



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Vorlesung Computergrafik Virtual Reality Modeling Language VRML

P.A.Henning
media::lab
**Karlsruhe University of Applied
Sciences**

VRML

- VRML = Virtual Reality Modeling Language

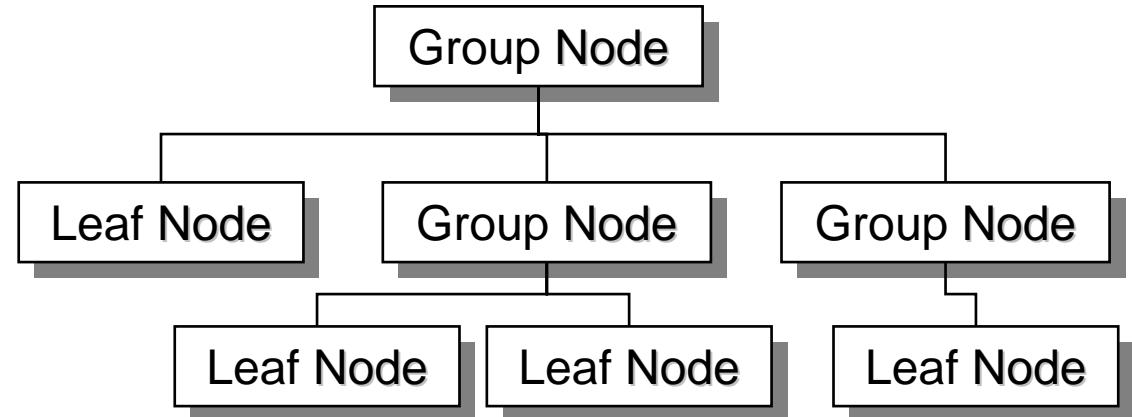
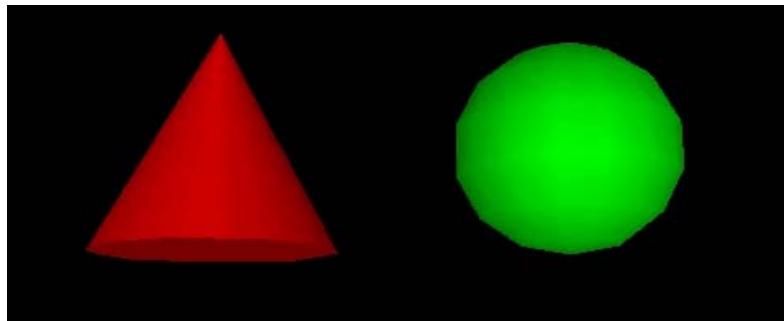


- Charakteristische Merkmale
 - einfache Sprache zur Beschreibung von 3D-Objekten
 - Daten im ASCII-Format, plattformunabhängig
 - der Benutzer, nicht der Computer, erforscht die 3D-Welt
 - eine VRML-Welt ist interaktiv
 - eine VRML-Welt vereint 2D- und 3D-Objekte, Animation und Multimedia-Effekte
- Geschwindigkeit: kann nicht mit proprietären VR-Lösungen konkurrieren
- Konformität: Szenen werden auf bestimmte Browser optimiert
- Prozedurale Sprachanbindung: Skriptknoten umständlich
 - EAI nicht standardisiert

VRML-Browser

- Ein VRML-Browser ist typischerweise als Plugin Komponente für den WWW-Browser angebunden
- Der VRML-Browser liest eine *.wrl*-Datei ein und parst diese
- Der VRML-Browser stellt eine Benutzerschnittstelle bereit
- Gängige VRML-Browser
 - Octagaplayer (www.octaga.com) !!!!
 - Cosmoplayer (SGI-Tochter CosmoSoftware)
 - WorldView (Intervista Inc.)
 - Blaxxun Contact (Insolvenz April 2002) ☹

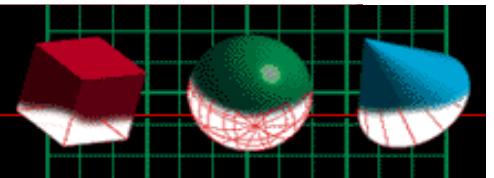
VRML Szenengraph



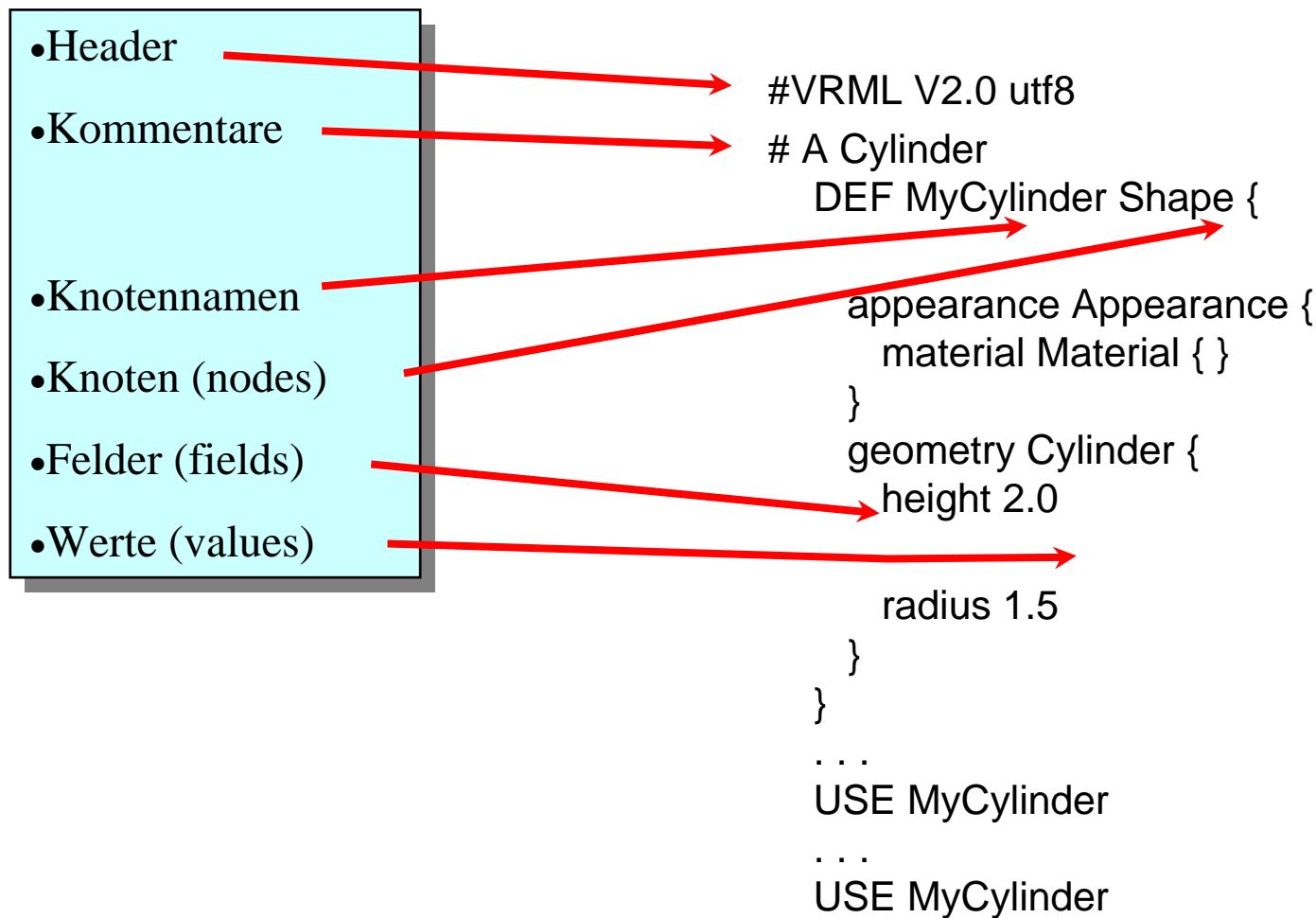
```
#VRML V2.0 utf8
Transform
{
    translation -3 0 0
    children
    [
        Shape
        {
            appearance Appearance
            {
                material Material { diffuseColor 0 1 0 }
            }
            geometry Sphere { radius 1}
        }
    ]
}
```

VRML-Java3D

The Working Group



VRML Dateistruktur



VRML(X3D) in XML-Syntax

```
<X3D ID="my3DViewer" transparency="true">
<TRANSFORM>
  <SHAPE>
    <GEOMETRY>
      <SPHERE DEF="mySphere" RADIUS="2" />
    </GEOMETRY>
  </SHAPE>
</TRANSFORM>
</X3D>

<SCRIPT ID="myScript1" type="text/javascript">
setCameraPos() {
  my3DViewer.document.camera.position = "10 10 -10";
  my3DViewer.document.all[ mySphere ].setValue("color","1 0 1");
}
</SCRIPT>
```



Knoten in VRML

In VRML gibt es 54 verschiedene Knotentypen, und die Möglichkeit eigene Knotentypen zu definieren.

- **Shapes and geometry:** Box, Cone, Coordinate, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, Normal, PointSet, Shape, Sphere, Text
- **Appearance:** Appearance, Color, FontStyle, ImageTexture, Material, MovieTexture, PixelTexture, TextureCoordinate, TextureTransform
- **Grouping:** Anchor, Billboard, Collision, Group, Inline, LOD, Switch, Transform

Knoten in VRML II

- **Animation:** ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator, TimeSensor
- **Interaction:** CylinderSensor, PlaneSensor, ProximitySensor, SphereSensor, TouchSensor, VisibilitySensor
- **Environment:** AudioClip, Background, DirectionalLight, Fog, PointLight, Sound, SpotLight
- **Viewing:** NavigationInfo, Viewpoint
- **Miscellaneous:** Script, WorldInfo

VRML Nodes

- Elemente eines Knoten
 - eindeutiger Name
 - Datentyp
 - Felder (fields)
 - Ereignisse (events)
- "group nodes" haben ein oder mehrere Knoten als Kinder
 - Anchor, Billboard, Collision, Group, Transform
- "bindable nodes" enthalten Knoten für verschiedenartige Formen
 - Background, Fog, NavigationInfo, Viewpoint
- "script node" enthält Java-Script zum Steuern von Ereignissen in der VRML-Welt

VRML Datentypen

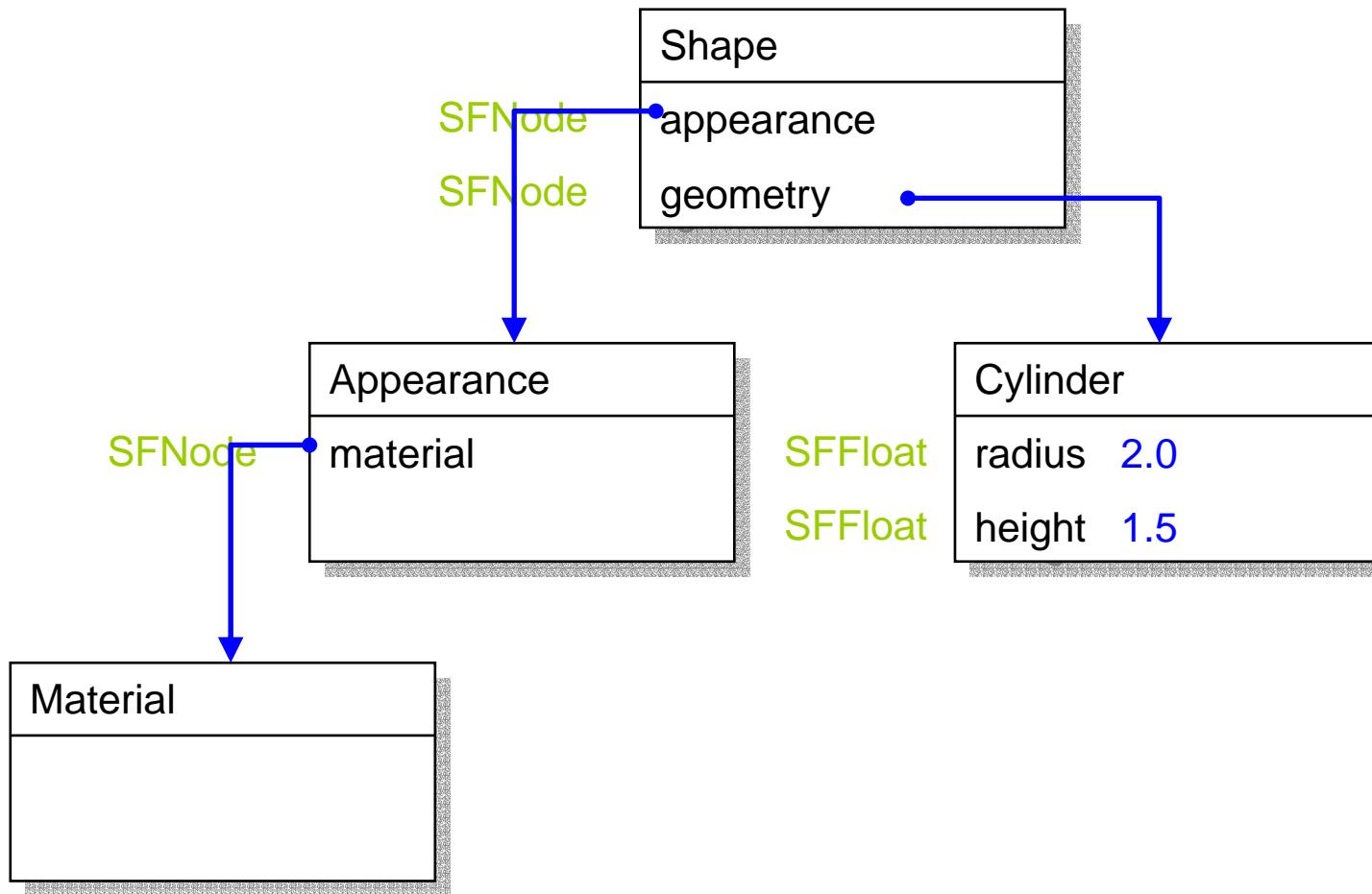
Ein Knoten kann 0 oder mehr Felder haben. Jedes Feld besitzt

