



VEKTORIA-MANUAL

VORGEFERTIGE

GEOMETRIEN



Vorgefertigte Geometrien in Vektoria

Inhalt



/// **UEBERSICHT**



/// **GEOMETRIEPRIMITIVE**



/// **SWEEEPING & EXTRUDING**



/// **UEBUNGEN**



/// **ELLIPSOID-// MAPPING**



Kapitel 1

Übersicht über Geometrien



/// UEBERSICHT ///



2 // / / /

3 // / / /

4 // / / /

5 // / / /



PROF. DR. TOBIAS BREINER
VEKTORIA MANUAL

3 VON 61
VORGEFERTIGTE GEOMETRIEN



Geometrien



Oberflächenform der Objekte bestehend

aus Vertieces und Polygonen und
Zusatzinformationen der Facetten wie
Flächennormalem, Vertice-Normalen,
Texturkoordinaten, Texturpointer, ...

Synonyme in anderen Szenegrafen:

Shapes, Meshes, GeoData

In anderen Szenegrafen auch oft intern
abgelegt als Indexed Face Sets:





Parametrisierbare geometrische Primitive

CGeo: Die Geometrieklasse in Vektoria



Polygonveränderungsmethoden:



void Flip();

Kehrt alle Normalenvektoren der
Geometrie um.
Muss vor der Geometrieinitialisierung
aufgerufen werden.





Kapitel 2

Kapitel 2



/// GEOMETRIEPRIMITIVE



Planare Primitive

CGeoTriangle
CGeoQuad

2D-Dreieck
2D-Viereck (aus 2 Dreiecken)

Die planaren Primitive sind jeweils als TriangleStrip realisiert und eignen sich vor allem für Böden oder Billboards.



Konkave Primitive als Strip

CGeoSphere

Kugel

CGeoEllipsoid

Ellipsoid

CGeoDome

Kuppel

2 // / / / Diese konkaven Primitive sind jeweils als TriangleStrip realisiert. Dies hat den Vorteil, dass sie ca. 30% schneller als die entsprechenden List-Pendants sind (siehe nächste Seite), aber den Nachteil, dass sie nicht beliebig repetativ texturierbar sind.

3 // / / / CGeoEllipsoid und CGeoDome können übrigens auch nur Teilbereiche des jeweiligen Körpers erzeugen)

4 // / / /



Konkave Primitive als List

CGeoSphereTiled	Kugel
CGeoEllipsoidTiled	Ellipsoid
CGeoDomeTiled	Kuppel

1 // / / / Diese konkaven Primitive sind die TriangleList-Pendants zu CGeoSphere, CGeoEllipsoid und CGeoDome (siehe Seite vorher).

2 // / / / Sie sind zwar ca. 30% langsamer, aber dafür leichter weitermodellierbar und können beliebig repetativ texturiert werden.

3 // / / / Sie eignen sich daher eher für SkyDomes oder für Körper, die eine feine Granularität benötigen, z.B. Planeten mit einer animierten Wassertextur.





Sweep-Primitive (als Strip)

1 // / / / CGeoCone

CGeoCylinder

CGeoTube

Kegel

Zylinder

Röhre

(optional auch gebogene)

2 // / / / GeoSlice

CGeoArrow

3D-Kuchenstückform

3D-Pfeil

3 // / / /

Diese extrudierten Primitive leiten sich allesamt von der Klasse CSweep ab und sind daher TriangleStrips.

4 // / / / Die Böden, Enden oder Innenwände der jeweiligen Körper können optional auch weggelassen werden.

5 // / / /



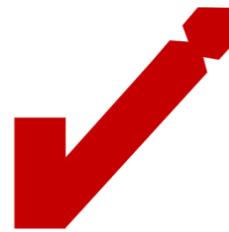
Debugging-Primitive

CGeoInfoVertex
CGeoInfoGeo

visualisierte Vertexdaten
visualisierte Geometriedaten

Diese Primitive sind speziell für das optische Debugging entwickelt worden.
Sie können z.B. bei Vertices die Position, die Normale, die Tangente und die Bitangente anzeigen.
Oder bei übergebenen Geometrien die Daten aller Vertices.





Polyeder-Primitive

CGeoTetraeder

CGeoCube

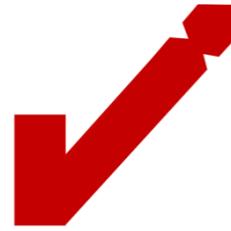
Tetraeder
Quader,
kann natürlich auch für die
Erzeugung eines Würfels
verwendet werden

CGeolkosaeder CGeoFrustum

Ikosaeder Pyramidenstumpf

Diese Primitive sind als Triangle List implementiert.



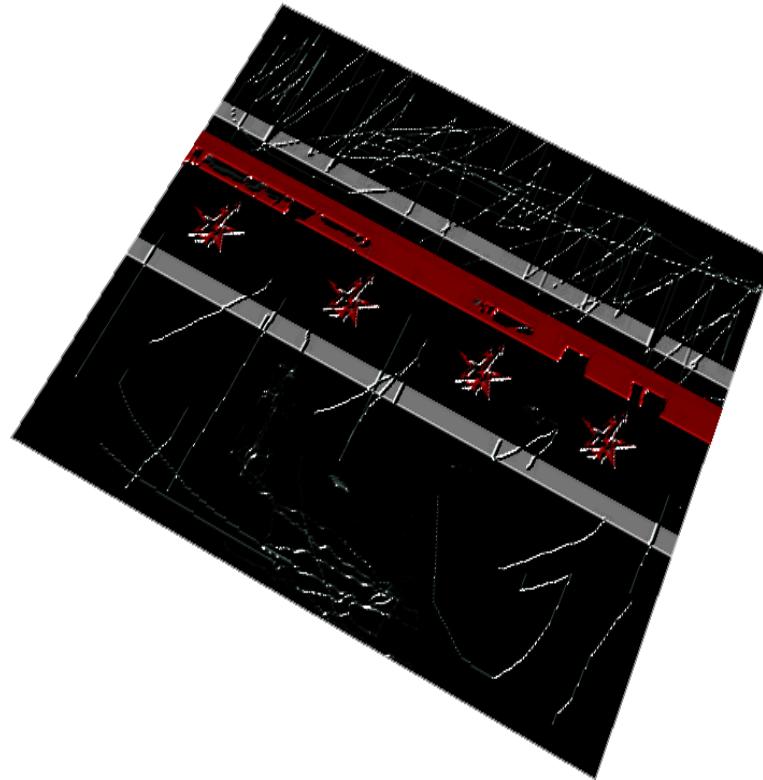


Parametrisierbare geometrische Primitive

CGeoQuad



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





Parametrisierbare geometrische Primitive

CGeoQuad::Init



Initialisierungsmethode von CGeoQuad:

void Init

(float fxSize,

float fySize,

CMaterial * pmaterial);

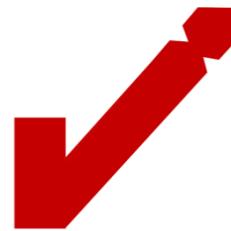
Breite des Rechtecks

Höhe des Rechtecks

Zeiger auf Material,
welches auf das Rechteck
gemappt wird.



CGeoSphere::Init



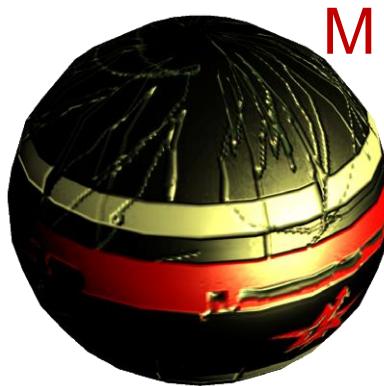
1 // / / /

2 // / / /

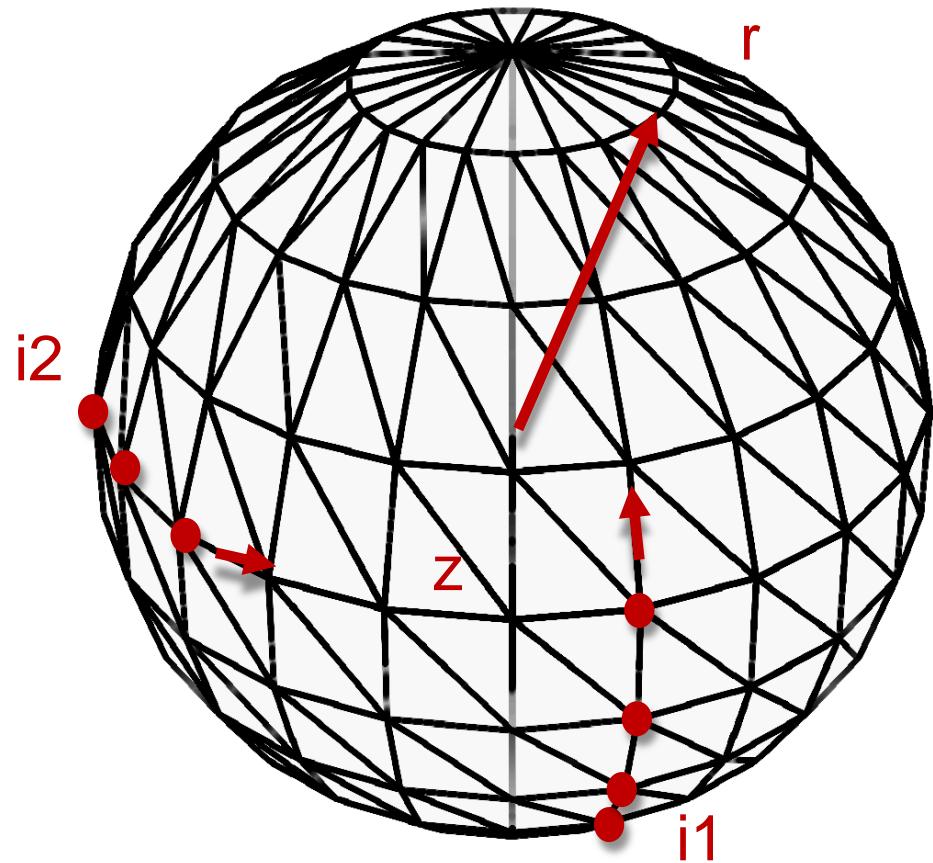
3 // / / /

4 // / / /

5 // / / /



r = fRadius
M = pMaterial
i1 = iLongitude
i2 = iLatitude





Parametrisierbare geometrische Primitive

CGeoSphere



Initialisierungsmethode von **CGeoSphere**:

1 // / / / void Init

(

2 // / / / float fRadius,

Radius der Kugel

3 // / / / CMaterial * pmaterial,

Zeiger auf das Material, welches
auf die Kugel gemappt wird.

