Im Kurs können nicht alle Einzelheiten behandelt werden.
Deshalb der Hinweis auf die umfassende VRML97-Spezifikation im Internet:
http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm

Definition eigener Knotentypen:
Prototypen

Schlüsselwort "PROTO"
2-teilige Definition:

- Deklaration der Felder, (z.T.) öffentlich
- Definition, privat (gekapselt) – eigentliche Objektbeschreibung
  – darin Bezugnahme auf die neuen Felder mit Schlüsselwort "IS"

umfasst einzelnen Knoten oder ganze "Mini-Szene"

- Parametrisierung über den öffentlichen Deklarationsteil
- PROTO-Definition allein erzeugt noch kein neues Objekt!
- Verwendung in Szenen genauso wie die vorgegebenen Knotentypen

Jede Instanz eines Prototyps ist unabhängig von allen anderen,
Änderungen einer Instanz betreffen nur sie selbst
(– Gegensatz zu DEF / USE !)

- der erste (oberste) Knoten in der Prototyp-Definition vererbt den Knotentyp (Verwendungszweck) an den Prototyp
- Gültigkeitsbereich von Namen innerhalb einer Prototyp-Def. ist auf diese Def. beschränkt

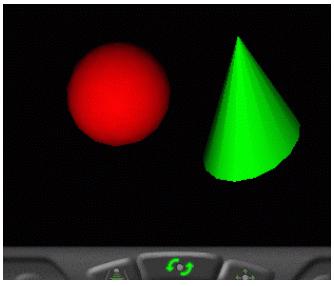
Beispiel 1:
Def. eines neuen Knotentyps für geometrische Objekte, um Position, Farbe etc. komfortabler angeben zu können
der neue Knotentyp soll "Koerper" heißen

```
#VRML V2.0 utf8

PROTO Koerper
   [            # Deklaration
   exposedField SFVec3f verschiebe 0 0 0
   exposedField SFVec3f verzerre 1 1 1
   exposedField SFColor farbe 0.8 0.8 0.8
   exposedField MFString adressen [ ]
   exposedField SFNode geometrie NULL
   ]
```

```
   {           # Definition
   Transform
      {
      translation IS verschiebe  # Schnittstelle
      scale IS verzerre          # nach außen!
      children
         [
         Shape
            {
            geometry IS geometrie
            appearance Appearance
               {
               material Material
                  { diffuseColor IS farbe }
               texture ImageTexture
                  { url IS adressen }
               }
            }
         ]
      }
   }
# Verwendung des Prototyps
Koerper  # rote Kugel
   {
   farbe 1 0 0
   verschiebe -2 0 0
   geometrie Sphere { }
   }
Koerper  # gruener Kegel, flachgedrueckt
   {
   farbe 0 1 0
   verschiebe 1 1 0
   verzerre 0.7 1 1
   geometrie Cone { }
   }
```

Ergebnis:



Beispiel 2:
Nicht nur ein einzelner Körper, sondern ein Arrangement (Mini-Szene) wird als Prototyp definiert – hier eine Sitzbank, anschließend werden mehrere Instanzen auf einer Bodenfläche verteilt

```
#VRML V2.0 utf8
# proto2.wrl: Drei Instanzen des Parkbank-Prototyps
PROTO Bank    # Prototyp einer Parkbank
   [
   exposedField SFColor SitzFarbe 1 0 0 # Deklaration
   exposedField SFColor BeinFarbe 0 1 0
   ]
   {
   Group                                  # Definition
      {
      children
         [
         Transform     # Sitz
            {
            translation 0 0.03 0
            children
               DEF Sitz Shape
                  {
                  appearance Appearance
                     {
                     material Material
                        { diffuseColor IS SitzFarbe }
                     }
                  geometry Box
                     { size 0.6 0.05 0.2 }
                  }
            }
         Transform     # Lehne
            {
            translation 0 0.2 -0.1
            rotation 1 0 0 1.4
            scale 1 1 0.8
            children USE Sitz # interne
                              # Wiederverwendung!
            }
```