- Typ
- Name
- Default-Wert

Typen sind

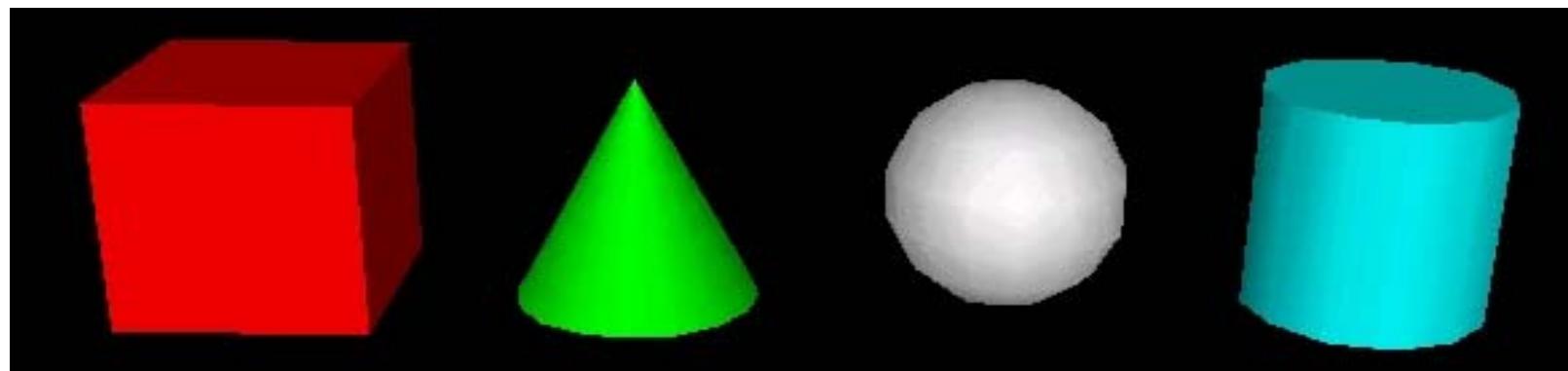
- SFNode / MFNode = Single Field Node / Multiple Field Node, hier gehört ein/mehrere VRML-Knoten hin.
- SFBool
- SFColor/MFColor = Drei Gleitkommawerte zwischen 0.0 und 1.0, ggf. mehrfach
- SFFloat/MFFloat
- SFImage
- SFInt32/MFInt32
- SFRotation/MFRotation
- SFString/MFString, SFTime/MFTime
- SFVec2f/MFVec2f, SFVec3f/MFVec3f

Szenengraf



VRML-Primitive I

- Grafikprimitive
 - Box
 - Cone
 - Sphere
 - Cylinder



VRML Primitive II

- Sphäre (Sphere)
 - einfachste geometrische Form
 - das Innere ist unsichtbar
 - Syntax:

```
Sphere {  
    radius 1  
}
```
- Kegel (Cone)
 - Zwei Teile: Seiten und Boden
 - beide Teile sichtbar oder unsichtbar (TRUE, FALSE)
 - Syntax:

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

VRML Primitive III

- Zylinder (Cylinder)
 - drei Teile: Seite, Boden und Deckelfläche
 - Teile sichtbar oder unsichtbar (TRUE, FALSE)

```
Cylinder {  
    bottom TRUE  
    side TRUE  
    top TRUE  
    height 2  
    radius 1  
}
```

- Quader (Box)
 - ersetzt "cube" in VRML 1.0
 - "size"-Feld für Höhe, Breite und Tiefe des Quaders
 - kann nicht von Innen betrachtet werden

```
Box {  
    size 2 2 2  
}
```

VRML Primitive IV

- Text
 - Text-Formen sind flach und haben keinen 3D-Effekt
 - gesamter Text in einer Farbe
 - Zeilenumbruch mit Komma
 - "maxExtent": maximale physische Länge des Textes
 - "length": maximale Länge des Textes
 - "fontstyle": Spezifizieren der Schrift (font family, style, size, etc.)
 - Syntax:

```
Text {  
    string []  
    fontStyle NULL  
    length []  
    maxExtent 0.0  
}
```

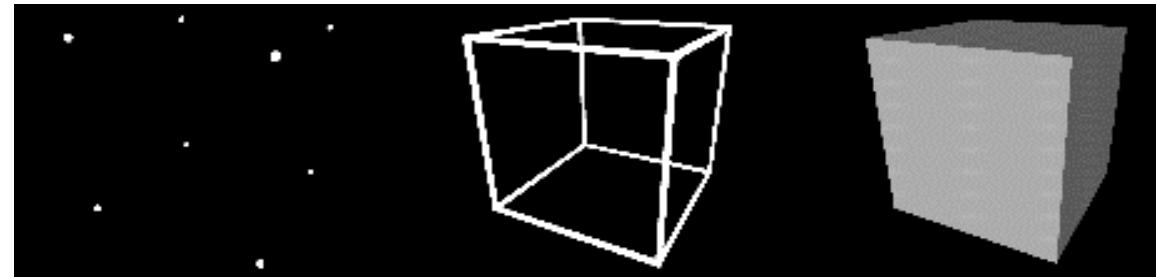
VRML Transformationen -Wiederholung

- Skalierung, dann Rotation, dann Translation einzelner Objekte
- "center"-Feld spezifiziert einen dreidimensionalen Punkt, um den die Objekte skaliert und rotiert werden
- "scaleOrientation"-Feld spezifiziert die Art und Weise, wie das Koordinatensystem eines Objekts rotiert wird
- Syntax:

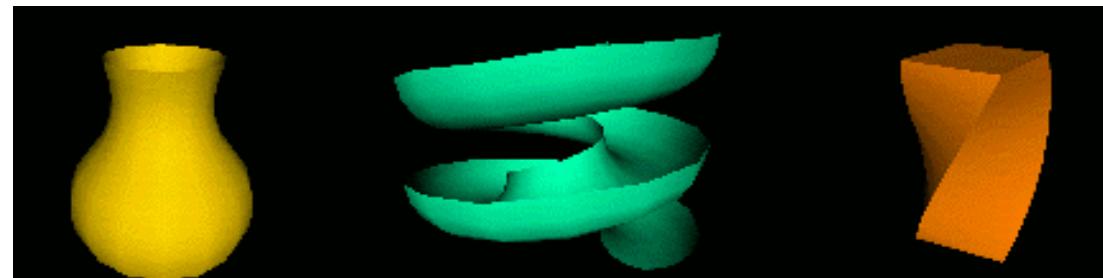
```
Transform {  
    center 0 0 0  
    scale 1 1 1  
    scaleOrientation 0 0 1 0  
    rotation 0 0 1 0   
    translation 0 0 0  
    children []  
}
```

VRML Sets, Extrusion, Grid

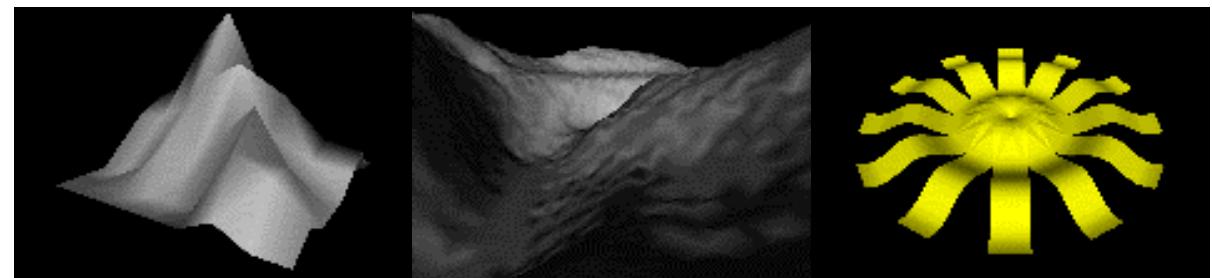
PointSet,
IndexedLineSet,
IndexedFaceSet:



Extrusion:



ElevationGrid:



VRML Sets I

PointSet

- Setzen von Punkten an den angegebenen Koordinaten
- Im "color"-Feld für jeden Punkt eine Farbe
- Dicke der Punkte kann nicht angegeben werden
- Syntax:

```
PointSet {  
    coord Coordinate {  
        point [ x1 y1 z1,  
                x2 y2 z2,  
                . . . ]  
    }  
    color Color {  
        color [ p1, p2, . . . ]  
    }  
}
```

VRML Sets II

- **IndexedLineSet**
 - Erzeugen von Linien zwischen mehreren Punkten
 - Drahtgitter-Modell
 - Koordinaten werden im "coord"-Feld abgelegt
 - Endpunkte der Linien im "coordIndex"-Feld als Indizes
 - Syntax:

```
IndexedLineSet {  
    coord Coordinate {  
        point [ x1 y1 z1, x2 y2 z2, . . . ] }  
    coordIndex [ p1, p2, . . . , -1,  
                 . . . ]  
    color Color {}  
    colorIndex []  
    colorPerVertex TRUE  
}
```

VRML Sets III

- **IndexedFaceSet**

- beschreibt ein Objekt durch Eckpunkte, so dass Flächen entstehen
- traditionelle Art 3D-Objekte zu beschreiben
- Koordinaten werden im "coord"-Feld abgelegt
- Eckpunkte der Flächen im "coordIndex"-Feld als Indizes
- Syntax:

```
• IndexedFaceSet {
    coord Coordinate { point [ ] }
    coordIndex [ p1, p2, . . . , -1, . . . ]
    color Color {}
    colorIndex []
    colorPerVertex TRUE
    texCoord Texturkoordinatenknoten
    texCoordIndex []
    ...
}
```

VRML Sets IV

- IndexedFaceSet weitere Felder:

```
    ...
    Normal Koordinatenknoten
    normalIndex[ ]
    normalPerVertex    TRUE
    creaseAngle  Zahl
    ccw  TRUE
    solid  TRUE
    convex  TRUE
}
```

VRML Sets V

- IndexedLineSet und IndexedFaceSet enthalten noch die Felder
 - [MFColor] color mit einem darunter gesetzten Farbknoten enthaltend (n Farben)
 - [MFInt32] colorIndex[]
 - [SFBool] colorPerVertex TRUE | FALSE
- Wenn colorPerVertex TRUE ist, dann
 - Falls colorIndex nicht vorhanden, werden die Vertizes nacheinander mit den Farben aus color eingefärbt (und dazwischen interpoliert)
 - Falls colorIndex vorhanden, wird jedem Vertex die Farbe aus color zugeordnet, welche dem laufenden Index in colorIndex entspricht.
- Wenn colorPerVertex FALSE ist, dann
 - Falls colorIndex nicht vorhanden, werden die Linien bzw. Flächen des Set nacheinander mit den Farben aus color eingefärbt (keine Interpolation)
 - Fall colorIndex vorhanden, wird jeder Linie/Fläche die Farbe aus color zugeordnet, welche dem laufenden Index in colorIndex entspricht.

VRML Extrusion

- 2D-Querschnitt definieren (crossSection)
- Linie im 3D-Raum definieren (spine)
- Umhüllen des Querschnitts entlang der Linie
- Syntax:

```
Extrusion {  
    crossSection [ 1 1, 1 -1, -1 -1, -1 1, 1 1 ]  
    spine [ 0 0 0, 0 1 0 ]  
    ...  
}
```

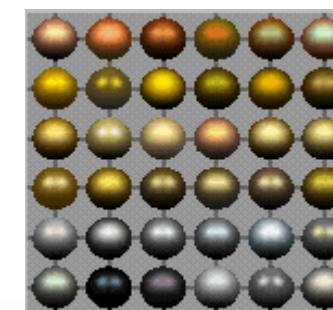
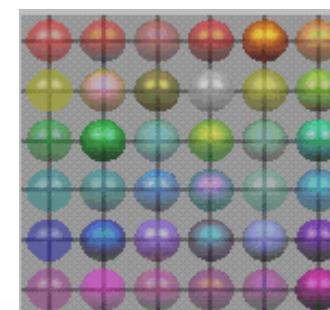
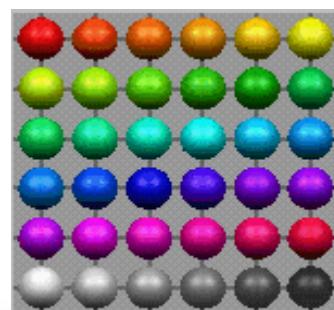
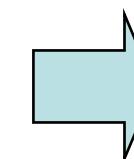
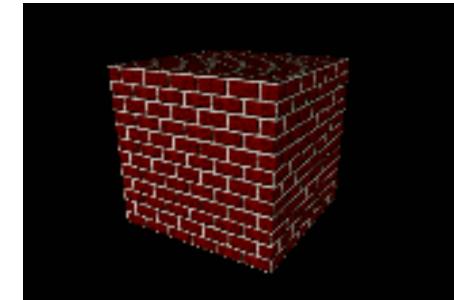
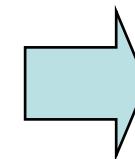
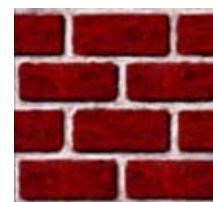
VRML ElevationGrid

- Erzeugen von Bergen, Planetenoberflächen etc. über der x-z-Ebene
- "creaseAngle" für facettenreiche Oberflächen
- "set_height" zur Animation der Höhen und Tiefen
- Darstellung von Farb-Facetten, "colourPerVertex" = FALSE
- benötigt weniger Rechenzeit als IndexedFaceSet

```
ElevationGrid {  
    xDimension 0  
    xSpacing 0.0  
    zDimension 0  
    zSpacing 0.0  
    creaseAngle 0  
    colorPerVertex TRUE  
    height [  
        x1, y1, z1,  
        x2, y2, z2,  
        . . .  
    ]  
}
```

Materialeigenschaften und Texturen

- Aussehen
 - Material
 - Farbe
 - Transparenz
 - Leuchten
 - ImageTexture
 - MovieTexture



Farbdarstellung

- Überschreibt Material-Eigenschaften
- wird benutzt bei IndexedFaceSet, IndexedLineSet, PointSet und ElevationGrid
- enthält eine Liste von RGB-Werten
- Syntax:

```
Color {  
    color [ r1 g1 b1,  
            r2 g2 b2,  
            . . . ]  
}
```

Beispiel ColorPerVertex I

```
#VRML V2.0 utf8
Group {
    children [
        Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 1.0 0.0 0.0
                }
            }
            geometry IndexedFaceSet {
                coord Coordinate {
                    point [
                        0.0 1.0 0.0,
                        -1.0 0.0 0.0,
                        -0.2 0.0 0.0,
                        0.2 0.0 0.0,
                        1.0 0.0 0.0
                    ] }
                }
            }
        }
    ]
}
```

Beispiel ColorPerVertex II

```
color Color {color [
    1.0 0.0 0.0,
    0.0 1.0 0.0,
    0.0 0.0 1.0,
    1.0 1.0 0.0,
    0.0 1.0 1.0,
    1.0 0.0 1.0,
    1.0 1.0 1.0,
    0.5 0.5 0.5,
    1.0 0.2 0.0
]
}
solid FALSE
creaseAngle      0.5
coordIndex [ 0, 1, 2, -1, 0, 3, 4 ]
colorIndex [ 0, 1, 2, -1, 3, 4, 5 ]
}
}
}
```

Texturen in VRML I

- VRML kann 3D-Objekte mit Bildern, Filmen, Muster umhüllen
- Texture überschreibt Material
- Transparenz eines Pixels über sogenannten "alpha channel", mit dem man Löcher in Objekten erzeugen kann
- ImageTexture unterstützt die Formate JPEG, PNG, GIF
- MovieTexture unterstützt das Format MPEG1
- PixelTexture erzeugt ein frei definierbares Muster für das 3D-Objekt
- Syntax:

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

Texturen in VRML II

```
MovieTexture {  
    loop FALSE  
    speed 1  
    startTime 0  
    stopTime 0  
    url []  
    repeats TRUE  
    repeatT TRUE  
}  
  
PixelTexture {  
    image 0 0 0  
    repeats TRUE  
    repeatT TRUE  
}
```

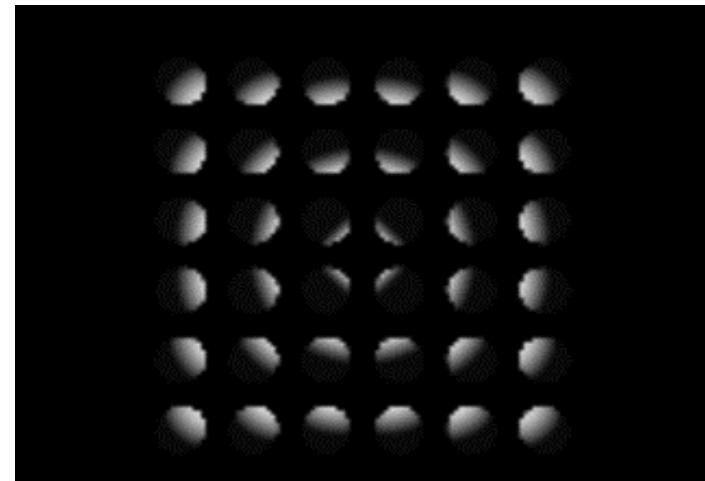
Materialdarstellung

- Farben, die bestimmen, wie Material auf Lichtquellen reagiert
 - bei "diffuseColor" wird ein Teil der auf das Material treffenden Lichtstrahlen in alle Richtungen reflektiert
 - bei "specularColor" wird ein Teil des Lichtes mit dem Einfallswinkel zurückreflektiert
 - durch "emissiveColor" kann ein Objekt selbst Licht abgeben
 - "ambientIntensity" strahlt Umgebungslicht zurück
 - Transparenz
- ```
Material {
 diffuseColor 0.8 0.8 0.8
 specularColor 0 0 0
 emissiveColor 0 0 0
 ambientIntensity 0.2
 shininess 0.2
 transparency 0
}
```

# PointLight

- Punktlichtquelle strahlt kugelsymmetrisch ab
- benötigt unter Umständen viel Rechenzeit
- Abschwächen des Lichtes konstant, linear oder quadratisch
- Objekte außerhalb des Radius werden nicht beleuchtet
- PointLight wird nicht von allen Browsern unterstützt
- Syntax:

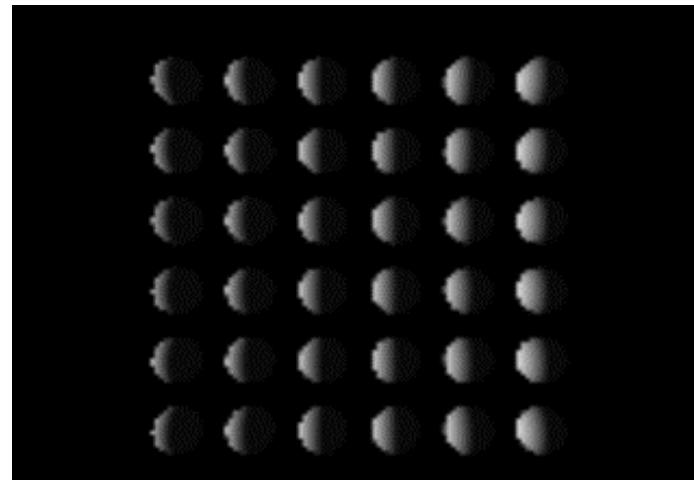
```
PointLight {
 on TRUE
 location 0 0 0
 intensity 1
 ambientIntensity 0
 radius 100
 attenuation 1 0 0
 color 1 1 1
}
```



# DirectionalLight

- Punktlichtquelle ist soweit entfernt, daß ihre Strahlen parallel zueinander erscheinen. Ergebnis: Licht strahlt in eine Richtung
- benötigt am wenigsten Rechenzeit von allen Lichtquellenarten
- Syntax:

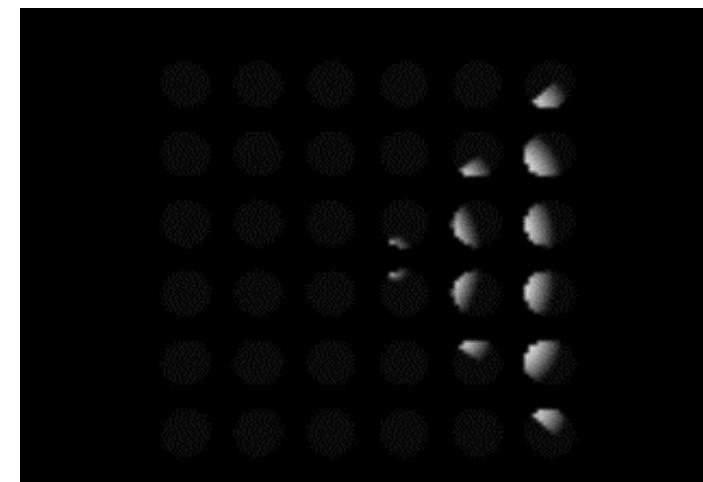
```
DirectionalLight {
 on TRUE
 direction 0 0 -1
 intensity 1
 ambientIntensity 0
 color 1 1 1
}
```



# SpotLight

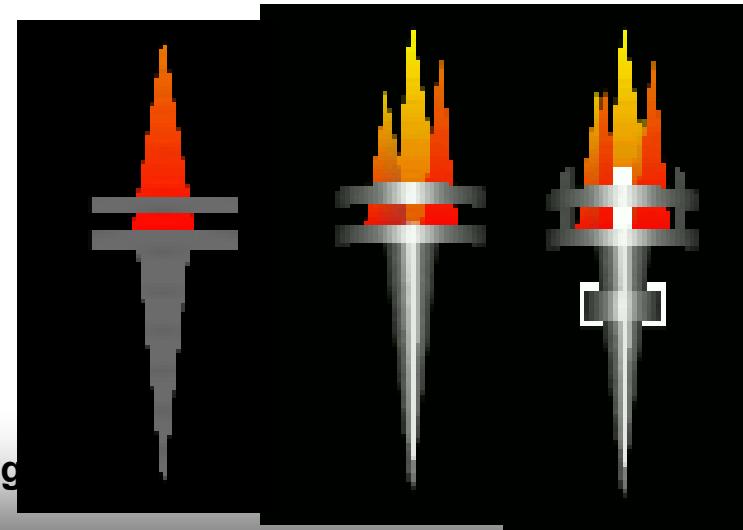
- Lichtquelle strahlt in Richtung der Achse des Lichtkegels
- benötigt am meisten Rechenzeit von allen Lichtquellenarten
- "beamWidth": Maximale Reichweite der Lichtstrahlen
- "cutOffAngle": Winkel zwischen Rotationsachse und Kegelmantel
- ideal verwendbar bei vielen Unebenheiten auf der Oberfläche
- beste Wirkung bei facettenreichen Objekten

```
SpotLight {
 on TRUE
 location 0 0 0
 direction 0 0 -1
 intensity 1
 ambientIntensity 0
 beamWidth 1.570796
 color 1 1 1
 cutOffAngle 0.785398
 radius 100
}
```



# Gruppenknoten

- Transform
- Anchor
- Inline
- Collision
- Billboard
- Level of Detail



Vorlesung Computergr

# VRML Transformationen -Wiederholung

- Skalierung, dann Rotation, dann Translation einzelner Objekte
- "center"-Feld spezifiziert einen dreidimensionalen Punkt, um den die Objekte skaliert und rotiert werden
- "scaleOrientation"-Feld spezifiziert die Art und Weise, wie das Koordinatensystem eines Objekts rotiert wird
- Syntax:

```
Transform {
 center 0 0 0
 scale 1 1 1
 scaleOrientation 0 0 1 0
 rotation 0 0 1 0
 translation 0 0 0
 children []
}
```

# Level of Detail

- LOD = Level Of Detail
- detailreiche Objekte werden erst vollständig angezeigt, wenn der Benutzer dicht vor dem Objekt steht
- **range** ist ein Array von Entfernungen ( n Werte)
- **level** ist ein Array von Objekten, die angezeigt werden (n+1 Werte)
- zwei oder drei Ebenen sind normalerweise ausreichend
- Ziel: Entfernen von Details (z.B. Texture, Text, ...)
- Syntax:

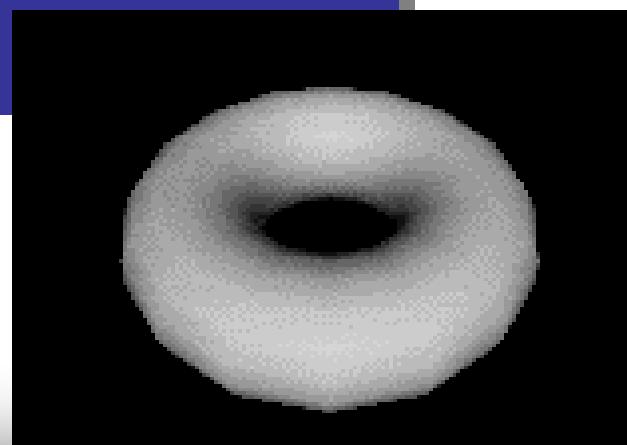
```
LOD {
 center 0 0 0
 level []
 range []
}
```

# Prototyping

- Erweiterung der Sprache durch Definition eigener beliebig komplexer Knoten
- Kapselung von Geometrie und Verhalten
- Parametrisierbarkeit durch Felder und Ereignisse
- Erstellen von Knotenbibliotheken

```
PROTO Donut [
 exposedField SFFloat radius 3
 exposedField SFFloat thickness 1
]
{
 ... Hier Definition in Primitiven ...
}
```

```
Verwendung z.B. als
geometry Donut{
 radius 3
 thickness 1
}
```



VRML

# Sonstige Knoten in VRML

Text

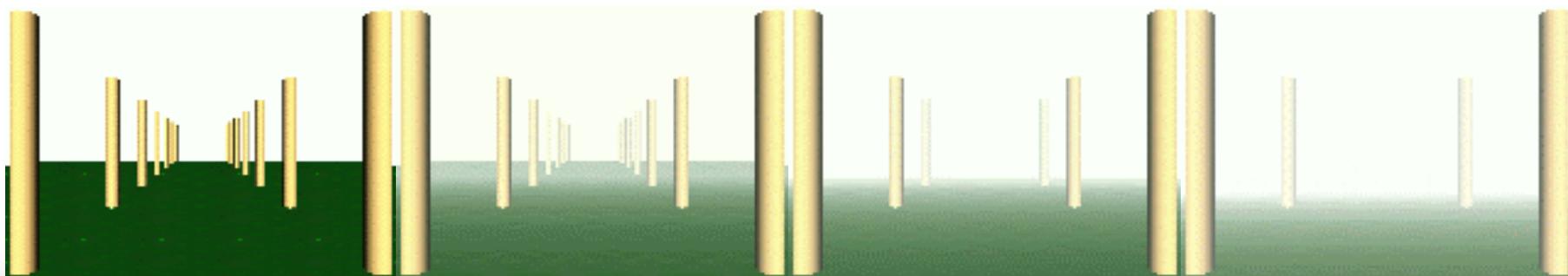
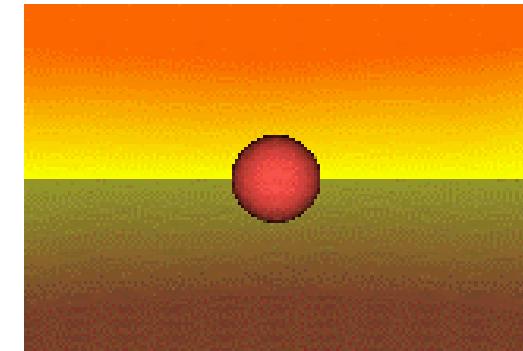
Sound und AudioClip

