BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Computergrafik II
Transformationen und Szenengraph

Bachelor Medieninformatik
Wintersemester 2011

Prof. Dr.-Ing. Hartmut Schirmacher
http://schirmacher.beuth-hochschule.de
hschirmacher@beuth-hoschule.de

Gliederung

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

- Transformationen und Matrizen (Wdh.)
  - Rechnen mit Matrizen
  - 3D-Transformationen
  - Zusammengesetzte Transformationen
  - Umrechnung zwischen Koordinatensystemen
- Scenegraph
  - Koordinatensysteme: Modell, Kamera und Welt
  - Hierarchisches Modellieren
  - Scenegraph-Semantik
  - Traversierung und Zusammensetzen von Transformationen
  - Raytracing mit Scenegraph
  - Implementierung und das Kompositum-Muster

Errata vergangener Vorlesungen

Schlick-Approximation

- Schlick-Approximation[1] der Fresnel-Gleichungen
  - Speziell für die Verwendung in der Computergrafik
  - Einfach zu berechnen und ausreichend gut
  - R = Relektionsfähigkeit, T = Transmissionsfähigkeit

$$
\begin{aligned}
R &= R_0 + (1 - R_0)(1 - \cos^2\theta_i)^5, \\
T &= 1 - R, \\
R_0 &= \left(\frac{\eta_2 - \eta_1}{\eta_2 + \eta_1}\right)^2 .
\end{aligned}
$$

  - Nochmal zur Erinnerung: T = 0 und R = 1 wenn folgende Bedingung verletzt ist:

$$\left(\frac{\eta_1}{\eta_2}\right)^2 (1 - \cos^2\theta_i) \leq 1$$

1) Christophe Schlick, *A Customizable Reflectance Model for Everyday Rendering*, Fourth Eurographics Workshop on Rendering, 1993.

Transformationen und Matrizen

Rechnen mit Matrizen (1)

- Matrix $\mathbf{A}$ mit m Zeilen und n Spalten
  - Element a_{ij} ist in Zeile i, Spalte j

$$\mathbf{A}_{(m,n)} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- Transponierte $\mathbf{A}^{\mathrm{T}}$
  - Vertausche Zeilen mit Spalten

$$\mathbf{A}^{\mathrm{T}}_{(m,n)} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

- Beispiel:

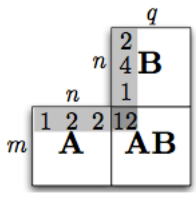
$$\mathbf{A} = \begin{bmatrix} 8 \\ 23 \\ 14 \\ 1 \end{bmatrix} \equiv \mathbf{A}^{\mathrm{T}} = \begin{bmatrix} 8 \\ 23 \\ 14 \\ 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 8 & 23 & 14 & 1 \end{bmatrix}$$

Rechnen mit Matrizen (2)

- Addition zweier Matrizen
  - Elementweise Addition $(\mathbf{A}+\mathbf{B})_{ij} = a_{ij} + b_{ij}$
  - Eigenschaften:
    - $(\mathbf{A}+\mathbf{B})+\mathbf{C} = \mathbf{A}+(\mathbf{B}+\mathbf{C})$ (assoziativ)
    - $\mathbf{A}+\mathbf{B} = \mathbf{B}+\mathbf{A}$ (kommutativ)
- Multiplikation zweier Matrizen
  - Berechnung:

$$\mathbf{C}_{(m,q)} = \mathbf{A}_{(m,n)}\mathbf{B}_{(n,q)} \quad \Rightarrow \quad \mathbf{C}_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

  - Zeilen- und Spaltenanzahl müssen kompatibel sein
  - „Zeile der ersten Matrix, Spalte der zweiten Matrix“
  - Eigenschaften:
    - $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ (assoziativ)
    - $\mathbf{AB} \neq \mathbf{BA}$ (nicht kommutativ)



Falk'sches Schema

Rechnen mit Matrizen (3)

- $\mathbf{I}$: Identität bzgl. der Multiplikation
  - $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

- $\mathbf{A}^{-1}$: Inverse bzgl. der Multiplikation
  - Wenn für $\mathbf{A}_{(m,m)}$ gilt: $\mathbf{P}=\mathbf{AQ}$
  - gibt es dann ein $\mathbf{B}_{(m,m)}$ mit $\mathbf{Q} = \mathbf{BP}$?
  - Wenn ja, dann ist $\mathbf{B} = \mathbf{A}^{-1}$ die Inverse von $\mathbf{A}$
  - Dann gilt $\mathbf{Q} = \mathbf{BP} = \mathbf{B}(\mathbf{AQ}) = (\mathbf{BA})\mathbf{Q}$
  - Also ist $(\mathbf{BA}) = \mathbf{I}$, also $(\mathbf{A}^{-1}\mathbf{A}) = (\mathbf{AA}^{-1}) = \mathbf{I}$

Wie kann die Inverse berechnet werden?
→ später

Lineare Abbildung / Funktion / Operator im R3

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN University of Applied Sciences