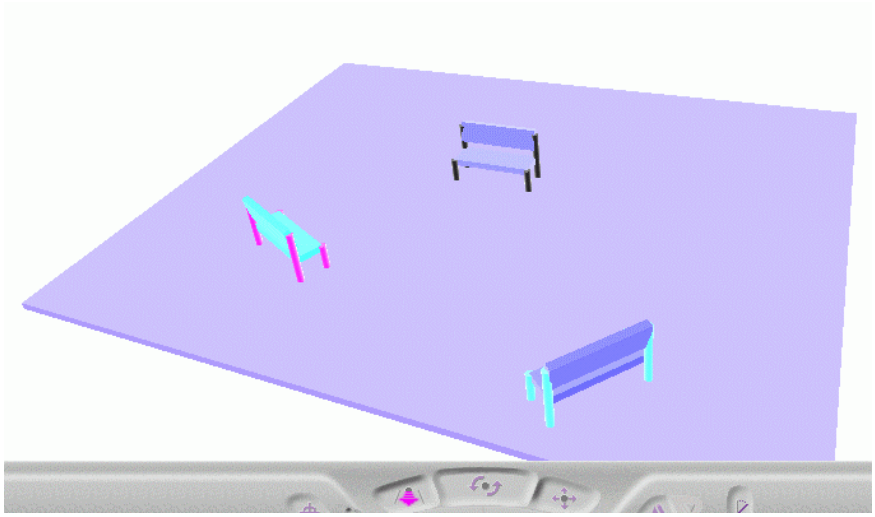
```
         Transform     # kurzes Bein
            {
            translation -0.3 -0.05 0.1
            children
               DEF Bein Shape
                  {
                  appearance Appearance
                     {
                     material Material
                        { diffuseColor IS BeinFarbe }
                     }
                  geometry Cylinder
                     { height 0.2 radius 0.025 }
                  }
            }
         Transform     # 2. kurzes Bein
            {
            translation 0.3 -0.05 0.1
            children USE Bein
            }
         Transform     # 1. langes Bein
            {
            translation -0.3 0.05 -0.1
            scale 1 2 1
            children USE Bein
            }
         Transform     # 2. langes Bein
            {
            translation 0.3 0.05 -0.1
            scale 1 2 1
            children USE Bein
            }
         ]
      }                # Group
   }                   # Prototyp Bank - Ende
DEF Szene Group
   {
   children
      [
      DEF Bank1 Transform   # Instanz 1
         {
         translation -1 -0.85 2
         rotation 0 1 0 2.7
         children Bank { }
         }
```

```
   DEF Bank2 Transform   # Instanz 2
      {
      translation 0 -0.85 0.7
      children Bank
         {
         SitzFarbe 0.5 0.5 0
         BeinFarbe 1 1 1
         }
      }
   DEF Bank3 Transform   # Instanz 3
      {
      translation 1.3 -0.85 2.5
      rotation 0 1 0 4
      children Bank
         {
         SitzFarbe 0.7 0.7 0
         BeinFarbe 0.7 0 0
         }
      }

   Transform    # Boden: Extra-Element (nicht aus
                # Prototyp)
      {
      translation 0 -1 1
      children
         Shape
            {
            appearance Appearance
               {
               material Material
                   { diffuseColor 0.4 0.5 0.01 }
               }
            geometry Box
               { size 5 0.05 4 }
            }
      }
   ]

}  # Szene (Group)
```

Ergebnis (hier wegen besserer Darstellbarkeit farblich invertiert):



Prototyp-Definitionen können auch in externe Dateien verlagert werden
(⇒ Szenengestaltung als Kombination und Variation vor-definierter Objektsammlungen aus Bibliotheken):

EXTERNPROTO-Definition

- Definitionsteil besteht nur noch aus URL-String
- Default: Zugriff auf die erste Prototyp-Definition in der referenzierten Datei
- Prototyp-Name kann an den Dateinamen mit "#" angehängt werden

Beispiel:

```
#VRML V2.0 utf8
# exproto1.wrl: Zugriff auf externen Prototyp
EXTERNPROTO Parkbank    # externer Prototyp
   [
   exposedField SFColor SitzFarbe   # Dekl.
   ]
   "pro_bank.wrl#Bank"  # Endung "#Bank" hier
                        # optional
```

```
DEF Szene Group
   {
   children
      [
      DEF Bank1 Transform   # Instanz 1
         {
         translation -2 -0.85 2.5
         rotation 0 1 0 2.7
         children Parkbank {}
         }
```
(... usw.)

