Zusammenfassung des Vortrages

3D-Visualisierung von bewegten Objekten

gehalten von Stephan Koops

am 8. Juni 2005

im Rahmen des Projektes „Ferienclub“

Stephan.Koops@informatik.haw-hamburg.de

Hochschule für angewandte Wissenschaften Hamburg

www.informatik.haw-hamburg.de

http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2005/vortraege.html

Inhaltsverzeichnis

Einführung

Dieses Dokument ist eine Zusammenfassung eines Vortrages, der am 8. Juni 2005 im Rahmen des Projektes „Ferienclub“ im Masterstudiengang Informatik an der HAW Hamburg gehalten wurde.
Der Schwerpunkt ist deshalb nicht nur VRML, sondern auch das Szenario, welches der Autor in dem Projekt umsetzen möchte: Die dreidimensionale Darstellung des Ferienclubs und den sich bewegenden Personen.
Aus dem genannten Grund wird VRML nicht in aller Tiefe vorgestellt, sondern nur als Übersicht dargestellt, um einen groben Überblick über die Möglichkeiten von VRML zu zeigen. Viele Details werden nicht dargestellt. Es wird auf das Quellenverzeichnis mit zahlreichen Links verwiesen.

Motivation für 3D-Darstellung

Es ist wesentlich netter, Sachen dreidimensional zu betrachten und darum und darin zu navigieren, statt sie - wie bisher üblich - nur zweidimensional auf Papier oder Bildschirm zu betrachten. Dies betrifft z.B. die klassische Architektur. Der Architekt zeichnet viele Bilder, mit denen er sich eine gute Vorstellung eines Gebäudes machen kann. Allerdings benötigt man als ungeübter Nicht-Architket doch ein gewisses Vorstellungsvermögen, um sich in Gedanken um und durch das fertige Gebäude zu bewegen.
Ein Gebäude hat den Vorteil, dass es im wesentlichen aus Flächen besteht, die neben- und übereinander stehen und liegen. Noch schwieriger wird es z.B. beim menschlichen Körper, wenn man die Darstellung auf zwei Dimensionen runterbrechen muss. Wenn man den Körper als dreidimensionales Objekt darstellt, ist der Körper (z.B. das Ader-System) wesentlich leichter zu erfassen. Anschaulicher als ein dreidimensionales Bild, in dem man navigieren kann ist da nur noch ein dreidimensionaler Film, in dem man navigieren kann, um z.B. das Herz beim Schlagen zu beobachten.
Der ganz realistische Fall ist in viele Fällen aus praktischen oder finanziellen Gründen nicht möglich, bzw. unverhältnismäßig. Man kann eben nicht durch die Adern schwimmen. In der klassischen Architektur ist es viel zu teuer, eine 1:1-Modell eines Hauses bauen, um es hinterher wieder wegzuwerfen.

Übersicht 3D-Produkte

Auf dem Markt gibt es eine ganze Reihe von Produkten, von denen hier einzelne aufgezählt werden. Diese Auzählung ist sicher bei weitem nicht vollständig:

- VRML / X3D
  VRML ist eine deklarative Beschreibung einer 3D-Welt. Der Anwender benötigt einen VRML-Viewer welcher die VRML-Welt anzeigt, z.B. ein Plugin für den Web-Browser.
  VRML heißt Virtual Reality Markup Language, später auch Virtual Reality Modelling Language)
- java3D
  Java-Api von Sun zum Anzeigen von dreidimensionalen Objekten
- Das Microsoft .Net-Framework hat ebenfalls eine 3D-API
- Anfy 3D12
- Shout 3D13
- Cult 3D14
- viewpoint

Hier soll nur die deklarative Beschreibung mittels VRML beschrieben werden. X3D ist der Nachfolger von VRML. Da es im Internet mehr Tutorials und Beispiele für VRML gibt, wird hier VRML vorgestellt. Es gibt Tools (siehe Quellverzeichnis), mit denen man VRML in X3D umwandeln kann und umgekehrt.

