

VEKTORIA-MANUAL **ERSTE** **PROGRAMME**



Prof. Dr. Tobias Breiner

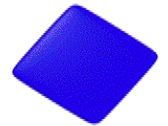
vektoria

www.vektoria-engine.com

Game Design

Inhalt

////BILLBOARDS //



////APPENDAGES //

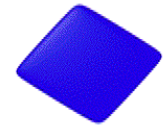
////SKY & ////GROUND //

////POINTING //

////STEREOSCOPIC ////PL. //



////BILLBOARDS



2////

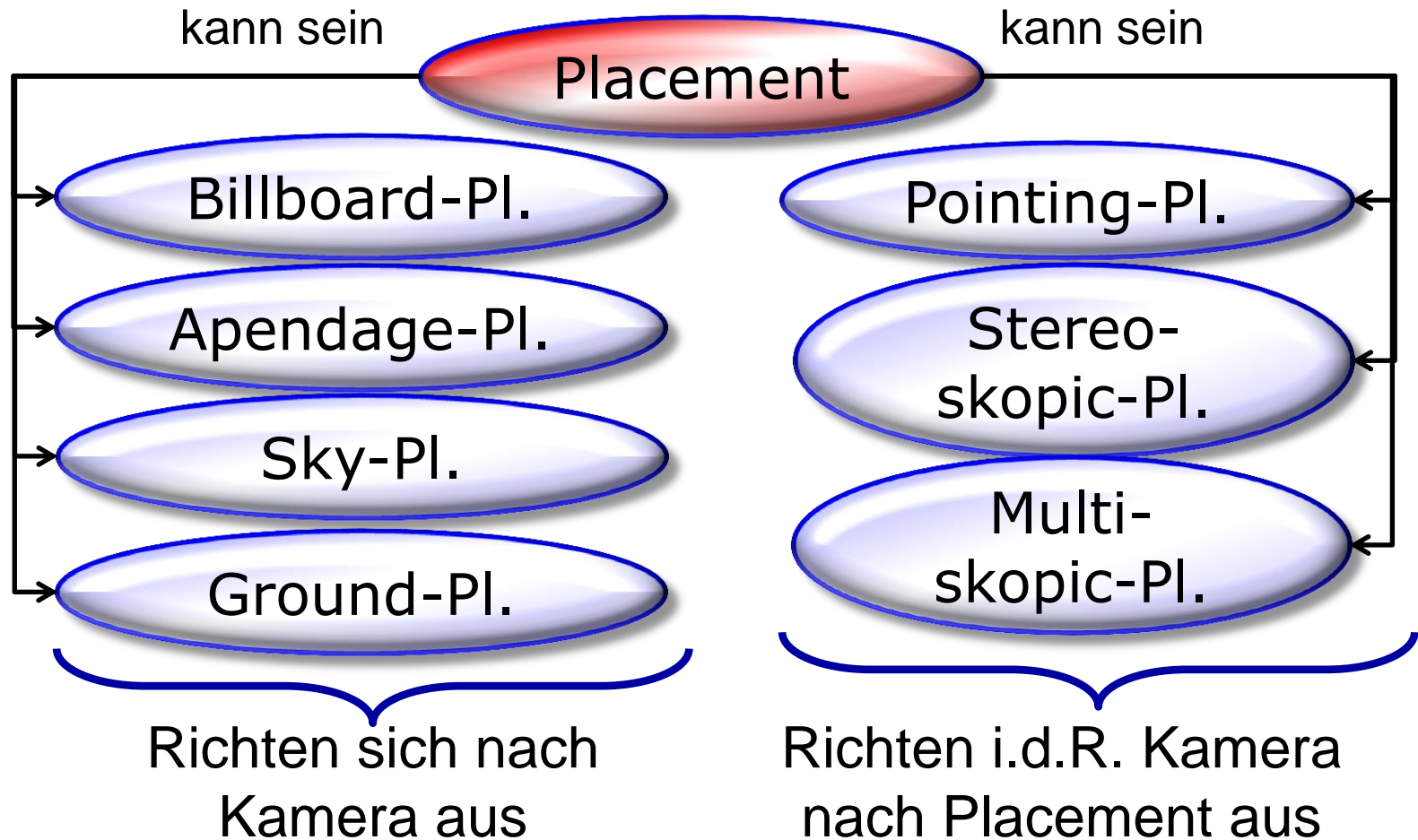
3////

4////

5////



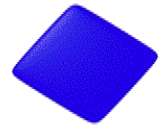
Besondere Placements



Billboards



//// **BILLBOARDS** //



2 //

3 //

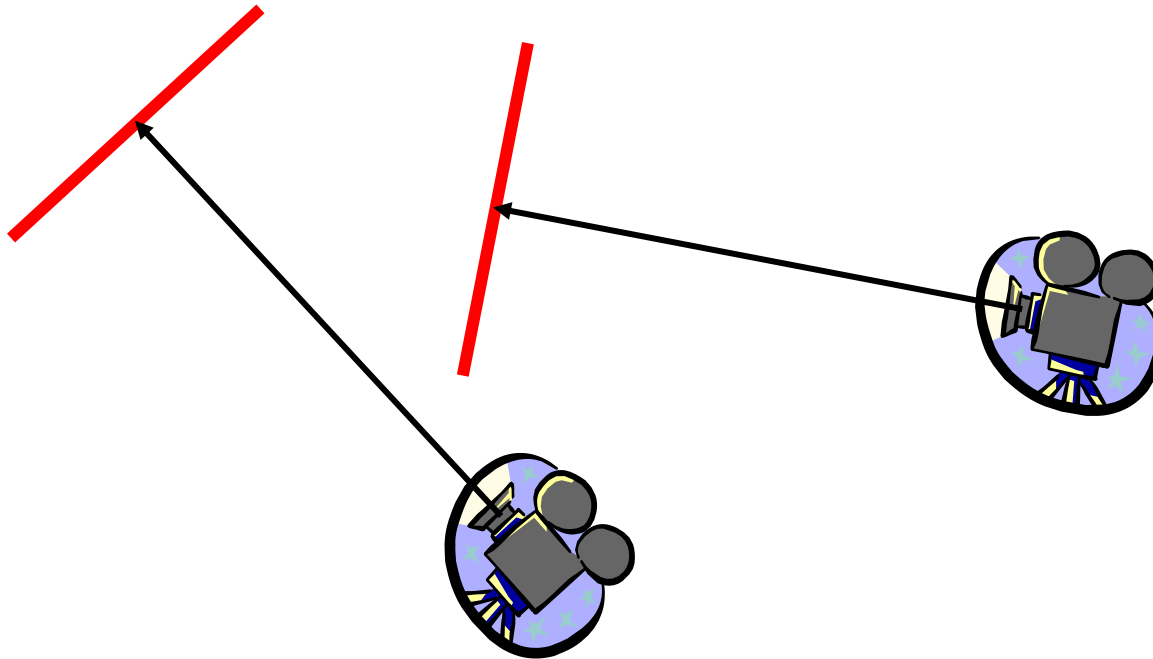
4 //

5 //




Was sind Billboards?

Billboards sind virtuelle Objekte (meist CGeoQuads), die immer in Richtung des Betrachters (Kamera) orientiert sind.




Billboards in Vektoria





Billboards sind in Vektoria als eine Eigenschaft von Placements implementiert



=>



Erlaubt große Flexibilität, denn man kann jede beliebige Geometrie als Billboard an das Billboard-Placement anhängen.





Billboards

SetBillboard()



void SetBillboard();

Diese Methode der Klasse CPlacement orientiert das untergeordnete Objekt stets in Richtung der Kamera aus.



void SetBillboardX();
void SetBillboardY();
void SetBillboardZ();

