VEKTORIA-MANUAL
BARR-MODELLIERUNG

Prof. Dr. Tobias Breiner

vektoria

www.vektoria-engine.com

Inhalt

Einfache Modellierungsmethoden

- Subdivition, Twisting, Tapering und Bending sind altbekante einfache Modellierungsmethoden aus den achziger Jahren.
- Sie sind zur schnellen Erstellung einfacher technischer Geometrien wie Gewinde, Prismen, Bögen, Eierformen, etc. gut geeignet.
- Trotzdem werden sie von fast keiner Game-Engine bzw. keinem Szenegrafen angeboten.

✔ Vektoria bietet als einziger Szenegraf neben den Barr-Methoden (Twisting, Tapering und Bending) auch neue Methoden wie Waving und Rippleing an. [Breiner 07]

✔ Zusätzlich kann man in Vektoria auch mit „Magneten“ arbeiten, welche Vertexgruppen kontrolliert anziehen, z.B. um Spikes oder Beulen zu erzeugen. [Breiner 07]

Barr-Methoden (Nichtlineare Globale Deformation)

Skalierung → Tapering

Rotation → Twisting

Transformation & Rotation → Bending

SUBDIVITION

2

3

4

5

Subdivide

Mit der Methode Subdivide der Klasse CTriangleList lassen sich die Polygone einer existierenden Geometrie sinnfällig unterteilen. Da alle aus einem alten Polygon neu entstanden Polygone danach in der Ursprungsebene des alten Polygons liegen, werden Anwendende danach erst einmal nichts bemerken, es sei denn sie befinden sich im Wireframe-Modus. Subdivition ist allerdings die Voraussetzung, dass andere Modellierungsmethoden wie Tapering, Twisting, Bending, Rippling und Magnet richtig funktionen.

```
void Subdivide(float fLengthMax);
```

Wandelt das Mesh in ein Subdivision-Mesh mit mehr Polygonen um, abhängig von der maximalen Kantenlänge fLengthMax

SubdivideX, -Y und -Z

```
void SubdivideX(float fLengthMax);
void SubdivideY(float fLengthMax);
void SubdivideZ(float fLengthMax);
```

Subdivition funktioniert auch in die Richtung einer einzigen Koordinatenachse.
Die maximale Kantenlänge fLengthMax bezieht sich dann nur auf die entsprechende Richtung.

ExtractSubdivition

```
CTriangleList * ExtractSubdivition(float fLengthMax);
CTriangleList * ExtractSubdivitionX(float fLengthMax);
CTriangleList * ExtractSubdivitionY(float fLengthMax);
CTriangleList * ExtractSubdivitionZ(float fLengthMax);
```

1
2
3
4
5

Es gibt zu allen einfachen Subdivide-Methoden auch entsprechende Extraktionspendants.
Diese generieren ein neues Subdivision-Mesh mit mehr Polygonen aus dem aktuellen Mesh, abhängig von der maximalen Kantenlänge fLengthMax. Das ursprüngliche Mesh bleibt unverändert. Das ist beispielsweise bei der Erzeugung mehrerer LODs von Vorteil.

CopyToTriangleList

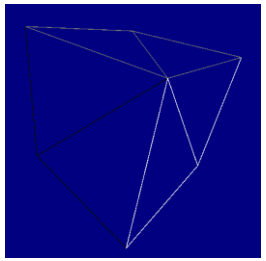
Subdivition arbeitet nicht mit TriangleStrips, sondern nur mit TriangleLists!

Wenn Sie trotzdem TriangleStrips verwenden wollen, müssen Sie erst mit der Methode CopyToTriangleList der Klasse CTriangleStrip aus dem Dreiecksstreifen eine entprechende Dreiecksliste erstellen.

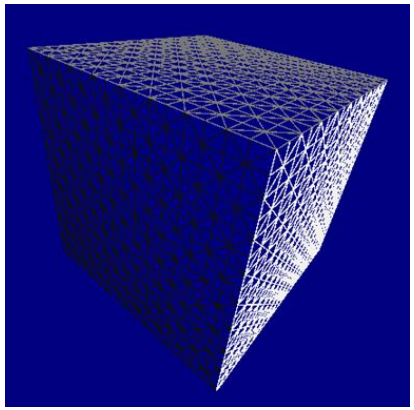
Aufgabe zu Subdivide

Übung „Subdivide Cube!“

1. Erzeugen Sie eine Wireframe-Szene mit einem Cube!

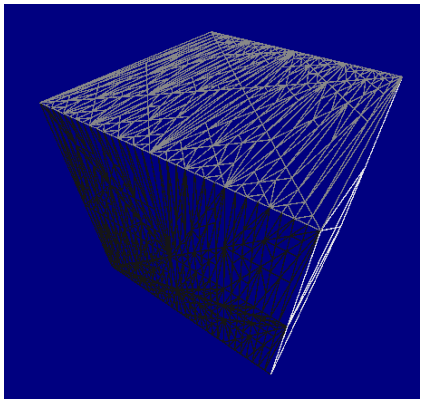


2. Wenden Sie Subdivide auf den Würfel an!



3. Freiwillige Übung für die schnellen Nerds:

4. Versuchen Sie folgende Würfelpolygonunterteilung zu erzeugen:



5.

Lösung zur Subdivide Cube-Übung (1/6)

Vektoria-Objekte in CGame.h

```
CRoot m_zr;
CScene m_zs;
CPlacement m_zpCamera;
CPlacement m_zp;
CGeoCube m_zgCube;
CHardware m_zh;
CFrame m_zf;
CViewport m_zv;
CCamera m_zc;
CParallelLight m_zl;
CMaterial m_zm;
```

Init-Methode in CGame.cpp

1
2
3
4
5

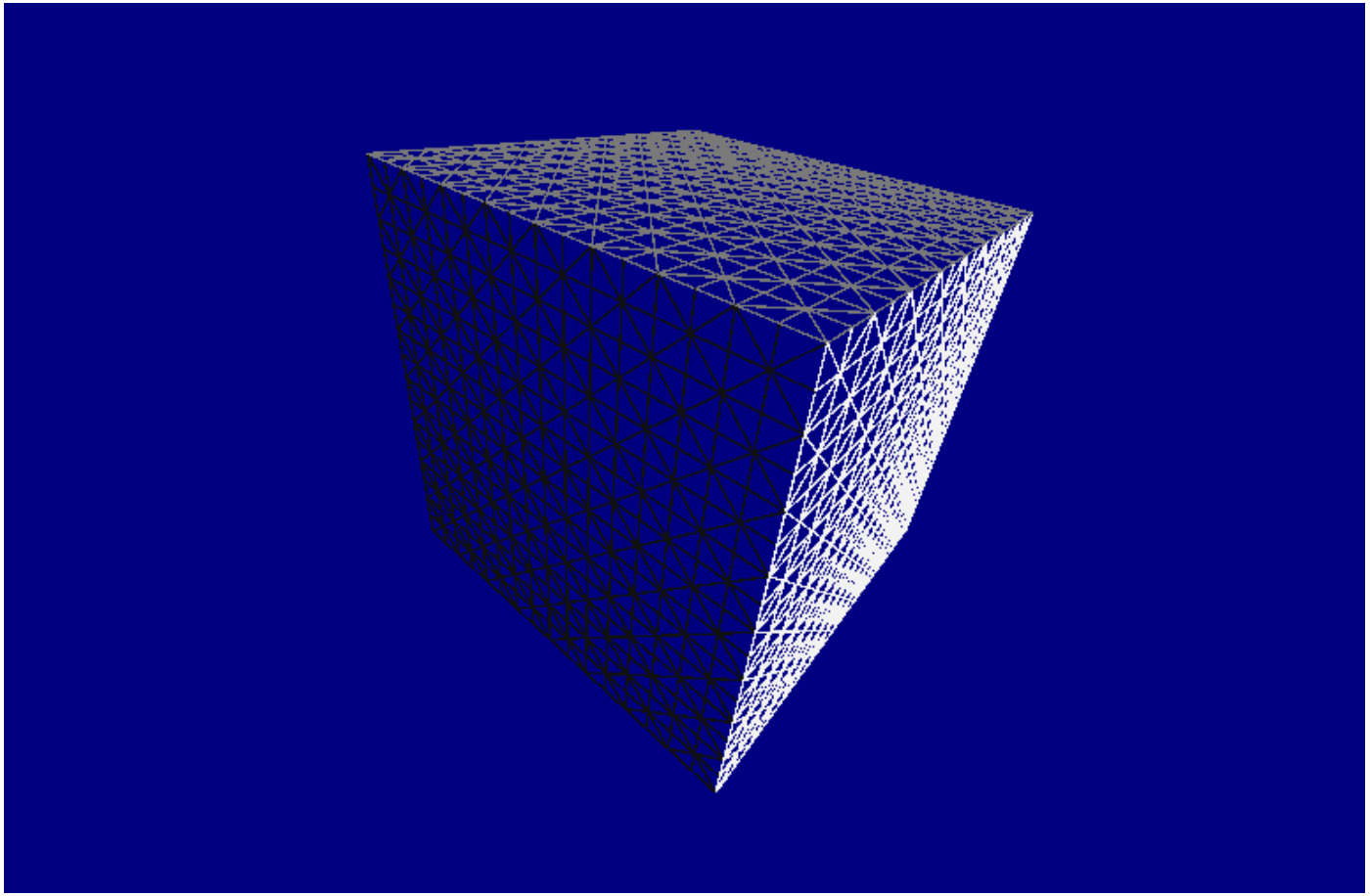
```
…
m_zc.Init();
m_zf.Init(hwnd,rectWnd.right, rectWnd.bottom,
       eRenderApi_DirectX11_Shadermodel50);
m_zv.InitFull(&m_zc);
m_zv.SetWireframeOn();
m_zl.Init(CHVector(1,0,1),CColor(1,1,1));
m_zgCube.Init(CHVector(1.0F,1.0F,1.0F),&m_zm);
m_zm.MakeTextureImage("textures\\white_image.jpg");
m_zr.AddFrameHere(&m_zf);
m_zf.AddViewport(&m_zv);
m_zr.AddScene(&m_zs);
…
```

Lösung zur Subdivide Cube-Übung (3/6)

Init-Methode in CGame.cpp

```
…
m_zs.AddPlacement(&m_zp);
m_zs.AddPlacement(&m_zpCamera);
m_zs.AddParallelLight(&m_zl);
m_zpCamera.AddCamera(&m_zc);
m_zp.Translate(CHVector(0,0,-3));
m_zp.AddGeo(&m_zgCube);
m_zgCube.Subdivide(0.2F);
…
```

Ausgabe



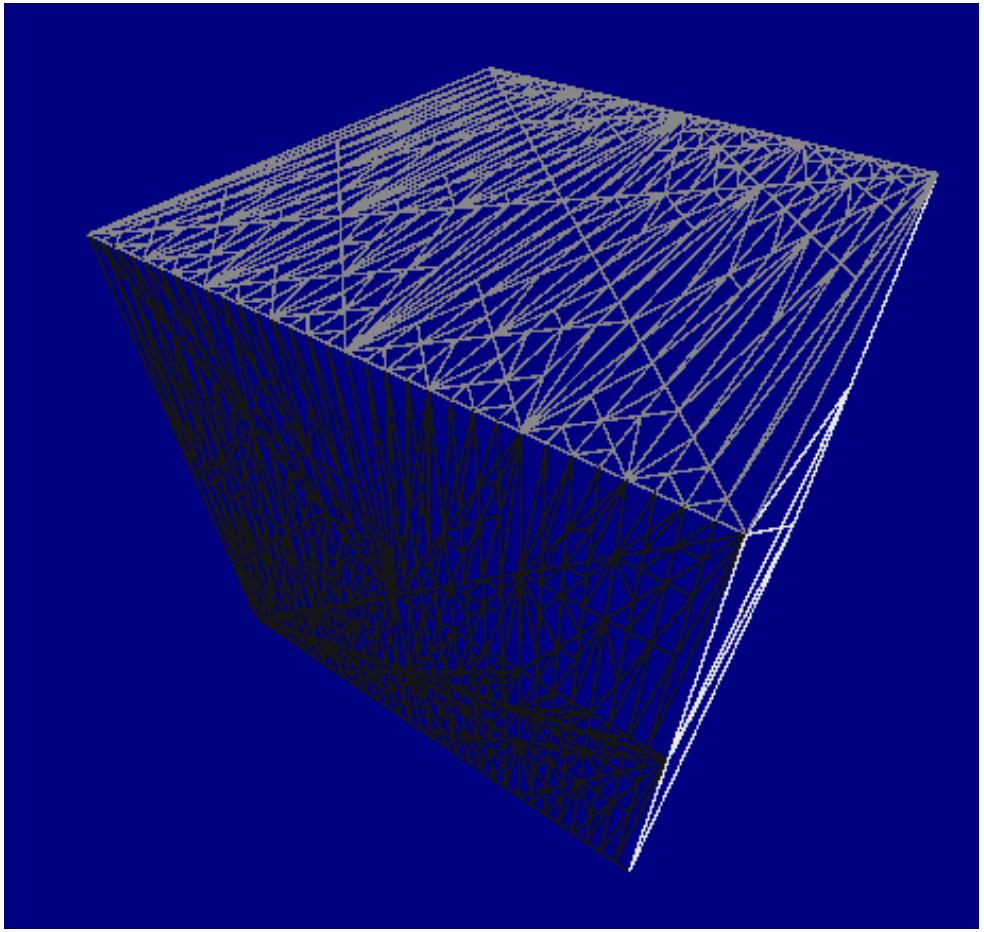
Lösung zur Subdivide Cube-Übung (5/6)

Init-Methode in CGame.cpp

```
// wie vorher
m_zs.AddPlacement(&m_zp);
m_zs.AddPlacement(&m_zpCamera);
m_zs.AddParallelLight(&m_zl);
m_zpCamera.AddCamera(&m_zc);
m_zp.Translate(CHVector(0,0,-3));
m_zp.AddGeo(&m_zgCube);
m_zgCube.SubdivideZ(1.0F);
m_zgCube.SubdivideX(0.1F);
```

… und hier noch die Lösung für die schnellen Nerds

Nerd-Ausgabe



Kapitel 2

1

TAPERING

3

4

5

Tapering-Methoden

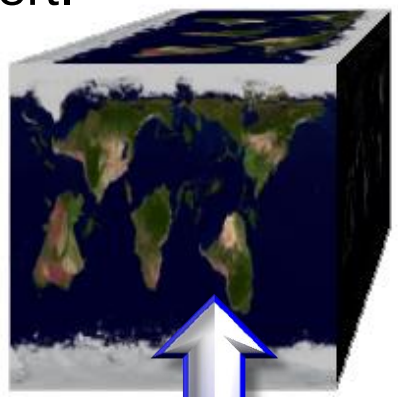
```
void TaperX(float fStrength,
            bool bInfluenceX=false,
            bool bInfluenceY=true,
            bool bInfluenceZ = true);
void TaperY(float fStrength,
            bool bInfluenceX=true,
            bool bInfluenceY=false,
            bool bInfluenceZ = true);
void TaperZ(float fStrength,
            bool bInfluenceX=true,
            bool bInfluenceY=true,
            bool bInfluenceZ = false);
```

Hier die Tapering-Methoden derKlasse CGeo. Der Parameter fStrength gibt die Stärke des Taperings in Abhängigkeit von der Entfernung zum lokalem Ursprung an. Zum Beispiel bedeutet ein Strength-Wert von 0.5 in 16 Einheiten Entferung eine 9-fache Skalierung (0.5*16+1=9). Die Influence-Parameter geben an, in welche Richtungen skaliert wird.

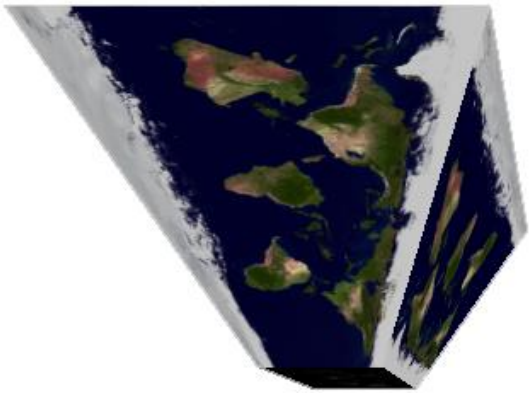
Beispielserie: Tapering auf Einheitswürfel

Im Folgenden ein paar Tapering-Beispiele mit verschiedenen Parametrisierungen.

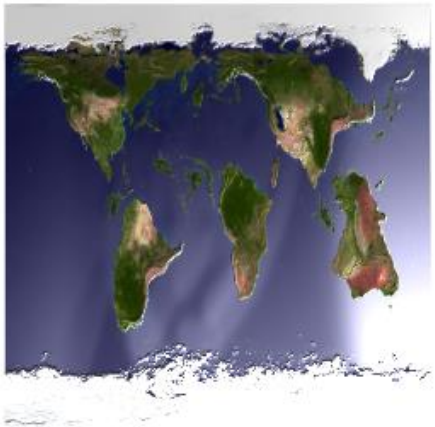
- Die Beispiele beziehen sich stets auf TaperY, es wird also immer bezüglich der Y-Achse getapert. TaperX und TaperZ würden zu gleichartigen Ergebissen auf den anderen Achsen führen.
- Das Tapering wurde stets auf einem Einheitswürfel appliziert.
- Zur besseren Optik wurde auf den Einheitswürfel eine planetare Textur gemappt.



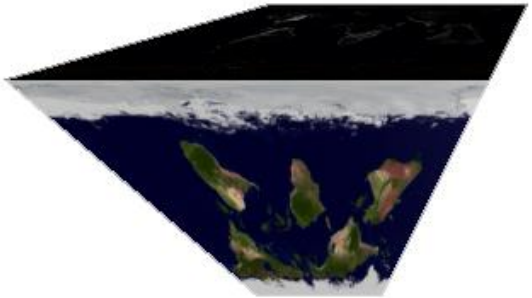
TaperY(0.5F, true, false, true)



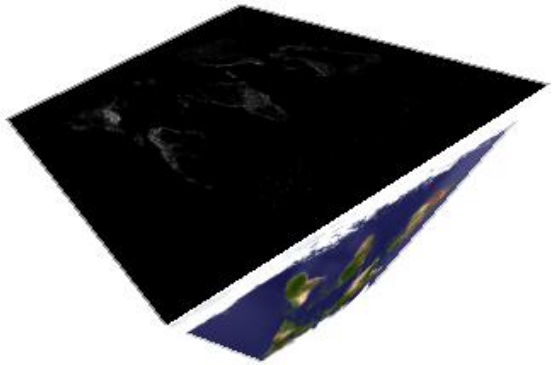
Sicht nahe X-Achse



Sicht nahe Y-Achse



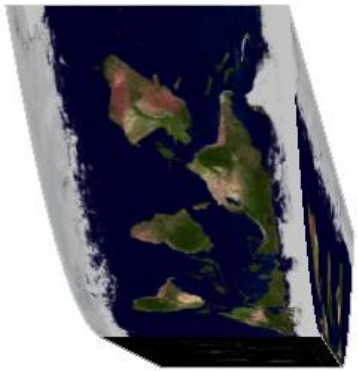
Sicht nahe Z-Achse



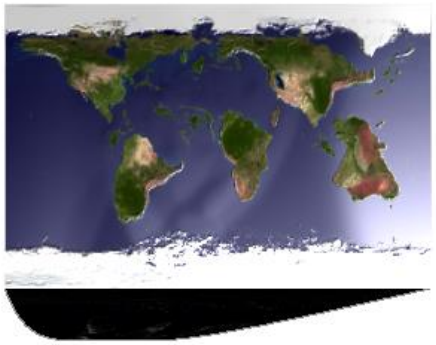
Sicht aus 1. Oktanten

Diese Influence-Parameter entsprechen den Default-Werten. TaperY(0.5) würde daher das selbe Ergebnis liefern.

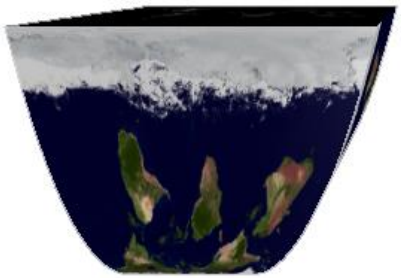
TaperY(0.5F, true, true, false)



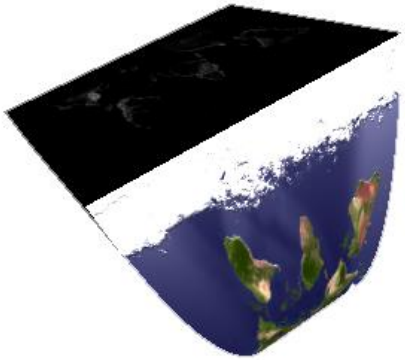
Sicht nahe X-Achse



Sicht nahe Y-Achse



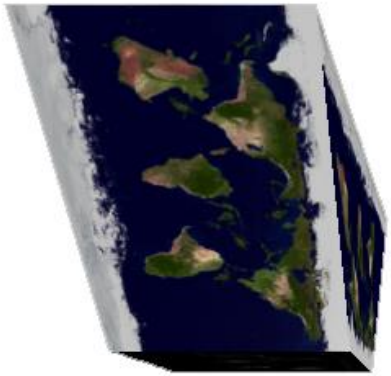
Sicht nahe Z-Achse



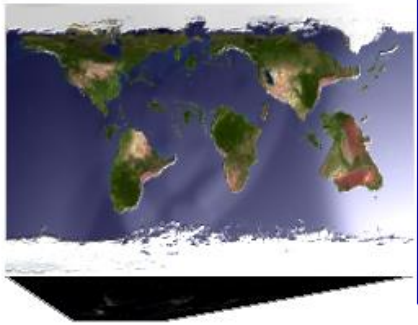
Sicht aus 1. Oktanten

Bei dieser Parametrisierung wird nur der X- und Y-Anteil (…,true, true, false) jeweils positiv um anderthalb (0.5F…) der Y-Position (TaperY) skaliert. Die Z-Anteile der Vertices bleiben unverändert.

TaperY(0.5F, true, false, false)

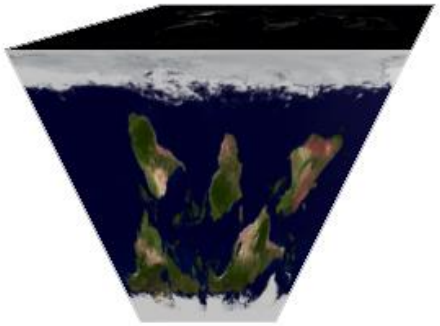


Sicht nahe X-Achse

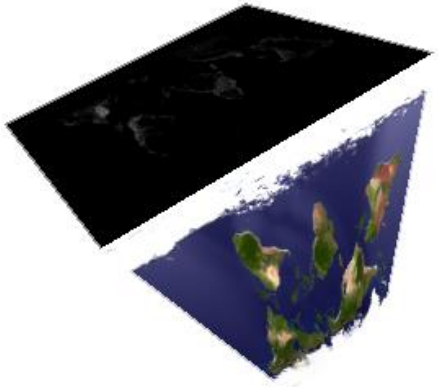


Sicht nahe Y-Achse

Hier gibt es nur ein Tapering des X-Anteils (…,true, true, false) bezüglich der Y-Achse (TaperY).

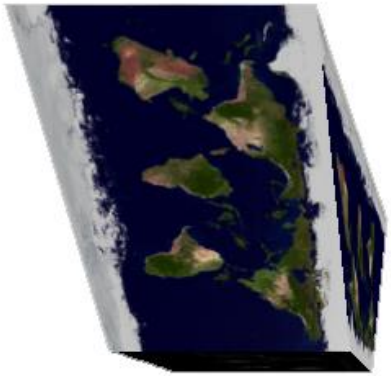


Sicht nahe Z-Achse

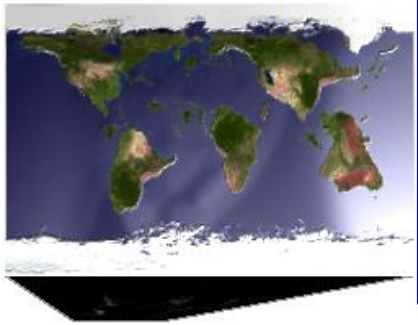


Sicht aus 1. Oktanten

TaperY(0.5F, true, false, false)

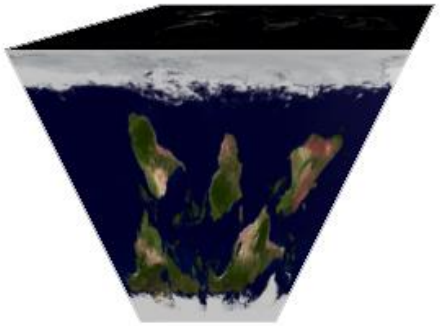


Sicht nahe X-Achse

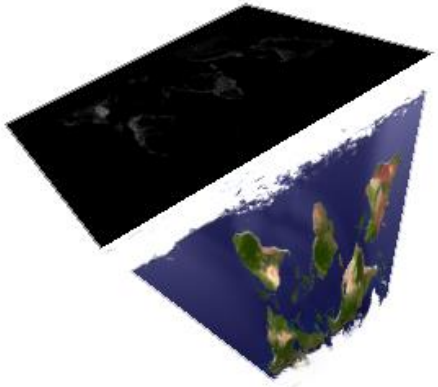


Sicht nahe Y-Achse

Hier gibt es nur ein Tapering des X-Anteils (…,true, true, false) bezüglich der Y-Achse (TaperY).

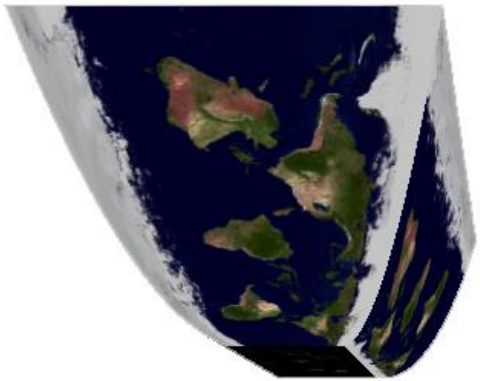


Sicht nahe Z-Achse

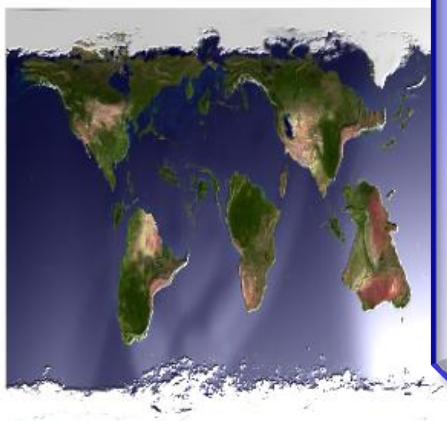


Sicht aus 1. Oktanten

TaperY(0.5F, true, true, true)

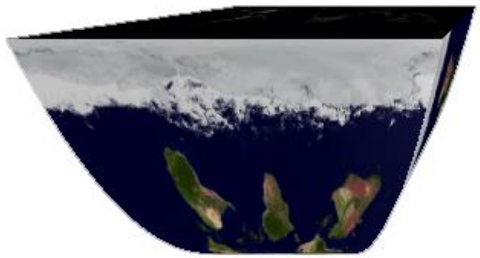


Sicht nahe X-Achse

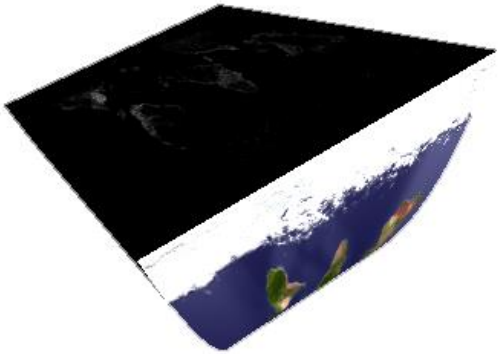


Sicht nahe Y-Achse

In diesem Beispiel werden alle Anteile, also X,Y und Z (…,true, true, false), mit dem Faktor 0.5F getapert.

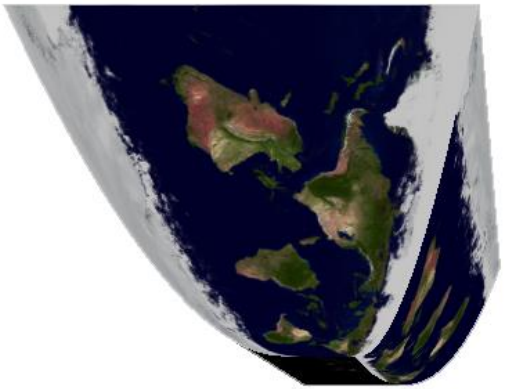


Sicht nahe Z-Achse



Sicht aus 1. Oktanten

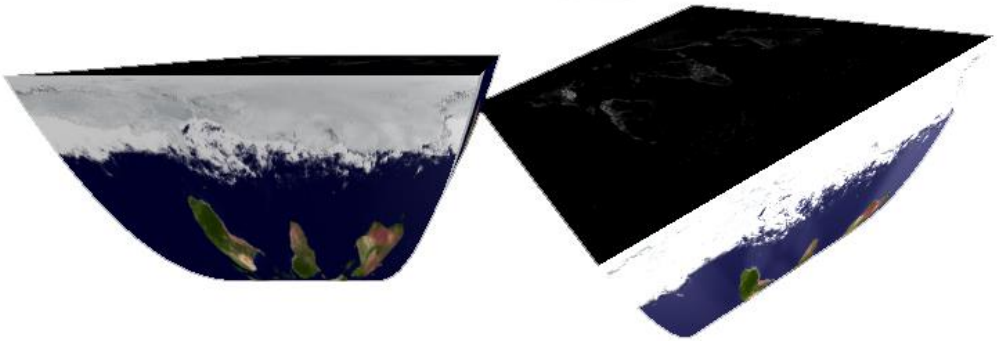
TaperY(0.7F, true, true, true)



Sicht nahe X-Achse

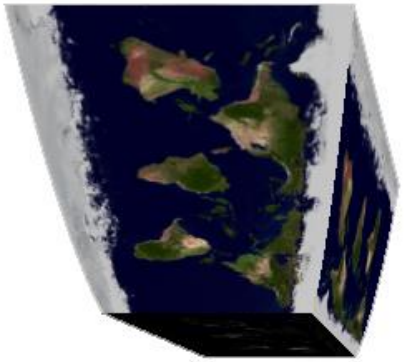
Sicht nahe Y-Achse

Größere Strength-Werte (hier 0.7F) bringen ein stärkeres Tapering …

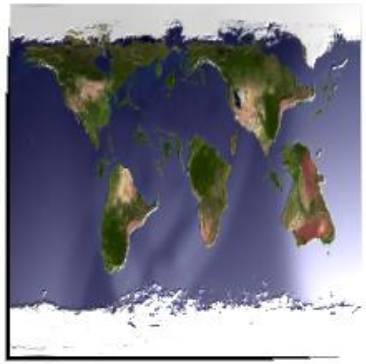


Sicht nahe Z-Achse Sicht aus 1. Oktanten

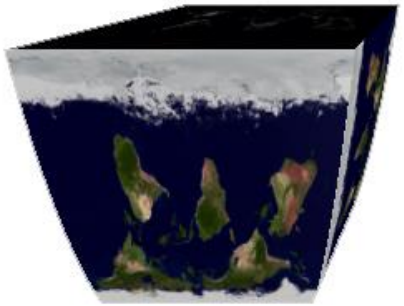
TaperY(0.2F, true, true, true)



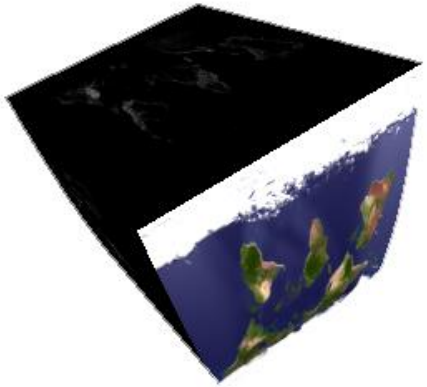
Sicht nahe X-Achse



Sicht nahe Y-Achse



Sicht nahe Z-Achse



Sicht aus 1. Oktanten

… und niedrigere Strength-Werte (hier 0.2F) führen naturgemäß zu einer kleineren Verzerrung.

Aufgabe „Ei-Tapering“

Übung zum Tapering

Erzeugen Sie eine Eierform mittels Tapering!

1

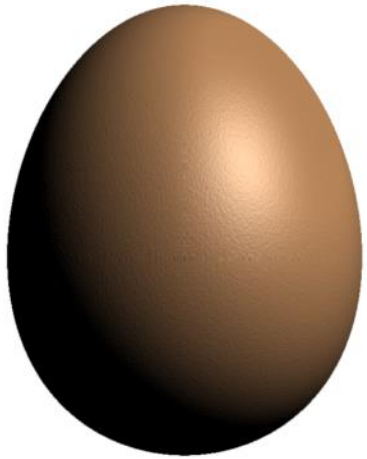
2

3

4

5

Freiwillige Übung für die schnellen Nerds:
Geben Sie dem Ei noch eine Image-Textur mit der typischen Eierschalenfarbe und eine Bumpmap-Textur für die Eierporen.



Nerd-Lösung „Ei Tapering“ (1/6)

Vektoria-Objekte in CGame.h

```
CRoot m_zr;
CScene m_zs;
CPlacement m_zpCamera;
CPlacement m_zp;
CGeoSphere m_zgSphere;
CHardware m_zh;
CFrame m_zf;
CViewport m_zv;
CCamera m_zc;
CParallelLight m_zl;
CMaterial m_zm;
```

Ausgangsgeometrie für das Ei ist eine Kugel, die mit Y-Tapering verzerrt werden soll.

Init-Methode in CGame.cpp


```
m_zc.Init();
m_zf.Init(hwnd,rectWnd.right, rectWnd.bottom,
     eRenderApi_DirectX11_Shadermodel50);
m_zv.InitFull(&m_zc);
m_zl.Init(CHVector(1.0F,1.0F,1.0F),
       CColor(1.0F,1.0F,1.0F));
m_zr.AddFrameHere(&m_zf);
m_zf.AddViewport(&m_zv);
m_zr.AddScene(&m_zs);
m_zs.AddPlacement(&m_zp);
m_zs.AddPlacement(&m_zpCamera);
m_zs.AddParallelLight(&m_zl);
m_zpCamera.AddCamera(&m_zc);
m_zp.Translate(CHVector(0.0F,-1.0F,-20.0F));
m_zp.AddGeo(&m_zgSphere);
…
```

Diesen Codeteil brauchen wir ja nicht nochmal zu erklären, kennen wir ja wir alles schon. (Foliensatz 02)

Init-Methode in CGame.cpp

Für die spätere Eiform brauchen wir zunächst eine Kugel.

Tapering wirkt sich nur auf die Vertices aus, daher enges Mesh wählen, z.B. 100 horizontale * 100 verikale Vertices!

```
…
m_zgSphere.Init(1.0F,&m_zm, 100,100,
     S_GEOELLIPSOIDMAPPING_QUADROBICYLINDRICAL);
…
```

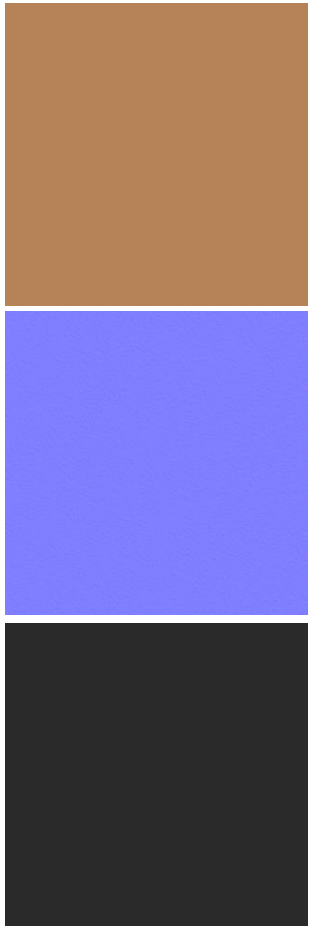
Die Mappingart S_GEOELLIPSOIDMAPPING_QUADROBICYLINDRICAL verhindert, dass die Granularitätsunterschiede zwischen Polen und Äquator der Polarkoordinatenkugel zu groß werden.

Nerd-Lösung „Ei Tapering“ (4/6)

Init-Methode in CGame.cpp

Eierschalenfarben ist in RGB ca. (182, 131, 88)

```
…
m_zm.MakeTextureImage
       ("textures\\ei_image.jpg");
m_zm.MakeTextureBump
       ("textures\\ei_bump.jpg");
m_zm.MakeTextureSpecular
       ("textures\\ei_specular.jpg");
…
```



Eier haben nur ein geringes Glanzlicht, daher für die spekulare Textur ein dunkles Grau wählen.

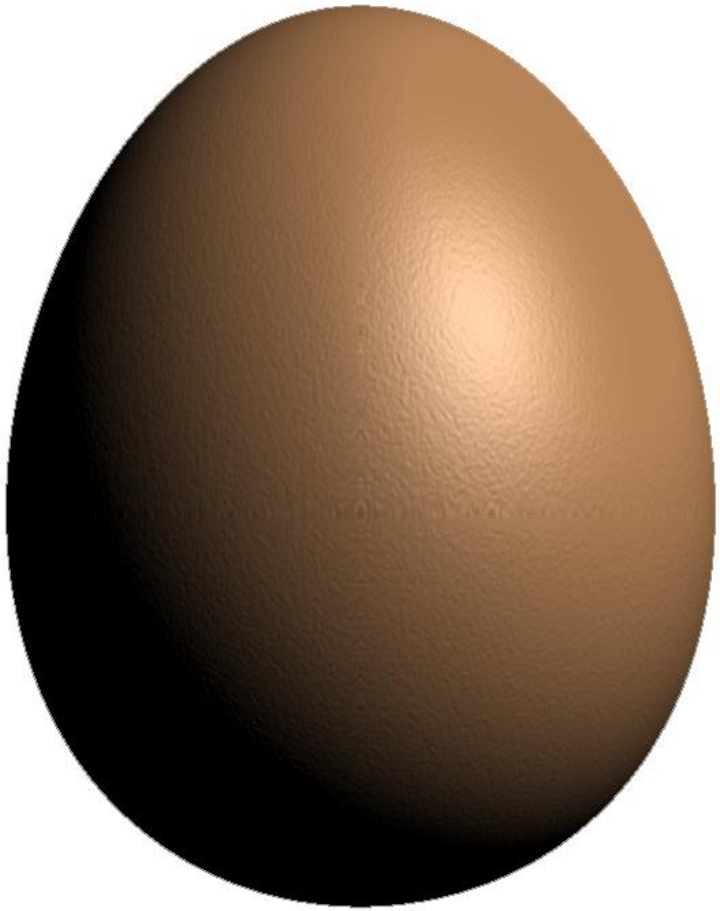
Init-Methode in CGame.cpp


```
…
CHMat m;
m.TranslateY(1.0F);
m_zgSphere.Transform(m);
m_zgSphere.TaperY(0.15F, false, true,false);
```

Die Kugelvertices werden durch Transform erst in den positiven Y-Bereich verschoben!

Dann folgt das eigentliche Tapering. Duch die Parametrisierung (…, false, true, false) wird nur die Y-Achse beeinflusst, so dass die Kugel oben in die Länge deformiert wird und eine Eiform entsteht. Die Stärke ist dabei mit 0,15 niedrig gewählt.

Nerd-Ausgabe



Kapitel 3

Twisting-Methoden

```
void TwistX(float faStrength);
void TwistY(float faStrength);
void TwistZ(float faStrength);
```

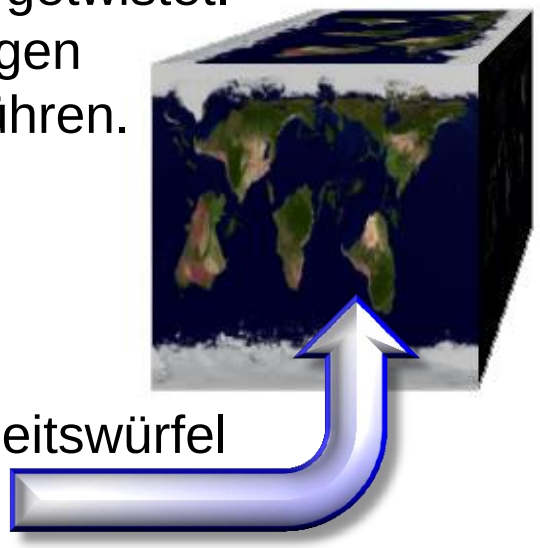
Der Parameter faStrength gibt die Twisting-Stärke als Drehwinkel pro Unit an.

Beispiel für TwistY: Ein faStrength-Wert von HALFPI „verquirlt" einen Vertex mit der Y-Position 1 um 90° und einen Vertex mit der Y-Position 3 um 270° um die Y-Achse.

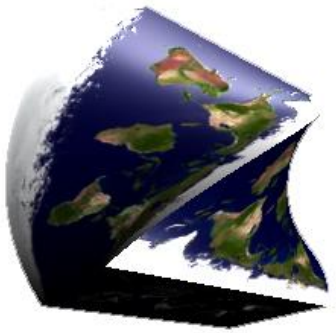
Beispielserie: Twisting auf Einheitswürfel

Im Folgenden ein paar Twisting-Beispiele mit verschiedenen Parametrisierungen.

- Die Beispiele beziehen sich stets auf TwistY, es wird also immer bezüglich der Y-Achse getwistet. TwistX und TwistZ würden zu gleichartigen Ergebissem auf den anderen Achsen führen.
- Das Twisting wurde stets auf einem Einheitswürfel appliziert.
- Zur besseren Optik wurde auf den Einheitswürfel eine planetare Textur gemappt.

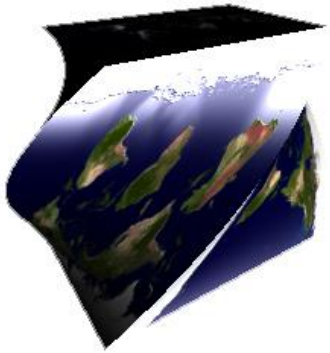


TwistY(QUARTERPI)

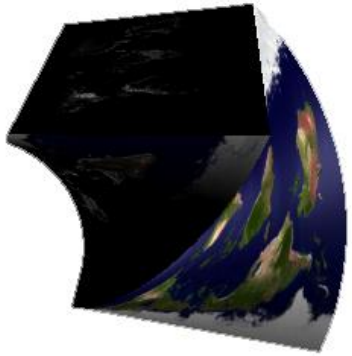


Sicht nahe X-Achse

Sicht nahe Y-Achse

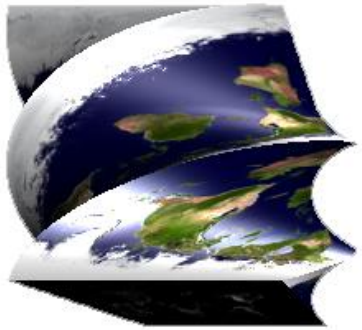


Sicht nahe Z-Achse



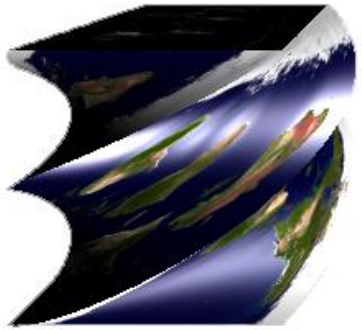
Sicht aus 1. Oktanten

TwistY(HALFPI)

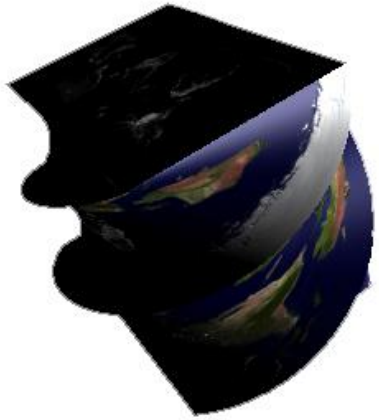


Sicht nahe X-Achse

Sicht nahe Y-Achse

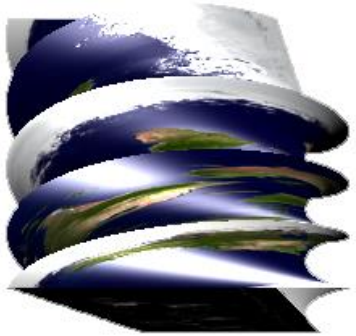


Sicht nahe Z-Achse

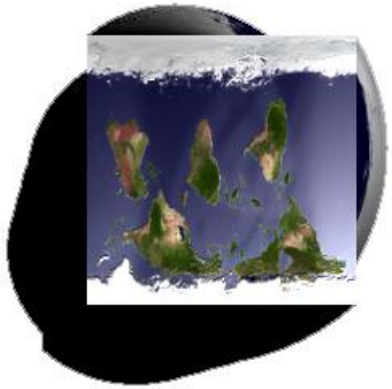


Sicht aus 1. Oktanten

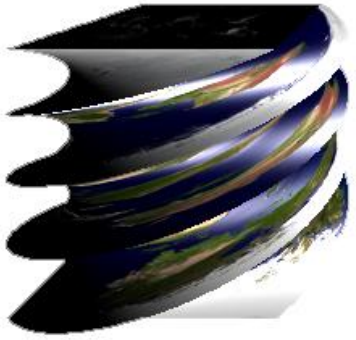
TwistY(PI)



