

Virtuelle Realität

X3D / VRML



G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de



X3D / VRML



- Was ist X3D/VRML?
 - Scenegraph & File-Format, plus ...
 - Multimedia-Support
 - Hyperlinks
 - Verhalten und Animationen
- Achtung: VRML \neq VR !
- Varianten:
 - VRML 1.0 (1995) (= Inventor, also kein VRML)
 - VRML 2.0 (1996)
 - VRML97 (1997) – ISO Standard, praktisch identisch zu VRML2
 - X3D (2003): ISO Standard, im wesentlichen andere Syntax, nämlich XML



Vorteile von X3D

- Die Spezifikation von VRML ist an einigen Stellen nicht eindeutig
 - In X3D präzisiert
- X3D hat 100+ Knoten (aufgeteilt in Components / Profiles)
 - VRML hat nur 54 Knoten
- X3D hat 3 verschiedene sog. "File Encodings":
 - **Classic**: sieht aus wie VRML; Suffix = **.wrl** oder **.x3dv**
 - Jede Software, die X3D lesen kann, kann (im Prinzip) auch VRML lesen
 - **XML**; Suffix = **.x3d**
 - das ist das Format, das man i.A. unter "X3D" versteht
 - **Binary** (XML braucht sehr viel Platz); Suffix = **.x3db**



X3D-"Browser"

- InstantReality:
 - <http://www.instantreality.org/>
 - Läuft auf allen 3 Plattformen
 - Implementiert (angeblich) V3.1 von X3D komplett
- FreeWrl:
 - <http://freewrl.sourceforge.net/>
 - Läuft auf Linux & Mac OS X
 - Implementiert das Subset (Profile) "Interchange" von X3D
- Cortona:
 - <http://www.parallelgraphics.com/products/cortona/>
 - Nur als Plugin für Web-Browser
- ...

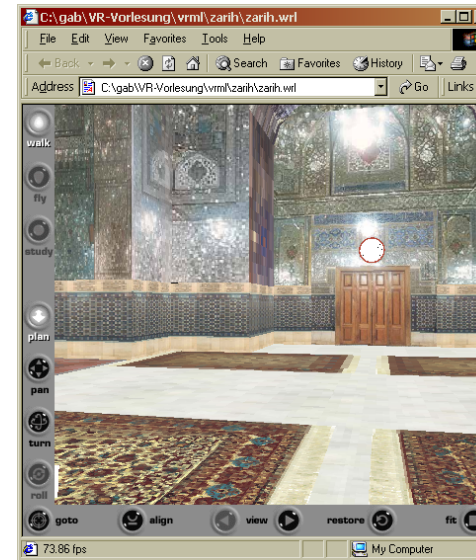




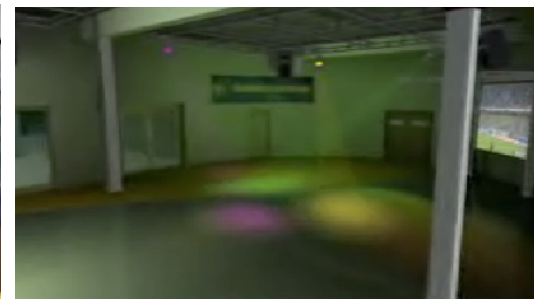
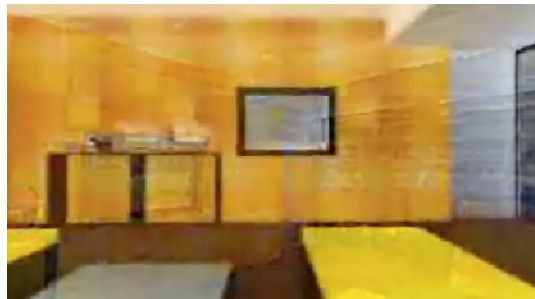
Beispiele



VRMLCity



Zarih



HSV-Arena (http://www.hsv-hshnordbank-arena.de/de/die_arena/die_arena_in_3d.html)
(leider nur mit proprietärem Plugin)



Literatur, References

- Bücher:
 - Don Brutzman, Leonard Daly:
X3D: Extensible 3D Graphics or Web Authors. Morgan Kaufman, 2007.
 - Andrea L. Ames, David R. Nadeau, and John L. Moreland:
The VRML 2.0 Sourcebook. John Wiley & Sons, 1996.
 - Hartman, Jed, and Wernecke:
The VRML 2.0 Handbook. Addison-Wesley, 1996.
- Online: Auf der Homepage zur Vorlesung
 - The Annotated VRML97 Reference
 - Der X3D-Standard: Knoten, Javascript, Java
- Die online Doku zu InstantReality:
 - Tutorials
 - Übersicht aller Knoten



Web-Seiten

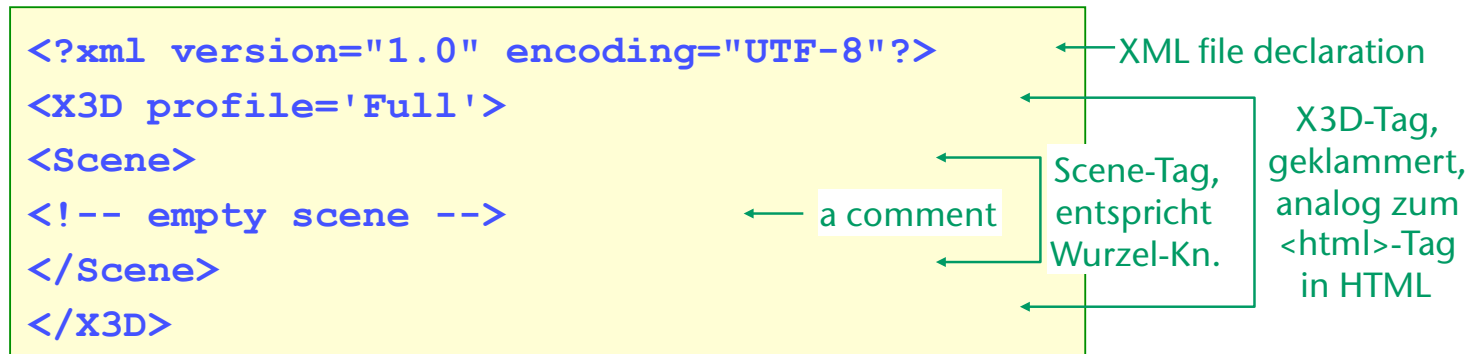
- Die Web-Seite zum X3D-Buch:
www.x3dgraphics.com
mit Bspielen, Tools, ...
- Eine "Meta"-Seite beim Web3D-Konsortium:
www.web3d.org/x3d/content/examples/X3dResources.html
mit Links zu Software für Viewer, Konverter, Authoring-Tools,
Plugins, Beispielen, Büchern, etc.



Encodings am Bsp. der trivialen X3D-Szene



- Als X3D-Encoding:



- Als ClassicVRML-Encoding:

```
#X3D V3.1 utf8
PROFILE Full
# empty scene
```

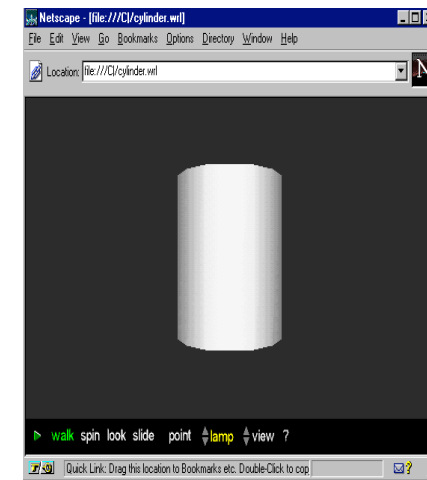
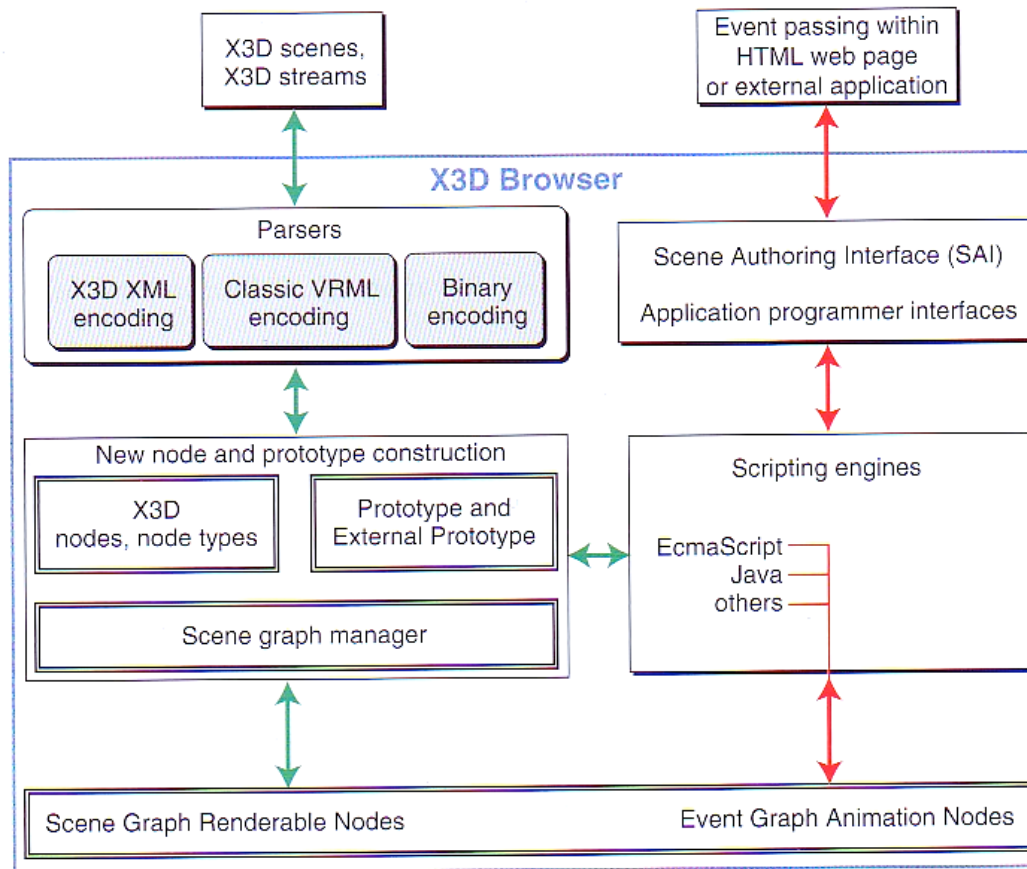
- und als VRML97:

```
#VRML V2.0 utf8
# empty scene
```

- Der Wurzel-Knoten für die gesamte Szene ist in VRML implizit!
- Keine Profiles und viel weniger Knoten in VRML97



- Der X3D-"Browser" (stand-alone oder embedded):





- Gründe für das XML-Encoding:
 - Ähnlichkeit zu HTML (Tags und Attribute: `<tag attr="val">...</tag>`)
 - XML ist ASCII (wie VRML97), also im Prinzip "human readable" (im Gegensatz zu binären Formaten)
 - XML ist ein weit verbreiteter Standard zur Beschreibung von Daten
 - XML ist eine Familie von Technologien: CSS, XSLT, Xpointer, ...
 - XML ist Lizenz-frei
- Gründe für das ClassicVRML-Encoding:
 - Legacy-Daten ("Altlasten")
 - Für Menschen leichter zu lesen und zu schreiben



Hello World

- In X3D (genauer: XML-Encoding):

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive'>
  <Scene>
    <Shape>
      <Text string="Hello" "world!" />
    </Shape>
  </Scene>
</X3D>
```



- In VRML:

```
#X3D V3.1 utf8
Shape {
  geometry Text {
    string [ "Hello" "world!" ]
  }
}
```

Tip: ASCII-Editor verwenden, der matching brackets erkennt und als Texteinheit behandeln kann



Knoten und Felder



- Knoten dienen zur Beschreibung ...
 - ... des **Szenengraphen** (die üblichen Verdächtigen):
 - Geometry, Transform, Group, Lights, LODs, ...
 - ... des **Verhaltensgraphen** (*behavior graph*), d.h., des Verhaltens der Objekte und bei User-Input
- Knoten = Menge von Feldern
 - "Single-valued Fields" und "Multiple-valued Fields"
 - Jedes Feld eines Knotens hat einen eindeutigen Namen
 - Diese Namen sind per Spezifikation vordefiniert
- Feldarten:
 - **field** = Daten im File
 - **eventIn**, **eventOut** = s.u., werden nicht gespeichert
 - **exposedField** = Kombination der drei (xxx, set_xxx, xxx_changed)



Feld-Typen

- Alle Feldtypen gibt es als "single valued"- (**SF**...) und als "multiple valued"-Variante (**MF**...)
- Beispiel für ein SFField:

```
<Material diffuseColor="0.1 0.5 1" />
```

X3D

```
material Material {  
    diffuseColor 0.1 0.5 1  
}
```

VRML

- MFField's sind im Prinzip nichts anderes als **Arrays**
 - Falls der **Grundtyp** ein **Tuple** ist (z.B. Farbe oder Vektor), sollte man in X3D die einzelnen Elemente mit Komma trennen. Beispiel:

```
"1 0 0, 0 1 0, 0 0 1"
```

- In VRML müssen MFField's mit [] geschrieben werden. Beispiel:

```
[ 1 0 0, 0 1 0, 0 0 1 ]
```



- Grundtypen: die üblichen Verdächtigen:

Field type	X3D example	VRML example
SFBool	true / false	TRUE / FALSE
SFInt32	12	-17
SFFloat	1.2	-1.7
SFDouble	3.1415926535	
SFString	"hello"	"world"

Erinnerung:
zu jedem
SF-Feld
gibt es ein
MF-Feld

- Etwas höhere Datentypen:

Field type	Beispiel
SFColor	0 0.5 1.0
SFColorRGBA	0 0.5 1.0 0.75
SFVec3f	1.2 3.4 5.6
SFMatrix3f	1 0 0 0 1 0 0 0 1
SFString	"hello"

Anmerkungen:

- Die Werte in SFColor müssen in [0,1] liegen
- Analog gibt es die Varianten ***2f**, ***3f** und ***4f**.



- Spezielle Feld-Typen:

Field type	X3D example	VRML example
SFImage	enthält spezielle Pixel-Encodings	
SFNode	<code><Shape> ... </Shape></code>	<code>Shape { ... }</code>
MFNode	<code><Shape>... , <Group>...</code> oder <code><Transform>...</code>	<code>Transform {</code> <code> children [...] }</code>
SFRotation	<code>0 1 0 3.1415</code>	
SFTime	<code>0</code>	



- Anmerkungen zu **SFImage**:

- Der Wert des Feldes ist eine Folge von Zahlen: Breite, Höhe, Anzahl Komponenten pro Pixel, Pixel, Pixel, ...
- Pro Kanal Werte im Bereich [0,255]
- Beispiel (bei 3 Komponenten): **0xFF0000** = Rot, **0x00FF00** = Grün, ...
- Beispiel für ein vollständiges **SFImage**:

```
2 4 3 0xFF0000 0xFF00 0 0 0 0 0xFFFFFFFF 0xFFFF00  
# w·h·c   red   green black..  white  yellow
```

- **SFImage** ist nur für sehr kleine Texturen gedacht und kommt nur im Knoten **PixelTexture** vor
 - Hintergrund: man wollte eine Möglichkeit haben, Texturen algorithmisch zu erzeugen (mittels Java)
- Für große ("richtige") Texturen verwende man PNGs oder JPGs und den Knoten **ImageTexture**



- Generelle Anmerkungen zum Design:
 - Das Design ist insofern **orthogonal**, als es zu jedem **SF**-Typ einen **MF**-Typ gibt
 - Das Design ist insofern **nicht orthogonal**, als manche Typen generisch sind (z.B. **SFBool**, **SFVec3f**), andere wiederum eine festgelegte Semantik haben (z.B. **SFColor**, **SFTime**, etc.)
 - Es ist nicht ganz klar, ob dies schlecht ist ...



Die Spezifikation der Knoten

- Knoten werden definiert durch ihre Felder und deren Bedeutung
- Die Syntax zur Definition von Knoten (vorerst):

```
Name_of_Node_Class {  
    type_of_field name_of_field_1 default_value  
    type_of_field name_of_field_2 default_value  
    ...  
}
```

- Bemerkungen:
 - Die Defaults werde ich im Folgenden meist weglassen
 - Auch werde ich nicht alle Felder aufzählen, nur die wichtigsten
 - Im folgenden werden nur einige wenige Knoten besprochen
- Fazit: schauen Sie in die Doku und das Tutorial!



- Beispiel:

```
Cone {  
    SFFloat  bottomRadius 1  
    SFFloat  height      2  
    SFFloat  side        TRUE  
    SFFloat  bottom      TRUE  
}
```

- Verwendung:

```
Cone { bottomRadius 1 height 2 }
```

VRML-Syntax

```
<Cone bottomRadius="1" height="2" />
```

XML-Syntax

- Bemerkung: Cone ist in XML-Syntax ein sog. **Singleton-Element**, d.h., es gibt kein öffnendes/schließendes Tag-Paar!



Knoten zur Beschreibung des graph. Szenengraphen

- Alle Geometrie-Knoten müssen Kind eines **Shape**-Knotens sein
- Definition:

```
Shape {  
    SFNode geometry NULL  
    SFNode appearance NULL  
}
```

- Achtung:
 - Das Feld **geometry** darf nur Geometrie-Knoten enthalten (es gibt etliche Klassen von Geometrie-Knoten)
 - Das Feld **appearance** darf nur einen Appearance-Knoten enthalten (es gibt nur eine Klasse von Appearance-Knoten)
- **Shape**-Knoten dienen dazu, Geometrie mit einer **Appearance** zu verknüpfen



- Der Appearance-Knoten dient zur Spezifikation des Aussehens einer Geometrie
- Definition:

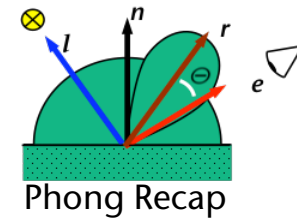
```
Appearance {  
    SFNode  material      NULL  
    SFNode  texture      NULL  
    SFNode  fillProperties NULL  
    ...  
}
```

- Auch hier gilt wieder: die Werte eines Feldes (hier: Instanzen einer Knotenklasse) müssen vom "richtigen" Typ (d.h., der richtigen Klasse) sein



- Der Material-Knoten:

```
Material {  
  SFFloat ambientIntensity 0.2  
  SFColor diffuseColor      0.8 0.8 0.8  
  SFColor emissiveColor     0 0 0  
  SFColor specularColor     0 0 0  
  SFFloat shininess         0.2  
  SFFloat transparency      0  
}
```

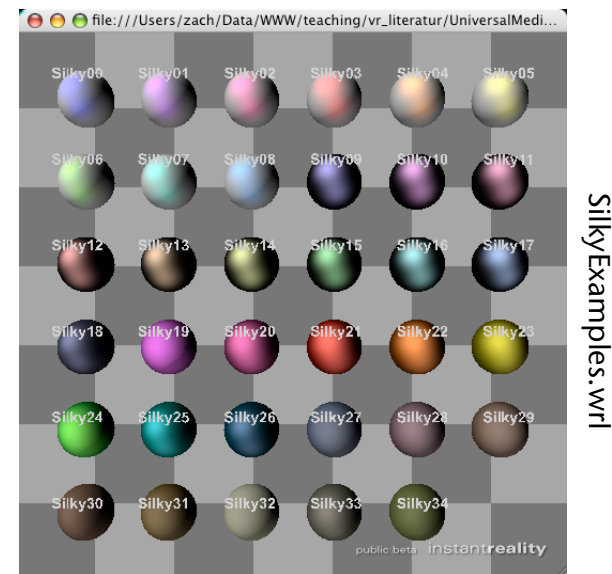
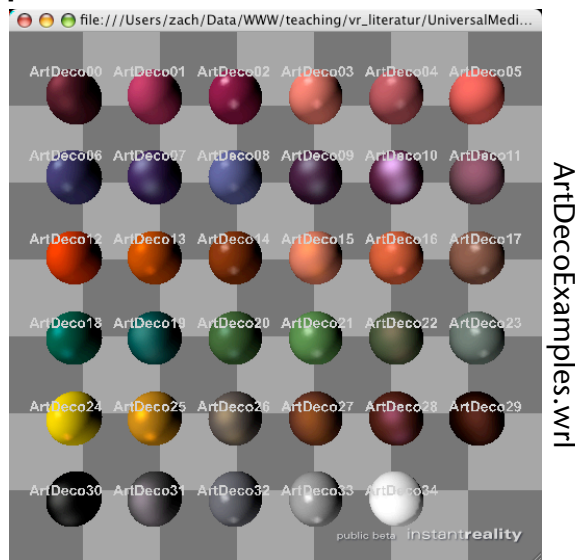


- Der Textur-Knoten:

```
ImageTexture {  
  MFString url      [ ]  
  SFBool repeats   TRUE  
  SFBool repeatT   TRUE  
}
```



- Auf der Homepage der Vorlesung finden Sie unter "*Online Literatur und Resources im Internet*" ein großes Archiv mit Materialien
- Beispiele:

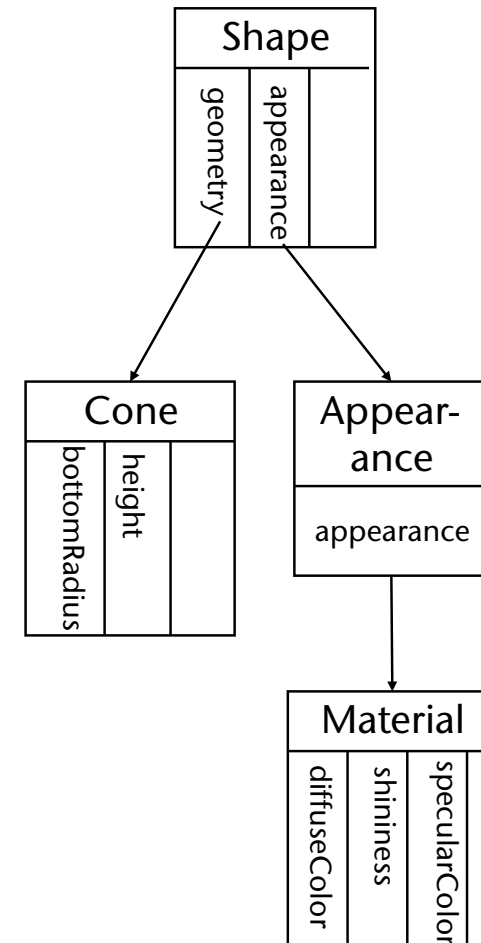


- Weitere Resource: ein Material-Editor (in Java)
<http://tog.acm.org/resources/applets/vrml/pellucid.html>



Ein erstes Beispiel

```
#X3D V3.1 utf8
Shape {
  geometry Cone {
    bottomRadius 1
    height      2
  }
  appearance Appearance {
    material Material {
      ambientIntensity 0.256
      diffuseColor     0.029 0.026 0.027
      shininess        0.061
      specularColor    0.964 0.642 0.980
    }
  }
}
```





Geometrie-Knoten für Terrain

- Allgemein für (diskrete) Flächen, die sich als Funktion über einer Ebene beschreiben lassen
- Definition:

```
ElevationGrid {  
    SFBool  normalPerVertex  TRUE  
    SFFloat creaseAngle      0.0  
    MFFloat height          []  
    SFInt32 xDimension       0  
    SFFloat xSpacing         1.0  
    SFInt32 zDimension       0  
    SFFloat zSpacing         1.0  
    SFBool  solid            TRUE  
}
```

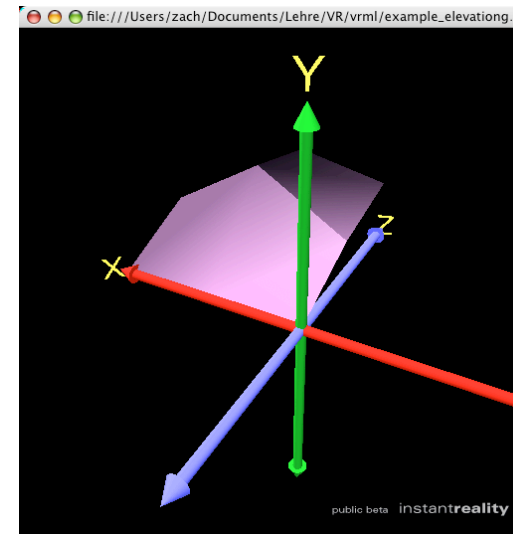


- Bedeutung der Felder:
 - **normalPerVertex** schaltet Beleuchtung pro Vertex mit Gouraud-Shading ein (die Normalen werden i.A. vom Browser berechnet)
 - **solid = TRUE** schaltet Backface-Culling ein
 - Tip: bei Terrain ausschalten
 - Alle Winkel zwischen 2 Polygonen über eine Kante hinweg (*dihedral angle*), die größer als **creaseAngle** sind, werden erhalten, d.h., die beteiligten Vertices werden für die beiden Polygone mit jew. einer eigenen Normale gerendert
- Anmerkungen:
 - Aus Matlab kann man Plots als ein solches VRML-ElevationGrid exportieren
 - Achtung: die Vierecke sind i.A. nicht planar → Flackern und andere Artefakte!



- Beispiel:

```
Shape {  
  appearance Appearance { ... }  
  geometry ElevationGrid {  
    height [  
      0.0 0.0 0.0  
      0.2 0.5 0.2  
      0.3 0.4 0.1 ]  
    xDimension 3  
    zDimension 3  
    xSpacing 0.5  
    zSpacing 0.5  
    solid false  
    #creaseAngle 1.5  
  }  
}
```



[example_elevationgrid.wrl](#)



Dreiecke

- Die allgemeinste Geometrie
- Für Dreiecke (und Vierecke) gibt es viele Varianten; hier nur 2
- Die einfachste Variante: **TriangleSet**
- Definition:

```
TriangleSet {  
  SFNode   coord      NULL  
  SFBool   ccw        TRUE  
  SFBool   normalPerVertex TRUE  
  SFBool   solid      TRUE  
  SFFloat  creaseAngle 0.0  
}
```

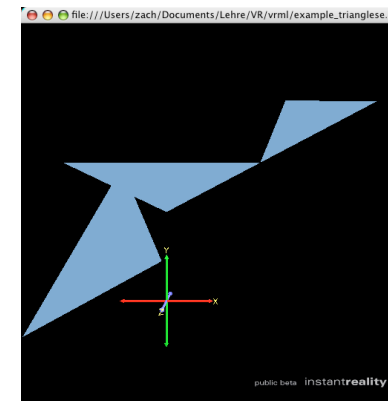
```
Coordinate {  
  MFVec3f point []  
}
```

- **coord** → **point** ist eine Liste von Koordinaten; je 3 aufeinanderfolgende ergeben einen Vertex; davon je 3 aufeinanderfolgende ergeben ein Dreieck
- **ccw** (*counter-clockwise*) gibt an, ob die Vertices im Uhrzeigersinn vorliegen oder nicht



■ Beispiel:

```
Shape {  
  appearance Appearance { ... }  
  geometry TriangleSet {  
    coord Coordinate {  
      point [ -2 0 3, -0 1 1, -1 3 0,  
              0 2 0, 2 3 1, -2 3 1,  
              3 5 -2, 2 3 1, 4 4 2 ]  
    }  
    solid FALSE  
    ccw TRUE  
  }  
}
```



[example_triangleset.wrl](#)

■ Bemerkung:

- das Komma ist in X3D/VRML ein *Whitespace*
- könnte man also weglassen; sollte man bei hand-geschriebenen Szenen aber nicht

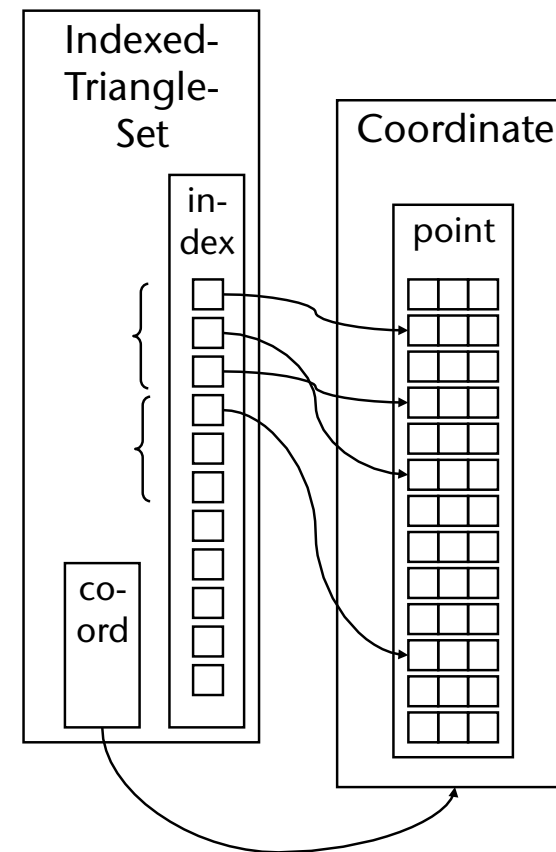


- Ein häufig vorkommender Knoten ist **IndexedTriangleSet**:

```
IndexedTriangleSet {  
  SFNode coord NULL  
  MFInt32 index []  
  SFBool ccw TRUE  
  SFBool normalPerVertex TRUE  
  SFBool solid TRUE  
  SFFloat creaseAngle 0.0  
}
```

```
Coordinate {  
  MFVec3f point []  
}
```

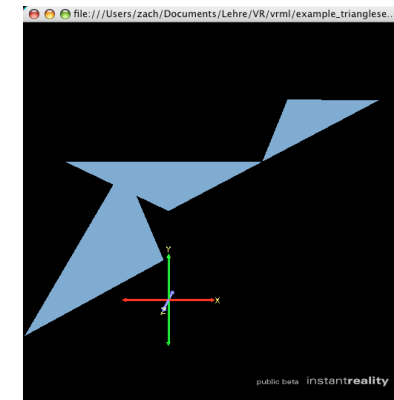
- Großer Vorteil: Speicher-Einsparung
- Denn: bei "normalen" Dreiecks-Meshes wird jeder Vertex im Schnitt von 6 Dreiecken "benutzt"





- Dasselbe Beispiel nochmal, diesmal mit **IndexedTriangleSet**:

```
Shape {  
  appearance Appearance { ... }  
  geometry IndexedTriangleSet {  
    index [ 0 1 2, 3 4 5, 6 4 7 ]  
    coord Coordinate {  
      point [ -2 0 3, -0 1 1, -1 3 0,  
              0 2 0, 2 3 1, -2 3 1,  
              3 5 -2, 4 4 2 ]  
    }  
    solid FALSE  
    ccw TRUE  
  }  
}
```



[example_indexedtrianglese.wrl](#)

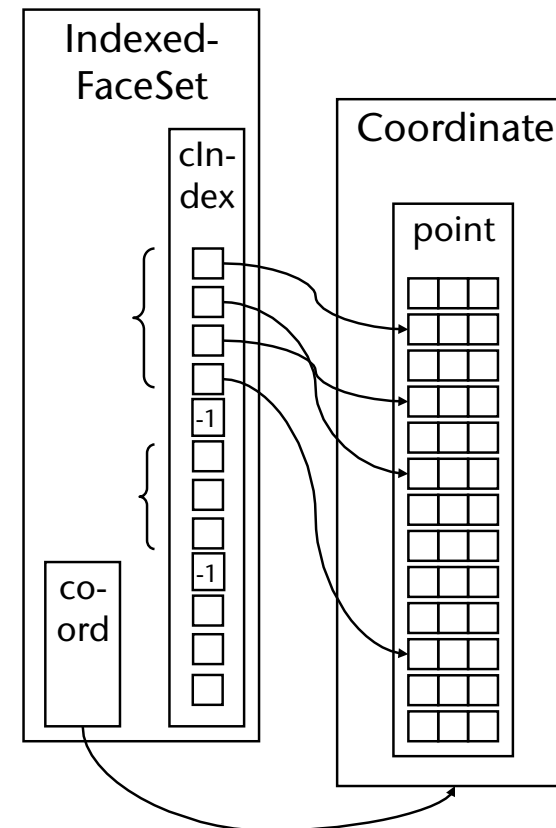


- Der häufigste Knoten-Typ ist (unnötigerweise) das **IndexedFaceSet**:

```
IndexedFaceSet {  
  SFNode coord NULL  
  MFInt32 coordIndex []  
  SFBool ccw TRUE  
  SFBool normalPerVertex TRUE  
  SFBool solid TRUE  
  SFFloat creaseAngle 0.0  
}
```

```
Coordinate {  
  MFVec3f point []  
}
```

- Unterschied zu **IndexedTriangleSet**: die -1 als "*Sentinel*"





- Vorteil: beliebige Polygone
- Anmerkung: viele Exporter exportieren **IndexedFaceSet** obwohl alle Pgone Dreiecke sind → Speicherverschwendung & langsames Rendering!
- Das Beispiel von vorhin nochmal als **IndexedFaceSet**:

```
Shape {  
  appearance Appearance { ... }  
  geometry IndexedTriangleSet {  
    coordIndex [ 0 1 2 -1 3 4 5 -1 6 4 7 -1 ]  
    coord Coordinate {  
      point [ -2 0 3, -0 1 1, -1 3 0,  
              0 2 0, 2 3 1, -2 3 1,  
              3 5 -2, 4 4 2 ]  
    }  
    solid FALSE  
    ccw TRUE  
  }  
}
```



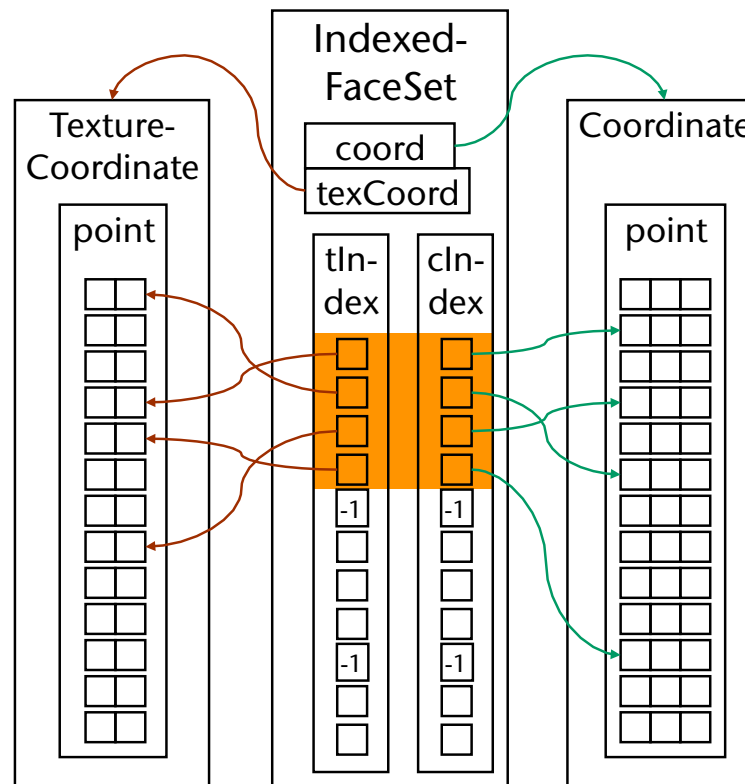
Spezifikation weiterer Attribute pro Vertex

- In allen Geometrie-Knoten kann man weitere Vertex-Attribute spezifizieren, z.B., Normalen oder Texturkoord. pro Vertex
- Hier am Beispiel Texturkoord. im **IndexedFaceSet**:

```
IndexedFaceSet {  
  SFNode coord  
  MFInt32 coordIndex  
  SFNode texCoord  
  MFInt32 texCoordIndex  
  SFBool ccw  
  SFBool normalPerVertex  
  SFBool solid  
}
```

```
TextureCoordinate {  
  MFVec2f point []  
}
```

```
Coordinate {  
  MFVec3f point []  
}
```





Weitere Geometrie-Knoten

- Es gibt noch viele weitere:

PointSet, LineSet, QuadSet, ...

IndexedLineSet, IndexedQuadSet, ...

TriangleStripSet, IndexedTriangleStripSet, ...

Box, Sphere, Cylinder, ...

Text, Extrusion, ...

- Viele 2D-Knoten, z.B.: **Arc2D, Polyline2D, ...**
- CAD-Knoten: **CADAssembly, NurbsPatchSurface, ...**