VRML

Geschichte

Bevor es losgeht noch ein kurzer Überblick über die Geschichte von 3D-Darstellungen im Web. 1994 gab es erste Vorschläge für VRML; die Version 1.0. wurde 1995 offiziel spezifiziert. Mit ihr ließen sich statische Welten definieren.
Schon Ende 1995 gab es Vorschläg für einen Nachfolger. Er wurde im August 1996 unter dem Namen VRML 2.0 verabschiedet. Jetzt waren Animationen und Interaktionen möglich. Außerdem gibt es jetzt Schnittstellen zu Javascript und Java. Im September 1997 wurde VRML 2.0 fast unverändert als ISO-Standard unter dem Namen VRML97 festgeschrieben.
Im gleichen Jahr wurde auch das VRML-Konsortium gegründet, welches als Ziel im wesentlichen die Weiterentwicklung und Förderung des kommerziellen Einsatzes von VRML hat. Das VRML-Konsortium ist in Arbeitsgruppen organisiert, es gibt z.B. Arbeitsgruppen für die Weiterentwicklung von VRML oder die Darstellung von Menschen (Humanoid-Anim-Gruppe). Das Konsortium hat 35 Mitglieder, z.B. Microsoft, Sun und Netscape.
1999 hat sich das VRML-Konsortium in Web3D-Konsortium umbenannt, um die Nähe zum w3c (World Wide Web Consortium) zu dokumentieren.
Im Dezember 2002 wurde X3D verabschiedet. Es ist der Nachfolger von VRML.

Beispiel

Nach alter Tradition nehmen wir als Einführungsbeispiel die Ausgabe des Textes „Hello World“. Damit es nicht so langweilig ist, ist die Welt auch zu sehen: HelloWorld [1].
Erklärt werden soll VRML an einem noch einfacherem Beispiel: einem Würfel. Die Datei hat folgenden Quelltext:

```
#VRML V2.0 utf8
Shape
{
  geometry Box { }
  appearance Appearance
  {
    material Material
    {
      diffuseColor 1.0 0.0 0.0
    }
  }
}
```

Die Datei fängt - ähnlich einer XML-Datei - mit einer kurzen Selbstbeschreibung an, dass die Datei eine VRML-Datei ist, gefolgt von der VRML-Versionsnummer und der Codierung. Die Notation mit dem # ist von Linux / Unix aus bekannt.

Szenegraph

In VRML ist die ganze Welt in einem sogenannten Szenengraphen aufgebaut. Der Szenegraph ist ein Baum, in dem die ganze Welt beschrieben wird. Das System stellt ein Koordinatenkreuz zur Verfügung, in dem alle Objekte angeordnet werden.

Shape: Form eines Objektes

Der Hauptknoten in dem Beispiel ist der `Shape`. Shape (deutsch: Form, Gestalt, Gebilde) ist eine Beschreibung für alle sichtbaren Objekte (Körper etc.). Mit `geometry` wird der Typ des Objektes beschrieben. Für alle Datenfelder sind Standardwerte vorgesehen, so dass man nicht alle Datenfelder angeben machen muss. Die Angaben in VRML werden entweder in Metern, Sekunden

1 http://www.lug-s.org/dokumentation/kurse/vrml/extra/javakurs/vrmlkurs.html

oder Anteil von 100% (z.B. 0.5 = 50%) angegeben.
Mögliche Geometrien sind:

- `Box`: Der Quader hat das Attribut `size`, wobei drei Werte angegeben werden: Größe in x-, y- und z-Richtung. Der Standardwert ist hier jeweils 2 Meter.
- `Cone`: Kegel. Bei einem Kegel lassen sich Höhe und Radius des Bodens festlegen. Außerdem kann man jeweils angeben, ob Mantel und Boden sichtbar sein sollen.
- `Cylinder`. Bei einem Zylinder wird ebenfalls die Höhe und der Radius angegeben. Auch hier lässt sich jeweils einstellen, ob der Boden, der Mantel und der Deckel sichtbar sein sollen.
- `Sphere`. Die einzustellenden Werte der Kugel sind recht übersichtlich: Nur der Radius kann angegeben werden.
- `Text`: Hier muss man den Text angeben. Außerdem kann man Angaben zur Schrift machen (Größe u.ä.) und zur Ausdehnung der anzuzeigenden Schrift.
- Mit dem `ElevationGrid` lassen sich beliebige Geländeformen darstellen, beispielsweise Gebirge. Das ElevationGrid ist als Raster organisiert (ähnlich einem Schachbrett, allerdings mit Rechtecken statt Quadraten), wobei jedem Eckpunkt eine unterschiedliche Höhe zugeordnet werden kann.
- Ein `PointSet` ist eine Liste oder Wolke von Punkten, welche zur Beschreibung von komplexeren Linien und Flächen verwendet wird. Auf Details wird hier jetzt - wie auch bei den folgenden Typen - nicht weite eingegangen.
- Ein `IndexedLineSet` ist ein Linienzug, d.h. es wird ein Liste von Punkten angegeben, die der Reihe nach durch eine Linie verbunden sind.
- Mit einem `IndexedFaceSet` lassen sich nahezu beliebige geometrische Objekte aus einzelnen Flächen zusammenbauen.
- `Extrusion`: Die Idee kommt aus der Fertigungstechnik, wo eine Platte in eine Form gepresst wird. Details dazu werden auch hier nicht betrachtet.

Appearance: Aussehen eines Objektes

Das Erscheinungsbild des Objektes wird mit `appearance` beschrieben. In dem Beispiel von eben wird das vordefinierte Erscheinungsbild `Appearance` verwendet, wobei das verwendende `material` des Würfels noch leicht modefiziert wird.

Material

Das Material beschreibt die Farben, die das Objekt hat. Bei den Farbangaben werden jeweils RGB-Farben (rot, grün, blau) angegeben, wobei die jeweiligen Farbanteile nicht von 0 bis 255 angegeben werden (wie in HTML) , sondern von 0.0 (0%) bis 1.0 (100%).

- Die `diffuseColor` ist die Eigenfarbe des Objektes
- Die `emissiveColor` ist die Farbe des Eigenleuchtens des Objektes
- Die `specularColor` ist die Farbe Reflexion des Objektes
- Die `shininess` ist der Reflexionsgrad.
- Die `transparency` gibt den Grad der Transparenz an. So lässt sich z.B. ein Fenster darstellen, welche ja nicht ganz durchsichtig sind.
- Die `ambientIntensity` ist die Leuchtkraft des Objektes

Mit specularColor und shininess lassen sich glänzende Objekte erzeugen.

Texturen

Auf die Objekte oder einzelne Flächen können auch Texturen angezeigt werden:

- Mit `ImageTexture` kann ein Bild (gif, jpeg, png) auf einer Fläche angezeigt werden
- Mit `MovieTexture` können sogar Filme angezeigt werden.
- Mit `PixelTexture` können in der Datei die Farben der einzelnen Pixel angegeben werden. Sinnvoll ist dies, wenn man nur wenige Pixel hat.

Mit `TextureTransform` kann die Größe der Texture angegeben werden. Außerdem kann das Objekt gedreht und verschoben werden, in Bezug auf den aktuellen Ursprung.

Koordinatensystem, Group und Transform

Mehrere Knoten lassen sich zu einer Gruppe (Knoten `Group`) zusammenfassen. Dort kann man dann den Unterknoten `children` angeben, welcher eine Liste (geklammert in `[]`) von weiteren Knoten enthält. Dies ist für die Strukturierung gut. Der Viewer benötigt manchmal auch Schwerpunkte von Objekten. Die werden dann für eine Gruppe ausgerechnet, und nicht für jeden einzelnen Knoten.
Wenn man Objekte nicht nur zusammenfassen will, sondern auch noch gegenüber dem Ursprung des virtuellen Koordinatensystems verschieben will, dann muss der Knotentyp `Transform` verwendet werden, welcher den Knotentyp Group um Verschiebung (`translation`), Drehung (`rotation`) und Skalierung (`scale`) erweitert.

Hintergrund

Der Knoten `BackGround` dient zur Gestaltung des Bodens und des Himmels der Welt. Besonders am Himmel ist dies interessant, weil jeweils Farben und Farbübergänge angeben werden können.