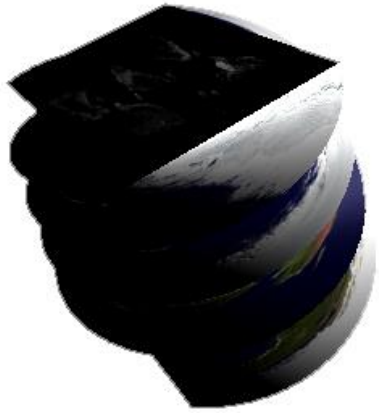
Sicht nahe X-Achse



Sicht nahe Y-Achse

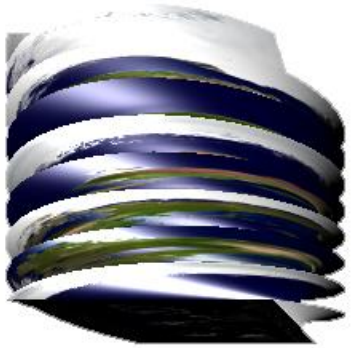


Sicht nahe Z-Achse



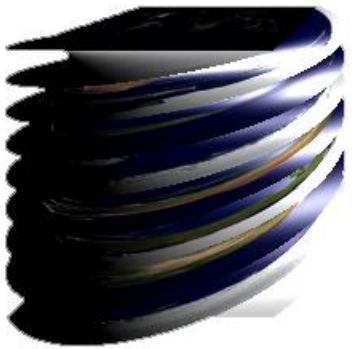
Sicht aus 1. Oktanten

TwistY(TWOPI)



Sicht nahe X-Achse

Sicht nahe Y-Achse



Sicht nahe Z-Achse

Sicht aus 1. Oktanten

Übung „Feder“

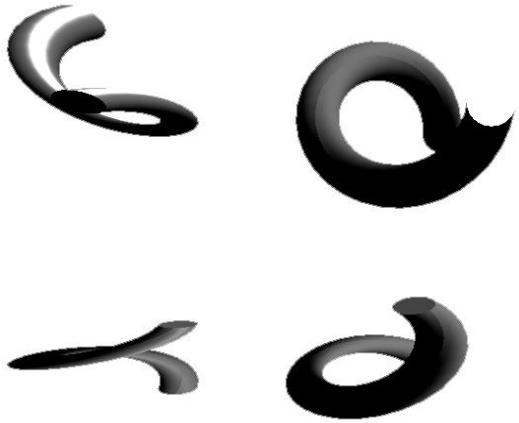
1

Erzeugen Sie eine technische Feder!
Tipp: Verschieben Sie zuerst einen Zylinder.
Twisten Sie ihn dann!

2

3

Freiwillige Übung für die schnellen Nerds:
Versuchen Sie den Durchmesser der Feder konstant zu halten!
Tipp: Applizieren Sie eine Skalierungsmatrix!



4

5

Lösung „Feder“ (1/7)

Vektoria-Objekte in CGame.h

```
CRoot m_zr;
CScene m_zs;
CPlacement m_zpCamera;
CPlacement m_zp;
CGeoCylinder m_zgCylinder;
CTriangleList *
      m_ptrianglelist;
CHardware m_zh;
CFrame m_zf;
CViewport m_zv;
CCamera m_zc;
CParallelLight m_zl;
CMaterial m_zm;
```

Ausgangsgeometrie für die Feder ist der Zylinder m_zgCylinder.
Da CGeoCylinder aber von der Klasse CTriangleStrip abgeleitet ist und Dreiecksstreifen nicht getwistet werden können, muss er vor dem Twisting erst in eine TriangleList umgewandelt werden, dafür brauchen wir die Containervariable m_ptrianglelist.

Init-Methode in CGame.cpp


```
m_zc.Init(0.2);
m_zf.Init(hwnd,rectWnd.right, rectWnd.bottom,
eRenderApi_DirectX11_Shadermodel50);
m_zv.InitFull(&m_zc);
m_zl.Init(CHVector(1,1,1),CColor(1,1,1));
m_zr.AddFrameHere(&m_zf);
m_zf.AddViewport(&m_zv);
m_zr.AddScene(&m_zs);
m_zs.AddPlacement(&m_zp);
m_zs.AddPlacement(&m_zpCamera);
m_zs.AddParallelLight(&m_zl);
m_zpCamera.AddCamera(&m_zc);
m_zp.Rotate(CHVector(0.2F,0.3F,0.4F).Normal(),1.0F);
m_zp.TranslateDelta(CHVector(0,0,-20));
…
```

Diesen Codeteil brauchen wir ja nicht nochmal zu erklären, kennen wir ja schon.

Init-Methode in CGame.cpp

1

Metall hat eine graue Grundfarbe in RGB (128, 128, 128)…

2

```
…
m_zm.MakeTextureImage("textures\\grey_image.jpg");
m_zm.SetTextureSpecularWhite();
…
```

3

… welche ein besonderes helles Glanzlicht aufweist, mit SetTextureSpecularWhite wird es maximiert.

4

Wenn wir wollten, könnten wir mit einer Bumpmap-Textur noch zusätzlichen Realismus erzeugen (z.B. für gebürstetes Metall)

5

Lösung „Feder“ (4/7)

Init-Methode in CGame.cpp

Für die Federform brauchen wir zunächst einen Zylinder (hier mit 0,3 Einheiten Durchmesser und einer Einheit Länge), …

```
…
m_zgCylinder.Init(0.3F, 0.3F, 1.0F, &m_zm);
CHMat m;
m.TranslateX(1.0F);
m_zgCylinder.Transform(m);
```

… den wir von der Y-Achse wegverschieben, hier als Beispiel um eine Einheit in Richtung der X-Achse. TranslateZ wäre aber genauso gegangen.

Init-Methode in CGame.cpp

Für die Federform brauchen wir zunächst einen Zylinder (hier mit 0,3 Einheiten Durchmesser und einer Einheit Länge), …

```
…
m_zgCylinder.Init(0.3F, 0.3F, 1.0F, &m_zm);
CHMat m;
m.TranslateX(1.0F);
m_zgCylinder.Transform(m);
```

… den wir von der Y-Achse wegverschieben, hier als Beispiel um eine Einheit in Richtung der X-Achse. TranslateZ wäre aber genauso gegangen.

Lösung „Feder“ (6/7)

Init-Methode in CGame.cpp

Den Zylinder müssen wir zuerst mittels CopyToTriangleList in eine Dreiecksliste umwandeln, da Dreiecksstreifen kein Twisting zulassen.

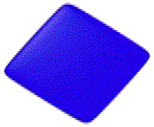
```
…
m_ptrianglelist = m_zgCylinder.CopyToTriangleList();
m_zp.AddGeo(m_ptrianglelist);

m_ptrianglelist->SubdivideY(0.05F);
m_ptrianglelist->TwistY(TWOPI);
```

Subdivide nicht vergessen, sonst fehlen die Vertices zum twisten!

Wir twisten mit 2*π einmal um die Y-Achse. Da wir vorher den Zylinder von der Y-Achse wegtransliert haben, entsteht dadurch eine Federform.

Ausgabe

Kapitel 4



Bending-Methoden

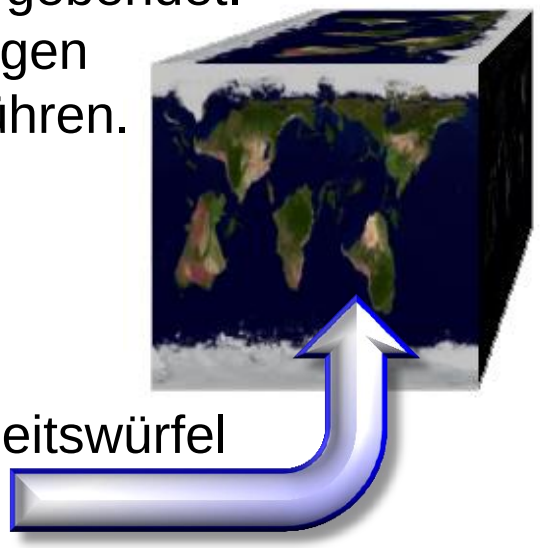
```
void BendX(float fRadius, float faSection);
void BendY(float fRadius, float faSection);
void BendZ(float fRadius, float faSection);
```

Mit den „Bending“-Methoden werden die Vertices eines geometrischen Körpers um den Winkel faSection (gegeben im Bogenmaß) mit dem Radius fRadius um die jeweilige kartesische Koordinatenachse gedreht. Daraus resultiert ein „verbogener“ Körper, der sich durch die Biegung zur Seite neigt.

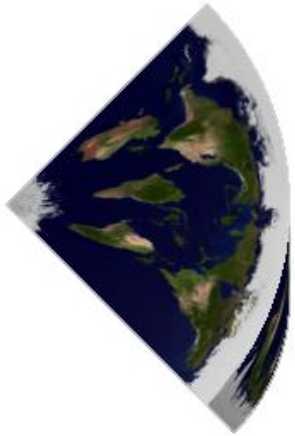
Beispielserie: Bending auf Einheitswürfel

Im Folgenden ein paar Bending-Beispiele mit verschiedenen Parametrisierungen.

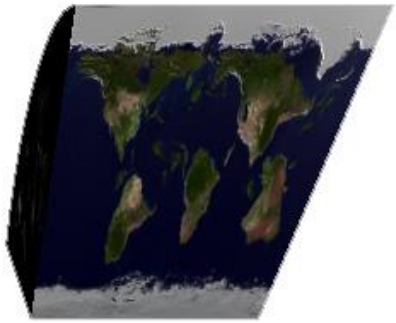
- Die Beispiele beziehen sich stets auf BendY, es wird also immer bezüglich der Y-Achse gebendet. BendX und BendZ würden zu gleichartigen Ergebissem auf den anderen Achsen führen.
- Das Bending wurde stets auf einen Einheitswürfel appliziert.
- Zur besseren Optik wurde auf den Einheitswürfel eine planetare Textur gemappt.