- Lineare Abbildung $F{:}\ R^3 \rightarrow R^3$. Für $\mathbf{u}$, $\mathbf{v}$ aus R^3 und s aus R gilt:
  - Additivität: $F(\mathbf{u}+\mathbf{v}) = F(\mathbf{u}) + F(\mathbf{v})$
  - Homogenität: $F(s \cdot \mathbf{u}) = s \cdot F(\mathbf{u})$
- x‘, y‘ und z‘ werden als lineare Kombinationen von x, y und z berechnet:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = F(x,y,z) = \begin{pmatrix} a_{11}x + a_{12}y + a_{12}z \\ a_{21}x + a_{22}y + a_{22}z \\ a_{21}x + a_{22}y + a_{22}z \end{pmatrix}$$

- Daher ist die Matrix-Schreibweise anwendbar:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = F(x,y,z) = \begin{bmatrix} a_{11}a_{12}a_{12} \\ a_{21}a_{22}a_{22} \\ a_{21}a_{22}a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{A}_{(3,3)} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Verkürzte Schreibweise mit Vektor $\mathbf{u} = (\mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z)^T$:

$$\mathbf{u}' = F(\mathbf{u}) = \mathbf{A}\mathbf{u}$$

Wiederholen:
Multiplikation Matrix-Vektor

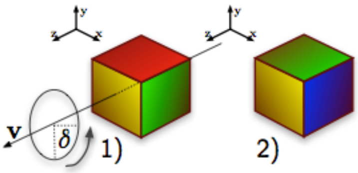
Beispiele für 3D-Transformationen

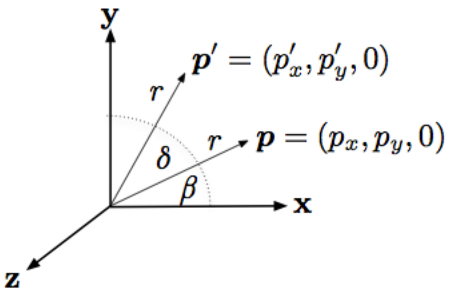
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN University of Applied Sciences

- Rotation
- Skalierung
- Spiegelung
- Scherung

Wiederholen:
- Berechnung der Operatoren
- Darstellung in 3x3-Matrixform

Rotation um z-Achse



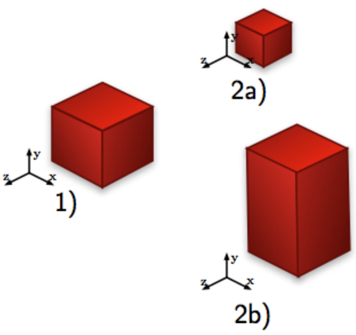


- Berechnung
- Matrixform
- Inverse

Hilfreich: zusammengesetzte Winkel

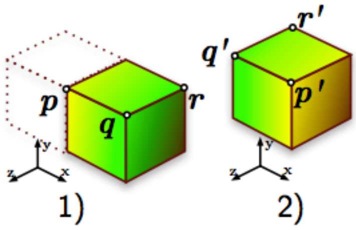
$$\begin{aligned}\sin(\alpha+\beta) &= \sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta)\\ \cos(\alpha+\beta) &= \cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta)\end{aligned}$$

Skalierung in y-Richtung



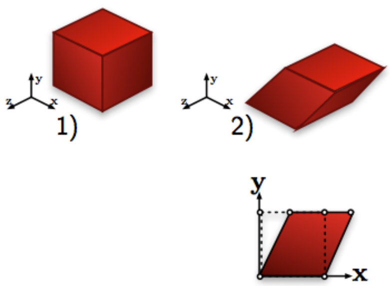
- Berechnung
- Matrixform
- Inverse

Spiegelung an Ebene x = 0



- Berechnung
- Matrixform
- Inverse

Scherung in x-Richtung



- Berechnung
- Matrixform
- Inverse

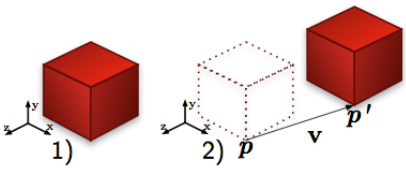
Beispiele für 3D-Transformationen

- Rotation
- Skalierung
- Spiegelung
- Scherung

Wiederholen:
- Berechnung der Operatoren
- Darstellung in 3x3-Matrixform

- Translation

Was unterscheidet die Translation von den anderen Transformationen?



Affine Erweiterung auf Homogene Koordinaten

- Erweitere die lineare Abbildung um einen additiven Term:

$$F(x,y,z) = \begin{bmatrix} a_{11} & a_{12} & a_{12} \\ a_{21} & a_{22} & a_{22} \\ a_{21} & a_{22} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- Schreibweise mittels 4x4-Matrix:

$$F(x,y,z,w) = \begin{bmatrix} a_{11} & a_{12} & a_{12} & t_x \\ a_{21} & a_{22} & a_{22} & t_y \\ a_{21} & a_{22} & a_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotation, Skalierung, ...

Translation

Zeige:
Diese Matrix liefert die gewünschte Abbildung

Homogene Koordinaten

- Was ist die Bedeutung der w-Koordinate?
  - Könnte man Sie für die erwähnten Operationen nicht einfach immer ignorieren?

    ja, könnte man im Prinzip!

  - Kann jedoch auch sehr elegant zur Unterscheidung von Punkten und Richtungen verwendet werden. Vereinbarung:
    - Positionsvektor (Punkt): $w = 1$
    - Richtungsvektor: $w = 0$
  - Nun transformiert dieselbe Matrix *Punkte* anders als *Richungen*:
    - Translation eines Punktes: Punkt wird *verschoben*

      nachrechnen!

    - Translation einer Richtung: Richtung bleibt *unverändert*
  - Bei der perspektivischen Projektion hat w noch weitere Bedeutung
    - Bei der Projektion entstehen w-Koordindaten mit beliebigen Werten

      (später)

    - Danach alle Koordinaten des resultierenden Vektors wieder durch w teilen

Rechnen mit Vektoren in homogenen Koordinaten (1)

- Subtraktion Punkt – Punkt

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} - \begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x - q_x \\ p_y - q_y \\ p_z - q_z \\ 0 \end{bmatrix}$$

- Addition Richtung + Richtung

$$\begin{bmatrix} u_x \\ u_y \\ u_z \\ 0 \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} u_x + v_x \\ u_y + v_y \\ u_z + v_z \\ 0 \end{bmatrix}$$

- Addition Punkt + Richtung

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} + \begin{bmatrix} u_x \\ u_y \\ u_z \\ 0 \end{bmatrix} = \begin{bmatrix} p_x + u_x \\ p_y + u_y \\ p_z + u_z \\ 1 \end{bmatrix}$$

- Multiplikation Skalar • Richtung

$$s \begin{bmatrix} u_x \\ u_y \\ u_z \\ 0 \end{bmatrix} = \begin{bmatrix} su_x \\ su_y \\ su_z \\ 0 \end{bmatrix}$$

Rechnen mit Vektoren in homogenen Koordinaten (2)

- Skalarprodukt zweier Richtungsvektoren

$$\begin{bmatrix} u_x \\ u_y \\ u_z \\ 0 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = u_x v_x + u_y v_y + u_z v_z + 0$$

- Andere Operationen (z.B. Addition Punkt+Punkt) sind eigentlich nicht definiert!

Darauf kommen wir bei der Shading Language noch einmal zurück...

- Implementierung Vektoren / Matrizen
  - Vektoren werden i.d.R. nur mit 3 Komponenten gespeichert
  - Je nach Kontext wird die richtige Operation verwendet

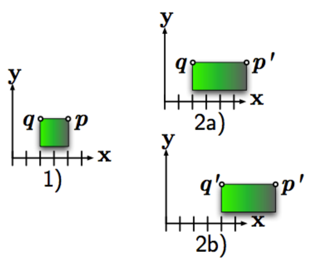
    → weniger Speicherbedarf

    → schnellere Berechnung

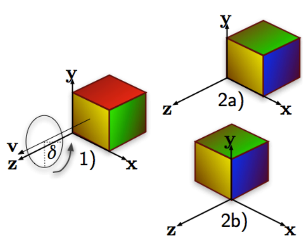
```
// transform a position vector
Matrix.transformPosition(p: Vector): Vector;

// transform a direction vector
Matrix.transformDirection(d: Vector): Vector;
```

Zusammengesetzte Transformationen / Composite Transformations

- Motivation
  - Einfache Transformationen beziehen sich immer auf den Ursprung
  - In sehr vielen Fällen möchte man jedoch ein Objekt bzgl. eines bestimmten Fixpunktes oder einer Achse transformieren.
  - Dafür bedient man sich zusammengesetzter Transformationen.



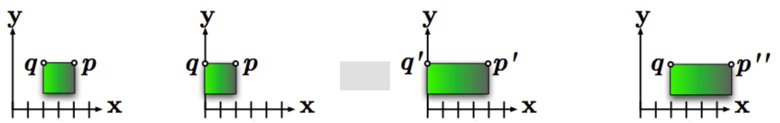
Objekt soll skaliert werden so wie in (2a) dargestellt. Skalierung (2b) verschiebt jedoch das Objekt!



Objekt soll 90° um seine Mittelachse gedreht werden (2a). Rotation um die Z-Achse (2b) verschiebt es jedoch.