Lichtquellen

Damit nicht alles gleich hell oder dunkel ist, lassen sich Lichtquellen von verschiendenen Typen definieren:

- Das `DirectionalLight` schickt paralleles Licht, wie es auf der Erde von der Sonne kommt.
- Das `PointLight` entspricht etwa einer Glühlampe, die Licht in alle Richtungen abstrahlt. Das Licht schwächt sich mit abnehmender Entfernung ab.
- Ein `SpotLight` sendet kegelförmiges Licht, wie z.B. eine Taschenlampe, wobei der Abstrachlwinkel angegeben werden kann.

Beobachter

In VRML gibt es einen Beobachter, welcher sich durch die Welt bewegt. Deshalb kann der Entwickler einige Angaben zu dem Beobachter machen. Der Beobachter wird im Knoten `NavigationInfo` beschrieben. In einigen Viewern lässt sich der Beobachter auch anzeigen.

- Die Größe des Beobachters lässt sich mit `avatarSize` beschreiben.
  1. Mindestabstand des Beobachters zu Körpern. Dies ist wichtig für die Kollisionskontrolle. Der Viewer kontrolliert, dass der Betrachter nicht durch Wände gehen kann, sonden einen Mindestabstand einhält. Der Standardwert ist 25 cm.
  2. Die Augenhöhe des Betrachters. Standard ist 1,60 m.
  3. Maximale Höhe, die der Betrachter beim Gehen überwinden kann. Dies ist z.B. für Stufen oder Treppen in Gebäuden wichtig. Die Standardhöhe ist 75 cm.
- Wie ein Bergarbeiter hat der Benutzer einen Helm mit Licht dran (`headlight`). Dies kann an- oder abgeschaltet werden.
- Die Laufgeschwindigkeit des Betrachters wird mit `speed` in m/s angegeben. (Standard: 1m/s)
- Der Beobachter kann sich mit verschiedenen Modi bewegen (`type`). Wenn mehrere Modi angegeben werden, wird per Default der erste gewählt, wobei der Anwender dann auch zu den anderen angegeben Modi wechseln kann.
  - `FLY`: Der Anwender kann sich beliebig durch die 3D-Welt bewegen.
  - `WALK`: Gehen mit Schwerkraft, d.h. nur auf Flächen. Das der Beobachter auch auf den Boden zurückkommt, macht der Viewer; da braucht sich der Entwickler nicht drum kümmern
  - `EXAMINE`: Der Anwender dreht das Objekt.
  - `ANY`: Der Benutzer kann sich einen Modi auswählen.
- Wie weit der Beobachter gucken kann, wird mit dem `visibilityLimit` festgelegt. Dies kann auch unendlich sein.

Viewpoints

`Viewpoints` dienen zum schnellen Bewegen des Beobachters zu vorgegebenen Punkten. Der Anwender kann im Viewer aus einer Liste der Viewpoints einen auswählen und kommt dann an die entsprechende Stelle.
Folgende Attribute des Viewpoints können angegeben werden:

- `position`: dreidimensionale Position im Raum
- Die Blickrichtung von dem angegebenen Punkt kann mit `orientation` angegeben werden. Dies ist ein Richtungsvektor (x, y, z) und ein Blickwinkel (relativ zum Boden).
- Damit sich der Anwender in der Liste der Viewpoints zurechtfindet, können die einzelnen Viewports einen Namen bekommen (`description`)
- Ob der Benutzer sofort bei dem angegeben Punkt sein soll, oder sich dort langsam hinbewegen soll, lässt sich mit `jump` steuern (`true` oder `false`).

Wiederverwendung von Code

In VRML ist es wie bei jeder guten Sprachen möglich, vorhandenen Code mehrfach zu benutzen. Knoten können Namen bekommen, um sie später zu referenzieren (`DEF` name). Außerdem kann man einmal beschriebene und erzeugte Objekte später erneut instantiieren (`USE` name).
Weiterhin lassen sich Protoypen definieren. Diese haben ein Interface, wo angegeben wird, welche Attribute oder Knoten variabel sind. Diese kann der Entwickler dann bei Instantiierung angeben, der Rest wird über die Implementierung geregelt. So lässt sich beispielsweise ein ganzes Auto definieren, wo man später nur noch die Farbe angeben muss.
Mit `Inline` kann man andere Dateien einbinden, grob vergleichbar mit einer include-Anweisung aus C.