Diese Methoden der Klasse CPlacement orientieren das untergeordnete Objekt ebenfalls in Richtung der Kamera, aber fixieren das Billboard an einer kartesischen Achse.



BillboardAngle-Funktion



```
void SetBillboardAngle(float fa);
```

Diese Methode der Klasse CPlacement dreht ein Billboard-Placement mit dem Winkel fa um die Sichtachse. Achtung, fa wird im Bogenmaß angegeben. Die Methode ist vor allem sinnvoll, um Partikeleffekte natürlicher aussehen zu lassen.

Und dies ist die dazugehörige Getter-Methode, welche den aktuell eingestellten Drehwinkel um die Sichtachse ausgibt. Sie gibt nur bei Billboards sinnvolle Werte aus.

```
float GetBillboardAngle();
```



BillboardScaling-Funktion



```
void SetBillboardScaling(float fx, float fy);
```



Diese Methode der Klasse CPlacement skaliert ein Billboard-Placement mit dem Faktor (fx,fy).

Partikel-Billboards können auf diese Weise z.B. wachsen.



Billboards Übung



Erzeugen Sie ein Billboard-Quad mit einer Alpha-Textur!

1

2

3

4

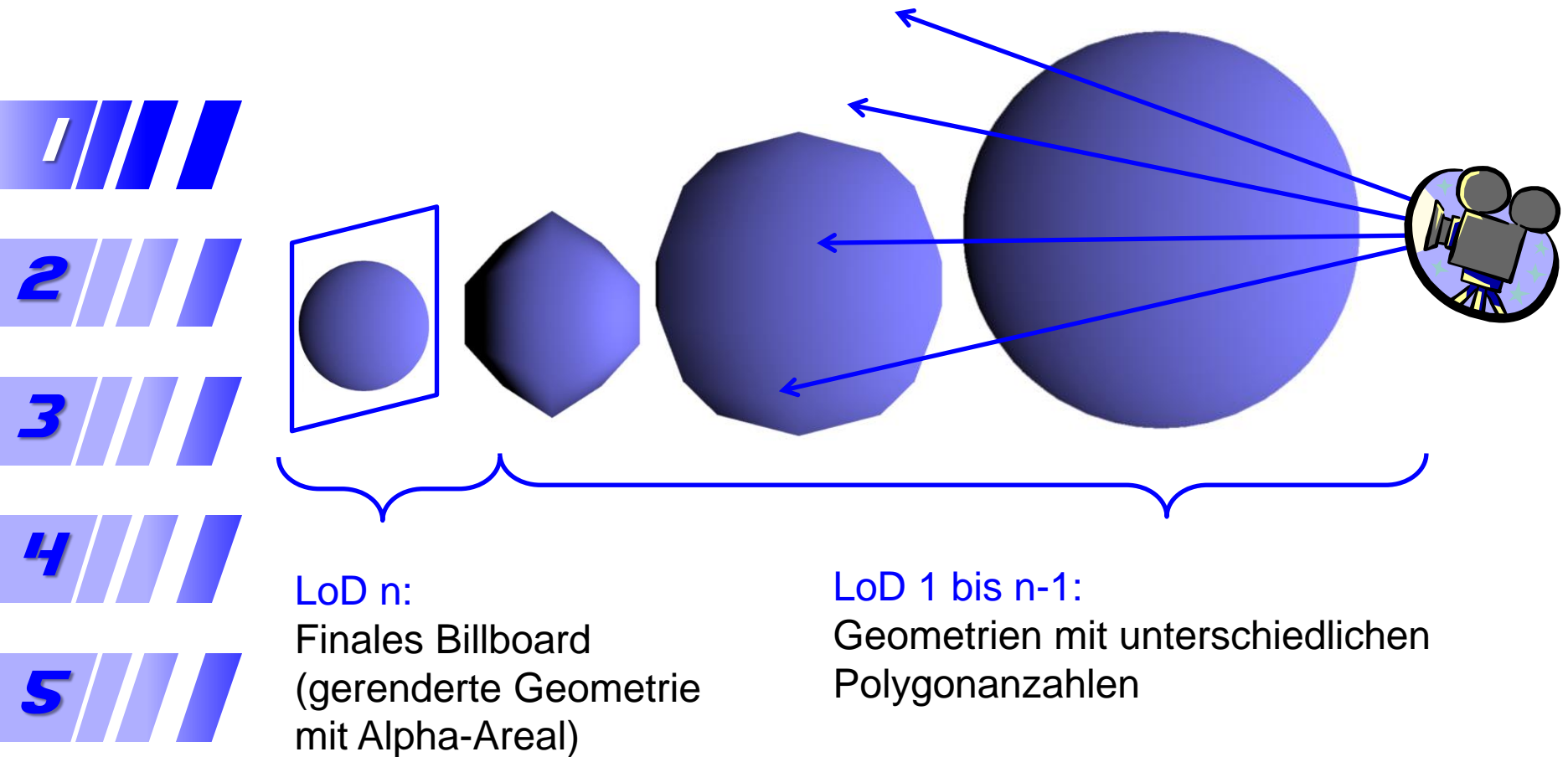
5

Freiwillige Zusatzaufgabe für die schnellen Nerds:

Erzeugen Sie einen Mond, der immer in Richtung des Betrachters zeigt!