Skalierung um (2,1,1); untere linke Ecke $\mathbf{q}$ bleibt fest

- Verschiebe alle Punkte um den Vektor ($\mathbf{0}$ - $\mathbf{q}$)
  - Der Vektor Vektor schiebt den Fixpunkt in den Ursprung
  - In diesem Beispiel reicht es aus, nur die X-Koordinate zu verschieben.
- Skaliere Objekt (alle Punkte) um gewünschte Größe
  - Bei jeglicher Skalierung bleibt der Punkt (0,0,0) immer unverändert!
- Verschiebe alle Punkte um die Inverse ($\mathbf{q}$ - $\mathbf{0}$) des ursprünglichen Vektors



... linker Rand in den Ursprung verschoben

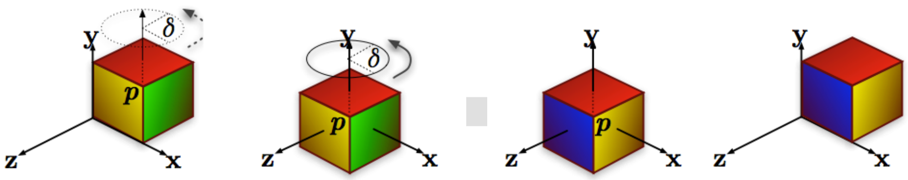
... in X-Richtung skaliert

... in ursprüngl. Position zurück geschoben

$$F(\mathbf{x}) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & q_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{3.} \cdot \underbrace{\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{2.} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & -q_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{1.} \cdot \mathbf{x} = \underbrace{\begin{bmatrix} 2 & 0 & 0 & q_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{zusammen}} \cdot \mathbf{x}$$

Rotation um den Mittelpunkt $\mathbf{p}$, parallel zur Y-Achse

- Verschiebe Mitte des Objekts in den Ursprung
  - Rotationsachse geht nun ebenfalls durch den Ursprung / ist die Y-Achse
- Drehe um die Y-Achse
- Verschiebe Mitte des Objekts an ursprüngliche Position zurück



... Mittelpunkt $\mathbf{p}$ in den Ursprung verschoben

... im Ursprung rotiert

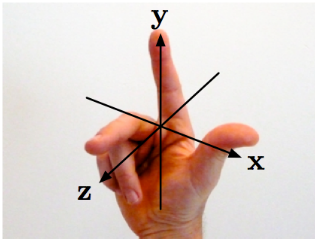
... an ursprüngl. Position

$$F(\mathbf{x}) = \underbrace{\begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{3.} \cdot \underbrace{\begin{bmatrix} \cos\frac{\pi}{2} & 0 & \sin\frac{\pi}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\frac{\pi}{2} & 0 & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{2.} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{1.} \cdot \mathbf{x} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & p_x - p_z \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & p_x + p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{zusammen}} \cdot \mathbf{x}$$

Kartesisches Koordinatensystem

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN University of Applied Sciences

- Wenn nichts anderes angegeben, betrachten wir Koordinaten meist im *kartesischen* Koordinatensystem
  - Drei Basisvektoren
    - $\mathbf{x} = (1,0,0)^T$
    - $\mathbf{y} = (0,1,1)^T$
    - $\mathbf{z} = (0,0,1)^T$
  - Diese Basis ist orthonormal
  - Diese Basis kann ein linkshändiges oder ein rechtshändiges System aufspannen; üblicherweise verwenden wir ein rechtshändiges



- Die Komponenten eines Vektors sind die Projektionen des Vektors auf den jeweiligen Basisvektor
  - $\mathbf{u} = (1, 2, 3)^T$ bedeutet: $\mathbf{u}{\cdot}\mathbf{x} = 1$, $\mathbf{u}{\cdot}\mathbf{y} = 2$, $\mathbf{u}{\cdot}\mathbf{z} = 3$
  - Das Skalarprodukt $\mathbf{u}{\cdot}\mathbf{x}$ ist die Länge der Projektion von $\mathbf{u}$ auf $\mathbf{x}$.

Zeichnung:
Projektion

Transformation zwischen Koordinatensytemen (1)

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN University of Applied Sciences

- Bisherige Vorstellung:
  - Eine Transformation „manipuliert“ Punkte und Vektoren
  - Dabei bleiben wir immer im gleichen Koordinatensystem
- Alternative Vorstellung:
  - Eine Transformation bildet Punkte von einem in ein anderes Koordinatensystem ab
  - Dabei bleibt die Struktur der Menge von Punkten, die transformiert werden erhalten. Lediglich die Basisvektoren werden transformiert.
- Das schöne dabei:
  - Für die Beschreibung der Transformation zwischen zwei Koordinatensystem genügt eine einzige 4x4-Matrix

Zeichnung:
zwei Koordinatensysteme

Transformation zwischen Koordinatensytemen (2)

- Kamera verschieben
  - Verschiebe Kamera um $\mathbf{v}$
  - oder: verschiebe Objekte um -$\mathbf{v}$
- Kamera rotieren
  - Rotiere Kamera um ω um die Y-Achse
  - Oder: rotiere Objekte um -ω um die Y-Achse
- Kamera: Implementierung
  - Generiere Kamera-Strahlen wie gewohnt vom Ursprung
  - Transformiere den Strahl, bevor er an die Szene übergeben wird

- Instancing
  - Modelliere ein Objekt in einem geeigneten Koordinatensystem
  - Plaziere es mehrfach an verschiedenen Orten in unterschiedlicher Orientierung und Größe (Skalierung)

