Entwicklung einer Internet 3D-Applikation mit VRML

Vertiefungsblock Kartographie

Andreas Christen

Dominic Moser

Leitung
Prof Dr Hurni

WS 98/99

ETH Zürich

Vorwort

Der vorliegende technische Bericht dient als Dokumentation des Vertiefungsblock K7. Dieser wurde im Wintersemester 1998/99 an der Eidgenössischen Technischen Hochschule in Zürich unter Leitung von Prof. Dr. L. Hurni durchgeführt.

Die Arbeit hat uns ermöglicht, die Visualisierungmöglichkeiten in VRML (Virtual Reality Modelling Language) näher kennenzulernen und zu vertiefen. Für die Unterstützung bei der Datenkonvertierung und Datenbeschaffung möchten wir an dieser Stelle D. Richard und A. Terribilini herzlich danken.

Zürich, den 4. Februar 1999.

Andreas Christen Dominic Moser

Zusammenfassung

Software

VRML (Virtual Reality Modelling Language) bietet gute Visualisierungmöglichkeiten mit einem einfachen Datenformat. Die benötigte Software zur Visualisierung ist ein Browser mit Plug-in und somit jedermann zugänglich.

Daten

In eine 3D-Szene lassen sich sowohl 3D-Daten als auch Rasterdaten integrieren. Von Vorteil sind dxf-Daten, da diese einfacher in VRML konvertiert werden können.

Die Datenmengen sind von grosser Bedeutung. Da die Modelle per Internet geladen werden, führen grosse Datensätze zu unangenehmen Ladezeiten.

Expo.01

Das Expo.01-Gebiet (Neuenburger-, Murten- und Bielersee) war unser Testperimeter für die Visualisierung eines Höhenmodell in Kmobination mit Vektor- und Rasterdaten. Dabei mussten 4 Zoomebenen definiert werden, die über Links miteinander verbunden sind. Weiter bestehen optimale Möglichkeiten Informationsdaten aus dem World Wide Web in das Modell zu integrieren.

Features

Verschiedenste Spezialeffekte können mit VRML im Modell verwirklicht werden. Die Palette führt von verschiedenen Belichtungen/Färbungen über Flüge bis zu animierten Objekten. Zudem ist VRML offen für den Einsatz von Scripts und Programmen in höheren Programmiersprachen (JavaScript, Java, etc.).

Abkürzungsverzeichnis

VRML	Virtual Reality Modelling Language
DHM	Digital Height Model (Digitales Höhenmodell)
ASCII	American Standard Code for Information Interchange
HSV	Hue, Saturation, and Value colour model
HTML	HyperText Markup Language
IEC	International Electrotechnical Comission
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
JTC 1	Joint Technical Committee 1
LOD	Level of Detail
MIDI	Musical Instrument Digital Interface
MIME	Multipurpose Internet Mail Extension
MPEG	Moving Picture Experts Group
PNG	Portable Network Graphics
RGB	Red Green and Blue colour model
RURL	Relative Uniform Resource Locator
UCS	Universal multiple-octet coded Character Set
UTF8	8-bit UCS Transformation Format
URL	Uniform Resource Locator
WWW	World Wide Web

Inhaltsverzeichnis

1. Aufgabenstellung

Das Ziel dieses Vertiefungsblockes ist die dreidimensionale Visualisierung eines Geländemodelles mit Hilfe der Skriptsprache VRML (Virtual Reality Modelling Language). Dabei soll auch geprüft werden, wie weit die Integration anderer Daten wie Sachdaten oder andere topographische Vektordaten möglich ist.

Als Grundlagedaten standen uns das Geländemodell der Schweiz (DHM25) sowie die Pixelkarte des Bundesamtes für Landestopographie in verschiedenen Massstäben zur Verfügung.

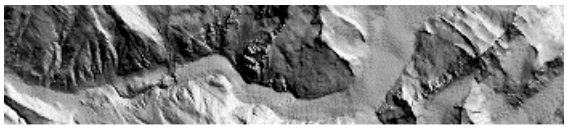
Konkrete Aufgaben:

- Visualisieren eines Ausschnittes aus dem Geländemodell des EXPO-01-Geländes.
- Überlagerung dieses Geländemodelles mit anderen Daten
- Integration von Informationen und Links
- Integration eines Modells einer der vier Arteplages der EXPO-01 (sofern Zeit und Daten vorhanden sind)
- Vergleich mit anderen 3D-Modellierungs- und Visualisierungspaketen.

2. Grundlagen/Verfügbare Daten

2.1 Digitales Höhenmodell DHM25

Digitale Höhenmodelle sind Datensätze, welche die dreidimensionale Form der Erdoberfläche beschreiben. Für jeden Punkt mit den Koordinaten x und y ist die Höhe z, als Grundlage entsprechender EDV-Anwendungen gespeichert.



Das neue digitale Höhenmodell DHM25 ist seit Ende 1996 für die ganze Schweiz flächendeckend verfügbar. Es beruht auf dem Höheninformationsgehalt der Landeskarte 1:25'000 (Basismodell) und den daraus abgeleiteten Höhenwerten in einer regelmässigen Gitteranordnung mit 25 Meter Maschenweite (Matrixmodell). Dieser Datensatz ist für Anwendungen mit hohen Genauigkeitsansprüchen geeignet.

Datenformat DHM25:

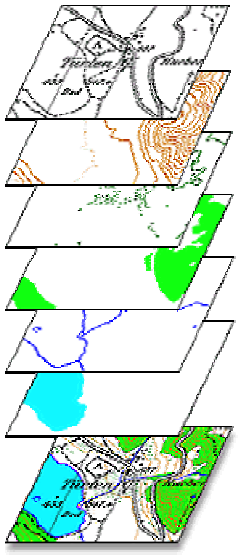
```
NEWHEADER
SYSTEM              PRIMOS        BUND.AMT F. LANDESTOPOGRAFIE   CH-3084 WABERN
FILENAME            DHM25>DHM>BL1109.DHM        SCHOEFTLAND           05.11.93
FORMAT              ASCII                       GANZZAHLIG,  6 STELLEN
RECORDLAENGE(CHAR.)       2040                    340 WERTE PRO RECORD
SCAN-VORLAGE                                    BLATT 1109   1:25000   GN 1982
SCAN-VORLAGE
SCANAUFLOESUNG(/MM)          0
KOORDINATENSYSTEM
AUSSCHNITTKOORD.(M)     637500    242000        BEGINN DES DHM25
AUSSCHNITTKOORD.(M)     655000    230000        ENDE DES DHM25
GITTERAUFLOESUNG(M)         25        25        HOEHEN IN DEZIMETER
GITTERDIMENSION(P.)        701       481        TOTAL    337181 PUNKTE
HOEHENBEREICH(DM)         4245      8709
KOMMENTAR
KOMMENTAR
KOMMENTAR
KOMMENTAR
KOMMENTAR        AUS INPUTFILE                   DHM 1109 GN 1982 MD 15.02.1992
KOMMENTAR                                       ANZAHL NULLHOEHEN =          0
ENDHEADER
5293  5281  5254  5231  5217  5206  5194  5185  5174  5160  5156  5157  5160
5162  5164  5163  5158  5155  5151  5150  5150  5147  5156  5169  5179  5186
5196  5205  5207  5208  5208  5209  5210  5210  5208  5194  5168  5135  5107
5086  5067  5050  5064  5088  5137  5187  5213  5225  5229  5252  5263  5264
5266  5267  5270  5271  5272  5271  5269  5266  5261  5246  5223
```

2.2 Weitere 3D Daten

Ausser den DHM25-Daten standen uns leider keine weiteren 3D-Daten zur Verfügung, weder von der Stadt Biel noch von der Arteplage Biel. Zusätzliche 3D-Daten wurden aus 2D-Daten generiert, wobei die zweidimensionalen Daten der Pixelkarte entnommen wurden. Zur Konstruktion dieser 3D-Elemente wird auf Kapitel 3.4 verwiesen.

2.3 Pixelkarte

Pixelkarten sind gescannte, d.h. elektronisch eingelesene Karten im Rasterformat. Das analoge Kartenbild wird beim Scanvorgang in sogenannte Pixel umgewandelt.



Pixelkarten sind reine Umsetzungen der Druckgrundlagen in eine digitale Form ohne direkten Bezug zu den einzelnen Kartenelementen. Die Karteninformation ist also in einzelne Farbebenen und nicht nach thematischen Objektgruppen getrennt. In der schwarzen Farbebene «Situation» zum Beispiel, befinden sich neben den Gebäuden, Strassen, Wege und Grenzen auch Felsschraffen, die Geröllsignatur und fast die gesamte Schrift.

Pixelkarten sind unter anderem für folgende Anwendungen geeignet:

- Visualisierungen von Karten am Bildschirm
- Planungsgrundlagen
- Einsatzzentralen (Fahrzeugortungs- und Überwachungssysteme)
- Kombination mit Höhenmodellen und Fernerkundungsdaten
- Verknüpfungen mit statistischen Daten
- Basiskarten von thematischen Karten

Die Karten werden mit 508dpi (20 Linien/mm) gescannt. Die Auflösung wurde so gewählt, damit einerseits die feinsten Linien von 0.05 mm abgebildet werden und andererseits die Datenmenge auf ein mögliches Minimum beschränkt werden kann.

Grundsätzlich sind alle topographischen Landeskarten der Schweiz mit dem Nachführungsstand der gedruckten Karten erhältlich. Von den thematischen Karten ist zurzeit nur die Strassenkarte der Schweiz 1:200'000 (PK200STR) erhältlich.

2.4 Karte der Kümmerly+Frey

Zusätzlich zur Pixelkarte des Bundesamtes für Landestopographie stand uns noch eine Karte der Kümmerly+Frey in digitaler Form zur Verfügung. Diese Karte war jedoch nicht in die einzelnen Farbebenen aufgeteilt, was jedoch nötig gewesen wäre, da in der gesamten Karte zuviel Information enthalten ist. Bei einem Überlagern des Geländemodells mit der gesamten Karte wäre das Geländemodell selber zu wenig zur Geltung gekommen. Dies führte dazu, dass wir für das Erstellen der zu überlagernden Bilder auf die Pixelkarten des Bundesamtes für Landestopographie zurückgriffen. Zudem betrug die Dateigrösse trotz Komprimierens noch einige Megabytes, was für unsere Zwecke zu gross war.

3. Verwendete Programme

Für die Visualisierung der VRML-Dokumente genügt ein Browser mit dem Plug-in ‚CosmoPlayer'. Für das Schreiben sowie Editieren der VRML Dokumente reicht ein sehr einfaches Textprogramm. Normalerweise wird aber nur das Hauptdokument von Hand geschrieben. Die grossen 3D-Koordinatendokumente werden mit VRML-Konvertern aus anderen Formaten erstellt. Solche Konverter können gratis vom Internet heruntergeladen werden. Eine Sammlung findet man unter: http://www.vislab.usyd.edu.au/vrml/. Die meisten Konverter können das 3D Standardformat ‚DXF' in das VRML1 Datenformat umwandeln. Da unsere Grundaten aus DHM25- und Vektordaten bestanden mussten wir noch weitere Konvertierprogramme anwenden um zuerst in das DXF Format zu gelangen. Weiter mussten wir grossen Wert auf Datenreduktion legen. Die für die Konvertierung verwendeten Programme werden nachfolgend beschrieben.