LoDs mit finalem Billboarding



Billboards

Schatten-Billboarding

1



Billboard
ohne
Schatten

2



Schatten
auf
normales
Billboard

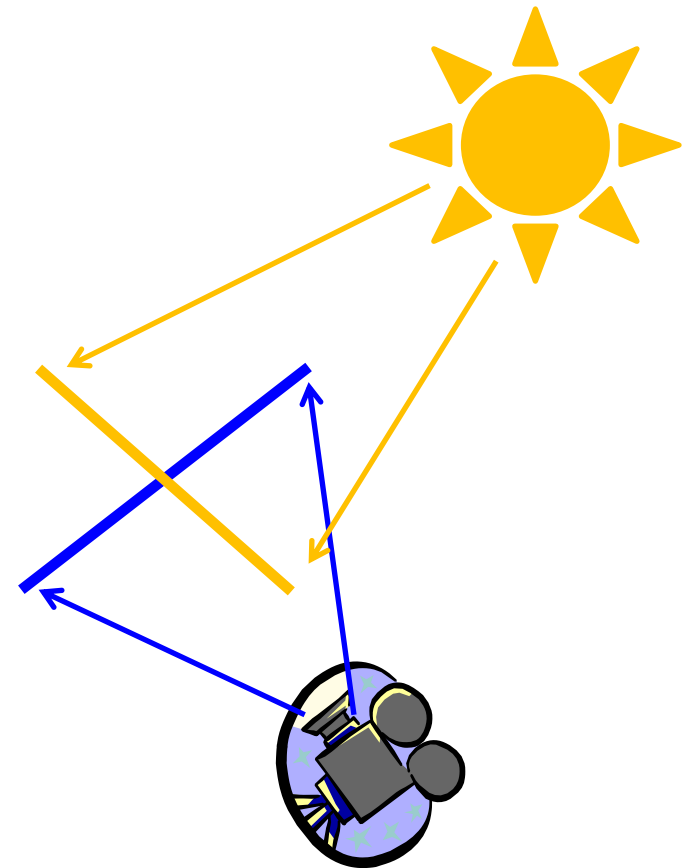
3



Billboard
mit
zweitem
Schatten-
billboard

4

5

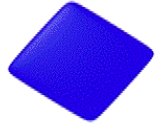




Kapitel 2

1

APPENDAGES



3

4

5





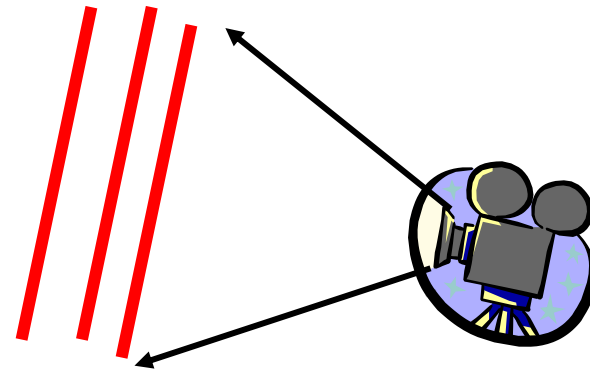
Billboards (Variationen)

- **Billboards mit Alpha-Textur:**
Billboards mit transparenten und opaken Anteilen (fast immer)
- **Multi-Billboards:**
Mehrere hintereinanderliegende Schichten von Billboards
- **Axis alined Billboards:**
Billboards, die sich nur um eine definierte Achse drehen können (in der Regel y-Achse).
- **Criss-Cross-Billboards:**
Billboards, die zusätzliche Flächen beinhalten, welche die Hauptfläche durchdringen
- **Video-Billboards:**
Billboards mit einer Videotextur



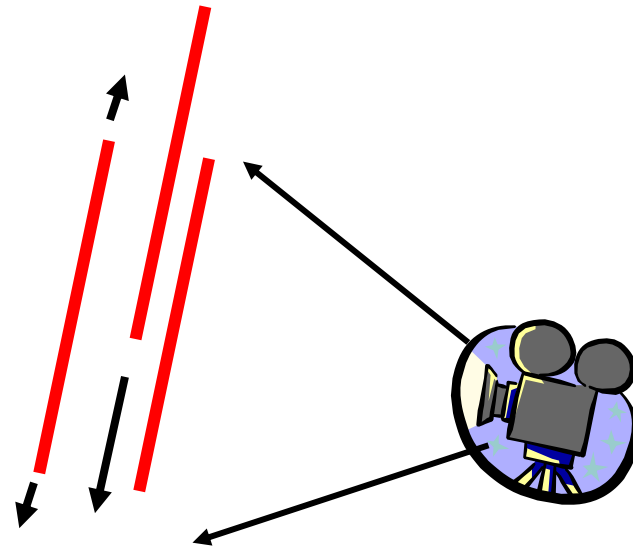
Multibillboards

- Mehrere hintereinanderliegende Schichten von Billboards
- Vorteile:
 - Tiefenschärfe (wenn Rendermodell dies unterstützt)
 - Interne perspektivische Verschiebungen bei Nahbetrachtung



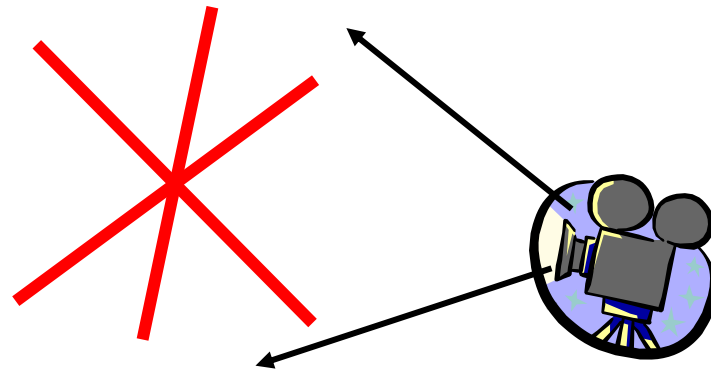
Multi-BBs mit Verschiebungen/Skalierungen

- Multibillboards mit Internen Verschiebungen und/oder Skalierungen
- Vorteile:
 - Formänderungen (z.B. für Wolkenveränderungen, Rauch,o.ä.)