Zeichnung: Instancing

Szenengraph (Scene Graph)

Motivation / Übersicht

- Bisher: Modellierung der Szene in Weltkoordinaten
  - Für Kugeln und Ebenen vielleicht ok
  - Für komplexere Geometrie nicht akzeptabel:
    - Koordinate jedes Vertices muß einzeln modelliert werden
    - Schwer, den Überblick zu behalten
- Ziel
  - Anschauliche Modellierung der Szenen-Aufteilung
    - Welches Objekt ist wo, welche Objekte gehören zusammen
    - Mehrfach-Instanzierung eines einmal modellierten Objekts
  - Einfache Methoden, um Objekte selektiv dynamisch zu transformieren
    - Verschieben, rotieren, skalieren, ...
  - Effiziente Berechnung und Repräsentation von Transformationen

Übliche Koordinatensysteme

- Weltkoordinaten W
  - Globales Referenzsystem für alle Objekte
- Objekt-/ Modell-/ lokale Koordinaten O
  - Lokales System, in dem das Objekt modelliert wird
- Kamera-/ Augenkoordinaten C
  - Lokales System, in welchem die Kamera modelliert wird (wie vorgestellt: Auge bei $\mathbf{0}$, Blick entlang $-\mathbf{z}$; $\mathbf{y}$ ist Up-Vektor)
- Transformationskette
  - Modell → Welt → Auge
  - Matrizen $\mathbf{M}$, $\mathbf{V}$
  - $\mathbf{MV}$ = „Modelview-Matrix“
  - Umgekehrter Weg geht über die Inverse $\mathbf{V}^{-1}\mathbf{M}^{-1}$

Modell-Transformation

Kamera-/ View-Transformation

$$O \underset{M^{-1}}{\overset{M}{\rightleftarrows}} W \underset{V^{-1}}{\overset{V}{\rightleftarrows}} C$$

Hierarchische Modellierung mit einem Szenegraph

Definition (Szenengraph):

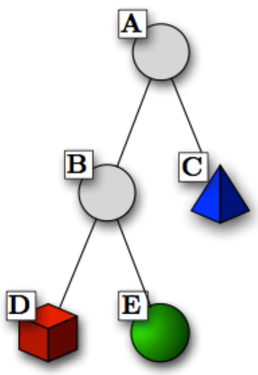
Ein Szenengraph ist eine Datenstruktur für Transformations-Hierarchien.

- Grundidee:
  - Modelliere die Geometrie der Objekte in lokalen Koordinaten.
  - Spezifiere die räumliche Lage und Skalierung der Objekte durch Transformationen.
  - Gruppiere zusammenhängende Transformationen hierarchisch.

- Hinweise
  - Typischerweise als gerichteter azyklischer Graph (DAG, Directed Acyclic Graph) modelliert.
  - Oft wird der Graph um weitere Attribute wie z.B. Materialien oder Objektverhalten erweitert.
  - Manchmal hält ein Szenengraph den gesamten Zustand der Applikation.

- Ähnliche Konzepte
  - HTML Box-Modell

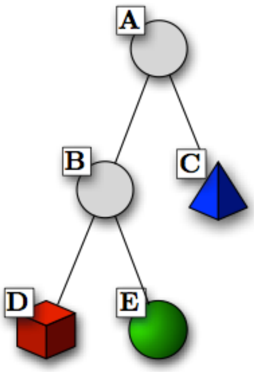
Komponenten des Szenengraphen



Ein generischer Szenengraph

- Knoten
  - Jeder Knoten repräsentiert eine (transformierte) Sub-Szene.
- Innerer Knoten
  - Gruppiert mehrere Objekte bzw. Szenenteile
- Blattknoten
  - Enthält die Objekte mit ihrer Geometriedefinition
- Kante
  - Koordinaten-Transformation zwischen Elternteil und Kind

Komponenten des Szenengraphen



Ein generischer Szenengraph

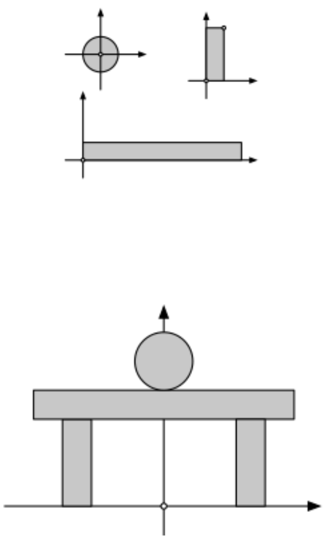
- Modell-Transformation
  - Kombiniere die Transformationen, die durch die Knoten gegeben sind
  - Folge dem Pfad von der Wurzel zu den Blättern
  - Multipliziere Matrizen (von links nach rechts) in der Reihenfolge der Knoten