Animation

Eine Animation besteht aus folgenden Komponenten:

- Sensoren (auslösen von und reagieren auf Ereignisse)
- Routen = Verbindungen zwischen Objekten
- Interpolatoren (berechneen Zwischenwerte)
- (geometrische) Objekte oder dazugehörige Transformknoten

Üblicherweise werden dabei folgende Felder verändert:

- translation
- rotation
- scale

Kompliziertere Aktionen sind auch möglich, z.B. das Hinzufügen oder Entfernen eines Knotens.

Sensoren

Sensoren lösen Ereignisse aus. Folgende Ereignisse sind definiert:

- Der `TimeSensor` ist ein Zeitgeber.
- Der `TouchSensor` reagiert auf Berührungen mit dem Mauszeiger
- Der `VisibilitySensor` registriert die Sichtbarkeit eines Objektes durch den Beobachter
- Der `ProximitySensor` wird aktiviert, wenn der Betrachter einen definierten (quaderförmigen) Bereich betritt. Dies kann man z.B. benutzen, damit sich eine Tür automatisch öffnet, wenn der Betrachter vor ihr steht.
- Der `PlaneSensor` dient der Verschiebung eines Objektes in der x-y-Ebene, wenn der Benutzer z.B. eine Schiebetür per Hand öffnen will.
- Der `CylinderSensor` ermöglicht die Drehung von Objekt um die y-Achse, z.B. für normale Türen.
- Der `SphereSensor` dient zum Rollen eines Objektes um den lokalen Ursprung, ebenfalls durch Ziehen mit der Maus.

Ereignisse

Ereignisse werden bei Veränderung eines Werts ausgelöst. Sie werden durch Felder der Knoten repräsentiert. Die Ereignisse werden durch Routen transportiert (siehe unten). Die Ereignisse entsprechen Methodenaufrufe aus der Objektorientierung.
In einigen Feldern können Ereignisse nur in die Knoten „rein“, aus machen nur „raus“ und bei manchen beides.

- `field`: Diese Felder können zur Laufzeit nicht mehr geändert werden.
- In einem Feld mit dem Typ `eventIn` können nur Ereignisse in den Knoten hinein.
- In einem Feld mit dem Typ `eventOut` können nur Ereignisse aus den Knoten herauskommen.
- Durch ein Feld mit dem Typ `exposedField` können Ereignisse in beide Richtung passieren.

Routen

Ereignisse werden entlang der Routen von einem Datenfeld eines Objektes zu einem Datenfeld eines Objektes verschickt. Beispiel:

```
ROUTE Uhr.fraction_changed TO Interpolator.set_fraction
```

Uhr ist hier ein TimeSensor, der Interpolatoren berechnet Zwischenwerte (siehe nächsten Abschnitt).

Von einem „Ausgang“ können Routen zu mehreren „Eingängen“ gehen und umgekehrt.

Interpolatoren

Zwischenwerte werden werden aufgrund von vorgegebenen Stützwerten berechnet.
Folgende Interpolatoren stehen zur Verfügung, wobei für Details auf die Quellen verwiesen wird:

- Mit einem `PositionInterpolator` können Positionen, d. h. dreidimensionale Vektoren x, y und z interpoliert werden.
- Der `CoordinationInterpolator` dient zur linearen Interpolation komplexer Flächen oder Körper.
- Der `NormalInterpolator` interpoliert über Normalenvektoren.
- Der `OrientationInterpolator` ermöglicht die Drehung um Achsen
- Um Farben kontinuierlich zu ändern, gibt es den `ColorInterpolator`, mit welchem RGB-Werte verändert werden können.
- Der `ScalarInterpolator` als einfachster Interpolator ermöglicht das Interpolieren einzelner Werte.