Beachte: Die Namen der Prototypen in der Datei und in der EXTERNPROTO-Deklaration müssen nicht übereinstimmen.

Knoten, die Darstellungen in Abhängigkeit vom Benutzer-standpunkt ändern

1. *LOD (level of detail) - Knoten*

dient der Bereitstellung alternativer, verschieden genauer Darstellungen von Objekten für unterschiedliche Entfernungs-intervalle

LOD	**range**	MFFloat	Reichweite-Intervallgrenzen (in aufsteigender Folge) – eine weniger als die Zahl der Intervalle
	level	MFNode	Auflistung der alternativen Objekte

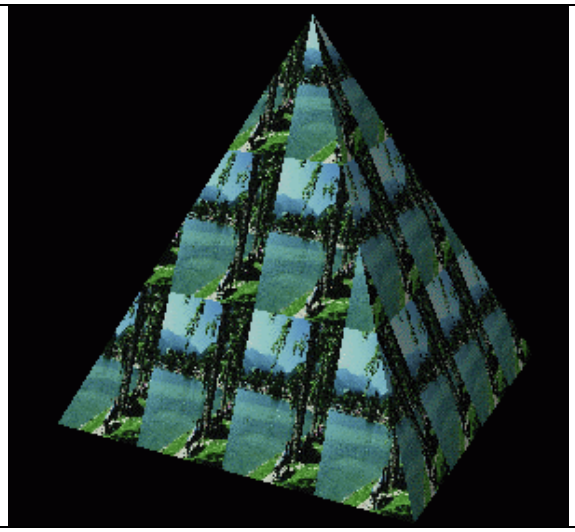
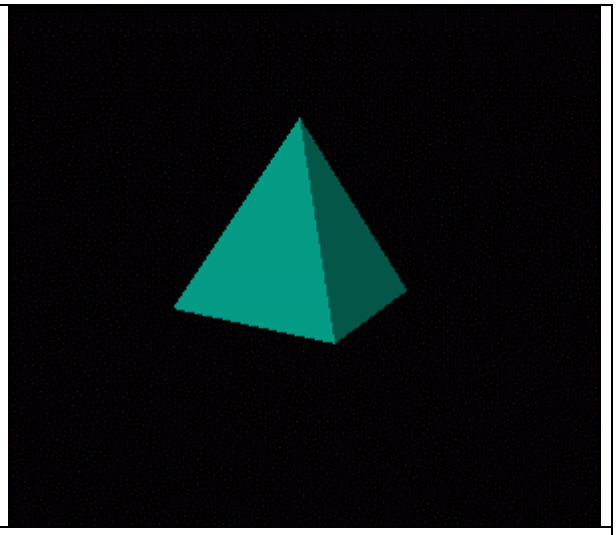
Beispiel: Eine Pyramide wird aus der Nähe mit genauem Texturmuster (aus einer Bilddatei) und aus mittlerem Abstand einfarbig dargestellt. In großem Abstand verschwindet sie ganz.