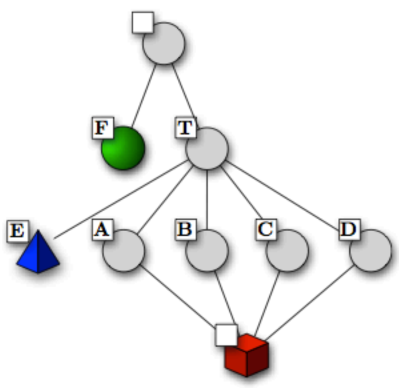
Beispiel:

Transformation $\mathbf{M}_D$ von lokalen Koordinaten O_D des Objekts D zu Weltkoordinaten:

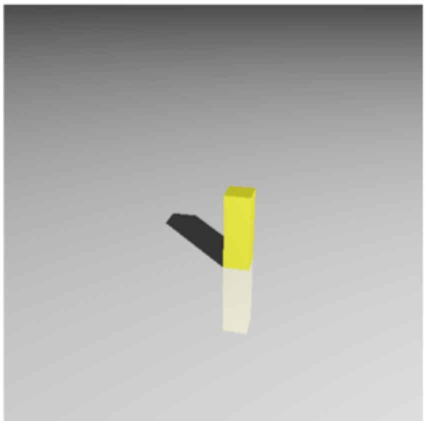
$\mathbf{M}_D = \mathbf{ABD}$

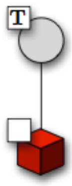
Beispielszene: Tisch mit Kugel





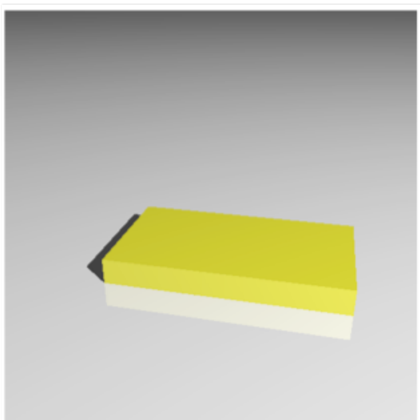
Beispielszene: Tisch mit Kugel





- Szenengraph eines einzelnen Tischbeins

Beispielszene: Tisch mit Kugel

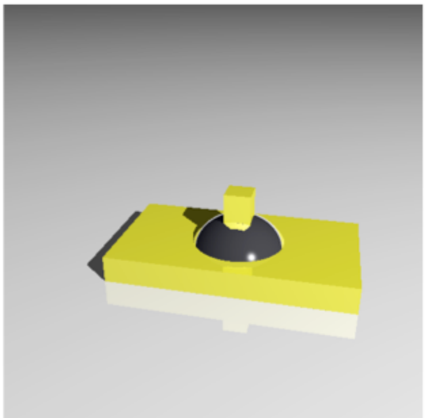


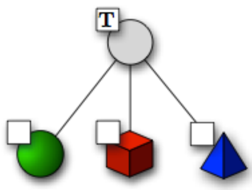
- Szenengraph der Tischoberfläche

Beispielszene: Tisch mit Kugel

- Szenengraph der Kugel

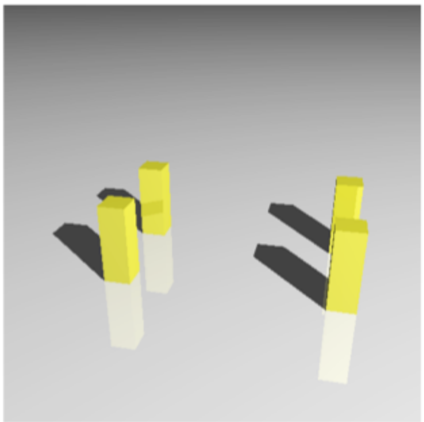
Beispielszene: Tisch mit Kugel

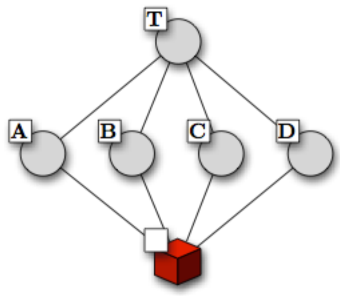




- Alle Tischteile in einem gemeinsamen Szenengraph

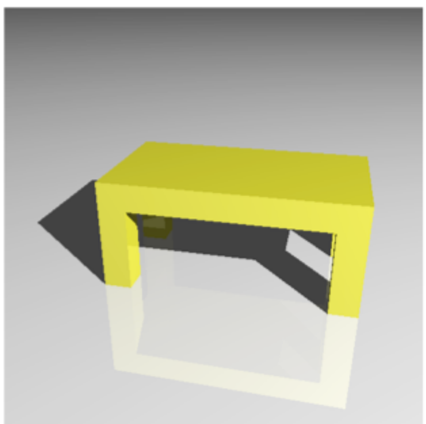
Beispielszene: Tisch mit Kugel

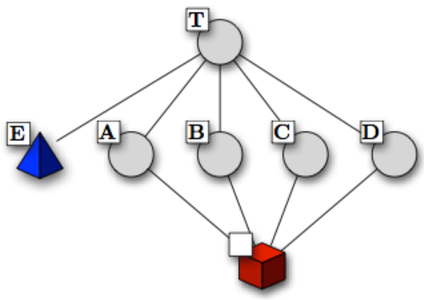




- Vier Instanzen des gleichen Tischbeins („Instancing“)