4 // / / / int iLongitude = 12,

Anzahl der Vertices von
"Pol zu Pol"

5 // / / / int iLatitude = 12,

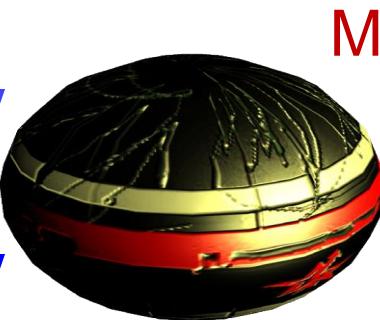
Anzahl der Vertices um den „Äquator“

5 // / / / Int eMapping);

UV-Mapping-Art



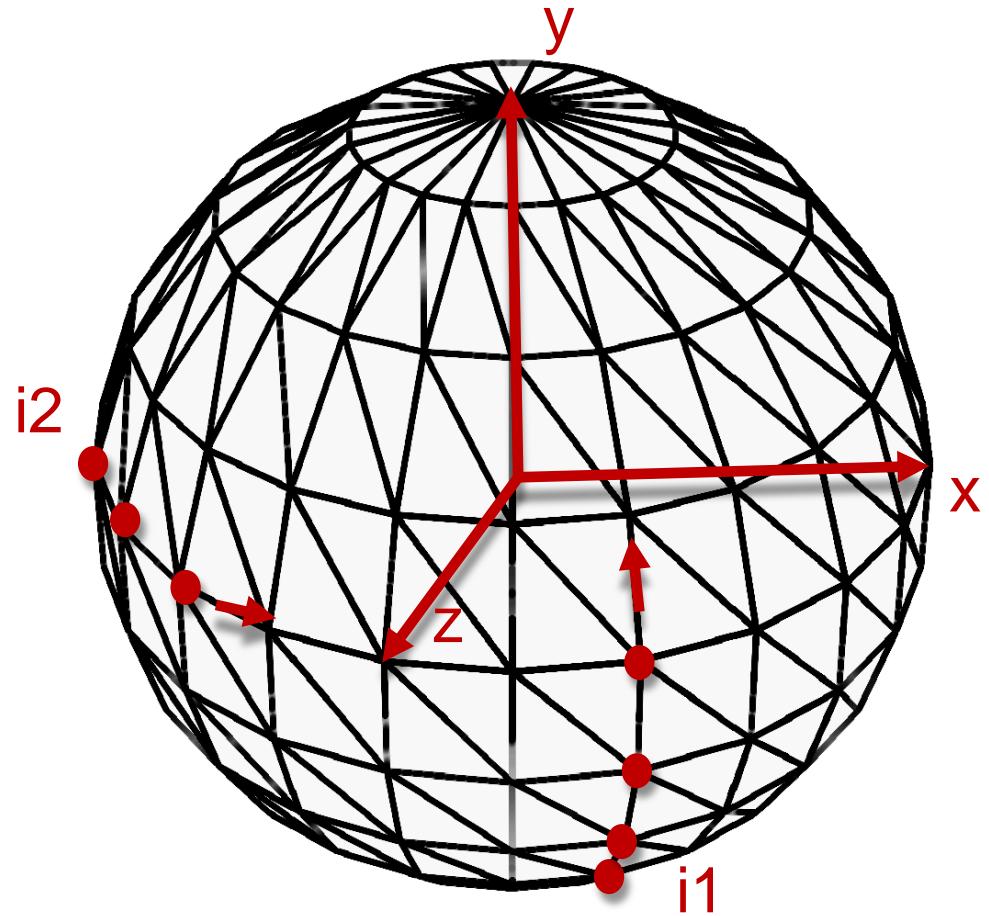
CGeoEllipsoid



M

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

x = vSize.m_fx
y = vSize.m_fy
z = vSize.m_fz
L = fLength
M = pmaterial
i1 = iLongitude
i2 = iLatitude





CGeoEllipsoid::Init



Initialisierungsmethode von **CGeoEllipsoid**:

1 // / / / void Init

2 // / / / (

CHVector vSize,

Größe in X,Y,Z-Richtung

CMaterial * pmaterial,

Zeiger auf Material, welches auf
das Ellipsoid gemappt wird.

4 // / / / int iLongitude = 12,

Anzahl der Vertices von
"Pol zu Pol"

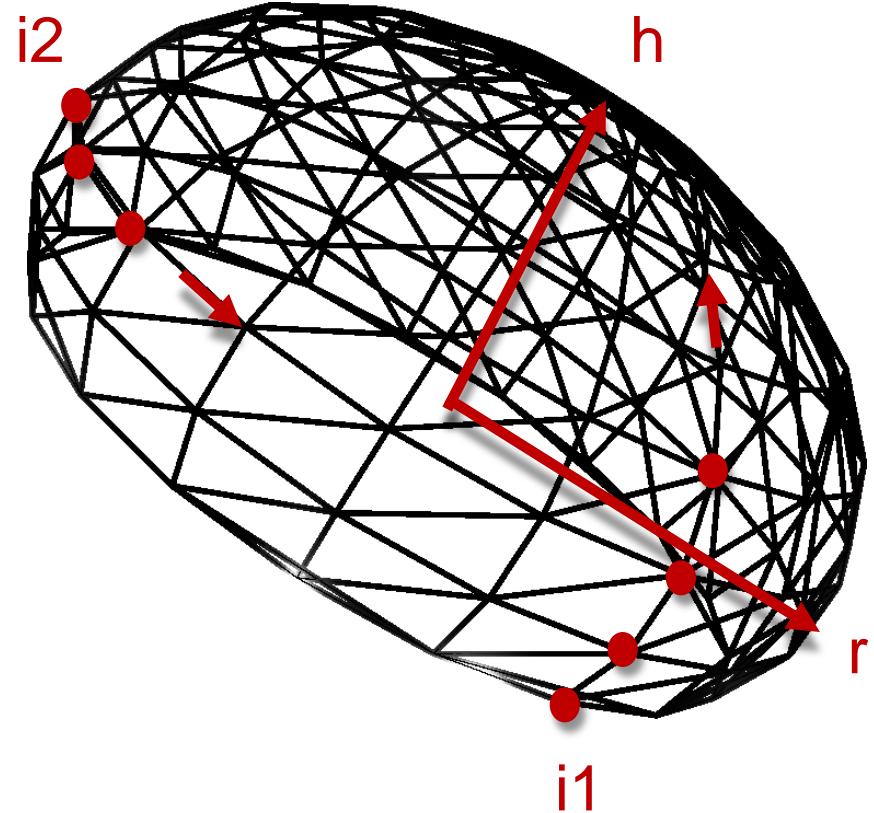
5 // / / / int iLatitude =12);

Anzahl der Vertices
um den „Äquator“



Parametrisierbare geometrische Primitive

CGeoDome



- 1 // / / /
- 2 // / / /
- 3 // / / / *h* = fHeight
 r = fRadius
 M = pmaterial
- 4 // / / / *i1* = iLongitude
 i2 = iLatitude
- 5 // / / /



CGeoDome::Init

Initialisierungsmethode von **CGeoDome**:

void Init(

float fRadius, float fHeight,

CMaterial * pmaterial,

bool bSkyDome = false,

int iLongitude= 24,

int iLatitude = 12,

int eMapping

= S_GEOELLIPSOIDMAPPING_CYLINDRICAL);

Radius und Höhe der Kuppel

Zeiger auf Material, welches auf
die Kuppel gemappt wird.

true, wenn für Skydome (dann zeigen
Normalenvektoren nach innen)

Anzahl der Vertices von "Pol zu Pol"

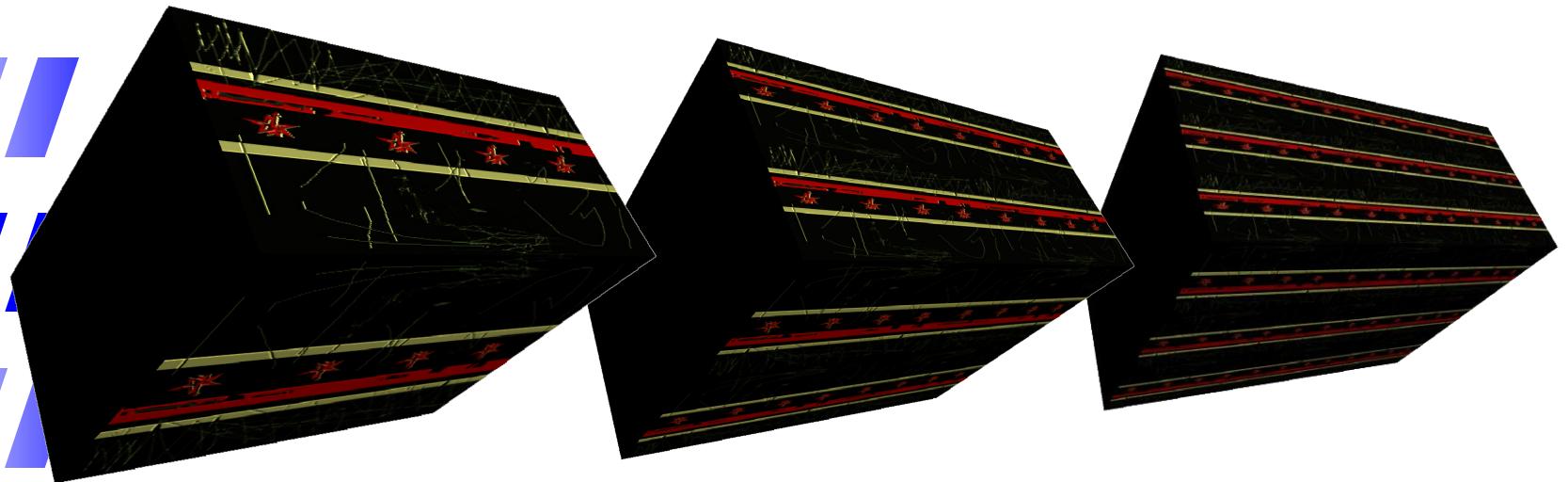
Anzahl der Vertices um den „Äquator“

Art des UV-Mappings



Parametrisierbare geometrische Primitive

CGeoCube



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

CGeoCube
mit
iTTextureRepeat = 1
(Default)