Unsere Daten nahmen folgenden Weg:

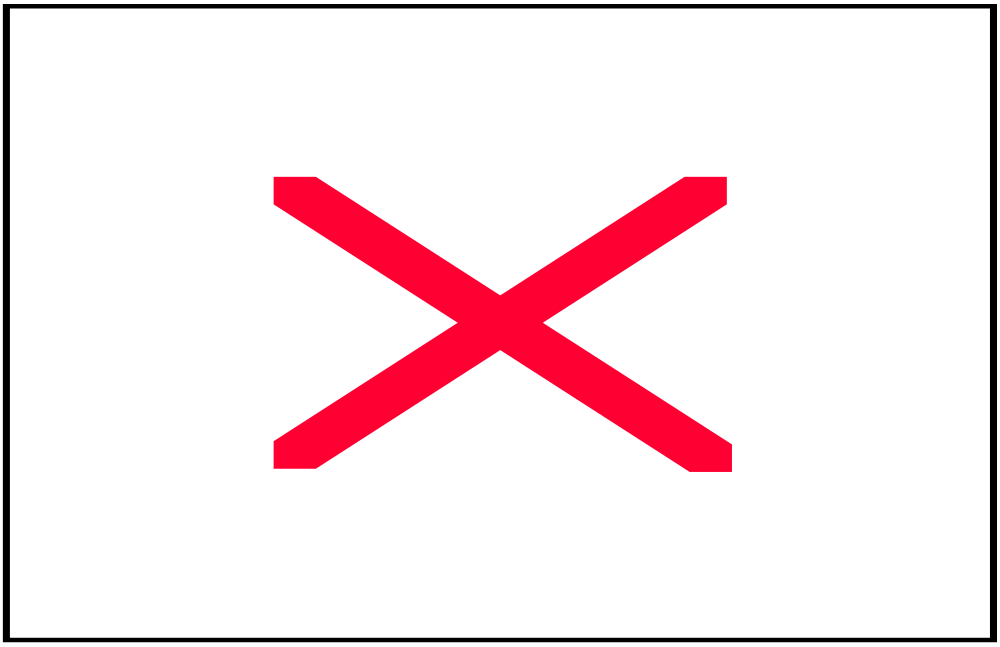


Abbildung 1: Konvertierprogramme

3.1 Umwandlung vom DHM25-Format ins dxf-Format

Bei dieser Umwandlung ging es darum, eine möglichst gute und angepasste Reduktion der Daten zu erreichen. Wir führten die Umwandlung in verschiedenen Schritten durch. Als Ursprungsdatensatz ist ein regelmässiges Punktraster mit Punktabstand 25 Meter gegeben.

Als erstes wurden die Daten in eine Binärdatei konvertiert. Der Algorithmus des zweiten Schrittes besteht darin, dass von Punkt zu Punkt untersucht wird, ob eine markante Höhenänderung stattfindet, wobei je nach Reduktionsgrad ein anderer Grenzwert festgelegt wird. Ist die Höhenänderung kleiner als der Grenzwert, so wird der Punkt weggelassen. Dies führt dazu, dass im Berggebiet eine hohe Punktdichte vorhanden ist, währenddem für einen See praktisch nur die Uferpunkte nötig sind. Somit ist das regelmässiges Punktraster nicht mehr vorhanden, d.h. alle Punkte müssen mit vollständigen Koordinatenangaben aufgeführt werden (im Gegensatz zum regelmässigen Punktraster, wo die Angabe des Punktabstandes, der Ausdehnung des Gebietes sowie aller Höhen genügt). Die Wahl der gewünschten Reduktion erfolgt durch Angabe eines maximalen Wertes für die Höhenungenauigkeit, die durch Weglassen von Punkten entsteht (beim Übersichtsmodell 'Seen' zum Beispiel beträgt dieser Wert 200 Meter).
Ein weiterer Schritt transformierte das Punktraster in eine Dreiecksvermaschung (TIN Format). Dabei sorgt der Delaunay-Algorytmus dass dabei möglichst grosse Dreieckswinkel gewählt werden. Als letztes muss noch die eigentliche Formatumformung ins DXF vorgenommen werden.

Ein anderer Lösungsansatz ist eine Reduktion, bei welcher in regelmässigen Abständen (je nach Reduktionsgrad) Punkte weggelassen werden, was als Resultat wieder ein regelmässiges Raster ergibt. Diese Reduktion nimmt jedoch keinen Bezug auf die wirkliche Topologie, d.h. in einem steilen Berggebiet wird die Punktanzahl im gleichen Masse reduziert wie in einer flachen Ebene oder einem See, und ist von daher für unser Projekt ungeeignet.

3.2 Umwandlung vom dxf-Format ins wrl-Format (VRML)

Für die Umwandlung des dxf-Files in ein wrl-File, also eine VRML-Datei, standen uns zwei Programme zur Verfügung. Zum einen war dies ein eher kleines Programm namens Wcvt2pov sowie Simply3D von Micrografx, ein Programm, welches dazu dient, 3D-Szenen zu erstellen. In beiden Fällen bestand die Aufgabe jedoch nur darin, das dxf-File zu laden und anschliessend als wrl-File wieder abzuspeichern.

Anhand eines Beispiel-Files wurden beide Programme auf ihre Leistungsfähigkeit überprüft. Die Vorteile von Simply3D liegen dabei eindeutig bei der Benutzerfreundlichkeit, so kann im Programm gleichzeitig mit dem Umwandeln auch noch das Plazieren von Lichtquellen oder das Ändern der Oberfläche eines Objektes vorgenommen werden. Ausserdem unterstützt das Programm direkt das Format VRML 2.0, dies im Gegensatz zu Wcvt2pov, welches nur VRML 1.0 unterstützt. Dies ist jedoch kein grosser Nachteil, da die Unterschiede zwischen den beiden Sprachen klein ist, und ein geeignetes Programm zur Umwandlung von VRML 1.0 nach VRML 2.0 bereitsteht (siehe Kapitel 3.3). Dadurch, dass Wcvt2pov ein eher kleines Programm ist und speziell für Umwandlungen konzipiert ist, ist es einfacher und schneller in der Anwendung. Zudem ist die Struktur der wrl-Datei um einiges übersichtlicher, und kann ohne Probleme weiterbearbeitet werden. Eine Bearbeitung muss gerade am Anfang noch von Hand durchgeführt werden, und zwar muss die Einstellung für die Anordnung der Eckpunkte der Polygonflächen auf ‚gegenuhrzeigersinn' (counterclockwise) geändert werden, da das Modell sonst nur von der Unterseite her sichtbar ist. Die Datei von Simply3D ist stark zerstückelt und unübersichtlich, so dass ein Bearbeiten des Quelltextes in einem Editor kaum möglich ist. Der grösste und entscheidende Nachteil von Simply3D ist jedoch, dass die Daten fehlerhaft in die wrl-Datei gespeichert werden. Wird die Datei in einem Browser geladen, so tritt eine Fehlermeldung, verursacht durch diese fehlerhaften Koordinatenwerte gefunden wurden. Tatsächlich findet man in der Koordinatenliste Zeilen, in denen Nachkommastellen von Koordinaten fehlen und durch einen Doppelpunkt ersetzt sind, was zu den Fehlermeldungen im Browser führt. Bevor man das Modell verwenden kann, müssen diese Doppelpunkte nun manuell entfernt werden; die Koordinaten müssen folglich als fehlerhaft angesehen werden. Was die Grösse der Datei betrifft, so schneidet Simply3D besser ab, was darauf zurückzuführen ist, dass bei den Koordinaten Nachkommastellen weggelassen werden (drei Nachkommastellen verglichen mit sechs bei Wcvt2pov).

Alles in allem erschien uns die fehlerhaften Simply3D-Dateien als zu schwerwiegend, was uns dazu bewog, die Umwandlung der dxf-Dateien mit dem Programm Wcvt2pov durchzuführen, gefolgt von einer Umwandlung ins Format VRML 2.0.

3.3 Umwandlung von VRML 1.0 nach VRML 2.0

Für die Umwandlung vom Format VRML 1.0 ins Format VRML 2.0 stand uns ein Programm auf DOS-Ebene namens vrml1to2 zur Verfügung. Aufgerufen wird das Programm mit vrml1to2 datei.wrl zieldatei.wrl.

3.4 Konstruktion von 3D-Elementen

3.4.1 Konstruktion der Schriften mit Simply3D

Die in den Modellen enthaltenen Schriften wurden allesamt im Programm Simply3D erstellt, wo ein eigenes Unterprogramm für diesen Zweck besteht.

3.4.2 Konstruktion der Häuser und Wälder (Modell Biel)

Für die Konstruktion der Häuser kam ein Programm zum Einsatz, das erlaubt, aus Vektordaten (z.B. Häuserkonturen, in unserem Fall digitalisiert aus der Landeskarte) und zusätzlichen Informationen (Höheninformationen) ein 3D-Modell zu generieren. Die Umrisse der Häuser von Biel wurden so als Vektoren mit einer Höhe versehen, und anschliessend auf das Geländemodell von Biel projiziert, damit die Position der Häuser auch auf das darunterliegende Gelände angepasst ist.

Die Waldkontur wurde ebenfalls digitalisiert. Auf die entstandene Fläche wurde zuerst ein unregelmässiges Raster gelegt, dann auf das Geländemodell projiziert und speziell gefertigte 3D-Bäume an die Stelle der unregelmässigen Stützpunkte eingefügt. Auf diese Weise entstand unser Waldmodell.

Anstelle der selbst digitalisierten Vektordaten könnten auch die VECTOR25-Daten verwendet werden.

3.4.3 Konstruktion der Arteplage (Modell Biel)

Für die Konstruktion eines Modells der Arteplage in Biel standen uns leider keine Daten zur Verfügung, sondern nur die Bilder der EXPO-01-Homepage, die auch der Öffentlichkeit zugänglich sind. Die momentan im Modell Biel enthaltene Arteplage ist ein sehr rudimentäres Modell, welches anhand Daten erstellt wurde, welche wir selber aus Abbildungen der Arteplage entnommen haben. So konnte eine Grundrissdarstellung der Arteplage verwendet werden, um die Koordinaten der Grundrissebene auszulesen. Die Höheninformationen wurden anschliessend anhand anderer Aufnahmen grob geschätzt. Mit Hilfe der so erhaltenen Koordinaten entstand das momentan aktuelle Modell, wobei zu hoffen wäre, in einem späteren Zeitpunkt auf aktuelle Daten der EXPO-01 zugreifen zu können.

3.5 Weitere 3D-Visualisierungsprogramme

3.5.1 Viscape

Abbildung 2: Visualisierungsformat Viscape

In der Startphase unseres Projekts testeten wir auch andere 3D Internet Browser Software. Viscape ist ein Plug-in ähnlich wie CosmoPlayer. Es visualisiert 3D Modelle die mit einem der drei Programmen VRT, 3D Webmaster oder Do3D programmiert wurden. Der Vorteil dieser Modellierungssprache liegt in animierten Objekten. Ohne grossen Aufwand können bewegte Objekte wie zum Beispiel bewegte Fahrzeuge oder gehende Personen integriert werden.

Aufgrund der grösseren Verbreitung vom Plug-in CosmoPlayer und der Anforderung im Projekt Expo.01 entschieden wir uns gegen Viscape und für VRML.

4. Das Dateiformat VRML 2.0

4.1 Allgemeiner Aufbau

4.1.1 Dateistruktur

VRML dient der Beschreibung kompletter Szenen oder Teilen, welche in andere Szenen eingefügt werden können. VRML-Dateien sind vereinbarungsgemäss mit der Endung ‚.wrl' gekennzeichnet und beinhalten lesbaren Text. Diese Sprache gleicht nicht unbedingt herkömmlichen Programmiersprachen. Zum Beispiel wird man vergeblich nach Ablaufstrukturen suchen. Sie wird zudem nicht übersetzt, sondern vom darstellenden Rechner abgearbeitet und direkt wiedergegeben.

Die erste Zeile einer VRML-Datei muss ein Kommentar enthalten mit der VRML-Version sowie dem verwendeten Zeichensatz. Der Kommentar wird mit dem Zeichen # eingeleitet und besitzt folgende Form:
#VRML V2.0 utf8.