Criss Cross Billboards

- Billboards mit zusätzlichen metaorthogonalen Schnittflächen
- Vorteile:
 - Tiefenschärfe (wenn Rendermodell dies unterstützt)
 - Interne perspektivische Verschiebungen bei Nahbetrachtung
 - Gut geeignet für Bäume





```
void SetAppendage();
```

Mit der Methode SetAppendage der Klasse CPlacement kann man Criss-Cross- & Multibillboards erzeugen. Dazu erzeugt man ein Billboard-Placement und hängt an dieses mehrere Appendage-Placements mittels AddPlacement an. An jedes Appendage-Billboard kann man wiederum ein Quad mit einer Alphatextur anheften.





Multibillboards Übung



Erzeugen Sie einen überzeugenden
Multibillboard-Nebel!

Freiwillige Zusatzaufgabe für die schnellen
Nerds:

Erzeugen Sie überzeugende Multibillboard-
Rauchschwaden, die aufsteigen und sich im
Himmel in Luft auflösen!





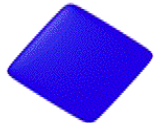
Kapitel 3



1

2

SKY & GROUND



4

5





Billboards Ground



`void SetGround();`

Mit der Methode `SetGround()` der Klasse `CPlacement` kann man Objekte erzeugen, die quasi in einer gewissen Position zur Kamera „fixiert“ sind.

Damit lassen sich zum Beispiel Armaturen erzeugen.





void SetSky();

1
2
3
4
5

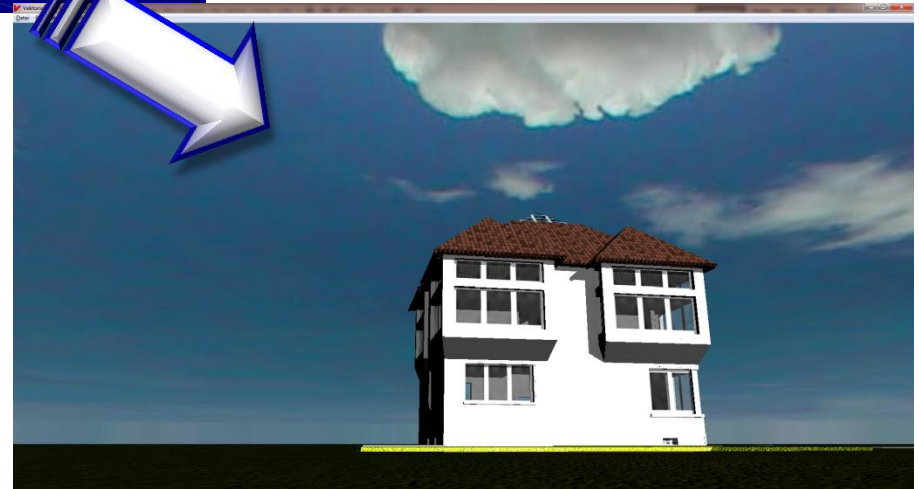
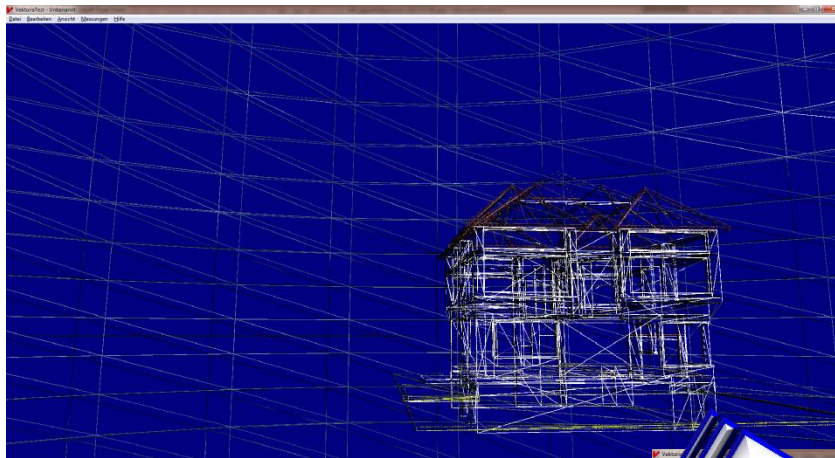
Mit der Methode SetSky() der Klasse CPlacement kann man Objekte erzeugen, die mit der Kamera mitlaufen, aber nicht die Orientierung der Kamera teilen.

Damit lassen sich hervorragend Sky-Domes, Sky-Boxes oder ähnliches anhängen. Für Multiskydomes bieten sich die schon bekannten Appendages an.