Viewpoint

NavigationInfo

Background

Fog



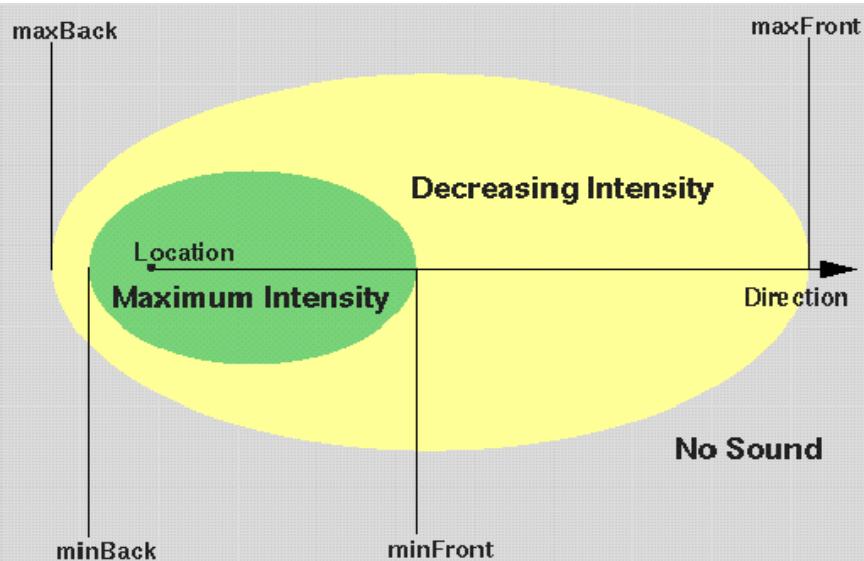
# VRML Background

- Zwei Hintergrund-Typen (können kombiniert werden)
  - Backdrop: Farbangabe für Himmel und Boden
  - Panorama: Einfügen von Bildern
- Darstellung benötigt weniger Rechenzeit als 3D-Objekte
- Hintergründe sind unendlich weit entfernt

```
Background {
 groundColor []
 groundAngle []
 skyColor [0 0 0]
 skyAngle []
 backUrl []
 bottomUrl []
 frontUrl []
 leftUrl []
 rightUrl []
 topUrl []
}
```

# VRML Sound I

- Drei Möglichkeiten
  - kontinuierlicher Hintergrund-Sound (z.B. Verkehr)
  - kontinuierlicher lokalisierter Sound (z.B. Wasserfall)
  - vom Benutzer ausgelöster Sound (z.B. Schalter)
- Sounddatei kann irgendwo im Internet liegen
- Austrahlungsort, -richtung und -winkel
- minimale und maximale Reichweite
- Textausgabe für Computer ohne Sound!
- Soundmodell



# VRML Sound II

```
Sound {
 direction 0 0 1
 intensity 1
 location 0 0 0
 maxBack 10
 maxFront 10
 minBack 1
 minFront 1
 priority 0
 source NULL
 spatialize TRUE
}
AudioClip {
 url [.wav- oder .mid-File]
 loop FALSE
 startTime 0
}
```

# Interface eines VRML Knotens

Ein Knoten enthält Felder mit den Eigenschaften

- offen für Eingangssereignisse (eventIn) oder
- erzeugt Ausgangssereignisse (eventOut) oder
- ist öffentliches Feld (exposedField) = Kombination von eventIn und eventOut

Das Verbinden von *eventOut* mit mehreren *eventIn* heißt fan-out, das Verbinden von *eventIn* mit mehreren *eventOut* fan-in

```
Transform{
 eventIn MFNode addChildren
 eventIn MFNode removeChildren
 exposedField SFVec3f center 0 0 0
 exposedField MFNode children []
 exposedField SFRotation rotation
 exposedField SFVec3 scale 1 1 1
 exposedField SFRotation scaleOrientation 0 0 1 0
 exposedField SFVec3f translation 0 0 0
 field SFVec3f bboxCenter 0 0 0
 field SFVec3f bboxSize -1 -1 -1
}
```

# Ereignisse und Routing

Knoten kommunizieren untereinander durch Ereignisse (Events)

- EVENTS ÜBERMITTELN INFORMATIONEN VERSCHIEDENER TYPEN
- EMPFANG: EVENTIN
- SENDEN: EVENTOUT
- UM FESTZULEGEN, WELCHE KNOTEN AUF WELCHE EVENTS REAGIEREN SOLLEN, WERDEN DEREN FELDER (EVENTOUT -> EVENTIN) MIT ROUTE VERBUNDEN
- WICHTIG: TYPEN VON EVENTOUT UND EVENTIN MÜSSEN IDENTISCH SEIN

**ROUTE Knoten1.isOver TO Knoten2.set\_on**

Knoten 2 setzt das Event set\_on auf den Wert des Events isOver von Knoten 1



# ROUTE

- Routen sind KEINE Knoten, sie verletzen die Baumstruktur
- Routen können stets nur von einem eventOut (exposed) zu einem eventIn (exposed) verlaufen, nicht umgekehrt
- Jedes der folgenden ROUTE-Beispiele verwendet eine korrekte Syntax:

```
DEF Schalter TouchSensor { enabled TRUE }
DEF Licht DirectionalLight { on FALSE }
```

```
ROUTE Schalter.enabled TO Licht.on oder
ROUTE Schalter.enabled TO Licht.set_on oder
ROUTE Schalter.enabled_changed TO Licht.on oder
ROUTE Schalter.enabled_changed TO Licht.set_on
```

# Sensoren in VRML

Sensoren sind Knoten, die

- nach Ablauf eines Timers oder
- bei bestimmten Aktionen des Benutzers

Ereignisse (Events) auslösen.

Sensoren werden aktiviert, wenn ein Objekt

- angeklickt wird: **TouchSensor**
- berührt wird: **ProximitySensor**
- bewegt wird durch eine Zylinderfläche, Ebene oder Kugelfläche:  
**CylinderSensor, PlaneSensor, SphereSensor**
- im Sichtfeld erscheint: **VisibilitySensor**

Sensoren können nach einer bestimmten Zeit aktiviert werden: **TimeSensor**

# TimeSensor

Der TimeSensor wandelt seinen *time*-Wert in der Zeit *cycleInterval* von 0 nach 1

```
TimeSensor {
 ExposedField SFTime cycleInterval 1
 ExposedField SFBool enabled TRUE
 ExposedField SFBool loop FALSE
 ExposedField SFTime startTime 0
 ExposedField SFTime stopTime 0
 EventOut SFTime cycleTime
 EventOut SFFloat fraction_changed
 EventOut SFBool isActive
 EventOut SFTime time
}
```

# Interpolatoren in VRML I

**Interpolatoren** sind stückweise lineare Funktionen, definiert durch n vorgegebene Punkte ( $t_i, f(t_i)$ ), zwischen diesen Punkten wird linear interpoliert

|          |          |          |          |       |
|----------|----------|----------|----------|-------|
| key      | t0       | t1       | t2       | ..... |
| keyValue | $f(t_0)$ | $f(t_1)$ | $f(t_2)$ | ..... |

Je nach Typ des zurückgelieferten Funktionswerts unterscheidet man

- |                                  |                 |
|----------------------------------|-----------------|
| • <b>ORIENTATIONINTERPOLATOR</b> | <b>MFRotate</b> |
| • <b>COORDINATEINTERPOLATOR</b>  | <b>MFVec3f</b>  |
| • <b>NORMALINTERPOLATOR</b>      | <b>MFVec3f</b>  |
| • <b>POSITIONINTERPOLATOR</b>    | <b>MFVec3f</b>  |
| • <b>SCALARINTERPOLATOR</b>      | <b>MFFloat</b>  |
| • <b>COLORINTERPOLATOR</b>       | <b>MFCOLOR</b>  |

# Interpolatoren in VRML II

Beispiele

```
DEF myIP OrientationInterpolator
{
 key { 0.0 , 0.5 , 1}
 keyValue { 0 0 0 1.57, 0 1 0 1.57, 0 0 0 1.57 }
}
```

```
DEF myPalette ColorInterpolator
{
 key { 0.0 , 0.33, 0.66 , 1}
 keyValue { 1 0 0, 0 1 0, 0 0 1, 1 0 0}

}
```

# Interpolatoren in der Animation I

1. Vorgabe fester Punkte der Animation
2. Interpolator berechnet Zwischenschritte für flüssigen Verlauf
3. **TimeSensor:** Event **fraction\_changed** im Intervall [0,1]
4. **Interpolator:** Abbildung von **fraction\_changed** auf jeweiligen Rückgabetyp

```
DEF myClock TimeSensor
{
 cycleTime 5
 loop TRUE
}
DEF myPalette ColorInterpolator
{
 key {0.0 , 0.33, 0.66 , 1}
 keyValue {1 0 0, 0 1 0, 0 0 1, 1 0 0}
}
ROUTE myClock.fraction_changed TO myPalette.set_fraction
```

# Interpolatoren in der Animation II

Mit ROUTE werden die Verbindungen zwischen den benötigten Knoten hergestellt

## TIMESENSOR

**EVENTOUT:** FRACTION\_CHANGED TYP: SKALAR SFLOAT IN [0, 1]

## COLORINTERPOLATOR

**EVENTIN:** SET\_FRACTION TYP: SFLOAT

**EVENTOUT:** VALUE\_CHANGED TYP: SF COLOR

## MATERIAL

**EVENTIN:** SET\_DIFFUSECOLOR TYP: SF COLOR

**ROUTE myPalette.value\_changed TO myMaterial.set\_diffuseColor**

# Interpolatoren in der Animation III

## Einrichten eines Startknopfes:

- Beim TimeSensor: startTime und stopTime haben Default-Wert 0
  - Normalerweise startet TimeSensor sofort (und damit auch die Animation), ggf. das exposedField startTime bzw. stopTime geeignet setzen
- 
- **TOUCHSENSOR SENDET BEIM ANKLICKEN DAS EVENT STARTTIME VOM TYP SFTIME**
  - **TIMESENSOR STARTET IM AUGENBLICK DES ANKLICKENS UND SOMIT AUCH DIE ANIMATION**

```
DEF myClock TimeSensor { ... }
DEF Button TouchSensor { ... }

ROUTE Button.touchTime TO myClock.startTime
```

# Interpolatoren in der Animation IV

## Einrichten eines Berührungsmelders:

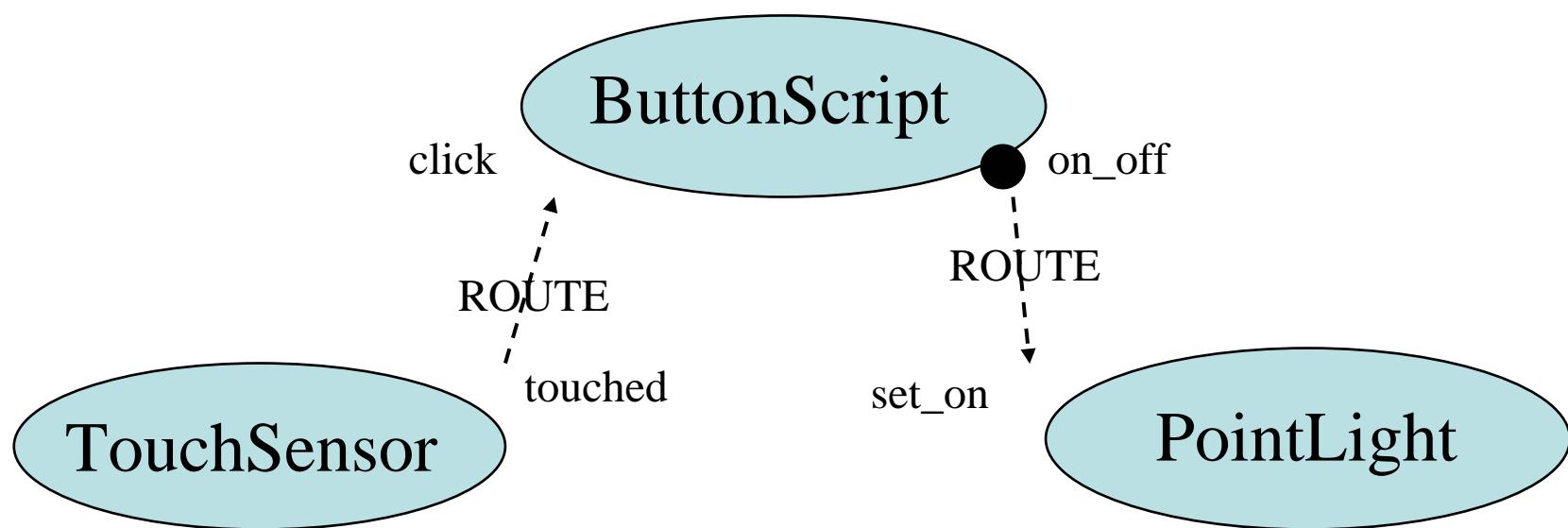
- PROXIMITYSENSOR SENDET DAS EVENT ENTERTIME VOM TYP SFTIME
- TIMESENSOR STARTET IM AUGENBLICK DES BERÜHRENS UND SOMIT AUCH DIE ANIMATION

```
DEF myClock TimeSensor { ... }
DEF Region ProximitySensor { ... }

ROUTE Region.enterTime TO myClock.startTime
ROUTE Region.exitTime TO myClock.stopTime
```

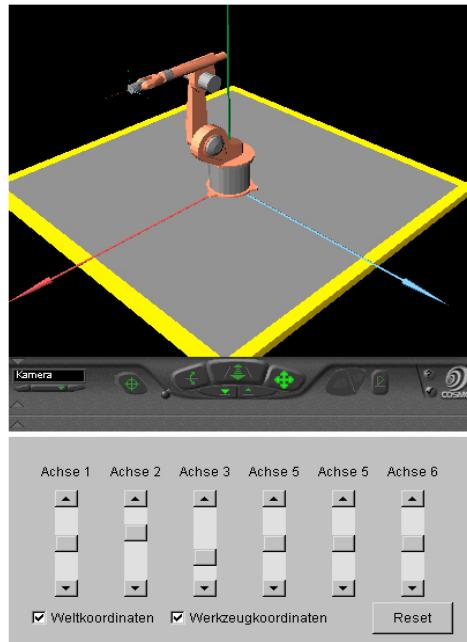
# Skript Knoten

- prozedurale Programmlogik in Szenengraphen integrieren
- Skript-Knoten empfangen und generieren Ereignisse
- Ereignisverarbeitung kann frei programmiert werden



# External Authoring Interface EAI

- Das EIA ermöglicht es, den Inhalt eines VRML-Browser-Fensters, das in einer Web-Seite (HTML) eingebettet ist, von einem externen Applet aus zu steuern, das ebenfalls in die Seite eingebettet ist.



- HTML**

```
<EMBED SRC= "VRMLFile.wrl" >
<APPLET code= "ExternalApplet.class" mayscript>
</APPLET>
```

- VRML**

```
#VRML V2.0 utf8
DEF Knotenname Transform ...
```

- Java-Applet**

```
vrmr.external.Browser;
vrmr.external.Node;
vrmr.external.field.*;
vrmr.external.exception.*;
```

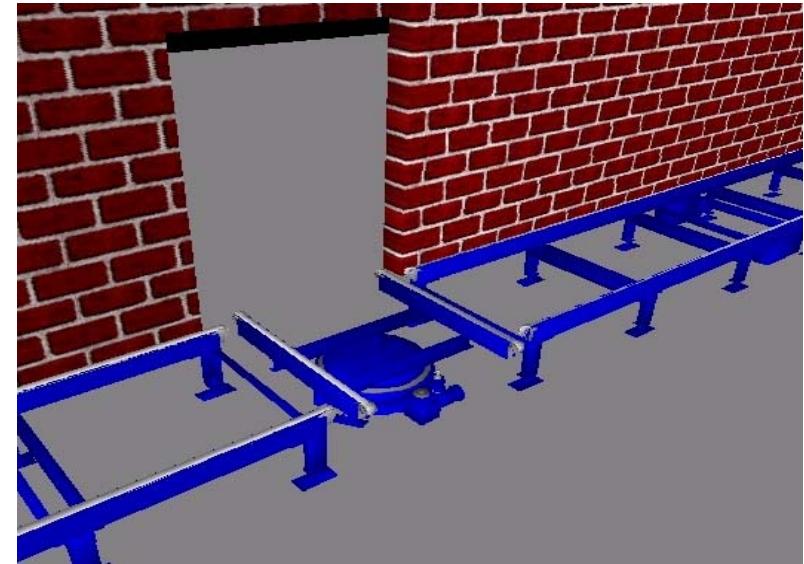
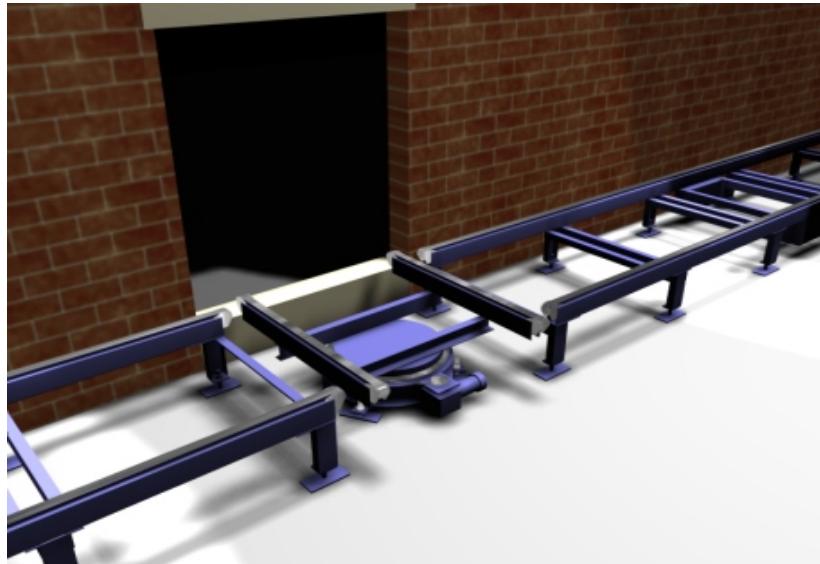
# EAI – Java Codebeispiel

```
Import java.applet.*;
import vrml.external.*;
Import.vrml.external.field.*;
public class myapplet extends Applet {
Browser b;
public myapplet {
 Browser b = Browser.getBrowser(this);
 Node transform = b.getNode("mytrans");
 EventInSFVec3f position_in =
(EventInSFVec3f) transform.getEventIn("translation");
 float neue_position[] = {0, 0, -2};
 position_in.setValue(neue_position);
}
}
```

# Gestaltungsrichtlinien Allgemein

- KURZE, ÜBERSICHTLICHE QUELLCODES
  - BENUTZUNG VON DEF UND USE
  - PROTOTYPEN
  - KOMPLIZIERTE OBJEKTE IN GETRENNTE DATEI, EINBINDEN MIT INLINE
  - Verbindungen mit Anchor
  - GÜNSTIG: ZUSAMMENGESETZTE OBJEKTE IN EINEM EINZIGEN  
TRANSFORM-KNOTEN ZUSAMMENFASSEN
  - DIESER OBJEKTE SIND DANN LEICHT POSITIONIERBAR
- ```
Transform           # gesamtes Objekt
{
    ....
    children
    [
        Transform { ... }      # Teil 1
        Transform { ... }      # Teil 2
        ....
    ]
}
```

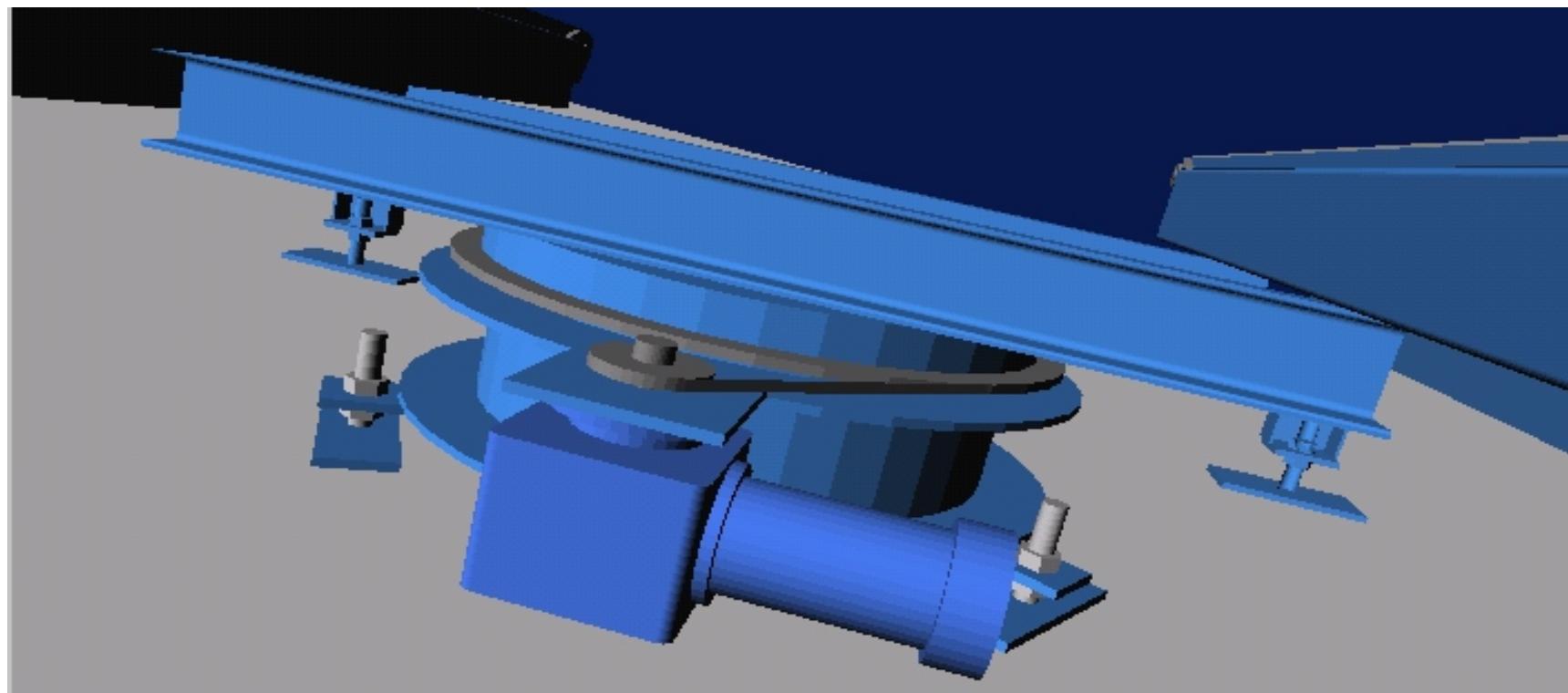
Konvertierung von CAD nach VRML



- Problematiken
 - Komplexität
 - Granularität und Gliederung
 - Fehler bei der Umsetzung der 3D Objekte

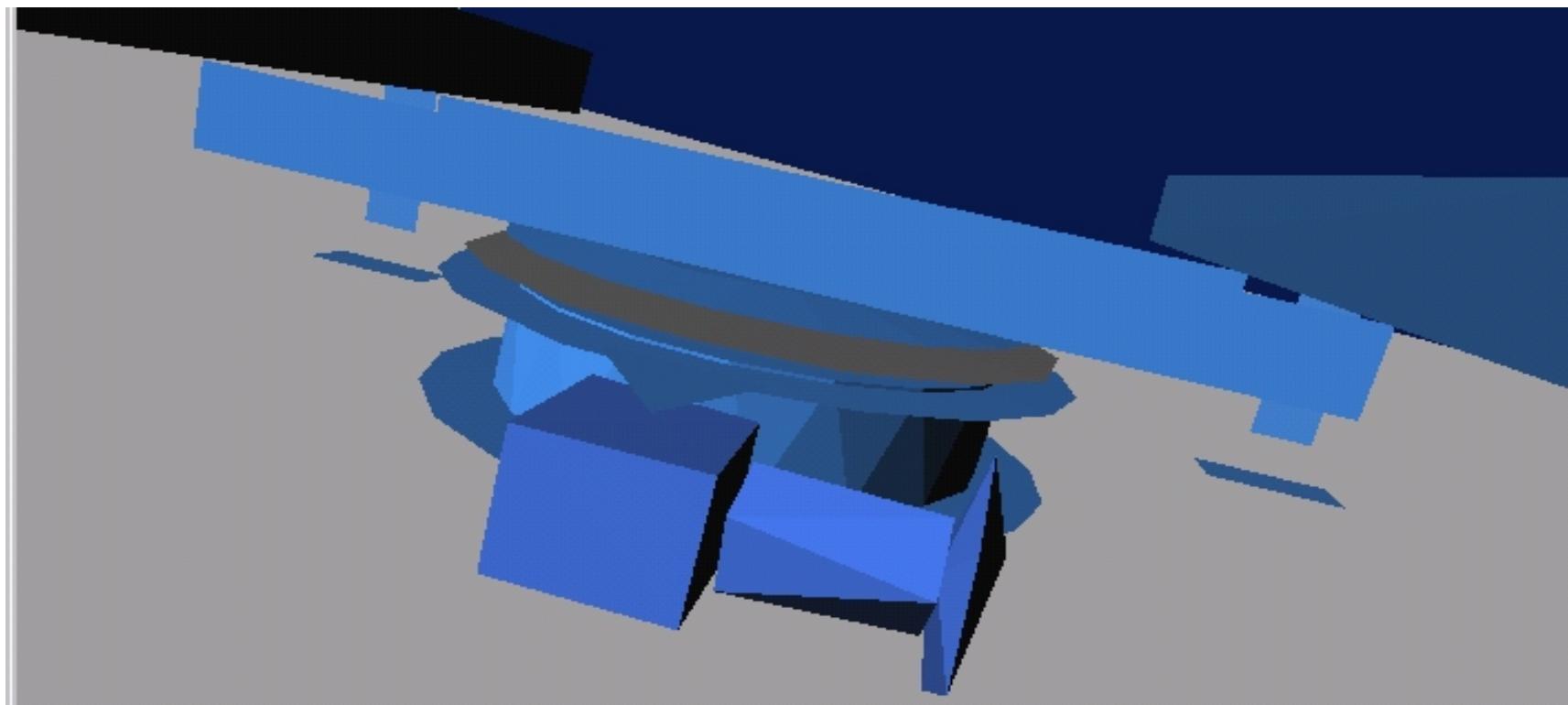
Polygondezimierung I

25747 Polygone (100%)

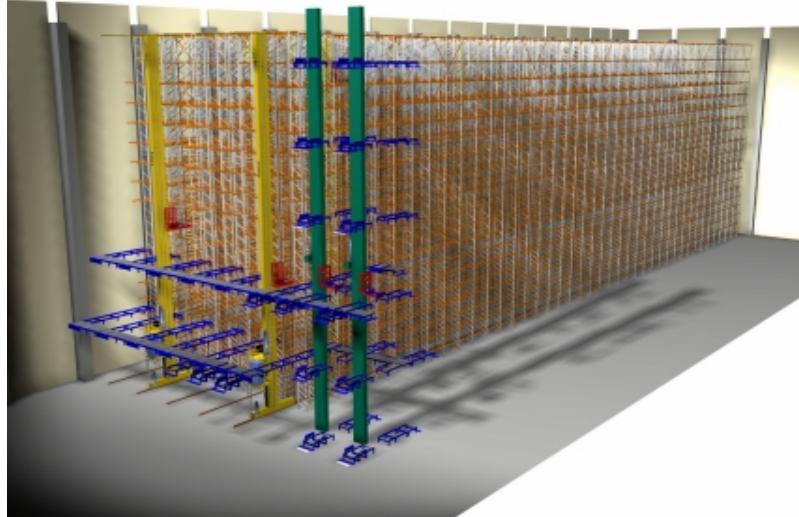


Polygondezimierung II

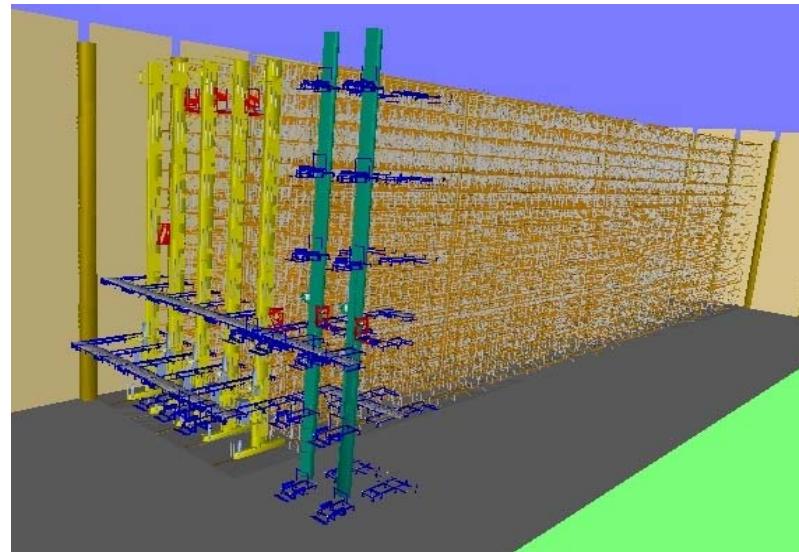
773 Polygone (3%)



AutoCAD

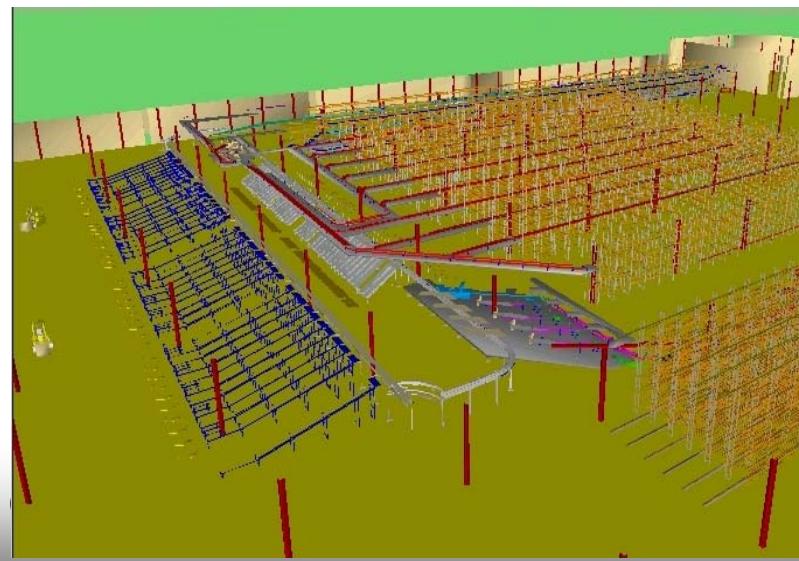
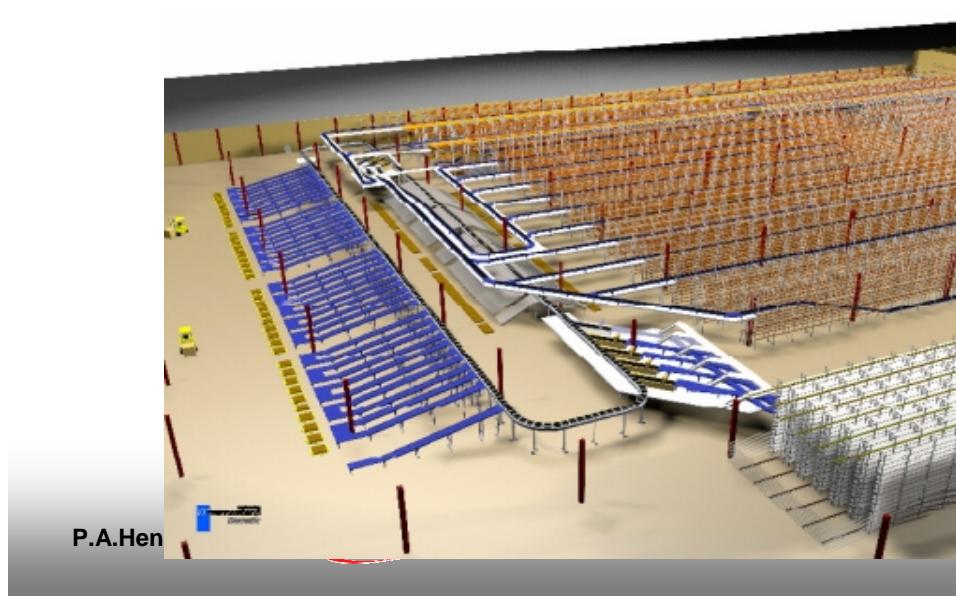


VRML



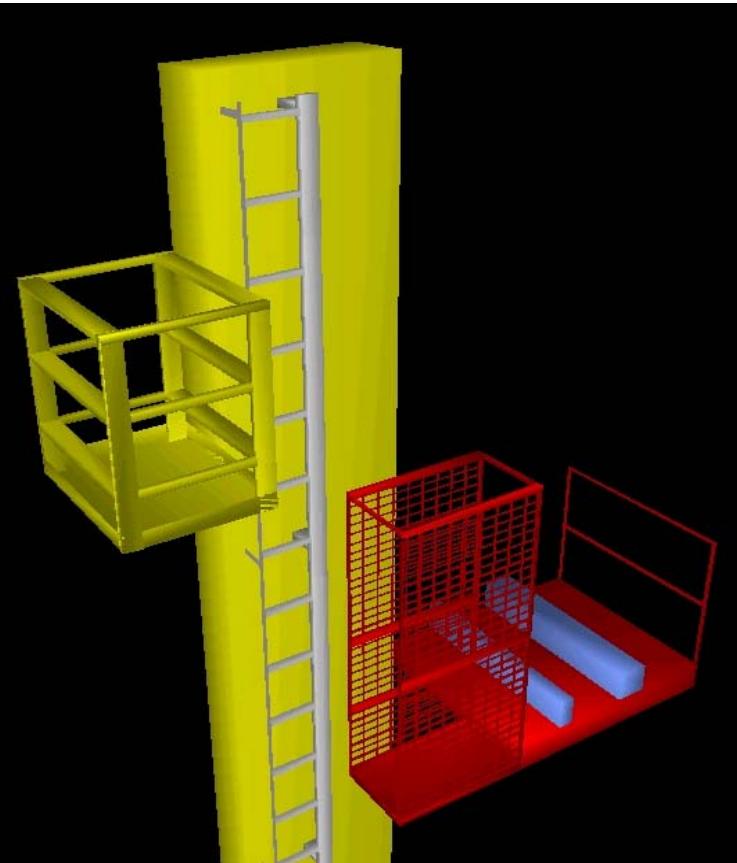
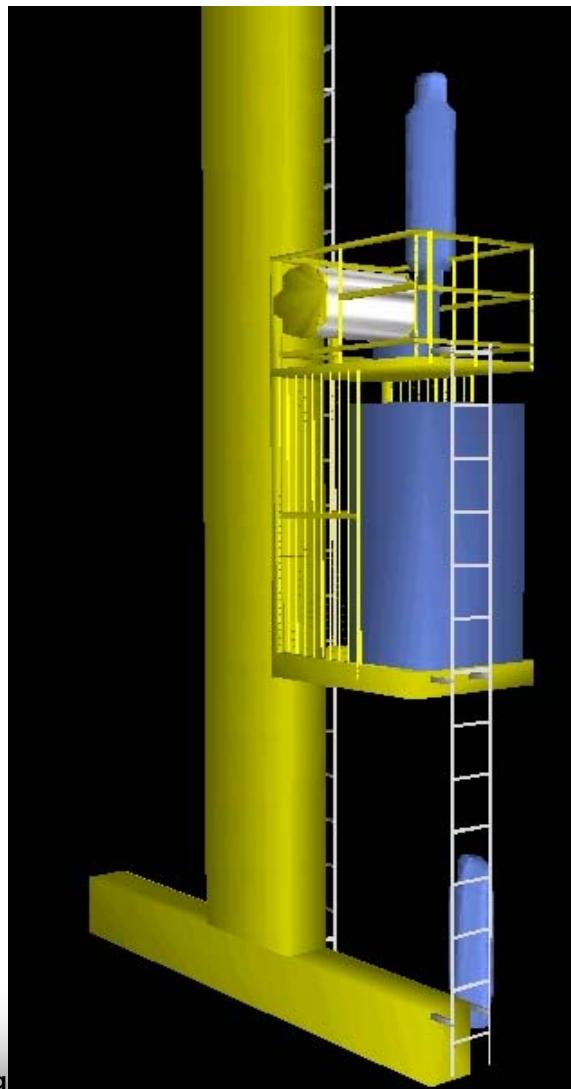
400.000 Polygone

500.000 Polygone



Transportfahrstuhl aus Hochregallager

8.000 Polygone



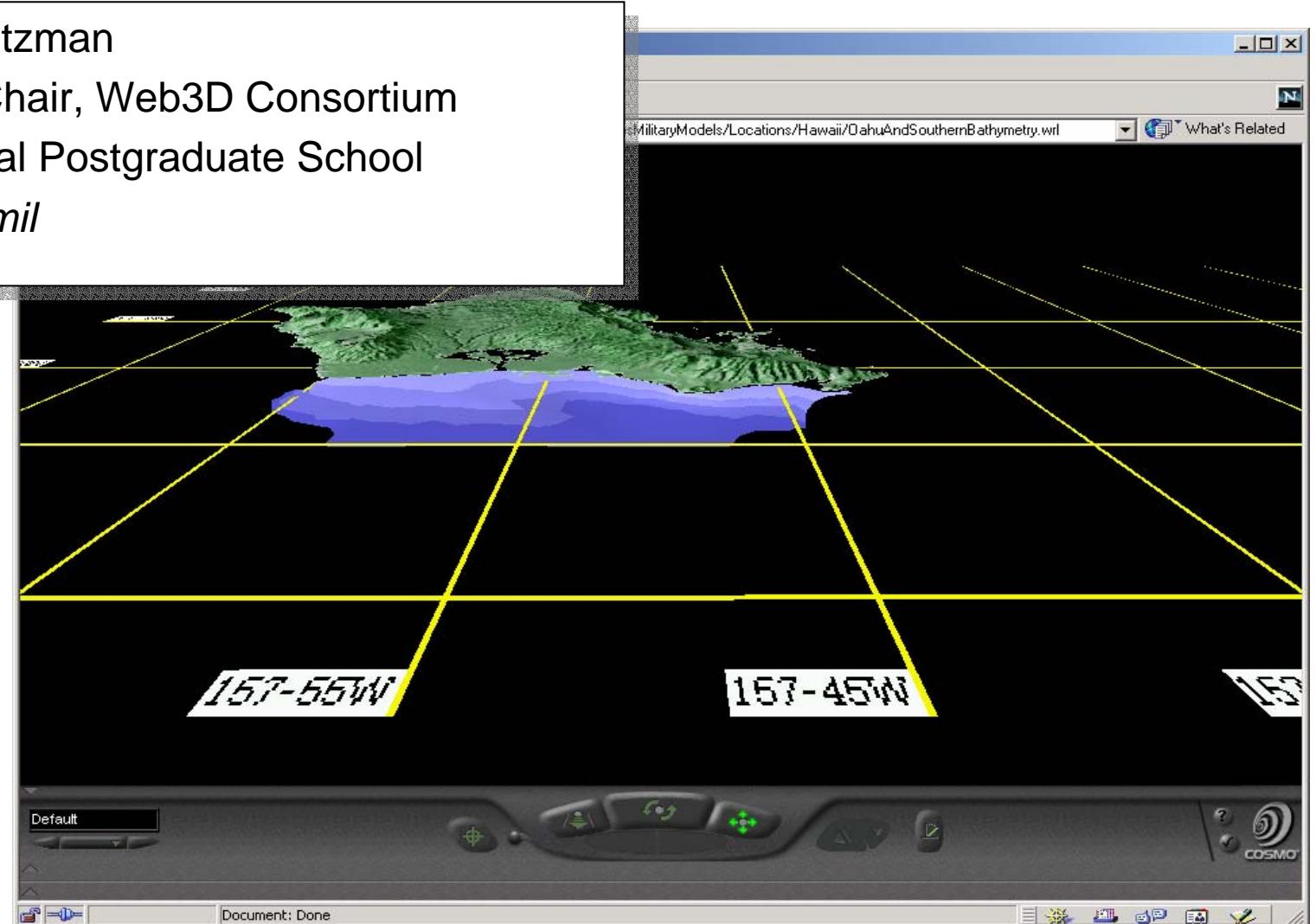
Oahu 3D model in progress

Beispiele von Don Brutzman

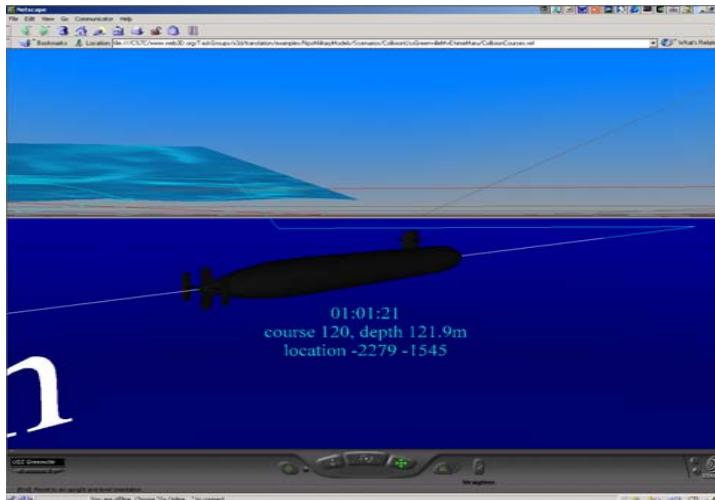
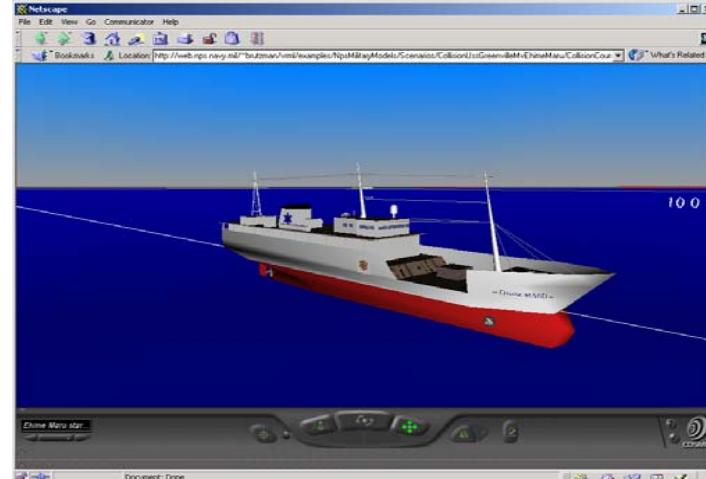
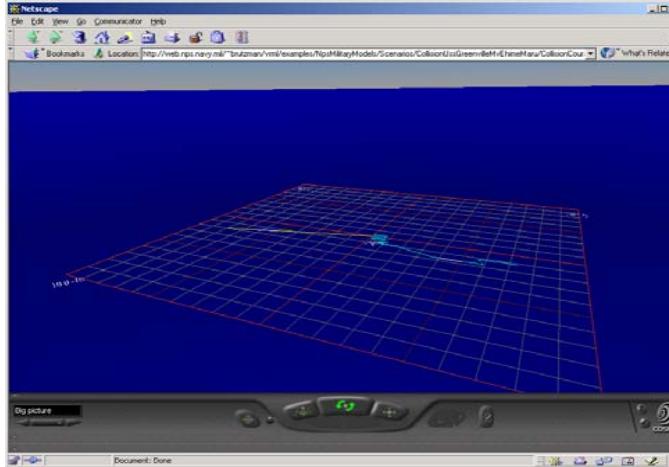
X3D Working Group Chair, Web3D Consortium

MOVES Institute, Naval Postgraduate School

brutzman@nps.navy.mil



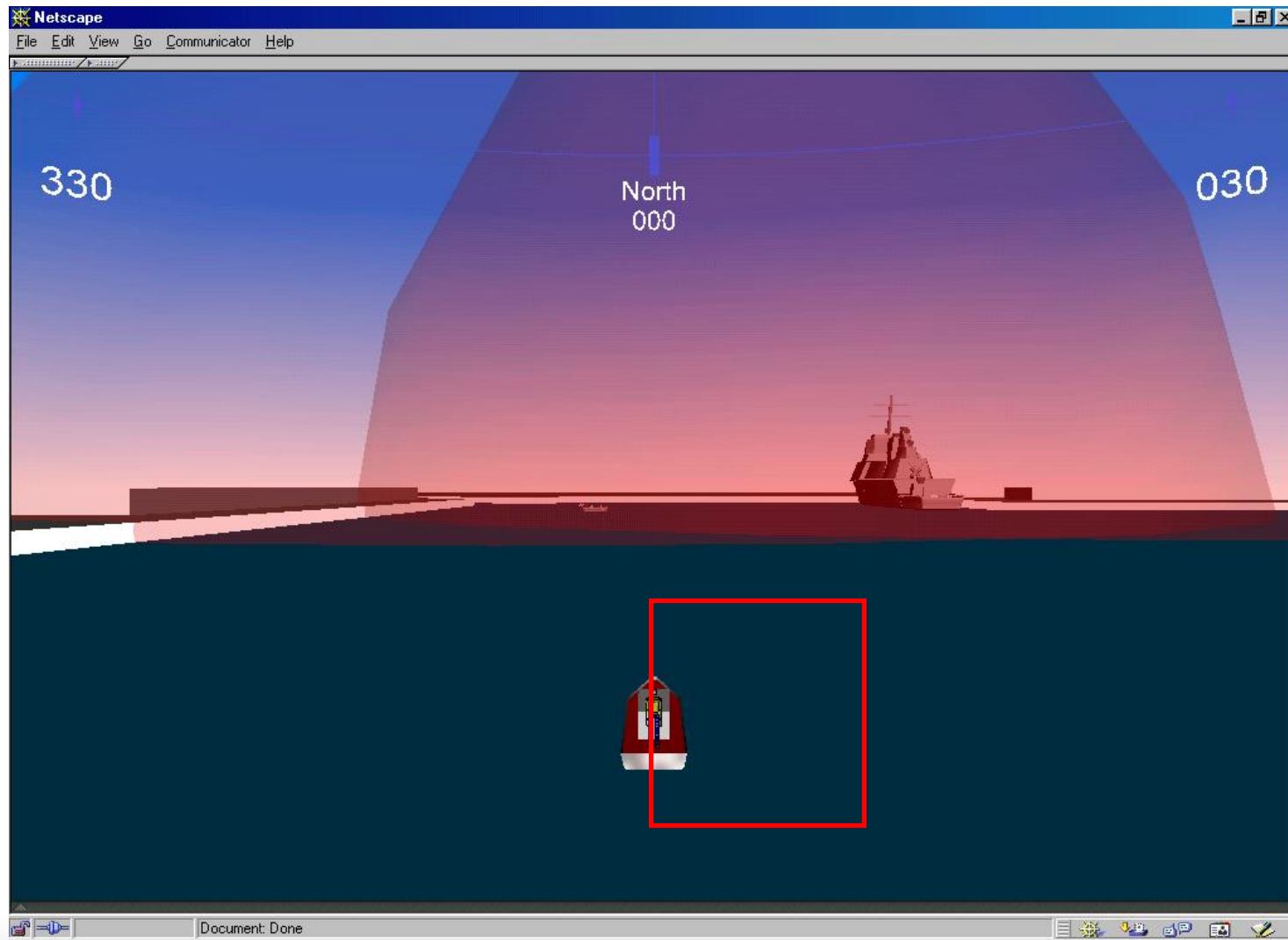
3D model library: collision USS GREENEVILLE



USS COLE attack by Al Queda



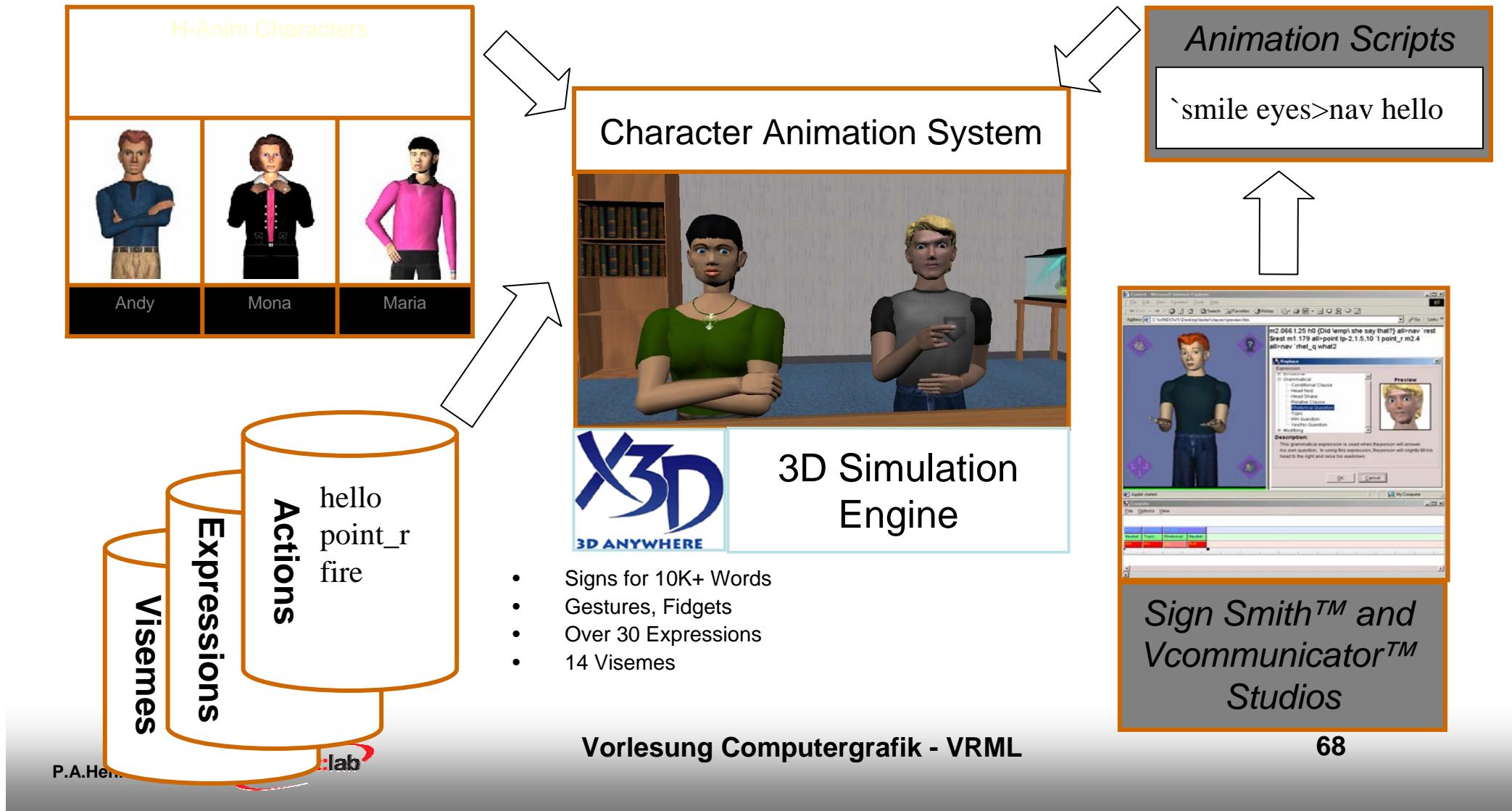
X3D view of Port Hueneme Scene



H-Anim exemplars by VCom3D



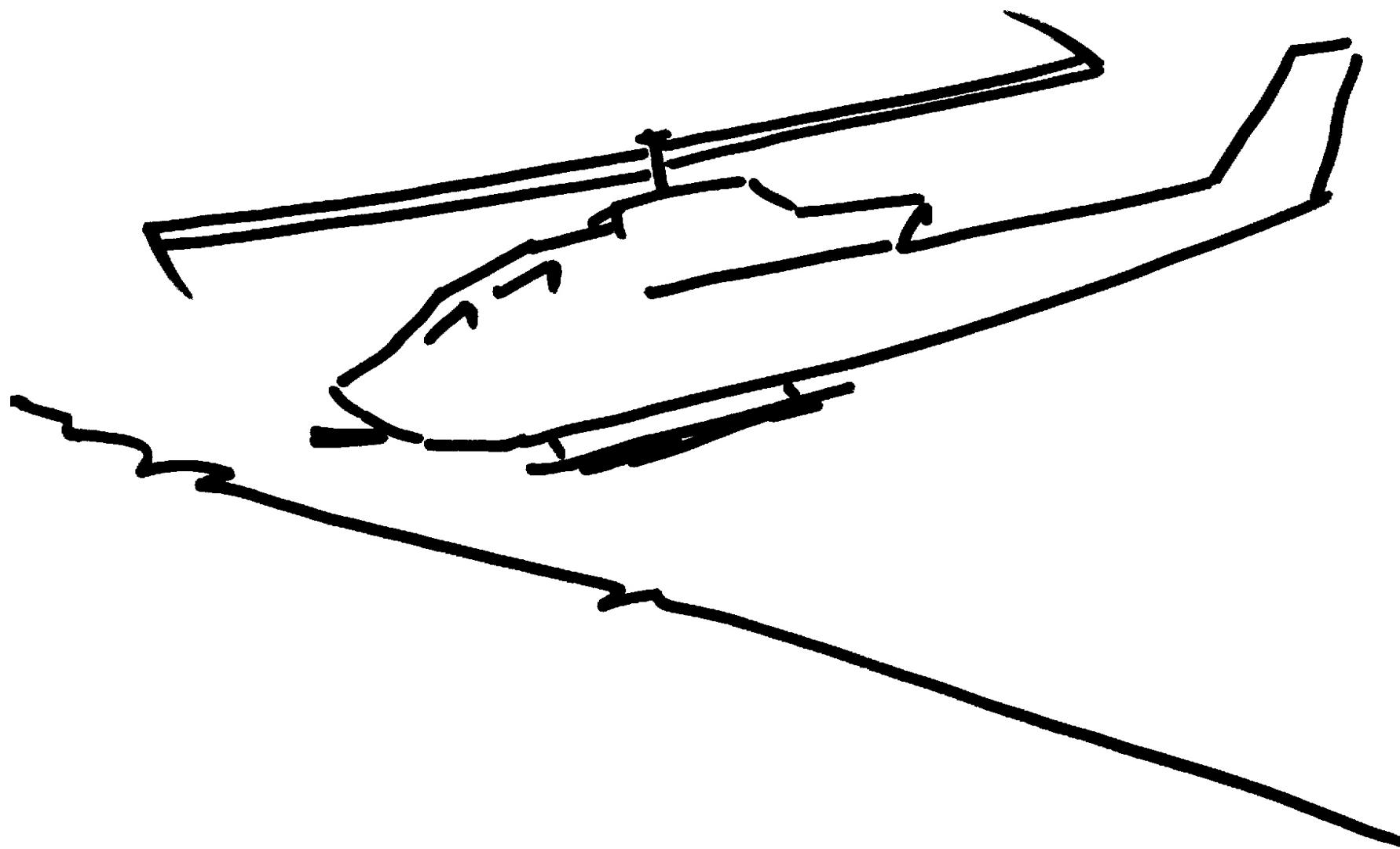
Vcom3D® Character Animation



Human team preparing to enter helicopter



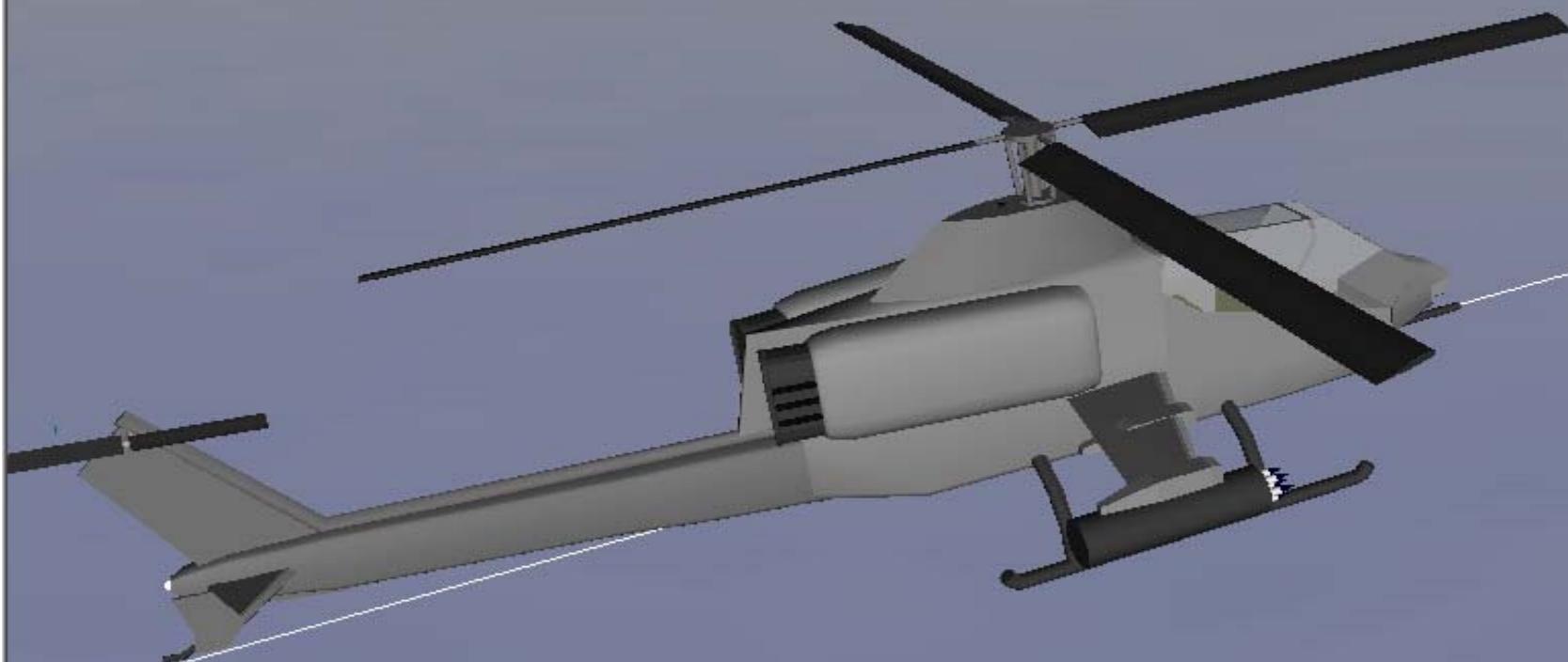
Scene 40: Helicopters on Patrol



Scene 40: Helicopters on Patrol



Scene 40: Helicopters on Patrol



Generic Hub Design Methodology for Battlespace Visualization + Semantics: DoD-wide Autoconversion Of Operation Orders into 3D Virtual Environments

Typical Operation Order:
hand-crafted “Word Document”

D. TANKS

1. **1ST PLATOON, CO A**

- a. AT 0100Z, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ISLAND TO SEIZE ATF OBJECTIVE A
- b. YOU ARE THE RECONNAISSANCE ELEMENT AND WILL LOAD IN AAAV
- c. ONCE YOU REACH THE BLD, YOU WILL FOLLOW IN TRACE OF 2ND PLATOON WHILE DRIVING TO ATF OBJECTIVE A.
- d. UPON CONSOLIDATION ON ATF OBJECTIVE A, ESTABLISH PERIMETER SECURITY IN THE VICINITY OF OBD 1100M0633.
- e. PREPARED TO ASSIST WITH THE TURNOVER TO TF APACHE

2. **2ND PLATOON, CO A**

- a. AT 0100Z, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ISLAND TO SEIZE ATF OBJECTIVE A.
- b. YOU ARE THE COMMAND ELEMENT AND WILL LOAD IN AAAV.
- c. ONCE YOU REACH THE BLD, YOU WILL FOLLOW IN TRACE OF 1ST PLATOON WHILE DRIVING TO ATF OBJECTIVE A.
- d. UPON CONSOLIDATION ON ATF OBJECTIVE A, ESTABLISH PERIMETER SECURITY IN THE VICINITY OF OBD 1100M0633.
- e. PREPARED TO ASSIST WITH THE TURNOVER TO TF APACHE

3. **3RD PLATOON, CO A**

- a. AT 0100Z, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ISLAND TO SEIZE ATF OBJECTIVE A.
- b. YOU ARE THE COMMAND ELEMENT AND WILL LOAD IN AAAV.
- c. ONCE YOU REACH THE BLD, YOU WILL FOLLOW IN TRACE OF 1ST PLATOON WHILE DRIVING TO ATF OBJECTIVE A.
- d. UPON CONSOLIDATION ON ATF OBJECTIVE A, ESTABLISH PERIMETER SECURITY IN THE VICINITY OF OBD 1100M0633.
- e. PREPARED TO ASSIST WITH THE TURNOVER TO TF APACHE

4. **4TH PLATOON, CO A**

- a. AT 0100Z, YOU ARE ON DRASTIC ALERT AND WILL BE PREPARED TO ASSUME THE MEDIUM OF THE MAIN EFFORT

USMTF Message:
Modified Operation Order

A. AT 1100, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ON TO HELICOPTER BLOCS TO TAKE AFV OBJECTIVE A PER DRAFT IN SECTION 3 OF THIS ORDER.

B. PER DRAFT IN SECTION 3 OF THIS ORDER, WITHDRAW TO AND DRAFT ON AFTER TF ARACHE ASSUMES CONTROL OF THE ASWFLD.

C. YOU ARE THE RECONNAISSANCE ELEMENT AND WILL LOAD IN AAAV 1.

2. 2ND PLATOON, CO A

A. AT 1100, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ON TO HELICOPTER BLOCS TO TAKE AFV OBJECTIVE A PER DRAFT IN SECTION 3 OF THIS ORDER.

B. PER DRAFT IN SECTION 3 OF THIS ORDER, WITHDRAW TO AND DRAFT ON AFTER TF ARACHE ASSUMES CONTROL OF THE ASWFLD.

C. YOU ARE THE COMMAND ELEMENT AND WILL LOAD IN AAAV 2.

3. 3RD PLATOON, CO A

A. AT 1100, CONDUCT AN AAAV ASSAULT ACROSS RED BEACH AND CONTINUE ON TO HELICOPTER BLOCS TO TAKE AFV OBJECTIVE A PER DRAFT IN SECTION 3 OF THIS ORDER.

B. PER DRAFT IN SECTION 3 OF THIS ORDER, WITHDRAW TO AND DRAFT ON AFTER TF ARACHE ASSUMES CONTROL OF THE ASWFLD.

C. YOU ARE THE SECURITY ELEMENT AND WILL LOAD IN AAAV 3.

4. REKTIVEEN, CO B

A. AT 1100, YOU ARE ON IMMEDIATE ALERT AND WILL BE PREPARED TO ASSUME THE MISSION OF THE MAIN REPORT.

XML-MTF free-text Operation Order

Operation Order:
GH4 Modified XML-MTF

Message Format Content Format

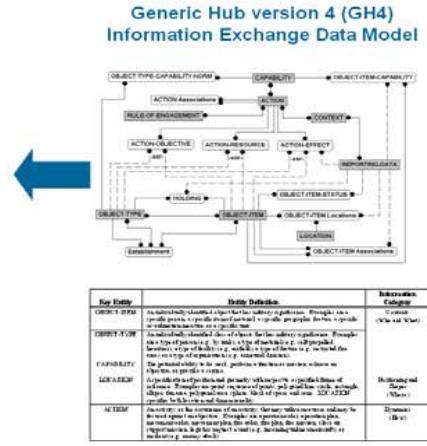
XML-MTF
Operations Order,
with GH4-tagged
order elements, and
without GENTEXT
sections!

Generic Battlespace Schema for Common Operational/Tactical Picture (COP/CTP)

Generic Hub (GH4) now defines an XML Schema for land/maritime/space command and control.

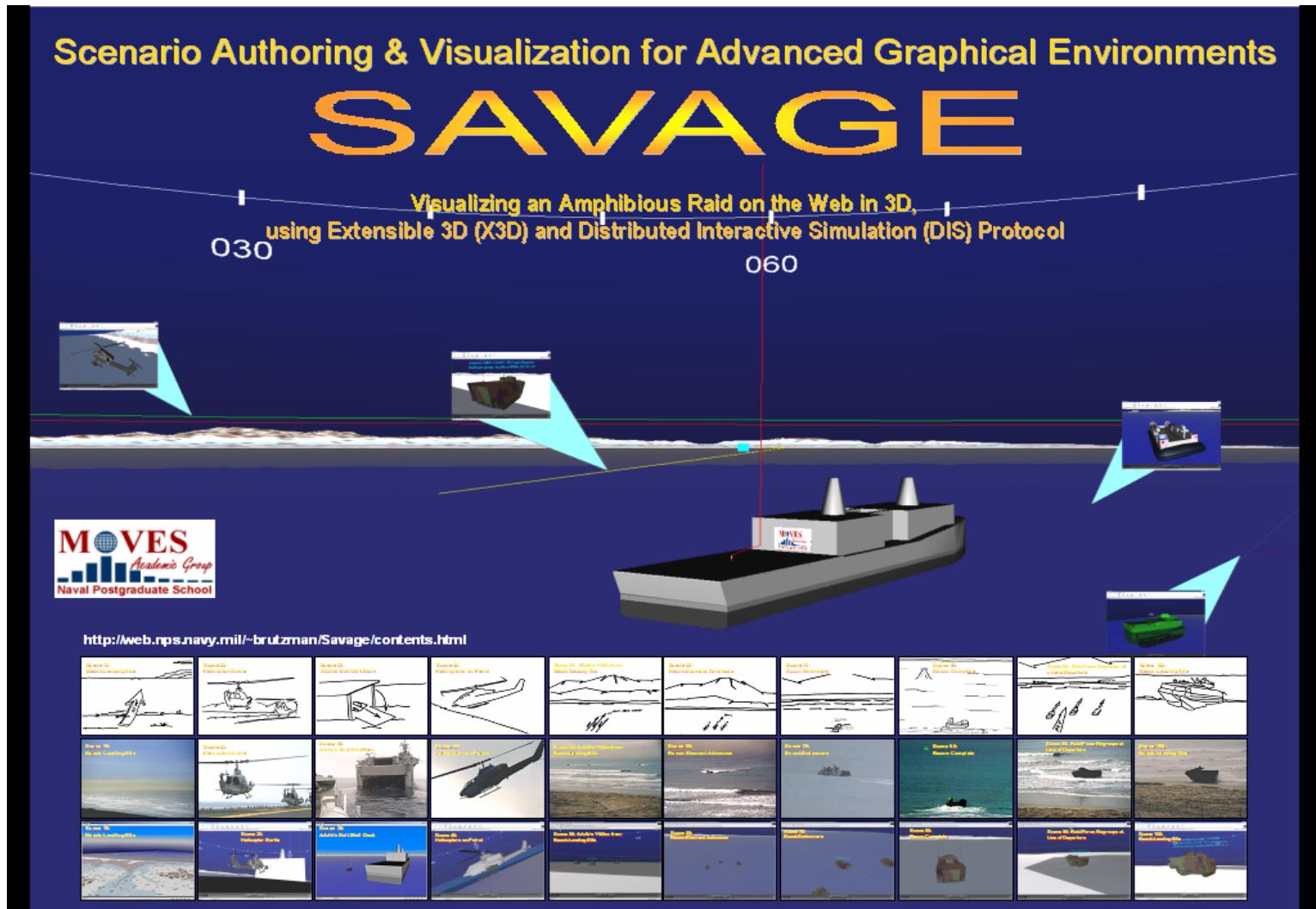
The GH4 Schema supplies agreed-upon semantic tags for a wide spectrum of joint military operations. Readable by humans and combat control systems.

Compatible concepts,
keyword-translatable
internationalization, and
approved for coalition use by
US and NATO Forces.



Operational Context for the Global Information Grid: Smarter Applications, Agents, Autonomous Systems

SAVAGE poster



Das wars zu VRML !