CGeoCube
mit
iTTextureRepeat = 2

CGeoCube
mit
iTTextureRepeat = 3



Initialisierungsmethode von **CGeoCube**:

1 // / / / void Init

(

2 // / / / CHVector vSize,

Kantenlänge des Quaders

3 // / / / CMaterial * pmaterial,

Zeiger auf Material, welches auf
den Quader gemappt wird.

4 // / / / int iTextureRepeat = 1,

Anzahl der Texturwieder-
holungen pro Kante

5 // / / / bool bFlip = false

Optionales Flag zum Umdrehen der
Normalenvektoren

);



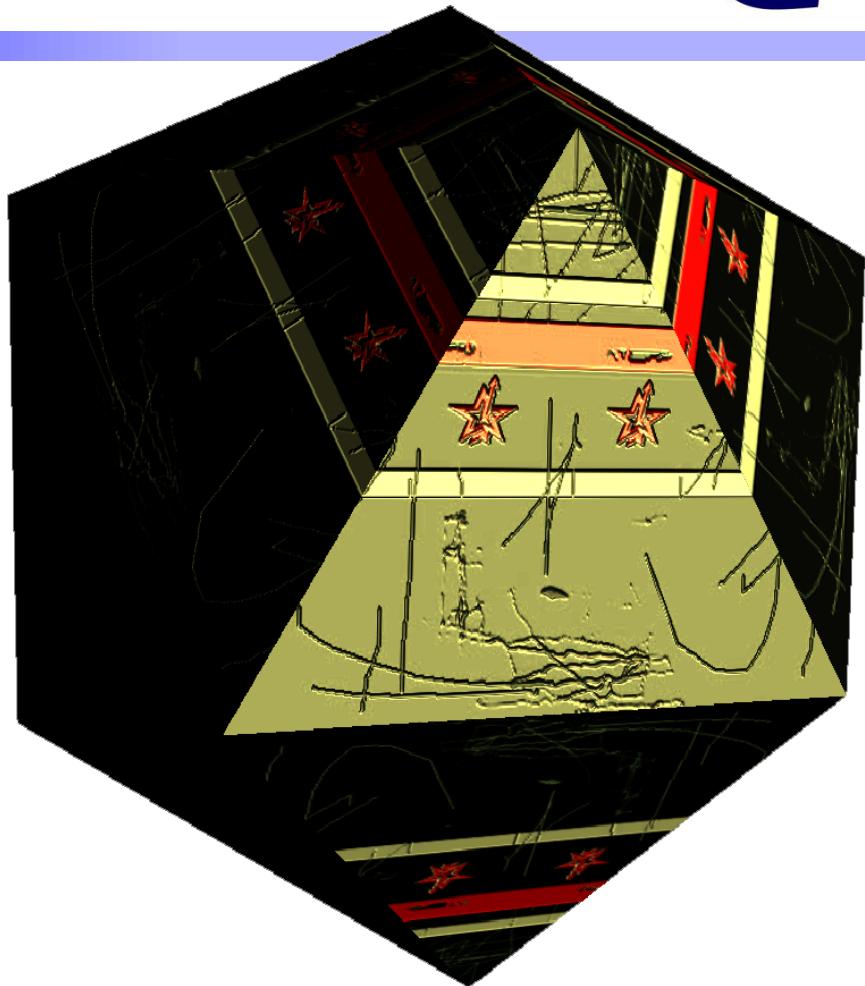


Parametrisierbare geometrische Primitive

CGeolkosaeder



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



CGeolkosaeder::Init



Initialisierungsmethode von **CGeolkosaeder**:

1 // / / / void Init

(

2 // / / / CHVector vSize,

Größe des Ikosaeders in X, Y und Z-Richtung (damit sind auch „verzerrte“ Ikosaeder möglich)

3 // / / / CMaterial * pmaterial,

Zeiger auf Material, welches auf den Würfel gemappt wird.

4 // / / / bool bFlip = false

Optionales Flag zum Umdrehen der Normalenvektoren

5 // / / /);



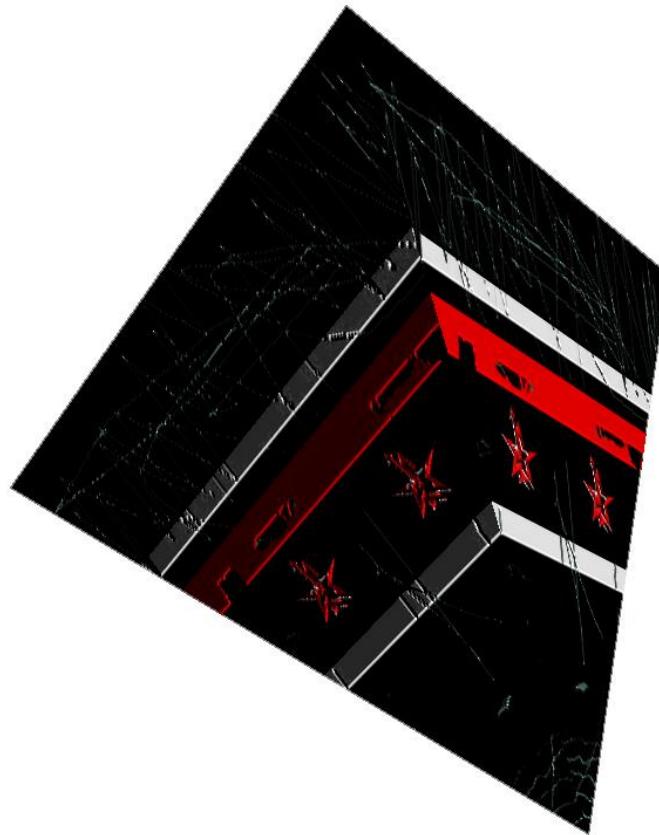


Parametrisierbare geometrische Primitive

CGeoTetraeder



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





Parametrisierbare geometrische Primitive

CGeoTetraeder::Init



Initialisierungsmethode von **CGeoTetraeder**:

```
1 //////////////////////////////////////////////////////////////////  
2 void Init  
(  
    CHVector vSize,  
  
    CMaterial * pmaterial,  
  
    bool bFlip=false  
);  
3 //////////////////////////////////////////////////////////////////  
4 //////////////////////////////////////////////////////////////////  
5 //////////////////////////////////////////////////////////////////
```

Größe in X,Y,Z-Richtung, bei unterschiedlichen Werten wird der Tetraeder „verzerrt“.

Zeiger auf Material, welches auf das Ellipsoid gemappt wird.

Optionales Flag zum Umdrehen der Normalenvektoren.



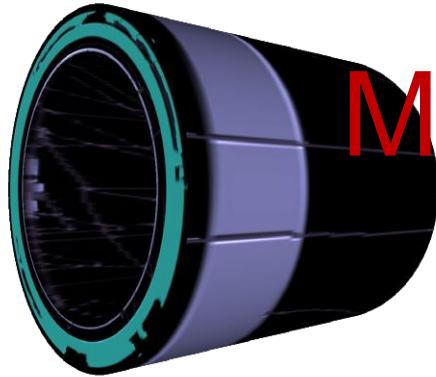


Parametrisierbare geometrische Primitive

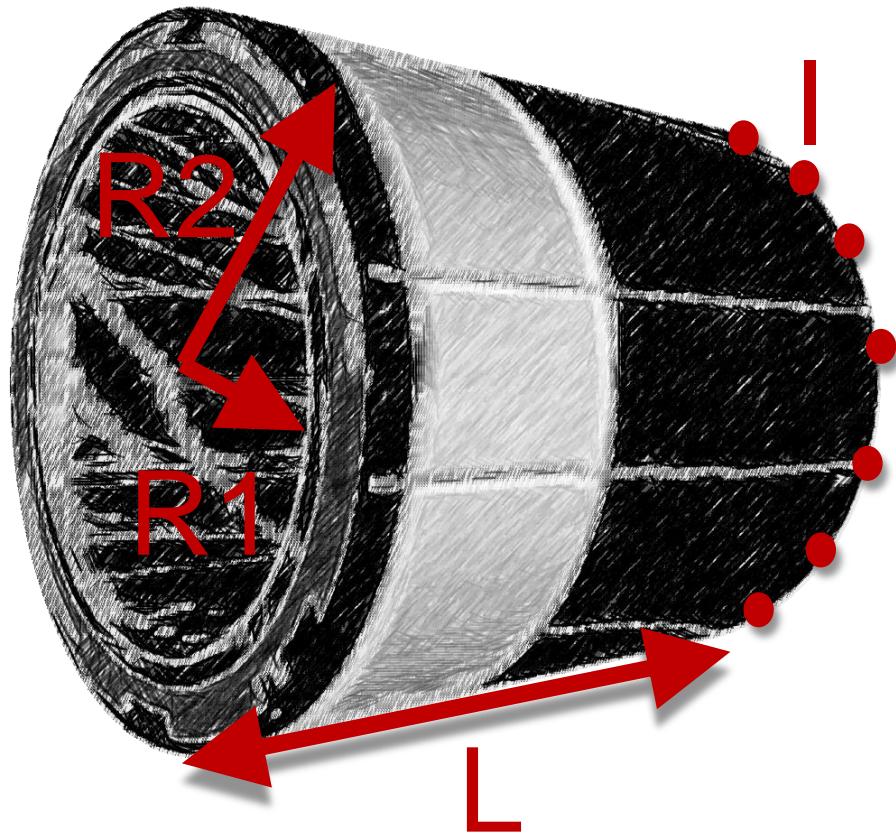
CGeoTube (gerade)



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



R1 = fRadiusInner
R2 = fRadiusOuter
L = fLength
M = pmaterial
I = iLongitude



CGeoTube::InitStraight

Initialisiert einen geraden Röhrenabschnitt:

void InitStraight(
 float fRadiusInner,

Innenradius der Röhre, wirkt sich
nur aus, wenn bInner auf true
gesetzt wurde.

 float fRadiusOuter,

Außenradius der Röhre, sollte
natürlich \geq fRadius Inner sein.

 float fLength,

Länge der Röhre.