Das Beispiel zeigt außerdem die Verwendung von Texturkoordinaten zur systematischen Plazierung und Wiederholung von Texturen auf **`IndexedFaceSet`**-Flächen.

```
#VRML V2.0 utf8
PROTO Pyramide
   [
   exposedField SFNode app NULL
   ]
   {
   Shape
      {
      appearance IS app
      geometry IndexedFaceSet
         {
         coord Coordinate
            {
            point [ 1 0 0, 0 0 1, -1 0 0, 0 0 -1,
                    0 1.5 0 ]
            }
         coordIndex [ 0 1 2 3 -1,
                      1 0 4 -1,
                      2 1 4 -1,
                      3 2 4 -1,
                      0 3 4 ]
         texCoord TextureCoordinate
            {
            point [ -2 -2, 2 -2, -2 0, 2 0,
                    0 3 ]
            }
         texCoordIndex
            [ 0 1 2 3 -1,
              2 3 4 -1,
              2 3 4 -1,
              2 3 4 -1,
              2 3 4 ]
         }
      }
   }
LOD
   {
   range [ 7, 15 ]
   level
      [
      Pyramide # level 0: bis Abst. 7 vom Betrachter
         {
         app Appearance  # genauestes Modell
            {
            material Material { }
```

```
            texture ImageTexture
               { url "wiessee.jpg" }
            }
         },
      Pyramide  # level 1: zwischen Abstand 7 und 15
         {
         app Appearance  # groeberes Modell
            {
            material Material
               { diffuseColor 0 0.7 0.6 }
            }
         },
      Shape { } # level 2: Abstand > 15 (nichts mehr)
   ]
}
```

Ergebnis:

	
aus der Nähe	*aus mittlerem Abstand*

2. *Billboard-Knoten*

- Projektion eines 2D-Bildes auf ein total transparentes Objekt
- das Objekt weist dem Betrachter immer dieselbe Seite zu (Rotation um festgelegte Achse, oder unabhängig von jeglicher Ausrichtung: *Screen Alignment*)
- nützlich bei Objekten, die als komplettes 3D-Modell sehr viel Platz und Zeit erfordern würden und wo man "nicht genau hinsieht" (z.B. Bäume im Hintergrund)

Billboard	**axisOfRotation**	SFVec3f	Drehachse (Default: 0 1 0, 0 0 0 = Screen Alignment)
	children	MFNode	Kindknoten

Beispiel: 2 Bäume als Billboards (flache Tafeln), Verwendung eines Fotos "baum.gif"

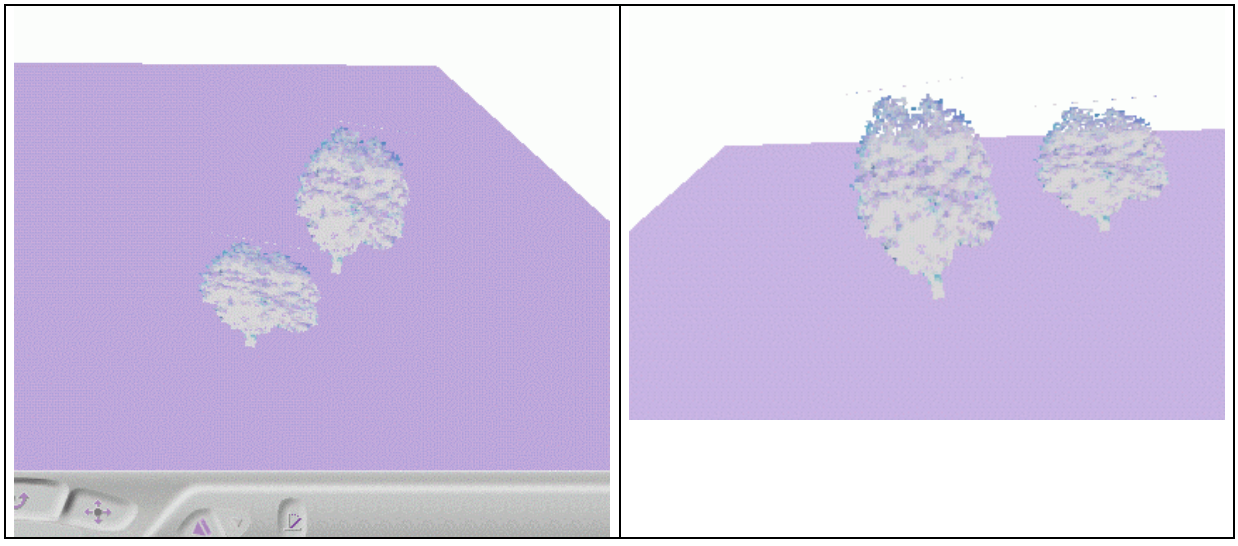
```
#VRML V2.0 utf8
# billboard.wrl: Baum als Billboard

Transform     # Baum
   {
   translation 3 -0.5 3
   children
      [
      DEF Baum Billboard   # Billboard-Anfang
         {
         axisOfRotation 0 1 0
         children
            Shape
                {
                appearance Appearance
                   {
                   texture ImageTexture
                      {
                      url "baum.gif"
                      repeatS FALSE
                      repeatT FALSE
                      }
                   }
                geometry Box
                    { size 1 1 0.01 }
                }
         }                  # Billboard-Ende
      ]
   }
```

```
Transform                    # Baum 2
   {
   translation 4 -0.25 2
   scale 1 1.5 1
   children USE Baum    # Instanziierung
   }

Transform                    # Boden
   {
   translation 0 -1 1
   children
       Shape
          {
          appearance Appearance
             {
             material Material
                { diffuseColor 0.6 0.8 0.3 }
             }
          geometry Box
             { size 14 0.01 8 }
          }
   }
```