Dazu werden Stützstellen zu den Eingangs- und Ausgangswerten angegeben. (`key` = Liste bzw. `keyValue` = Liste). Beispiel:

```
DEF TransparencyInterpolator ScalarInterpolator
{
  key [0.0 0.9 0.0]
  keyValue [
    0.0 # undurchsichtig
    0.1 # durchsichtig
    0.0 # undurchsichtig
  ]
}
```

In diesem Beispiel wird ein Objekt langsam durchsichtig (wenn man die Ereignisse und Routen entsprechend mit dem TransparencyInterploator koppelt) und dann schnell wieder undurchsichtig.

Logik in VRML

Die bisherigen Knoten etc. reichen für komplexe Animationen nicht aus. Wenn man z.B. auf ein Ereignis verschieden reagieren will, oder etwas speichern will, dann ist Logik nötig. Dazu wurden Schnittstellen zu JavaScript und Java geschaffen. JavaScript wird direkt in die VRML-Datei geschrieben. Für einfache Sachen ist das das Richtige.
Wenn man größere Aufgaben hat, ist Java geeigneter, weil der Code für JavaScript sonst zu unübersichtlich wird. Wenn man Netzwerkzugriff benötigt, kommt man mit JavaScript nicht weiter, weil JavaScript nicht netzwerkfähig ist. Für Java gibt es auch schönere Entwicklungsumgebungen als für JavaScript. Aus den genannten Gründen wird JavaScript nicht weiter betrachtet.

Knoten Script

Um zwischen VRML und der Programmiersprache kommunizieren zu können, gibt es als Schnittstelle einen VRML-Knoten `Script`.

```
DEF Name Script
{
  field    SFVec3f position [1 2 3]
  eventIn  SFFloat setHoehe
  eventOut SFInt32 hoeheChanged
  url "klassenname.class"
}
```

Die Feldnamen sind hier sehr willkürlich gewählt.
SF* = SingleField: Feld mit einem Wert (eine Farbe etc.)
MF* = MultipleField: Feld enthält Liste (von Farben, etc.)

Java-Packages

Der VRML-Viewer stellt einige Java-Packages zur Verfügung, gegen die der Entwickler programmieren kann:

- vrml.*
- vrml.field.*
- vrml.node.*

Die Implementierung des Skripts erbt von der Klasse vrml.Script. In diesen Klassen kann beliebig programmiert werden. Beim Start wird auf dem Knoten-Objekt die Methode initialize() aufgerufen, am Ende des Knotenlebens (Wechsel zu anderer URL, Knoten wird gelöscht, ...) wird die Methode shutdown() aufgerufen.
Wenn ein im Script-Knoten definiertes Ereignis auftritt, d.h. über eine Route eine Ereignis ankommt, wird die Methode processEvent(..) aufgerufen. Der Name und der Wert des Ereignisses können dann in der Methode abgefragt und verarbeitet werden.
In JavaScript kann man über die Punkt-Notation direkt auf die Attribute zugreifen, was die Bearbeitung etwas vereinfacht.

Wenn der Anwender sich die VRML-Welt in einem Web-Browser-Plugin anguckt, gibt es Einschränkungen, ähnlich wie man sie von Java-Applets kennt. Es ist z. B. nicht möglich zu beliebigen Rechnern Netzwerkverbindungen aufzubauen, sondern nur zu dem Rechner, auf dem die Datei gehostet ist.

X3D

X3D ist der Nachfolger von VRML. Die Umwandlung von VRML in X3D ist per Tool möglich (siehe Quellverzeichnis). Dies geschieht automatisch und quasi ohne Verlust. Wenn allerdings Skripte im Code sind, gibt es häufig Probleme mit der automatischen Umwandlung, so dass der Entwickler dann doch noch Hand anlegen muss.
Hier nochmal jetzt der Wüfel vom Anfang in VRML und in X3D nebeneinander:

```
# VRML V2.0 utf8



Shape {
  geometry Box {}
  appearance Appearance {
    material Material {
      diffuseColor 1.0 0.0 0.0
    }
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile="Full">
  <Scene>
    <Shape>
      <Box />
      <Appearance>
        <Material
          diffuseColor="1.0 0.0 0.0"
        />
      </Appearance>
    </Shape>
  </Scene>
</X3D>
```

