

VEKTORIA-MANUAL PLACEMENTS



Game Design

Inhalt



/// **VERTIEFUNG**

/// **GRUNDL. PLACEMENTS**

/// **MATRIXOPERATIONEN**

/// **SWITCHES**

/// **LODS**



/// PROF. /// DR. /// TOBIAS /// BREINER
/// VEKTORIA /// ENGINE /// MANUAL

2 VON 45
/// PLACEMENTS





/// VERTIEFUNG

2 // /

3 // /

4 // /

5 // /



PROF. DR. TOBIAS BREINER
VEKTORIA ENGINE MANUAL

3 VON 45
PLACEMENTS

Knotenobjekte von Szenengraphen

Virtuelle Szene

Gruppenknoten

- Gruppenbehälter
- Transformationen
- Switchknoten (Level-of-Detail, Animation etc.)
- ...

Blattknoten

- Lichter
- Geometrien
- Kameras
- Materialien, Texturen, Farben
- ...

Reale Szene

Gruppenknoten

- Gruppenbehälter
- Computer
- Kanäle
- Switchknoten (an, aus)
- ...

Blattknoten

- Eingabegeräte
- Sichtsysteme
- Sichtfenster
- Soundgeräte
- Basis-Render-API
- ...



Kapitel 2

Kapitel 2



/// **GRUNDL. PLACEMENTS** ///



3 // / / /

4 // / / /

5 // / / /





Grundlagen der Placements

Placements

Ein Placement ist verantwortlich für die hierarchische Objektplatzierung.

Synonyme in anderen Szenegrafen:

Transforms, TransformGroups, Instances, Locales, LocaleCoordinates, Placements, Objects

Hat eine interne homogene Matrix, die folgende Operationen ermöglicht:

- Translation
 - Rotation
 - Skalierung
 - Scherung,
 - Spiegelung

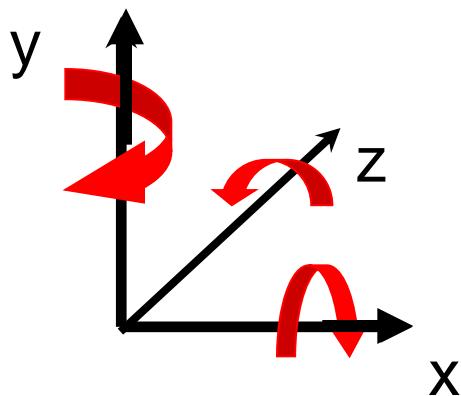
In Vektoria haben Placements noch weitere Attribute, z.B.:

- An- und Ausschalter
 - Bounding Boxes
 - Flags zur eventuellen Beschleunigung

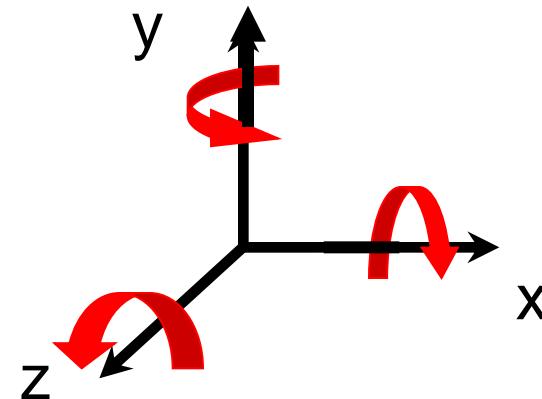
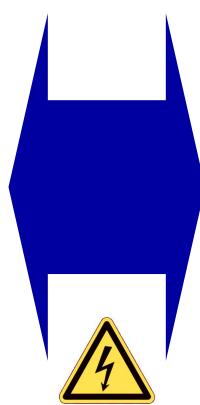


Rechts- vs. linkshändige Systeme

- 1 // / / /
- 2 // / / /
- 3 // / / /
- 4 // / / /
- 5 // / / /

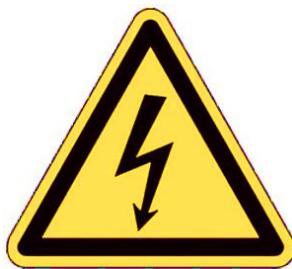


links:
(DirectX)
RealiMation
3D-Generation
...

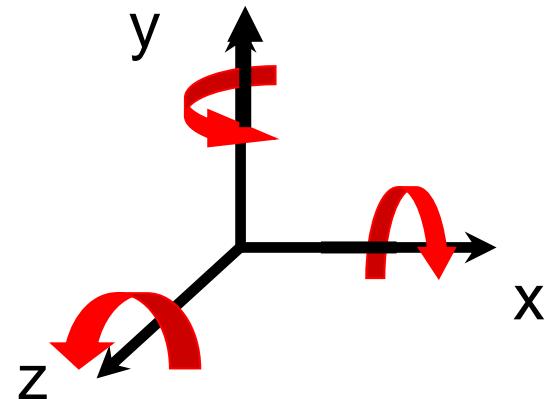


rechts:
OpenGL
Iris Inventor
Open Performer
Vektoria
...

Rechts- vs. linkshändige Systeme



Achtung!



Vektoria hat der Kompabilität zu den meisten anderen 3D-Systemen zuliebe ein rechtshändiges Koordinatensystem als auch rechtshändige Rotationsrichtungen, auch wenn DirectX normalerweise linkshändig ist. Die Systeme werden in Vektoria umgerechnet.

Kapitel 3

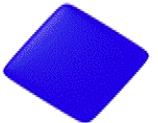
Kapitel 3



1 // / / /

2 // / / /

/// MATRIXOPERATIONEN ///



4 // / / /

5 // / / /



Matrix in dem Placement

1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & 1 \end{pmatrix}$$

Teilmatrizen:



Lineare Transformationen

- Skalierung
- Rotation
- Scherung
- Spiegelung



Translationen



Projektionen



CPlacement::SetMat



Hauptmethode:

- void SetMat(CHMat & m);

Setzt die homogene 4*4-Matrix **m**.

Die Default-Matrix des Placements ist die Einheitsmatrix.