 CMaterial * pmaterial,

 Materialpointer

 int iLongitude = 24,

Anzahl der Vertices (radial), je
höher dieser Wert, desto mehr
Polygone hat die Geometrie

 bool bInner = true);

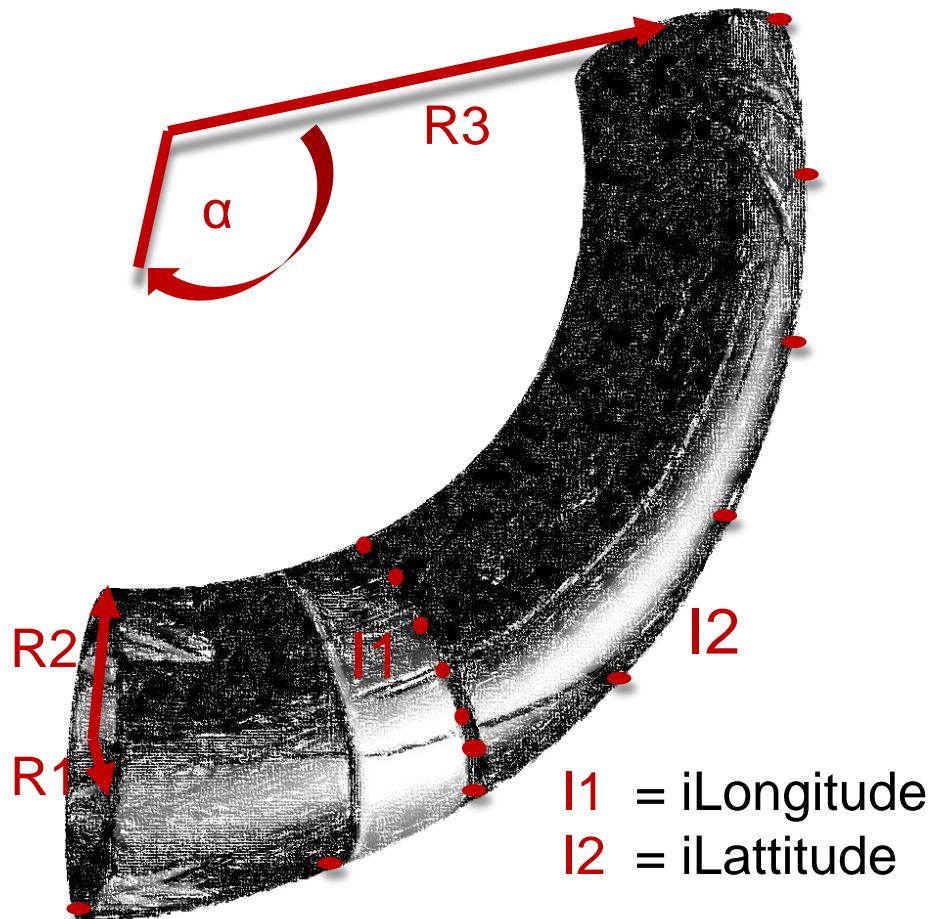
Hat Innenleben, wenn true

Parametrisierbare geometrische Primitive

CGeoTube (gebogen)



- 1 // / / /
 - 2 // / / /
 - 3 // / / /
 - 4 // / / /
 - 5 // / / /
- R1 = fRadiusInner
R2 = fRadiusOuter
R3 = fRadiusArc
 α = fa
M = pmaterial



CGeoTube::InitArc



void InitArc(float fRadiusInner, float fRadiusOuter,
float fRadiusArc, float faArc, CMaterial * pmaterial,
int iLongitude = 24, int iLatitude = 24, bool bInner = true);

1 // / / / InitArc initialisiert einen gebogenen Röhrenabschnitt.

2 // / / / **fRadiusInner:** Innenradius der Röhre, wirkt sich nur aus, wenn **bInner** auf **true** gesetzt wurde, **fRadiusOuter:** Außenradius der Röhre, sollte natürlich \geq fRadius Inner sein. **fRadiusArc:** Radius des Röhrenbogens, **faArc:** Winkel des Röhrenbogens im Bogenmaß, **pmaterial:** Pointer auf das Material, welches auf die Geometrie gelegt wird; **iLongitude:** iLongitude = Anzahl der Vertexunterteilungen um das Rohr; **iLatitude:** Anzahl der Vertexunterteilungen entlang des Rohres, **bInner:** gibt an, ob das innenleben erstellt werden soll.

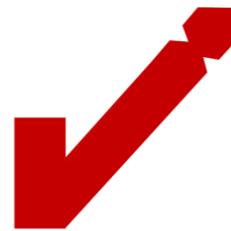
3 // / / /

4 // / / /

5 // / / /

6 // / / /





Parametrisierbare geometrische Primitive

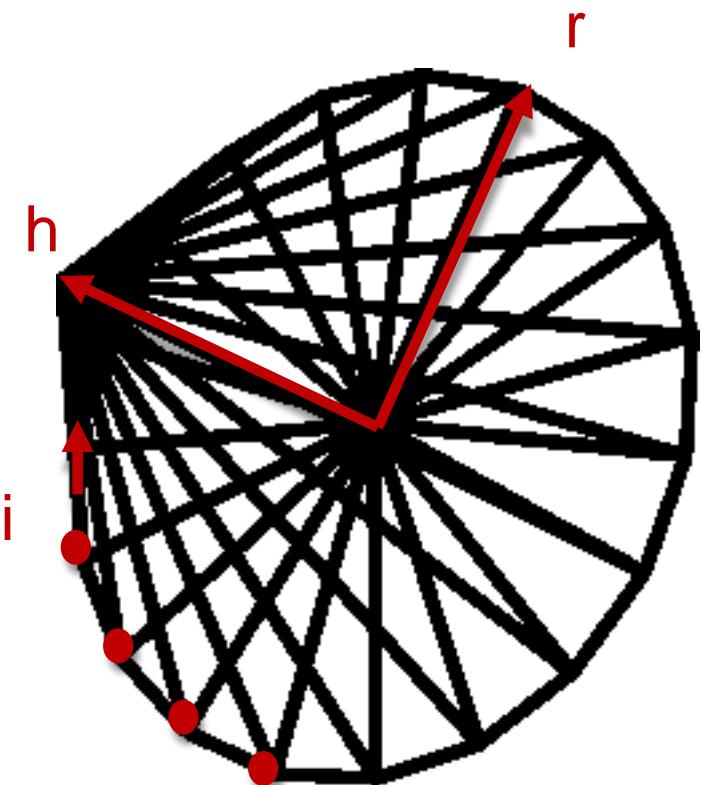
CGeoCone



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



r = fRadius
h = height
M = pmaterial
i = iLongitude



CGeoCone::Init

Initialisiert einen Kegel:

1 // / / /

void Init(

 float fRadius,

 Radius an der Basis des Kegels.

2 // / / /

 float fHeight,

 Höhe des Kegels

3 // / / /

 CMaterial * pmaterial,

 Materialpointer

4 // / / /

 int iLongitude = 24,

 Anzahl der Vertices (um den Kegel
herum), je höher dieser Wert, desto
mehr Polygone hat die Geometrie

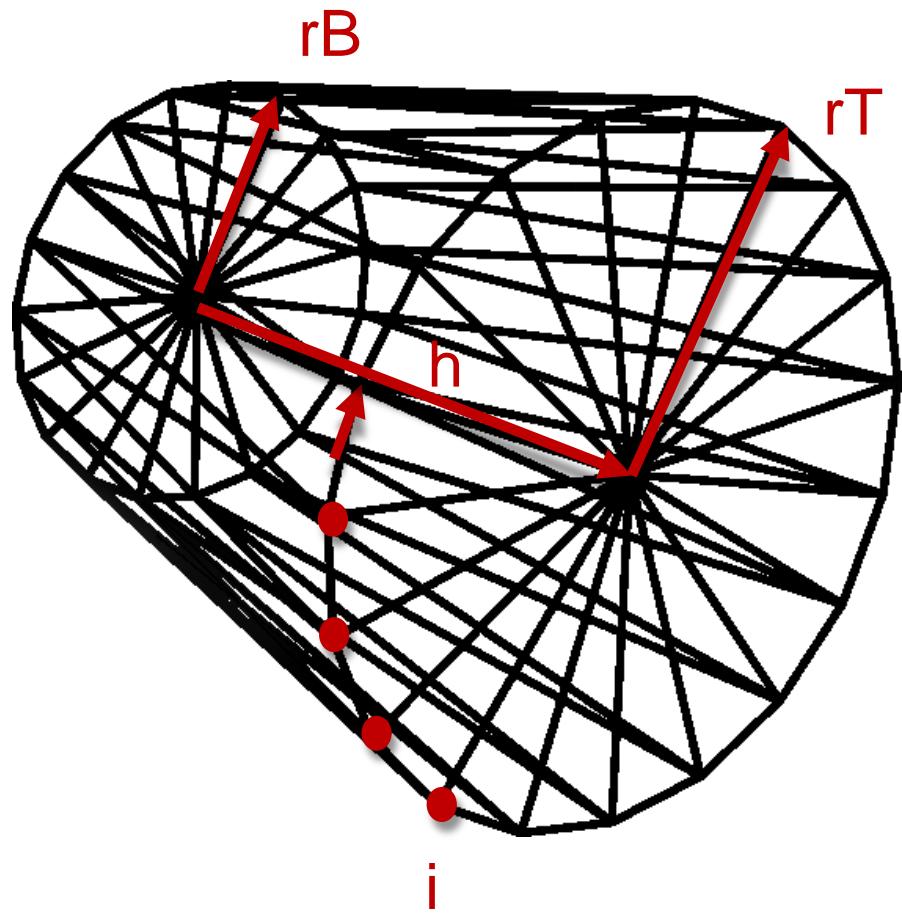
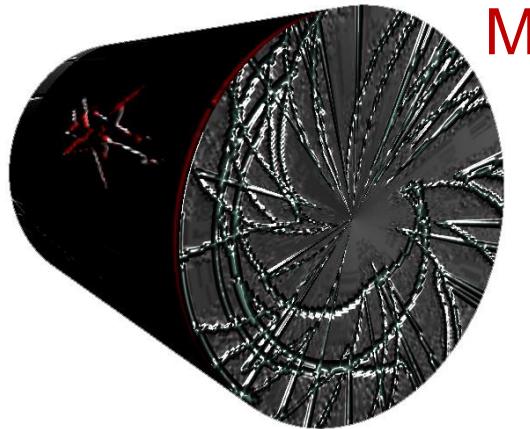
5 // / / /

 bool bHasBottom = true);

 Hat Basisfläche, wenn true



CGeoCylinder::Init



rB = fRadiusBottom
 rT = fRadiusTop
 h = height
 M = pmaterial
 i = iLongitude



CGeoCylinder::Init

Initialisiert einen Zylinder oder Zylinderschnitt:

void Init(

float fRadiusBottom,

Radius an der Zylinderbasis

float fRadiusTop,

Radius an der Zylinderspitze

float fHeight,

Höhe des Kegels

CMaterial * pmaterial,

Materialpointer

int iLongitude = 24,

Anzahl der radialen Vertices

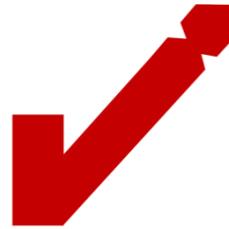
bool bHasBottom = true,

Hat Basisfläche, wenn true

bool bHasTop = true);

Hat Dachfläche, wenn true

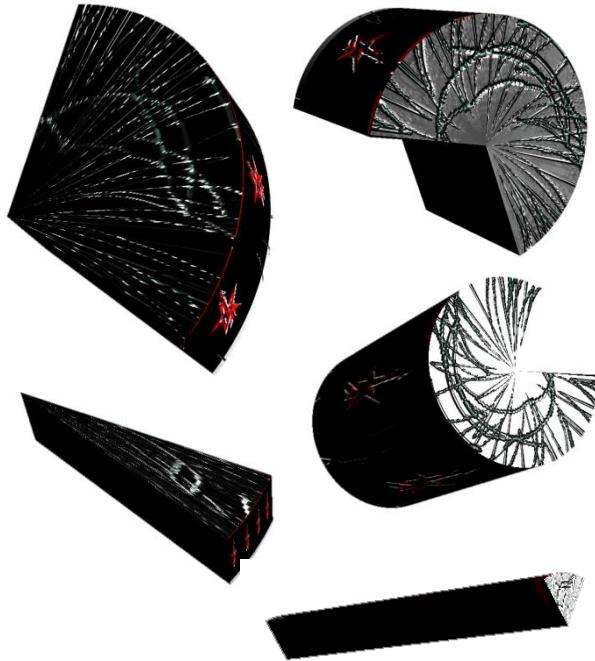




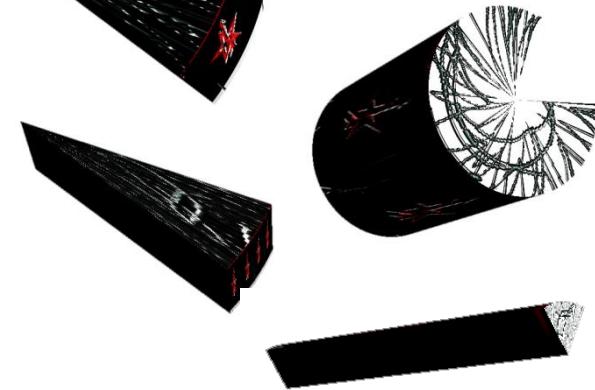
Parametrisierbare geometrische Primitive
CGeoSlice Vergleich



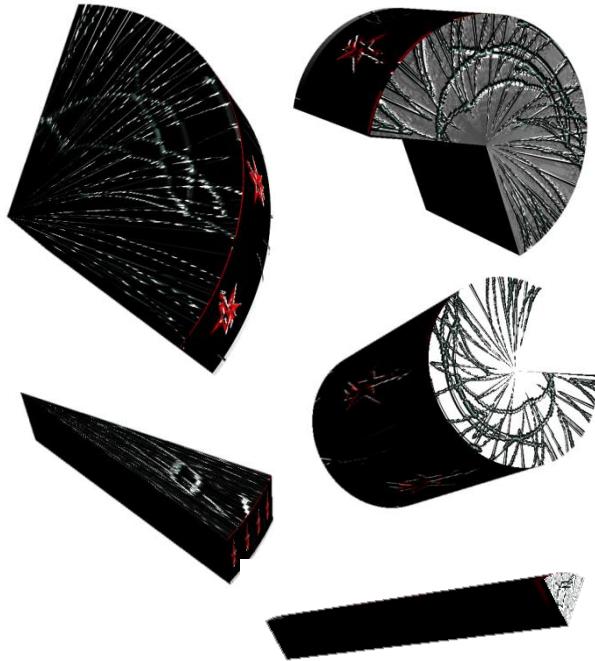
1 // / / /



2 // / / /



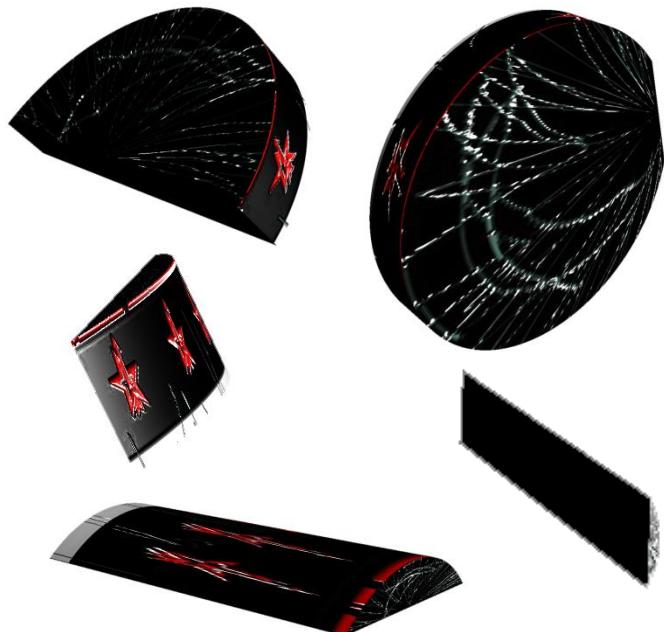
3 // / / /



4 // / / /

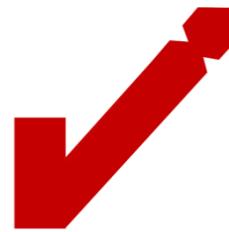
Verschiedene GeoSlice-
Körper mit
bPlanarBottom = false

5 // / / /



Verschiedene GeoSlice-
Körper mit
bPlanarBottom = true
(Default)





CGeoSlice::Init

Initialisiert einen tortenförmigen Körper

```
void Init(  
    float fLength,  
    float fRadius,  
    float faArcStart,  
    float faArcStop,  
    CMaterial * pmaterial,  
    int iLongitude = 24,  
    bool bHasFront = true,  
    bool bHasBack = true,  
    bool blsPlanarBottom = true);
```

Höhe des Tortenstücks
Radius der Torte
Start- und Stopwinkel im Bogenmaß
Materialpointer
Anzahl der radialen Vertices
Hat Front- bzw. Basisfläche, wenn true
Siehe vorherige Seite





Kapitel 3

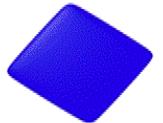
Kapitel 3



1 // / / /

2 // / / /

/// SWEEPING & EXTRUDING ///



4 // / / /

5 // / / /



Sweeping & Extruding



Sweeping:

Beim Sweeping führt man eine 2D-Fläche anhand eines Pfades durch den Raum. Das Volumen, welches die Fläche dabei durchstreich, wird als Sweep-Körper bezeichnet. In polygonalen Flächenmodellen werden die Ränder des Sweep-Körpers trianguliert.

Extruding:

Beim Extruding führt man eine 2D-Fläche entlang einer Gerade durch den Raum. Das Volumen, welches die Fläche dabei durchstreich, wird als extrudierter Körper bezeichnet. In polygonalen Flächenmodellen werden die Ränder des extrudierten Körpers trianguliert.