Es ist deutlich zu sehen, dass die Struktur von VRML und X3D im Prinzip gleich ist.
- VRML-Knoten werden XML-Elemente
- VRML-Datenfelder werden XML-Attribute

Vorteile von X3D

X3D hat ist den Vorteil, dass es modular in Profilen aufgebaut. Das hat zur Folge, dass Hersteller von Viewern die Möglichkeit haben, Spezial-Viewer zu entwickeln, die einen Teil optimal (z.B. sehr schnell) können, und andere Teile evtl. gar nicht.

folgende Profile sind in X3D definiert:
- Core Profile: ein ganz minimales Profil
- Interchange Profile: beschränkter Lichteinsatz, keine Interaktion
- Interactive Profile: nur eingeschränkte Interakion erlaubt
- MPEG-4 interactive Profile: MPEG-4 Standard
- Immersive Profile: volle VRML Funktionalität
- Full Profile: Volle X3D-Spezifikation

Hersteller haben nun die Möglichkeit, die Spezifikation zu erweitern, um spezielle Darstellungen zu ermöglichen, wenn sie einen passenden Viewer mitliefern.

Für einen Entwickler, der die Dateien per Hand schreibt, hat X3D den Vorteil, dass man nicht wie bei VRML nur eine geschweifte Klammer zu, sondern sieht auch, welchen Knoten er gerade beendet. Wer allerdings einen Editor verwendet, für den ist dies egal.

Viewer

- Web-Browser-Plugin von Cortona (für VRML): für private Nutzung kostenlos
- Web-Browser-Plugin von www.bitmanagement.de (VRML und X3D): zum Testen kostenlose Version
- Xj3D vom Web3D-Konsortium (für X3D):

Für PocketPCs und Handhelds

- Pocket Cortona

Editoren

- Dune ist ein einfacher grafischer Editor zum Bearbeiten von VRML-Dateien. Er ist nach etwa 5 Minuten intuitiv bedienbar. Ein Export zu X3D ist möglich; dazu werden entsprechende Java-Bibliotheken benötigt; der Autor hat dies aber nicht ausrobiert.
- Blender3d ist ein Editor, der sehr professionell aussieht. Auf dem Rechner des Autors gab es Probleme mit der Anzeige; wenn man Buttons o.ä. angeklickt hat, dann verschob sich das angeklickte Element um ein paar Pixel. Vermutlich ist der Editor überdimensioniert.
- mjbWorld ist ein sehr technisch orientierter Editor
- X3D-Edit ist ein einfacher Editor vom Web3D-Konsortium

Beitrag zum Ferienclub

Der Beitrag des Autors zum Ferienclub soll es sein, den Ferienclub dreidimensional anzeigen. Es sollen das Gelände selber angezeigt werden, die Angebote des Ferienclubs wie Sportstätten, Bars und Restaurants und ähnliches. Wenn der Anwender im Club eine Bar anklickt, dann kann er z.B. ein Bier bestellen, welches er in 5 Minuten abholen will.
Weiterhin können auch die Angebote animiert werden, z.B. Tennis spielende Personen.

In dem Gelände sollen dann auch die sich dort bewegenden Personen angezeigt werden. Dabei sollen die Farben der Kleidung der Personen angezwigt werden, je nach dem, was die Personen gerade an haben. Außerdem soll es möglich sein - wenn die Daten entsprechend zur Verfügung stehen -, die Gesichter der Personen anzuzeigen.

Zum Einstieg wird dies evtl. statisch, d.h. die Personen bewegen sich nicht. In der Architektur wird jedoch vorgesehen, dass sich die Personen auch bewegen, d.h. die Positionen der Personen müssen ständig aktualisiert werden. Dies soll sowohl für einen bestimmten Zeitpunkt sein (heute nachmittag, 15 Uhr oder der aktuelle Zeitpunkt), als auch für einen Zeitraum, entweder das jeweils aktuelle, d.h. in Realtime, als auch in einem anderen Zeitbereich.
Szenarien sind z.B.