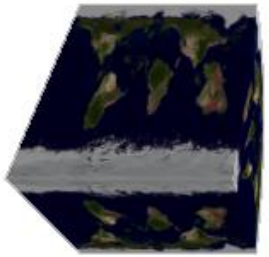
BendY(1.0F, QUARTERPI)



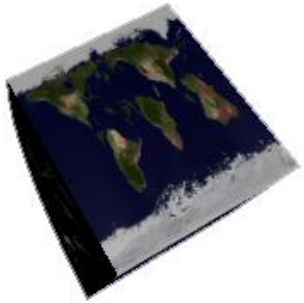
Sicht nahe X-Achse



Sicht nahe Y-Achse

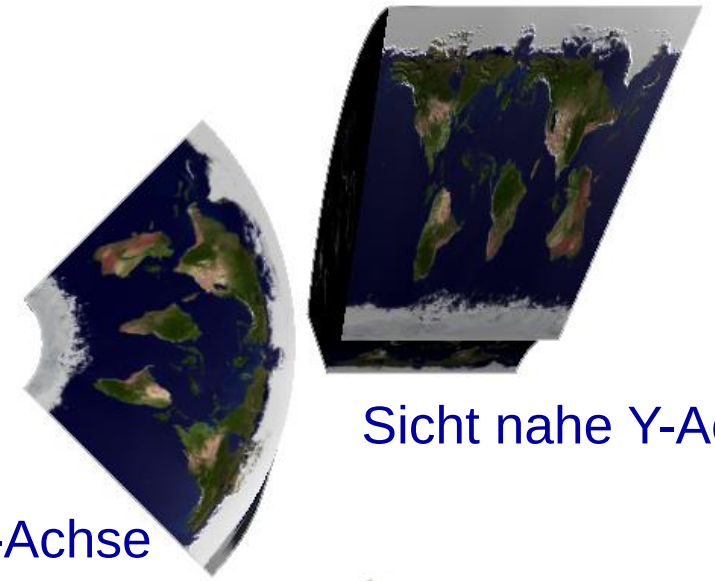


Sicht nahe Z-Achse



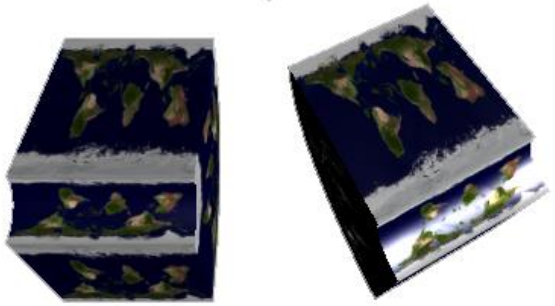
Sicht aus 1. Oktanten

BendY(1.5F, QUARTERPI)



Sicht nahe Y-Achse

Sicht
nahe X-Achse

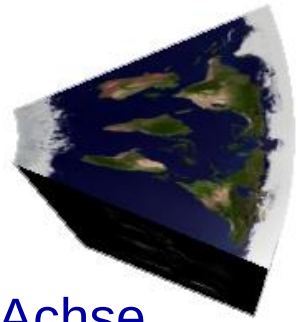


Sicht nahe Z-Achse

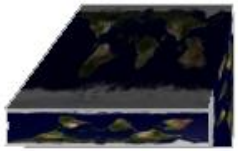
Sicht aus 1. Oktanten

BendY(1.5F, PI/8.0F)

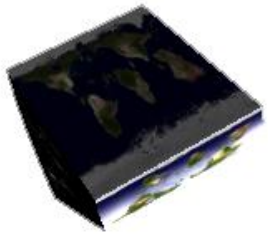
Sicht nahe Y-Achse



Sicht
nahe X-Achse

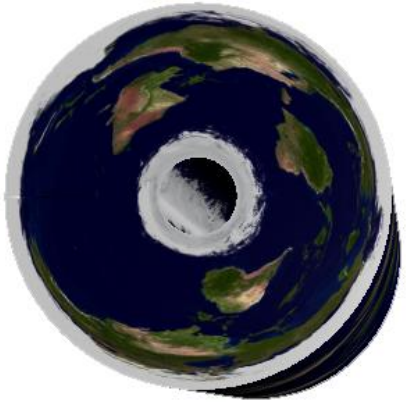


Sicht nahe Z-Achse

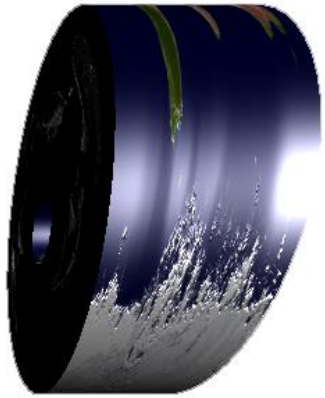


Sicht aus 1. Oktanten

BendY(1.5F, PI)



Sicht nahe X-Achse

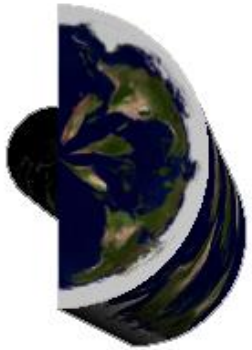


Sicht nahe Y-Achse

Sicht nahe Z-Achse

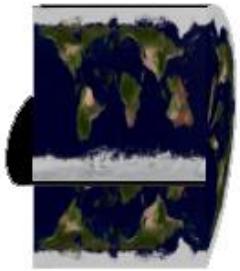
Sicht aus 1. Oktanten

BendY(0.5F, HALFPI)



Sicht nahe X-Achse

Sicht nahe Y-Achse

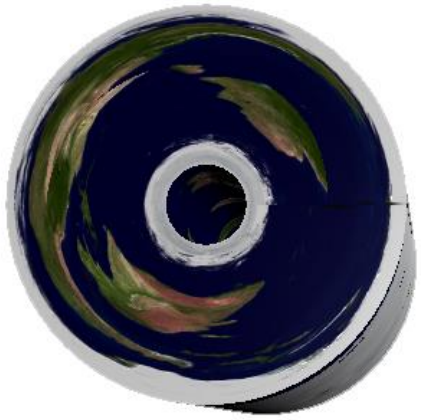


Sicht nahe Z-Achse

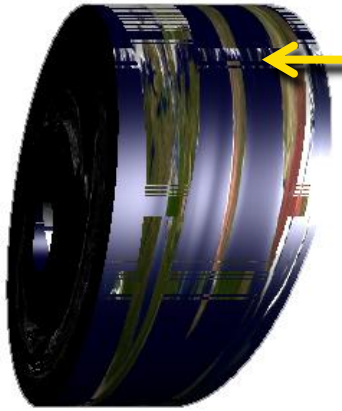
Sicht aus 1. Oktanten

Befindet sich der Drehpunkt im Körper, entstehen hässliche Artefakte, denn dann wird die Drehrichtung einiger Polygone geändert. Dies führt zum Backface-Culling einiger Körperteile.

BendY(1.5F, TWOPI)



Sicht nahe X-Achse



Sicht nahe Y-Achse

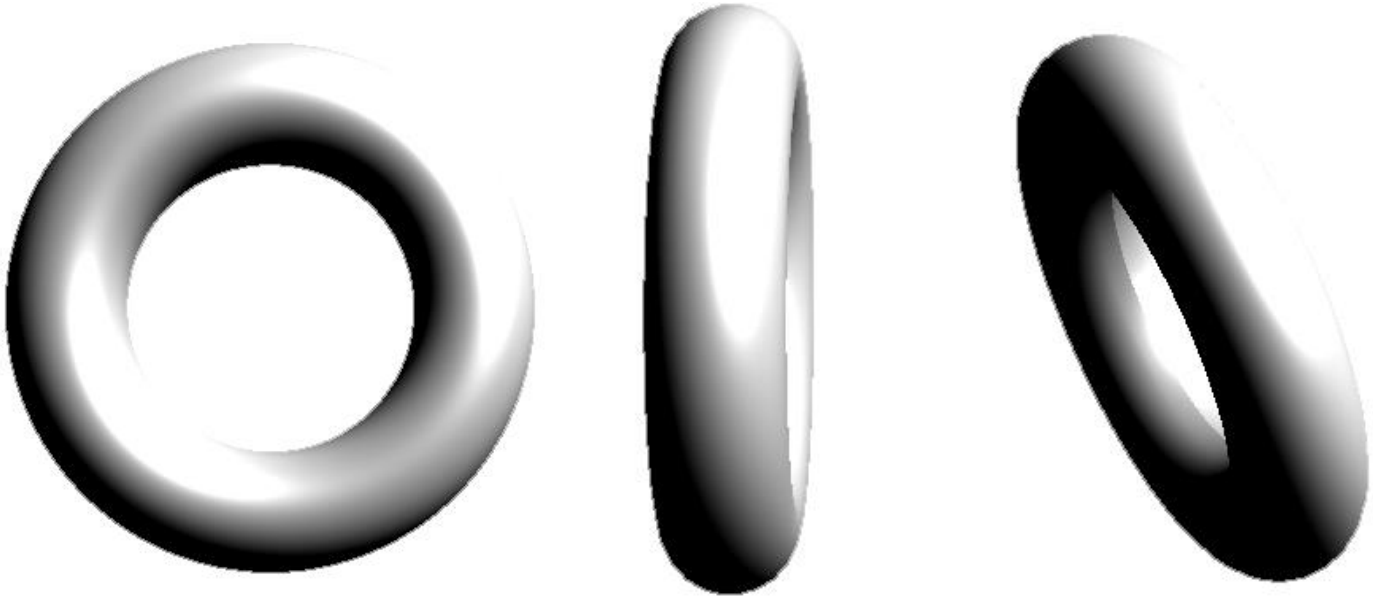
Ebenfalls vorsichtig muss man sein, wenn der Körper in sich selbst gebeugt wird, denn auch hier können hässliche Artefakte durch Polygonüberlagerungen entstehen.

Sicht nahe Z-Achse

Sicht aus 1. Oktanten

Übung „Torus“

Erzeugen Sie einen Torus!



Freiwillige Übung für die schnellen Nerds:
Erzeugen Sie eine Kette!

Kurzlösung „Torus“ (1/1)

Veränderung der Init-Methode

Erst einen Zylinder erzeugen, hier mit Radius 0,3 und 2,0 Einheiten Höhe. Ober und Unterseite brauchen wir nicht.

```
…
m_zgCylinder.Init(0.3F,0.3F,2.0F, &m_zm, 36,
false, false);
…
…
…
…
m_ptrianglelist =
        m_zgCylinder.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->BendY(1.0F, PI);
…
```

TriangleList aus einem TriangleStrip extrahieren, damit wir auf diesem ein Subdivide durchführen können.

Zylinder so weit biegen, bis sich seine Enden berühren, hier also PI eintragen, denn 2*π ≡ 360°.

Übung „Möbius“

1. Erzeugen Sie ein Möbiusband!
   Tipp: Erst twisten, dann benden!

2. Freiwillige Übung für die schnellen Nerds:
   Erezugen Sie zwei Möbiusbänder,
   die miteieinander ineinander hängen!

Kurzlösung „Möbiusband“ (1/1)

Veränderung der Init-Methode

Erst einen flachen Quader erzeugen.

```
…
m_zgCube.Init(CHVector(0.1F,1.0F,1.0F),&m_zm);
…
…
m_zgCube.Subdivide(0.03F);
…
…
…
…
m_zgCube.TwistY(HALFPI);
…
m_zgCube.BendY(2.5F, PI);
…
```

Subdivide nicht vergessen, damit genügend Vertices vorhanden sind!

Quader um 180° verquirlen, daher wegen der Gesamthöhe 2 (negativen Quaderbereich nicht vergessen!) HALFPI eintragen, denn 2*HALFPI = PI ≡ 180°.

Verquirlter Quader so weit biegen, bis sich seine Enden berühren, hier also PI eintragen, denn 2*π ≡ 360°.

Kapitel 5

Rippleing-Methoden

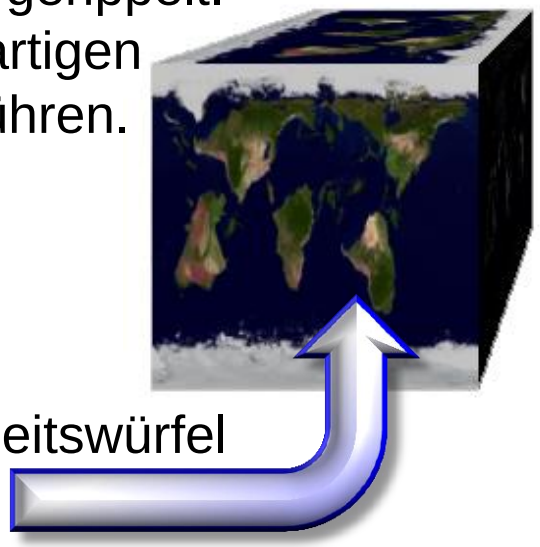
```
void RippleY (   float fAmplitude,
                 int iHarmonics,
                 float faPhase       = 0.0F,
                 bool bInfluenceX = true,
                 bool bInfluenceY = false,
                 bool bInfluenceZ = true);
```

Mit der RippleY-Methode werden die Vertices eines geometrischen Körpers iHarmonics-Mal um die Y-Achse geriffelt. Mit faPhase kann man die Riffel radial um den Winkel faPhase drehen. Die Influence-Parameter geben an, welche kartesischen Tupel der Vertices beeinflusst werden.

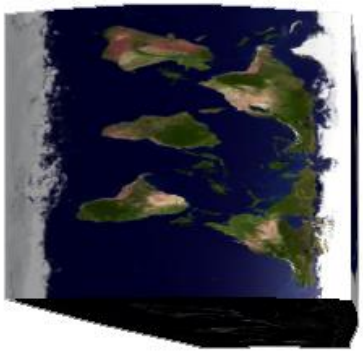
Beispielserie: Rippleing auf Einheitswürfel

Im Folgenden ein paar Rippleing-Beispiele mit verschiedenen Parametrisierungen.

- Die Beispiele beziehen sich stets auf RippleY, es wird also immer bezüglich der Y-Achse gerippelt. RippleX und RippleZ würden zu gleichartigen Ergebissem auf den anderen Achsen führen.
- Das Rippleing wurde stets auf einem Einheitswürfel appliziert.
- Zur besseren Optik wurde auf den Einheitswürfel eine planetare Textur gemappt.

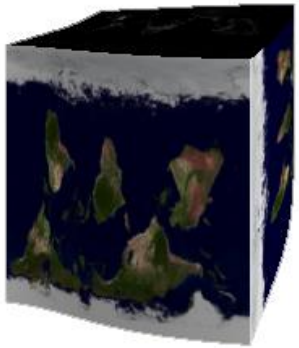


RippleY(0.3F, 1, 0.0F, true, false, true)

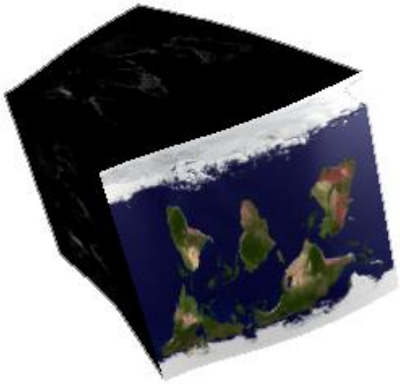


Sicht nahe X-Achse

Sicht nahe Y-Achse

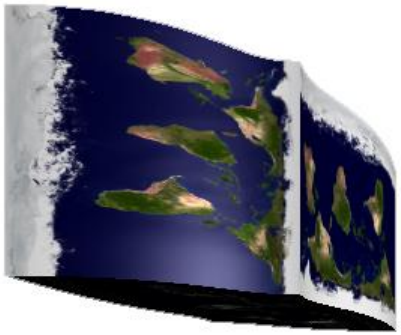


Sicht nahe Z-Achse



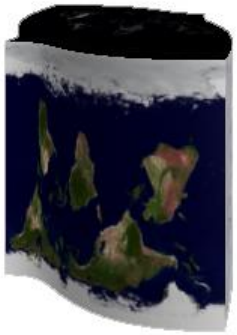
Sicht aus 1. Oktanten

RippleY(0.3F, 2, 0.0F, true, false, true)

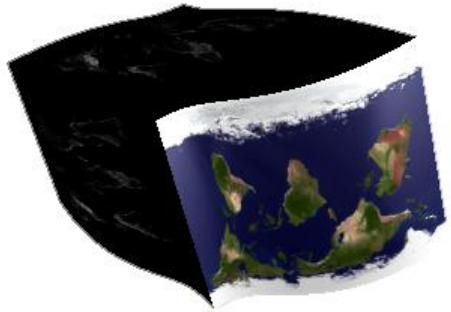


Sicht nahe X-Achse

Sicht nahe Y-Achse

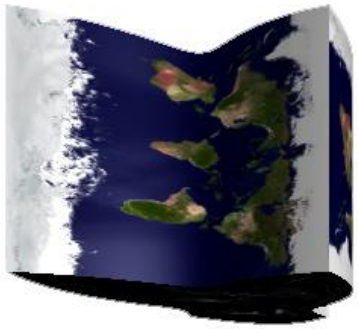


Sicht nahe Z-Achse



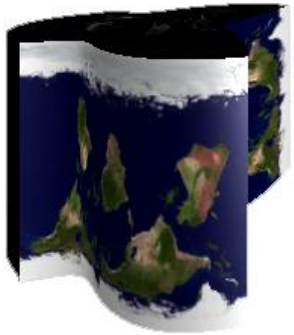
Sicht aus 1. Oktanten

RippleY(0.3F, 3, 0.0F, true, false, true)

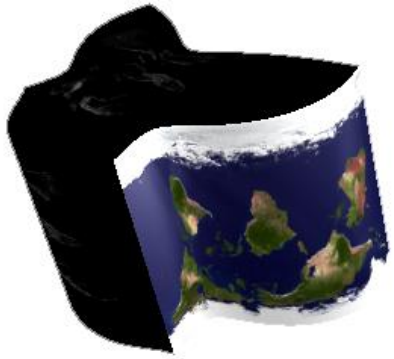


Sicht nahe X-Achse

Sicht nahe Y-Achse

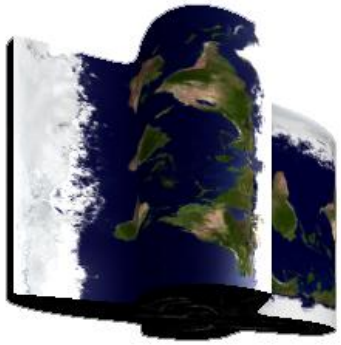


Sicht nahe Z-Achse



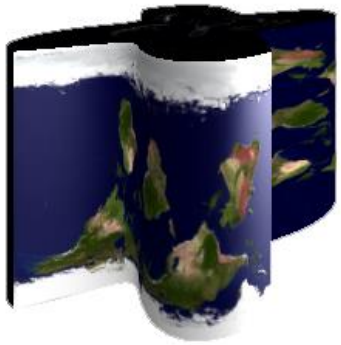
Sicht aus 1. Oktanten

RippleY(0.3F, 4, 0.0F, true, false, true)

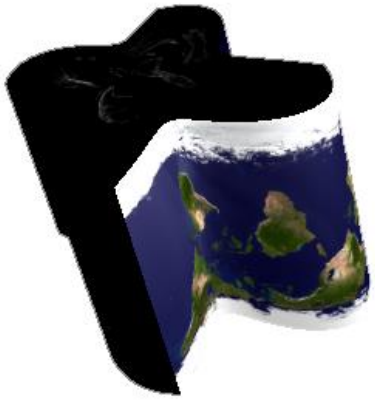


Sicht nahe X-Achse

Sicht nahe Y-Achse

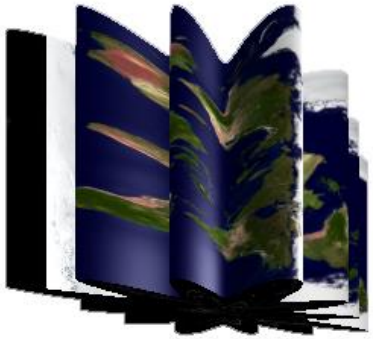


Sicht nahe Z-Achse

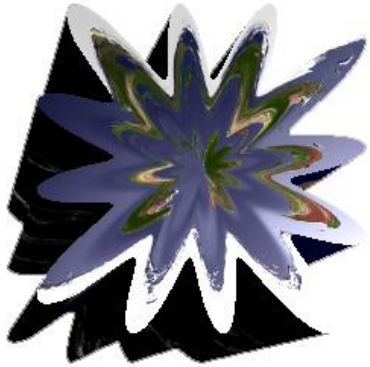


Sicht aus 1. Oktanten

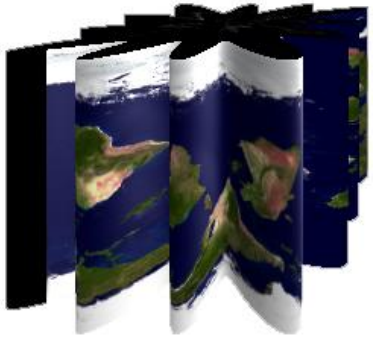
RippleY(0.3F, 12, 0.0F, true, false, true)



Sicht nahe X-Achse



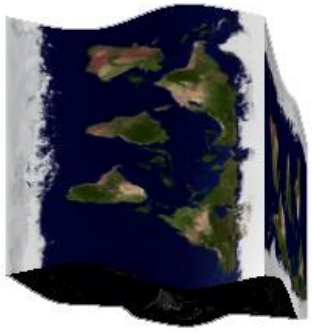
Sicht nahe Y-Achse



Sicht nahe Z-Achse

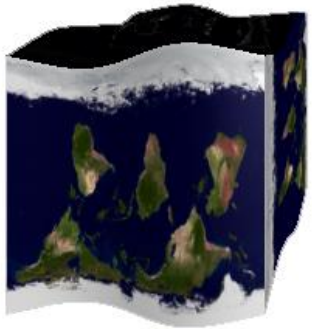
Sicht aus 1. Oktanten

RippleY(0.1F, 4, 0.0F, false, true, false)

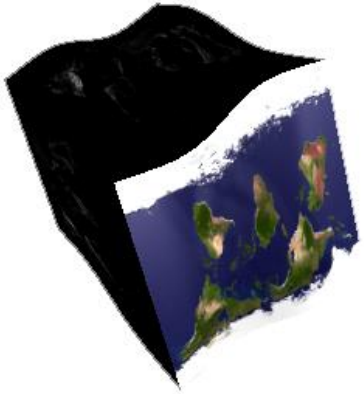


Sicht nahe X-Achse

Sicht nahe Y-Achse

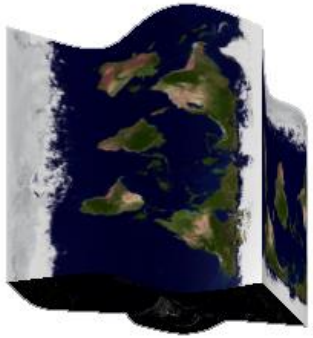


Sicht nahe Z-Achse



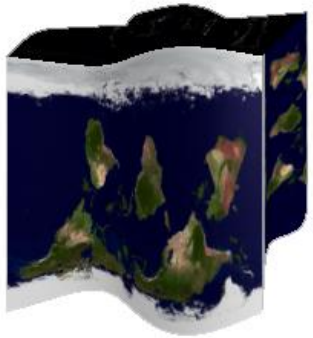
Sicht aus 1. Oktanten

RippleY(0.1F, 4, 0.0F, true, true, true)