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /



CPlacement::Unit

1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /

```
void Unit();
```

Generiert Einheitsmatrix für das Placement
=> Placement wird wieder auf Ursprungsposition transformiert.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$





Placement-Matrixoperationen

CPlacement Rotationen

1 // / / /
void RotateX(float fa);
void RotateY(float fa);
void RotateZ(float fa);

Generiert Rotationsmatrix für das Placement um x, y oder z-Achse mit Winkel fa im Bogenmaß

2 // / / /
void Rotate(float fx, float fy, float fz, float fa);

3 // / / /
Generiert beliebige Rotationsmatrix für das Placement um die Drehachse fx, fy, fz mit Bogenmaßwinkel fa

4 // / / /
void Rotate(CHVector & v, float fa);

5 // / / /
Generiert beliebige Rotationsmatrix für das Placement um die Drehachse v mit Bogenmaßwinkel fa



CPlacement Rotationen



void Rotate(CHVector & vTo, CHVector & vFrom);

Erzeugt eine Rotationsmatrix für das Placement,
welche Richtungsvektor **vFrom** genau in die Richtung
des Richtungsvektors **vTo** rotieren würde

1 // / / /

2 // / / /

3 // / / /



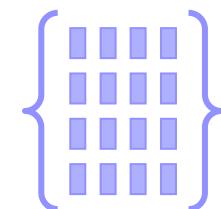


Rotationen nicht kommutativ!



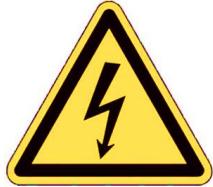
- 1 // / / /
- 2 // / / /
- 3 // / / /
- 4 // / / /
- 5 // / / /

$$M_{Rot1} \bullet M_{Rot2} \neq M_{Rot2} \bullet M_{Rot1}$$



Placement-Matrixoperationen

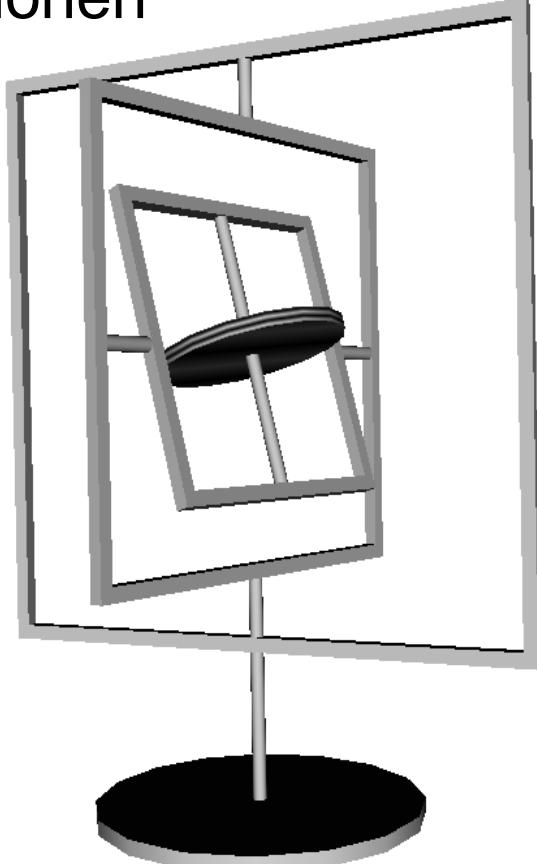
Gimbal Lock



Achtung 3: Gimbal Lock bei Rotationen

Verlust eines Freiheitsgrades =>

2 Singularitäten

1 // / / / 2 // / / / 3 // / / / 4 // / / / 5 // / / / 
Entgegen einer weit verbreiteten Meinung sind Quaternionen nicht immer die Lösung für Gimbal Lock!

Statt dessen: Umrechnung der globalen Orientierungen in lokale Koordinatensysteme





Placement-Matrixoperationen

CPlacement Skalierungsmethoden



`void Scale(float fx, float fy, float fz);`

1 // / / / Generiert Skalierungsmatrix mit drei verschiedenen Skalierungswerten in x, y und z-Richtung für das Placement.

Beispiel: `m_zp.Scale(1,1,2)` verdoppelt die Länge aller Objekte, die an `m_zp` hängen in lokaler z-Richtung

`void Scale(float f);`

2 // / / / Generiert uniforme Skalierungsmatrix

Beispiel: `m_zp.Scale(3)` macht alle Objekte, die an `m_zp` hängen dreifach so groß.





Placement-Matrixoperationen

CPlacement Skalierungsmethoden



```
void Translate(CHVector & v);
```

Generiert Verschiebungsmatrix um den Vektor v.

Beispiel: m_zp.Translate(CHVector(1,0,0)) verschiebt alle Objekte, die an m_zp hängen, um eins in lokaler x-Richtung.

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /





Placement-Matrixoperationen

CPlacement Skalierungsmethoden



void TranslateX(float fx);
void TranslateY(float fy);
void TranslateZ(float fz);

Generiert Verschiebungsmatrizen in Richtung der jeweiligen Koordinatenachsen.





CPlacement Deltamethoden

```
void RotateXDelta(float fa);
void RotateYDelta(float fa);
void RotateZDelta(float fa);
void RotateDelta(float fx, float fy, float fz, float fa);
void RotateDelta(CHVector & v, float fa);
void TranslateDelta(CHVector & v);
void ScaleDelta(float fx, float fy, float fz);
```

...

Durch die Deltamethoden kann man die entsprechende Transformation nacheinander ausführen.

Beispiel: m_zp.Translate(CHVector(0,0,1));
m_zp.RotateYDelta(HALFPI); verschiebt erst in z-Richtung und rotiert dann um 90 Grad in Y-Richtung.





Placement-Verknüpfungsmethoden

CPlacement Verknüpfungsmethoden



1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /

```
void AddGeo(CGeo * pgeo);
void AddCamera(CCamera * pcamera);
void AddPointLight(CPointLight * ppointlight);
void AddSpotLight(CSpotLight * pspotlight);
void AddPlacement(CPlacement * pplacement);
void AddAudio(CAudio * paudio);
void AddEmitter(CEmitter * pemitter);
```

Hängt wahlweise eine Geometrie (pgeo), eine Camera (pcamera), eine punktförmige Lichtquelle (ppointlight), einen Scheinwerfer (pspotlight) ein anderes Placement (pplacement), ein 3D Sound an (paudio) oder ein Emitter (pemitter) an. Zu jeder Add- gibt es noch eine Sub-Version, z.B. SubGeo, die das Objekt wieder abhängt.





Verknüpfungs- und Matrixmethoden des Placements

Übung zu Placements



- Lassen Sie die ErdKugel aus dem letzten „Hallo Erde!“-Programm um die Pole drehen!

Freiwillige Zusatzaufgaben für die schnellen Nerds:

- Imitieren Sie die tatsächliche Drehung der Erde mit einem Polneigungswinkel von 23,4 Grad !

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /





Lösung zur Übung Placements

Veränderung in Tick-Methode



```
1 /////
2 /////
3 /////
4 /////
5 /////

void CGame::Tick(float fTime, float fTimeDelta)
{
    m_zpSphere.RotateY(fTime);
    m_zpSphere.TranslateDelta(CHVector(0,0,-5));

    m_zr.Tick(fTimeDelta);
}
```



Kapitel 4

Kapitel 4



1 // / / /

2 // / / /

3 // / / /

/// SWITCHES



5 // / / /



PROF. DR. TOBIAS BREINER
VEKTORIA ENGINE MANUAL

24 VON 45
PLACEMENTS



CPlacement – Schaltmethoden

- void SwitchOn();
- void SwitchOff();

Schaltet das Placement mitsamt seiner Unterhierarchie ein (SwitchOn) oder aus (SwitchOff).

1 // / /

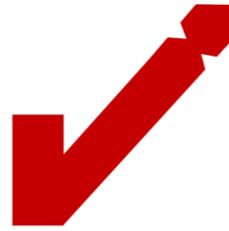
2 // / /

3 // / /

4 // / /

5 // / /





Placement-Schaltmethoden

Übung zu Switches



Lassen Sie die Erdkugel mit 1/2Hz
abwechselnd erscheinen und
verschwinden!

Freiwillige Zusatzaufgabe für die
schnellen Nerds:

- Lassen Sie die Erdkugel erst nicht
sichtbar sein, um dann nach 2
Sekunden SOS zu morsen!

1 // / /

2 // / /

3 // / /

4 // / /

5 // / /





Placement-Schaltmethoden

Übungslösung zu Switches



Veränderungen des „Hallo Erde!“-Programmes in der Game::Tick()-Methode, damit die Kugel mit ½ Herz blinkt:

```
void CGame::Tick( float fTime,
                  float fTimeDelta)
{
    if((int)(fTime*2.0F)%2==0)
        m_zpSphere.SwitchOn();
    else
        m_zpSphere.SwitchOff();

    m_zr.Tick(fTimeDelta);
}
```



Kapitel 5

Kapitel 5



1 // / / /

2 // / / /

3 // / / /

4 // / / /

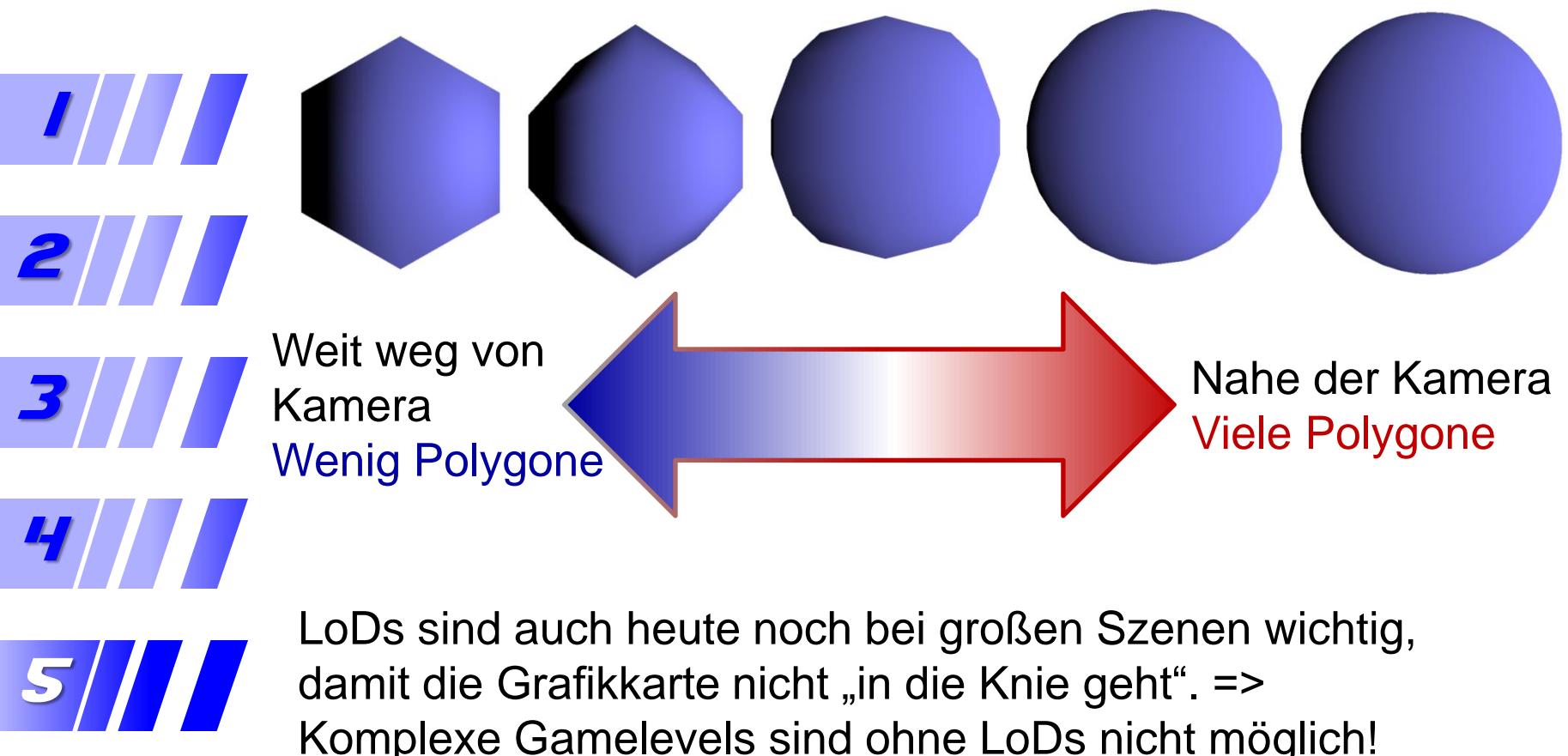
/// **LODS**



PROF. DR. TOBIAS BREINER
VEKTORIA ENGINE MANUAL

28 VON 45
PLACEMENTS

Level of Detail



CPlacement – Level of Details

Kameraentfernungsabhängige Schaltmethode:

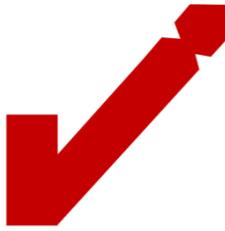
- void SetLoD(float fDistNear, float fDistFar);

Gibt Level of Detail-Grenzen an:

fDistNear ist der Abstand zu Kamera, ab dem das Objekt sichtbar / aktiv wird.

fDistFar ist der Abstand zu Kamera, ab dem das Objekt wieder unsichtbar / inaktiv wird.





Placements - LoDs

Übung zu LoDs



Lassen Sie den Abstand einer Kugel von der Kamera sinusförmig variieren! Geben Sie drei LoD-Stufen an, die durch den Abstand automatisch aktiviert werden!

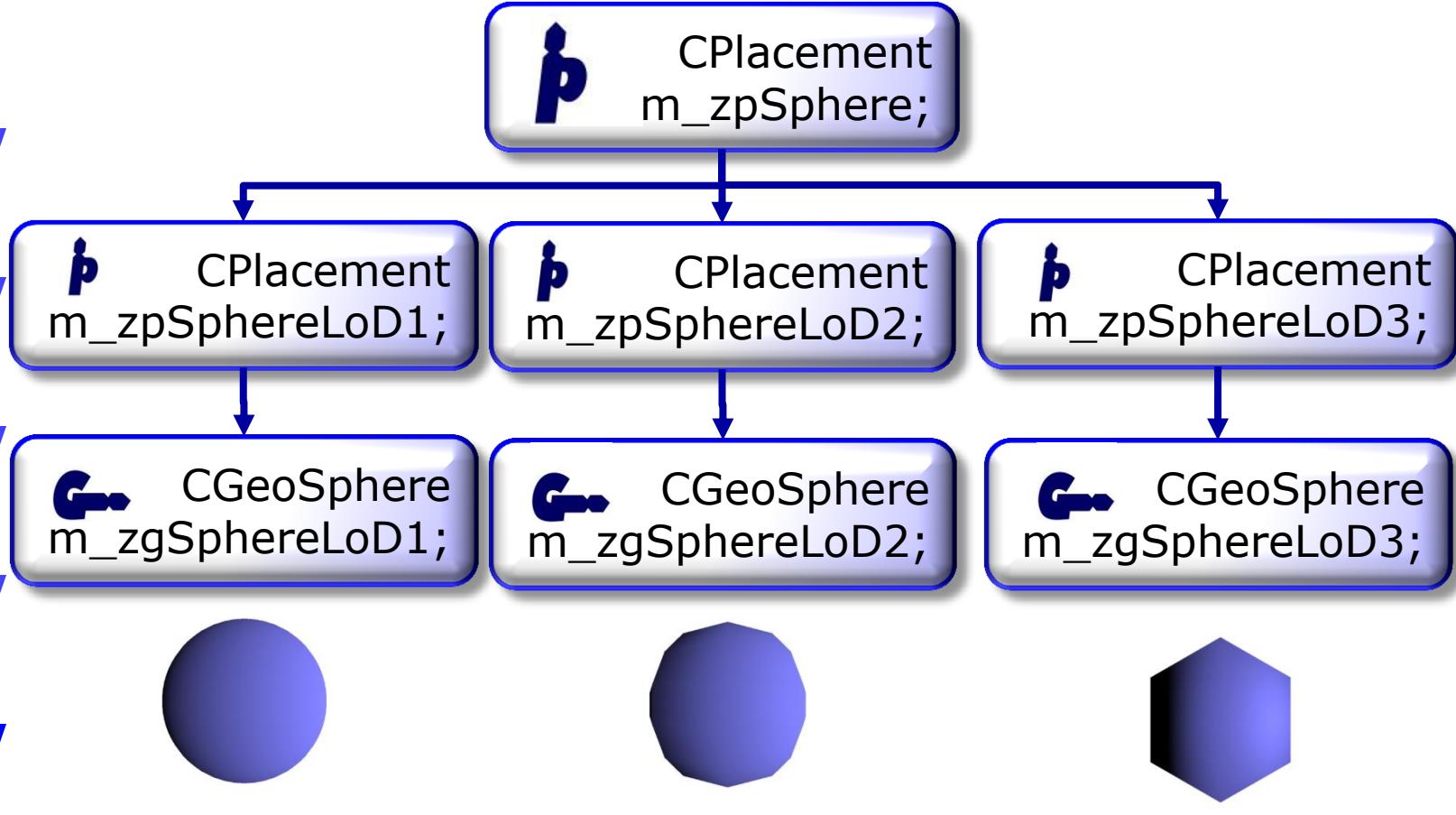
Freiwillige Zusatzaufgabe für die schnellen Nerds:

- Ersetzen Sie das hinterste LoD durch ein GeoQuad mit gemappter Kugeltextur!





Objekthierarchie der LoDs





Lösung LoD-Übung (2/6)

Objekte in der CGame.h



1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /

```
CRoot m_zr;  
CScene m_zs;  
CHardware m_zh;  
CFrame m_zf;  
CViewport m_zv;  
CCamera m_zc;  
CParallelLight m_zl;  
CMaterial m_zm;  
...
```

```
...  
CPlacement m_zpCamera;  
CPlacement m_zpSphere;  
CPlacement m_zpSphereLoD1;  
CPlacement m_zpSphereLoD2;  
CPlacement m_zpSphereLoD3;  
CGeoSphere m_zgSphereLoD1;  
CGeoSphere m_zgSphereLoD2;  
CGeoSphere m_zgSphereLoD3;
```





Lösung LoD-Übung (3/6)

Init-Methode in CGame.cpp



Hier das, was wir schon aus der „Hallo-Kugel!“-Anwendung kennen:

```
m_zr.Init(psplash);
m_zc.Init();
m_zf.Init(hwnd);
m_zv.InitFull(&m_zc);
m_zl.Init(CHVector(1,0,1),CCColor(1,1,1));
m_zm.MakeTextureImage("textures\\white_image.jpg");
m_zr.AddFrameHere(&m_zf);
m_zf.AddViewport(&m_zv);
m_zr.AddScene(&m_zs);
m_zs.AddPlacement(&m_zpSphere);
m_zs.AddPlacement(&m_zpCamera);
m_zs.AddParallelLight(&m_zl);
m_zpCamera.AddCamera(&m_zc);
```

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /





Lösung LoD-Übung (4/6)

Init-Methode in CGame.cpp



Und die eigentliche LoD-Logik:

```
m_zgSphereLoD1.Init(2.0F,&m_zm,48,48);
m_zgSphereLoD2.Init(2.0F,&m_zm,12,12);
m_zgSphereLoD3.Init(2.0F,&m_zm,6,6);
m_zpSphere.Translate(CHVector(0,0,-3));
m_zpSphere.AddPlacement(&m_zpSphereLoD1);
m_zpSphere.AddPlacement(&m_zpSphereLoD2);
m_zpSphere.AddPlacement(&m_zpSphereLoD3);
m_zpSphereLoD1.AddGeo(&m_zgSphereLoD1);
m_zpSphereLoD2.AddGeo(&m_zgSphereLoD2);
m_zpSphereLoD3.AddGeo(&m_zgSphereLoD3);
m_zpSphereLoD1.SetLoD(0.0F,3.0F);
m_zpSphereLoD2.SetLoD(3.0F,5.0F);
m_zpSphereLoD3.SetLoD(5.0F,100.0F);
```

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /





Lösung LoD-Übung (5/6)

Tick-Methode in CGame.cpp



1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /

```
void CGame::Tick(float fTime, float fTimeDelta)
{
    m_zpSphere.TranslateZ(-4.0F+2.0F*sin(fTime));

    m_zr.Tick(fTimeDelta);
}
```





Lösung LoD-Übung (6/6)

Fini-Methode in CGame.cpp



In die Fini-Methode brauchen wir hier noch nichts reinzuschreiben.

Wir haben ja weder dynamischen Speicherplatz allokiert, der wieder freigegeben werden müsste, noch müssen wir einen Gewinner ausgeben

3 // / / /

4 // / / /

5 // / / /



Beispiel zum Schluss

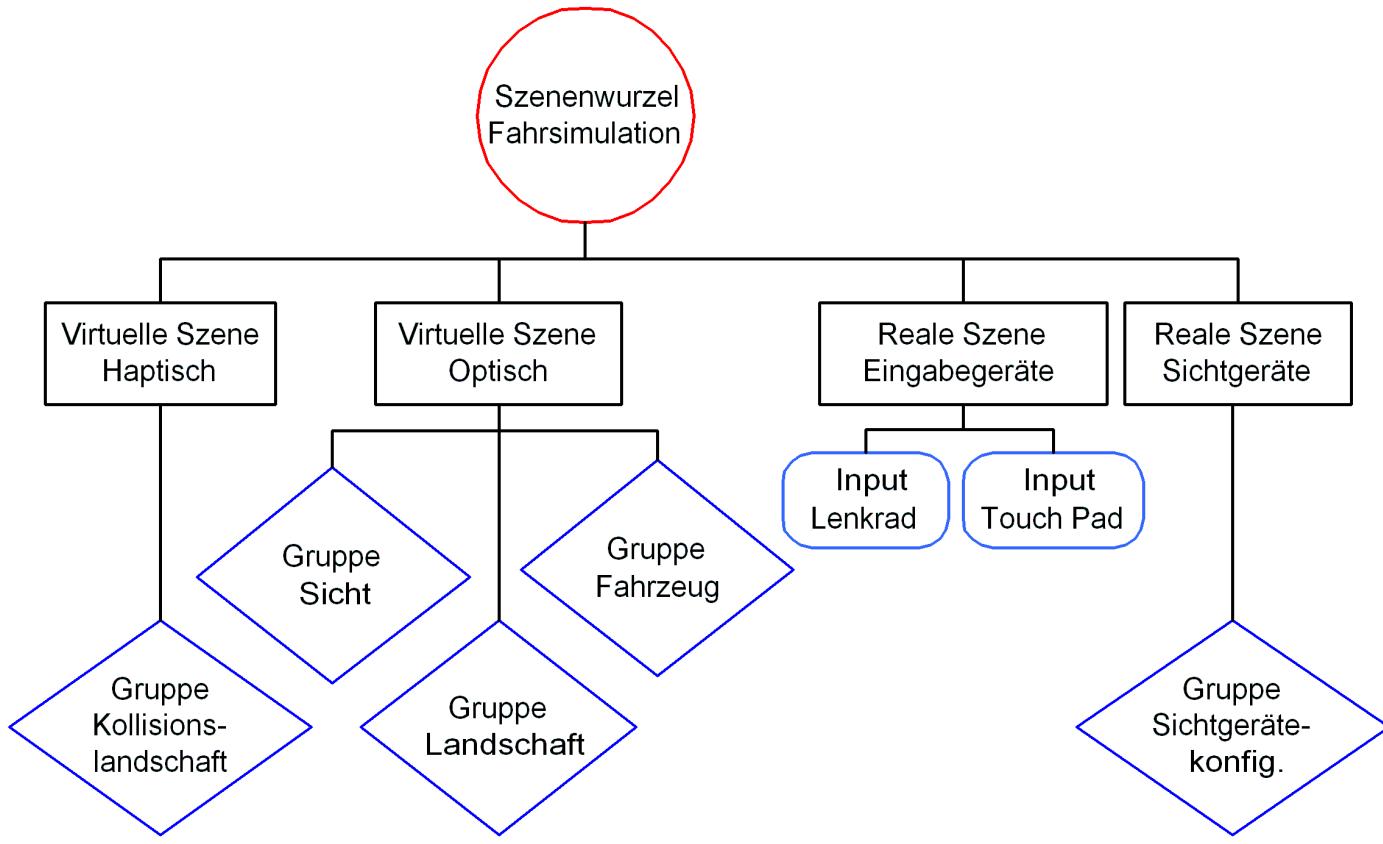


Zuletzt noch ein Beispiel für die Objekthierarchie einer konkreten Fahrsimulation



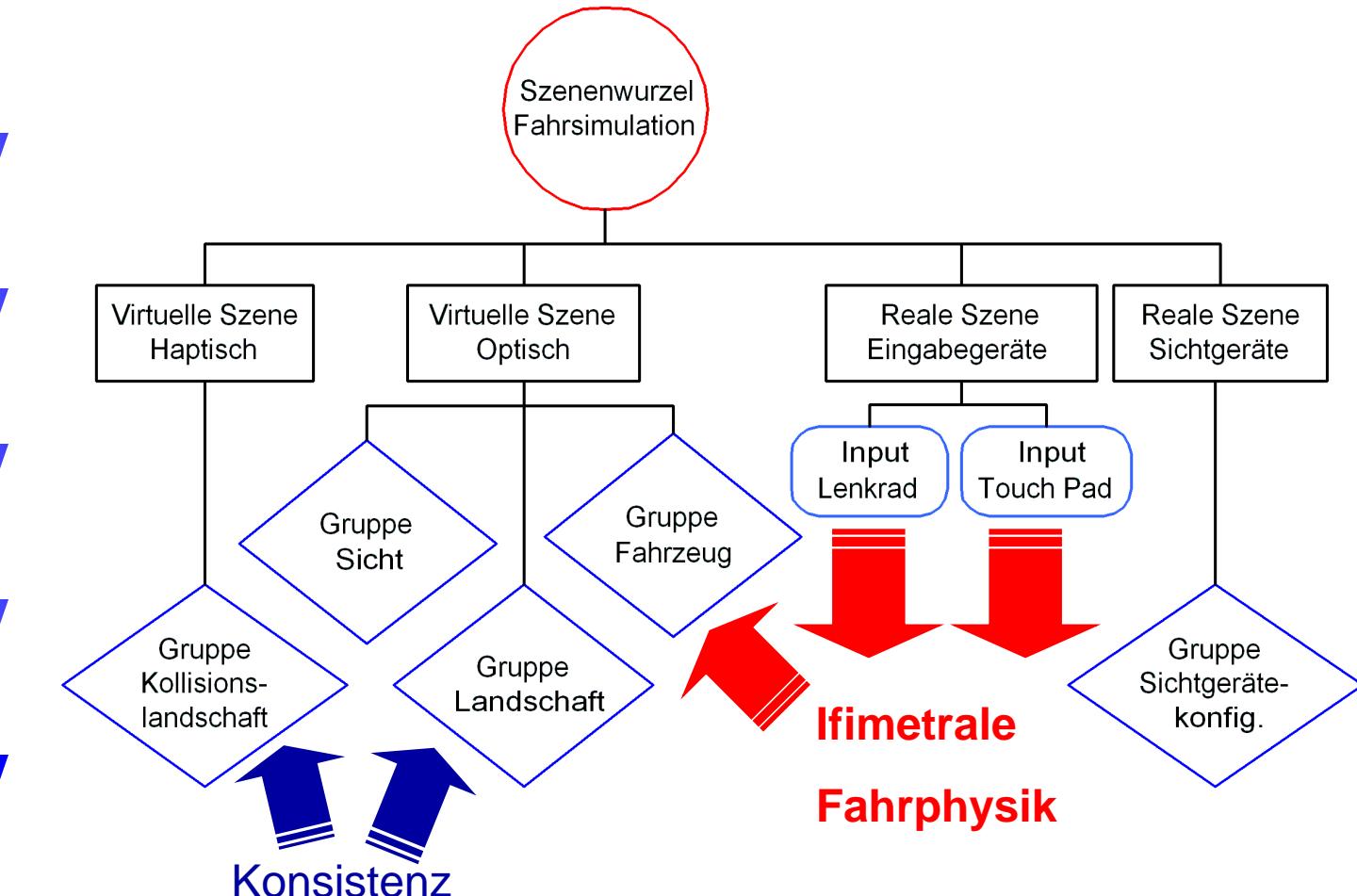
Bildung der Objekthierarchie

Hierarchie der Gesamtszene



Bildung der Objekthierarchie

Hierarchie der Gesamtszene



Bildung der Objekthierarchie

Hierarchie des Fahrzeuges



1 // /

2 // /

3 // /

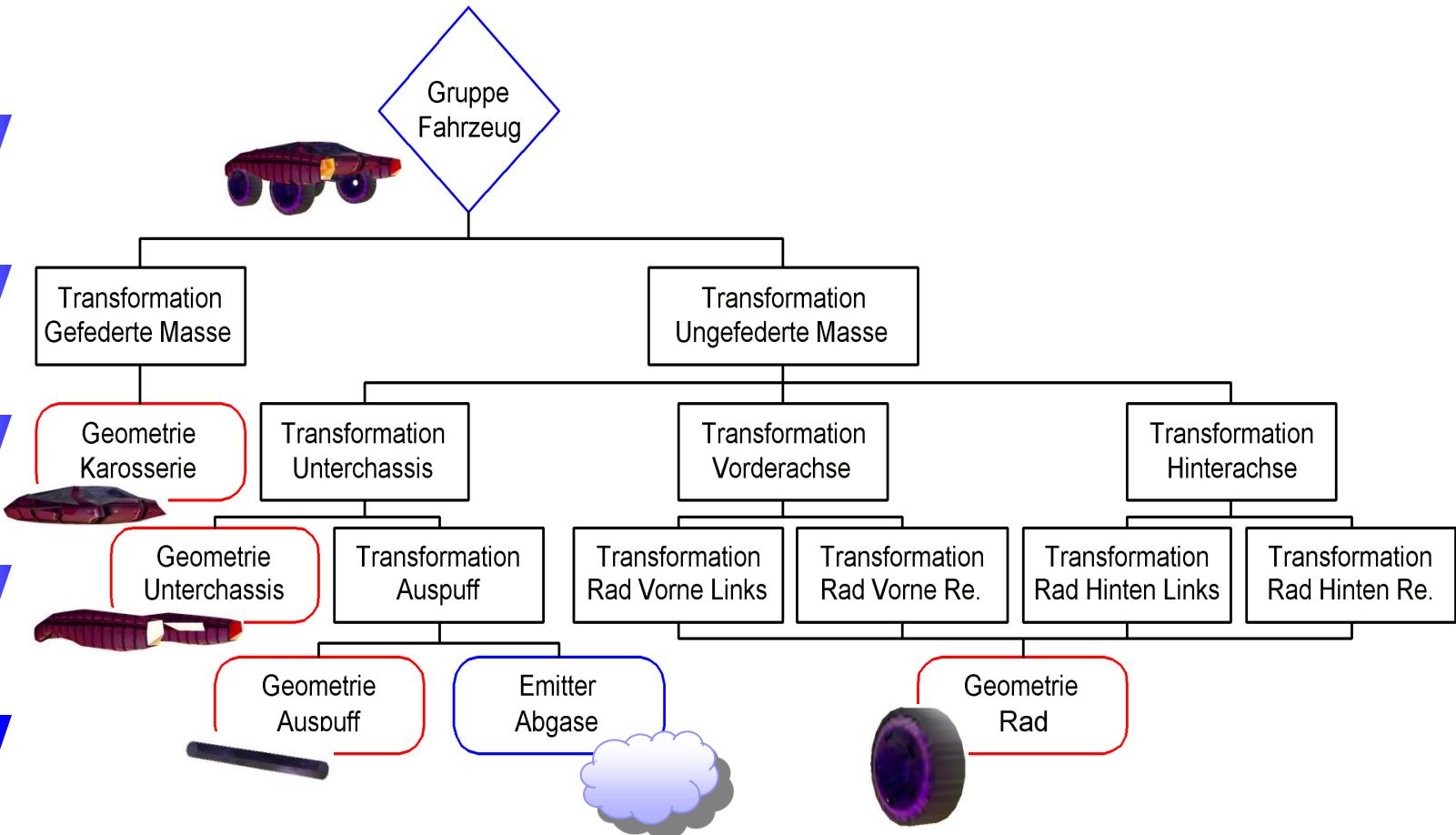
4 // /

5 // /



Bildung der Objekthierarchie

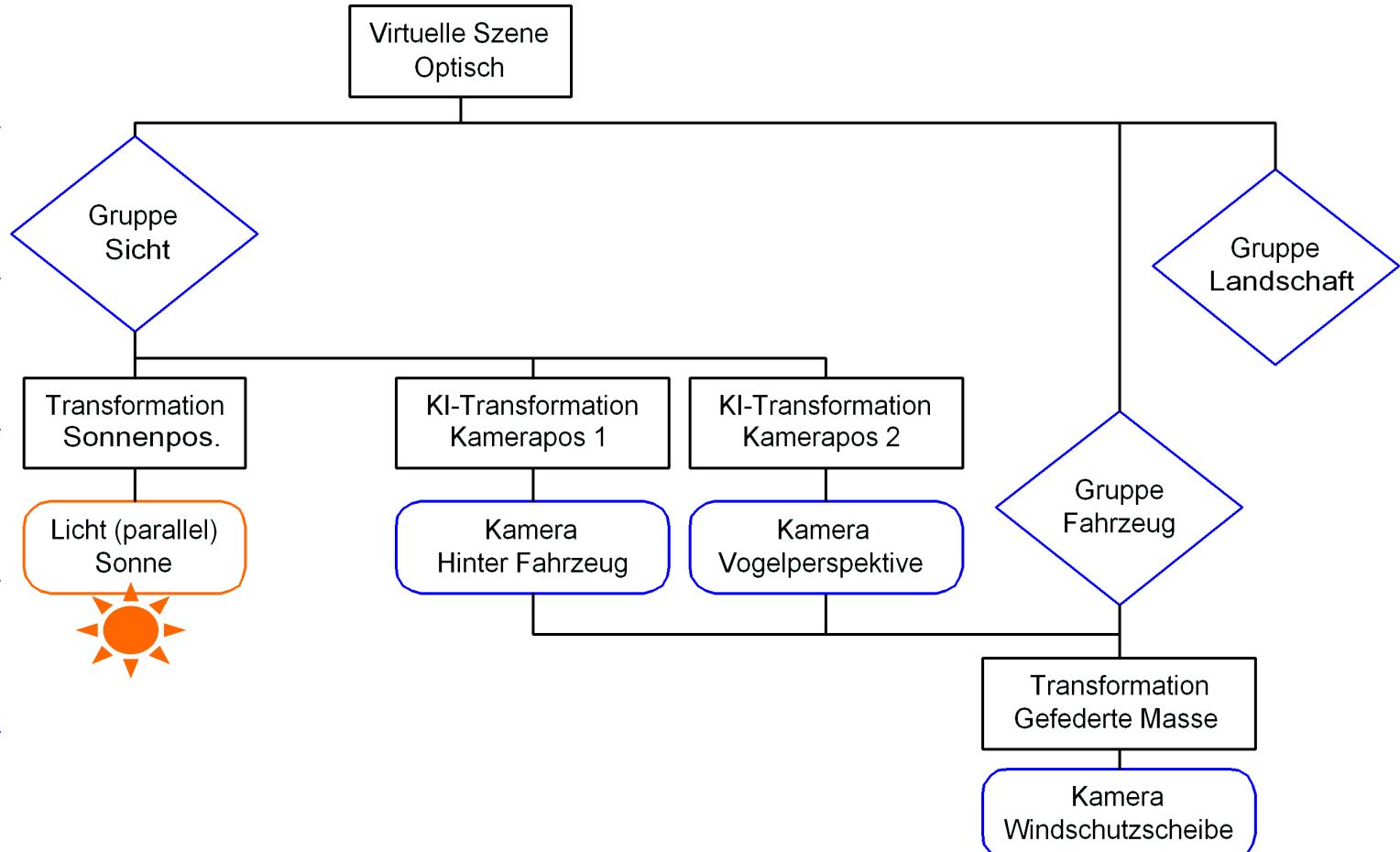
Hierarchie des Fahrzeuges



Bildung der Objekthierarchie

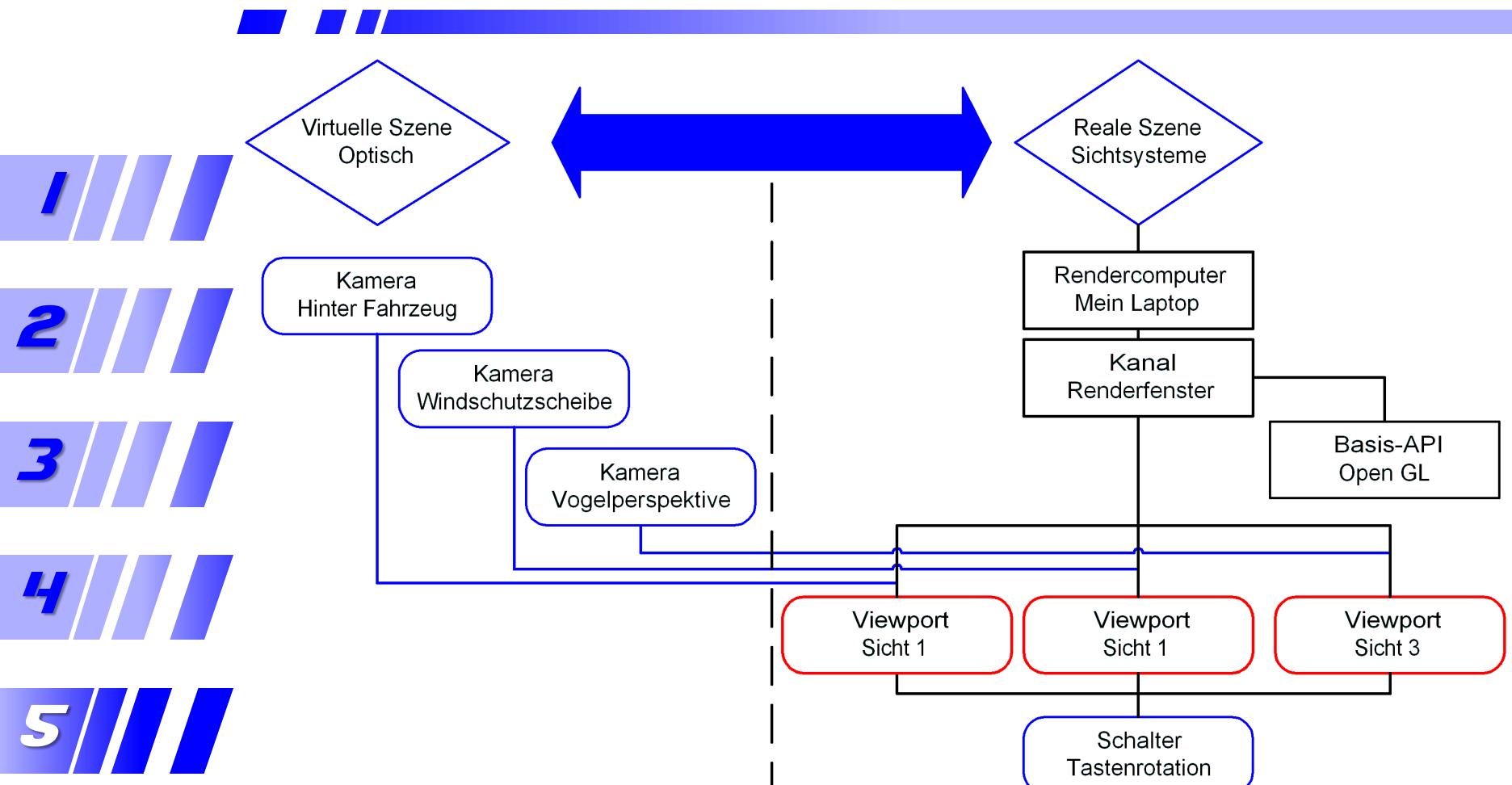
Hierarchie der Virtuellen Szene

1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /



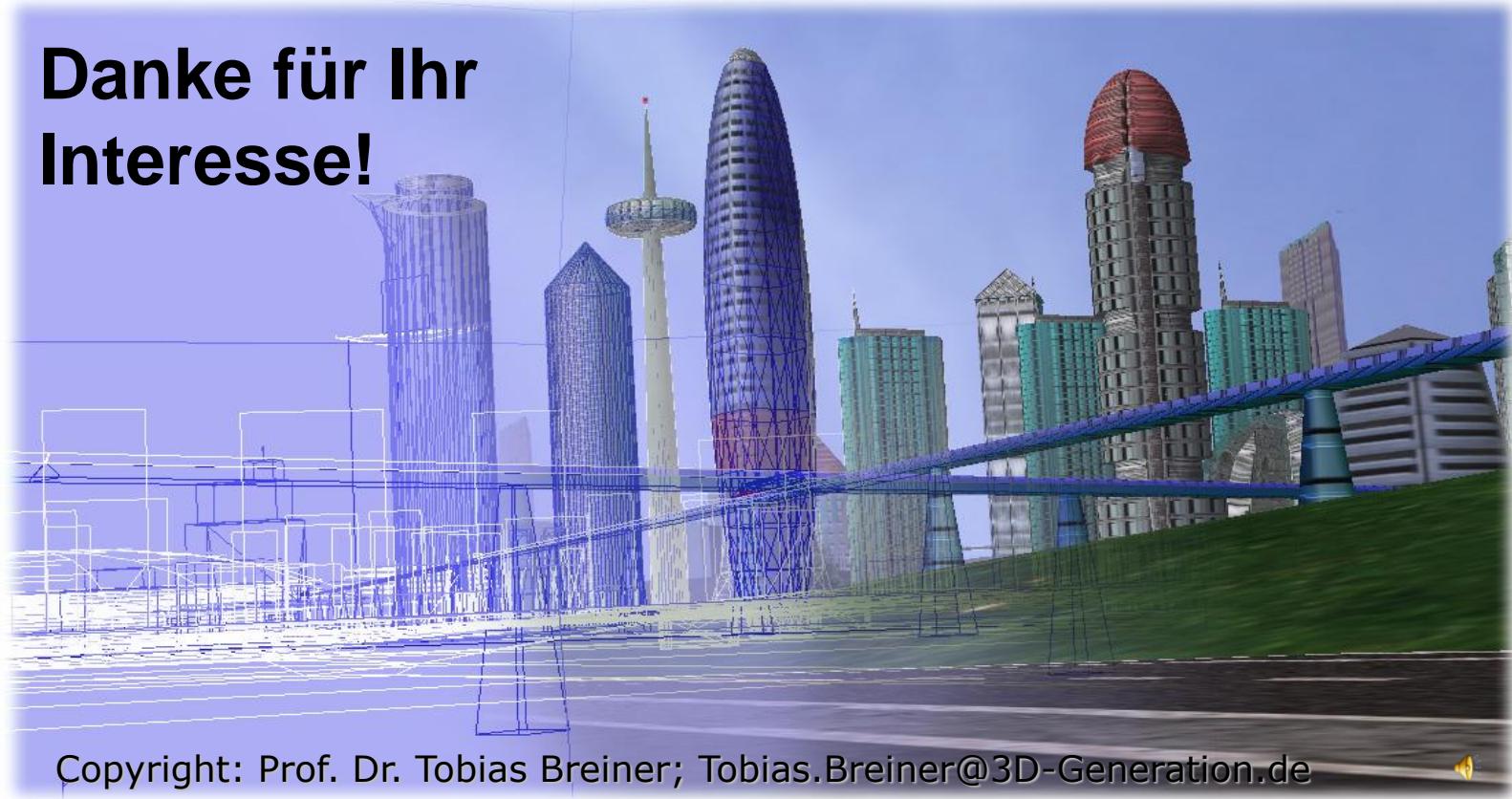
Bildung der Objekthierarchie

Virtuelle vs. Reale Szene



|||||GAME ||||OVER

Danke für Ihr
Interesse!



Copyright: Prof. Dr. Tobias Breiner; Tobias.Breiner@3D-Generation.de



||||PROF. |||DR. |||TOBIAS |||BREINER
||||VEKTORIA |||ENGINE |||MANUAL

45 VON 45
||||PLACEMENTS