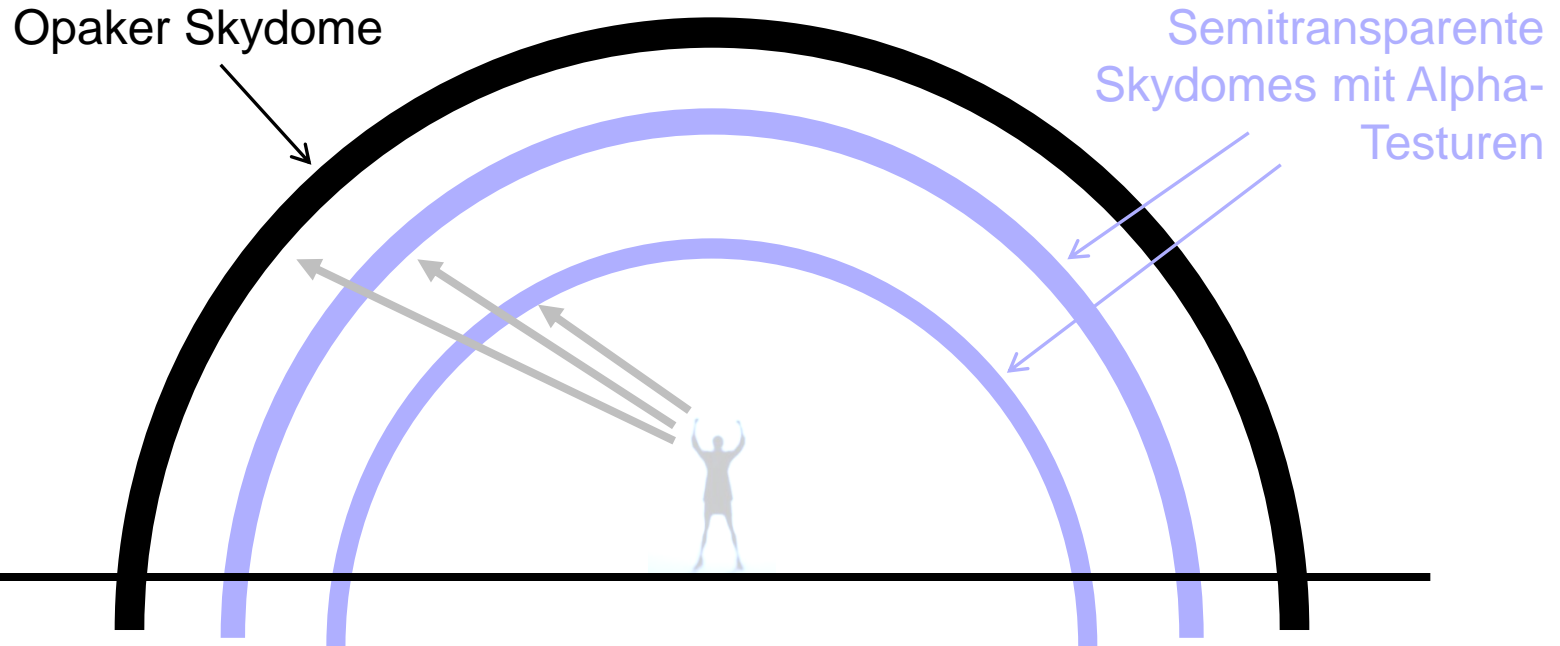
Sky- und Ground-Placements

Skydome



Sky- und Ground-Placements

Multi-Skydomes



Mehrere Schichten:

äußerste opak, innere halbtransparent





Sky- und Ground-Placements Übung



Erzeugen Sie einen Skydome mit
Sternenhintergrund!

Freiwillige Zusatzaufgabe für die schnellen
Nerds:

Erzeugen Sie einen 3er-Skydome mit einem
Sternenhintergrund und 2 Schichten von
interstellarem Nebel!

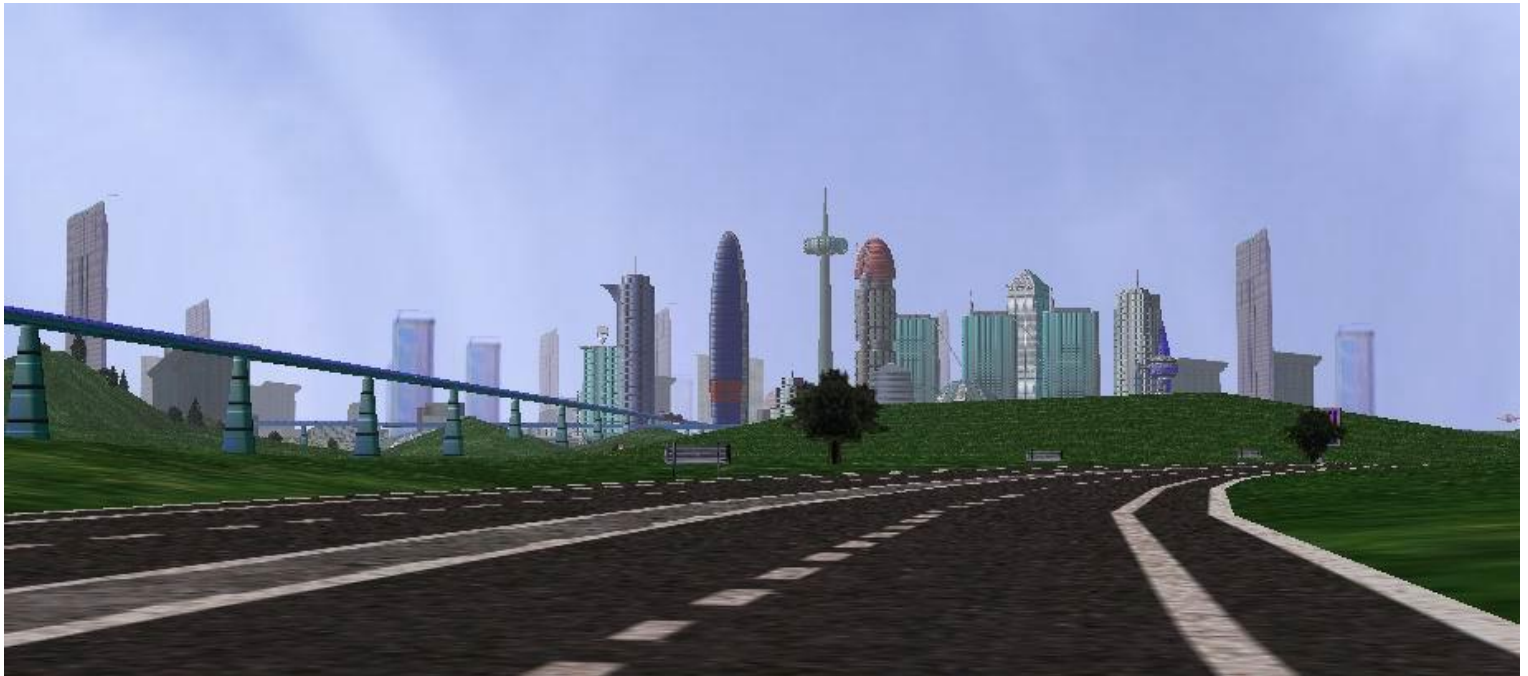


Die „fünf Freunde“ der 3D-Echtzeit

- I. Viel in Initialisierung verschieben!
- II. Maschinencodenahe programmieren!
- III. Polygone einsparen!
- IV. Angemessene physikalische Modelle verwenden!
- V. Performanten Szenegraphen wählen!