Vergleich Sweeping & Extruding

1 // / / / Vektoria-Sweep-Körper

2 // / / / Sweep-Körper

3 // / / / Extrudierte Körper

Die Menge der extrudierten Körper ist eine Untermenge der Sweep-Körper, denn eine Extrudierungsgerade kann als spezieller Sweepfaden angesehen werden:

Extrudierte Körper sind somit Sonderformen der Sweep Körper.

Vektoria lässt zusätzlich Skalierungen der Fläche, als auch Shading-Modifikationen beim Durchlaufen des Pfades zu:

Sweep-Körper sind somit Sonderformen der Vektoria-Sweep Körper.





Sweeping & Extruding

Sweep- & Extruding-Klasse



Die Klasse **CGeoSweep** generiert beliebige Vektoria-Sweep-, Sweep- und Extruding-Körper

Sie ist übrigens die Basisklasse für:

- CGeoCone,
- CGeoCylinder,
- CGeoSlice und
- CGeoTube.





Sweeping & Extruding



CGeoSweep::Init

void Init(CHVector vSize, CMaterial * pmaterial,
CHMats & msPath, CHVectors & vs);



vSize:

Allgemeine Skalierung des Sweep-Körpers
in x,y- und z-Richtung



pmaterial:

Material, welches auf den Sweep-Körper
aufgebracht wird



msPath:

gibt den Sweep-Pfad in Form einer Liste
homogener Matrizen an
Punkte des Querschnitts



vs:





Sweeping & Extruding



Tipps und Schliche



Sind harte anstatt weich geshadeter Kanten erwünscht, so lässt sich dies durch einmaliges Wiederholen des Flächenpunktes bzw. der Pfadmatrix erreichen.



Im Gegensatz zu reinen Sweep-Körpern lassen Vektoria-Sweep-Körper durch Angabe von skalierten Matrizen auch Verdickungen und Verdünnungen entlang des Pfades zu.



Für die Extrudierung werden einfach zwei zueinander translierte Matrizen verwendet. Soll der extrudierte Körper oben und unten geschlossen sein, so werden noch an den beiden Stellen (transliert und nichttransliert) zwei weitere nullskalierte Matrizen hinzugefügt.



Kapitel 4

Kapitel 4



1 // / / /

2 // / / /

3 // / / /

UEBUNGEN



5 // / / /



PROF. DR. TOBIAS BREINER
VEKTORIA MANUAL

43 VON 61
VORGEFERTIGTE GEOMETRIEN





Übungen

Hallo Würfel!-Übung



- Ersetzen Sie die Kugel im „Hallo Kugel!“-Programm durch einen Würfel!

1 // / /

2 // / /

3 // / /

4 // / /

5 // / /

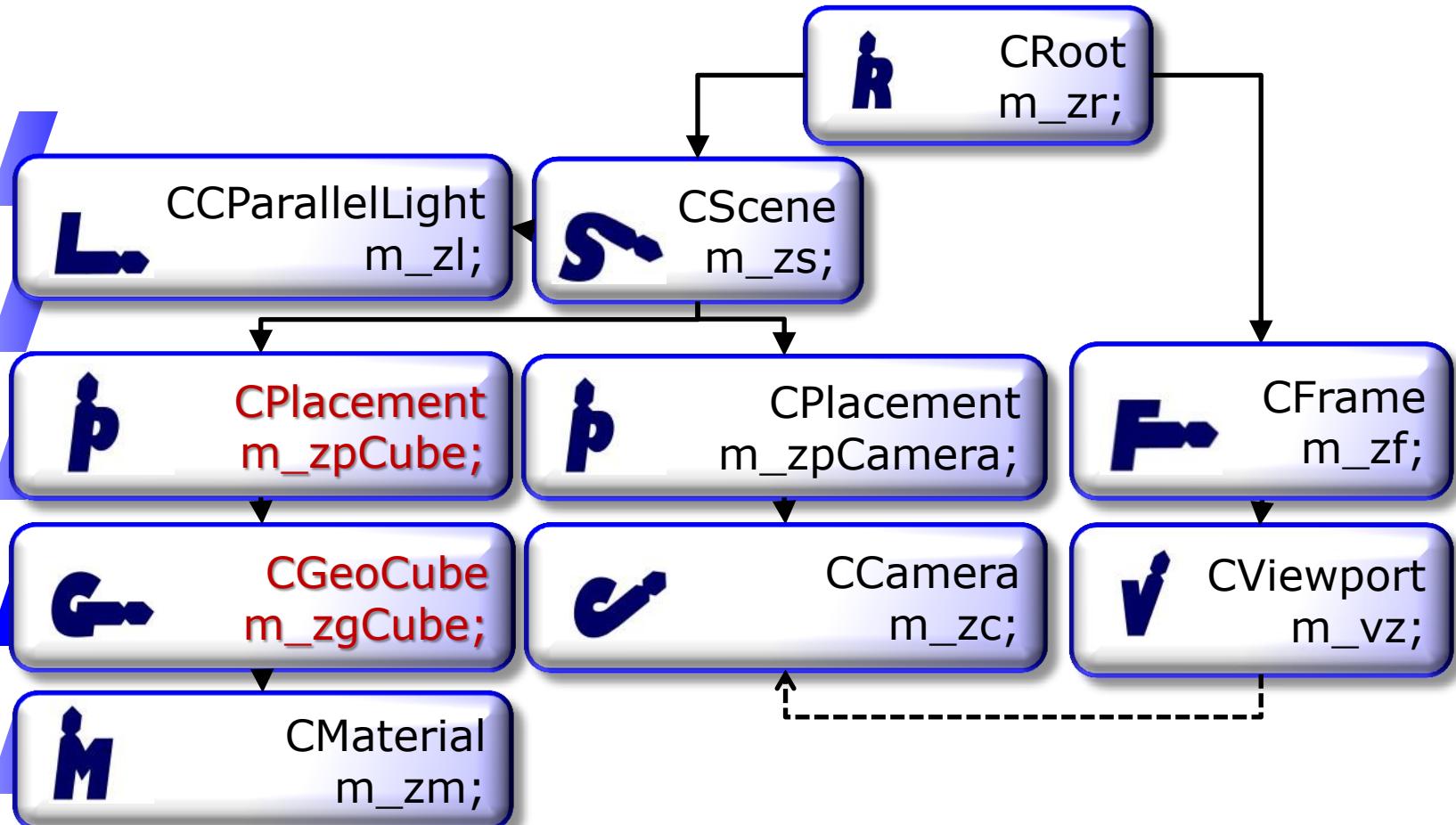
Freiwillige Zusatzaufgaben für die schnellen Nerds:

- Ersetzen Sie die Kugel durch einen Tetraeder!
- Ersetzen Sie die Kugel durch einen Ikosaeder!



Hallo Würfel!-Lösung

Objekthierarchie „Hallo Würfel!!“





Übung Dreikörper



1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /

- Erweitern Sie die „Hallo Würfel!“-Anwendung dahingehend, dass ein Würfel, eine Kugel und ein Tetraeder gleichzeitig nebeneinander sichtbar sind!

Freiwillige Zusatzaufgabe für die schnellen Nerds:

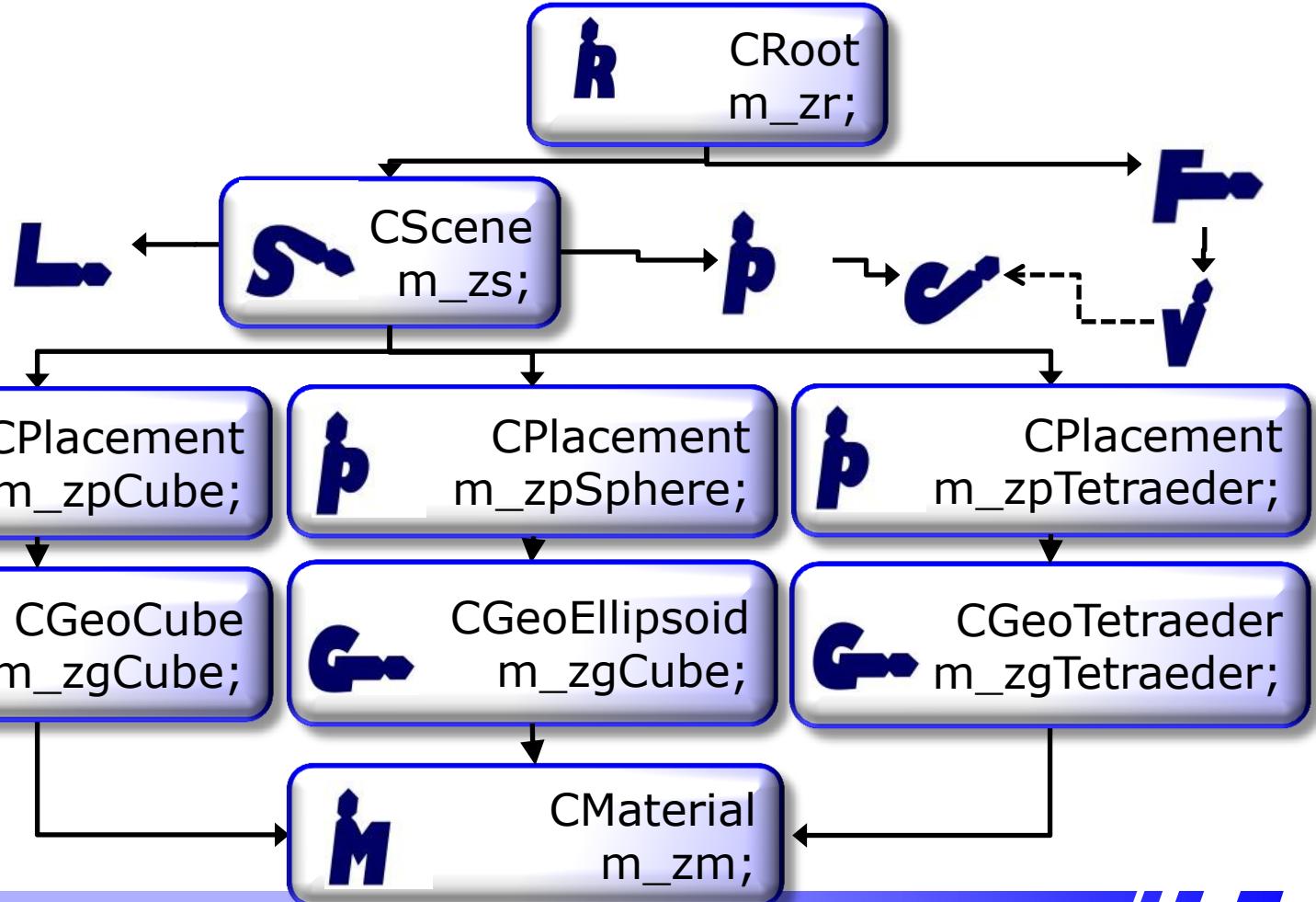
- Lassen Sie die drei Körper in einem Riesen-Ikosaeder stehen!