Ergebnis aus 2 Ansichten (Farben wieder invertiert):



3. *Nebel*

- einstellbar: Farbe des Nebels, Stärke, Nebeltyp (exponentielle oder lineare Abnahme der Sichtbarkeit)
- auch für Dunst geeignet

Fog	**color** **fogType** **visibilityRange**	SFColor SFString SFFloat	Farbe "LINEAR", "EXPONENTIAL"

Als Beispiel wird eine Szene mit einer Kirche und einer Laterne in Nebel getaucht:

```
#VRML V2.0 utf8
# kirche.wrl

DEF Laterne Transform
   {
   translation -2.5 0 2.5
   children
      [
      PointLight {}  # Punktlichtquelle
      DEF Lampe Shape
         {
         appearance Appearance
            {
            material Material
               { emissiveColor 1 1 1 }
            }
         geometry Sphere
            { radius 0.2 }
         }
      DEF Mast Transform
         {
         translation 0 -0.55 0
         children
            Shape
               {
               appearance Appearance
                  {
                  material Material
                     {
                     ambientIntensity 1
```

```
                    diffuseColor 0.3 0.9 1
                    }
                }
             geometry Cylinder
                {
                radius 0.05
                height 0.9
                }
             }
          }
       ]
    }

DEF Kirche Transform
    {
    translation 0 0 0
    children
       [
       DEF Schiff Shape
          {
          appearance Appearance
             {
             texture ImageTexture
                {
                url "seite.jpg"
                repeatS FALSE  # Verzerren
                repeatT FALSE  # statt Wiederholen
                }
             }
          geometry Box
             { size 4 2 2 }
          }
       DEF Dach Transform
          {
          rotation 1 0 0 0.785  # 45-Grad-Drehung
          translation 0 1 0
          children
             Shape
                {
                appearance Appearance
                   {
                   texture ImageTexture
                      { url "dach.jpg" }
                   }
                geometry Box { size 3.95 1.75 1.75 }
                }
          }
```

```
      DEF Turm Transform
         {
         translation -1.7 2 0
         children
            Shape
               {
               appearance Appearance
                  {
                  texture ImageTexture
                     {
                     url "turm.jpg"
                     repeatS FALSE
                     repeatT FALSE
                     }
                  }
               geometry Box { size 1 6 1 }
               }
         }
      DEF Kuppel Transform
         {
         translation -1.7 6 0
         children
            Shape
               {
               appearance Appearance
                  {
                  texture ImageTexture
                     { url "dach.jpg" }
                  }
               geometry Cone
                  {
                  bottomRadius 0.6
                  height 2
                  }
               }
         }
      ]
   }
DEF Boden Transform
   {
   translation 0 -1 1
   children
      Shape
         {
         appearance Appearance
            {
            material Material
               { diffuseColor 0.5 0.6 0.01 }
```

```
            }
         geometry Box
            { size 14 0.01 8 }
         }
   }

Background
   {
   skyColor [ 0.8 0.8 0.9 ]
   }
Fog
   {
   color 0.8 0.8 0.8
   fogType "EXPONENTIAL"
   visibilityRange 15
   }
```

Ergebnis: