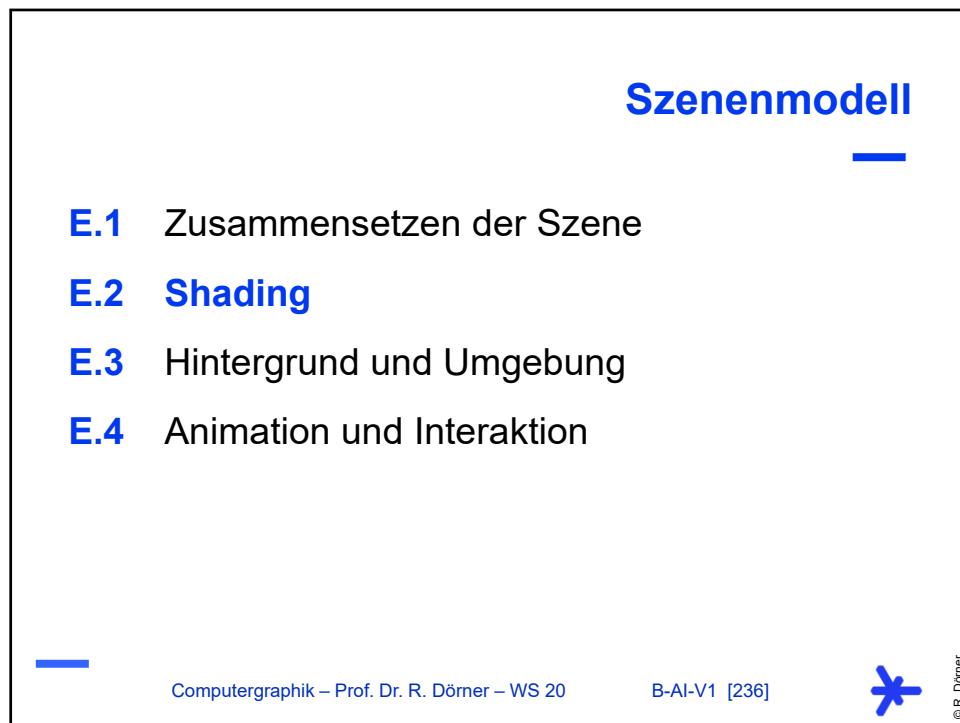
- E.1 Zusammensetzen der Szene**
- E.2 Shading**
- E.3 Hintergrund und Umgebung**
- E.4 Animation und Interaktion**

Szenenmodell





235



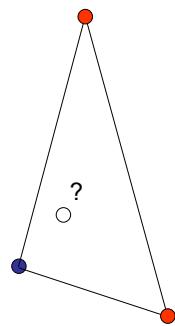
© R. Dörner

236

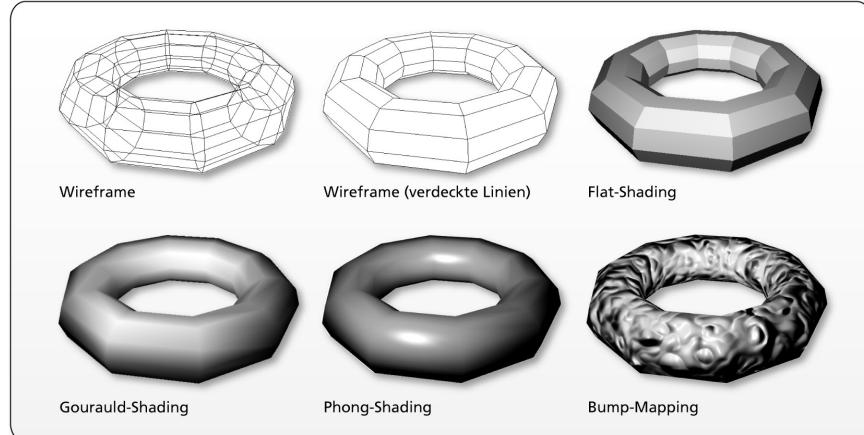


Shading

- 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



Shading: Beispiele



Flat Shading

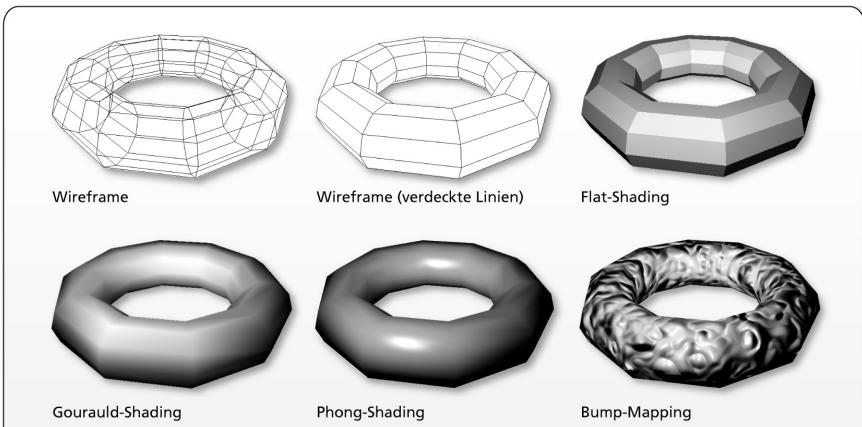
- Jede Fläche bekommt genau eine Farbe zugewiesen
- Welche Farbe? Verschiedene Möglichkeiten der Auswahl
 - z.B.: es wird die Farbe des Eckpunktes gewählt, der als Letzter in der Flächenliste steht
- Problem: Flat-Shading hebt den Fehler hervor, dass runde Objekte durch eckige Polygonnetze angenähert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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [239]

© R. Dörner

239

Shading: Beispiele



Wireframe

Wireframe (verdeckte Linien)

Flat-Shading

Gouraud-Shading

Phong-Shading

Bump-Mapping

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [240]

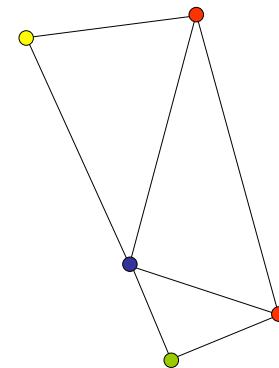
© R. Dörner

240



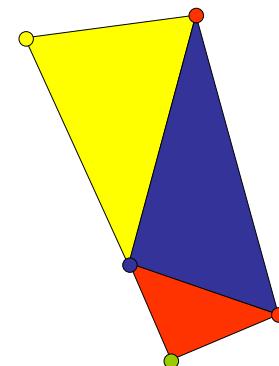
Gouraud-Shading

- 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)



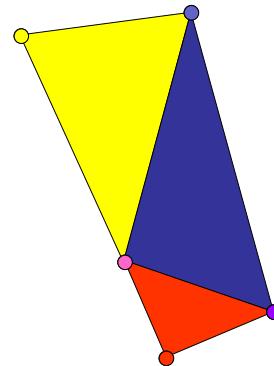
Gouraud-Shading

- 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 20

B-AI-V1 [243]



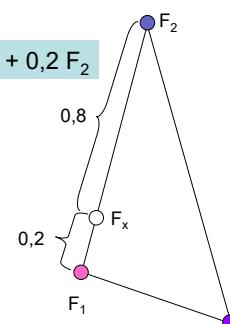
© R. Dörner

243

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 20

B-AI-V1 [244]



© R. Dörner

244



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 20 B-AI-V1 [245] © R. Dörner

245

Gouraud-Shading

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

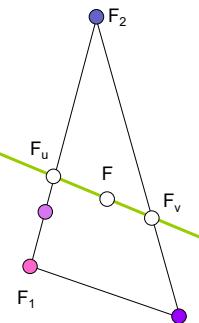
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [246] © R. Dörner

246



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

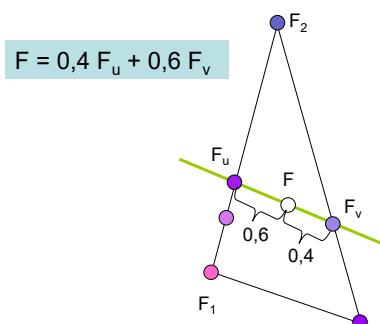


$$\text{Lineare Interpolation: } F = a F_u + b F_v \\ a + b = 1$$



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

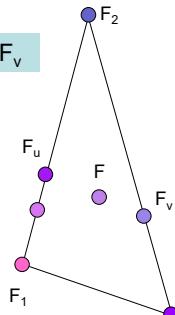


$$\text{Lineare Interpolation: } F = a F_u + b F_v \\ a + b = 1$$



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

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


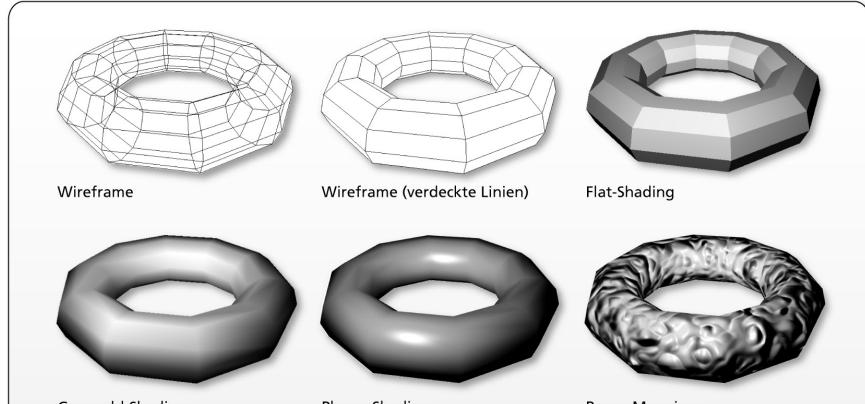
Lineare Interpolation:
 $F = a F_u + b F_v$
 $a + b = 1$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [249]

© R. Dörner

249

Shading: Beispiele



Wireframe

Wireframe (verdeckte Linien)

Flat-Shading

Gouraud-Shading

Phong-Shading

Bump-Mapping

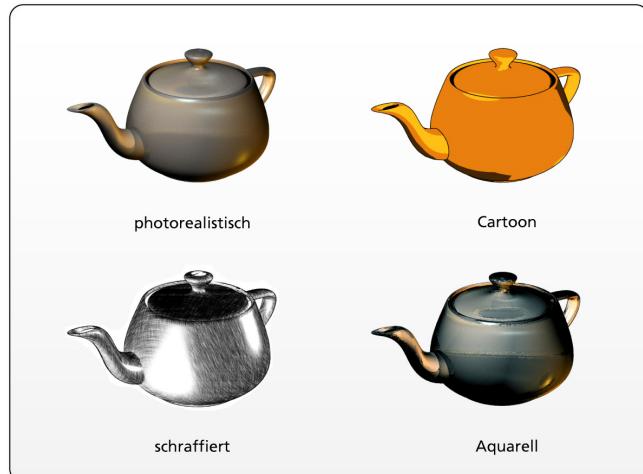
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [250]

© R. Dörner

250



Weitere Shading Verfahren



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [251]



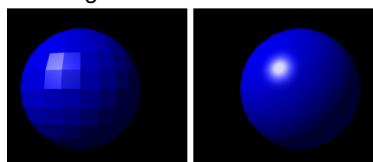
© R. Dörner

251

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 20

B-AI-V1 [252]



© R. Dörner

252



Szenenmodell

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



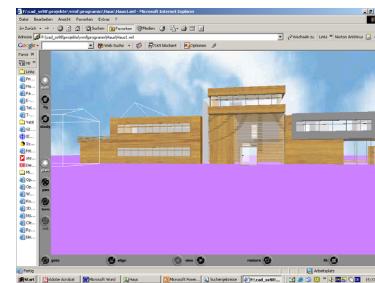
Hintergrund und Umgebung

- 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)



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 20

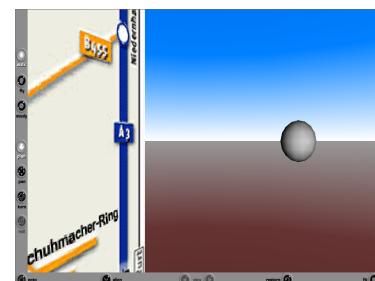
B-AI-V1 [255]



255

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 20

B-AI-V1 [256]



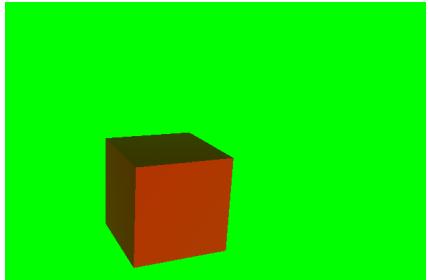
© R. Dörner

256



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
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [257]

© R. Dörner

257

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 20 B-AI-V1 [258]

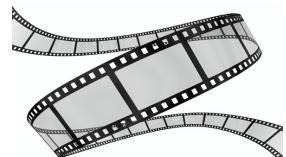
© R. Dörner

258



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 20

B-AI-V1 [259]



© R. Dörner

259

Animation in VRML

Uhr

TimeSensor Node
(liefert Wert aus [0,1])

Wert der über die Zeit verändert wird

Transform Node
translation 4 5 3

0.75

0 0 7

Verbindung über Routen

Beschreibung der Animation

PositionInterpolator Node

Tabelle: 0-> 1 0 0, 0.5-> 0 0 5, 1-> 0 0 9

Werte, die nicht in der Tabelle: Interpolation

Computergraphik – Prof. Dr. R. Dörner – WS 20

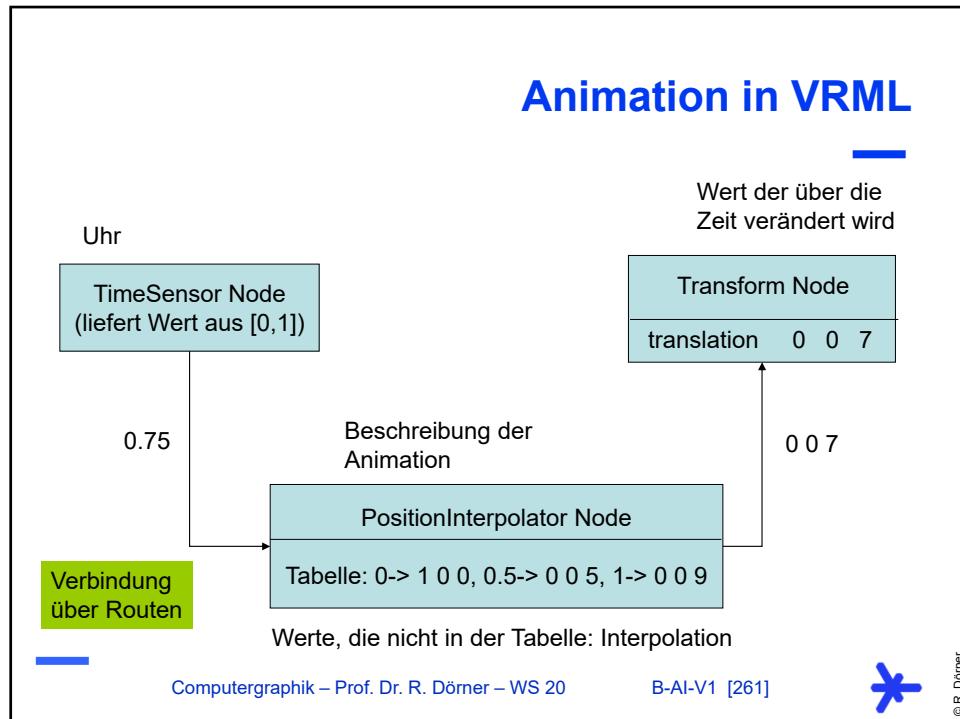
B-AI-V1 [260]



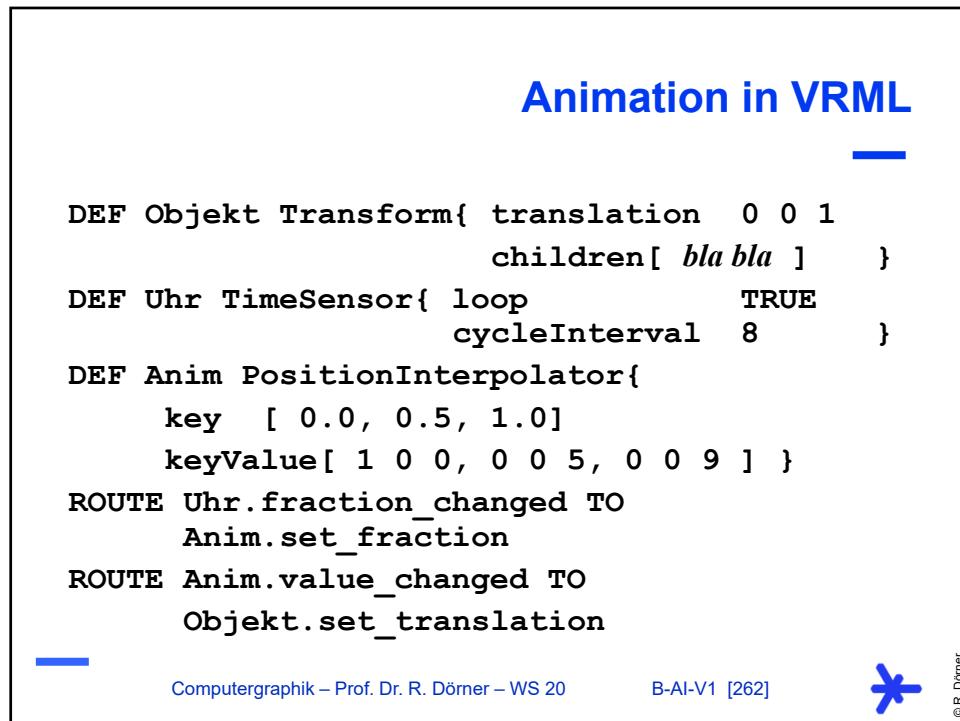
© R. Dörner

260





261



262



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 20

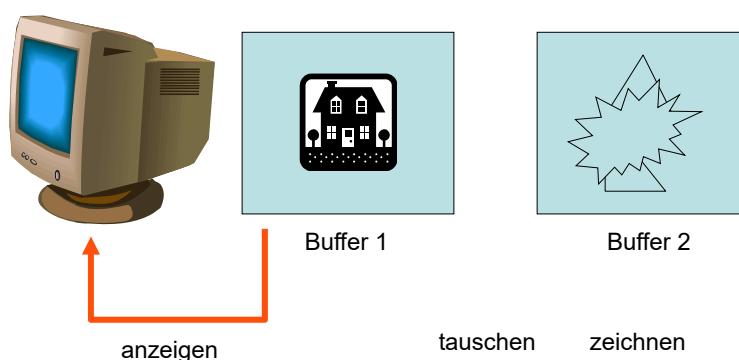
B-AI-V1 [263]



© R. Dörner

263

Double Buffering



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [264]



© R. Dörner

264



Begriff „Quaternion“



```
using UnityEngine;
using System.Collections;

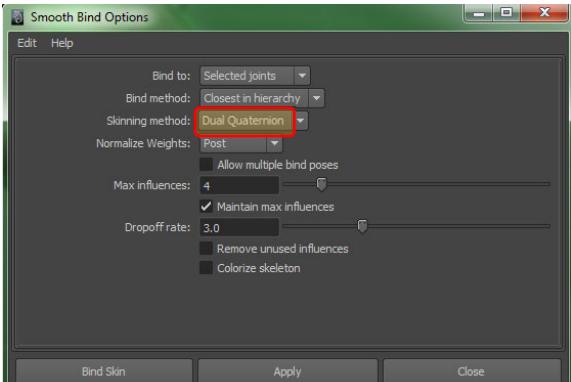
public class LookAtScript : MonoBehaviour
{
    public Transform target;

    void Update ()
    {
        Vector3 relativePos = target.position - transform.position;
        transform.rotation = Quaternion.LookRotation(relativePos);
    }
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [265]

265

Begriff „Quaternion“



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [266]

266



Begriff „Quaternion“

UNREAL ENGINE

FQuat

Variables

Name	Description
<code>F₀</code>	float <code>W</code> The quaternion's <code>w</code> -component
<code>F₁</code>	float <code>X</code> The quaternion's <code>x</code> -component
<code>F₂</code>	float <code>Y</code> The quaternion's <code>y</code> -component
<code>F₃</code>	float <code>Z</code> The quaternion's <code>z</code> -component

Functions

Name	Description
<code>f₄</code>	float <code>AngularDistance</code> (<code>float const FQuat & Q</code>) Find the angular distance between two rotation quaternions (in radians)
<code>f₅</code>	[CORE_API] (APITimeline)CoreMath(CORE_API_1) void <code>CalcTangents</code> (<code>const FQuat & PrevP, const FQuat & NextP, const FQuat & PrevR, const FQuat & NextR, const FPoint & PrevT, const FPoint & NextT</code>) Calculate tangents between given points
<code>f₆</code>	bool <code>ContainsNaN</code> Utility to check if there are any non-finite values (NaN or Inf) in this Quat.
<code>f₇</code>	void <code>DiagnoseNaN</code>

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [267]

267

3D Translation und 3D Rotation

- Freiheitsgrade (engl. Degrees of Freedom, DOF):
 - 3D Translation: 3 DOF
 - 3D Rotation: 3 DOF
- 3D Rotation
 - Kamera: pan, roll, tilt
 - Objekt: yaw, roll, pitch
gieren, rollen, nicken
- 3D Rotation kann durch drei Winkel spezifiziert werden

boom

truck

dolly

pan

roll

tilt

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [268]

268



Eulerwinkel

- 3 Winkel einer 3D Rotation heißen Eulerwinkel
- Angabe einer 3D Rotation durch ein 3-Tupel (α, β, γ)
- Beispiel: $(30^\circ, -10^\circ, 200^\circ)$
- Festlegung notwendig:
Welcher Winkel gehört zu
welcher Achse
 - wird durch Konvention
festgelegt
 - Problem: unterschiedliche
Konventionen (z.B. Luftfahrt z-y-
x vs. Computeranimation x-y-z)

Leonhard Euler 1707-1783

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [269]

269

Eulerwinkel

- 3 Winkel einer 3D Rotation heißen Eulerwinkel
- Angabe einer 3D Rotation durch ein 3-Tupel (α, β, γ)
- Beispiel: $(30^\circ, -10^\circ, 200^\circ)$
- Festlegung notwendig:
Welcher Winkel gehört zu
welcher Achse
 - wird durch Konvention
festgelegt
 - Problem: unterschiedliche
Konventionen (z.B. Luftfahrt z-y-
x vs. Computeranimation x-y-z)

Quelle: guerrilliacg.org

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [270]

270



Drehmatrizen

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

Eulerwinkel (α, β, γ)
Reihenfolge: 1. x , 2. y , 3. z

$$R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$$

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

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

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

Computergraphik – Prof. Dr. R. Dörner – WS 20
B-AI-V1 [271]

271

Probleme von Eulerwinkel

- Unterschiedliche Konventionen (Fehlerquelle)
- Fehlende Eindeutigkeit

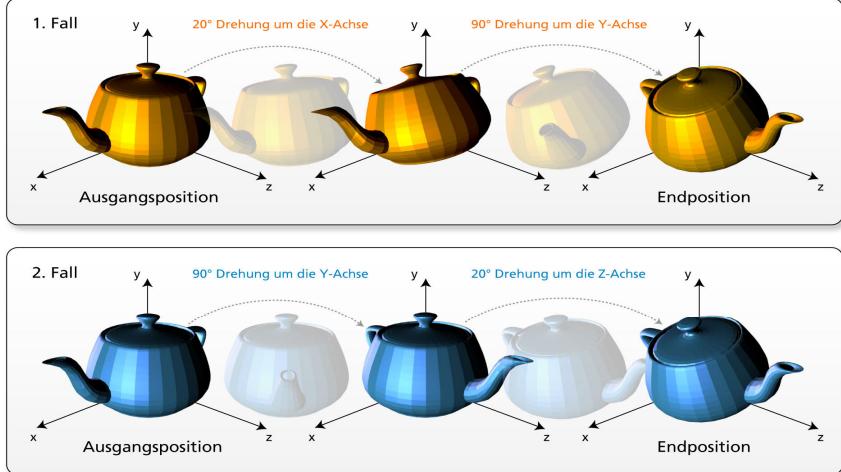


Computergraphik – Prof. Dr. R. Dörner – WS 20
B-AI-V1 [272]

272



Fehlende Eindeutigkeit



1. Fall

20° Drehung um die X-Achse

90° Drehung um die Y-Achse

Ausgangsposition

Endposition

2. Fall

90° Drehung um die Y-Achse

20° Drehung um die Z-Achse

Ausgangsposition

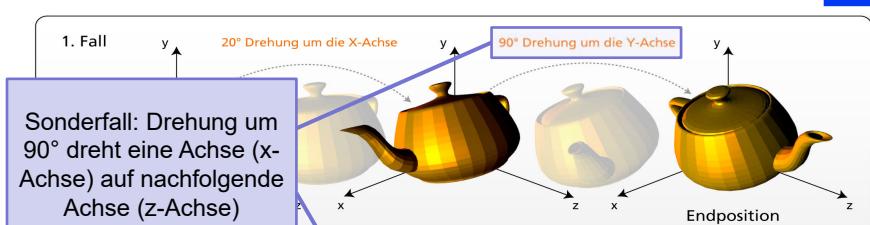
Endposition

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [273]

273

Fehlende Eindeutigkeit



1. Fall

20° Drehung um die X-Achse

90° Drehung um die Y-Achse

Sonderfall: Drehung um 90° dreht eine Achse (x-Achse) auf nachfolgende Achse (z-Achse)

Ausgangsposition

Endposition

2. Fall

90° Drehung um die Y-Achse

20° Drehung um die Z-Achse

Ausgangsposition

Endposition

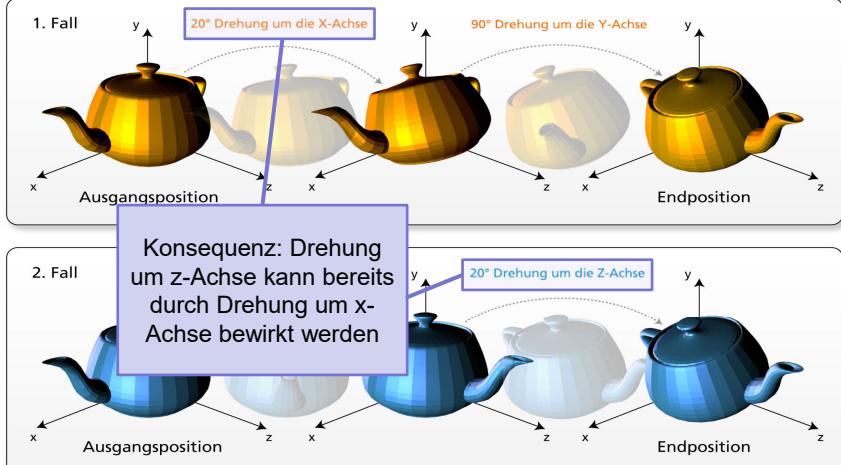
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [274]

274



Fehlende Eindeutigkeit



1. Fall

20° Drehung um die X-Achse

Ausgangsposition

90° Drehung um die Y-Achse

Endposition

2. Fall

20° Drehung um die Z-Achse

Ausgangsposition

Endposition

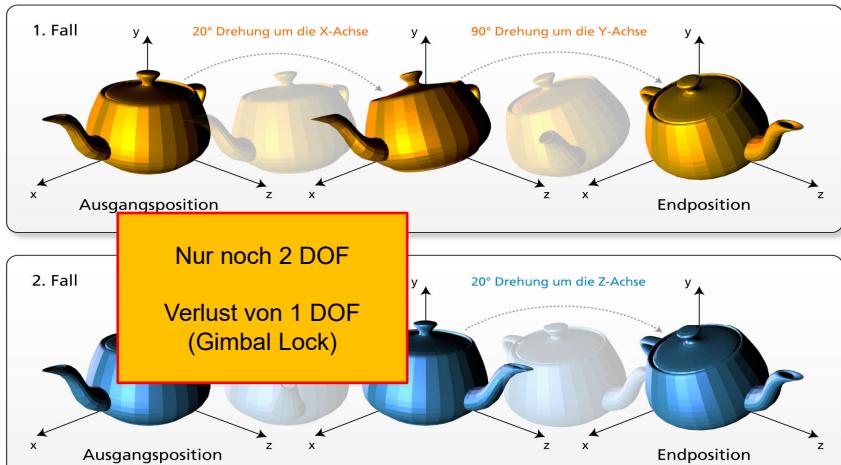
Konsequenz: Drehung um z-Achse kann bereits durch Drehung um x-Achse bewirkt werden

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [275]

275

Fehlende Eindeutigkeit



1. Fall

20° Drehung um die X-Achse

Ausgangsposition

90° Drehung um die Y-Achse

Endposition

2. Fall

20° Drehung um die Z-Achse

Ausgangsposition

Endposition

Nur noch 2 DOF
Verlust von 1 DOF (Gimbal Lock)

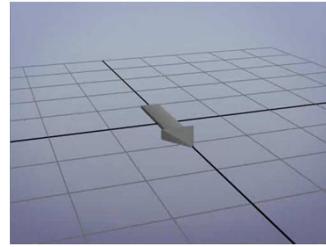
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [276]

276



Gimbal Lock



Quelle: gamedev.org

Nur noch 2 DOF
Verlust von 1 DOF
(Gimbal Lock)

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [277]

277

Probleme von Eulerwinkel

- Unterschiedliche Konventionen (Fehlerquelle)
- Fehlende Eindeutigkeit
- Verlust von DOF (Gimbal Lock)
- Keine Invarianz gehen Drehung der Drehachsen
- Schwierige Interpolation, z.B. bei Keyframing



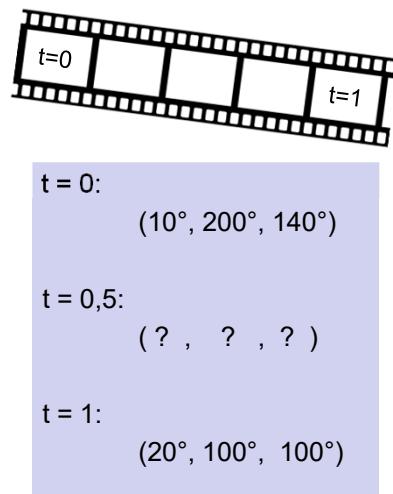
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [278]

278



Interpolation von Eulerwinkel

- Keyframing:
Basisverfahren in der Computeranimation
- Werte für $t = 0$ und $t = 1$ angeben, dazwischen automatisch interpolieren
- Winkel unabhängig voneinander interpolieren ergibt ggf. unerwünschte Resultate



Computergraphik – Prof. Dr. R. Dörner – WS 20

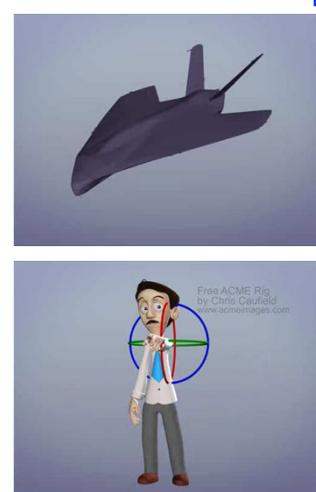
B-AI-V1 [279]



279

Interpolation von Eulerwinkel

- Keyframing:
Basisverfahren in der Computeranimation
- Werte für $t = 0$ und $t = 1$ angeben, dazwischen automatisch interpolieren
- Winkel unabhängig voneinander interpolieren ergibt ggf. unerwünschte Resultate



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [280]

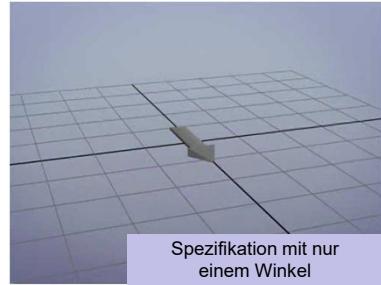


280



Alternative Spezifikation

- Spezifikation von 3D Rotationen mit nur einem Winkel
- Statt Eulerwinkel:
 - Drehwinkel
 - Richtung einer Drehachse (3D Vektor)
 - (Drehpunkt, Default: Ursprung)



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [281]



281

Spezifikation von 3D Rotationen

(α, β, γ)

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

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

$$\varphi, \vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [282]



282

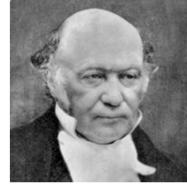


Verallgemeinerung komplexer Zahlen

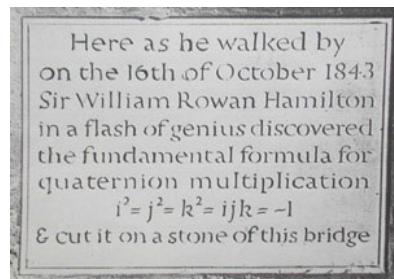
- Erweiterung der Zahlenebene (komplexe Zahlen) auf 4D
- Komplexe Zahlen mit einem Realteil und drei Imaginäranteilen i, j und k
- Hamilton-Zahlen \mathbb{H} (synonym: Quaternionen)
- Rechenregeln:

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$$

$$i \cdot j = k, i \cdot k = -j, j \cdot k = i$$



William R. Hamilton
1805 - 1865



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [283]



283

Rechenoperationen für Quaternionen

Schreibweise eines Quaternionen:

$$\hat{q} = (w, \begin{pmatrix} x \\ y \\ z \end{pmatrix}) = (w, \vec{r})$$

Addition von zwei Quaternionen:

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

Betrag eines Quaternionen:

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

Multiplikation von zwei Quaternionen:

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

Inverses eines Quaternionen:

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [284]



284



Rechenregeln für Quaternionen

- Quaternionen haben algebraische Struktur eines Schiefkörpers, d.h. „beinahe“ Körper, aber Multiplikation nicht kommutativ
- Es gilt u.a.
 - Assoziativgesetz
 - Distributivgesetz
 - Bilinearität
 - Existenz eines Inversen (außer für Nullelement)

$$\exists \hat{q}_1, \hat{q}_2 \in \mathbb{H}: \hat{q}_1 \cdot \hat{q}_2 \neq \hat{q}_2 \cdot \hat{q}_1$$

$$(\hat{q}_1 \cdot \hat{q}_2) \cdot \hat{q}_3 = \hat{q}_1 \cdot (\hat{q}_2 \cdot \hat{q}_3)$$

$$(\hat{q}_1 + \hat{q}_2) \cdot \hat{q}_3 = \hat{q}_1 \cdot \hat{q}_3 + \hat{q}_2 \cdot \hat{q}_3$$

$$\hat{q}_1 \cdot (\hat{q}_2 + \hat{q}_3) = \hat{q}_1 \cdot \hat{q}_2 + \hat{q}_1 \cdot \hat{q}_3$$



Einheitsquaternion und 3D Rotation

- Einheitsquaternionen haben Länge 1
- Einheitsquaternionen repräsentieren eine 3D Rotation

$$\hat{q} = (\cos \varphi, \sin \varphi \cdot \vec{r}) \text{ mit } |\vec{r}| = 1$$

$$\Rightarrow |\hat{q}| = 1$$

denn

$$|\hat{q}| = \sqrt{\cos^2 \varphi + \sin^2 \varphi \cdot \vec{r} \cdot \vec{r}}$$

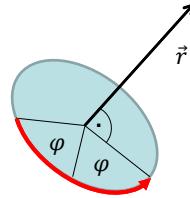
$$= \sqrt{\cos^2 \varphi + \sin^2 \varphi \cdot 1} = 1$$

$\hat{q} = (\cos \varphi, \sin \varphi \cdot \vec{r})$ repräsentiert eine Drehung um $2 \cdot \varphi$ um die Drehachse \vec{r} falls $|\vec{r}| = 1$



Einheitsquaternion und 3D Rotation

- Einheitsquaternionen haben Länge 1
- Einheitsquaternionen repräsentieren eine 3D Rotation
- Inverses eines Einheitsquaternionen: einfach Imaginärteile negieren
- Produkt zweier Einheitsquaternionen sowie Inverses ist wieder Einheitsquaternion, also: Einheitsquaternionen bilden eine (Lie-) Gruppe
- Einheitsquaternionen können als die Einheits-3-Sphäre im 4D euklidischen Raum aufgefasst werden



$\hat{q} = (\cos\varphi, \sin\varphi \cdot \vec{r})$ repräsentiert eine Drehung um $2 \cdot \varphi$ um die Drehachse \vec{r} falls $|\vec{r}| = 1$



Berechnung von 3D Rotationen

- Wir wollen Punkt $P(x, y, z)$ rotieren um Achse $\vec{r} = (u, v, w)^T$ um Winkel φ . Gesucht ist $P'(x', y', z')$
- Wir definieren folgende Quaternionen:

$$\hat{p} = (0, \begin{pmatrix} x \\ y \\ z \end{pmatrix}) \quad \hat{e} = (\cos \frac{\varphi}{2}, \frac{1}{\sqrt{u^2+v^2+w^2}} \cdot \sin \frac{\varphi}{2} \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix})$$

- Wir berechnen

$$\hat{q} = \hat{e} \cdot \hat{p} \cdot \hat{e}^{-1} = (0, \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix})$$



Konkatenation

- Mehrere Rotationen hintereinander ausführen
- Einheitsquaternionen aufmultiplizieren
- Assoziativgesetz: mehrere Rotationen können unabhängig vom zu transformierenden Punkt in ein Einheitsquaternion zusammengefasst werden

$$\hat{q} = \hat{e} \cdot \hat{p} \cdot \hat{e}^{-1}$$

$$\begin{aligned}\hat{q} &= \hat{e}_n \cdot \dots \cdot \hat{e}_2 \cdot \hat{e}_1 \cdot \hat{p} \cdot \hat{e}_1^{-1} \cdot \hat{e}_2^{-1} \cdot \dots \cdot \hat{e}_n^{-1} \\ &= (\hat{e}_n \cdot \dots \cdot \hat{e}_2 \cdot \hat{e}_1) \cdot \hat{p} \cdot (\hat{e}_1^{-1} \cdot \hat{e}_2^{-1} \cdot \dots \cdot \hat{e}_n^{-1}) \\ &= (\hat{e}_n \cdot \dots \cdot \hat{e}_2 \cdot \hat{e}_1) \cdot \hat{p} \cdot (\hat{e}_n \cdot \dots \cdot \hat{e}_2 \cdot \hat{e}_1)^{-1} \\ &= \hat{r} \cdot \hat{p} \cdot \hat{r}^{-1} \quad \text{mit } \hat{r} = \hat{e}_n \cdot \dots \cdot \hat{e}_2 \cdot \hat{e}_1\end{aligned}$$



Matrixrepräsentation

- Computergraphik nutzt 4×4 Matrizen u.a. für Berechnung von Translation, Skalierung, perspektivische Verzerrung
- In Shader auf der GPU ist Quaternion kein Basisdatentyp (in GLSL gibt es `mat4`, `vec4`)
- Problem: Wie fügen sich Berechnungen mit Quaternionen ein?
- Lösung: Wirkung des Quaternions durch zugehörige Matrix beschreiben (Idee: bilineare Abbildung)

Sei $\hat{q} = (w, \begin{pmatrix} x \\ y \\ z \end{pmatrix})$ mit $|\hat{q}| = 1$,

dann ist die zu \hat{q} gehörige Matrix

$$M_{\hat{q}} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$



Matrixrepräsentation

Sei $\hat{q} = (w, \begin{pmatrix} x \\ y \\ z \end{pmatrix})$ mit $|\hat{q}| = 1$,

dann ist die zu \hat{q} gehörige Matrix

$$M_{\hat{q}} = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) & 0 \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) & 0 \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [291]

291

Extraktion Quaternion aus Drehmatrix

- Aus jeder Drehmatrix (orthonormale Matrix mit Determinante +1) lässt sich ein zugehöriges Quaternion extrahieren
- Idee des Algorithmus: Koeffizientenvergleich bzw. Vergleich der Spur von Drehmatrix und Matrix $M_{\hat{q}}$

```


/* Construct a unit quaternion from rotation matrix. Assumes matrix is
 * used to multiply column vectors on the left; viewer = mat void. Works
 * correctly for right-handed coordinate system and right-handed rotations.
 * Translational perspective component is ignored. */
Quat QT_FromMatrix(Quat mat)

/* This algorithm avoids near-zero dividers by looking for a large component
 * — first w, then x, y, or z. When the trace is greater than zero,
 * *|w| is greater than 1/2, which is as small as a largest component can be.
 * *|x|, *|y|, and *|z| are the largest diagonal entry corresponds to the largest of |w|,
 * *|x|, or |z|, one of which must be larger than |w|, and at least 1/2. */
Quat quat;
float tr, s;
tr = mat[X][X] + mat[Y][Y] + mat[Z][Z];
if (tr >= 0.0) {
    s = sqrt(tr + mat[0][0]);
    quat.w = s * 0.5;
    s = 0.5 / s;
    quat.x = (mat[Z][Y] - mat[Y][Z]) * s;
    quat.y = (mat[X][Z] - mat[Z][X]) * s;
    quat.z = (mat[Y][X] - mat[X][Y]) * s;
} else {
    int h = X;
    if (mat[X][X] > mat[X][X]) h = Y;
    if (mat[X][X] > mat[X][X]) h = Z;
    switch (h) {
#define caseMacro(x,y,z,I,J,K) \
        case I: \
            s = sqrt((mat[I][I] - (mat[J][J] + mat[K][K])) + mat[W][W]); \
            quat.w = s * 0.5; \
            s = 0.5 / s; \
            quat.x = (mat[I][J] + mat[J][I]) * s; \
            quat.y = (mat[K][I] + mat[I][K]) * s; \
            quat.z = (mat[K][J] - mat[J][K]) * s; \
        break; \
        caseMacro(x,y,z,X,Y,Z); \
        caseMacro(y,x,z,X,Y,Z); \
        caseMacro(z,x,y,X,Y,Z); \
#undef caseMacro
    }
    if (mat[W][W] < 1.0) {
        s = 1.0 / sqrt(mat[W][W]);
        quat.w *= s; quat.x *= s; quat.y *= s; quat.z *= s;
    }
    return (quat);
}


```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [292]

292

Quelle: Ken Shoemake

Quaternen in der Computergrafik

- 3D Rotation von Punkten um Drehachse und Winkel
- Finden einer Rotation, die eine Richtung in eine andere Richtung überführt
- Interpolation von Orientierungen auf Großkreisen (nicht linear, sondern sphärisch linear)
- Interaktionstechniken für die Rotation (z.B. ARCBALL)
- Realisierung des Skinnings bei einer Character Animation
- Realisierung von Watermarking in Bildern
- KI für menschliche Bewegung
- ...

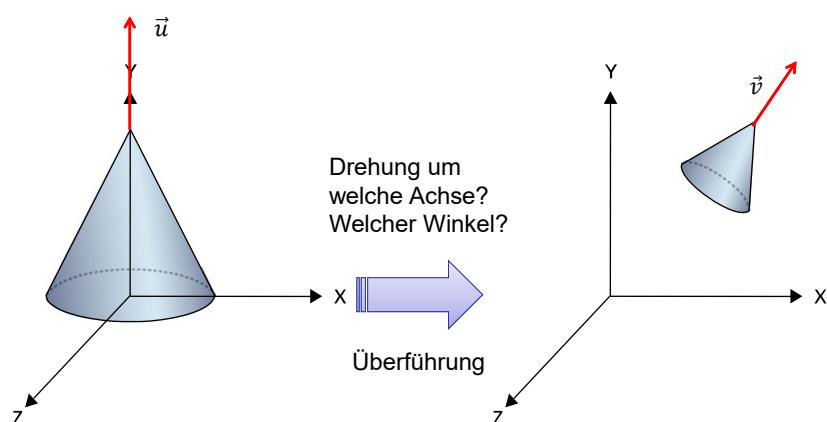
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [293]



293

Überführung von Richtungen



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [294]



294



Überführung von Richtungen

Gegeben sind zwei Vektoren \vec{u} und \vec{v} . Gesucht ist Quaternion \hat{q} , das eine Rotation beschreibt, die \vec{u} in dieselbe Richtung wie \vec{v} zeigen lässt.

1.) Normalisiere \vec{u} , \vec{v} (Division durch den Betrag von \vec{u} bzw. \vec{v})

2.) Berechne $\vec{r} = \frac{\vec{u} \times \vec{v}}{|\vec{u} \times \vec{v}|}$

3.) Sei 2φ der Winkel zwischen \vec{u} und \vec{v} .

Dann gilt: $k := \langle \vec{u}, \vec{v} \rangle = \cos 2\varphi$ und $|\vec{u} \times \vec{v}| = \sin 2\varphi$

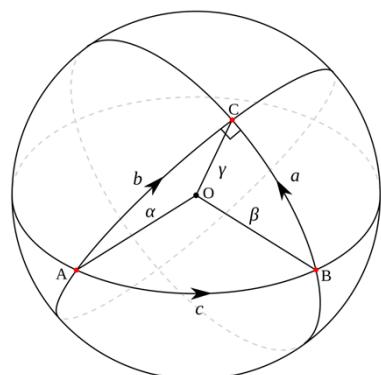
$$4.) \hat{q} = (\cos \varphi, \sin \varphi \cdot \vec{r}) \\ = \left(\frac{\sqrt{2(1+k)}}{2}, \frac{1}{\sqrt{2(1+k)}} \cdot (\vec{u} \times \vec{v}) \right)$$

Sonderfall: $k = -1$,
d.h. beide Vektoren zeigen
in entgegengesetzte Rich-
tung



Sphärische lineare Interpolation

- Interpolation von Orientierungen zwischen zwei Keyframes
- Bei Einheitsquaterniionen: gesucht wird kürzeste Verbindung (Geodäte) zwischen zwei Punkten auf einer 4D Kugeloberfläche (Teil eines Großkreises)

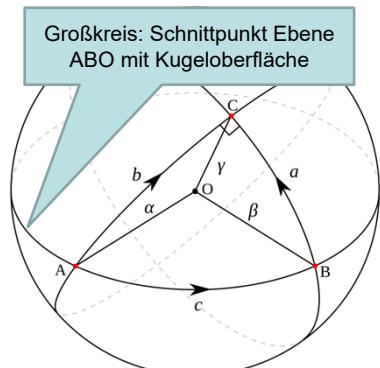


By derivative work: Pbros13 (talk) RechtwKugeldreieck.svg: Traced by User: Stannered from a PNG by en:User:Ric007 - RechtwKugeldreieck.svg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=4375381>



Sphärische lineare Interpolation

- Interpolation von Orientierungen zwischen zwei Keyframes
- Bei Einheitsquaterniomen: gesucht wird kürzeste Verbindung (Geodäte) zwischen zwei Punkten auf einer 4D Kugeloberfläche (Teil eines Großkreises)

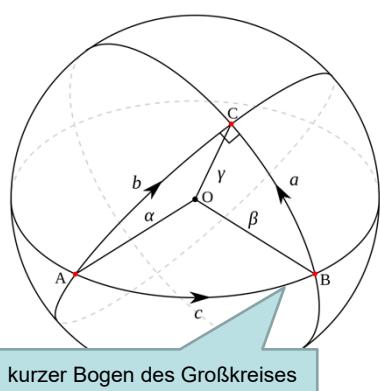


By derivative work: Pbrosks13 (talk) RechtwKugeldreieck.svg: Traced by User: Stannered from a PNG by en:User:R1661 - RechtwKugeldreieck.svg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=4375381>



Sphärische lineare Interpolation

- Interpolation von Orientierungen zwischen zwei Keyframes
- Bei Einheitsquaterniomen: gesucht wird kürzeste Verbindung (Geodäte) zwischen zwei Punkten auf einer 4D Kugeloberfläche (Teil eines Großkreises)
- Vorteil: Interpolation mit konstanter Geschwindigkeit
- Ken Shoemake (1985): erste Anwendung von Quaterniomen in der Computergrafik



By derivative work: Pbrosks13 (talk) RechtwKugeldreieck.svg: Traced by User: Stannered from a PNG by en:User:R1661 - RechtwKugeldreieck.svg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=4375381>



Sphärische lineare Interpolation

Gegeben: $\hat{q}_0 = (w_0, \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix})$, $\hat{q}_1 = (w_1, \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix})$, $t \in [0,1]$, mit $|\hat{q}_0| = |\hat{q}_1| = 1$

Gesucht: $\hat{q}_t = \text{slerp}(\hat{q}_0, \hat{q}_1, t)$

$$k = w_0 \cdot w_1 + x_0 \cdot x_1 + y_0 \cdot y_1 + z_0 \cdot z_1$$

$$\varphi = \arccos(k)$$

$$\hat{q}_t = \frac{\sin(\varphi \cdot (1-t))}{\sin \varphi} \cdot \hat{q}_0 + \frac{\sin(\varphi \cdot t)}{\sin \varphi} \cdot \hat{q}_1$$

Test, ob Interpolation entlang
kurzen Bogen:
falls $k < 0$, dann \hat{q}_1 ersetzen
durch:

$$(-w_1, \begin{pmatrix} -x_1 \\ -y_1 \\ -z_1 \end{pmatrix})$$

für Winkel nahe 0° (d.h. $k \approx 1$)
lineare Interpolation verwenden

Computergraphik – Prof. Dr. R. Dörner – WS 20

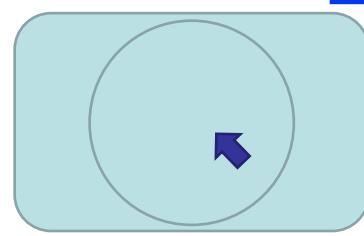
B-AI-V1 [299]



299

ARCBALL

- Idee: 3D Objekt / 3D Szene in einer Hüllkugel
- Dragging der Maus auf dem Bildschirm wird abgebildet auf eine Rotation der Hüllkugel und ihres Inhaltes



Quelle: <https://www.youtube.com/watch?v=YXNgjyv8W0I>

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [300]



300



ARCBALL

Gegeben: Screenkoordinate Mittelpunkt Kugel (m_x, m_y), Screenradius r ,
Screenkoordinate Start Dragging (s_x, s_y), Orientierung am Start \hat{q}_0
und aktuelle Mausposition auf dem Screen (t_x, t_y)

Berechne:

$$\vec{u} = \begin{pmatrix} (s_x - m_x)/r \\ (s_y - m_y)/r \\ 0 \end{pmatrix} \quad \vec{u}' = \vec{u} + \begin{pmatrix} 0 \\ 0 \\ \sqrt{1 - |\vec{u}|^2} \end{pmatrix}$$

$$\vec{v} = \begin{pmatrix} (t_x - m_x)/r \\ (t_y - m_y)/r \\ 0 \end{pmatrix} \quad \vec{v}' = \vec{v} + \begin{pmatrix} 0 \\ 0 \\ \sqrt{1 - |\vec{v}|^2} \end{pmatrix}$$

Drehe Objekt mit $\hat{q} = (\langle \vec{u}', \vec{v}' \rangle, \vec{u}' \times \vec{v}')$ $\cdot \hat{q}_0$

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [301]



301

Vorteile von Quaternionen

- Kompakte und einfache Repräsentation von 3D Rotationen
- Effiziente Implementierung (weniger Rechenoperationen als bei Eulerwinkel, keine trigonometrischen Funktionen)
- Keine Nachteile wie bei Eulerwinkel (z.B. kein Gimbal Lock)



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [302]

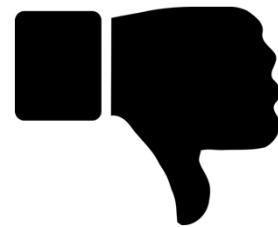


302



Nachteile von Quaternionen

- Bei Animation der Kamera wird unerwünschtes Rollen der Kamera eingefügt
- Unintuitiv: \hat{q} und $-\hat{q}$ repräsentieren die gleiche 3D Rotation (Flipping, Fallunterscheidungen notwendig)
- Rotation um 0° und um 360° (unterschiedlich!) werden durch gleiches Quaternion repräsentiert, da dieselbe Orientierung resultiert (mehrere Keyframes zur Unterscheidung notwendig)



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [303]

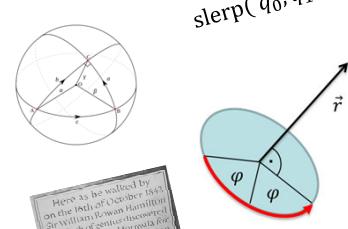


303

Quaternionen

- Quaternionen sind Zahlen (Verallgemeinerung komplexer Zahlen)
- Quaternionen beschreiben 3D Rotationen vorteilhaft
- Quaternionen führen zu effizienten Algorithmen
- Quaternionen haben zahlreiche Anwendungen (z.B. Berechnung 3D Rotation, sphärische lineare Interpolation) in der Computergrafik

$$\hat{q} = (w, \begin{pmatrix} x \\ y \\ z \end{pmatrix})$$



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [304]



304



—
**Teil F:
Objektmodelle**
—

305

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



306



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,
        0 0 0,
        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)

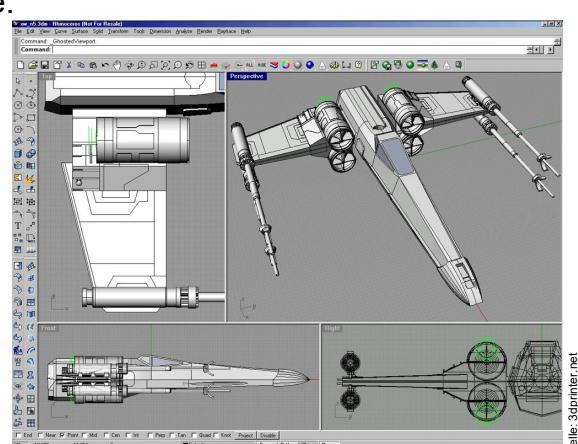


Erstellung von Objektmodellen

Modellierwerkzeuge:

Beispiele:

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



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)

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [311]

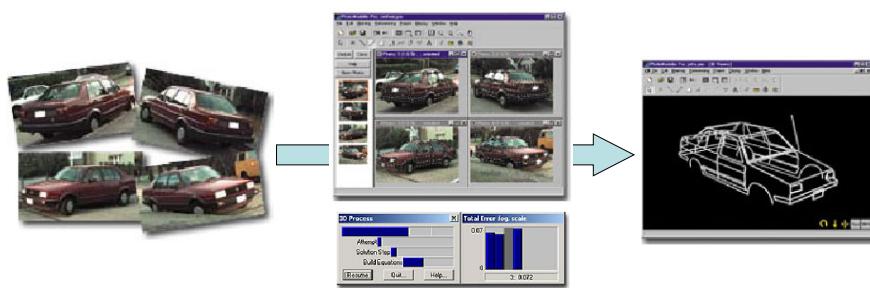


© R. Dörner

311

Erstellung von Objektmodellen

3D Rekonstruktion:



Quelle:
www.photomodeler.com

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [312]



© R. Dörner

312



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



Erstellung von Objektmodellen

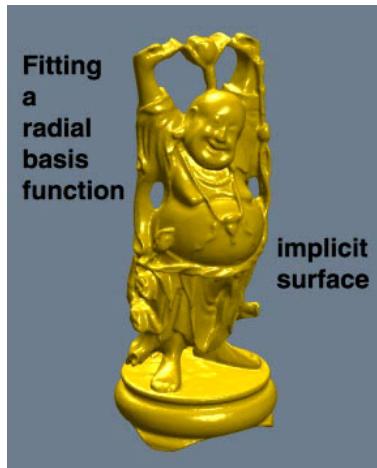
3D Scanner:



Quelle:
Konica Minolta
Europe GmbH



Erstellung von Objektmodellen



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

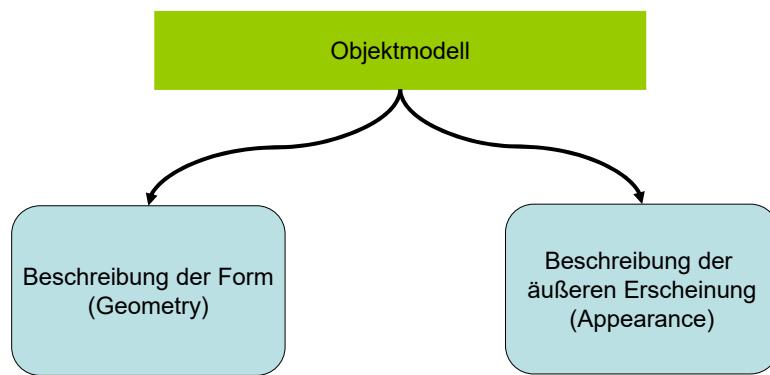
B-AI-V1 [315]



© R. Dörner

315

Umfang von Objektmodellen



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [316]



© R. Dörner

316



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)?
- } ⇒ Farbmodelle
⇒ Texturen
⇒ Parameter von Beleuchtungsmodellen, erweiterte Farbmodelle



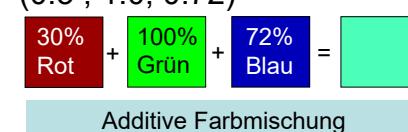
Objektmodelle

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



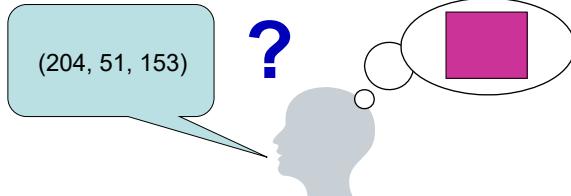
Farbmodelle

- 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:

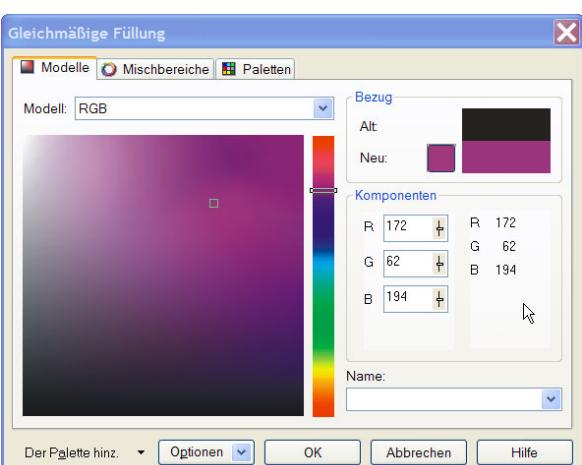


- RGB ist ein Geräte-Farbmodell
- Es gibt auch andere Farbmodelle, z.B. Wahrnehmungs-Farbmodelle, mit denen Menschen einfacher Farbe spezifizieren können

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [321]  © R. Dörner

321

Farbmodelle



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [322]  © R. Dörner

322



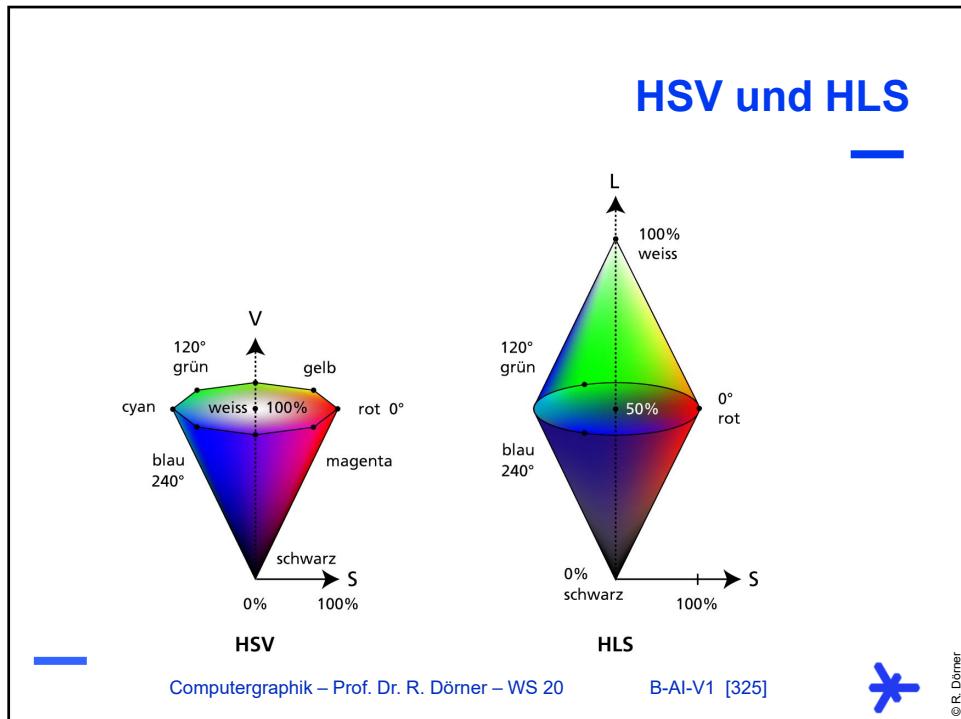
Farbmodelle

HLS System ist Wahrnehmungs-Farbmodell

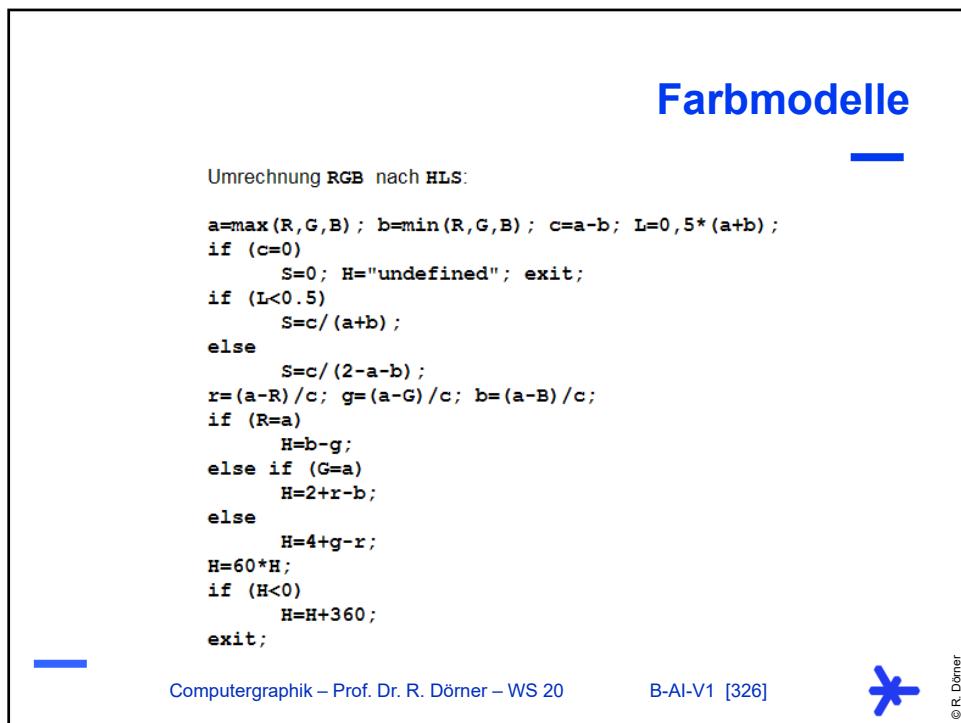
- Hue: Farbton, spezifiziert als Winkel im Farbkreis
- Lightness: Helligkeit, spezifiziert als Wert aus $[0,1]$
- Saturation: Sättigung, spezifiziert als Wert aus $[0,1]$

Farbmodelle

Farbkreise:



325



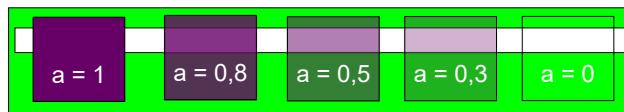
326



Erweiterte Farbmodelle

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

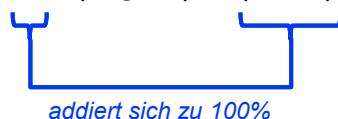
- Beispiel:



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):

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



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  
}
```

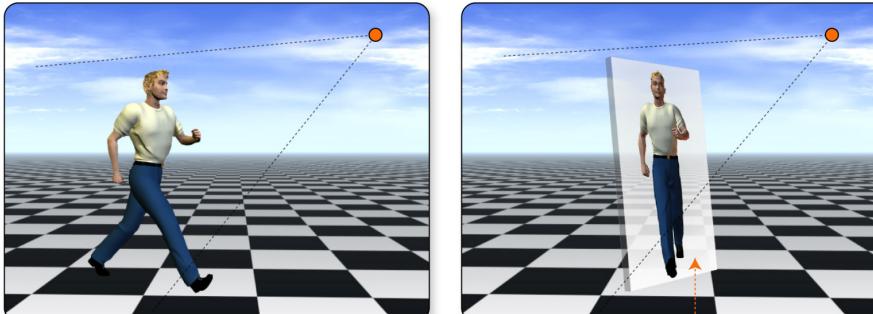


Texturen

- 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



Texturen: Beispiel Billboard



Projection plane through object

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [331]

© R. Dörner

331

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 20

B-AI-V1 [332]

© R. Dörner

332



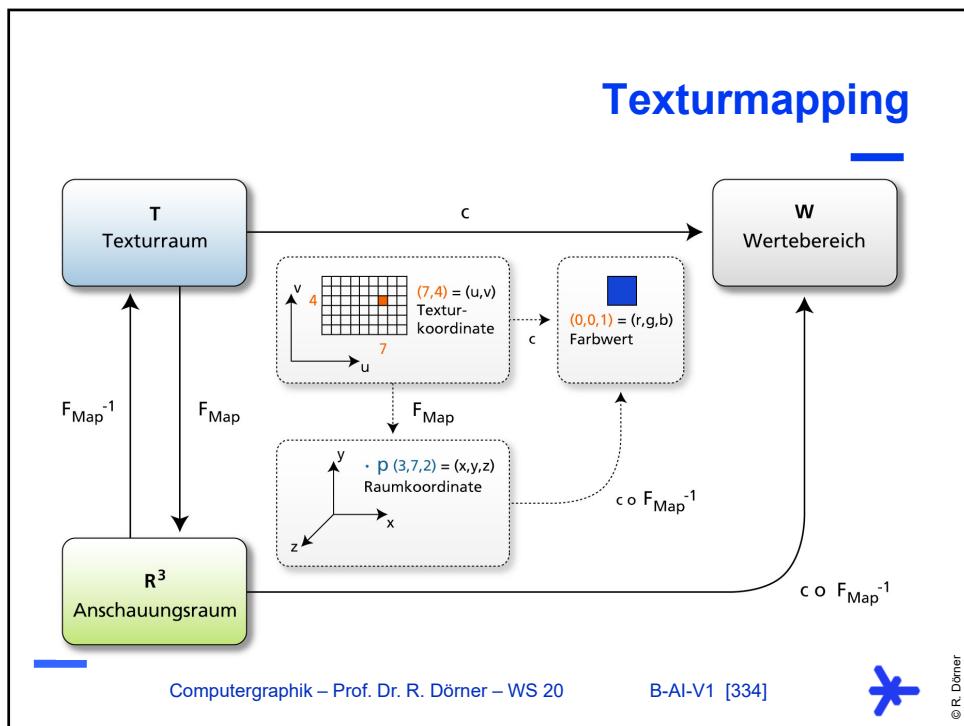
Texturmapping

- Gegeben
 - Textur (im Texturkoordinatensystem, üblich normiert auf $[0,1]$): jedem Wert in $[0,1]$ $x [0,1]$ 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?

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [333] © R. Dörner

333



334



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [335] © R. Dörner

335

Texturmapping

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [336] © R. Dörner

336



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

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

$P(x_0, y_0, z_0)$

u_0 v_0

Texture-Mapping

x y z

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [337]

© R. Dörner

337

Hüllkörper beim Texturmapping

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

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [338]

© R. Dörner

338



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 Texturmapping?
 - Angabe einer mathematischen Abbildungsvorschrift
 - Verwendung Hüllkörper
 - Angabe der Texturkoordinaten direkt pro (Eck-) Punkt im Sinne einer Wertetabelle

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

$P(x_0, y_0, z_0)$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [339]

© R. Dörner

339

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [340]

© R. Dörner

340



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:

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [341]

© R. Dörner

341

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

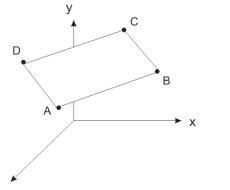
Resultierende Texturierung:

342



Vergrößerung des Texturraums

3D-Welt:



Texturraum:



Resultierende Texturierung:



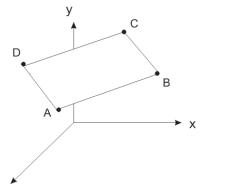
Punkt A: 1.5, 2.0
Punkt B: 0.0, 2.0
Punkt C: 0.0, 0.0
Punkt D: 1.5, 0.0

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [343]  © R. Dörner

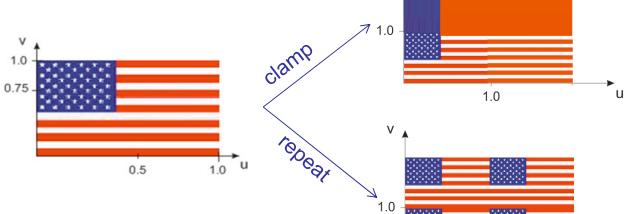
343

Vergrößerung des Texturraums

3D-Welt:



Texturraum:



Resultierende Texturierung:



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

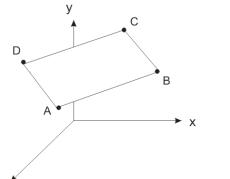
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [344]  © R. Dörner

344

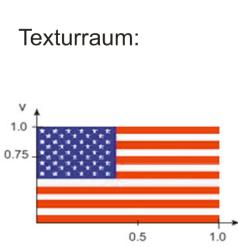


Vergrößerung des Texturraums

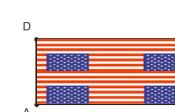
3D-Welt:



Texturraum:



Resultierende Texturierung:



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

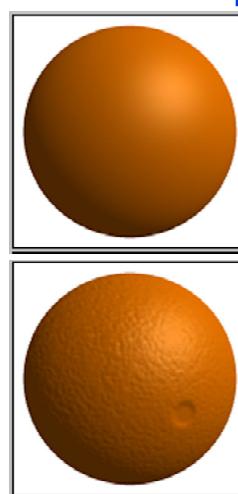
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [345]

© R. Dörner

345

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 20 B-AI-V1 [346]

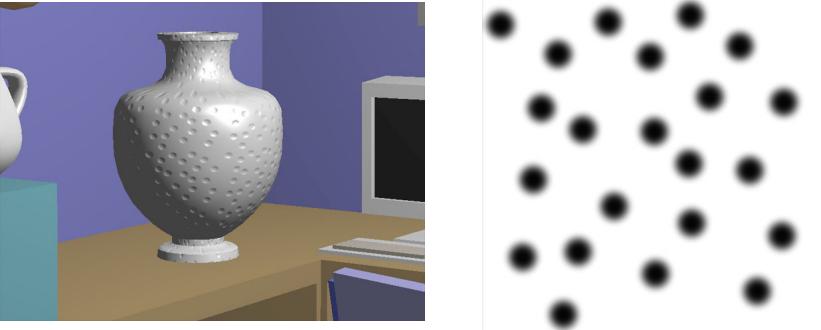
© R. Dörner

346



Bump Mapping

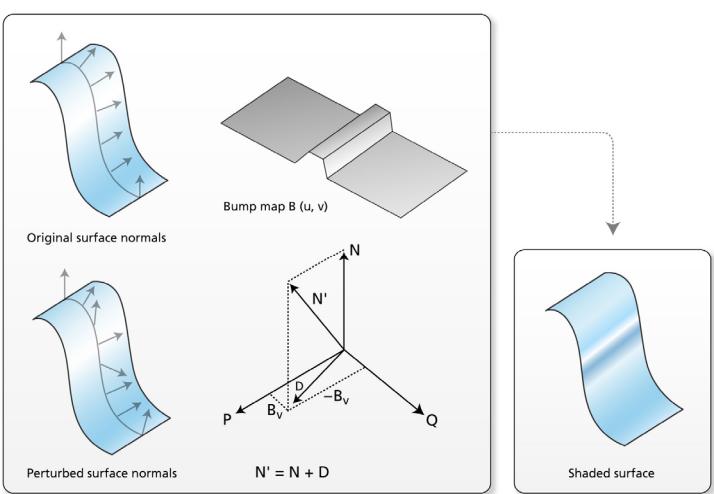
Quelle:
Watt / Polycarpo
3D Games



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [347] © R. Dörner

347

Bump Mapping



Original surface normals
Perturbed surface normals
Bump map $B(u, v)$
 $N' = N + D$
Shaded surface

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [348] © R. Dörner

348



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



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [349]

© R. Dörner

349

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [350]

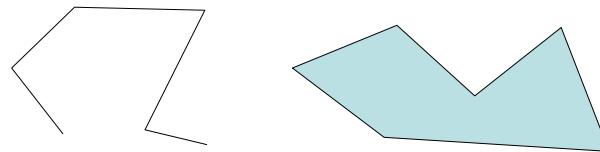
© R. Dörner

350



Polygone und Polygonzüge

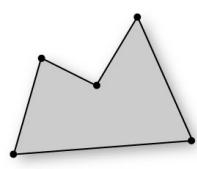
- Spezifiziert durch
 - Koordinaten der Eckpunkte
 - Angabe, welcher Punkt mit welchem verbunden ist
 - Beschreibung von Linien und Flächen



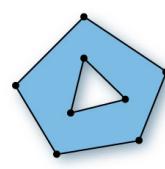
- Offene vs. geschlossene Polygone

Polygonflächen

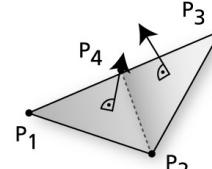
- Geschlossene Polygonzüge bilden Polygonflächen
- Sonderfälle:



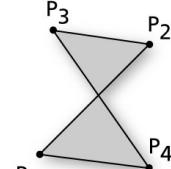
Fläche mit
"Einbuchtung"
(konkave Fläche)



Fläche mit "Loch"

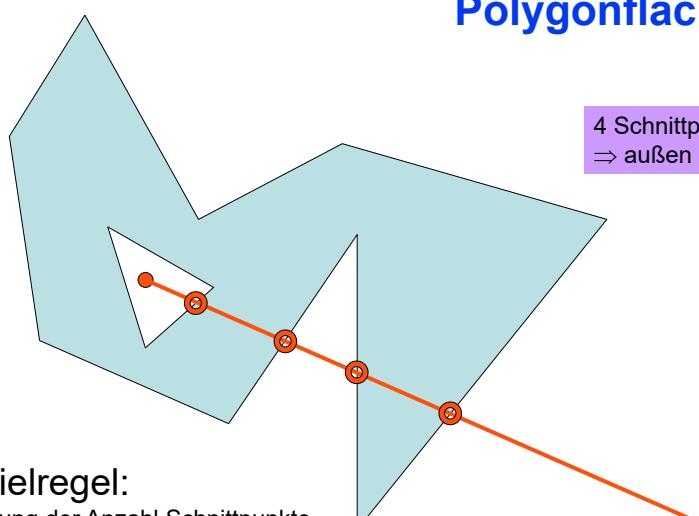


Polygon, das
nicht in einer
Ebene liegt



selbst-
schneidendes
Polygon

Polygonflächen



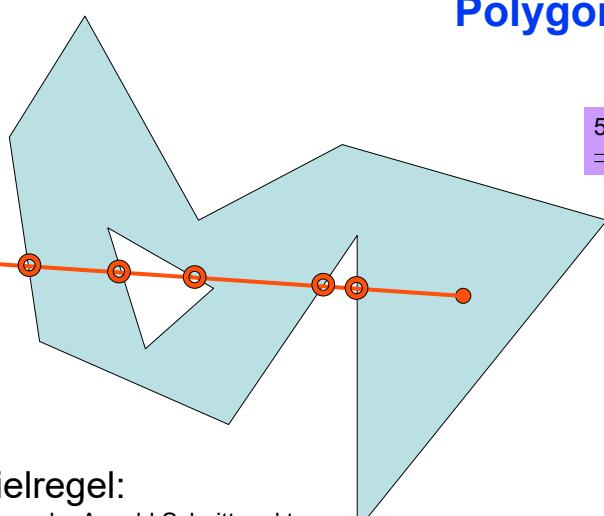
4 Schnittpunkte
⇒ außen

Beispielregel:
Bestimmung der Anzahl Schnittpunkte
mit dem Polygonzug

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [353] © R. Dörner

353

Polygonflächen



5 Schnittpunkte
⇒ innen

Beispielregel:
Bestimmung der Anzahl Schnittpunkte
mit dem Polygonzug

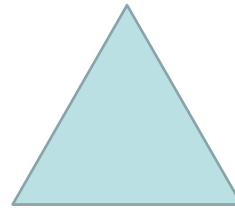
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [354] © R. Dörner

354



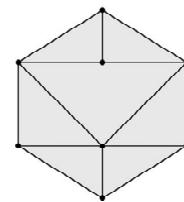
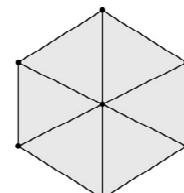
Polygonflächen

- 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)



Polygonflächen

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



Polygonflächen

Algorithmen zur Triangulierung

- Beispiel: DeLaunay
 - Basiert auf Voronoi Zerlegung: Verbindung von Punktenpaaren mit benachbarten Voronoi - Gebieten
 - Umkreis jeden Dreiecks enthält keine andere 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 20

B-AI-V1 [357]

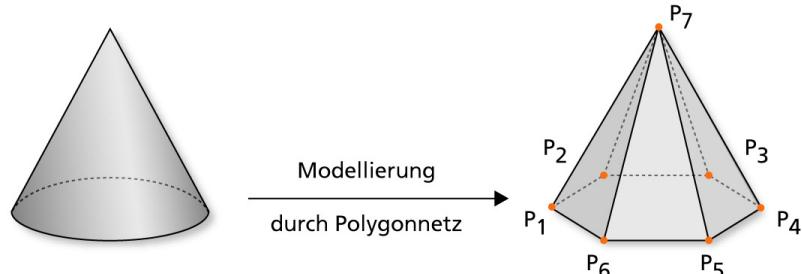


© R. Dörner

357

Näherungen

- Polygonnetze können manche Objekte nur Näherungsweise beschreiben



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [358]



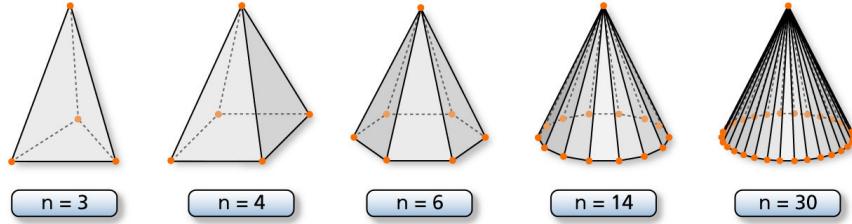
© R. Dörner

358



Näherungen

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



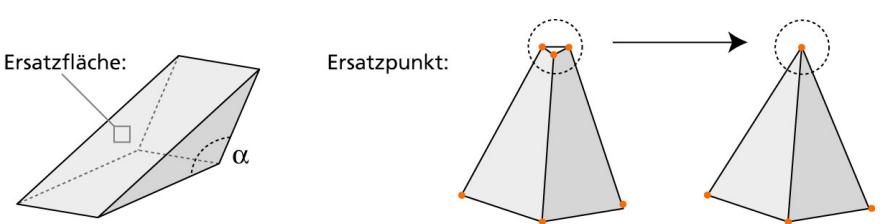
$n = 3$ $n = 4$ $n = 6$ $n = 14$ $n = 30$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [359]  © R. Dörner

359

Näherungen

- Polygonnetze können mit speziellen Algorithmen simplifiziert werden



Ersatzfläche:
Kriterium: kleiner Winkel α

Ersatzpunkt:
Kriterium: Eckpunkt weniger als α voneinander entfernt

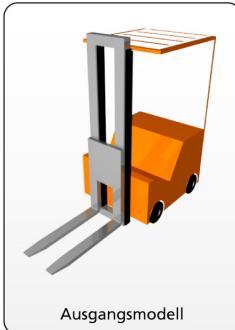
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [360]  © R. Dörner

360



Näherungen

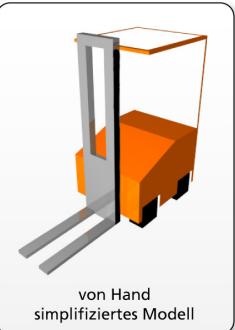
- In manchen Fällen ist Simplifizierung von Hand notwendig (Bewahrung von Semantik)
- Idee: für ein Objekt *mehrere* Objektmodelle anfertigen mit unterschiedlicher Näherung / Detaillierung (Levels-of-Detail, LOD)



Ausgangsmodell



automatisch
simplifiziertes Modell

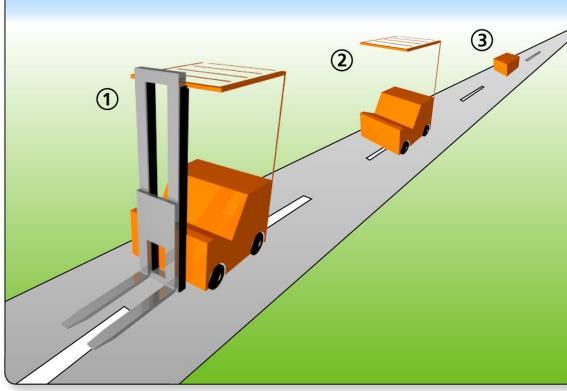


von Hand
simplifiziertes Modell

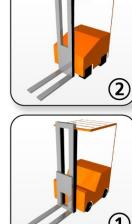
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [361]  © R. Dörner

361

LOD








Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [362]  © R. Dörner

362



Näherungen



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [363]  © R. Dörner

363

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [364]  © R. Dörner

364



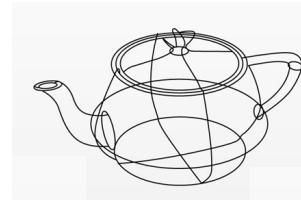
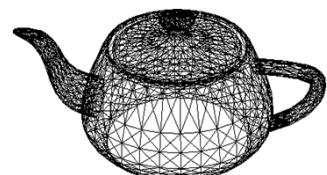
Objektmodelle

- 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



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"> • Nur ein y-Wert für jedes x • Keine vertikalen Tangenten 	<ul style="list-style-type: none"> • Gleichung kann mehr Lösungen als gewollt haben • Richtung der Tangente ist schwer zu ermitteln 	<ul style="list-style-type: none"> • Keine Mehrdeutigkeiten • Tangentenvektoren (keine unendliche Steigung) • Parameter als Zeit interpretierbar

Computergraphik – Prof. Dr. R. Dörner – WS 20

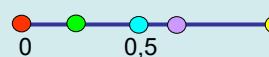
B-AI-V1 [367]

© R. Dörner

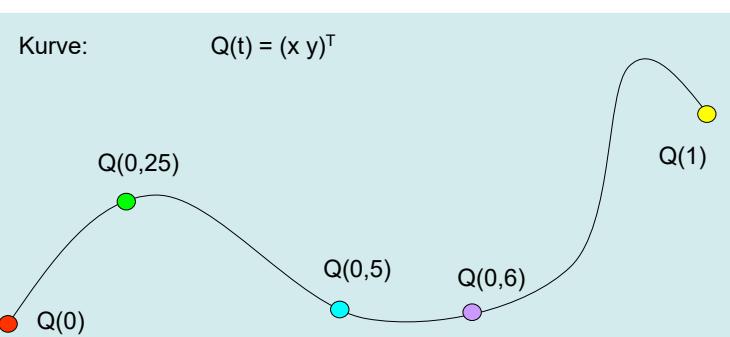
367

Parameterkurven

Parameter: t



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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [368]

© R. Dörner

368



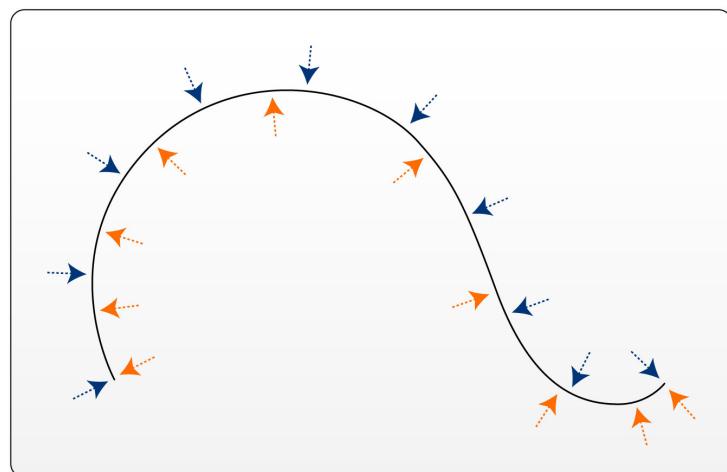
Parameterkurven

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



Unterschiedliche Parametrisierung



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

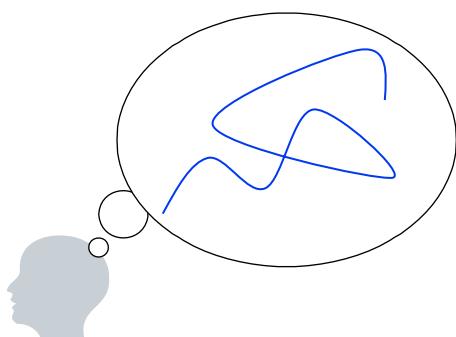
```
// Näherung Kreis durch Polygon mit 100 Eckpunkten
for( float t=0.0; t < 1.0; t += 0.01 ) {
    vArray.push( cos(2.0 * PI * t), sin(2.0 * PI * t) );
}
myDrawingFunction( vArray ); // zeichne die Eckpunkte
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [371]

© R. Dörner

371

Spezifikation von Kurven



Problem:

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [372]

© R. Dörner

372



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

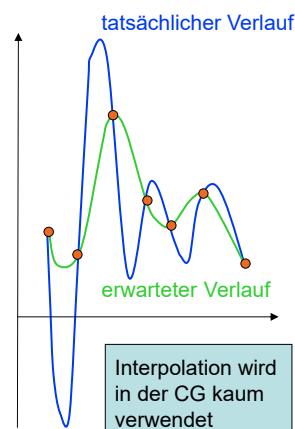
Möglichkeit C:

Approximative Darstellung

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



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 oder zwei Stützstellen und zwei Tangenten gegeben werden
- Grad $k > 3$:
Rechenaufwendig, nur in speziellen Anwendungen



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 20

B-AI-V1 [377]



© R. Dörner

377

Beispiel: Hermite Kurve

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

zwei Stützpunkte

zwei Tangenten
vektoren

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [378]



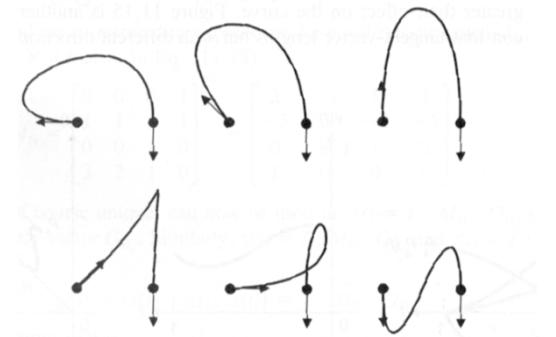
© R. Dörner

378



Beispiel: Hermite Kurve

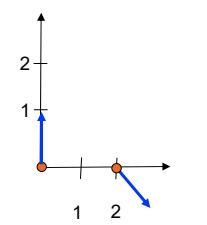
6 Beispiele:
Gleiche Stützpunkte, unterschiedliche Tangentenvektoren



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [379] © R. Dörner

379

Beispiel: Hermite Kurve



$Q(0) = [0 \ 0]$
 $Q(1) = [2 \ 0]$
 $Q\left(\frac{1}{3}\right) = \left[\frac{4}{9} \ \frac{2}{9}\right]$

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

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

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [380] © R. Dörner

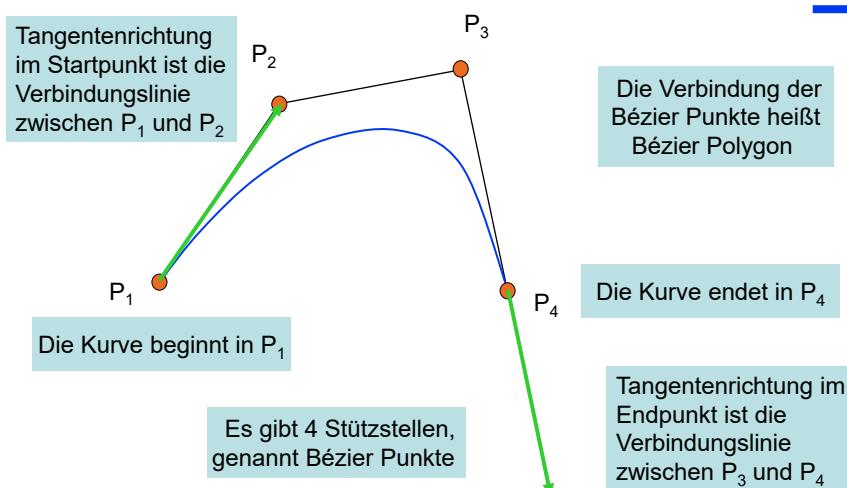
380



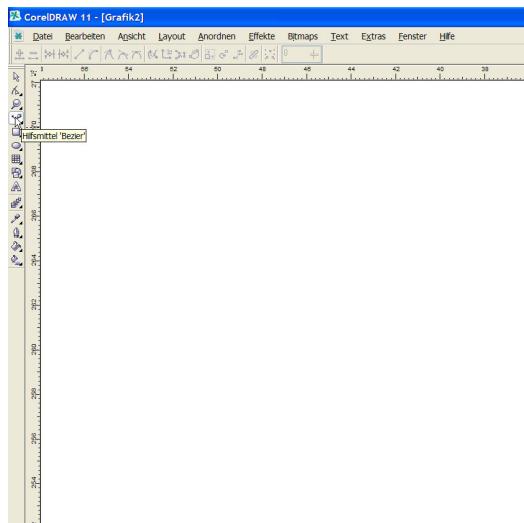
Bézier - Kurven

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

Eigenschaften einer Bézier – Kurve (1)



Beispiel: Bézier-Kurve in CorelDraw



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [383]



© R. Dörner

383

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 20

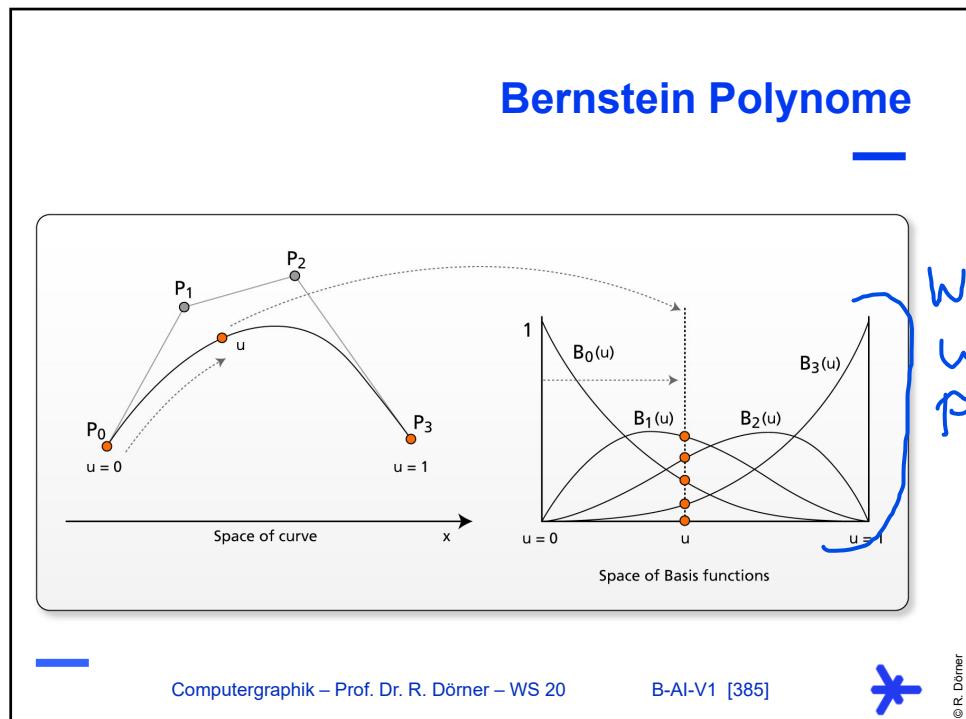
B-AI-V1 [384]



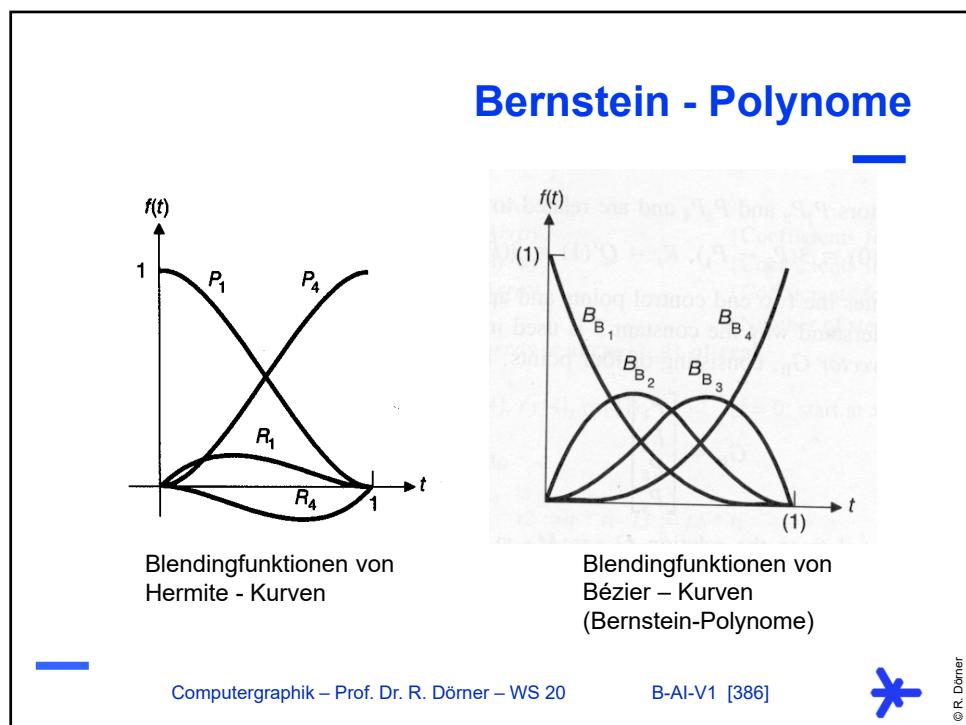
© R. Dörner

384





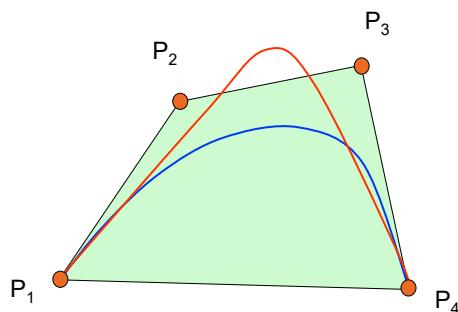
385



386



Eigenschaften einer Bézier – Kurve (2)



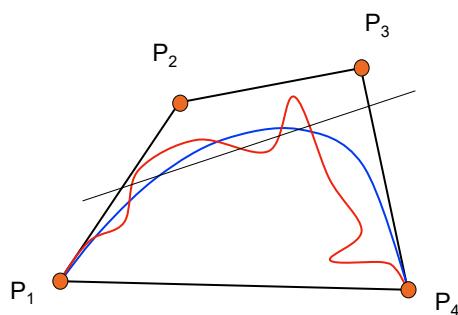
Aus der Sum – of –Unity folgt:

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



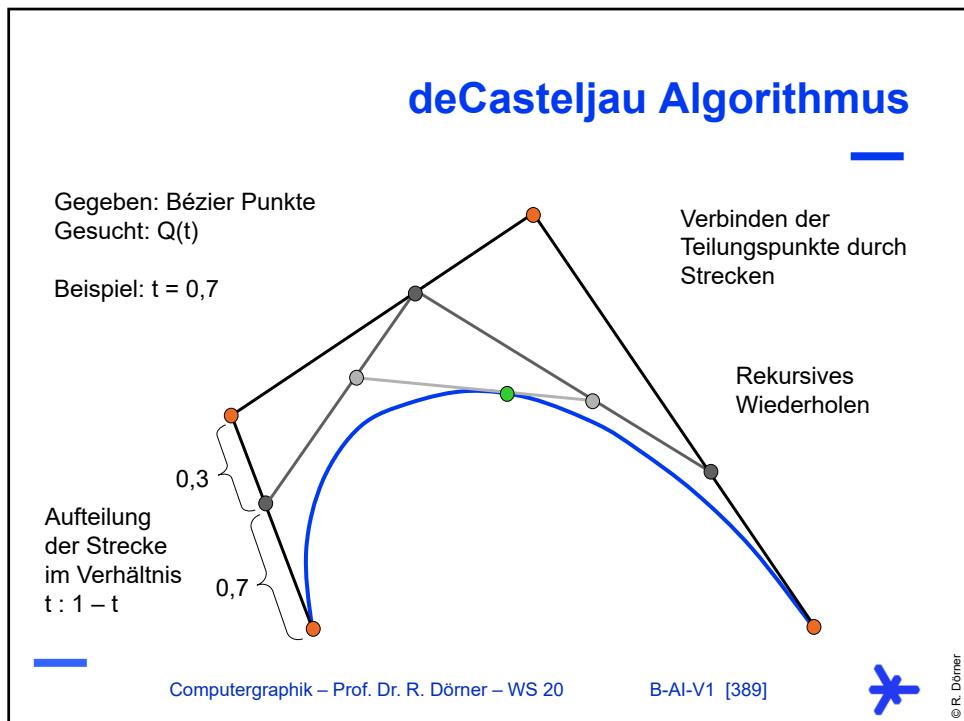
Eigenschaften einer Bézier – Kurve (3)



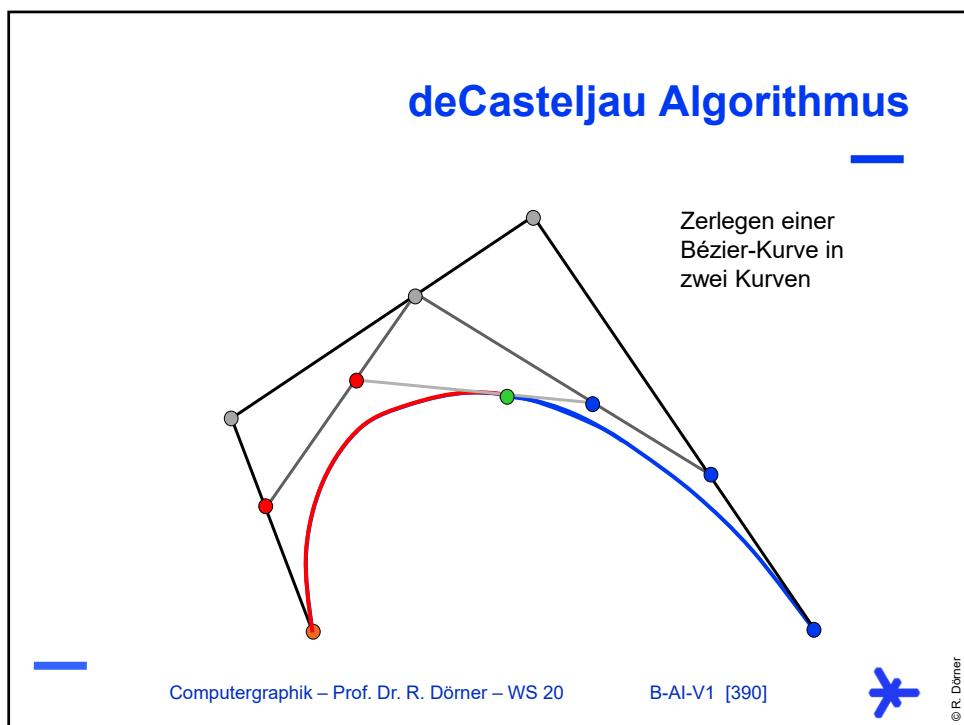
Bézier – Kurven haben die **Variation-Diminishing-Property**, d.h. die Kurve hat höchstens so viele Wendepunkte wie ihr Kontrollpolygon

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





389

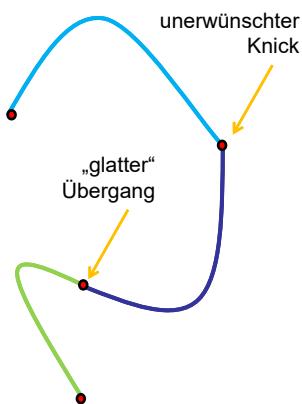


390



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 20

B-AI-V1 [391]

© R. Dörner

391

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 20

B-AI-V1 [392]

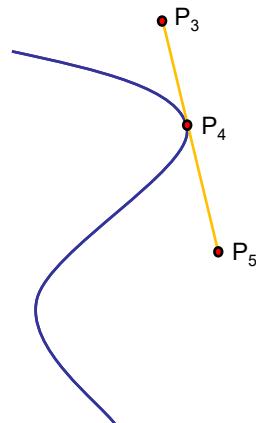
© R. Dörner

392



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

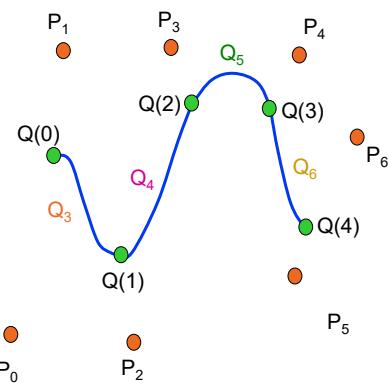
Knoten: $Q(0), Q(1), Q(2), Q(3), Q(4)$

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“)

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [397]



© R. Dörner

397

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

$$T_i = \begin{bmatrix} \left(\frac{t - t_i}{t_{i+1} - t_i} \right)^3 & \left(\frac{t - t_i}{t_{i+1} - t_i} \right)^2 & \left(\frac{t - t_i}{t_{i+1} - t_i} \right) & 1 \end{bmatrix}$$

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [398]



© R. Dörner

398



Kurvensegmente uniformer B-Splines

- Damit lässt sich Q_i in der allgemeinen Form darstellen als

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



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

Beispiel Q₃

Beispiel Q₄

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [401]

© R. Dörner

401

B-Splines: Local Control Property

Änderung von P₁:
Nur Q₃ und Q₄
müssen neu
berechnet werden
Rest des B – Splines
bleibt gleich.

Beispiel Q₃

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [402]

© R. Dörner

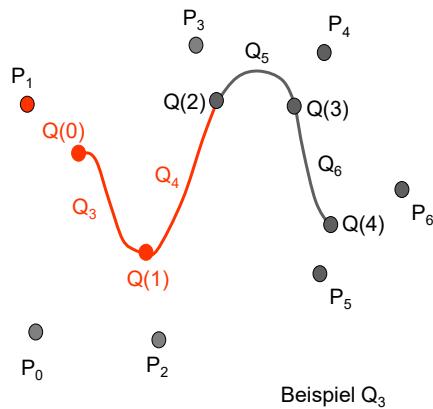
402



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.



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 $T = [t_0, t_1, \dots, t_{n+k}]$:

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



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 $Q(t)$

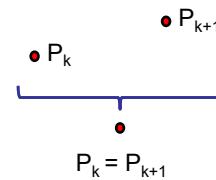
$$Q(t) = \vec{p} = \sum_{i=0}^n N_{i,k}(t) \cdot \vec{p}_i \quad , t \in [t_0, t_{n+k}]$$

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



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



Mehrfaache 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 20

B-AI-V1 [409]



© R. Dörner

409

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 20

B-AI-V1 [410]



© R. Dörner

410

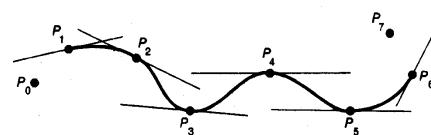


Eigenschaften von NURBS

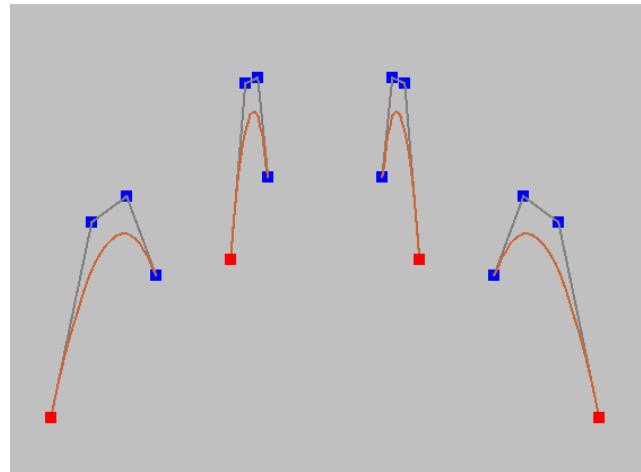
- Haben Eigenschaften der B-Splines (Stetigkeit, lokale Kontrolle, konvexe Hülle für die einzelnen Kurvensegmente)
- Invarianz gegenüber geometrischen Transformationen, wie z.B. Rotation (d.h. nur die Kontrollpunkte müssen transformiert werden und nicht jeder Punkt der Kurve)
- NURBS können Kegelschnitte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

Definition von Kurven

- Es gibt eine große Spannbreite weiterer Kurvendefinitionen
 - β -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 20

B-AI-V1 [413]



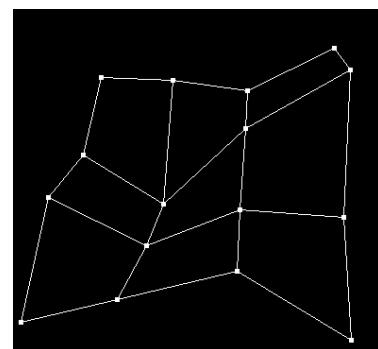
© R. Dörner

413

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 20

B-AI-V1 [414]



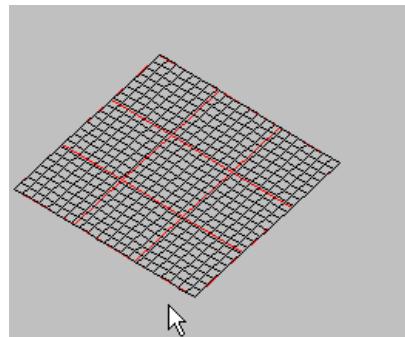
© R. Dörner

414



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
 - ...



Computergraphik – Prof. Dr. R. Dörner – WS 20

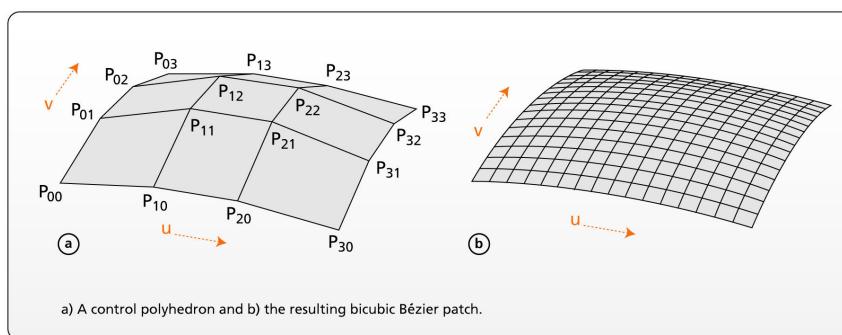
B-AI-V1 [415]



© R. Dörner

415

Beispiel: Bézier-Patches



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [416]

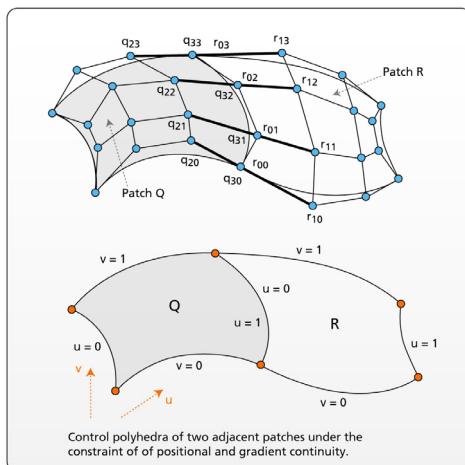


© R. Dörner

416



Zusammensetzen von Patches



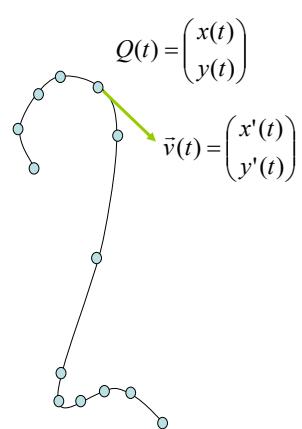
Flächenstücke (Patches) können zu größeren Flächen zusammengesetzt werden, ähnlich wie ein B-Spline aus lauter Kurvensegmenten besteht.

Bestimmte Stetigkeitsbedingungen 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

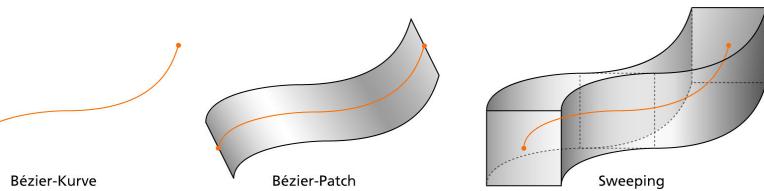
Kantenliste

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 20

B-AI-V1 [421]



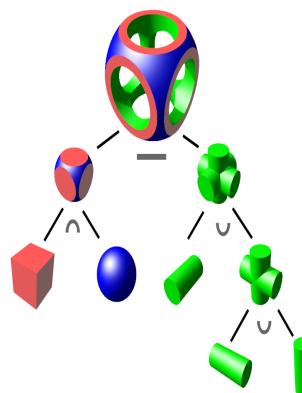
© R. Dörner

421

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 20

B-AI-V1 [422]



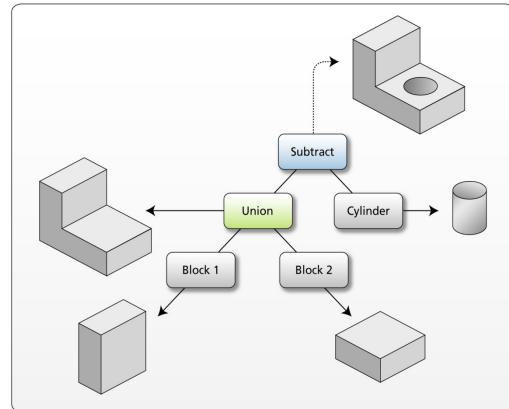
© R. Dörner

422



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 20

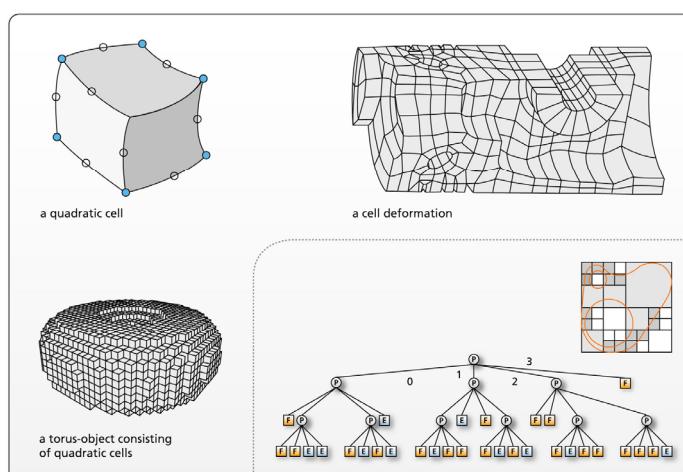
B-AI-V1 [423]



© R. Dörner

423

Volumenbeschreibungen



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [424]



© R. Dörner

424



Volumenbeschreibungen

(a)

(b)

(c)

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [425]

© R. Dörner

425

Quadtree und Partikelsystem

Quadtree

Partikelsystem

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [426]

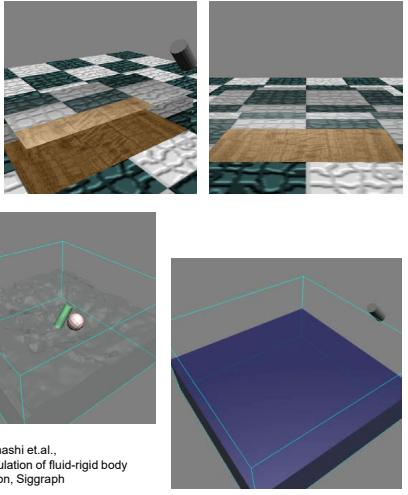
© R. Dörner

426



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 20

B-AI-V1 [427]

© R. Dörner

427

Sonderfall: Gase

Cloud Simulation



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

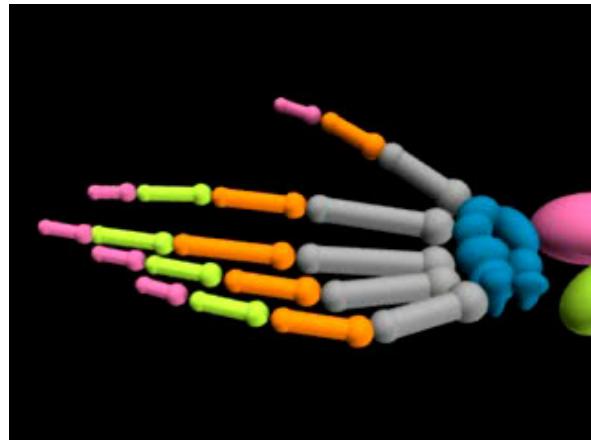
B-AI-V1 [428]

© R. Dörner

428



Sonderfall: Bewegliche Modelle



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

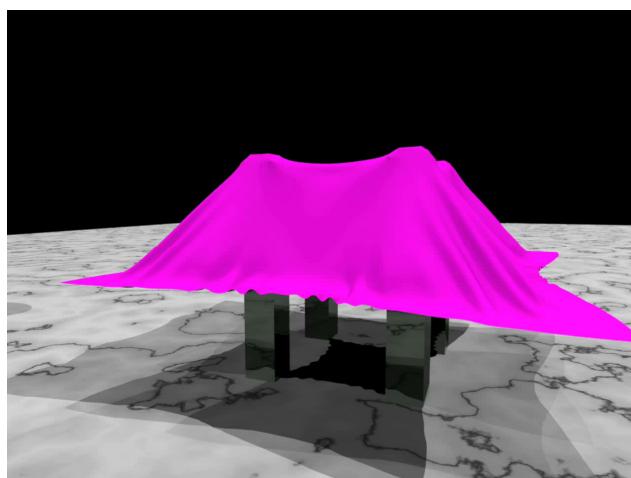
B-AI-V1 [429]



© R. Dörner

429

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 20

B-AI-V1 [430]



© R. Dörner

430



Kriterien für die Auswahl von Objektmodellen



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [433]



© R. Dörner

433

Rendering und OpenGL

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

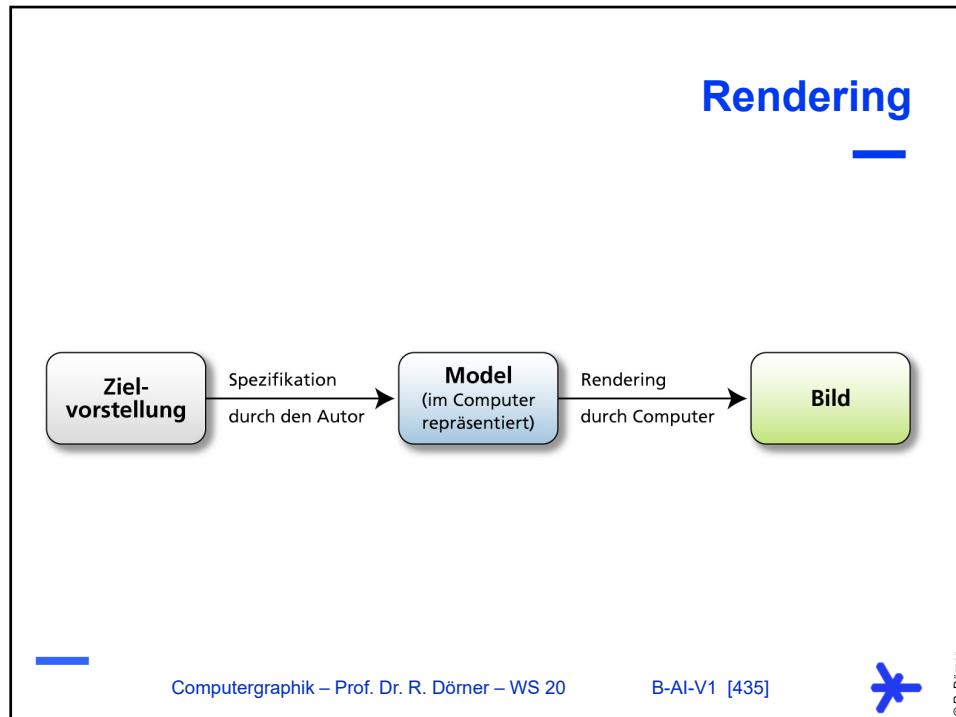
B-AI-V1 [434]



© R. Dörner

434





435

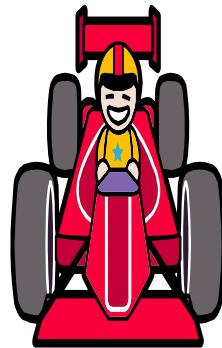


436

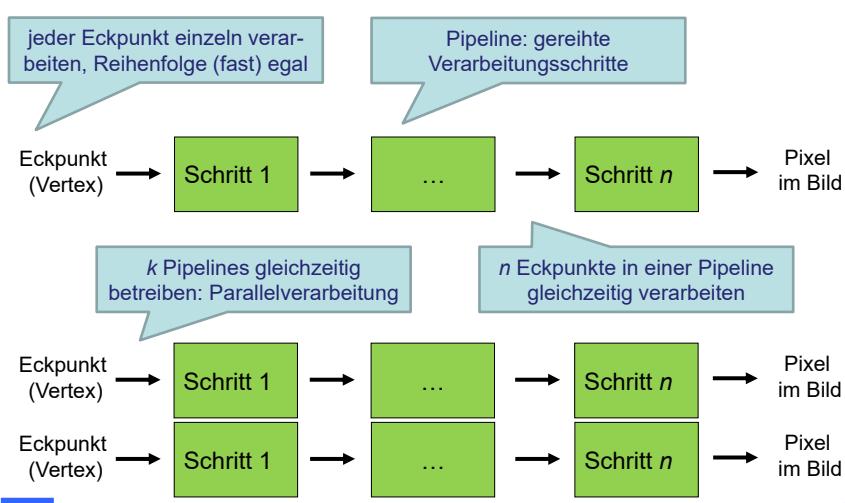


Hardware-Beschleunigung

- 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).



Parallelität und Pipelining



OpenGL Renderpipeline

- OpenGL ist ein offener Standard für das Rendering (www.opengl.org)
- OpenGL hat eine Renderpipeline standardisiert
- Grafikchip-Hersteller (wie z.B. ATI oder nVidia) haben diese Renderpipeline in „Hardware gegossen“
- Rendering geht deutlich schneller, ABER: keine Flexibilität mehr, wie das Rendering durchgeführt wird



Varianten beim Rendering



photorealistisch



Cartoon



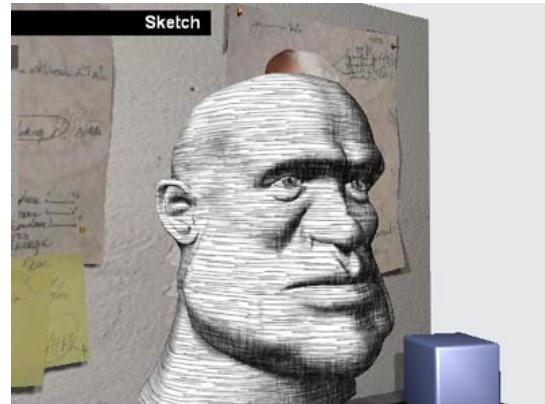
schraffiert



Aquarell



Beispiele von Shadern



Quelle: Virtools (www.3ds.com)

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [441]



441

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 20

B-AI-V1 [442]



© R. Dörner

442



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 20

B-AI-V1 [443]



© R. Dörner

443

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 20

B-AI-V1 [444]



© R. Dörner

444



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 20

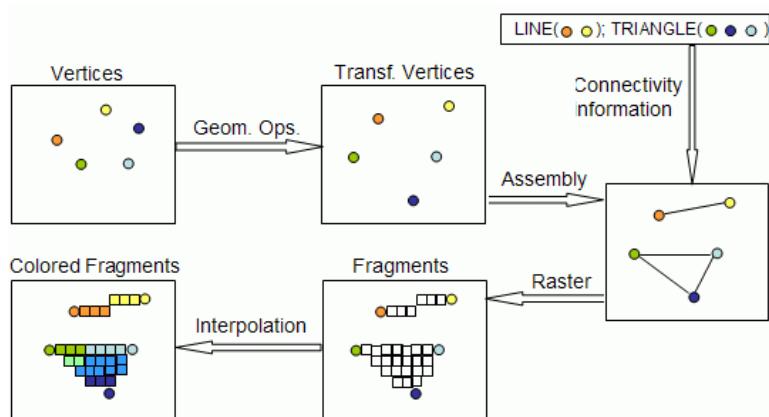
B-AI-V1 [445]



© R. Dörner

445

Überblick der Renderpipeline



Quelle:
<http://www.lighthouse3d.com/opengl/glsl>

Computergraphik – Prof. Dr. R. Dörner – WS 20

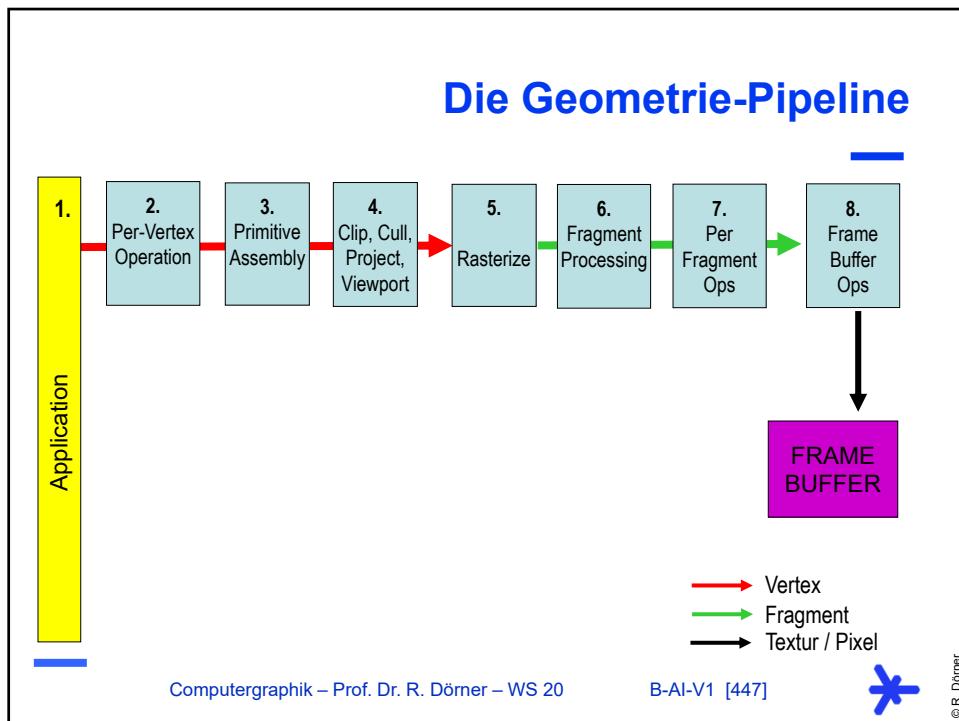
B-AI-V1 [446]



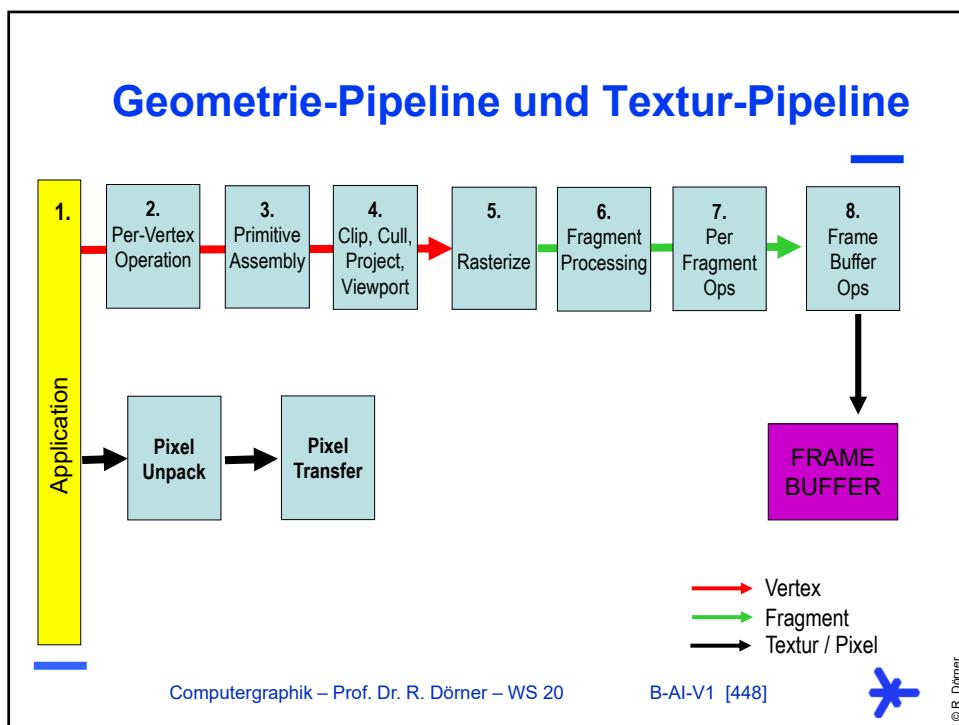
© R. Dörner

446



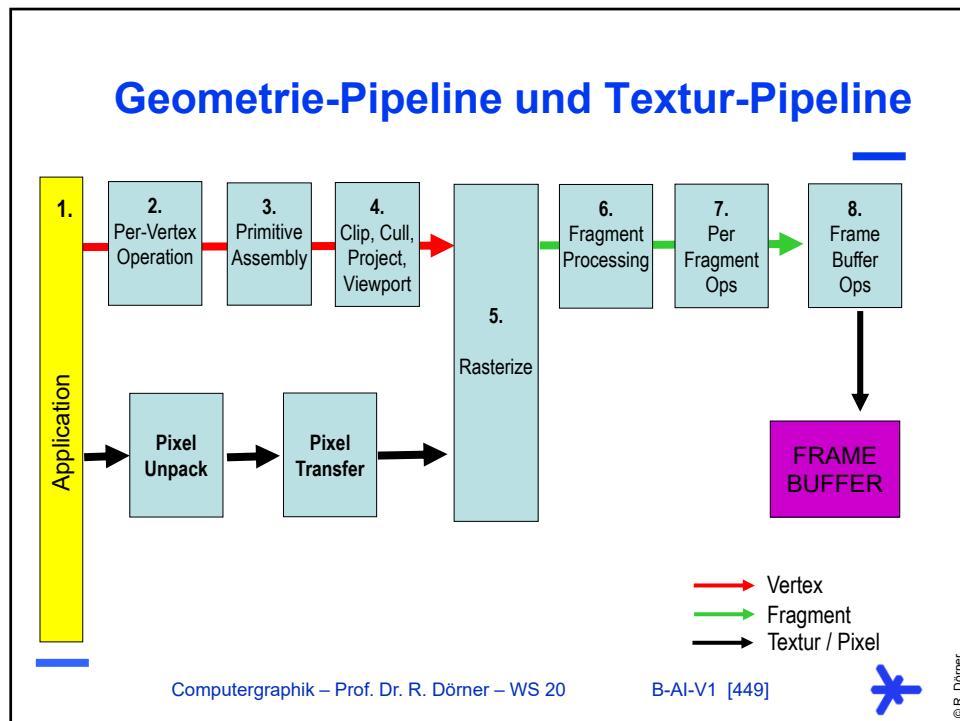


447

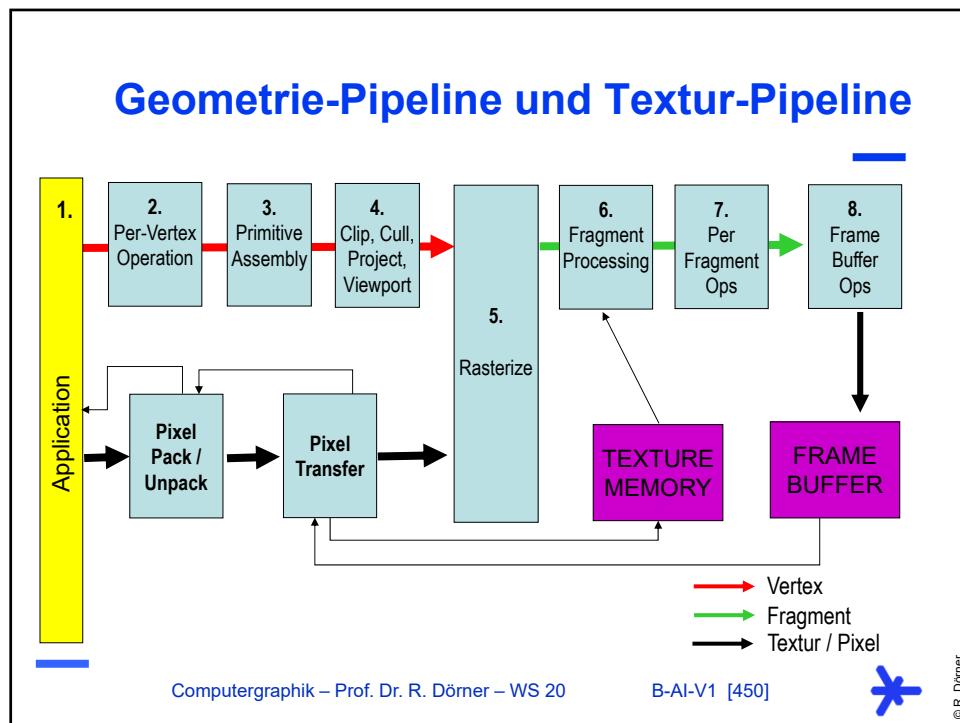


448



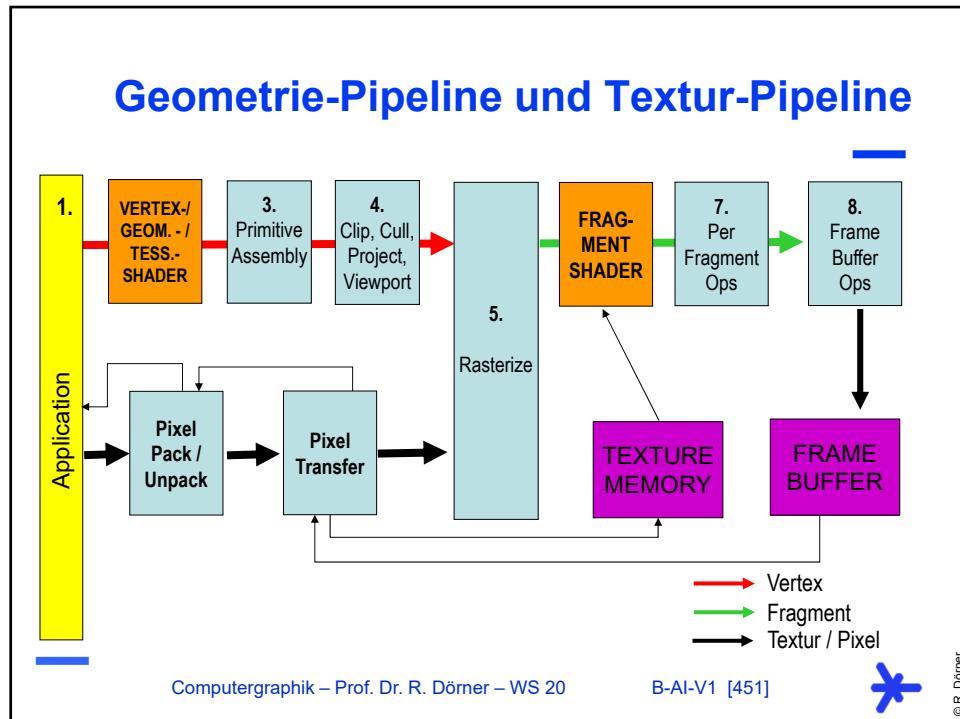


449

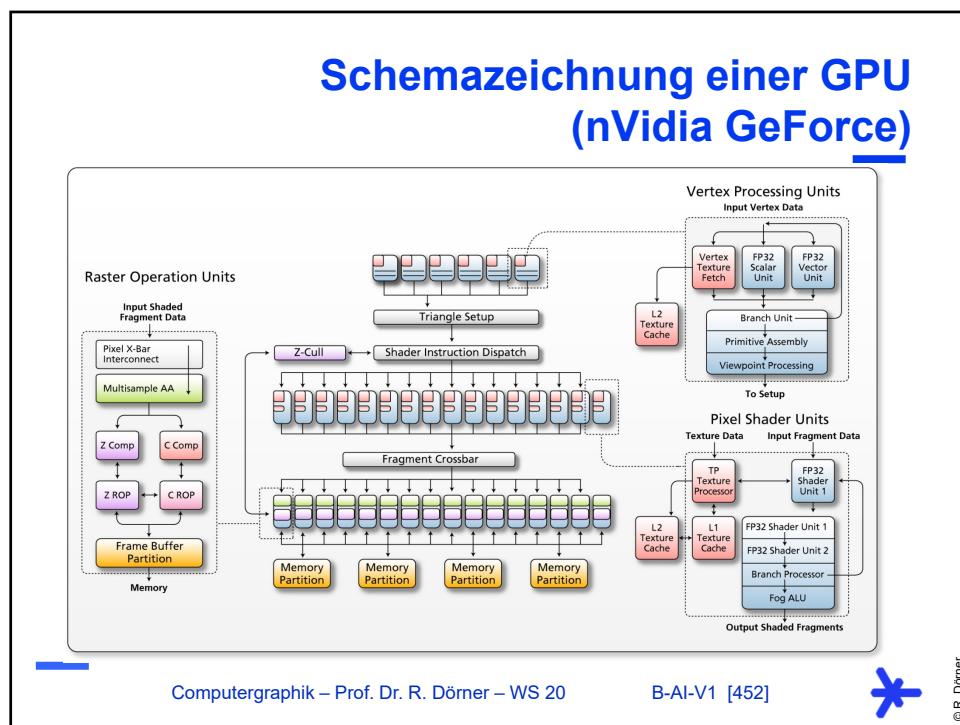


450





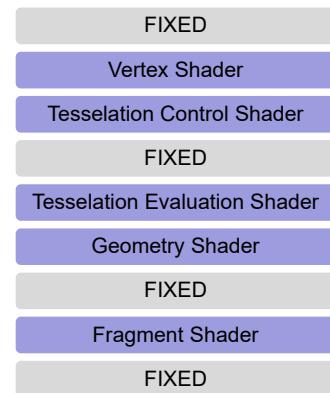
451



452

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

- 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`



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);`

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [461]



© R. Dörner

461

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 20

B-AI-V1 [462]



© R. Dörner

462



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 20

B-AI-V1 [463]



© R. Dörner

463

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 20

B-AI-V1 [464]



© R. Dörner

464



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`



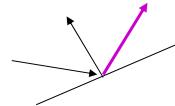
Eingebaute Funktionen

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



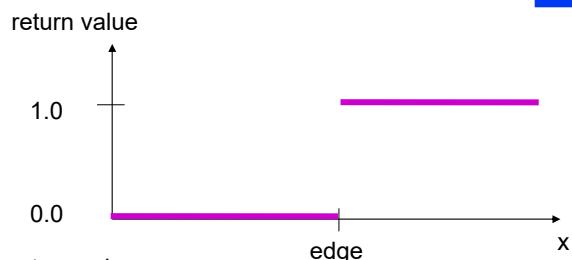
Eingebaute Funktionen

- 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 $\langle Nref, I \rangle$ kleiner 0, ansonsten $-N$
- Die Funktion `reflect(I, N)` liefert die Reflektion des Eingangsvektors I an der Normalen N

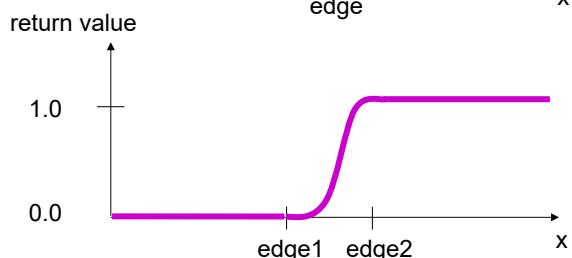


Eingebaute Funktionen

- `step(
edge,
x)`



- `smoothstep(
edge1,
edge2,
x)`



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [469]



© R. Dörner

469

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 20

B-AI-V1 [470]



© R. Dörner

470



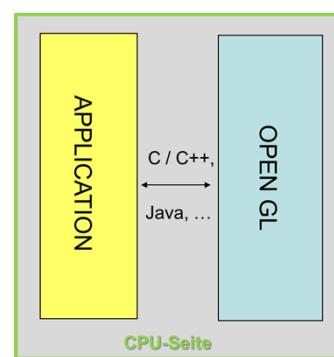
Rendering und OpenGL

- 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 wird durch ein Industriekonsortium weiter entwickelt: die Khronos Group (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



Khronos Group
Over 100 companies creating
authoring and acceleration standards
Board of Promoters



OpenGL

WebGL

473

WebGL und HTML

- Nutzen des Canvas – Elemente 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>

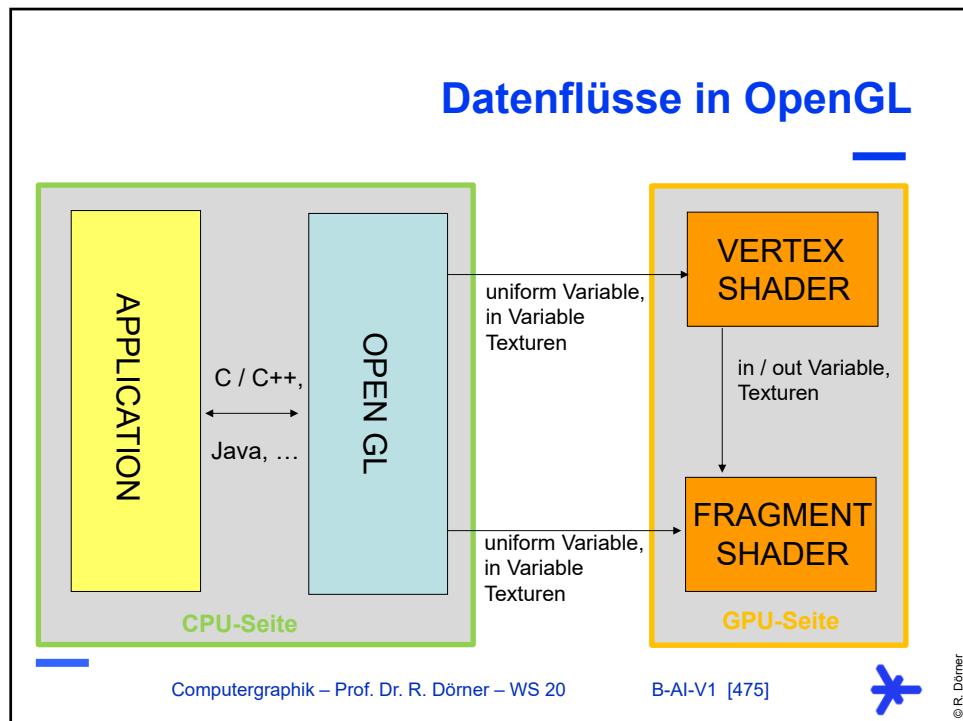
window.onload = function init() {
  canvas =
    document.getElementById("glc" );
  gl = WebGLUtils.setupWebGL( canvas );
  if ( !gl ) {
    alert( "WebGL isn't available" );
  }
  gl.clearColor( 0.9, 0.9, 1.0, 1.0 );
  USW.
```

Computergraphik – Prof. Dr. R. Dörner – WS 20

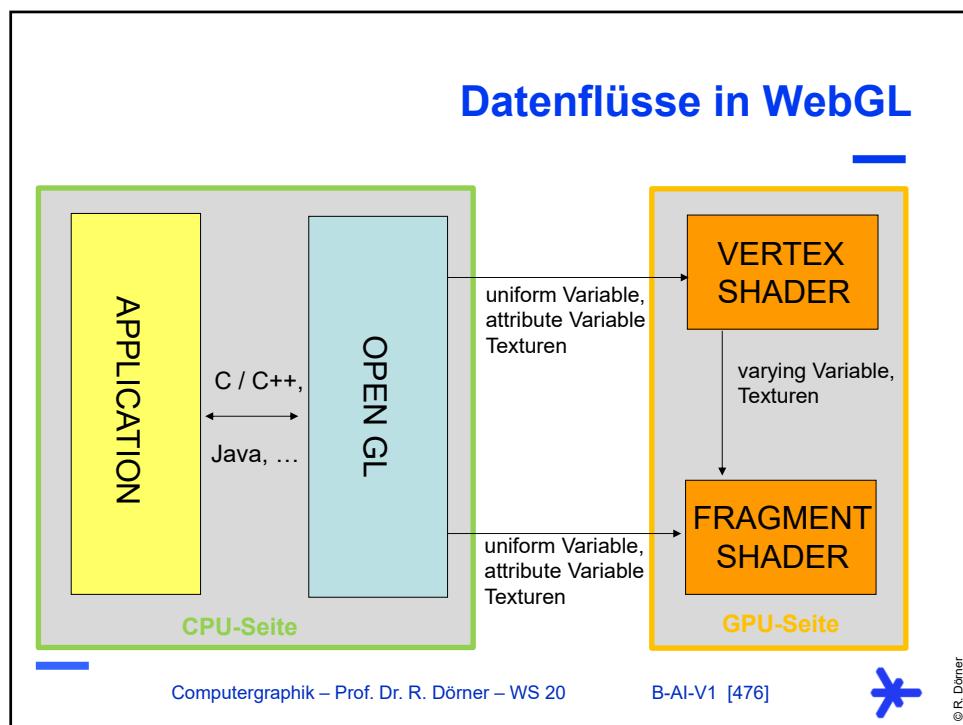
B-AI-V1 [474]

474





475

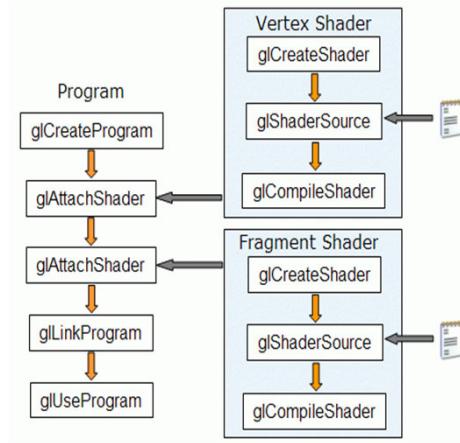


476



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



Quelle: Lighthouse3D

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [477]



© R. Dörner

477

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 20

B-AI-V1 [478]



© R. Dörner

478



Beispiel Uniform Variable in WebGL

```
gl.useProgram(program);

var aFlag = true;
var aNumber = 0.4;
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);
```



479

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

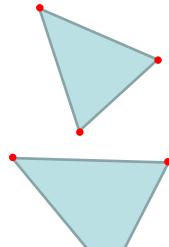


Vertex Operationen

- 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

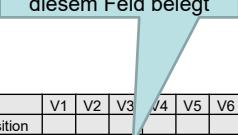
jeder Vertex (mit zugehörigen Daten) wird einzeln (evtl. parallel zu anderen Vertices) verarbeitet

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [483]

483

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [484]

484

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [485]



485

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; };
```

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [486]



486



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`

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [487]

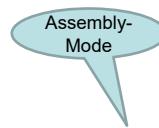


487

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 );
```



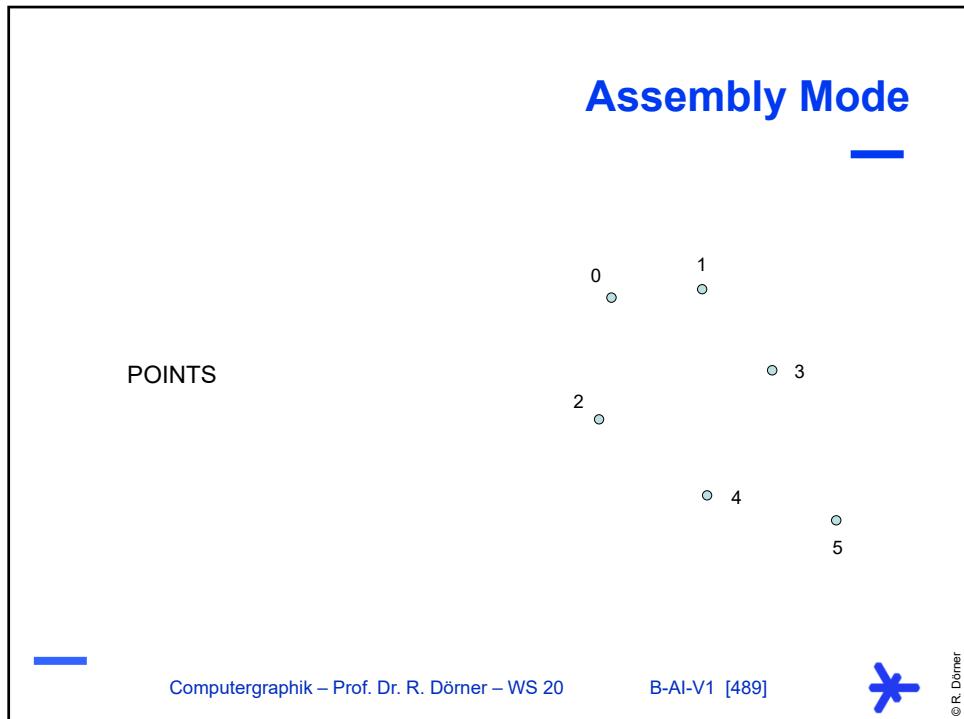
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [488]

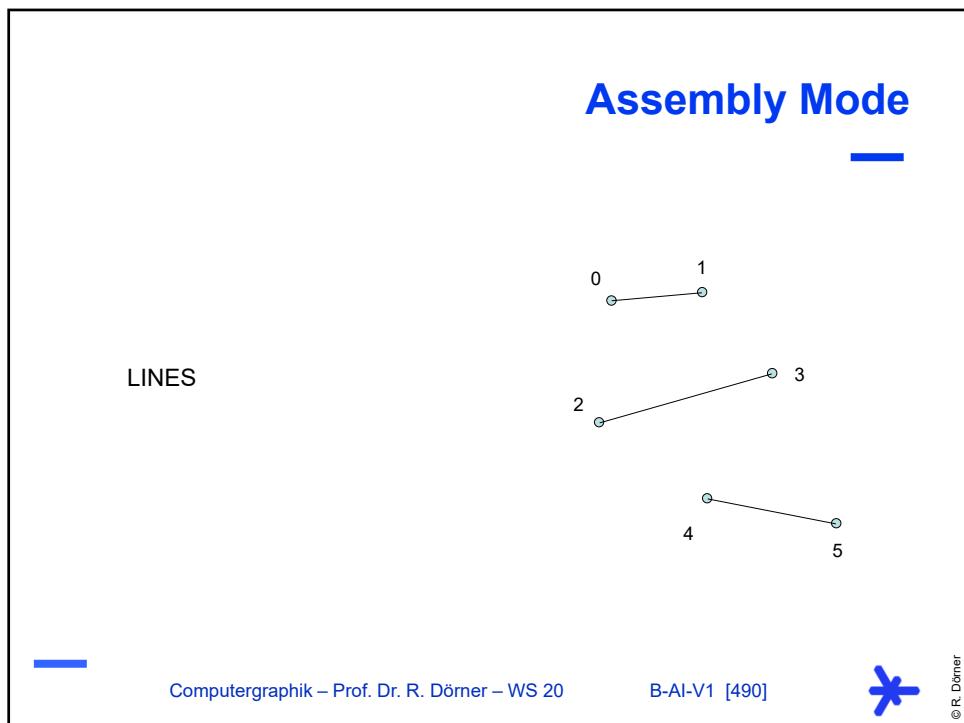


488





489

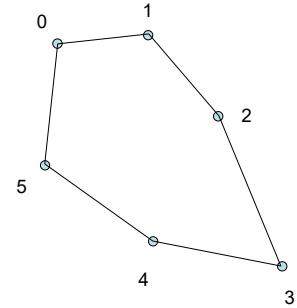


490



Assembly Mode

LINE_LOOP



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [491]

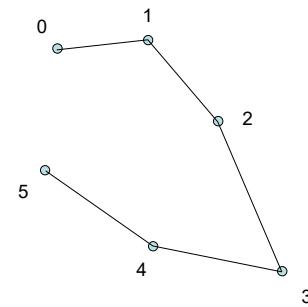


© R. Dörner

491

Assembly Mode

LINE_STRIP



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [492]



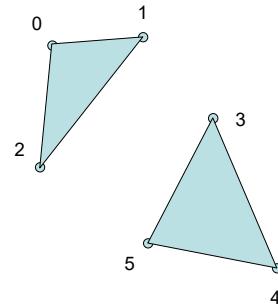
© R. Dörner

492



TRIANGLES

Assembly Mode



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [493]

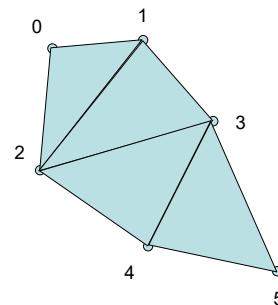


© R. Dörner

493

TRIANGLE_STRIP

Assembly Mode



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [494]



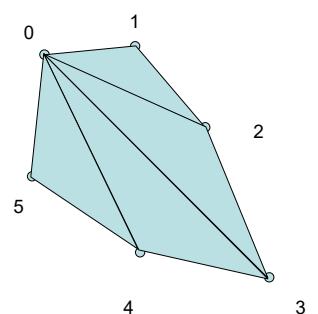
© R. Dörner

494



Assembly Mode

TRIANGLE_FAN

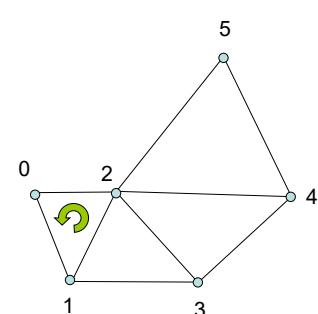


Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [495]  © R. Dörner

495

TriangleStrips

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



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

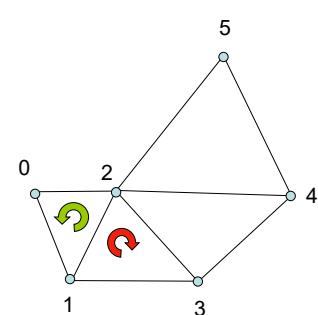
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [496]  © R. Dörner

496



TriangleStrips

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



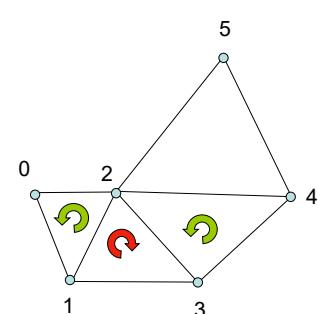
Strip: 0, 1, 2, 3, 4, 5

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [497]  © R. Dörner

497

TriangleStrips

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



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

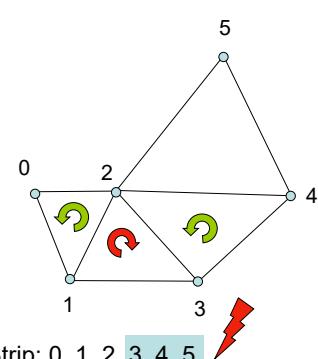
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [498]  © R. Dörner

498



TriangleStrips

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



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

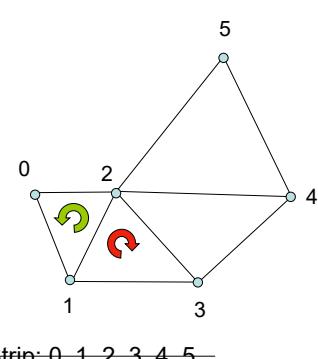
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [499]

© R. Dörner

499

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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [500]

© R. Dörner

500



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [501]

© R. Dörner

501

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [502]

© R. Dörner

502



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [503]

© R. Dörner

503

Vertex Operationen

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [504]

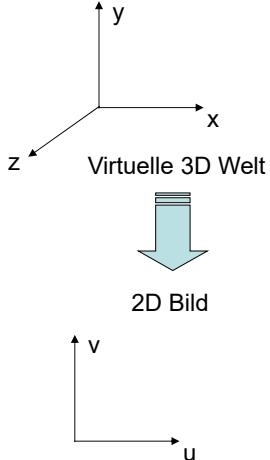
© R. Dörner

504



Projektion

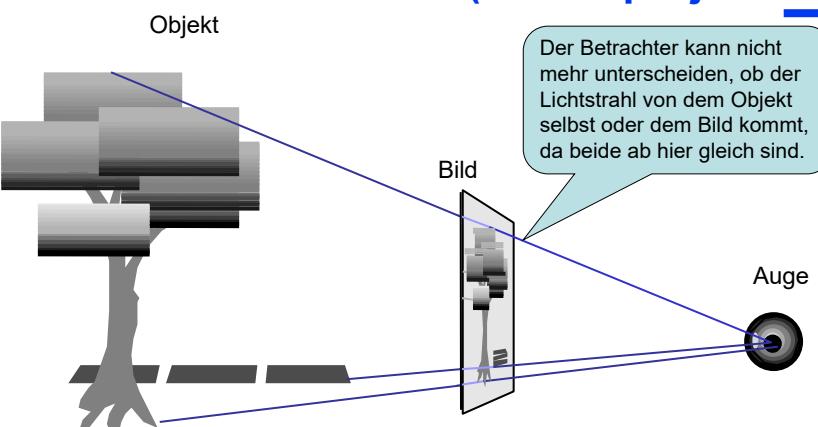
- Geometrische Transformationen wie Skalierung, Rotation, Translation bleiben in der virtuellen Welt
- Im Rendering müssen wir von der 3D Welt auf ein 2D Bild kommen
- Reduktion der Dimensionalität nennt man Projektion



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [505] © R. Dörner

505

Perspektivische Projektion (Zentralprojektion)

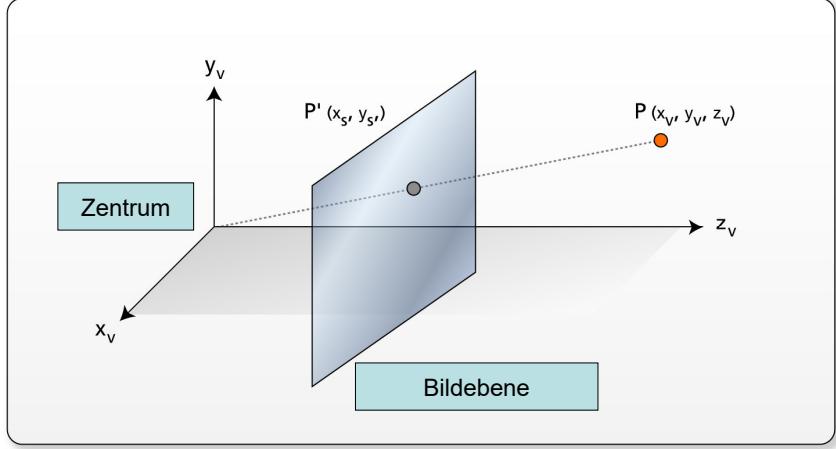


Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [506] © R. Dörner

506



Ausgangssituation



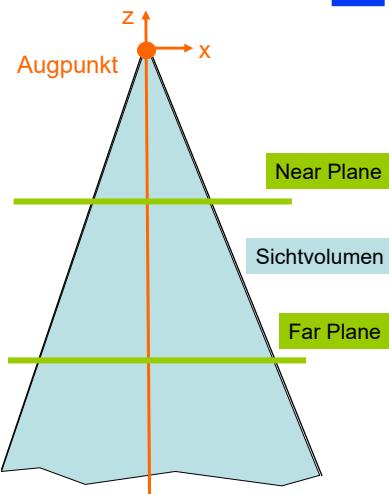
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [507]

© R. Dörner

507

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 20 B-AI-V1 [508]

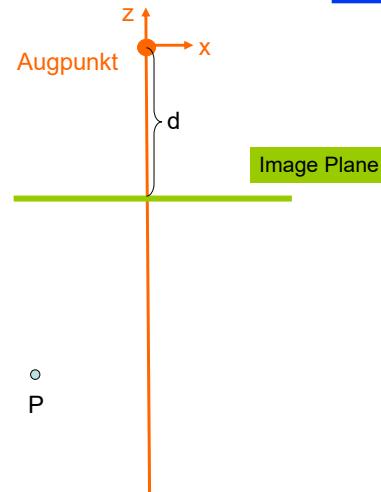
© R. Dörner

508



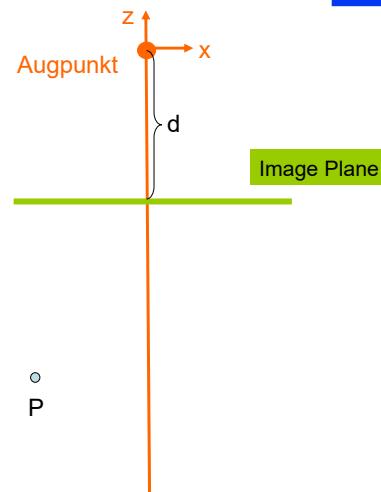
Perspektivische Projektion

- 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



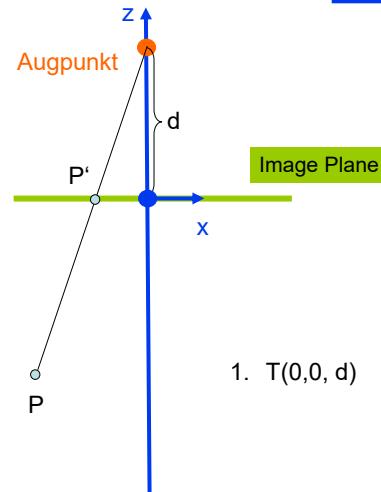
Perspektivische Projektion

- Wir verschieben die Bildebene in die xy-Ebene



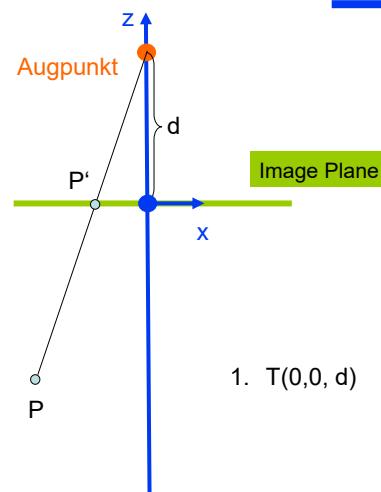
Perspektivische Projektion

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



Perspektivische Projektion

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



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 20

B-AI-V1 [513]

© R. Dörner

513

Perspektivische Projektion

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [514]

© R. Dörner

514



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?

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

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [515]



© R. Dörner

515

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

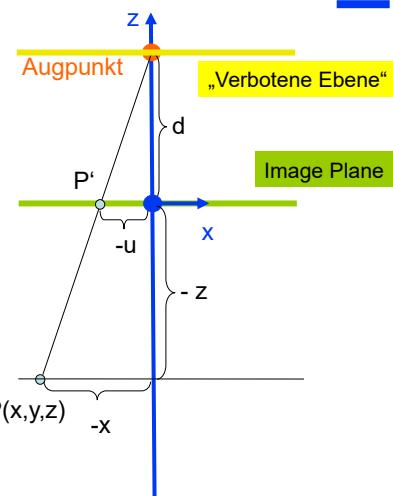
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1-z/d \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \frac{d-z}{d} \end{pmatrix}$$

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

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

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

Definitionslücke
für $z = d$



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [516]



© R. Dörner

516

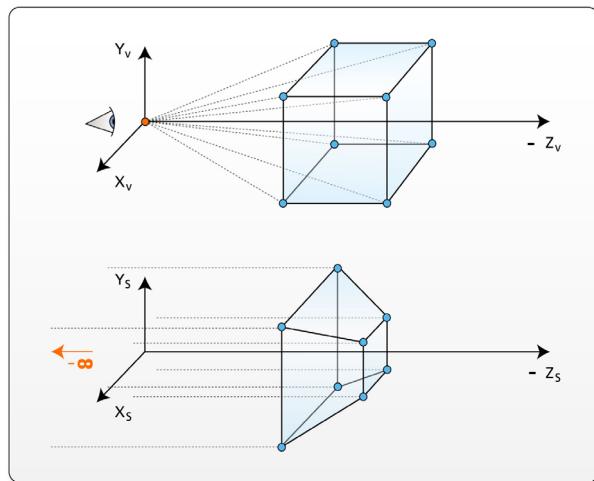


Perspektivische Verzerrung

$M_{\text{per}}(d)$ 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_{per} die Welt verzerrt.

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



Weitere Projektionsarten

- Haben bislang nur perspektivische Projektion betrachtet

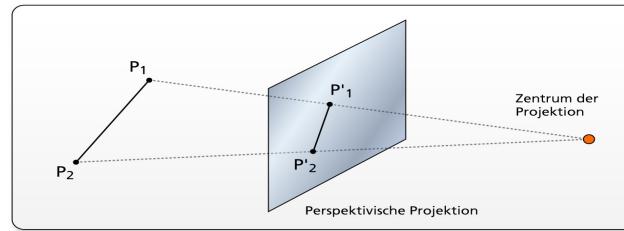
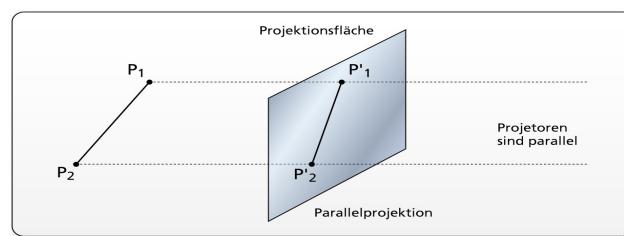


Weitere Projektionsarten

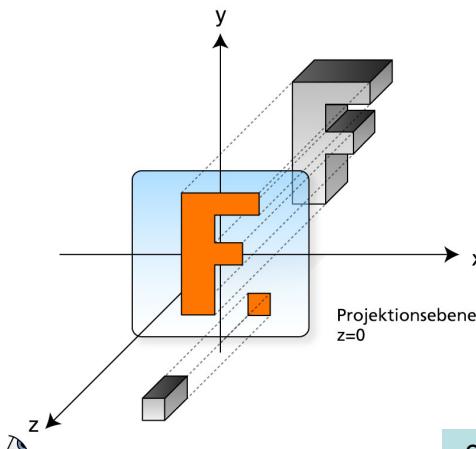
- Haben bislang nur perspektivische Projektion betrachtet
- Lassen wir den Abstand d zwischen Zentrum und Bildebene ins unendliche wachsen, dann sind die Projektionsstrahlen parallel: Parallelprojektion
- Parallelprojektion verkleinert entfernte Objekte nicht



Parallelprojektion



Parallelprojektion: Standardfall



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

Sichtvolumen ist ein Quader.

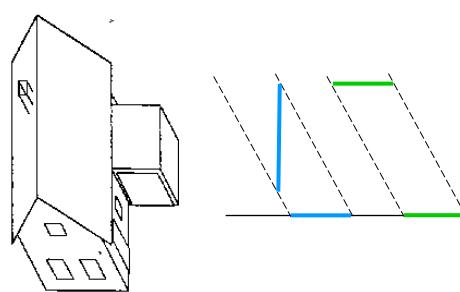
`ortho(l, r, b, t, z, f)`

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [521]

© R. Dörner

521

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 20 B-AI-V1 [522]

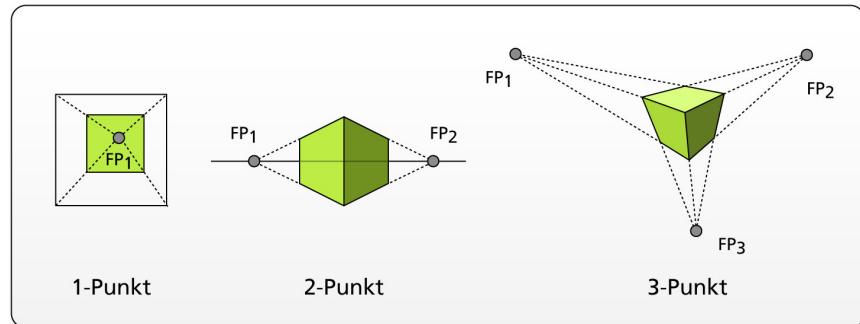
© R. Dörner

522

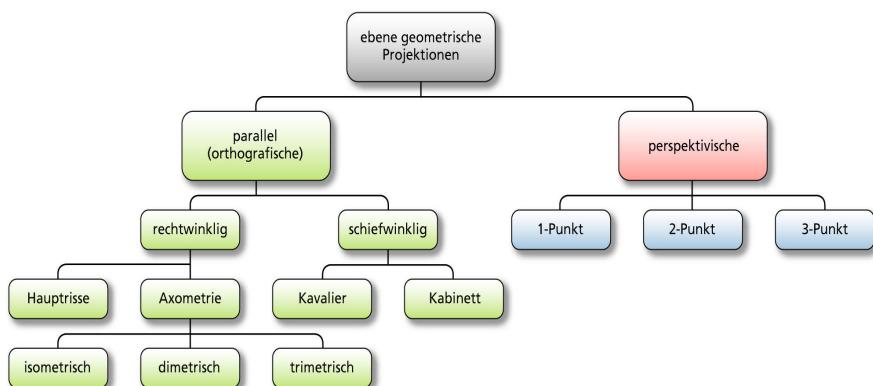


Weitere Projektionsarten

- Verschiedene Projektionsarten bei Zentralprojektion:



Projektionsarten



Transformationsmatrizen als Alleskönner

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & 1 \end{bmatrix}$$

Skalierung

Rotation

Translation

Projektion



Vertex Operationen

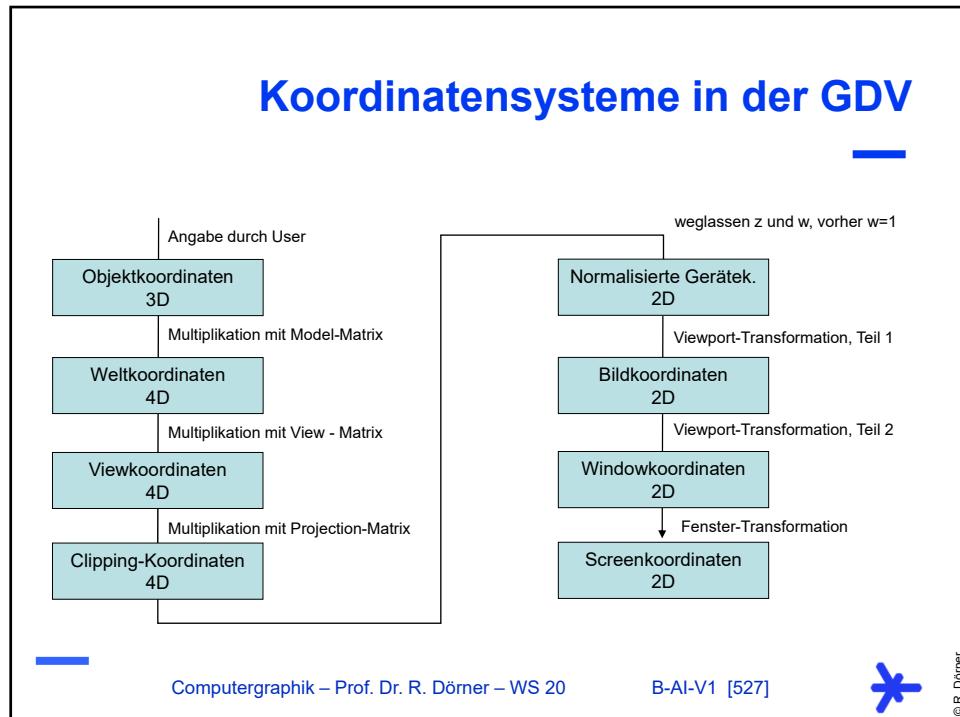
H.1 Übergabe der Vertex-Informationen

H.2 Projektion

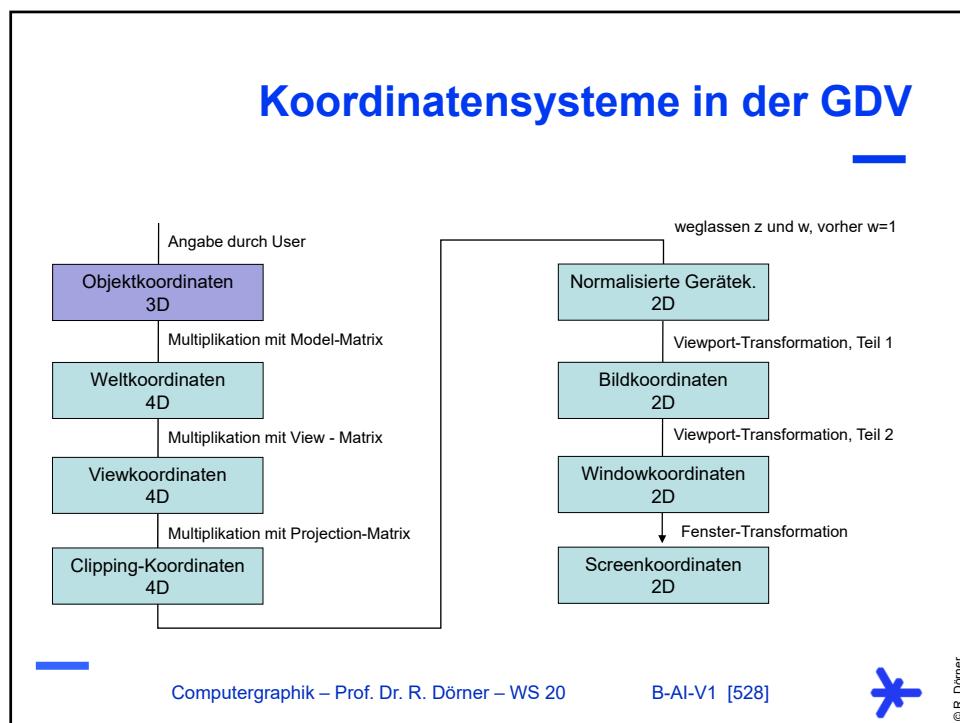
H.3 Umrechnung Koordinatensysteme

H.4 Aufgaben des Vertex-Shader





527



528



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 $A \cdot B$

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 20

B-AI-V1 [529]



529

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));          (2)
model = mult(model, rotate(180.0, [0.0, 0.0, 1.0]));    (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 20

B-AI-V1 [530]



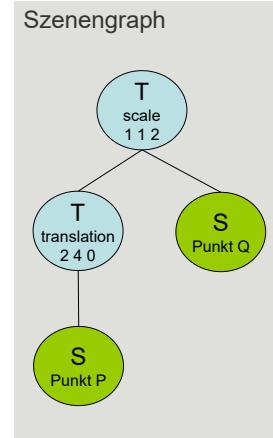
© R. Dörner

530



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 20

B-AI-V1 [531]

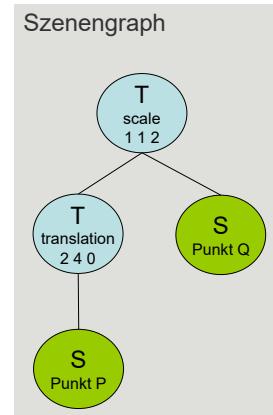


© R. Dörner

531

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 20

B-AI-V1 [532]



© R. Dörner

532



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

Aktueller Stack:
/ / / / / /

Szenengraph

```

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

```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [533] © R. Dörner

533

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

```

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

```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [534] © R. Dörner

534



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

Szenengraph

```

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

```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [535]

© R. Dörner

535

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

```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [536]

© R. Dörner

536



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

```

Computergraphik – Prof. Dr. R. Dörner – WS 20
B-AI-V1 [537]
© R. Dörner

537

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:

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

Szenengraph

```

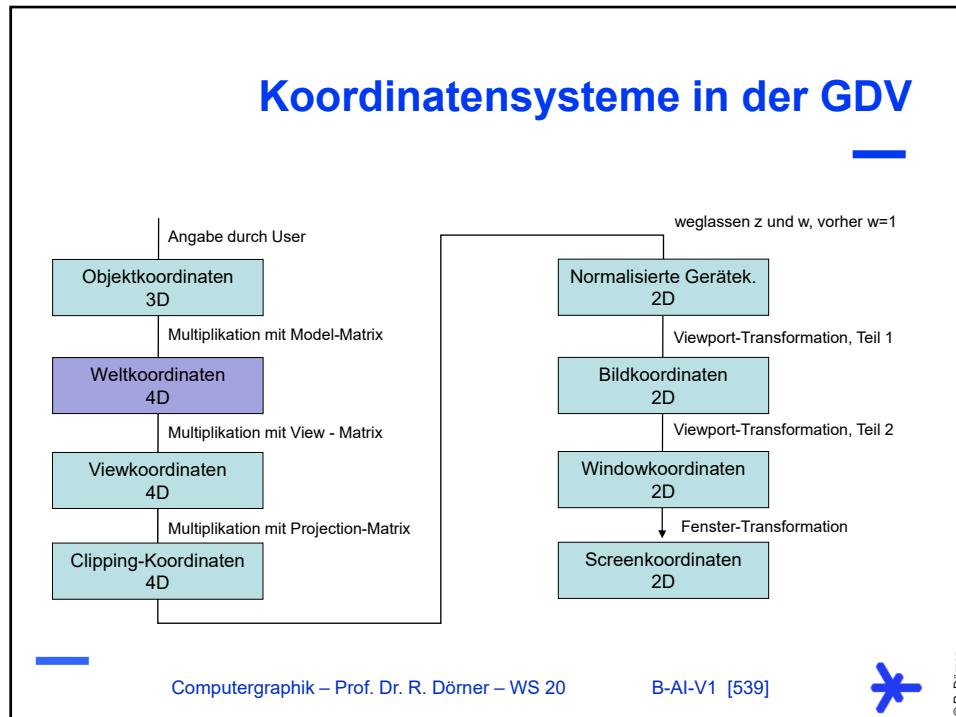
graph TD
    T1[T scale 1 1 2] --> T2[T translation 2 4 0]
    T1 --> S1[S Punkt Q]
    T2 --> S2[S Punkt P]

```

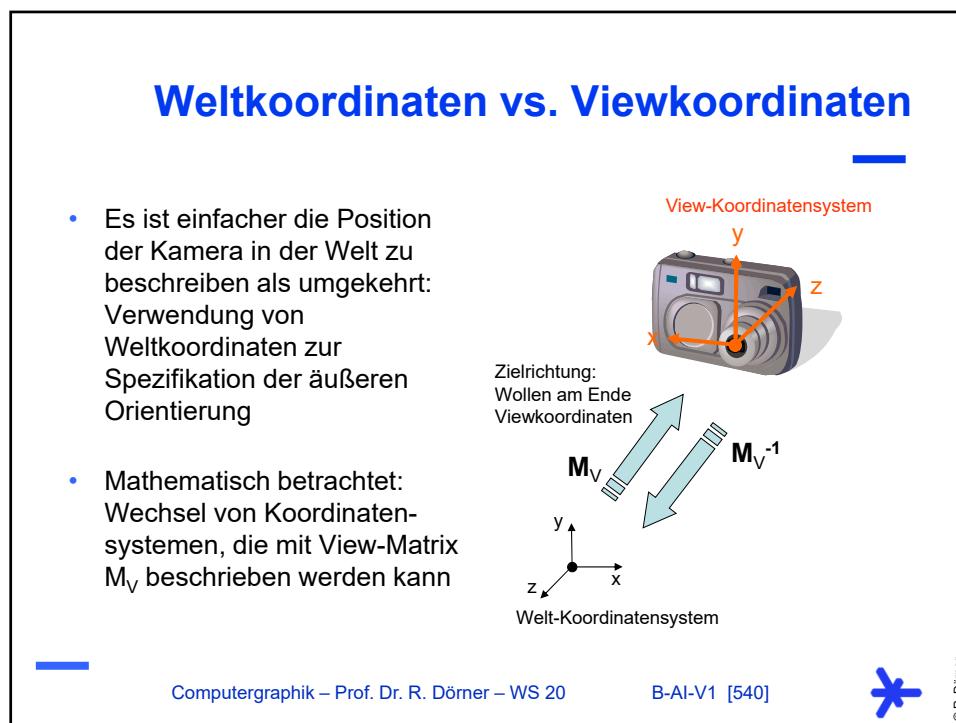
Computergraphik – Prof. Dr. R. Dörner – WS 20
B-AI-V1 [538]
© R. Dörner

538





539

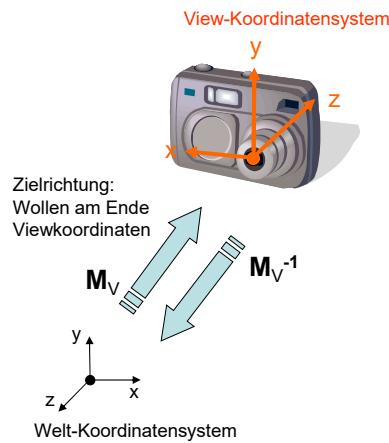


540



Weltkoordinaten vs. Viewkoordinaten

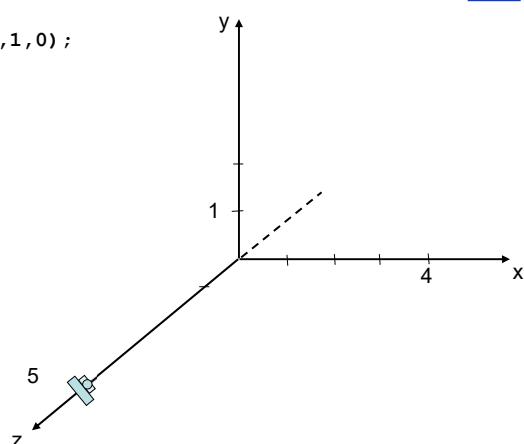
- Die View-Matrix kann durch folgende Werte spezifiziert werden:
 - Punkt: Koordinaten der Kameraposition
 - Punkt: Koordinaten des Viewing Reference Point
 - Vektor: Up-Vektor
- Eine Funktion `lookAt` selbst schreiben, die aus den Werten die 4x4 View-Matrix aufstellt
- Alternativ: Die View-Matrix kann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



Beispiel: Umrechnung Welt- zu Viewkoordinaten

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

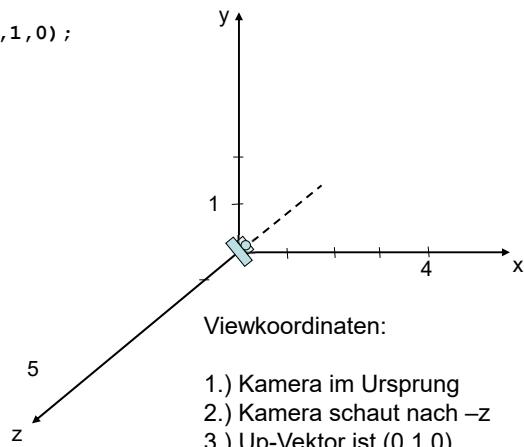
T(0, 0, -5)



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 20

B-AI-V1 [543]



© R. Dörner

543

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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [544]



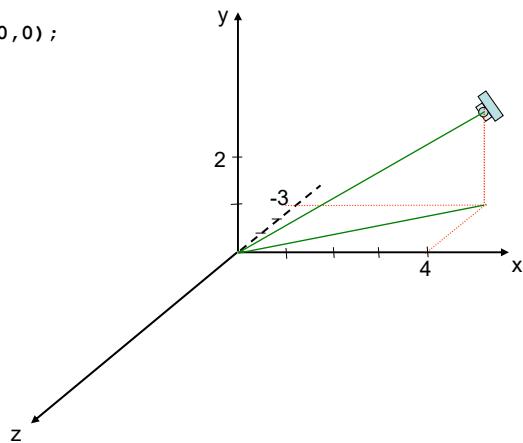
© R. Dörner

544



Beispiel: Umrechnung Welt- zu Viewkoordinaten

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [545]

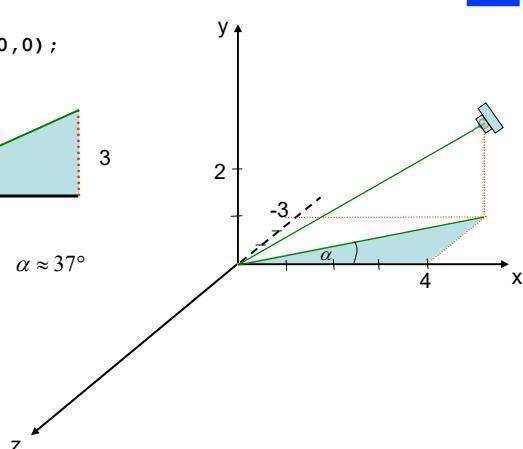
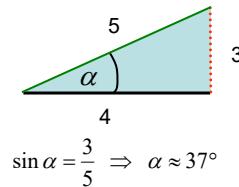


© R. Dörner

545

Beispiel: Umrechnung Welt- zu Viewkoordinaten

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [546]



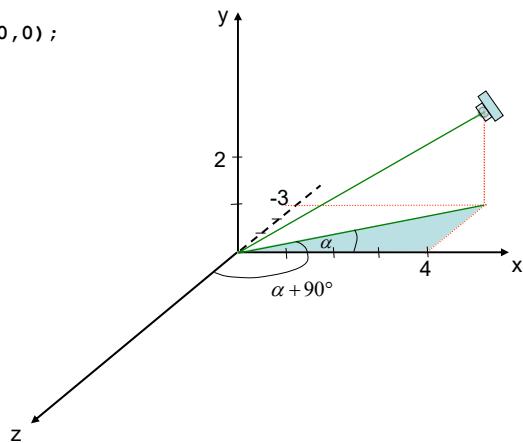
© R. Dörner

546



Beispiel: Umrechnung Welt- zu Viewkoordinaten

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [547]

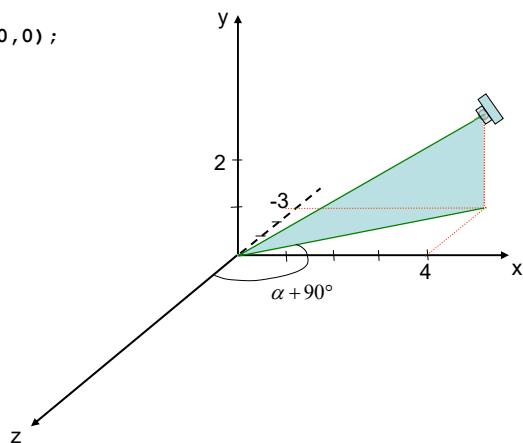


© R. Dörner

547

Beispiel: Umrechnung Welt- zu Viewkoordinaten

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



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [548]



© R. Dörner

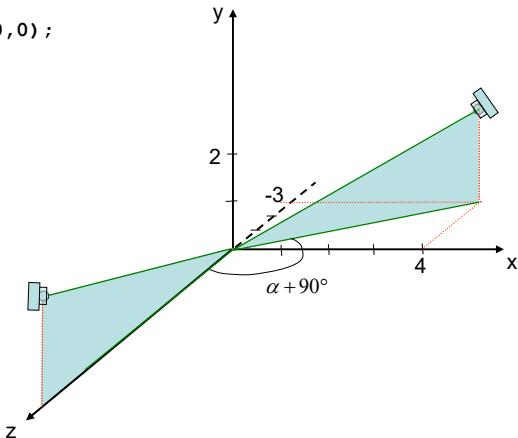
548



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 20

B-AI-V1 [549]



© R. Dörner

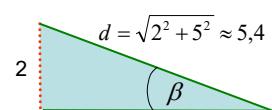
549

Beispiel: Umrechnung Welt- zu Viewkoordinaten

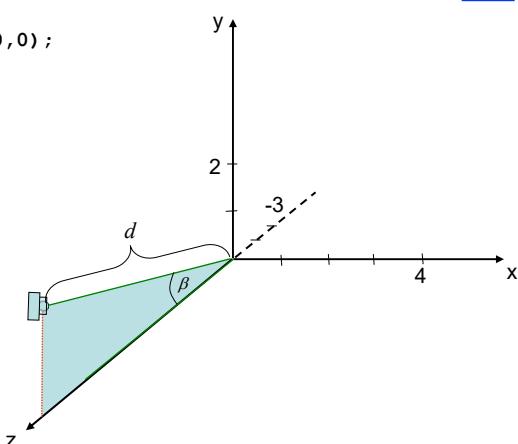
`lookAt(4, 2, -3, 0, 0, 0, 1, 0, 0);`

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 20

B-AI-V1 [550]



© R. Dörner

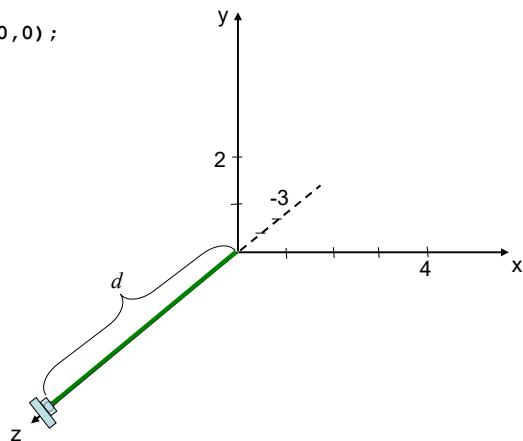
550



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 20

B-AI-V1 [551]



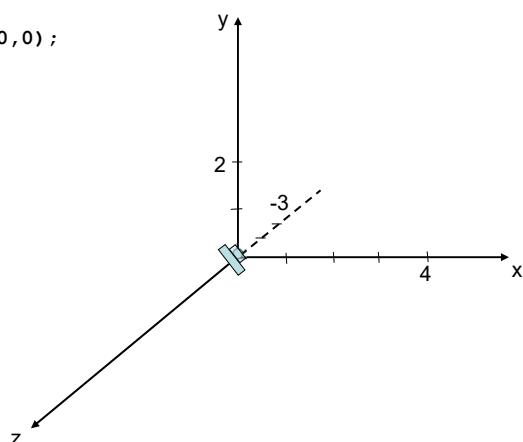
© R. Dörner

551

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 20

B-AI-V1 [552]



© R. Dörner

552



Beispiel: Umrechnung Welt- zu Viewkoordinaten

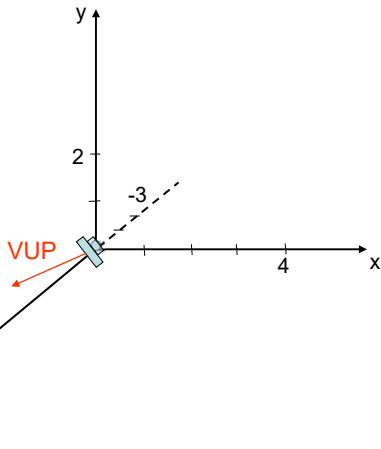
`lookAt(4, 2, -3, 0, 0, 0, 1, 0, 0);`

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 20

B-AI-V1 [553]

© R. Dörner

553

Beispiel: Umrechnung Welt- zu Viewkoordinaten

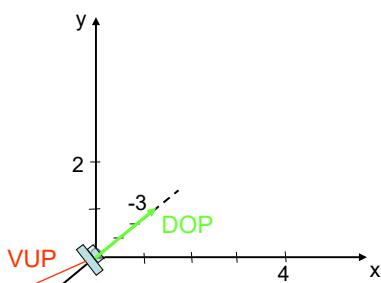
`lookAt(4, 2, -3, 0, 0, 0, 1, 0, 0);`

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 20

B-AI-V1 [554]

© R. Dörner

554



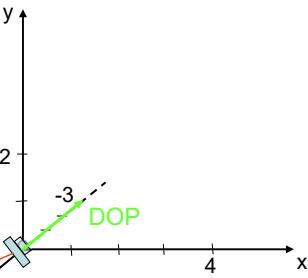
Beispiel: Umrechnung Welt- zu Viewkoordinaten

`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 Normalen-
vektor 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 20

B-AI-V1 [555]



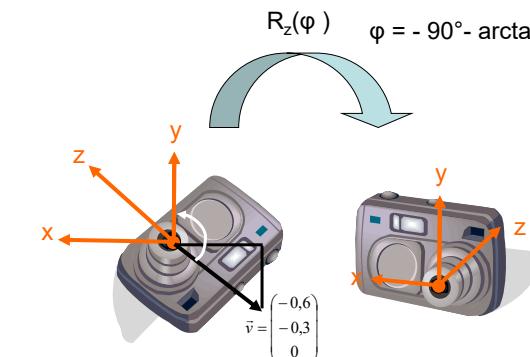
© R. Dörner

555

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. $R_z(\varphi)$



Derzeitige Situation

Gewünschte Zielsituation

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [556]



© R. Dörner

556



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

```
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]));
```

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [557]



© R. Dörner

557

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. $R_z(\varphi)$

$$M = R_z(\varphi) \cdot T(0, 0, -d) \cdot R_x(\beta) \cdot R_y(-(\alpha + 90^\circ))$$

Diese Befehle
bewirken
das Gleiche

```
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]));
```

Computergraphik – Prof. Dr. R. Dörner – WS 20

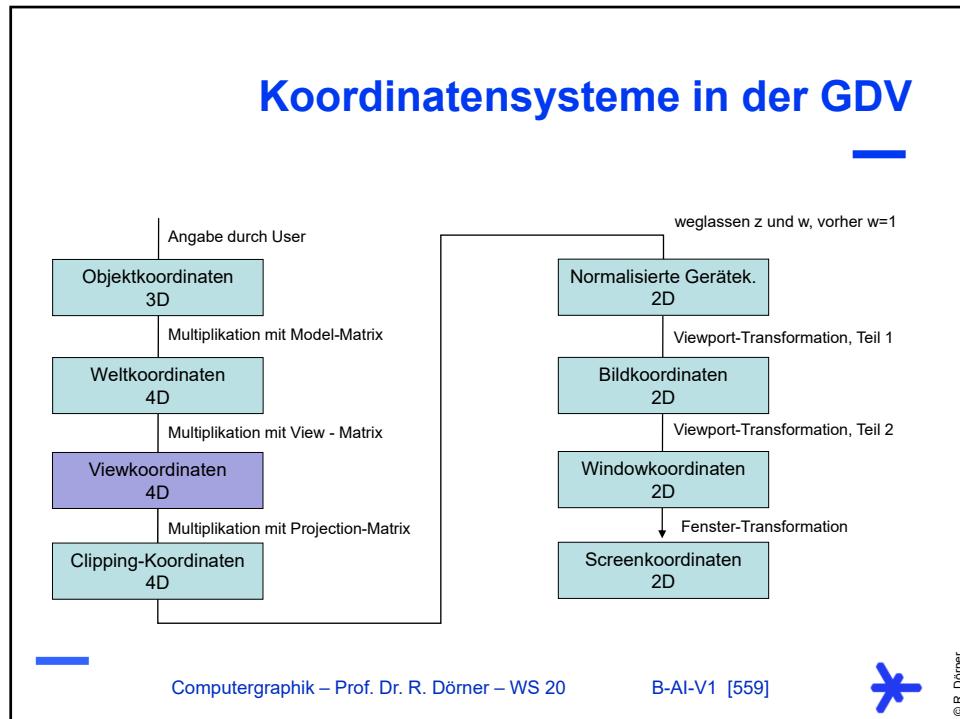
B-AI-V1 [558]



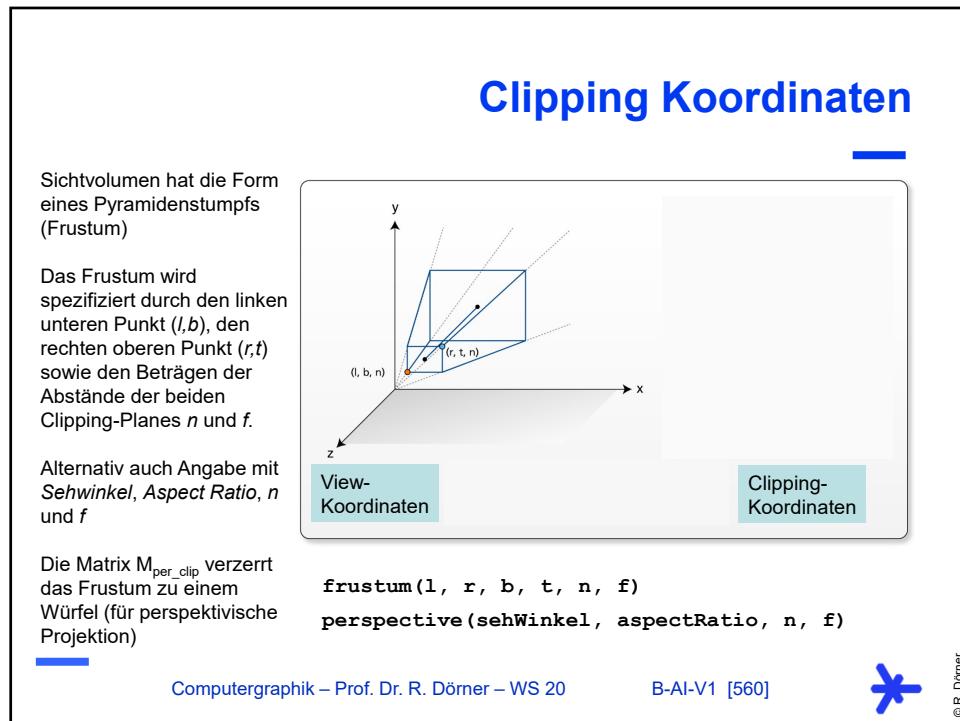
© R. Dörner

558





559



560



Clipping Koordinaten

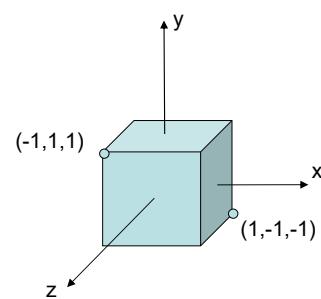
- Spezifikation des Sichtvolumens:
`left, right, bottom, top, | near |, | far |`
- Matrix für Wechsel von View- zu Clipping Koordinaten:

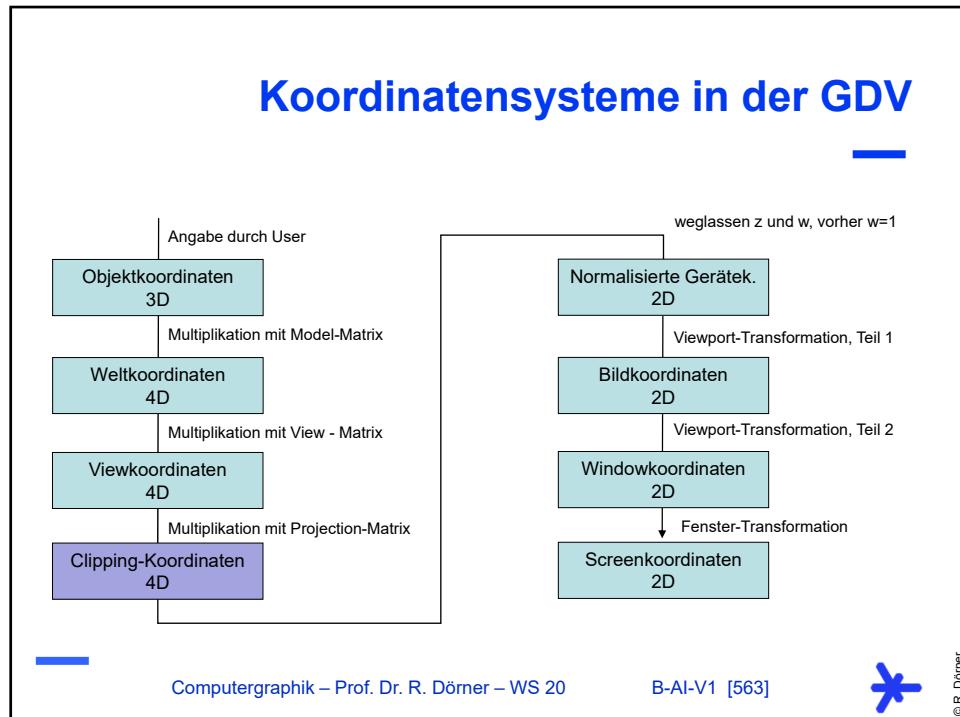
$$M_{\text{per_clip}} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$



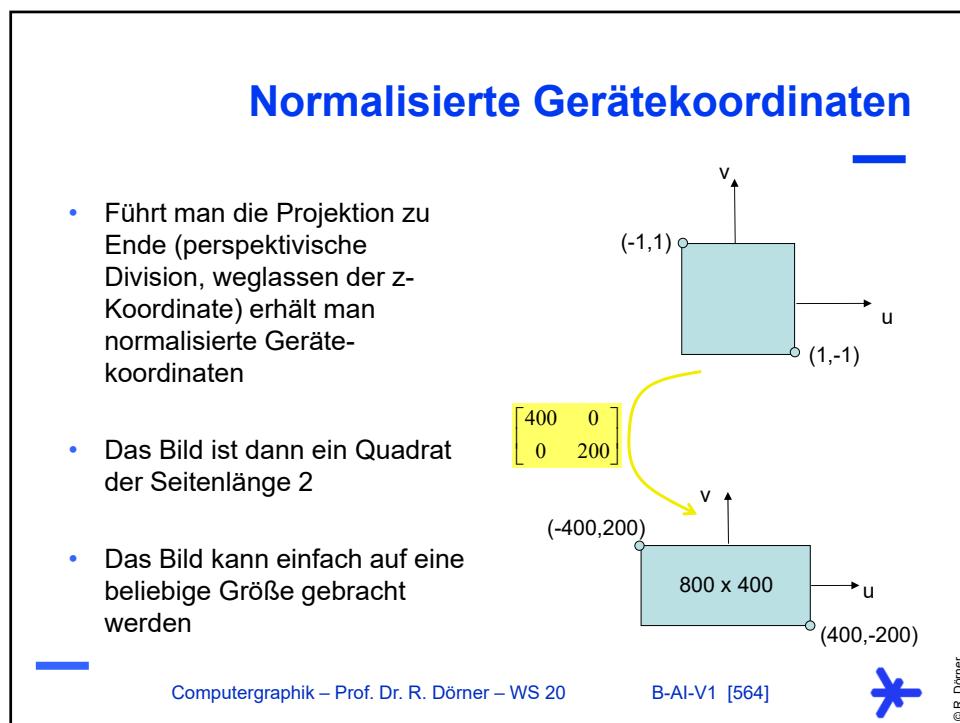
Clipping Koordinaten

- $M_{\text{per_clip}}$ 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



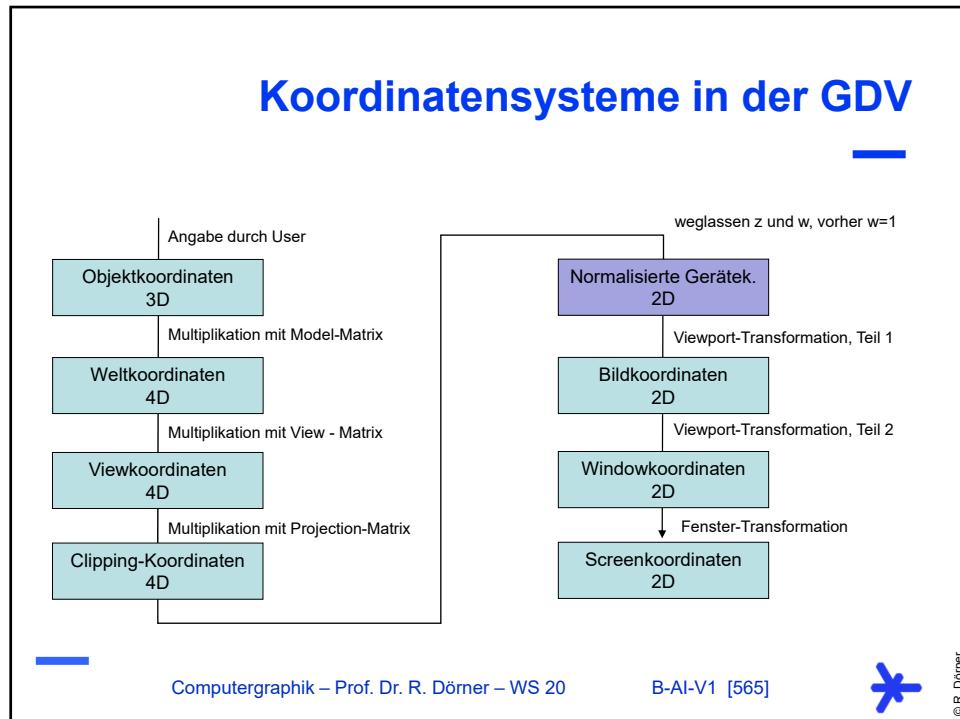


563

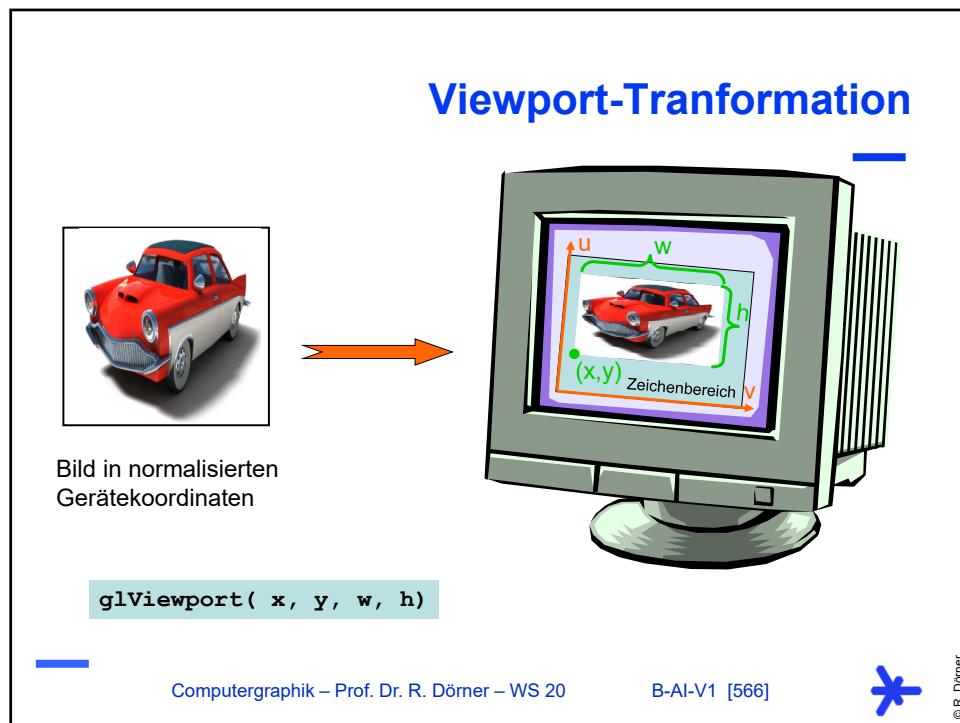


564



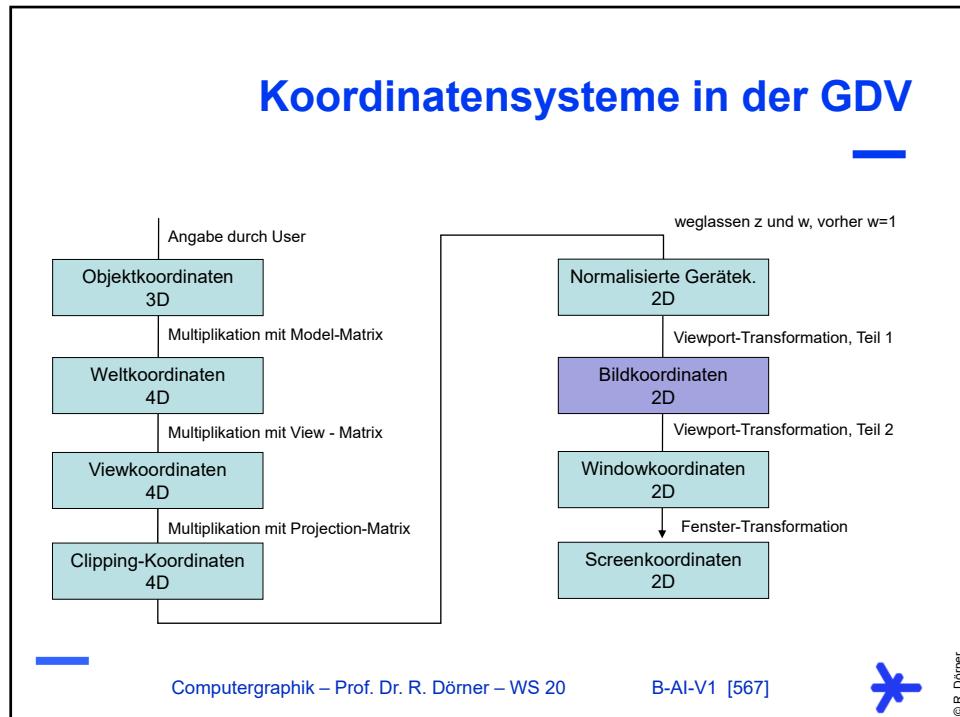


565

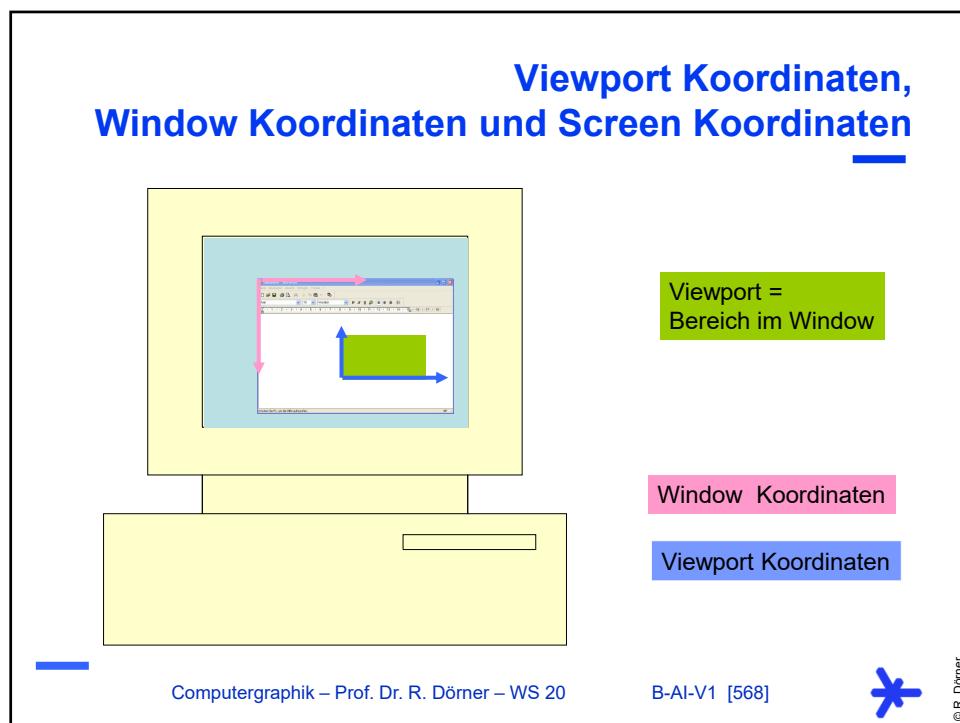


566



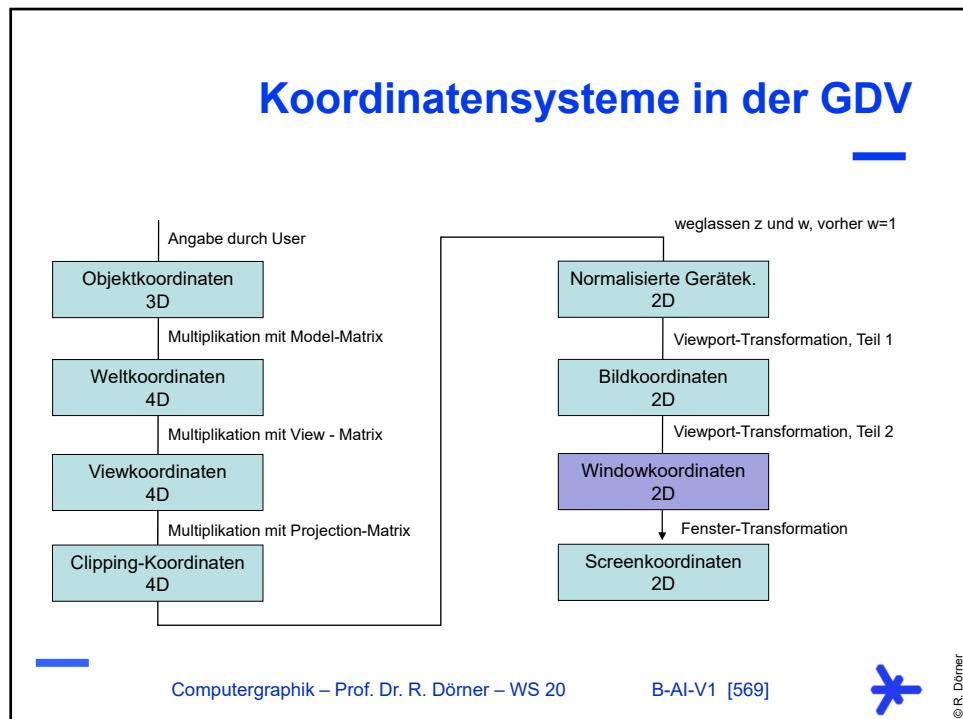


567

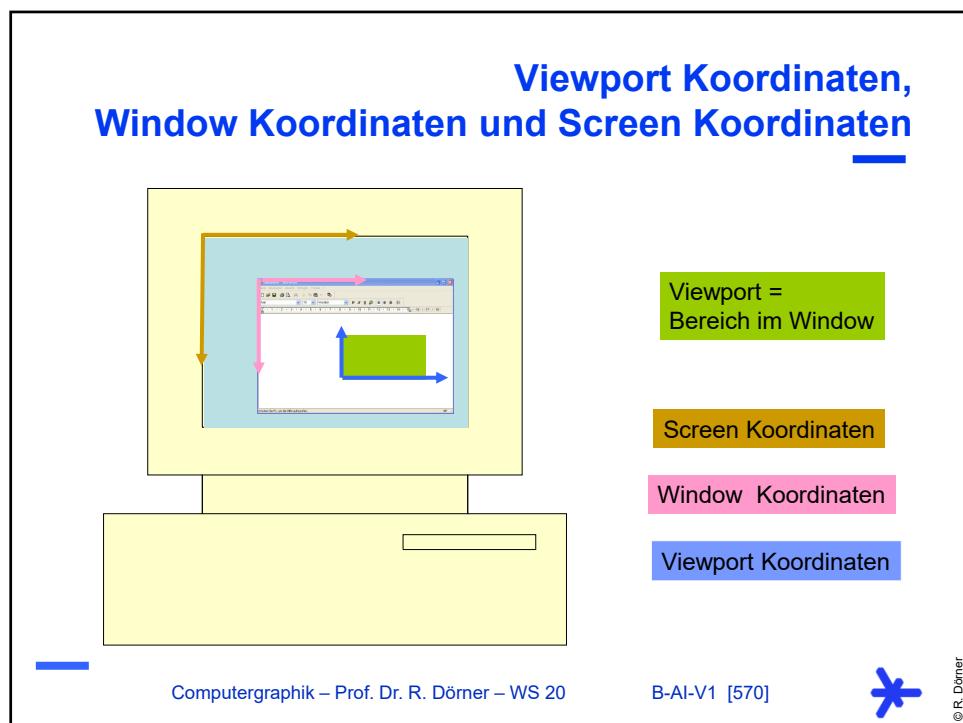


568



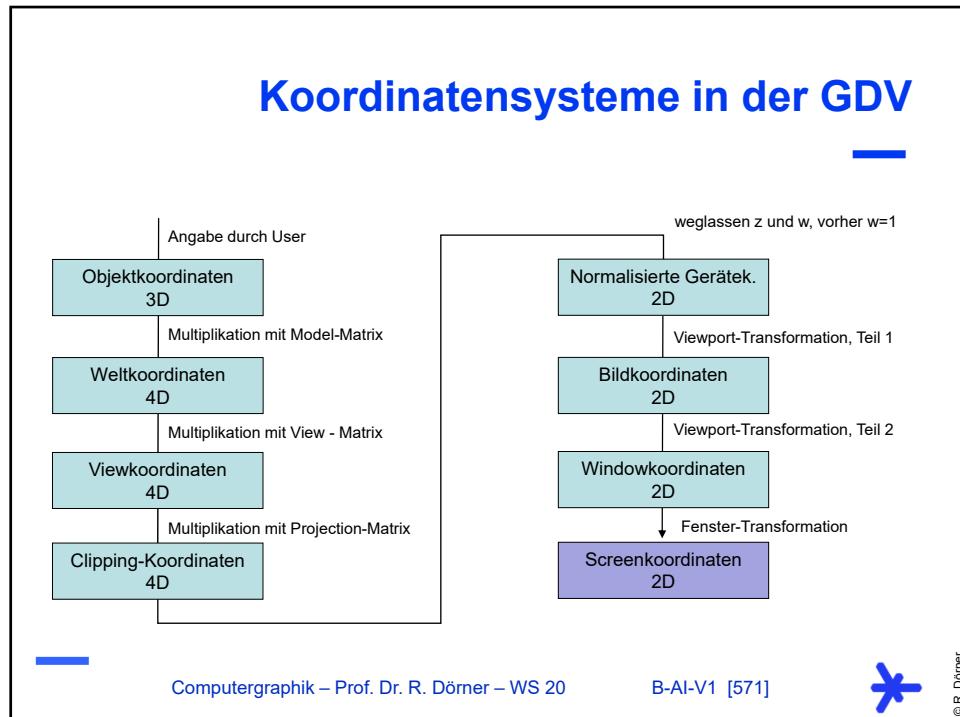


569

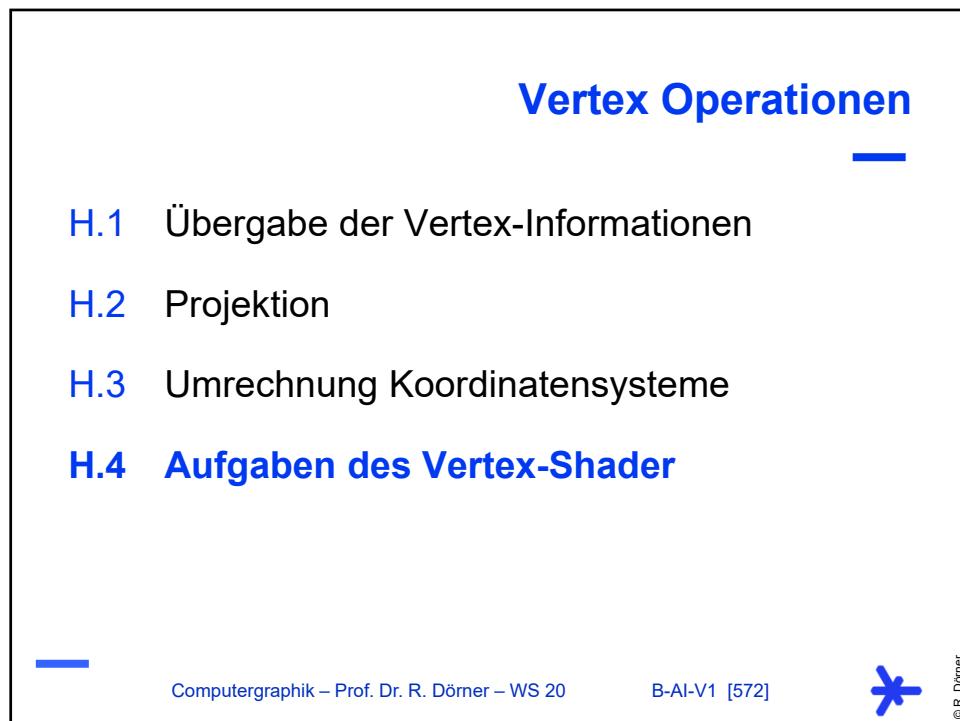


570





571

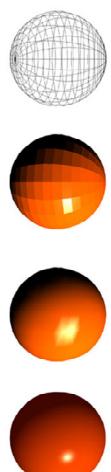


572



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



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [573]  © R. Dörner

573

GLSL Beispiel

```
uniform mat4 modelMat;
uniform mat4 viewMat;
uniform mat4 projectionMat;
attribute vec3 position;
attribute vec4 vColor;
varying vec4 fColor;

void main() {
    fColor = vColor;
    gl_Position =
        projectionMat * viewMat * modelMat *
        vec4(position, 1);
}
```

Übergabe der Matrizen
für diesen Eckpunkt: Position und Farbe
keine Beleuchtungsrechnung: der Farbwert wird einfach weitergereicht
Ergebnis des Vertex Shaders wird in vordefinierte Variable geschrieben
Übergabe des Wertes an Fragment-Shader
Konstruktor für 4D-Vektor, da 4x4 Matrix

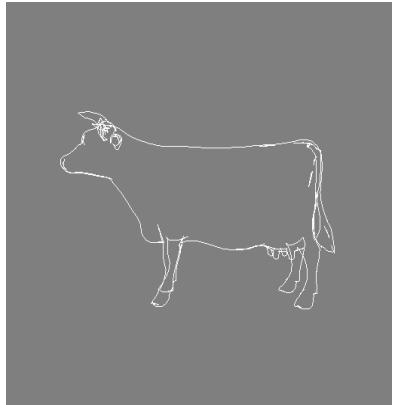
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [574] 

574



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

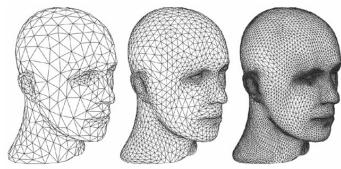
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [575]

© R. Dörner

575

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 20 B-AI-V1 [576]

© R. Dörner

576

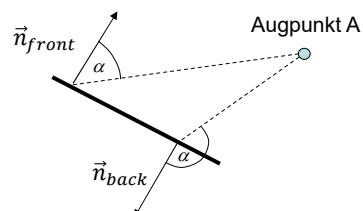


Teil I: Culling, Clipping & Rasterisierung

577

Culling

- Entfernen von Primitiven, die nicht auf dem Bild erscheinen können
- Grund: Rechenzeitersparnis
- View-Frustum-Culling
 - Alles außerhalb des Sichtvolumens entfernen
 - Einfach in Clipping Koordinaten durchzuführen
- Backface-Culling
 - Entfernen der abgewandten Flächen



578

Culling

- Entfernen von Primitiven, die nicht auf dem Bild erscheinen können
- Grund: Rechenzeiterersparnis
- View-Frustum-Culling
 - Alles außerhalb des Sichtvolumens entfernen
 - Einfach in Clipping Koordinaten durchzuführen
- Backface-Culling
 - Entfernen der abgewandten Flächen

\vec{n}_{front} \vec{n}_{back} \vec{a} \vec{p} α

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 20 B-AI-V1 [579]  © R. Dörner

579

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 $\Leftrightarrow |\alpha| < 90^\circ$

$-90^\circ < \alpha < 90^\circ \Leftrightarrow \frac{\langle \vec{n}_{front}, \vec{a} - \vec{p} \rangle}{|\vec{n}_{front}| \cdot |\vec{a} - \vec{p}|} > 0$

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)
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [580]  © R. Dörner

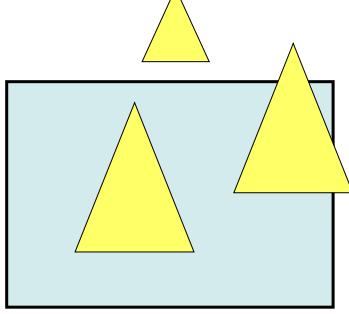
580



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

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [581]

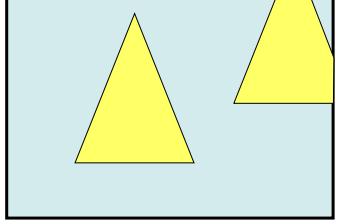
© R. Dörner

581

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

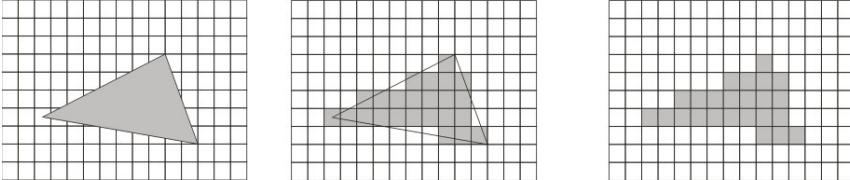
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [582]

© R. Dörner

582



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 20

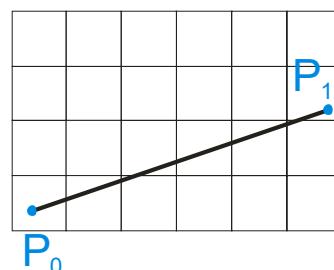
B-AI-V1 [583]

© R. Dörner

583

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 20

B-AI-V1 [584]

© R. Dörner

584



Bresenham's Algorithmus

aktueller Pixel Entscheidungspunkt
Nordost-Pixel
Ost-Pixel

Entscheidungsregel:
Liegt die Linie über dem Entscheidungspunkt, so wird der Nordost-Pixel gewählt, ansonsten der Ost-Pixel

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [585]

© R. Dörner

585

Bresenham's Algorithmus

```
void zeichneLinie(int x0, int y0, int x1, int y1) {
    int dy = y1 - y0; int dx = x1 - x0;
    int x = x0; int y = y0;
    int entscheidung = -dx;
    int schrittX = -2 * dx; int schrittY = 2 * dy;
    while(x <= x1) {
        setzePixel(x, y);
        x++;
        entscheidung = entscheidung + schrittY;
        if (entscheidung > 0) {
            y++;
            entscheidung = entscheidung + schrittX;
        } // Nordost-Pixel wählen
    }
}
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [586]

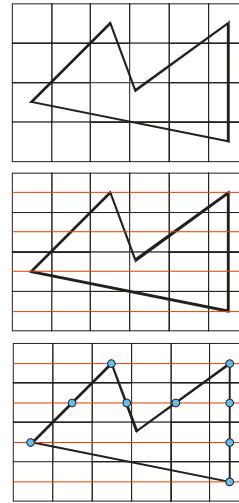
© R. Dörner

586



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 20

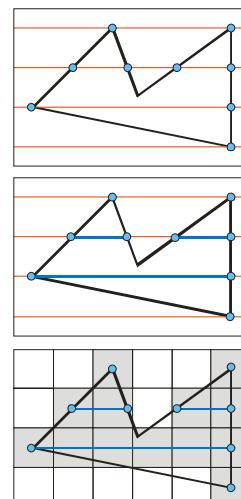
B-AI-V1 [587]

© R. Dörner

587

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



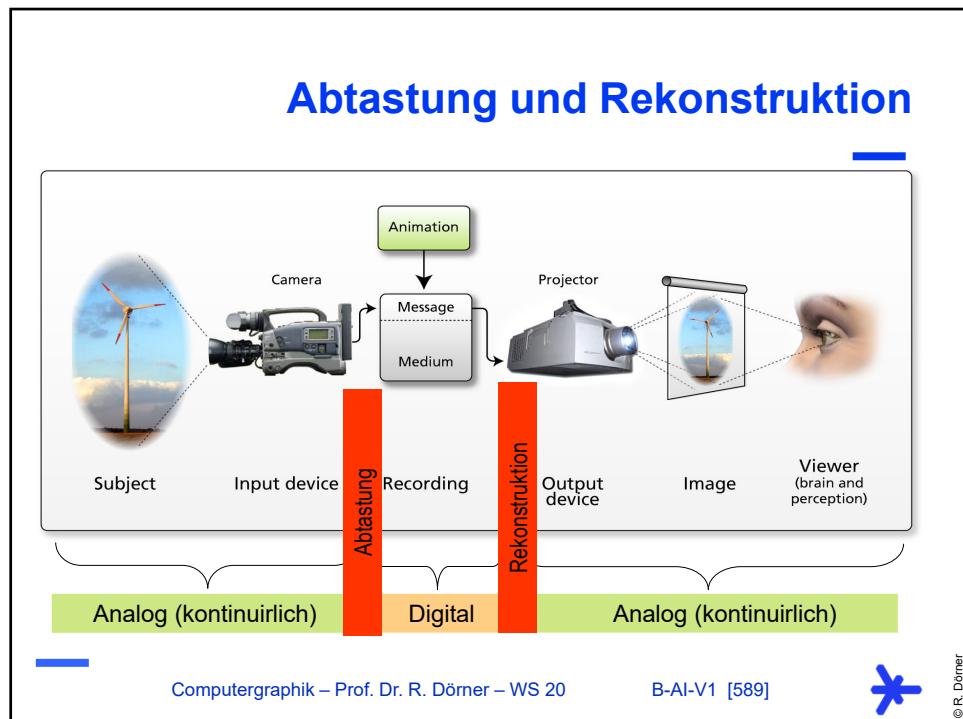
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [588]

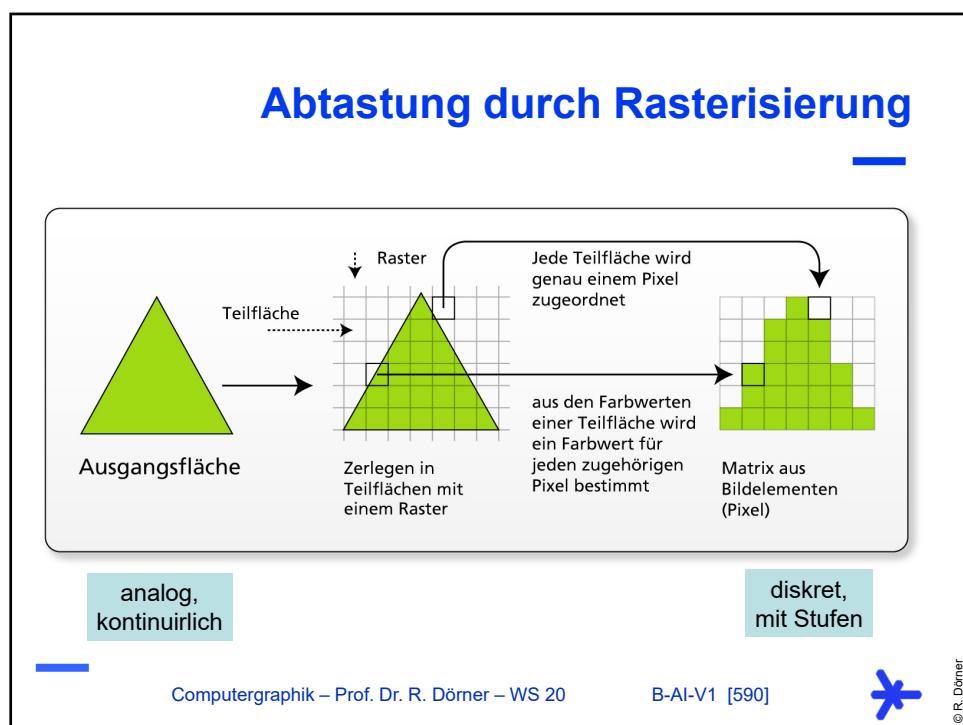
© R. Dörner

588





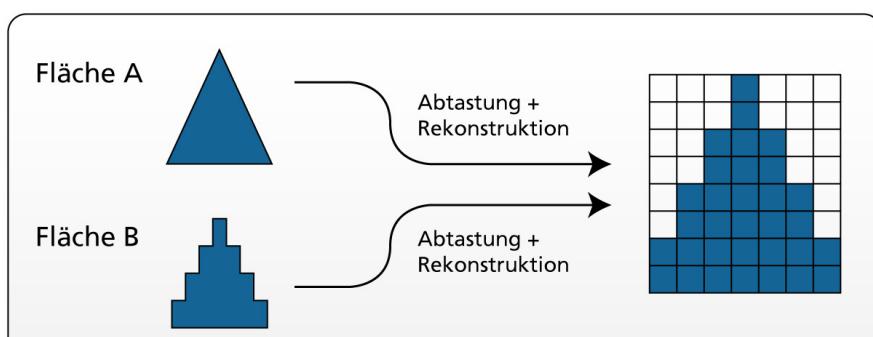
589



590



Aliase



Fläche A

Fläche B

Abtastung + Rekonstruktion

Abtastung + Rekonstruktion

Nyquist-Theorem gibt Kriterium vor, ob Aliase auftreten oder nicht.
Wird Kriterium verletzt, ist keine verlustfreie Rekonstruktion möglich.

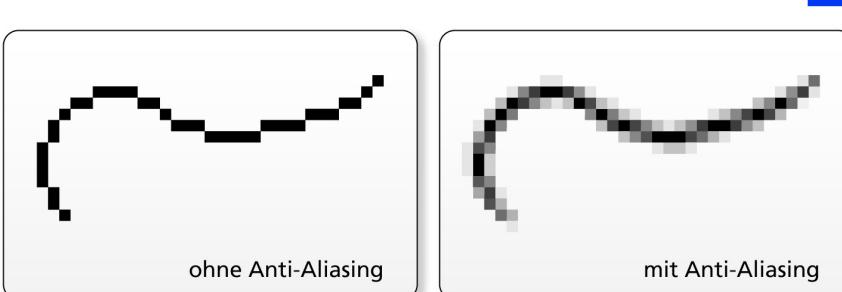
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [591]

© R. Dörner

591

Anti-Aliasing



ohne Anti-Aliasing

mit Anti-Aliasing

Verschiedene Anti-Aliasing Verfahren

- Filterung mit Tiefpass-Filter (Bild wird unschärfer)
- Intern mit höherer Auflösung arbeiten (Super-Sampling)

Computergraphik – Prof. Dr. R. Dörner – WS 20

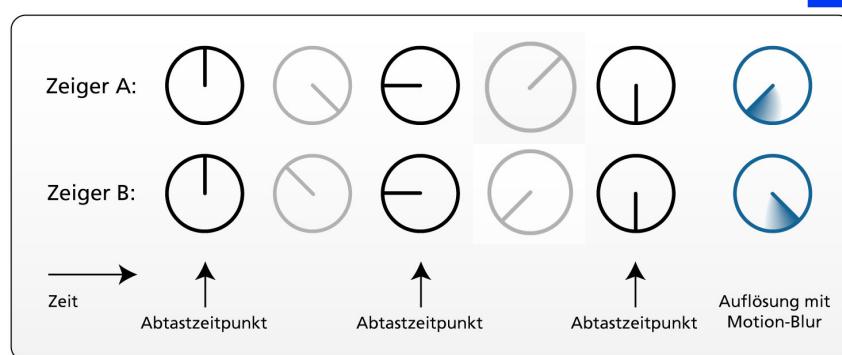
B-AI-V1 [592]

© R. Dörner

592



Zeitliches Anti-Aliasing



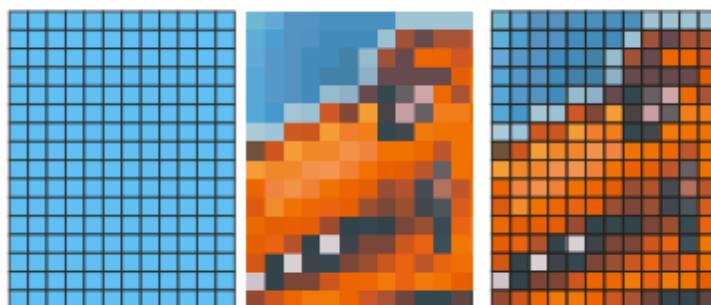
Aliasing-Effekte treten auch durch diskrete Frames auf

- Objekte bewegen sich scheinbar rückwärts
- Kleine Bewegungen können zu Pixel – Flackern führen



Filterung von Texturen

Fall 1: Bild und Textur sind gleich groß:

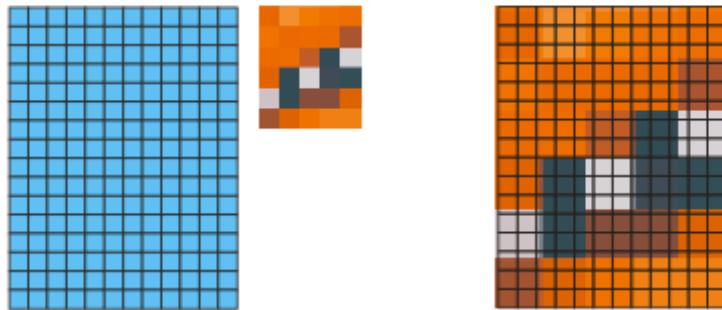


Zu einem Bildpixel gehört ein Texturpixel



Filterung von Texturen

Fall 2: Bild ist größer als Textur:



Ein Texturpixel wird auf mehrere Bildpixel vergrößert

Filterung von Texturen

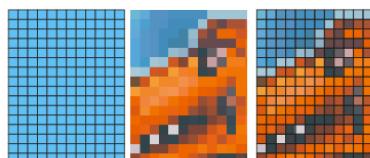
Fall 3: Bild ist kleiner als Textur:



*Mehrere Texturpixel werden verkleinert,
um in ein Bildpixel zu passen.*

Filterung von Texturen

Fall 1: Bild und Textur sind gleich groß:



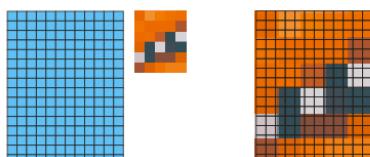
Zu einem Bildpixel gehört ein Texturpixel

Fall 3: Bild ist kleiner als Textur:



Mehrere Texturpixel werden verkleinert, um in ein Bildpixel zu passen.

Fall 2: Bild ist größer als Textur:

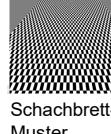


Ein Texturpixel wird auf mehrere Bildpixel vergrößert

Vermeidung von „pixelig“ wirkenden Texturen durch „glätten“ mit einem sogenannten „Filter“.



Artefakte und Filterung



–

ungefiltert

Nearest-Filter

Nearest-Filter,
versch. Größen

Linear-Filter,
versch. Größen



Textur ohne Filterung



Quelle:
Watt / Pollicarpo
3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [599]



© R. Dörner

599

Textur mit Filterung



Quelle:
Watt / Pollicarpo
3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [600]

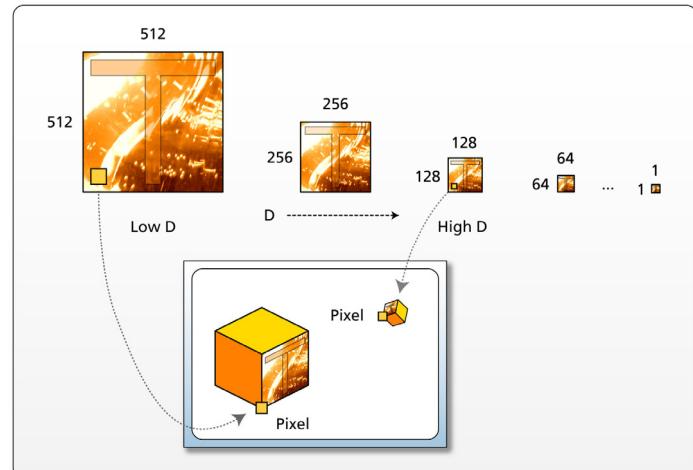


© R. Dörner

600



Textur-Filterung und MipMaps



Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [601]

© R. Dörner

601

Speichern von MipMaps



Quelle:
Watt / Polcarpo
3D Games

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [602]

© R. Dörner

602



—
Teil J:
Fragment Operationen
—

603

Fragment Operationen

- J.1** Vom Fragment zum Pixel
- J.2** Fragment Shader
- J.3** Verdeckungsrechnung

604



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 20

B-AI-V1 [607]



© R. Dörner

607

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 20

B-AI-V1 [608]



© R. Dörner

608



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

```
Übergabe vom Vertex-Processing
varying vec4 fColor;

void main() {
    gl_FragColor = 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
```

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [611]

611

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 $f = 0.0$;

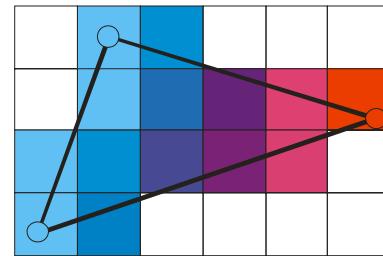
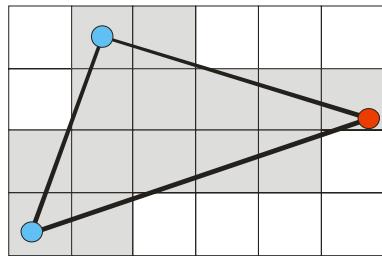
Im Vertex Shader:
Dieser Vertex setzt $f = 1.0$;

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [612]

612



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 Eckpunktdaten 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(){
    gl_FragColor =
        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

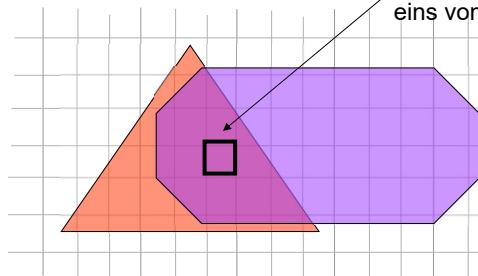
J.2 Fragment Shader

J.3 **Verdeckungsrechnung**



Verdeckungsrechnung

In diesem Pixel liegen zwei Fragmente:
eins vom Dreieck, eins vom Sechseck



Objekte verdecken sich: bei undurchsichtigen Objekten sollte die Farbe des Fragments gewählt werden, das von dem Objekt stammt, das am nächsten zum Betrachter ist.

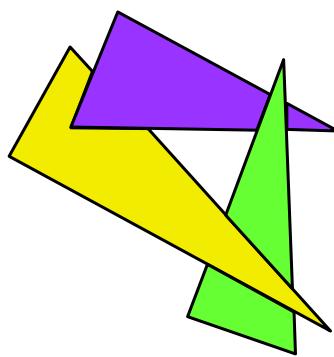
Painter-Algorithmus

- 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 20

B-AI-V1 [619]

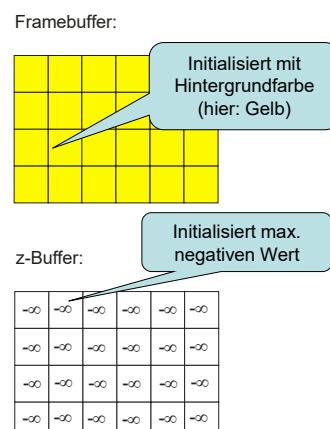


© R. Dörner

619

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 20

B-AI-V1 [620]



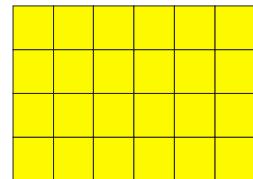
© R. Dörner

620



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 20

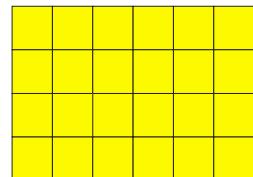
B-AI-V1 [621]

© R. Dörner

621

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 20

B-AI-V1 [622]

© R. Dörner

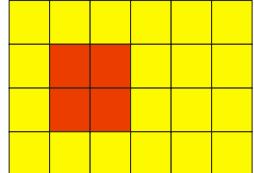
622



z-Buffer Algorithmus

Der Hintergrund wird immer überschrieben.

Framebuffer:



z-Buffer:

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	-0.5	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [623]

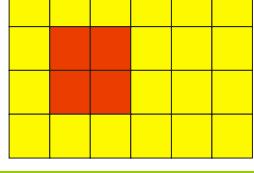
© R. Dörner

623

z-Buffer Algorithmus

Wollen nun diese 3 Fragmente eintragen.

Framebuffer:



z-Buffer:

$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.4	-0.5	$-\infty$	$-\infty$
$-\infty$	-0.2	-0.5	-0.5	$-\infty$	$-\infty$
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Je größer der z-Wert desto näher an der Kamera desto eher ist es sichtbar, weil es andere verdeckt.

z-Wert des Fragments ist größer. Also wird es eingetragen.

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [624]

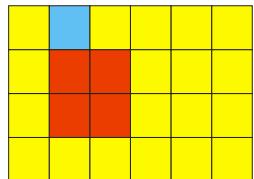
© R. Dörner

624



z-Buffer Algorithmus

Framebuffer:



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-Wert des Fragments ist kleiner.
Also wird es nicht beachtet.**

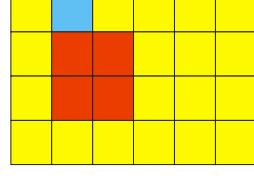
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [625]

© R. Dörner

625

z-Buffer Algorithmus

Framebuffer:



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-Wert des Fragments ist größer.
Also wird es eingetragen.**

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [626]

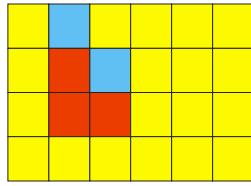
© R. Dörner

626

z-Buffer Algorithmus

Das Resultat ist gleich, egal ob zuerst das rote Viereck und dann das blaue Dreieck gerendert wird, oder umgekehrt.

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 20 B-AI-V1 [627]

© R. Dörner

627

z-Buffer Algorithmus

Initialisiere Framebuffer $fb[x,y]$ mit der Hintergrundfarbe
Initialisiere z-Buffer $zb[x,y]$ mit -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 20 B-AI-V1 [628]

© R. Dörner

628



z-Buffer Algorithmus in WebGL

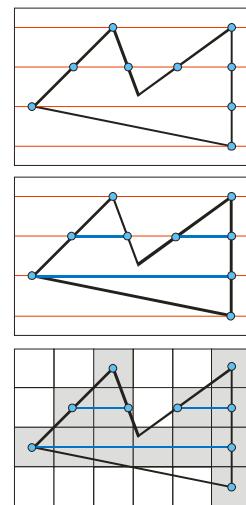
- z-Buffer wird automatisch angelegt, `gl_FragDepth` wird automatisch mit Wert belegt
- Spezielle GL-Befehle für den z-Buffer
- Near-Plane und Far-Plane möglichst dicht aneinander, damit keine Genauigkeitsprobleme im z-Buffer auftreten

```
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
- Rechenzeitersparnis 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

631

GDV Anwendungen

- K.1 Grafik-APIs
- K.2 Autorenwerkzeuge
- K.3 Anwendungsbeispiele



632



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)
 - 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 20 B-AI-V1 [635]



© R. Dörner

635

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.scenegraph.js:26](#)
Module: Scene

Represent the scene graph it self.

Constructor

CGSSceneGraph (canvas , context)
Defined in: [src/classes.scenegraph.js:26](#)

Parameters:

- canvas: HTMLElement
a handler to the canvas HTML element
- context: CanvasRenderingContext2D
context to render on

[Index](#) [Methods](#) [Properties](#)

Item Index

Methods

addNode	pickNodes
deselectAll	removeNode
deselectNode	render
invalidateTheme	selectNode
pickNode	setCanvasDimension

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [636]



636



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

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [637]



© R. Dörner

637

jMonkey

- Szenengraph-API für Java, weitere Funktionen (3D Games)
- 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 20

B-AI-V1 [638]



638



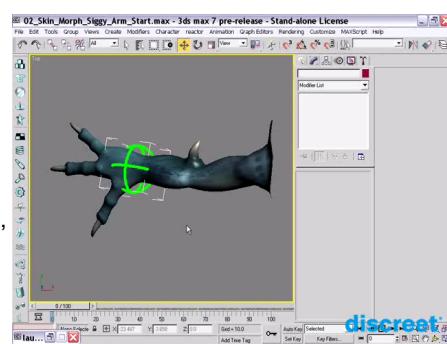
GDV Anwendungen

- K.1** Grafik-APIs
- K.2** Autorenwerkzeuge
- K.3** Anwendungsbeispiele

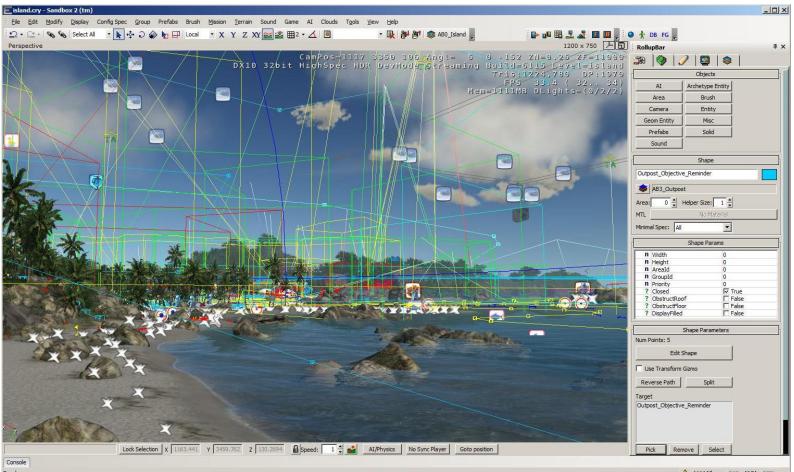


Grafik Werkzeuge und Skriptsprachen

- 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, ...



Autorenwerkzeuge von Game Engines



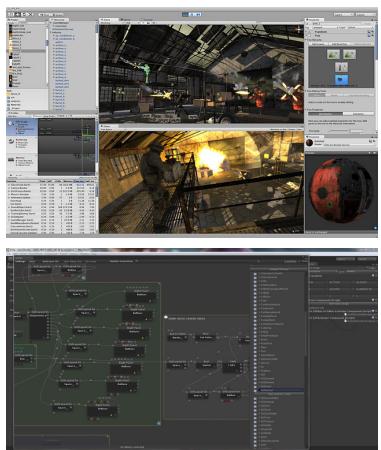
Quelle: crytek.com

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [641]

641

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 20 B-AI-V1 [642]

642



GDV Anwendungen

K.1 Grafik-APIs

K.2 Autorenwerkzeuge

K.3 Anwendungsbeispiele

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [643]



© R. Dörner

643

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?



Lie. Blaue - Terragen-V0.8.44 - 07/2002

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [644]



© R. Dörner

644



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 20

B-AI-V1 [645]



© R. Dörner

645

Visualisierung



Quelle:
Deutscher Wetterdienst

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [646]

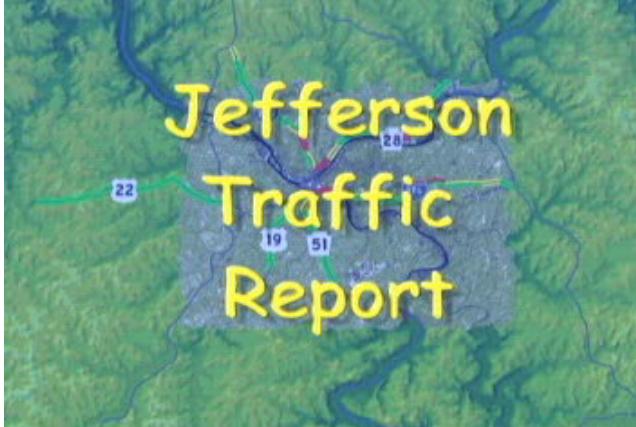


© R. Dörner

646



Visualisierung

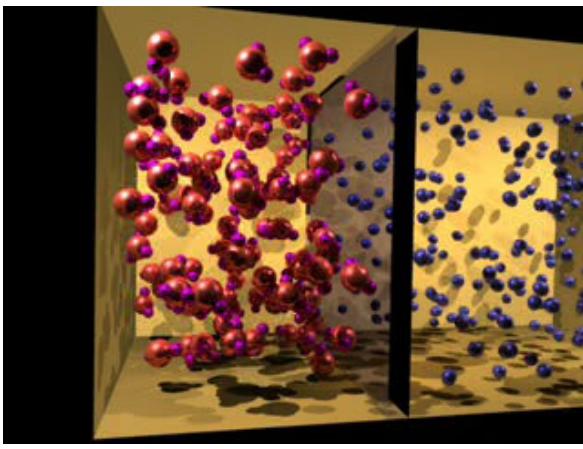


Quelle:
Andre Guezic:
3D Traffic Visualization
in Real Time
www.trianglesoftware.com

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [647]  © R. Dörner

647

Visualisierung



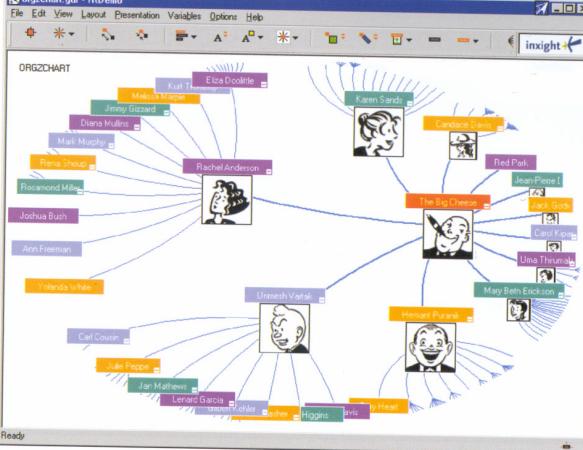
Quelle:
Brian Mirtich
Timewarp Rigid Body Simulation
Siggraph 2000

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [648]  © R. Dörner

648



Visualisierung



The screenshot shows a network visualization tool window titled "ORG2CHART". The window contains a graph with various nodes, each representing a person with a small portrait and a name label. The nodes are color-coded and connected by lines representing relationships. The names visible include Elias Doherty, Karen Sands, Candace Lewis, Reid Paul, Sean Pease, Karen Goss, Jimmy Goss, Diana Muller, Mark Murphy, Brian O'neill, Rosamond Miller, Joshua Bush, Ann Freeman, Linda White, Carl Cullen, John Pepple, Jan Mathew, Leonard Garcia, Steven Wohle, Jolene, Lewis, and Helen. The background is white, and the overall layout is a radial or hierarchical structure.

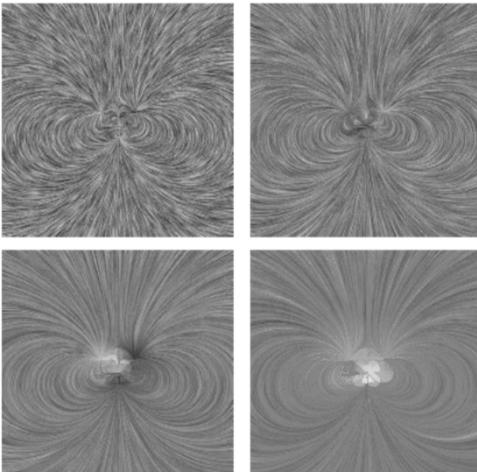
Quelle:
Robert Spence,
Information Visualization

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [649]

© R. Dörner

649

Visualisierung



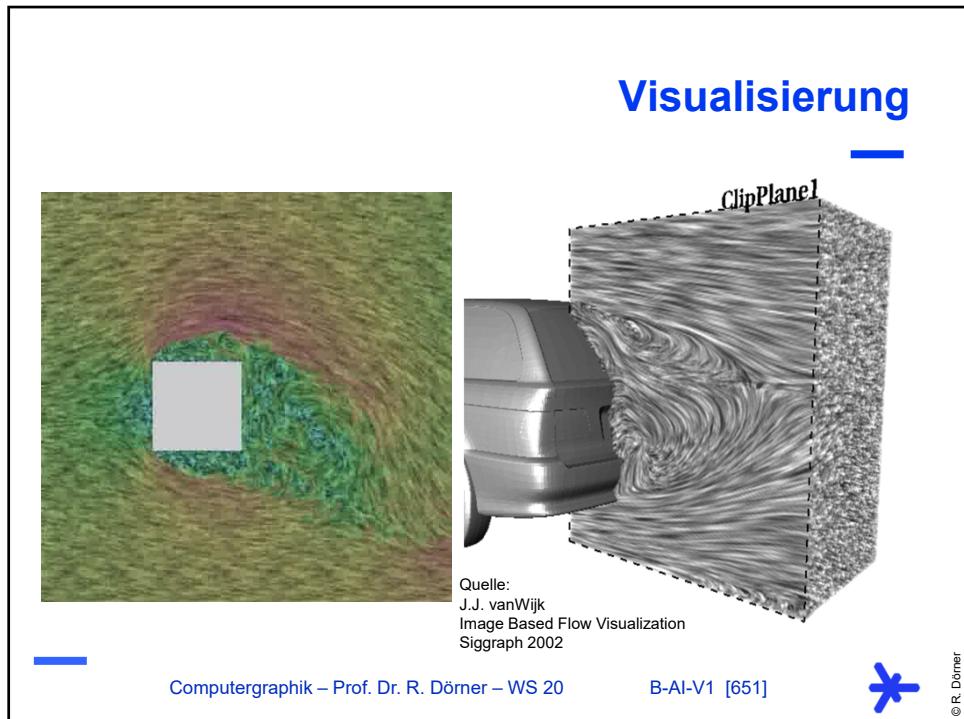
The block contains four grayscale images arranged in a 2x2 grid. Each image displays a complex, swirling pattern of lines or streaks, creating a sense of motion or flow. The patterns are dense and radiate from a central point, resembling a star or a magnet. The images are likely visualizations of network data or flow fields.

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [650]

© R. Dörner

650





651



652



Visualisierung



Quelle:
GCF

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [653]

© R. Dörner

653

Visualisierung



Quelle:
Fraunhofer IGD

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [654]

© R. Dörner

654



Visualisierung

Virtual Colonoscopy

camera path

artificial horizon

(c) Dirk Bartz, Univ. of Tübingen

Quelle:
Dirk Bartz, Universität Tübingen

Quelle:
Visible Human Project,
National Library of Medicine,
Maryland, USA

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [655]

© R. Dörner

655

Visualisierung und GIS

File Edit View Theme Graphics Window Help

Water Resources

Sample Points

LANDS

WATER BODIES

WATERBORN

water_monitoring

Lake Nipigon

Results of Sampling

NITRATE FLUORIDE

Sample 7 Sample 6 Sample 5

Attributes of Sampling Points

SAMPLE	CAPTURE	LONGITUDE	NITRATE	FLUORIDE
7	43013	110442	0.2	5.3
4	43001	110544	0.3	0.2
5	43000	110544	0.3	0.2
1	43047	1102086	0.5	0.2
2	43048	1102086	0.7	0.2
3	43105	1101460	0.3	0.2
4	43104	1102154	0.1	0.3

Quelle:
www.gis.com

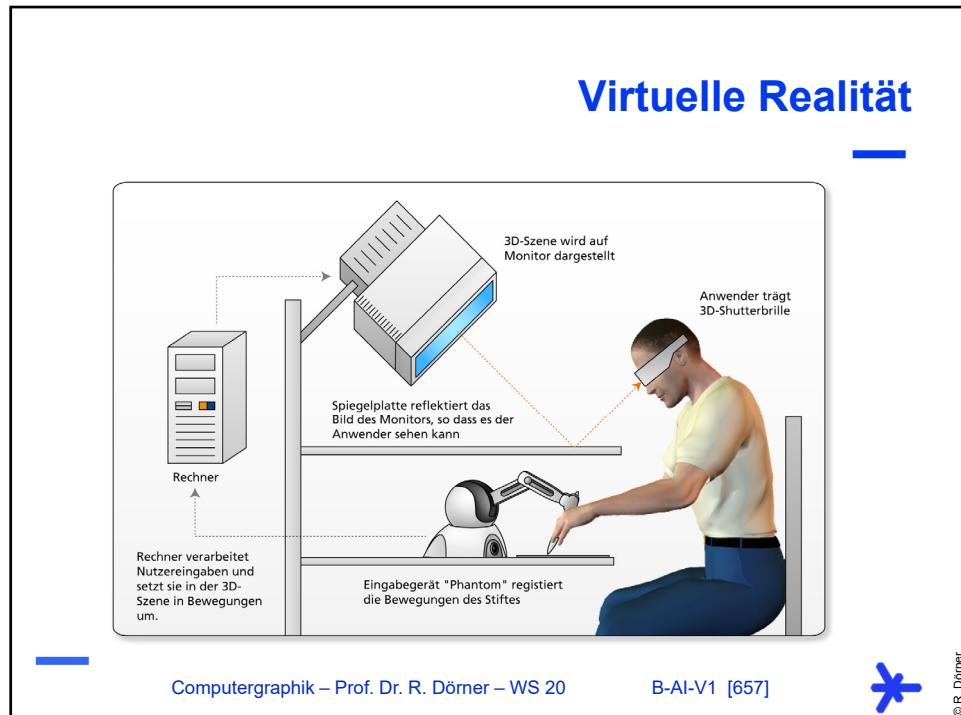
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [656]

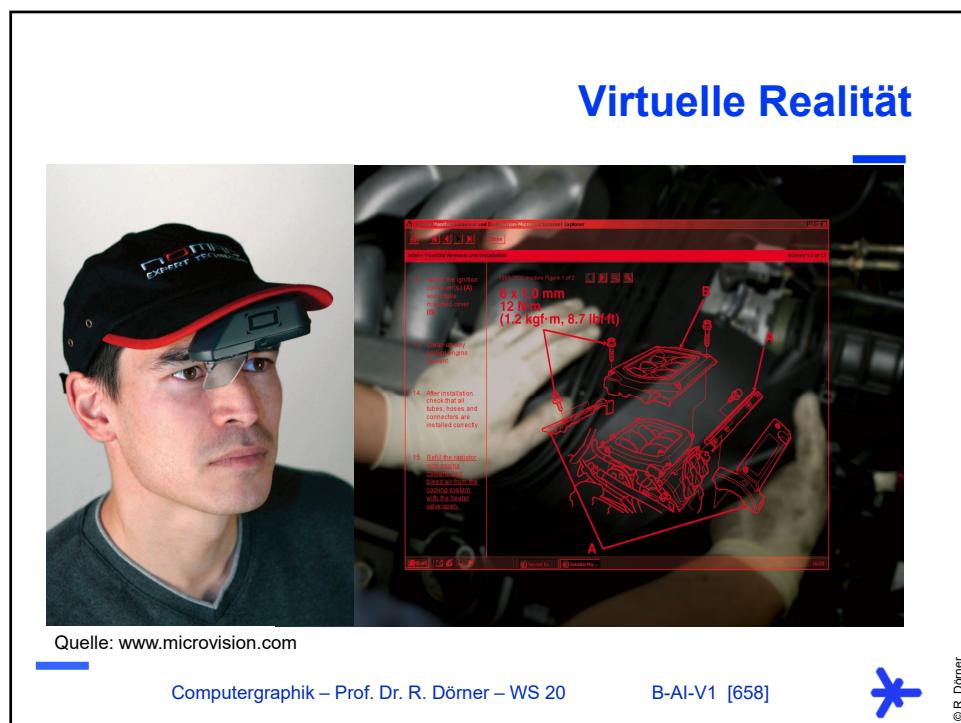
© R. Dörner

656





657



658



Tiled Wall / Stereodarstellung



Abb. 5.7 Tiled Wall aus Monitoren am Beispiel des Stony Brook's Reality Deck

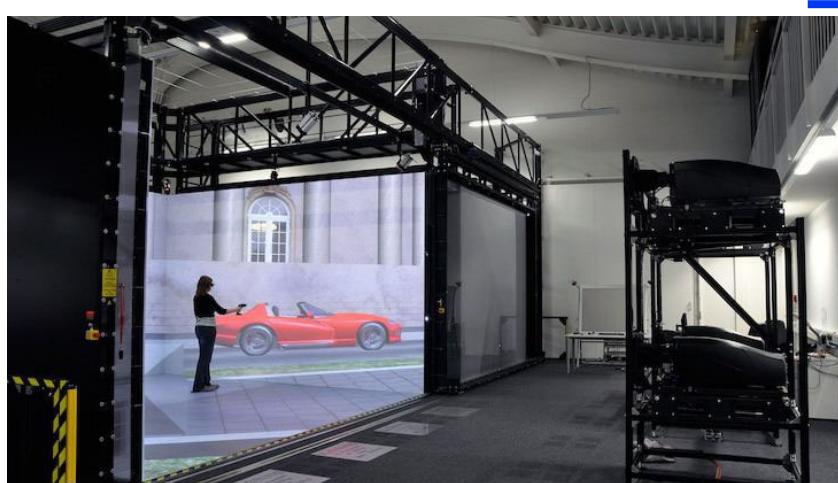
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [659]



659

CAVE



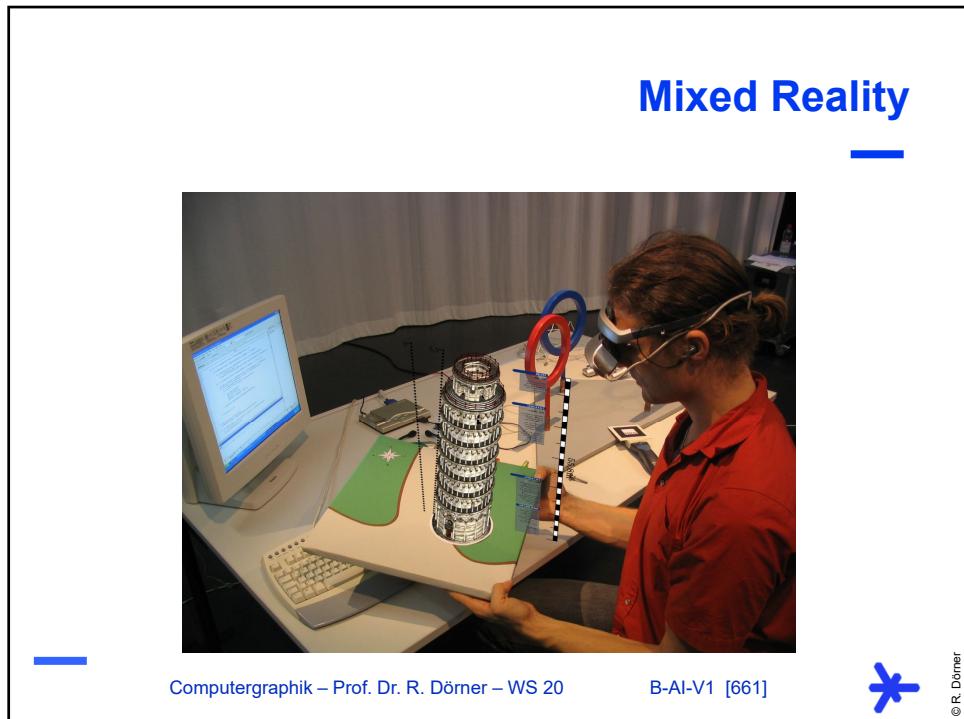
Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [660]

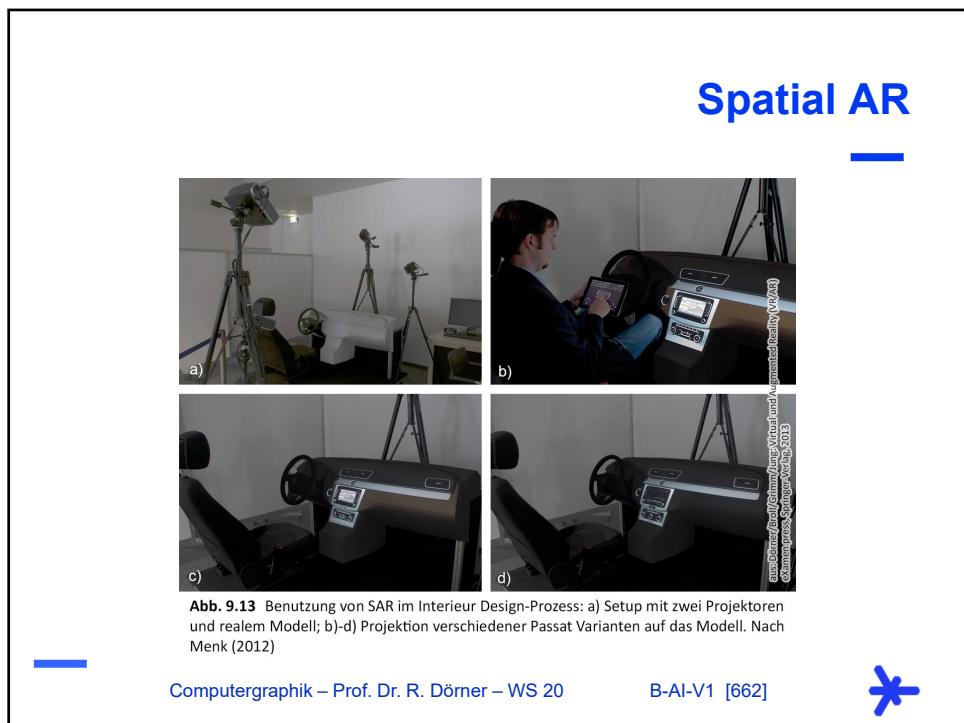


660





661



662



Mobile AR

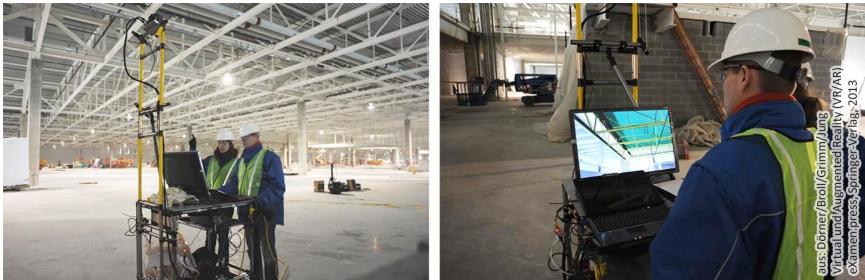


Abb. 9.15 Einsatz des entwickelten mobilen AR-Systems zur Untersuchung eines Hallenneubaus
(Quelle: Volkswagen AG)

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [663]

663

Magic Lens

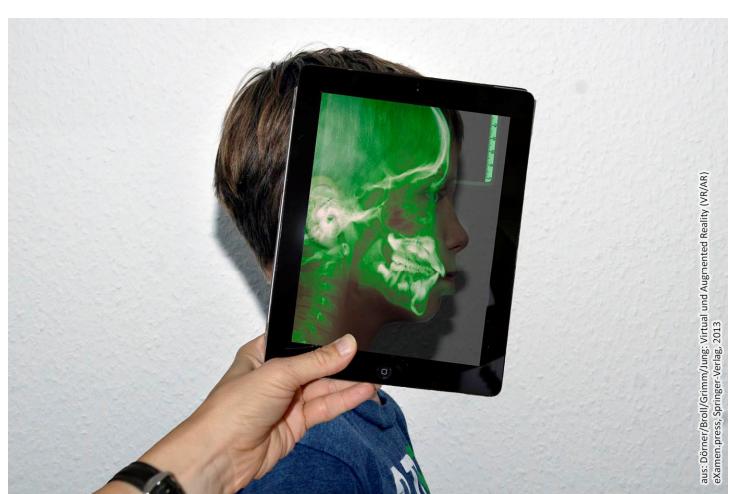


Abb. 8.6 Beispiel einer Magic Lens

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [664]

664



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 20 B-AI-V1 [665]

665

Mixed Reality

Face Tracking Live Demo

L. Vacchetti, V. Lepetit, P. Fua

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [666]

666



Computer Games



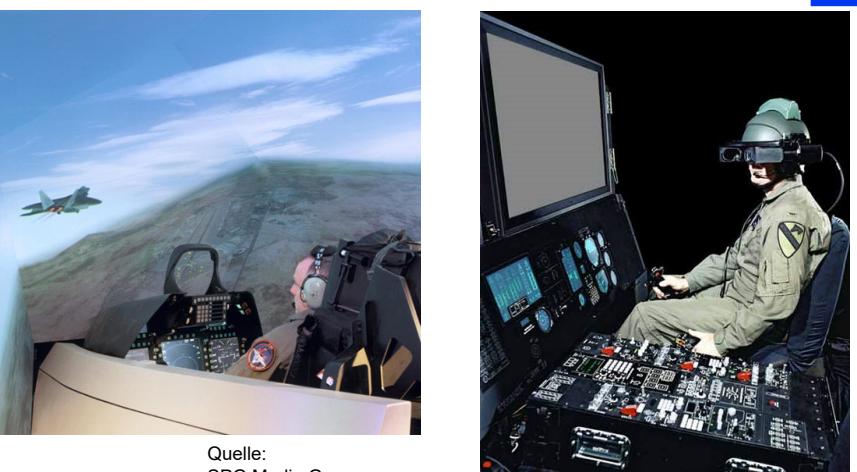
Quelle:
Vivendi, Rockstar Games, EIDOS

Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [667]

© R. Dörner

667

Learning & Training



Quelle:
SPG Media Group

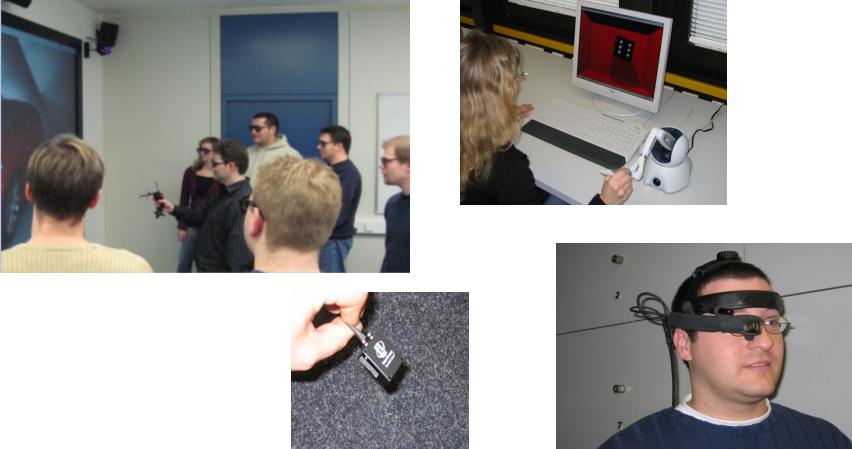
Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [668]

© R. Dörner

668



Visualisierungslabor



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [669]

669

Multi-Touch Displays



Computergraphik – Prof. Dr. R. Dörner – WS 20 B-AI-V1 [670]

670



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 20

B-AI-V1 [673]



© R. Dörner

673

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 20

B-AI-V1 [674]



© R. Dörner

674



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 20

B-AI-V1 [675]



© R. Dörner

675

ENDE

Computergraphik – Prof. Dr. R. Dörner – WS 20

B-AI-V1 [676]



© R. Dörner

676