Polygoneinsparungen am Beispiel



- Stratopolis-Testlandschaft besteht aus ca. 300.000 Polygonen
- 9*9 km großes virtuelles Areal
- Lief schon anno 1998 über 20fps



Beschleunigung für Echtzeitanwendungen

Polygoneinsparungen

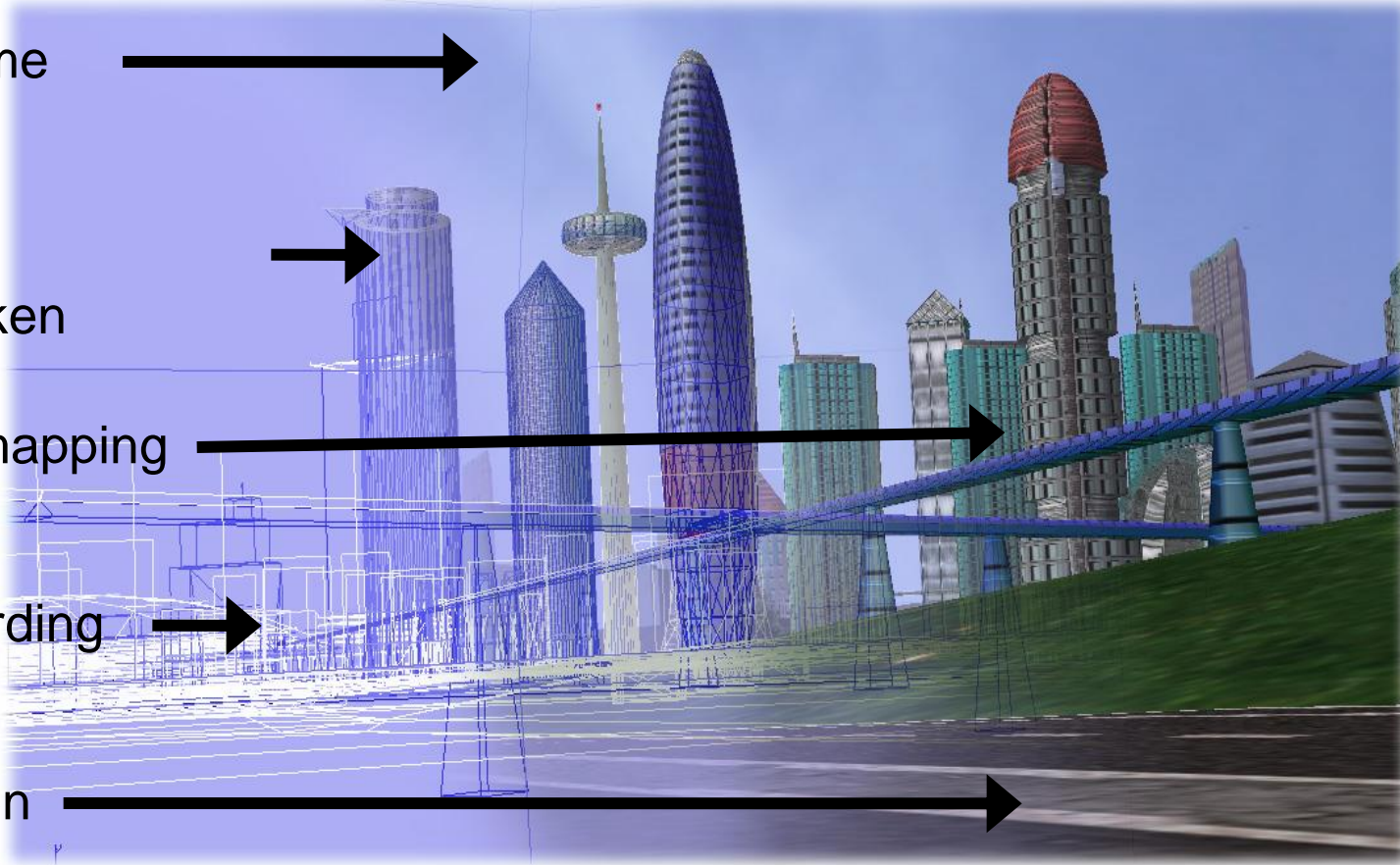
1 // // // Skydome →

2 // // // L.o.D.-
Techniken →

3 // // // Bumpmapping →

4 // // // Billboarding →

5 // // // Texturen →



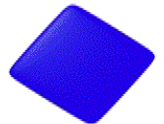
Kapitel 4

1

2

3

POINTING-PLACEMENTS



5



Knotenobjekte der Szenegraphen

Zielgerichtete Kameras



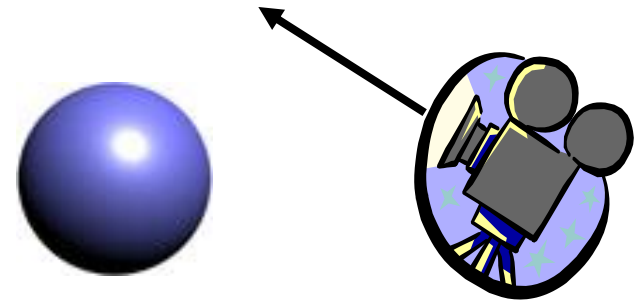
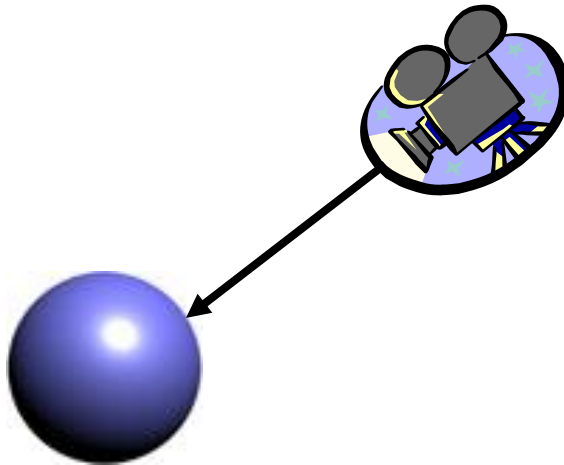
zielgerichtete

2 Positionen (Ort und Ziel)



zielungebundene

Position (Ort) und
Orientierung (Zielrichtung)



In Vektoria sind zielgerichtete Kameras, durch sogenannte „pointing placements“ eleganter gelöst.





Pointing Placements

Pointing Placements



An Pointing Placements lassen sich nicht nur Kameras anhängen, sondern prinzipiell auch:

- Geometrien,
- Emitter,
- Spot-Lights,
- Andere Placements

Dies erlaubt eine hohe Flexibilität.

Beispiele:

- Spot-Lichter, die immer ein spezielles Objekt anleuchten.
- Objekte, die immer auf ein anderes Objekt ausgerichtet sind (Scharfschützen, Magneten) etc.





Pointing Placements

Placementorientierte P.P.



```
void SetPointing(CPlacement * pplacementPointing);
```

macht, dass das Placement automatisch sich in Richtung des Placements orientiert, der durch pplacement gegeben ist



Vektororientierte P.P.



Ein Pointing-Placement kann nicht nur auf ein anderes Placement, sondern alternativ auf einen speziellen Raumpunkt ausgerichtet sein.

`void SetPointing(CHVector * pvectorPointing);`

macht, dass sich das Placement automatisch in Richtung des Raumpunkts orientiert, der durch `pvectorPointing` gegeben ist.



Pointing-Pl. ausschalten



```
void SetPointingOff();
```

Schaltet Pointing-Funktion des Placements wieder aus, so dass das Placement wieder „normal“ ist.



Kapitel 5

Kapitel 5

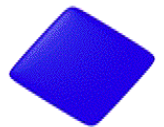
1

2

3

4

STEREOSCOPIC PL.



PROF. DR. TOBIAS BREINER
VEKTORA MANUAL

36 VON 41
SPEZIELLE PLACEMENTS

Stereoskopie



Um stereoskopische Kameras elegant zu managen,
gibt es folgende Methoden in CPlacement:

1

2

3

4

5



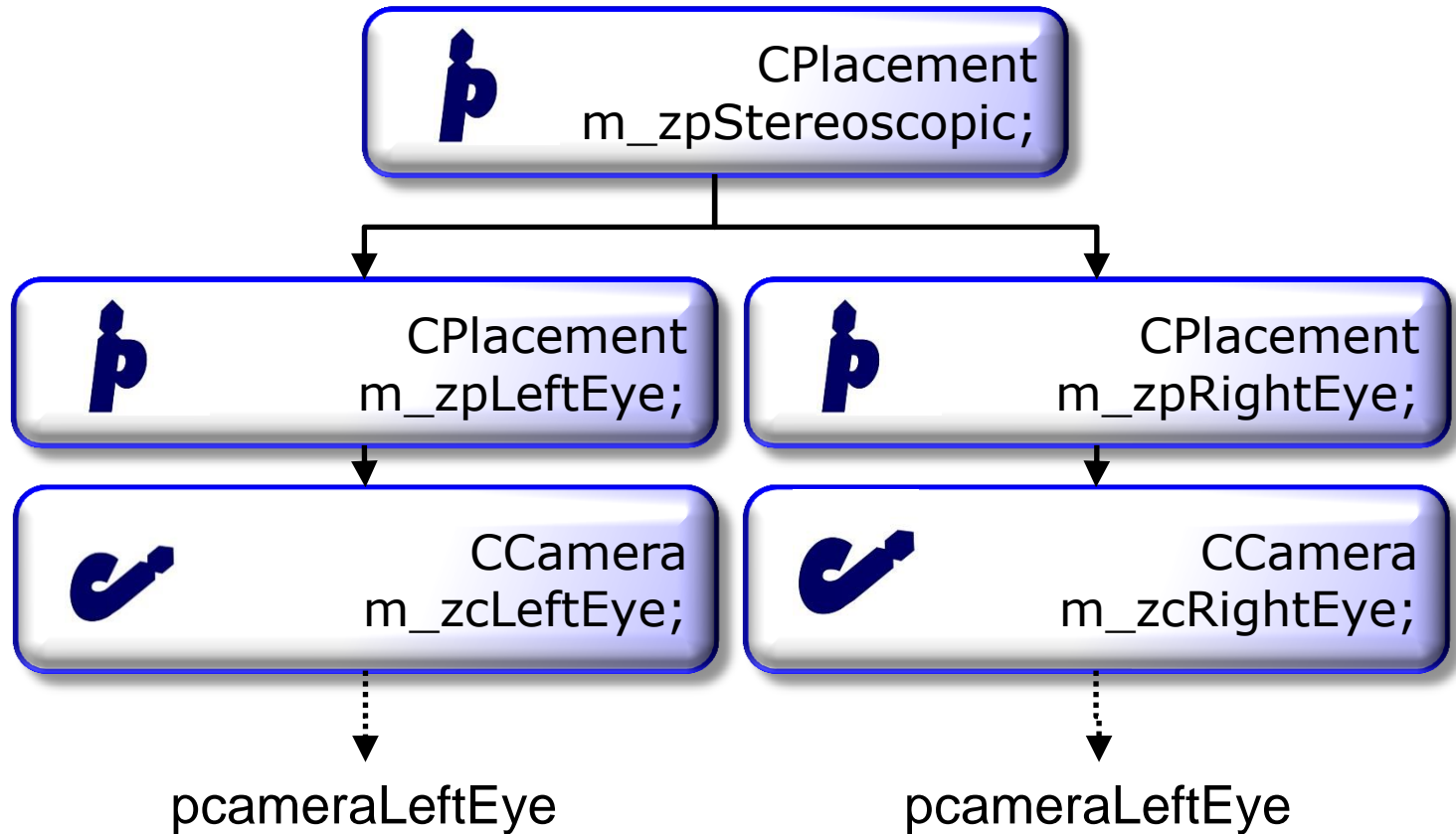
Stereoskopie



Erzeugt eine komplexe Struktur aus drei Placements und zwei Kameras für Stereoskopie

```
void MakeStereoscopicCameras(  
    CCamera *pcameraLeftEye, // Pointer zur linken Kamera  
    CCamera *pcameraRightEye, // Pointer zur rechten Kamera  
    float fEyeDistance, // Distanz zwischen linker und rechter Kamera  
    float fFocusDistance, // Fokussierungsdistanz für die Kameras  
    float faFov=2.0F, // Horizontaler Öffnungswinkel im Bogenmaß  
    float fNearClipping=0.1F, // Nahschnittebene des Sichtfrustrums  
    float fFarClipping=1000.0F); // Fernschnittebene des Sichtfrustrums
```







`void KillStereoscopicCameras();`

1 // // //
Zerstört die aufgebaute Struktur für stereoskopische Kameras wieder.

2 // // //
`void SetStereoscopicParameters
(float fEyeDistance, float fFocusDistance);`

3 // // //
Verändert die Parameter der stereoskopischen Struktur, wenn vorhanden

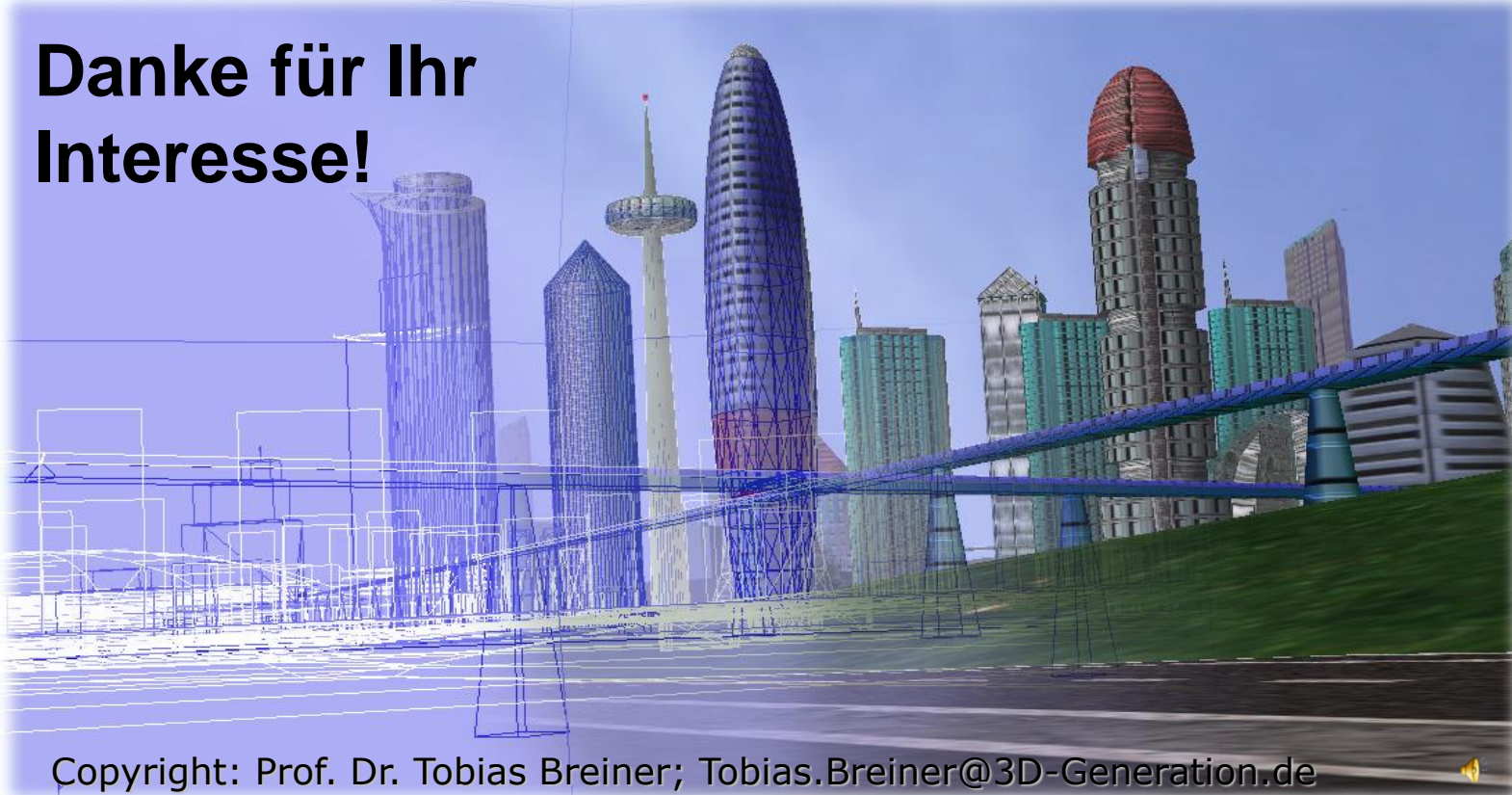
4 // // //
`bool GetStereoscopicParameters(float & fEyeDistance,
float & fFocusDistance);`

5 // // //
Gibt die Parameter der stereoskopischen Struktur zurück, gibt true aus, wenn vorhanden



/// **GAME OVER**

**Danke für Ihr
Interesse!**



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



/// **PROF. /// DR. /// TOBIAS /// BREINER**
/// **VEKTORA /// MANUAL**

41 VON 41
/// **SPEZIELLE /// PLACEMENTS**