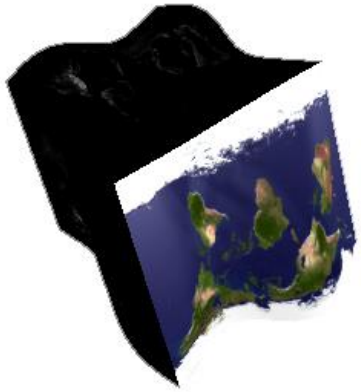


Sicht nahe X-Achse

Sicht nahe Y-Achse

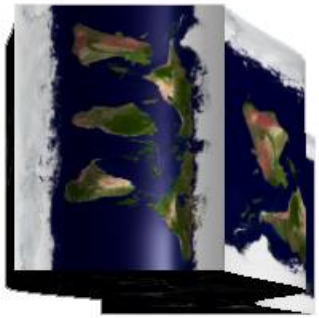


Sicht nahe Z-Achse

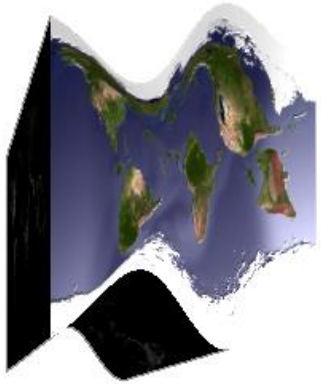


Sicht aus 1. Oktanten

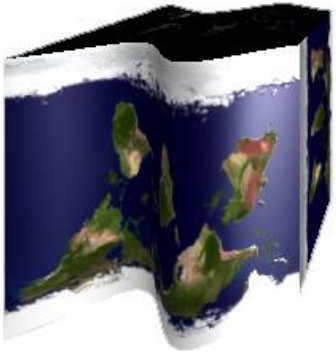
RippleY(0.3F, 5, 0.0F, false, false, true)



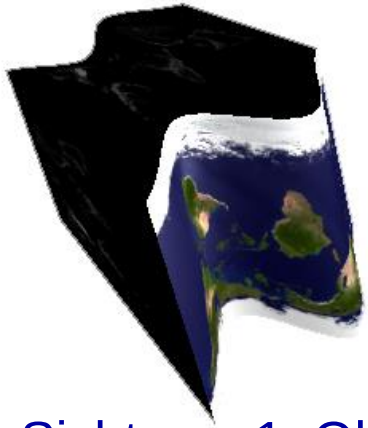
Sicht nahe X-Achse



Sicht nahe Y-Achse



Sicht nahe Z-Achse



Sicht aus 1. Oktanten

Rippleing-Methoden

```
void RippleYHard (      float fAmplitude,
                        int iHarmonics,
                        float faPhase = 0.0F,
                        bool bInfluenceX=true,
                        bool bInfluenceY=false,
                        bool bInfluenceZ = true);
```

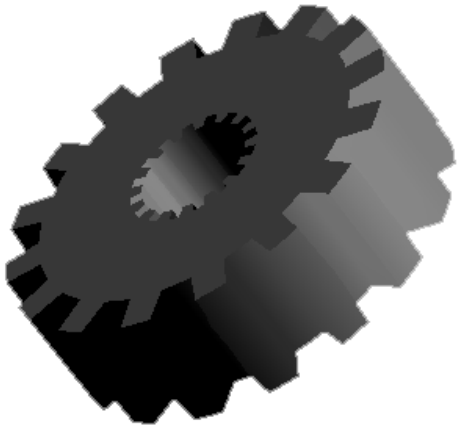
RippleY erzeugt sinoide Riffel.
RippleYHard erzeugt dagegen harte Rifflekanten.

Aufgabe zu Rippleing

Übung „Riffelkörper“

Erzeugen Sie folgende Körper:

Körper A



Körper B

Körper C

Freiwillige Übung für die schnellen Nerds:
Texturieren Sie die Körper sinnfällig.

Kurzlösung „Riffelkörper“ (1/3)

Körper A (wichtigste Veränderungen)

```
…
m_zgCylinder.Init(0.3F, 0.3F, 1.0F, &m_zm, 120);
…
m_ptrianglelist =
       m_zgCylinder.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->RippleY(0.5F,5,0.0F);
…
```

1 2 3 4 5

Kurzlösung „Riffelkörper“ (2/3)

Körper B (wichtigste Veränderungen)

1
2
3
4
5

```
…
m_zgTube.InitStraight(0.1F,0.3F,0.2F,&m_zm,240);
…
m_ptrianglelist = m_zgTube.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->RippleYHard(0.1F,16,0.0F);
…
```

Kurzlösung „Riffelkörper“ (3/3)

Körper C (wichtigste Veränderungen)

1
2
3
4
5

```
…
m_zgTube.InitStraight(0.1F,0.3F,0.2F,&m_zm,240);
…
m_ptrianglelist =
       m_zgTube.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->RippleY(0.1F,16,0.0F);
m_ptrianglelist->BendY(1.0F,TWOPI);
…
```

Waving-Methoden

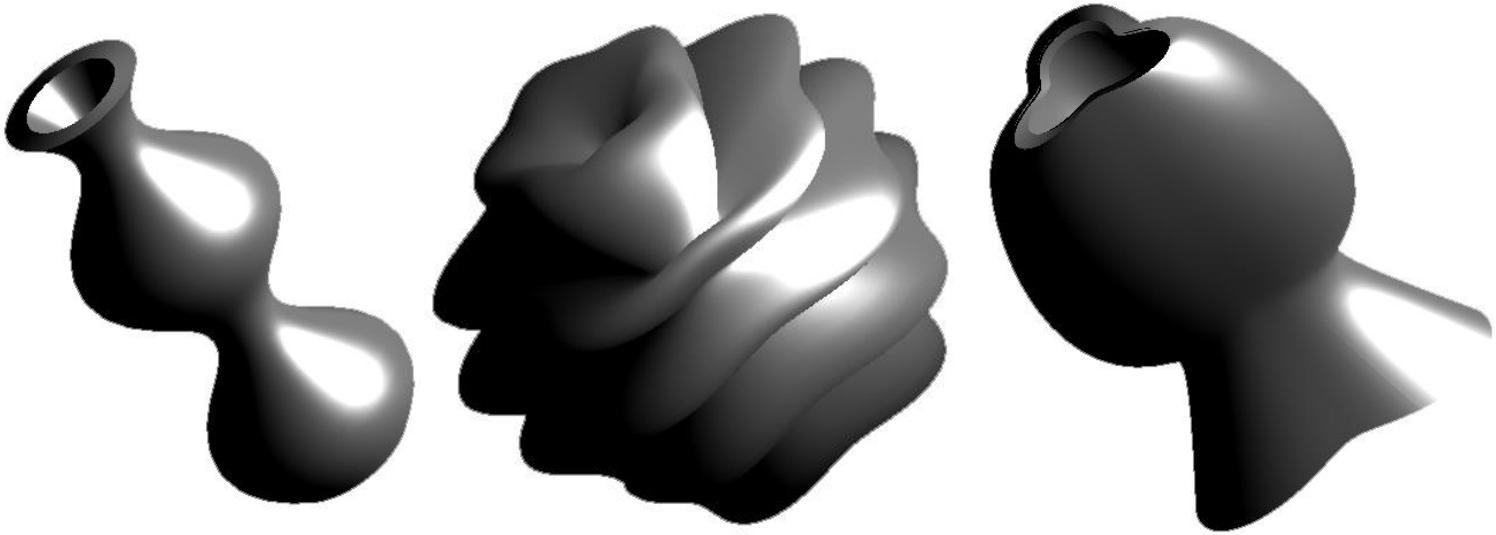
```
void WaveX(     float fAmplitude,
                float fWavelength,
                float faPhase = 0.0F,
                bool bInfluenceX=true,
                bool bInfluenceY=false,
                bool bInfluenceZ = true);
```

WaveX erzeugt sinoide Wellendeformationen in Richtung der X-Achse mit der Amplitude fAmplitude, der Wellenlänge fWavelength, der Phasenverschiebung im Bogenmaß faPhase und den Beeinflussungsflags bInfluenceX, bInfluenceY und bInfluenceZ.
WaveY und WaveZ wirken entsprechend in den anderen Achsenrichtungen.

Aufgabe zu Waving

Übung „Wellenkörper“

Erzeugen Sie folgende Körper:



Körper A Körper B Körper C

Freiwillige Übung für die schnellen Nerds:
Texturieren Sie die Körper sinnfällig!

Kurzlösung „Wellenkörper“ (1/3)

Körper A (wichtigste Veränderungen)

1
2
3
4
5

```
…
m_zgTube.InitStraight(0.1F,0.15F, 1.0F, &m_zm,
240);
…
m_ptrianglelist =   m_zgTube.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->
       WaveY(0.5F,0.5F,0.0F, true, false, true);
…
```

Kurzlösung „Wellenkörper“ (2/3)

Körper B (wichtigste Veränderungen)

1
2
3
4
5

```
…
m_zgSphere.Init(0.5F,&m_zm, 140,120);
…
m_ptrianglelist =
       m_zgSphere.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->
       RippleY(0.1F,5,0.0F, true, false, true);
m_ptrianglelist->
       WaveY(0.5F,0.2,0.0F, true, true, true);
…
```

Kurzlösung „Wellenkörper“ (3/3)

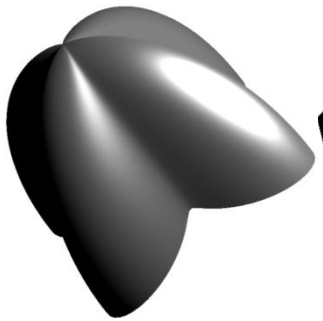
Körper C (wichtigste Veränderungen)

1
2
3
4
5

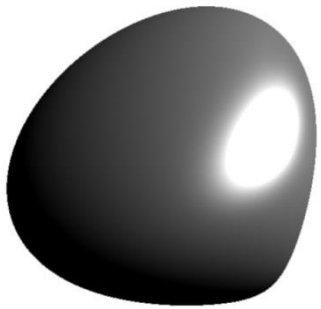
```
…
m_zgTube.InitStraight(0.2F,0.25F, 1.0F,
       &m_zm, 240);
…
m_ptrianglelist = m_zgTube.CopyToTriangleList();
m_ptrianglelist->SubdivideY(0.05F);
…
m_ptrianglelist->
       WaveY(0.4F,1.0F,0.2F, true, true, true);
m_ptrianglelist->TaperY(-0.7F,true, false, true);
m_ptrianglelist->RippleY(0.3F,3,0.0F);
…
```

Übung „Barr-Körper“ (ohne Lösung)

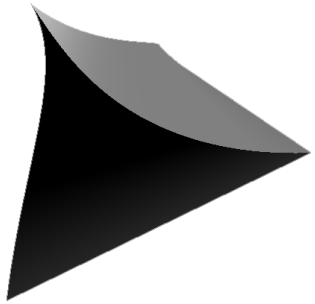
Erzeugen Sie folgende Körper:


Körper A

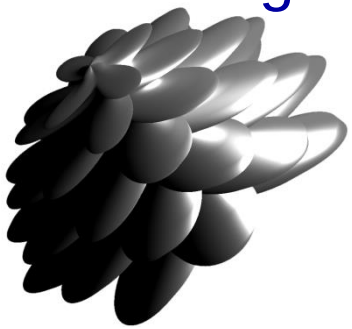

Körper B

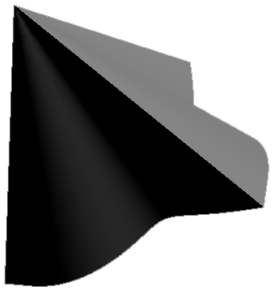

Körper C


Körper D


Körper E

Freiwillige Übung für die schnellen Nerds:


Körper F


Körper G


Körper H


Körper I


Körper J

GAME OVER

Danke für Ihr Interesse!