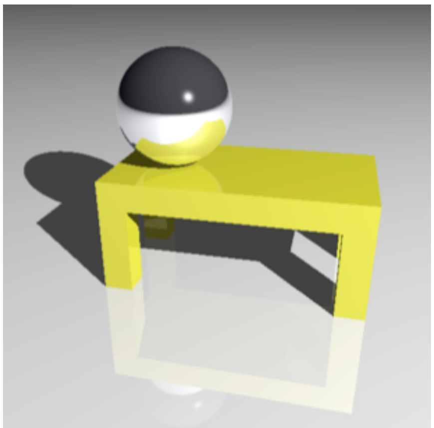
Beispielszene: Tisch mit Kugel

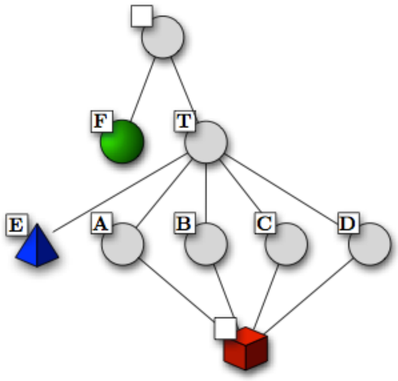




- Vollständiger Szenengraph für den Tisch

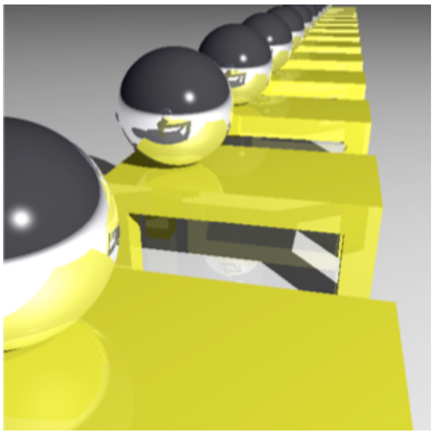
Beispielszene: Tisch mit Kugel

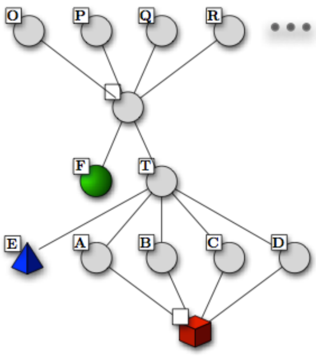




- Vollständiger Szenengraph für den Tisch

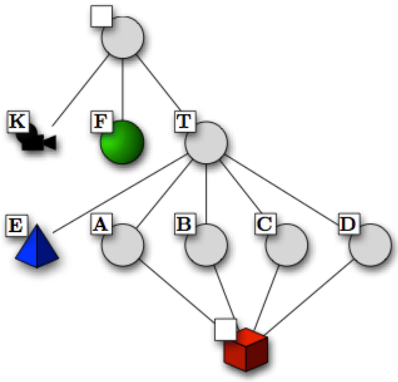
Beispielszene: Tisch mit Kugel





- Kunstwerk mit vielen Tischen

Zusammensetzen der Transformationen



- Beispiele
  - Kamera K → Welt
  - Tischfläche E → Welt
  - Tischbein → Welt
  - Welt → Kamera K
  - Welt → Tischbein
  - Kugel F → Kamera
  - Kamera → Tischfläche E

Raytracing von Szenengraphen

- Ohne Szenengraph
  - Alles in Weltkoordinaten definiert
- Mit Szenengraph
  - Verwende drei Koordinatensysteme
- Weltkoordinaten
  - Beleuchtungsberechnung
  - Erzeugung von Schatten- und Sekundärstrahlen
- Lokale Objektkoordinaten / Modellkoordinaten
  - Geometriedefinition (z.B. Kugel immer im Ursprung)
  - Schnittpunktberechnung
- Kamerakoordinaten
  - Erzeuge Primärstrahlen vom Urspung durch die Bildebene

Raytracing von Szenengraphen: Transformationsabfolge

- Erzeugung der Primärstrahlen
  - Erzeuge im Kamera-System C und transformiere Strahl in Weltkoordinaten W
- Rekursive Traversierung des Szenegraphen
  - Akkumuliere Transformationen während der Traversierung
    - Transformation M: $O \rightarrow W$
    - Transformation M^{-1}: $W \rightarrow O$
  - Transformiere den Strahl in lokale Objektkoordinaten O
  - Berechne den Schnittpunkt in den lokalen Koordinaten O
  - Transformiere den Trefferpunkt zurück nach W
- Beleuchtungsberechnung
  - Berechne die Beleuchtung im Trefferpunkt in Weltkoordinaten W