- Eltern wollen wissen, wo ihre Kinder gerade sind
- Eltern wollen wissen, wo ihre Kinder tagsüber waren.
- Jemand will sehen, wo seine Freunde gerade sind.

Da es in dem Projekt im wesentlichen um Techniken geht und nicht um rechtliche (konkret: datenschutzrechtliche) Aspekte, wird der Datenschutz hier erstmal vernachlässigt.

benötigte Informationen von anderen

Um dieses Ziel zu erreichen benötige ich von anderen Projektteilnehmern einige Informationen:

- Wer ist und war wann wo?
- Wo war Person X zu einem bestimmten Zeitpunkt oder in einem Zeitraum?
- Wer war in der Nähe von einem bestimmten Ort (wieder Zeitpunkt und Zeitraum)?
- Welche Farbe hat die Kleidung?
- Können wir die Gesichter der Personen anzeigen?

Die Informationen über die Positionen der Personen lassen sich aus der RFID-Datenbank lesen. Da die Positionen der Personen ständig auf Aktualität geprüft werden müssen, ist es nicht sinnvoll, die Daten ständig über WebServices abzufragen, weil dies zuviel Datenverkehr über das Netzwerk verursachen werden, welches ja teilweise auch WLAN ist, d.h. die Bandbreite ist nochmal schlechter.

Quellen

VRML und X3D

- http://www.vrml2.de/, 2005-05-26
- http://www.debacher.de/vrml/, 2005-05-27
- http://www.web3d.org/, 2005-05-26
- http://www-lehre.inf.uos.de/~okrone/DIP/node35.html, 2005-06-02
- XML-basierte 3D-Szenebeschreibungssprache - Praxistauglichkeit des VRML-Nachfolgers X3D (Dilpomarbeit von Stefan Rother an der HAW, 2003)
- http://www.hoepnet.de/index.php?id=35&action=detail&sid=21&cat=22, 2005-06-02
- http://www.oreilly.de/catalog/perlmodger/manpage/vrmlpm.htm, 2005-06-03
- http://www.foruma.de/vrml-tutorial/index.htm, 2005-06-03
- http://www.uni-essen.de/hrz/beratung/hrzblatt/hrz150/vrml.html, 2005-06-03
- http://www.csv.ica.uni-stuttgart.de/vrml/howto.html, 2005-06-06
- http://www.gisuser.com.au/GU/content/2001/GU47/gu47_feature/gu47_feature_3.html, 2005-06-07
- http://www.lug-s.org/dokumentation/kurse/vrml/extra/javakurs/vrmlkurs.html, 2005-05-26
- http://www.google.de/search?q=vrml, immer :-)
- Multimedia-Labor der HAW: http://mmlab.haw-hamburg.de/, 2005-06-06

Viewer

- Cortona: Web-Browser-Plugin: http://www.parallelgraphics.com/products/cortona/, 2005-05-26
- Cortona: Viewer für Pocket PC und Handhelp PC http://www.parallelgraphics.com/products/cortonace/, 2005-06-02
- Web-Browser-Plugin für VRML und X3D: http://www.bitmanagement.de/, 2005-05-28
- Xj3D: http://www.web3d.org/x3d/applications/xj3d/, 2005-06-03

Editoren

- http://www.csv.ica.uni-stuttgart.de/homes/js/linuxtag/editoren.html, 2005-06-07
- Blender3D: http://www.blender3d.org, 2005-05-28
- Dune: http://www.csv.ica.uni-stuttgart.de/vrml/dune/index_white_dune.html, 2005-05-28
- X3D-Edit: http://www.web3d.org/x3d/content/README.X3D-Edit.html, 2005-05-28
- mjbWorld: http://www.euclideanspace.com/mjbWorld/index.htm, 2005-06-02
- VRML-Autorenerkzeuge: http://www.csv.ica.uni-stuttgart.de/homes/js/linuxtag/editoren.html, 2005-06-07

Konvertierung

- Umwandlung von VRML nach X3D und umgekehrt: http://ovrt.nist.gov/v2_x3d.html, 2005-06-05