Objekthierarchie

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





Lösung zu Übung Dreikörper (2/7)

Vektoria-Objekte in CGame.h



```
CRoot m_zr;
CScene m_zs;
CPlacement m_zpCamera;
CPlacement m_zpSphere;
CPlacement m_zpCube;
CPlacement m_zpTetraeder;
CGeoEllipsoid m_zgSphere;
CGeoCube m_zgCube;
CGeoTetraeder m_zgTetraeder;
CFrame m_zf;
CViewport m_zv;
CCamera m_zc;
CParallelLight m_zl;
CMaterial m_zm;
```

1 // / / /

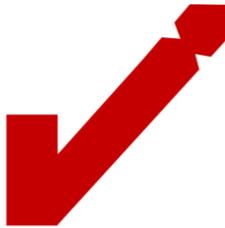
2 // / / /

3 // / / /

4 // / / /

5 // / / /





Lösung zu Übung Dreikörper (3/7)

Init-Methode in CGame.cpp



```
void CGame::Init(HWND hwnd, HWND hwndDX,  
                  RECT rectWnd)  
{  
    m_zc.Init(1.2F);  
    m_zf.Init(hwnd, rectWnd.right, rectWnd.bottom);  
    m_zv.InitFull(&m_zc);  
    m_zr.AddFrameHere(&m_zf);  
    m_zf.Addviewport(&m_zv);  
    m_zm.MakeTextureDiffuse  
        ("textures\\white_image.jpg");  
    m_zl.Init(CHVector(1,0,1),CColor(1,1,0.5));  
    m_zr.AddScene(&m_zs);  
    m_zs.AddPlacement(&m_zpCamera);  
    m_zs.AddParallelLight(&m_zl);
```

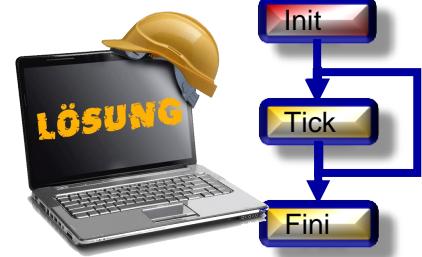
- 1 // / / /
- 2 // / / /
- 3 // / / /
- 4 // / / /
- 5 // / / /





Lösung zu Übung Dreikörper (4/7)

Init-Methode in CGame.cpp



```
m_zs.AddPlacement(&m_zpSphere);  
m_zs.AddPlacement(&m_zpCube);  
m_zs.AddPlacement(&m_zpTetraeder);  
m_zpCamera.AddCamera(&m_zc);  
m_zpSphere.Translate(CHVector(0,0,-5));  
m_zpSphere.AddGeo(&m_zgSphere);  
m_zpCube.Translate(CHVector(-3.9,0,-5));  
m_zpCube.AddGeo(&m_zgCube);  
m_zpTetraeder.Translate(CHVector(3.9,0,-5));  
m_zpTetraeder.AddGeo(&m_zgTetraeder);  
m_zgSphere.Init(CHVector(2.0F,2.0F,2.0F),&m_zm);  
m_zgCube.Init(CHVector(1.0F,1.0F,1.0F),&m_zm);  
m_zgTetraeder.Init(CHVector(2.0F,2.0F,2.0F),  
    &m_zm);  
}
```

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /





Lösung zu Übung Dreikörper (5/7)

Tick-Methode in CGame.cpp



```
void CGame::Tick(float fTimeDelta)
{
    m_zr.Tick(fTimeDelta);
}
```

1 // / / /

2 // / / /

3 // / / /

4 // / / /

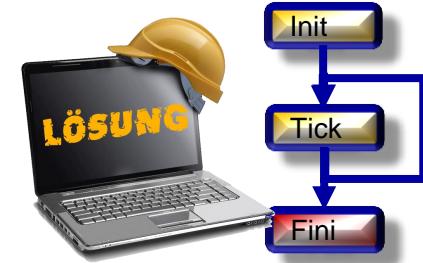
5 // / / /





Lösung zu Übung Dreikörper (6/7)

Fini-Methode in CGame.cpp



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

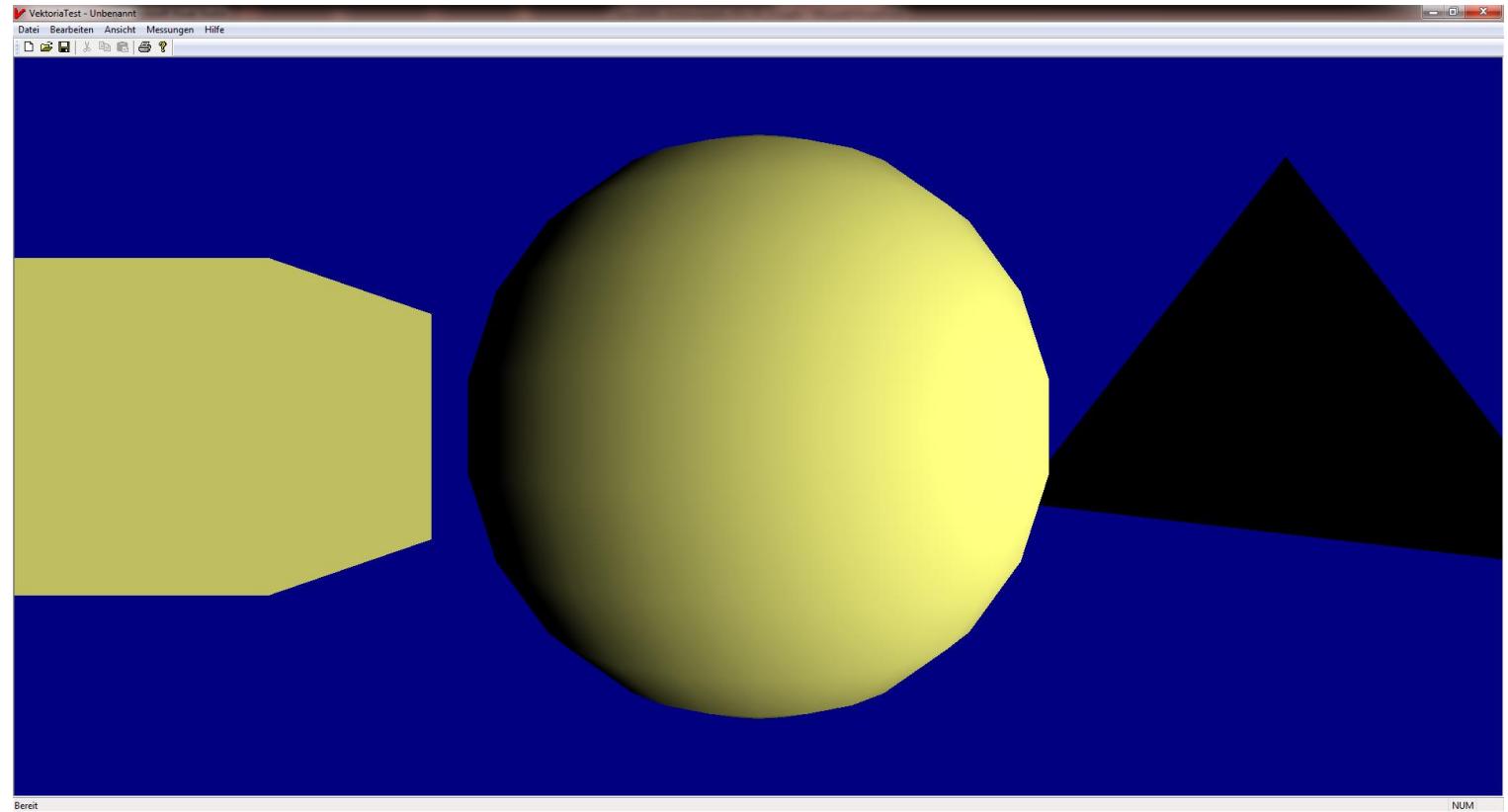
Wir haben ja weder dynamischen Speicherplatz allokiert, der wieder freigegeben werden müsste, noch müssen wir einen Gewinner ausgeben





Lösung zu Übung Dreikörper (8/7)

Ausgabe





Übung Texturierung



- 1 // / / /
- Mappen Sie auf Box und Tetraeder jeweils eine andere Textur!

2 // / / /

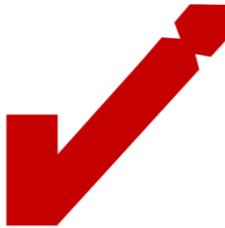
Freiwillige Zusatzaufgabe für die schnellen Nerds:

- 3 // / / /
- Wiederholen Sie die Textur auf jeder Seite des Würfels drei * drei = neun Mal!

4 // / / /

5 // / / /





Übung Quad



- Stellen Sie eine Hintergrundfläche hinter die Körper, mappen Sie eine Sternentextur drauf!

Freiwillige Zusatzaufgabe für die schnellen Nerds:

- Drehen Sie den Quad um die Sichtachse!