Raytracing von Szenengraphen: Rekursive Traversierung

- Rekursiver Abstieg
  - Elegante Traversierung eines DAG mittels rekursivem Abstieg
  - Der Zustand wird implizit auf dem Prozeß-Stack mitgeführt
  - Üblicherweise erfolgt der Abstiegt *Depth-First*
    - d.h. Kinder werden vor ihren Eltern abgearbeitet
- Implementierung
  - Kann auch ohne Rekursion implementiert werden
  - Dabei muß der Stack explizit verwaltet werden

Raytracing-Algorithmus ohne Szenengraph

```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
  // loop over all pixels in image
  for all (i,j) from (0,0) to (image.width-1, image.height-1) do
  {
    // generate ray from eye point through pixel center
    ray ← generate_ray(i, j, image.width, image.height, camera);

    // find first intersection point with scene objects
    hit ← intersect(ray, scene);

    // calculate light intensity/color at hit point
    color ← shade(hit, ray, scene);

    // set corresponding pixel color in image
    image.pixel(i,j) ← color;
  }
}
```

Raytracing-Algorithmus mit Szenengraph (1)

```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
  // loop over all pixels in image
  for all (i,j) from (0,0) to (image.width-1, image.height-1) do
  {
    // generate ray in world coordinates
    rayWC ← generate_ray(i, j, image.width, image.height, camera);

    // find first intersection point with scene objects
    hit ← intersect(rayWC, scene);

    // calculate light intensity/color at hit point
    color ← shade(hit, rayWC, scene);

    // set corresponding pixel color in image
    image.pixel(i,j) ← color;
  }
}
```

Raytracing-Algorithmus mit Szenengraph (3)

```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
  // loop over all pixels in image
  for all (i,j) from (0,0) to (image.width-1, image.height-1) do
  {
    // generate ray in world coordinates
    rayWC ← generate_ray(i, j, image.width, image.height, camera);

    // start recursive descent to calculate intersections
    hitWC ← scene.root().intersectRec(rayWC, Matrix.Id, Matrix.Id);

    // calculate light intensity/color in world coordinates
    color ← shade (hitWC, rayWC, scene);

    // set corresponding pixel color in image
    image.pixel(i,j) ← color;
  }
}
```

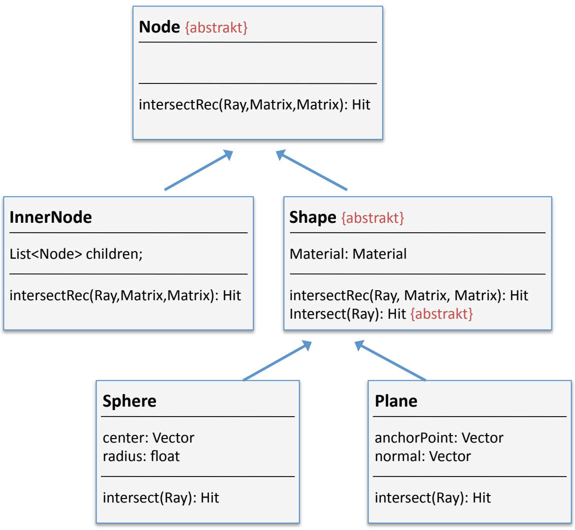
Algorithmus: Traversierung der inneren Knoten

```
// recursive scene graph traversal for inner nodes
node.intersectRec(rayWC: Ray, toWorld: Matrix, fromWorld: Matrix)
{
  // accumulate transformation from this node to world coords
  toWorld ← toWorld • this.transform();

  // accumulate inverse transformation from world to this node
  fromWorld ← this.inverseTransform() • fromWorld;

  // recursively descend down the scene graph
  hit ← Hit.Infinity;
  for all child in this.children() do {
    hitChild ← child.intersectRec(rayWC, toWorld, fromWorld);
    // compare which hit is closer to the ray origin
    if(hitChild.isCloserThan(hit))
      hit = hitChild;
  }
  return hit;
}
```

Algorithmus: Schnittpunktberechnung in den Blättern

```
// recursive scene graph traversal for leaf nodes / shapes
shape.intersectRec(rayWC, toWorld, fromWorld)
{
  // accumulate transformation from this node to world coords
  toWorld ← toWorld • this.transform();

  // accumulate inverse transformation from world to this node
  fromWorld ← this.inverseTransform() • fromWorld;

  // transform ray to local coordinates
  rayLC ← rayWC.transform(fromWorld);

  // calculate intersection in local coordinates
  hitLC = this.intersect(rayLC);

  // transform back to world coordinates
  return hitLC.transform(toWorld);
}
```

Raytracing-Szenengraph und das Kompositum-Muster



- Kompositum-Entwurfsmuster
  - siehe z.B. wikipedia.de
- Identisches Interface für
  - Atomare Knoten (Blätter)
  - Zusammengesetzte Knoten (innere Knoten)
- Polymorphie
  - Innerer Knoten implementieren intersectRec() anders als Blätter
- Schnittpunktberechnung
  - Nur Blätter besitzen die eigentliche Schnittpunktberechnung intersect()