Übung Mondbasis



- Erstellen Sie eine Mondbasis aus Röhren, Silos, Sweep-Körper, Walls und Windows, etc.

Freiwillige Zusatzaufgabe für die schnellen Nerds:

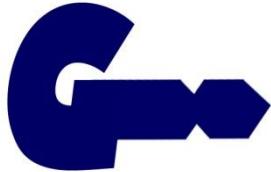
- Lassen Sie einzelne Teile der Mondbasis aufblinken!





Kapitel 5

Kapitel 5



1 // / / /

2 // / / /

3 // / / /

4 // / / /



/// ELLIPSOID-/// MAPPING



Ellipsoid-Mapping-Klassen

1 // / / / Jede von CGeoEllipsoid abgeleitete Klasse,
also:

- CGeoEllipsoid
- CGeoSphere
- CGeoDome

2 // / / / hat ein eMapping-Parameter.

3 // / / / Damit lässt sich die Art und Weise modifizieren, wie eine
Textur aufgebracht wird:

4 // / / /

5 // / / /

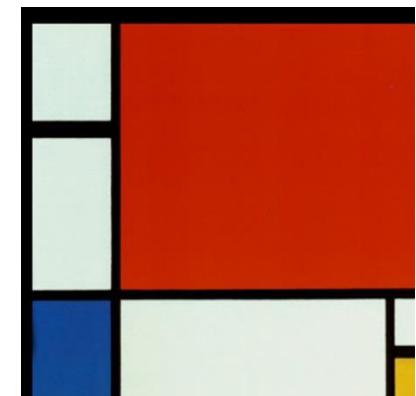


Ellipsoid-Mapping-Arten

Es existieren zurzeit folgende Mapping-Arten

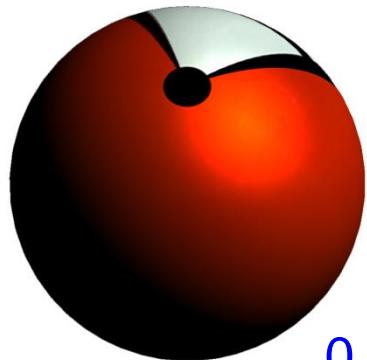
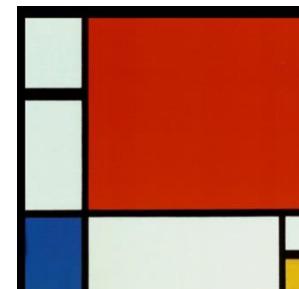
- 1 // / / / 0: S_GEOELLIPSOIDMAPPING_CYLINDRICAL (default)
- 2 // / / / 1: S_GEOELLIPSOIDMAPPING_QUADROCYLINDRICAL
- 3 // / / / 2: S_GEOELLIPSOIDMAPPING_BICYLINDRICAL
- 3 // / / / 3: S_GEOELLIPSOIDMAPPING_QUADROBICYLINDRICAL
- 4 // / / / 4: S_GEOELLIPSOIDMAPPING_ORTHOGRAPHIC
- 5 // / / / 5: S_GEOELLIPSOIDMAPPING_BIORTHOGRAPHIC

Im Folgenden sollen die Mapping-Arten
anhand einer Textur des Künstlers
Piet Mondrian veranschaulicht werden

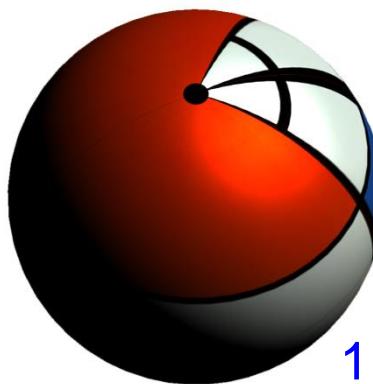


Ellipsoid-Mapping

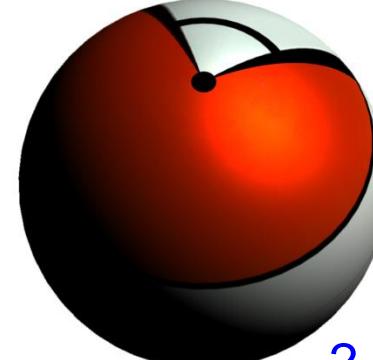
Ellipsoid-Mapping-Arten



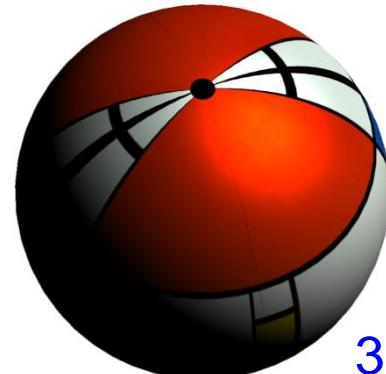
0
cylindrical



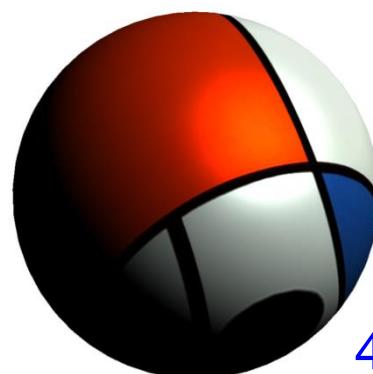
1
quadrocylindrical



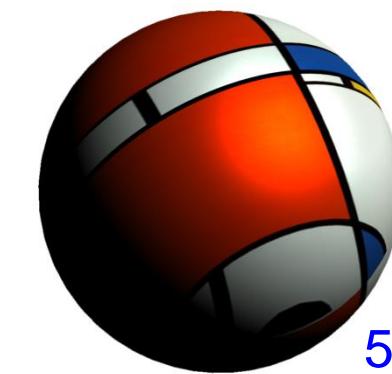
2
bicylindrical



3
quadrobicylindrical



4
orthographic



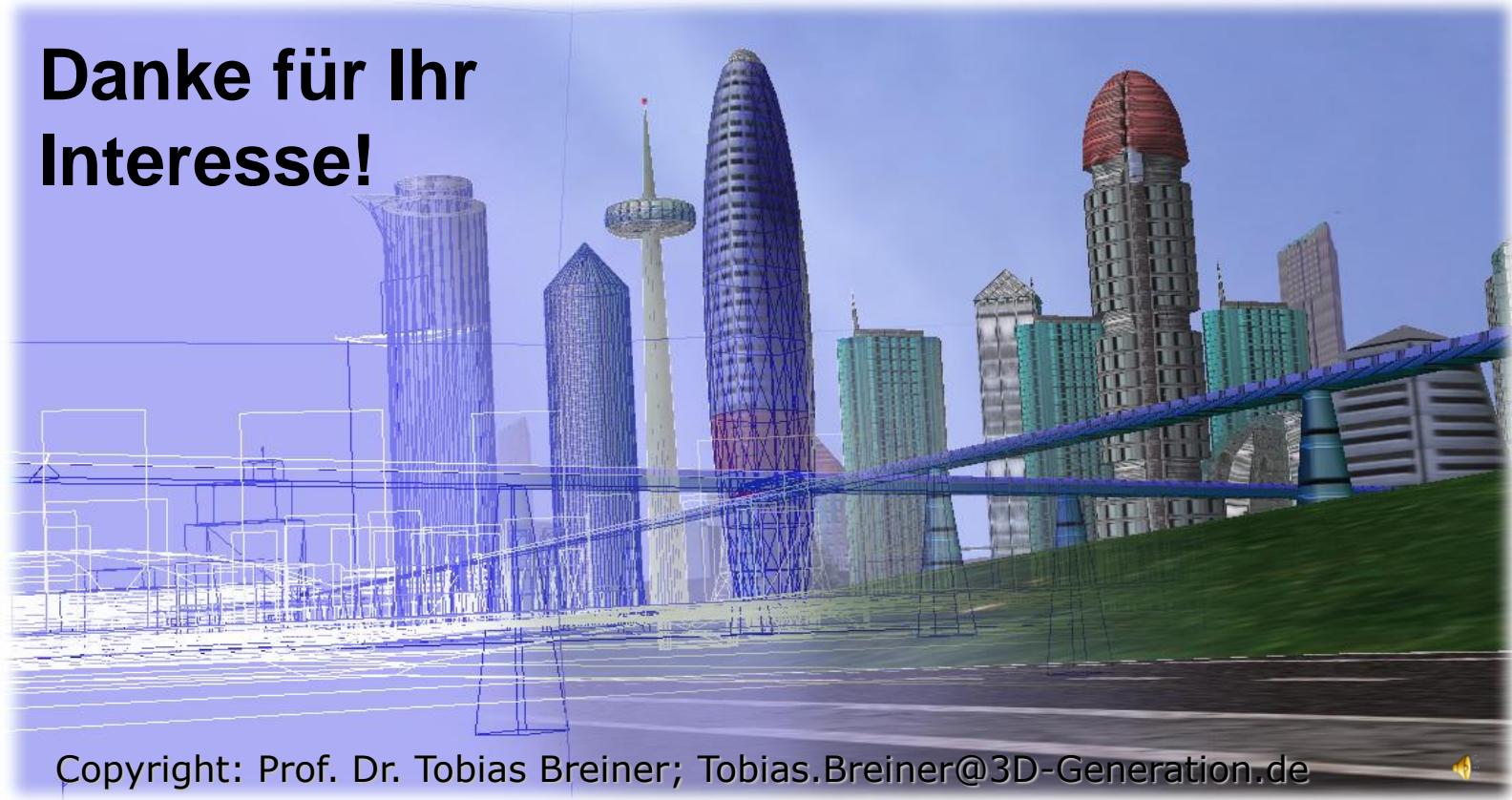
5
biorthographic

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



|||||GAME ||||OVER

Danke für Ihr
Interesse!



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



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

61 VON 61
||||VORGEFERTIGTE ||||GEOMETRIEN