Ein wichtiger Punkt, der zu beachten ist, ist die Unterscheidung von Gross- und Kleinschreibung. Zudem dürfen einige Schlüsselwörter nicht als Namen verwendet werden, es handelt sich dabei um Wörter, die zum Syntax von VRML gehören:
DEF, EXTERNPROTO, FALSE, IS, NULL, PROTO, ROUTE, TO, TRUE, USE, eventIn, eventOut, exposedField, field.

Da die im ASCII-Format geschriebenen Dateien sehr umfangreich werden können, sind von verschiedenen Seiten Kompressionsverfahren im Gespräch, wobei jedoch noch nichts beschlossen ist.

4.1.2 Knoten

Arten von Knoten

Eine VRML-Datei kann eine Menge von Objekten oder Knoten enthalten, sowie Prototypen und Routen. Knoten können ihrerseits wieder andere Knoten enthalten, was eine hierarchische Struktur ergibt.

Als Gruppenknoten bezeichnet man Knoten, die eine beliebige Anzahl Kindknoten enthalten. Diese Kindknoten können selber auch Gruppenknoten oder Blattknoten sein (z.B. Gestaltungsknoten, Lichtquellen und Blickpunkte). Diese können zwar keine Kindknoten mehr enthalten, dafür sogenannte untergeordnete Knoten (z.B. geometrische Knoten, Materialeigenschaften). Diese untergeordneten Knoten können jedoch nur innerhalb der Blattknoten aufgetreten. Die Definition von Knotentypen ist in der Spezifikation von VRML 2.0 enthalten, weitere Knotentypen können in der Form von Prototypen in einer VRML-Datei definiert und dann weiter verwendet werden.

Felder

Zur Beschreibung der Eigenschaften von Knoten werden Felder (fields) verwendet, so beschreibt zum Beispiel das einzige Feld des Knotens Sphere den Radius der Kugel: `Sphere { radius 2 }`. Die Angabe der Felder erfolgt nach dem Knotentyp in geschweiften Klammern, wobei jeweils ein Feldname und mindestens ein Wert angegeben wird.

Um die Übersichtlichkeit der VRML-Datei zu steigern, wird die Beschreibung eines Knotens oft über mehrere Zeilen verteilt, man rückt die Felder des Knotens ein und positioniert Anfang und Ende eines Knotens übereinander, was dann folgendermassen aussieht:

```
Sphere {
   radius 2
}
```

Mehrere Felder eines Knotentyps werden zeilenweise übereinander geschrieben, die Reihenfolge ist dabei nicht festgelegt. Wird ein Feld eines Knotens nicht explizit aufgeführt, so wird eine Standardeinstellung (Default-Wert) verwendet. Jedes Feld hat einen bestimmten Feldtyp, wobei zwischen Feldern unterschieden wird, die entweder nur einen Wert (Feldtyp fängt mit SF für single-valued field an) oder beliebig viele Werte enthalten können (Feldtyp beginnt mit MF für multiple-valued field). Werden für ein MF-Feld mehrere Werte angegeben, so müssen sie in eckige Klammern eingeschlossen und durch Kommas getrennt werden.

Ereignisse und Routen

Viele Knoten besitzen noch die Fähigkeit, verschiedene Ereignisse (events) empfangen oder senden zu können, was die Kommunikation zwischen den Knoten ermöglicht. Auf diese Weise können auch Daten übertragen werden. Ereignisse enthalten immer einen Wert und eine Zeitangabe, die als Zeitmarke (timestamp) bezeichnet wird. Durch empfangene Ereignisse können Felder geändert werden (z.B. ändert **set_position** das Feld **position**), oder umgekehrt können gesendete Ereignisse die Änderung von Feldern mitteilen (**position_changed** teilt die Änderung des Feldes **position** mit).

Die Schlüsselwörter field, exposedField, eventIn und eventOut geben an, wie auf Felder und Ereignisse eines Knotens zugegriffen werden kann:

- field: Zu diesem Feld gehören keine Ereignisse; bei der Definition des Knotens können Wertangaben erfolgen, die den voreingestellten Wert überschreiben.
- eventIn: Ein Ereignis kann empfangen werden; das Ereignis kann Ziel in einer ROUTE-Anweisung sein und auch durch einen Script-Knoten geändert werden.
- eventOut: Ein Ereignis kann erzeugt werden; das Ereignis kann Ausgangspunkt in einer ROUTE-Anweisung sein, und sein Wert kann durch einen Script-Knoten gelesen werden.
- exposedField: Kombiniert alle Eigenschaften der drei anderen Zugriffsarten.

Die Kommunikation zwischen zwei bestimmten Knoten erfolgt dadurch, dass die vom ersten Knoten erzeugten Ereignisse dem zweiten Knoten zugeleitet werden. Die Verbindung zwischen dem Knoten, der ein Ereignis sendet, und dem Knoten, der dieses Ereignis empfängt, wird als Route bezeichnet.

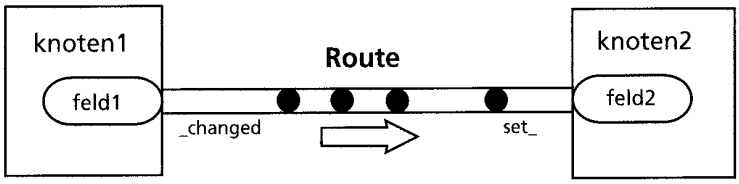


Abbildung 3: Schematische Darstellung einer Route-Anweisung

Die Anweisung beginnt mit dem Schlüsselwort ROUTE und hat folgende Form:

```
ROUTE knoten1.feld1_changed TO knoten2.set_feld2
```

Die Anhänge ‚_changed' und ‚set_' sind empfohlene Konventionen, aber keine strengen Gesetze. Die ROUTE-Anweisung wird meistens auf der obersten Knotenebene angegeben, und zwar gewöhnlich am Ende nach allen Knoten.

4.1.3 Untergeordnete Knoten

Viele Knoten können untergeordnete Knoten enthalten, die nur dort erscheinen können, also nicht als selbständige Knoten verwendet werden können. Beispiele dafür sind Knoten für die geometrische Struktur (**Shape**) oder die Oberflächenbeschaffenheit (**Appearance**). Beide Knoten können wiederum untergeordnete Knoten enthalten.

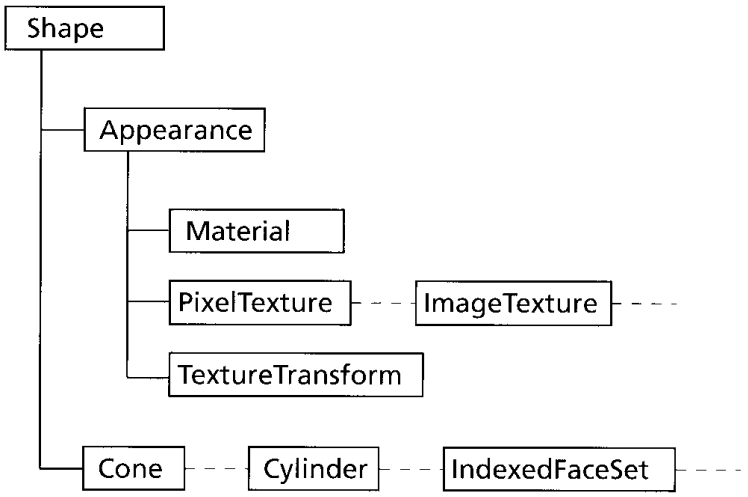


Abbildung 4: Knotenstruktur

4.1.4 Gruppenknoten

Als Gruppenknoten werden Knoten bezeichnet, die viele andere Knoten zu einer Gruppe zusammenfassen, sie können in ihrem **children**-Feld andere Knoten als Kindknoten enthalten. Folgende Knoten können als Gruppenknoten verwendet werden: **Group**, **Transform**, **LOD** und **Switch**. Innerhalb des Gruppenknotens spielt die Reihenfolge, in der die enthaltenen Knoten und Felder auftreten, keine Rolle, dies im Gegensatz zu VRML 1.0, wo die Reihenfolge noch von Bedeutung war.

4.1.5 Gestaltung sichtbarer Objekte

Alle sichtbaren Objekte bestehen einerseits aus der geometrischen Struktur, der noch ein bestimmtes Aussehen (Farbe, Transparenz, Oberflächenstruktur) zugeordnet wird. Zusätzlich sind noch Lichtquellen nötig, damit für den Betrachter überhaupt etwas sichtbar wird.

4.1.5.1 Geometrische Struktur

Was die geometrische Struktur betrifft, so gibt es vordefinierte geometrische Knoten wie Kugel (**sphere**), Quader (**box**), Kegel **(cone**) oder Zylinder (**cylinder**). Zum Konstruieren von virtuellen Welten sind aber auch beliebig strukturierte Gebilde nötig, was mit Hilfe vermaschter Polygone erfolgt. Zu diesem Zweck existiert der geometrische Knoten **IndexedFaceSet**. Die Eckpunkte werden in Form einer Liste im Feld **point** des untergeordneten Knoten **Coordinate** aufgeführt und dort fortlaufend numeriert, wobei der erste Punkt den Index Null erhält. Auf dieser Basis können nun Polygone definiert werden. Der **IndexFaceSet**-Knoten verfügt über eine Anzahl weiterer Felder, die ihrerseits wieder untergeordnete Knoten enthalten können, wie dies beim **coord**-Feld der Fall ist. Diese Knoten dienen der Beschreibung der geometrischen Eigenschaften des geometrischen Knotens, zu ihnen gehören ausser den Koordinaten noch Farben, Normalen auf die Polygonflächen und Koordinaten für Texturen. Diese Auslagerung der geometrischen Eigenschaften ist Voraussetzung für die mehrfache Nutzung der enthaltenen Daten durch mehrere geometrische Knoten. Für eine genauere Beschreibung des IndexedFaceSet-Knotens wird hier auf Kapitel 4.2.8 (IndexedFaceSet) verwiesen.

Neben den vordefinierten geometrischen Knoten und dem **IndexedFaceSet**-Knoten gibt es noch einige andere geometrische Knoten, die jedoch nur selten verwendet werden. Zur Darstellung von Punktmengen wird der **PointSet**-Knoten verwendet, wobei jedem Punkt durch Verwendung eines **Color**-Knotens eine eigene Farbe zugewiesen werden kann. Die Koordinaten werden dabei in einem **Coordinate**-Knoten angegeben. Analog zum **IndexedFaceSet**-Knoten gibt es noch den **IndexedLineSet**-Knoten, wo nur Linienzüge (Polylinien) dargestellt werden, denen noch Farben zugewiesen werden können. Der **ElevationGrid**-Knoten eignet sich besonders für die Darstellung von Geländeformationen, wobei die Berechnung der Fläche aus Höhenwerten über einem rechteckigen Raster erfolgt. Der Nachteil dabei ist, dass die Höhen in einem regelmässigen Raster bekannt sein müssen. Der Fläche können dann nachträglich noch Farben oder Texturen zugewiesen werden. Eine weitere Möglichkeit zur Erstellung eines Körpers besteht darin, ein ebenes Flächenstück entlang einer Linie durch den Raum zu bewegen, um einen sogenannten Translationskörper zu erhalten, der **Extrusion**-Knoten bietet diese Möglichkeit. Der **Text**-Knoten enthält Text in Form von einer oder mehreren Zeichenfolgen; die dazugehörigen Darstellungseigenschaften werden im untergeordneten **FontStyle**-Knoten beschrieben.

4.1.5.2 Aussehen

Für das äussere Erscheinungsbild ist im **Shape**-Knoten das **appearance**-Feld vorhanden. Hier kann ein **Appearance**-Knoten angegeben werden, in dessen

Feldern die Materialeigenschaften oder eine Textur angegeben werden können. Für die Materialeigenschaften ist wiederum ein eigener Unterknoten vorhanden (**Material**), der verschiedene Felder für Reflexion (wovon es mehrere Arten gibt), Glanz, Transparenz sowie Eigenleuchten enthält. Die Koeffizienten der Reflexion bestimmen die Farbe des geometrischen Knoten.

4.1.5.3 Texturen

Durch Überlagerung der Polygonflächen mit einem vorhandenen Bild können kompliziertere Oberflächen simuliert werden und so das Berechnen aufwendiger Polygonflächen vermieden werden. Dafür stehen verschiedene Knotentypen zur Verfügung. Der einfachste Fall ist der Knotentyp **PixelTexture**, bei dem ein Feld **image** die Pixel-Struktur direkt enthält. Mit dem Knotentyp **ImageTexture** können komplette Graphik-Dateien im Format JPEG, GIF oder PNG direkt verwendet werden. Der Knoten enthält dann ein **url**-Feld, in dem auf die Datei verwiesen wird. Ausser Bildern und Bitmustern können auch Videos (müssen im MPEG1-Format vorliegen) als bewegte Texturen eingefügt werden, dies erfolgt mit dem **MovieTexture**-Knoten.

Bei Texturen wird ein zweidimensionales Koordinatensystem mit den Koordinaten u und v verwendet. Die u-Achse verläuft waagrecht am unteren Rand einer Textur, die v-Achse senkrecht am linken Rand, jeweils vom Wert 0 bis 1. Auf der Oberfläche eines beliebigen geometrischen Objektes wird ein ebenfalls zweidimensionales Koordinatensystem verwendet, die Koordinaten s und t werden aus den 3D-Koordinaten x, y und z berechnet und 1:1 auf die Koordinaten u und v der Textur abgebildet. Liegen s und t ausserhalb des Intervalls 0 und 1, so kann angegeben werden, ob sich die Textur wiederholen soll (wie eine Kachel). Mit Hilfe der Knotentyps **TextureTransform** kann die Textur noch verschoben, gedreht und skaliert werden.

4.1.6 Transformationen

Als Bezugssystem wird ein dreidimensionales, rechtshändiges, kartesisches Koordinatensystem verwendet, in welchem die x-Achse horizontal nach rechts, die y-Achse senkrecht nach oben und die z-Achse horizontal nach vorne verläuft. Als Standardeinheit wird für Distanzen Meter verwendet, Winkel werden im Bogenmass angegeben. Folgende Transformationen können auf die Knoten angewendet werden:

- Verschiebung (Translation)
- Rotation
- Skalierung (Massstabsänderung)

Die Transformationen wirken auf die Koordinaten der in den Knoten enthaltenen Punkte. Sie werden im Transform-Knoten angegeben, und wirken auf alle Kindknoten, die im Transform-Knoten enthalten sind. Als Kindknoten können auch weitere Transform-Knoten verwendet werden, so können mehrere Transformationen ineinander verschachtelt werden; sie werden dann von unten nach oben ausgeführt.

4.1.7 Beleuchtung und Material

4.1.7.1 Lichtquellen

Oft sind in einer Szene mehrere Lichtquellen vorhanden, die Beleuchtung der Objekte ergibt sich dann aus der Summe dieser Lichtquellen. Es stehen verschiedene Arten von Lichtquellen zur Verfügung: gerichtetes Licht (DirectionalLight), Punktlicht (PointLight), Spotlicht (SpotLight) sowie ambientes Licht, das nicht durch einen Knoten produziert wird wie die anderen drei, sondern das sich aus gestreuten Anteilen der anderen Lichtquellen zusammensetzt.

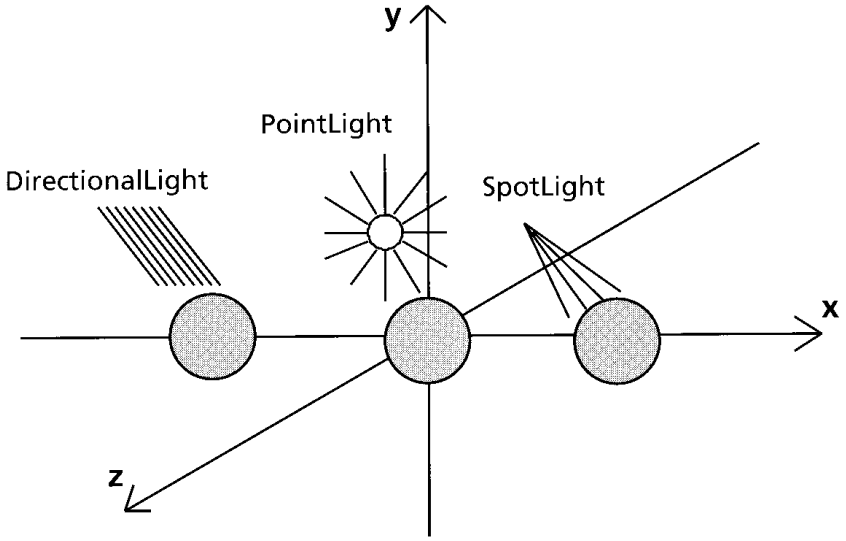


Abbildung 5: Verschiedene Arten von Lichtquellen

Bei jeder Lichtquelle kann im Feld ambientIntensity angegeben werden, wie stark sie sich am gesamten ambienten Licht beteiligt. Das Feld on ist ein weiteres Feld, das allen Lichtquellen gemeinsam ist. Durch Zuweisen des Wertes TRUE/FALSE kann das Licht ein- und ausgeschaltet werden. Punktlicht und Spotlicht wirken nur innerhalb einer bestimmten Reichweite, das gerichtete Licht wirkt dagegen nur auf Objekte innerhalb der eigenen Gruppe, dies aber unabhängig von der Entfernung.

4.1.7.2 Materialeigenschaften

Die Materialeigenschaften werden im Knoten Material festgelegt, der ein Unterknoten des Appearance-Knoten darstellt. Folgende Materialeigenschaften können dabei verändert werden:

- Diffuse Reflexion: Das Licht wird von einer Fläche mit gleicher Intensität in alle Richtungen reflektiert.
- Spiegelnde Reflexion: Glänzende Flächen erzeugen eine gerichtete Reflexion des Lichtes, wobei gilt: Einfallswinkel = Ausfallswinkel (bezogen auf die Normale). Durch Angabe eines Wertes für den Glanz kann noch bestimmt werden, wie stark die Intensität des reflektierten Lichtes abnimmt, wenn das Objekt nicht exakt unter dem Reflexionswinkel betrachtet wird.

- Ambiente Reflexion: Dieser Wert gibt an, wie gross der Anteil der ambienten Beleuchtung ist, der reflektiert wird. Die Grundfarben werden dabei von der diffusen Reflexion übernommen.
- Transparenz: Der Anteil des auftreffenden Lichtes, der eine Fläche durchdringt, wird mit diesem Wert bestimmt. Bei einem Wert von 0 ist die Fläche undurchsichtig, bei einem Wert von 1 total durchsichtig.
- Eigenleuchten: Jedes geometrische Objekt kann selber Licht aussenden (Glüheffekt), deren Licht wirkt jedoch nicht auf andere Objekte.

4.1.8 Blickpunkte

Die Wiedergabe des Szene ist stark von der Position des Betrachters sowie seiner Blickrichtung abhängig. In VRML 2.0 wird mit Blickpunkten (Viewpoint) gearbeitet, die eine Position in der Szene und eine Orientierung enthalten, zusätzlich wird der dargestellte Ausschnitt der Szene mittels einem Blickwinkel zwischen 0 und 180 bestimmt.

In einer Szene können beliebig viele Blickpunkte gesetzt werden, so können zum Beispiel interessante Ansichten des Objekts abgespeichert und dann beliebig abgerufen werden. Normalerweise wird beim Laden der erste Viewpoint der Liste verwendet, ausser wenn einer der Viewpoints mittels DEF durch einen Namen gekennzeichnet ist. Haben mehrere einen Namen, wird wiederum der erste der Liste verwendet. Diese Liste wird dann im Browser zur Auswahl geboten.

Der aktuelle Blickpunkt kann automatisiert durch die Szene bewegt werden, Hierzu muss durch Ereignisse entweder die Position beziehungsweise die Orientierung geändert werden (siehe Kapitel 1.13 über Animation).

4.1.9 Mehrfachverwendung von Knoten

Der Umfang der VRML-Datei kann dadurch verkleinert werden, dass mehrfach auftretende identische Knoten nur einmal definiert werden, und dann später dieser existierende Knoten einfach wieder aufgerufen wird. Durch Voranstellen des Schlüsselworts DEF wird dem Knoten ein Name zugewiesen; was dann wie folgt aussieht:

```
DEF name knotentyp {felder, ereignisse}
```

Diese Knoten können dann beliebig oft verwendet werden, indem auf ihren Namen Bezug genommen wird mit dem Schlüsselwort USE. Diese weiteren Knoten zeigen nur auf das Original, es werden keine Kopien davon erstellt. Meistens wird dem Aufrufen des vorhandenen Knoten noch ein Transform-Knoten vorangestellt, da das Objekt ja nicht am selben Ort zu stehen kommen soll. Wird ein Name mehrfach mittels DEF vergeben, so wird jeweils auf das zuletzt definierte Objekt zurückgegriffen.

4.1.10 Prototypen

Die Menge der Knotentypen kann innerhalb einer Datei durch die Definition von Prototypen erweitert werden. Ein Prototyp enthält die Beschreibung einer Mini-Szene auf der Basis von bereits existierenden Knoten und Prototypen sowie gegebenenfalls auch Routen. Durch die Definition eines Prototyps erscheint das Objekt jedoch noch nicht in der Szene, dies erfolgt erst, wenn der Prototyp wie ein Knotentyp in der VRML-Datei verwendet wird. Die Definition eines Prototyps kann sich auch in externen Dateien befinden; öfter verwendete Prototypen können so zentral gehalten werden.

4.1.11 Level of Detail (LOD)

Dieser Gruppenknoten fasst mehrere Knoten zusammen, die dann in Abhängigkeit von der Entfernung zum Betrachter einzeln wiedergegeben werden. So können mehrere Modelle des gleichen Objekts definiert werden, die sich einzig in der Detailtreue unterscheiden. Bei der Wiedergabe der Szene kann so ein Effekt wie im richtigen Leben simuliert werden: Je näher man sich dem Objekt nähert, desto mehr Details werden erkennbar. Zusätzlich können dadurch die Ladezeiten verringert werden, solange sich der Betrachter noch in grossem Abstand vom Objekt befindet.

4.1.12 Verknüpfung mit dem WWW

Gewisse Knoten können eine Verbindung zum Internet (Hyperlink) beinhalten. Der Knotentyp Anchor, der zu den Gruppenknoten gehört, enthält ein Feld url, das einen Verweis auf ein Dokument enthält, wobei sich das Dokument sowohl im Internet wie auch auf der eigenen Festplatte befinden kann. Der Hyperlink eines Anchor-Knotens wird ausgeführt, sobald einer seiner Kindknoten angeklickt wird.

```
Anchor {
    url "http://www.testadresse.com/beispiel"
    parameter "target=frame3"
}
```

Je nach Einstellungen wird der Hyperlink im selben oder in einem neuen Fenster geöffnet, ferner kann in der neuen Szene ein Blickpunkt angegeben werden, der beim Laden aktiv wird, indem folgende Form verwendet wird: „url#viewpointName“. Enthält der Anchor-Knoten einen URL, der nur aus dem Namen eines Blickpunktes besteht (zum Beispiel „#Aussicht3“), so bezieht er sich auf die aktuelle Szene; wird ein Kindknoten angeklickt, so wird der angegebene Blickpunkt geladen.

Es ist nicht nur möglich, zu einem Dokument im Internet zu wechseln, sondern auch den Inhalt einer VRML-Datei in die aktuelle Szene einzufügen. Dafür steht der Knotentyp Inline zur Verfügung. Auch dieser Knotentyp enthält ein url-Feld, aus dem ein URL auf die VRML-Datei weist.

4.1.13 Animation eines Objektes

Einfache Animationen können durch die Kombination von geeigneten Knoten ausschliesslich mit den Mitteln von VRML 2.0 (also als reine VRML-Lösung) realisiert werden, komplexere Animationen dagegen erfordern den Einsatz von Script oder Programmen in höheren Programmiersprachen.

Eine Bewegung kann beispielsweise dadurch erreicht werden, dass Vektoren, welche die Koordinaten von Punkten enthalten, in Abhängigkeit von der Zeit geändert werden. Normalerweise wird der TimeSensor-Knoten verwendet, der dann die Zeitimpulse für die Animation liefert. Mit Hilfe dieser Zeitwerte können aus vorgegebenen Vektoren laufend neue Vektoren berechnet werden. Dabei werden die Zeitwerte als Argumente zur Interpolation zwischen Vektoren an vorgegebenen Stützstellen verwendet. Für die Interpolation stehen verschiedene Knoten zur Verfügung. Für Vektoren könne sowohl der CoordinateInterpolator als auch der PositionInterpolator verwendet werden. Die Ergebnisse der Interpolation können dann als Ereignisse an Transform-Knoten oder an Knoten mit veränderbaren Koordinaten übertragen werden. Gesteuert wird die Übertragung der Ereignisse durch ROUTE-Anweisungen, wobei die beteiligten Knoten Namen tragen müssen (Zuweisung der Namen erfolgt durch Schlüsselwort DEF).

Eine Animation muss sich nicht ausschliesslich auf die Position beziehen, so kann eine Animation auch darin bestehen, dass sich die Orientierung (Blickrichtung des Betrachters) oder die Farbe eines Objektes ändert. Dafür sind ein OrientationInterpolator und ein ColorInterpolator vorhanden.

4.1.14 Sensoren und Interaktion

Sensoren sind immer an geometrische Knoten gebunden, und ermöglichen eine Interaktion des Betrachters mit dem Objekt. Die Interaktion kann dabei mit einem Zeigegerät wie der Maus erfolgen, indem ein Objekt angeklickt oder mit der Maus in die Nähe eines Objekts navigiert wird; andere Sensoren liefern Ereignisse, wenn der Betrachter in einen vordefinierten Bereich der Szene eindringt oder wenn ein bestimmter Bereich für ihn sichtbar wird.

Der TouchSensor-Knoten kann dazu verwendet werden, eine Lichtquelle ein- und auszuschalten, oder eine Animation zu starten. Bei der reinen Berührung des Objekts reagiert der Sensor durch Senden eines Ereignisses isOver mit dem Wert TRUE, wird das Objekt dann angeklickt folgt ein Ereignis isActive, ebenfalls mit dem Wert TRUE. Solange isOver den Wert TRUE zurückgibt, werden gleichzeitig auch Ereignisse geliefert, die Punkte und Normalen von der Oberfläche des berührten Knotens enthalten.

Der ProximitySensor ist eine Art Bewegungsmelder, der ein quaderförmiges Gebiet überwacht. Dringt der Betrachter in dieses Gebiet ein, so wird sendet der Sensor ein Ereignis isActive mit dem Wert TRUE sowie weitere Ereignisse mit Angaben über die Position und die Orientierung, wenn sich der Betrachter weiterbewegt.

Das Ziehen (drag) und Drehen von sichtbaren Objekten kann mit Hilfe von Zeigegerät-Sensoren oder Drag-Sensoren realisiert werden. Die Bewegung des Zeigegerätes, meistens der Maus, wird durch diese Sensoren in kontinuierlich gesendete Ereignisse mit Positionswerten oder Rotationswerten umgesetzt. Diese Ereignisse können dann direkt per ROUTE-Anweisung an den Transform-Knoten geleitet werden, der die zum Sensor gehörigen geometrischen Knoten enthält. Die erstellt eine direkte Verbindung zwischen Maus- und Objektbewegung, und das Objekt macht so jede Bewegung der Maus mit.

4.1.15 Script-Knoten, Scripts und Programme

4.1.15.1 Aufgaben

Oft reichen die Möglichkeiten von VRML 2.0 nicht aus, um komplexere Bewegungsabläufe zu berechnen, oder werden für die Kommunikation mit anderen Rechnern weitergehende Funktionen benötigt. Dies kann mit Hilfe des Script-Knoten erfolgen, worin ein Programm enthalten ist, eine Folge von Anweisungen in einer Script-Sprache, die vom VRML-Browser interpretiert werden können. Das Programm besteht aus Ausweisungen einer Programmiersprache, die aber vor der Anwendung übersetzt oder compiliert werden.

Aufgaben, die mit Hilfe von Script-Knoten gelöst werden können:

- Empfangene Ereignisse verarbeiten, und je nach Ergebnis weitere Ereignisse versenden.
- Ausführlicherer Berechnungen durchführen für das Verhalten von Objekten und für Animationen.
- Zusammengesetzte Werte in ihre Komponenten zerlegen oder aus Komponenten zusammensetzen.
- Leistungen des VRML-Browsers nutzen, um zum Beispiel weitere Knoten in eine Szene einzufügen oder eine neue Szene zu laden.
- Kommunikation mit Servern im Internet über TCP/IP-Protokolle durchführen.

4.1.15.2 Aufbau

Innerhalb eines Script-Knotens können beliebig viele Felder und Ereignisse definiert werden, wobei die Felder zur Speicherung von Daten und die Ereignisse zum Transfer von Werten zwischen Script und Programm und den Knoten der Szene dienen. Analog zu den normalen Knoten können zu und von den Ereignissen Routen definiert werden, um Werte weiterzugeben oder Werte zu empfangen. Zusätzlich verfügen Script-Knoten noch über direktere Möglichkeiten, um mit anderen Knoten zu kommunizieren.

Es können innerhalb des Script-Knotens nur Sprachen verwendet werden, die der VRML-Browser unterstützt. Zurzeit sind das die Sprachen JavaScript und Java, dazu kommen noch C/C++, Perl, TCL, oder VisualBasic. Die Einbindung von einem Programm in einen Script-Knoten erfolgt über dessen Feld url. Ein Script. das im url-Feld enthalten ist, beginnt immer mit der Bezeichnung der Sprache. Externe Dateien müssen durch bestimmte Dateinamen-Extensions gekennzeichnet sein.

Scripts und Programme können lokale Variablen enthalten, um Werte zwischenzuspeichern, jedoch sind diese Werte bei einem späteren Aufruf des Programms nicht mehr vorhanden, es bleiben nur Werte gespeichert, die in den Feldern eines Script-Knotens gespeichert werden.

4.1.15.3 Ereignis-Verarbeitung

Die Ausführung des Programms wird erst gestartet, sobald der Script-Knoten, der das Programm enthält, ein Ereignis zugewiesen erhält. Zur Verarbeitung der Ereignisse gibt es im Programm Funktionen oder Methoden. Bei JavaScript muss für jedes Ereignis eine eigene Funktion erstellt werden, die denselben Namen wie das Ereignis trägt. Der Aufbau eines Script-Knotens mit einem darin enthaltenen Script kann beispielsweise wie folgt aussehen:

```
Script {
    eventIn      SFTime      pushButton
    eventOut     SFInt32     pushPop
    url "javascript:
        function pushButton() {
            if (pushPop == 1)
            pushPop = 0;
            else
            pushPop = 1;
    }"
}
```

Befindet sich das obige Script in einer externen Datei 'button.js', so hat der Script-Knoten folgende Form:

```
Script {
    eventIn      SFTime      pushButton
    eventOut     SFInt32     pushPop
    url "http://www.beispiel.com/button.js"
}
```

In beiden Fällen kann vom Script-Knoten in dem Beispiel ein Ereignis pushButton empfangen werden, wodurch die gleichnamige Funktion gestartet wird.

4.1.15.4 Zugriff auf Knoten

Vom Script-Knoten empfangene eventIn-Ereignisse werden an die gleichnamigen Funktionen des Scripts übergeben und verarbeitet. Allenfalls vorhandenen eventOut-Ereignissen kann von einer Funktion einen Wert zugewiesen werden, wobei das Senden erst erfolgt, wenn die Funktion beendet ist. Erfolgen mehrere Zuweisungen an dasselbe eventOut-Ereignis innerhalb der Funktion, so wird nur der letzte Wert gesendet.

Der Zugriff von Scripts und Programmen auf die Felder und Ereignisse anderer Knoten kann erfolgen, sofern sie dem Script-Knoten untergeordnet sind. Hierzu müssen die Knoten entweder in einem seiner Felder mit dem Feldtyp SFNode oder MFNode vereinbart werden, oder es muss aus einem solchen Feld eine Referenz mit USE auf einen benannten Knoten hergestellt werden. Um den direkten Zugriff auf diese Knoten zu ermöglichen, muss der Wert des Feldes directOutput auf TRUE gesetzt werden.

```
DEF Verschieb Transform {}
Script {
    field SFNode knoten USE Verschieb
    eventIn SFVec3f position
    directOutput TRUE
    url "javascript:
        function position (wert) {
            knoten.set_translation = wert;
        }"
}
```

In diesem Beispiel wird ein Transform-Knoten verwendet, in dessen translation-Feld das Script im Script-Knoten einen Wert schreibt, den er selber per Ereignis position erhält; die Übergabe des Wertes erfolgt implizit durch ein Ereignis an den Transform-Knoten. Bei einer solchen Referenz auf einen anderen Knoten kann nicht auf einfache Felder zugegriffen werden, es können nur Felder mit der Zugriffsart exposedField verwendet werden, die sowohl ein Lesen wie ein Schreiben ermöglichen. Auf eventIn-Felder kann nur geschrieben werden, eventOut-Felder lassen nur ein Lesen zu.

4.2 Beschreibung der wichtigsten Knoten

In diesem Kapitel sollen einige der wichtigsten Knoten erklärt werden, die in unserem Projekt verwendet wurden. Die Beispiele beziehen sich somit auch immer auf eines unserer Modelle, um die Theorie zu veranschaulichen.

4.2.1 Anchor

Der Anchor-Knoten ladet den Inhalt einer URL, sobald ein Objekt aktiviert (angeklickt) wird, dass sich innerhalb des Anchor-Knotens befindet. So können Verknüpfungen zu anderen Objekten oder Internet-Seiten erstellt werden, wobei sich der Link auf einem Objekt befindet.

```
DEF Arteplage Anchor {
    description "Arteplage Biel"
    url "http://www.expo-pk.ch/de/art/biel/index.html"
    children [
        Inline {url „arteplage.wrl“}
    ]
}
```

Hier wird ein Anchor namens ‚Arteplage‘ definiert. Als erstes wird unter **children** das Objekt eingefügt, auf das sich der Anker beziehen soll. Das Feld **description** bietet die Möglichkeit, zusätzliche Informationen zum Anker aufzuführen, die in der Fusszeile des Browsers erscheint, sobald sich der Mauszeiger über dem Objekt befindet. Das Feld **url** beinhaltet die eigentliche Verknüpfung zu einer anderen Datei, sei es nun eine Internetseite, eine VRML-Datei oder eine andere Datei. In der Form url{#nord} springt der Browser zum Viewpoint ‚nord‘ in der gleichen Datei. Zusätzlich existiert ein Feld **parameter**, wo zusätzliche Information angegeben werden kann, die dann vom Browser interpretiert wird (zum Beispiel, wenn die Zieldatei in einem neuen Fenster geöffnet werden soll).

4.2.2 Appearance

Der Appearance-Knoten beschreibt die visuellen Eigenschaften eines Objekts, wie das Material und die Oberflächenbeschaffenheit, Texturen, die dem Modell überlagert werden können.

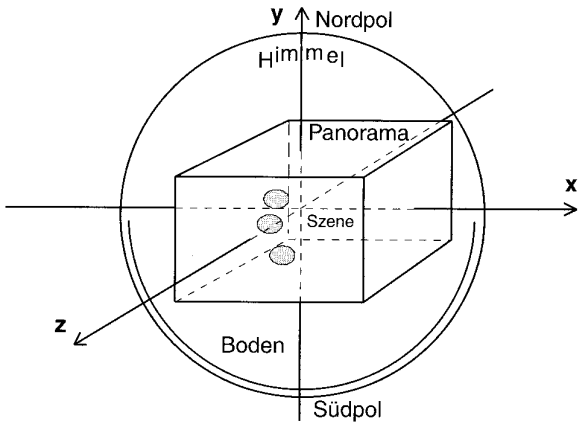
```
appearance Appearance {
    material Material {
        ambientIntensity 0.0386667
        diffuseColor     0.55 0.47 0.14
        specularColor    0.3 0.3 0.3
        shininess        0.1
    }
    texture ImageTexture {
        url "see.gif"
    }
    textureTransform TextureTransform {
        scale 0.985 -1.35
    }
}
```

Das Feld **material** sollte einen Knoten des Typs 'Material' enthalten. Ist das **material**-Feld NULL, werden alle Beleuchtungselemente ignoriert und die Farbe des Objektes wird auf (1,1,1) gesetzt. Ebenso wird nach **texture** ein Knoten der Texture-Gruppe erwartet, wobei wir in unserem Fall den ImageTexture-Knoten verwenden, um das Geländemodell mit dem Bild der Karte zu überlagern.

Das Feld ‚**textureTransform**' dient dazu, am zu überlagernden Bild Veränderungen wie zum Beispiel eine Massstabsänderung anzubringen.

4.2.3 Background

Mit Hilfe des Background-Knotens wird ein Hintergrund gestaltet, es können damit Boden und Himmel simuliert werden, oder es können beliebige Bilder als Hintergrundpanorama verwendet werden. Das Panorama wird durch bis zu sechs Bilder definiert, die auf die sechs Seiten eines unendlich grossen Würfels um die Szene abgebildet werden. Geladen werden die Bilder in den sechs Feldern **backUrl**, **bottomUrl**, **frontUrl**, **leftUrl**, **rightUrl** und **topUrl**. Der Hintergrund hat die Form von zwei Kugelschalen für den Boden und den Himmel, wobei der Himmel etwas hinter dem Boden verläuft. Beide Kugelschalen haben einen unendlich grossen Radius.




```
Background {
    groundColor [0.4 0 1, 0.4 0 1]
    groundAngle [1.5]
    skyColor [ 0.3 0.5 0.9, 0.3 0.5 0.9, 0.8 0.9 1, 0.8 0.9 1]
    skyAngle [1.3, 1.7, 2]
}
```

GroundColor gibt die Farbe des Bodens an für verschiedene Winkel zwischen Nadir (Punkt genau unterhalb des Benutzers) und Horizont. Die erste aufgeführte Farbe gilt für den Winkel, der im Nadir beginnt, und gilt bis zur ersten Richtung, die im Feld **groundAngle** aufgeführt ist. Die Werte unter **groundAngle** dürfen den Wert Pi/2 nicht überschreiten, und die Anzahl der Werte muss immer um eins kleiner sein als die Anzahl der Farben, die unter **groundColor** aufgelistet sind. Dasselbe gilt auch für die Felder **skyColor** und **skyAngle**, wobei hier die Farbverteilung im Zenit beginnt und beim Horizont endet.

4.2.4 Billboard

Der Billboard-Knoten wird verwendet, um zu erreichen, das ein Objekt (oder eine Gruppe von Objekten) immer in Richtung des Betrachters ausgerichtet ist; das lokale Koordinatensystem eines Objektes wird immer so ausgerichtet, dass die Z-Achse immer Richtung Betrachter zeigt. Die Schriften, welche die verschiedenen Orte in unseren Modellen bezeichnen, verwenden diesen Billboard-Knoten, so dass der Text immer lesbar ist, egal wo man sich befindet.

```
Billboard {
    children [
        Inline {url "text_Biel2.wrl"}
    ]
    axisOfRotation 0 0 0
}
```

Anstatt die Objekte, auf die sich der Billboard-Knoten beziehen soll, mit dem Feld **children** anzugeben, gibt es auch die Möglichkeit, eine Box zu definieren (**bboxCenter** und **bboxSize**) und somit alle Objekte innerhalb dieser Box einzubeziehen.

Das Feld ‚**axisOfRotation**' definiert eine Achse im lokalen Koordinatensystem des Objekts, um welche die Rotation ausgeführt wird (anstelle der Z-Achse wird so eine neue Achse definiert, die dann immer in Richtung des Betrachters zeigt.

4.2.5 DirectionalLight

Bei diesem gerichteten Licht kommen alle Lichtstrahlen aus der gleichen Richtung und sind parallel. Die wichtigsten Einstellungen, die vorgenommen werden können sind die Intensität sowie die Richtung. Das Licht bestrahlt alle Objekte, die sich in der gleichen Gruppe befinden wie das Licht selbst.

```
DEF Licht2 DirectionalLight {
    direction     -1 -1 -1
    intensity     0.4
    on            TRUE
}
```

Da die Lichtstrahlen parallel verlaufen (und somit eigentlich aus dem Unendlichen kommen), muss die Lichtquelle nicht fest im Raum plaziert werden, sondern es muss nur die Richtung angegeben werden, in welcher die Strahlen verlaufen (was im Feld **direction** erfolgt).

Weitere Einstellungen betreffen die Farbe des Lichtes (**color**) sowie die Intensität, wobei einerseits die direkte Emission des Lichts (**intensity**) sowie die Umgebungsabstrahlung (**ambientIntensity**) von Bedeutung ist. Die Einstellungen bestimmen, ob nur Objekte heller erscheinen, die direkt von einem Lichtstrahl getroffen werden, oder ob auch ein diffuser Lichteinfall erfolgt. Dies entspricht eher den wirklichen Verhältnissen. Das Feld **on** beschreibt, ob das Licht eingeschaltet ist oder nicht, und macht das Licht somit steuerbar und durch geeignete ROUTE-Anweisungen von ausserhalb ansprechbar.

4.2.6 Fog

Der Fog-Knoten ist eine Möglichkeit, um Nebel zu simulieren. Dies geschieht, indem Objekte in Abhängigkeit der Distanz mit derselben Farbe (natürlich meistens weiss) eingefärbt werden und so nicht mehr voneinander unterscheidbar sind.

```
Fog {
    color             1 1 1
    fogType           „LINEAR“
    visibilityRange   400
}
```

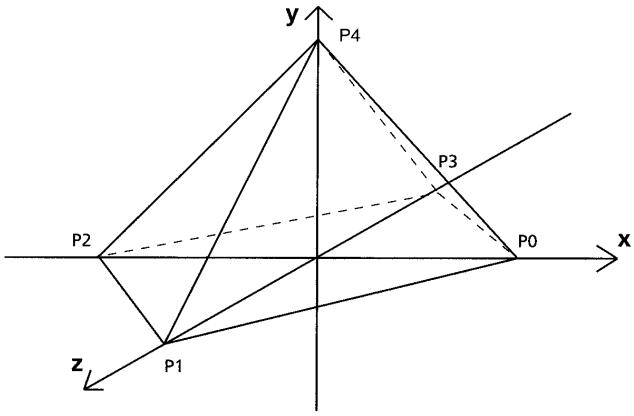
Das Feld **color** beschreibt die Farbe, mit der die Objekte eingefärbt werden (als Standard ist weiss gesetzt). Im Feld **fogType** wird die Art angegeben, wie der Nebel in Abhängigkeit der Distanz zunehmen soll: „LINEAR“ oder „EXPONENTIAL“, was einen natürlicheren Effekt ergibt. Das Feld **visibilityRange** gibt die Sichtgrenze an, alles was weiter entfernt ist verschwindet im Nebel. Durch Setzen des Werts 0.0 wird der Nebel ausgeschaltet.

4.2.7 Group

Dieser Knoten dient der Gruppierung mehrerer Objekte, die dann im weiteren Verlauf als eine Einheit auftreten und so dann auch angesprochen werden können. Die Gruppierung kann entweder über ein **children**-Feld erfolgen oder via Definition einer Box (**bboxCenter** und **bboxSize**), wodurch alle Objekte innerhalb dieser Box angesprochen werden.

4.2.8 IndexedFaceSet

Der IndexFaceSet-Knoten stellt eine 3D-Form dar, die aus Flächen (Polygonen) gebildet wird. Diese Flächen werden anhand einer Koordinatenliste (Feld **coord** enthält einen Unterknoten des Typs Coordinate, in welchem die Koordinaten aller Punkte aufgeführt sind) sowie einer Liste der Dreiecksflächen im Feld **coordIndex** (wo aufgeführt ist, welche drei Punkte eine Fläche bilden). Die einzelnen Polygone werden dabei durch Angabe des Werts -1 voneinander getrennt. Von den zwei Seiten einer Polygonfläche ist jeweils nur eine sichtbar, was durch die Angabe der Normalen bestimmt wird. Schaut man auf eine sichtbare Polygonseite, so sind die Punkte entgegen dem Uhrzeigersinn sortiert (ccw: counter-clockwise). Indem man das Feld **ccw** auf FALSE setzt, erreicht man, dass diejenigen Seiten sichtbar werden, die im Uhrzeigersinn sortiert sind. Weist man dem Feld **solid** ein FALSE-Wert zu, erreicht man, dass beide Seiten sichtbar werden. Anhand eines Beispiels einer Pyramide soll dieser Aufbau veranschaulicht werden.



Der VRML-Quelltext sieht dann wie folgt aus:

```
Transform
    children [
        Shape {
            geometry IndexedFaceSet {
                coord Coordinate { point [
                     1   0   0,      # Punkt 0 auf der pos. x-Achse
                     0   0   1,      # Punkt 1 auf der pos. z-Achse
                    -1   0   0,      # Punkt 2 auf der neg. x-Achse
                     0   0   -1,     # Punkt 3 auf der neg. z-Achse
                     0   1.5 0 ]     # Punkt 4: Pyramidenspitze
                }
                coordIndex [
                     0, 1, 2, 3, -1,   # Grundfläche
                     1, 0, 4, -1,      # 1. Seitenfläche
                     2, 1, 4, -1,      # 2. Seitenfläche
                     3, 2, 4, -1,      # 3. Seitenfläche
                     0, 3, 4, -1       # 4. Seitenfläche
                ]
                ccw TRUE
                creaseAngle 3.14
                solid FALSE
            }
        }
    ]
}
```

Das Modell der Arteplage im Modell Biel wurde genau in dieser Art konstruiert, wobei die Datenmengen natürlich einiges grösser sind als in diesem Beispiel. Neben den bereits erwähnten Feldern existieren noch einige mehr: Das **color**-Feld kann einen Color-Knoten enthalten, der Farbwerte für die Polygonflächen oder

die Eckpunkte enthält. Über den Farbindex (**colorIndex**) können die Farbwerte des verwendeten Color-Knotens den Polygonflächen oder Eckpunkten zugeordnet werden, wobei die Zuordnung vom Wert des Feldes **colorPerVertex** abhängt: beim Wert TRUE werden die Farben den Eckpunkten, beim Wert FALSE den Polygonflächen zugeordnet.
Im Feld **normal** kann ein Normal-Knoten verwendet werden, in welchem die Normalen explizit aufgeführt werden, um dem Browser die Berechnung derselben abzunehmen. Über das Feld **normalIndex** werden die Normalen zugewiesen, wobei je nach Wert des Feldes **normalPerVertex** die Zuweisung an die Eckpunkte oder an die Polygonflächen erfolgt (analog zur Farbzuweisung).
Das Feld **texCoord** kann einen Knoten des Typs TextureCoordinate enthalten. Die Eckpunkte können so auf die im TextureCoordinate-Knoten definierten Koordinaten abgebildet werden. Den Eckpunkten können die Textur-Koordinaten des verwendeten TextureCoordinate-Knotens über einen Index (**texCoordIndex**) zugewiesen werden. Die Ereignisse **set_colorIndex**, **set_coorIndex**, **set_normalIndex** sowie **set_texCoordIndex** ermöglichen es Scripts, diese Indizes abzuändern.

4.2.9 Inline

Dies ist ein Gruppen-Knoten, der Daten von einer bestimmten Adresse im World Wide Web liest und darstellt.

```
Inline {
    url "Biel_Geb.wrl"
}
```

Das Feld **url** enthält eine Liste von WWW-Adressen, wo die Daten zu finden sind. Der Zeitpunkt, wann die ‚Children' gelesen und angezeigt werden ist nicht definiert. Mit Hilfe der Box-Funktion (**bboxCenter** und **bboxSize**)kann der Zeitpunkt gesteuert werden, indem zum Beispiel die Children erst gelesen werden, wenn die umgebende Box für den Betrachter sichtbar ist.

4.2.10 Material

Der Material-Knoten definiert die Oberflächeneigenschaften von geometrischen Knoten, d.h. eines oder mehrerer Objekte.

```
material Material {
    ambientIntensity    0.0499347
    diffuseColor        0.4 0.1 0
    specularColor       0.3 0.3 0.3
    shininess           0.1
    transparency        0.4
    emissiveColor       0 0 0
}
```

Die Felder des Material-Knotens bestimmen, wie das Licht an einem Objekt reflektiert wird um Farbe zu erzeugen:

- **ambientIntensity**: Definiert, wieviel Umgebungslicht der Lichtquellen von der Oberfläche reflektiert werden soll.
- **diffuseColo**r: In Abhängigkeit des Winkels unter dem das Licht der verschiedenen Lichtquellen auf der Oberfläche auftrifft, wird mehr oder weniger diffuses Licht reflektiert.
- **emissiveColo**r: Dient der Modellierung von leuchtenden Objekten, die selber Licht abstrahlen.
- **specularColor** und **shininess**: Diese beiden Felder bestimmen die Reflexionen auf der Körperoberfläche (zum Beispiel leuchtende Flecken auf einer Kugeloberfläche). Wenn der Winkel, unter der das Licht auf der Oberfläche auftrifft, etwa dem Winkel zwischen Oberfläche und Betrachter entspricht, wird der das **specularColor**-Feld bei den Lichtberechnungen hinzugefügt. Das Feld **shininess** beeinflusst die Art der Reflexion: ein kleiner Wert liefert eine weiche und flächenmässig grössere Reflexion, ein grosser Wert eher eine scharfe, kleine Reflexion.
- **transparenc**y: Bestimmt die Transparenz eines Objektes, wobei der Wert 1.0 völlig durchsichtig und 0.0 völlig undurchsichtig bedeutet.

4.2.11 NavigationInfo

Dieser Knoten enthält Informationen über das Navigieren im Modell, so über die Grösse des Körpers, in dem sich der Betrachters durch den Raum bewegt (Avatar), über die Geschwindigkeit und die Art der Bewegung und über Sicht- und Lichtverhältnisse.

```
NavigationInfo {
    avatarSize      [0.25, 1.6, 0.75]
    headlight       TRUE
    speed           1.0
    type            [„FLY“, „WALK“, „EXAMINE“]
    visibilityLimit 0.0
}
```

Das Feld **avatarSize** definiert die physikalische Grösse des Benutzer in der virtuellen Welt, was vor allem bei Kollisionen mit Objekten von Bedeutung ist. Der erste Wert ist der Abstand, auf den man einem Objekt nähern kann. Der zweite Wert bestimmt die Höhe über Boden, die der Höhe der Augen des Benutzer über dem Boden entspricht, und nach der das Programm die Sicht bestimmen soll. Der dritte Wert schlussendlich beschreibt die Höhe eines Objektes. Über das der User sich bewegen kann, also gewissermassen die Höhe einer Treppenstufe, die man noch erklimmen kann. Das Feld **headlight** bestimmt, ob ein Scheinwerfer in der Betrieb ist, der immer in die Sichtrichtung des Betrachters zeigt. Dieser Scheinwerfer ist vom Typ ‚DirectionalLight‘ mit den Eigenschaften intensity = 1, color = (1 1 1), ambientIntensity = 0.0, und direction = (0 0 –1). Die Geschwindigkeit, mit der sich der Benutzer durch die virtuelle Welt bewegt, wird im Feld **speed** festgelegt, wobei die Einheit des Wertes Meter pro Sekunde (m/s) ist.

Die möglichen Arten der Bewegung werden im Feld **type** festgelegt, und welche Art der Bewegung beim Starten aktiv sein soll. Die Art der Bewegung bestimmt die Interaktion zwischen Benutzer und der Welt, in der er sich bewegt. Wird „WALK“ als **type** angegeben, so soll der Browser auch eine „WALK“-Schnittstelle zur Verfügung stellen. Die meisten Browser sollten folgende Navigationsarten unterstützen: „ANY“, „WALK“, „EXAMINE“, „FLY“ und „NONE“.

Wenn „ANY“ nicht in der Liste aufgeführt wird, so werden nur genau diejenigen Navigationsarten angeboten, die in der Liste aufgeführt sind. Steht „ANY“ hingegen in der Liste, so bietet der Browser alle ihm zur Verfügung stehenden Arten an, und der User kann zwischen allen wählen. Die Navigationsart "WALK" wird gebraucht, um eine virtuelle Welt zu Fuss entdecken zu können. Es sollte ein Gelände vorhanden sein sowie eine Anziehungskraft (Gravitation), die den Benutzer auf der Oberfläche des Geländes behält. Das Navigieren mit "FLY" ist ähnlich, nur ist die Gravitation ausgeschaltet; der Betrachter kann sich also fliegend durch die virtuelle Welt bewegen. Die Navigationsart "EXAMINE" wird gebraucht, um individuelle Objekte zu untersuchen, was durch Zoom- und Rotiermöglichkeiten unterstützt wird. Wird der Wert des Feldes **type** auf "NONE" gesetzt, so wird jegliche Art von Navigation ausgeschaltet. Der Betrachter kann sich dann nur mit Hilfe von Viewpoint- und Anchor-Knoten durch die Welt bewegen.

4.2.12 ProximitySensor

Dieser Sensor erzeugt ein Ereignis, sobald der User in ein vordefiniertes Gebiet eintritt, das Gebiet verlässt oder sich innerhalb des Gebietes bewegt. Das Gebiet, eine Box, wird anhand der Felder **center** und **size** definiert. Durch Senden des Wertes TRUE oder FALSE an das Feld **enabled** kann der Sensor ein- und ausgeschaltet werden.

Der ProximitySensor-Knoten erzeugt Ereignisse, die im Feld **isActive** ausgegeben werden. Befindet sich der User innerhalb der Box, wird der Wert TRUE ausgegeben, sonst FALSE. Beim jedem Eintritt ins Gebiet wird ein **enterTime**-Ereignis erzeugt, beim Verlassen ein **exitTime**-Ereignis.

Während der Zeit, wo sich der Benutzer im Gebiet befindet, generiert der Knoten zwei Ereignisse, welche die aktuelle Position (**position_changed**) und Orientierung (**orientation_changed**) ausgeben. Jeder ProximitySensor arbeitet unabhängig von allen anderen ProximitySensoren, auch wenn ein Gebiet von verschiedenen Sensoren abgedeckt ist. Wird ein size-Element auf Null gesetzt, so ist das gleichbedeutend, wie wenn man das enabled-Feld auf FALSE setzen würde, der Sensor kann keine Ereignisse auslösen.

```
DEF User ProximitySensor {
    size 2000 2000 2000
}
```

Zuerst wird der ProximitySensor definiert, in diesem Fall wurde nur die Grösse festgelegt, zudem könnte der Quader auch anders positioniert werden (durch Verwendung des Feldes **center**).

```
ROUTE User.position_changed TO box.set_translation
ROUTE User.orientation_changed TO box.set_rotation
```

Im späteren Verlauf des Programms werden die vom ProximitySensor gelieferten Ereignisse verwendet, um einen Flug durch das Modell zu steuern, indem die aktuelle Betrachter-Position für weitere Berechnungen verwendet wird.

4.2.13 Script

Zur Steuerung des Verhaltens von Objekten in eines Szene können Scripts und Programme verwendet werden; ihr Einsatz erfolgt immer im Rahmen von Script-Knoten. Dabei gibt es grundsätzlich zwei Möglichkeiten der Script-Verwendung:

- Liegt ein Script oder Programm als externe Datei auf einem Server vor, so wird mittels URL im **url**-Feld des Script-Knotens darauf zugegriffen.
- Ein Script oder ein Programm kann direkt im Script-Knoten selbst enthalten sein, und zwar ebenfalls im **url**-Feld; als Protokoll wird dabei zum Beispiel javascript: bei JavaScript oder javabc: bei Java-Programmen angegeben.

```
DEF pushScript Script {
    url "javascript:

    function pushButton() {

        if (pushPop == TRUE)

            {pushPop = FALSE;
            mode = new MFString('NavigationMode');
            pushPop2 = 0; }

        else

            {pushPop = TRUE;
            mode = new MFString('ChooseMode');
            pushPop2 = 1;}
    }"
    eventIn      SFTime      pushButton
    eventOut     SFBool      pushPop
    eventOut     MFString    mode
    eventOut     SFInt32     pushPop2
}
```

Es können beliebig viele Definitionen von Feldern und Ereignissen folgen, die vom Script oder Programm benötigt werden; einerseits dienen sie zur Übergabe von Werten der zu empfangenden (**eventIn**) und zu sendenden (**eventOut**) Ereignisse, andererseits in Form von Feldern zum Speichern von Werten.

4.2.14 Shape

Der Shape-Knoten ist ein Gestaltungsknoten, der dazu verwendet wird, sichtbare Objekte in einer Szene aufzubauen. Er beinhaltet eine geometrische Struktur (**geometry**), die durch ihr durch ihre Oberfläche, durch Linien oder durch Punkte beschrieben wird, und ein Aussehen (**appearance**). Diese zwei Felder können untergeordnete Knoten enthalten, das geometry-Feld einen geometrischen Knoten (zum Beispiel IndexedFaceSet) und das **appearance**-Feld einen Appearance-Knoten, der das Aussehen beschreibt.

```
Shape {
    appearance NULL
    geometry Sphere {radius 1}
}
```

Der Wert NULL im Feld **appearance** besagt, dass keine Beschreibung des Aussehens angegeben wird. Ist das Feld **geometry** NULL, so wird das Objekt nicht gezeichnet.

4.2.15 Sphere

Der Sphere-Knoten definiert eine Kugel, deren Zentrum im Mittelpunkt des lokalen Koordinatensystems liegt, und deren Radius im Feld **radius** angegeben wird.

```
Sphere {
    radius 2
}
```

4.2.16 SpotLight

Der Spotlight-Knoten definiert eine Lichtquelle, die vom einem bestimmten Punkt aus Licht in eine vorgegebene Richtung aussendet, wobei das Licht auf einen bestimmten Öffnungswinkel beschränkt ist. Der Spotlight-Knoten ist in einem lokalen Koordinatensystem definiert, und macht somit Änderungen dieses Koordinatensystems ebenfalls mit.

```
DEF Spot1 SpotLight {
    location 40 80 -170
    direction 0 -1 0
    on FALSE
}
```

Das **location**-Feld bestimmt die Position des Lichtes, das **direction**-Feld die Richtung. Durch Zuweisen des Wertes TRUE oder FALSE an das Feld **on** kann das Licht ein- und ausgeschaltet werden. **Radius** bestimmt die radiale Ausdehnung des Lichtes, Objekte die sich ausserhalb dieses Radius befinden, werden nicht beleuchtet. Das Feld **cutOffAngle** bestimmt den äusseren Winkel, bis zu dem noch Licht ausgestrahlt wird. Das **beamWidth**-Feld bestimmt einen inneren Winkel, bis zu dem das Licht in voller Intensität ausgestrahlt wird. Zwischen den beiden Winkeln nimmt die Intensität ab, und erreicht beim äusseren Winkel den Wert Null.

4.2.17 Switch

Der Switch-Knoten ermöglicht das Wählen zwischen verschiedenen Knoten, die im Feld **choice** aufgeführt sind. Das Feld **whichChoice** bestimmt, welcher der aufgeführten Knoten gewählt wird, wobei der erste Knoten den Index 0 erhält.

```
DEF GelSwitch Switch {
    choice [
        Group {
            children [
                DEF Testtest Transform {
                    children [
                        DEF GELAENDE Inline {url "Koord.wrl"}
                    ]
                }
            ]
        }
        Group {
            children [
                DEF ANCHOR3 Anchor {
                    description "Gelände"
                    url ""
                    parameter ""
                    children [
                        DEF GELAENDE Inline {url "Koord.wrl"}
                        DEF TSensor TouchSensor {
                            enabled FALSE
                        }
                    ]
                }
            ]
        }
    ]
    whichChoice 0
}
```

Steht im whichChoice-Feld ein negativer Wert oder ein Wert, der grösser ist als die Anzahl der Knoten, wird keiner der Knoten ausgewählt. Unabhängig des Wertes in whichChoice können alle aufgeführten Knoten Ereignisse empfangen und auch senden.

4.2.18 Text

Der Text-Knoten definiert ein flaches Textobjekt, das sich in de Z=0-Ebene des lokalen Koordinatensystems befindet. Text-Knoten können mehrere Textstränge enthalten, wobei die Textstränge im Feld **string** enthalten sind.

```
DEF PosText Text {
    string ""
    fontStyle FontStyle {
        size 0.06
    }
}
```

Das **fontStyle**-Feld enthält einen FontStyle-Knoten der beschreibende Eigenschaften wie Schriftgrösse (font size), Schriftart (font family), Schriftstil (font style) und die Richtung der Textstränge enthält. Das **maxExtent**-Feld limitiert und komprimiert alle Textstränge, wenn die Länge des längsten Textstrangs länger ist als der im Feld angegebene Wert, sonst erfolgt keine Kompression. Das **length**-Feld enthält einen Wert (oder mehrere Werte für verschiedene Textstränge), der die exakte Länge eines Textstrangs angibt. Entspricht die tatsächliche Länge nicht diesem Wert, so wird der Textstrang komprimiert oder ausgedehnt.

4.2.19 TextureTransform

Der Texture-Transform-Knoten definiert eine 2D-Transformation, die auf Texturkoordinaten angewendet wird. Das bestimmt die Art, wie die Texturkoordinaten auf die geometrische Oberfläche abgebildet werden, der sie zugeordnet sind.

```
textureTransform TextureTransform {
    scale 1 -1
}
```

Die Transformation besteht aus einer Translation (**translation**-Feld), einer Rotation (**rotation**) um einen festgelegten Mittelpunkt (**center**) sowie einer Massstabsänderung (**scale**), wobei für jede der zwei Achsen ein eigener Massstab angegeben werden kann.

4.2.20 TimeSensor

Der TimeSensor erzeugt Ereignisse in Abhängigkeit der Zeit, die verstrichen ist. Er kann angewendet werden, um Simulationen und Animationen zu steuern, um periodische Aktivitäten zu kontrollieren oder um einzelne Ereignisse auszulösen.

```
DEF Flug2Time TimeSensor {
    loop FALSE
    enabled TRUE
    cycleInterval 60
}
```

Zwei Ereignisse dieses Sensors senden nur diskrete Werte: **isActive** sendet den Wert TRUE, solange der Sensor läuft, und **cycleTime** sendet eine Zeit beim Start (zum Zeitpunkt **startTime**) und dann am Anfang jedes neuen Durchgangs. **Time** hingegen sendet dauernd die absolute Zeit, und **fraction_changed** gibt an, wieviel des aktuellen Zyklus bereits verstrichen ist (in Werten zwischen 0 und 1). Die Länge eines Zyklus wird im Feld **cycleInterval** angegeben (in Sekunden), wobei der Wert grösser als Null sein sollte. Hat das Feld **loop** den Wert TRUE, so wird das Zeitintervall endlos wiederholt, falls keine Stopzeit (**stopTime**) angegeben wird, oder sonst so lange, bis die Stopzeit erreicht ist. Der TimeSensor kann durch Senden des Wertes FALSE ans Feld **enabled** ausgeschaltet werden, das heisst er sendet keine Ereignisse mehr.

4.2.21 TouchSensor

Ein TouchSensor verfolgt die Position und der Zustand der Maus und reagiert, wenn die Maus ein Objekt berührt, das im TouchSensor-Knoten enthalten ist.

```
DEF TextTouch TouchSensor {
    enabled TRUE
}
```

Der Sensor kann durch Senden des Wertes TRUE oder FALSE an das **enabled**-Feld ein- und ausgeschaltet werden; im ausgeschalteten Zustand reagiert der Sensor nicht und sendet auch keine Ereignisse. Ereignisse werden generiert, wenn die Maus auf ein Objekt innerhalb des TouchSensor-Knotens zeigt. Solange sich die Maus über dem Objekt befindet, sendet der Sensor ein **isOver**-Ereignis mit dem Wert TRUE. Während der Betrachter sich mit der Maus über dem Objekt befindet, kann er durch Mausklick ein **isActive**-Ereignis (TRUE) auslösen. Das Loslassen der Maustaste, also der Übergang vom aktiven in den inaktiven Zustand des Sensors, generiert ein Ereignis, das die aktuelle Zeit enthält (**touchTime**) und das zum Starten oder Stoppen einer Animation verwendet werden kann. Während der Zeit, in der **isOver** den Wert TRUE hat, werden laufend Ereignisse erzeugt, die jeweils den aktuellen Schnittpunkt von Maus und Objekt (**hitPoint_changed**), die Normale am Schnittpunkt (**hitNormal_changed**) sowie gegebenenfalls die Textur-Koordinaten am Schnittpunkt enthalten. Ist die Maustaste gedrückt, so wird die Bewegung der Maus als Ziehbewegung interpretiert.

4.2.22 Transform

Der Transform-Knoten ist ein Gruppenknoten, der ein Koordinatensystem für seine Kinder definiert in Relation zum übergeordneten Koordinatensystem.

```
DEF Arte_Biel Transform {
    translation -78 -16 -7
    scale 0.3 0.3 0.3
    rotation 1 0 0 1.57
    children [
        Inline {url "arteplage.wrl"}
    ]
}
```

Die möglichen Transformationen sind eine Translation (**translation**), eine Rotation (**rotation**), ein Massstab (**scale**) samt einer Massstabsorientierung (**scaleOrientation**) sowie eine Mittelpunktsangabe (**center**). Die Reihenfolge der Transformationen ist so, dass zuerst die Massstabsänderung, dann die Rotation und am Schluss die Translation durchgeführt wird.

4.2.23 Viewpoint

Der Viewpoint-Knoten definiert einen spezifischen Ort im lokalen Koordinatensystem, von welcher aus der Betrachter die Szene sehen kann. Werden mehrere solcher Knoten verwendet, so ist immer der als erstes aufgeführte Viewpoint der aktive (beim Laden der Szene), ausser wenn einer der Viewpoints explizit mittels Schlüsselwort einen Namen zugewiesen erhält. Sind mehrere mit einem Namen bezeichnet, so kommt wieder der erste zum Zug. Im Browser existiert eine Liste der vorhandenen Aussichtspunkte, aus welcher der gewünschte ausgewählt werden kann.

```
DEF sued Viewpoint {
    position 0 400 600
    orientation 1 0 0 -0.7
    description "Sicht von Sued"
}
```

Mit den Feldern **position** und **orientation** wird die Lage und die Blickrichtung (relativ zur Standard-Orientierung) im lokalen Koordinatensystem festgelegt. Die ersten drei Werte im Feld **orientation** definieren einen Vektor im Raum, welcher die Drehachse festlegt. Der vierte Wert gibt den Winkel (in Radian) an, um den gedreht wird. Das Feld **jump** bestimmt, ob der Betrachter direkt zum Aussichtspunkt springt (und so auf dem Weg dorthin unterwegs keine Ereignisse auslösen kann - durch Aktivieren eines Sensors zum Beispiel). Durch Ändern des Wertes im Feld **fieldOfView** kann der betrachtete Ausschnitt der Szene vergrössert werden; ein kleines Blickfeld entspricht dabei etwa einer Zoom-Funktion, ein weites Blickfeld einer Weitwinkel-Funktion, vergleichbar mit einer Kamera. Das **description**-Feld enthält eine Beschreibung des Aussichtspunktes, die browserspezifisch dargestellt wird.

4.3 Beispiele zur Anwendung der Knoten (integrierte Features)

In den meisten Fällen konnten die Knoten in der Form integriert werden, wie sie im vorherigen Kapitel beschrieben ist, vor allem was die Beleuchtung sowie die Aussichtspunkte betrifft. Das Erscheinungsbild der Szene war jedoch nicht der einzige Punkt, der in das Modell integriert werden sollte. Zusätzlich sollte auch die Interaktivität zwischen Benutzer und Modell ausgebaut werden, wofür Kombinationen von Knoten und Integration von Script zur Anwendung kamen. Im folgenden sollen einige Features beschrieben werden, welche in unseren Modellen realisiert wurden.

4.3.1 Verknüpfung der einzelnen Modelle

Beim Erstellen der Verknüpfungen der einzelnen Modelle wollten wir erreichen, dass mit dem Anklicken von verschiedenen Stellen im Modell auch verschiedene Untermodelle aufgerufen werden. Dies hat sich nachträglich als problematisch herausgestellt, da von VRML 2.0 keine Funktion bereitgestellt wird, womit Teilgebiete eines Modells angesprochen werden können. Eine Möglichkeit besteht darin, das Modell selber in kleinere Untermodelle aufzuteilen, die dann jeweils einzeln ansprechbar wären. Aufgrund der Dateistruktur unserer Modelle, in welchen die Punkte nicht mehr in einem regelmässigen Raster sondern jeweils mit den Koordinatenwerten aufgeführt sind, kam das für unser jedoch nicht in Frage.

Dasselbe gilt für das Bild, welches dem Modell überlagert wird; auch hier können nicht Teilgebiete des Bildes angesprochen werden. Unsere nächste Überlegung bestand darin, das zu überlagernde Bild in kleinere Bilder aufzuteilen und das Modell mit diesen Bildern zu überlagern. Die Links zu den anderen Modellen würden dann auf die kleineren Bilder verteilt. VRML unterstützt diese Variante jedoch auch nicht; das zu überlagernde Bild kann per Massstab verkleinert werden, doch erscheint es dann mehrmals auf dem Modell. Folgende Funktionen standen uns zur Lösungssuche zur Verfügung:

- TouchSensor-Knoten: Dieser Sensor reagiert, wenn sich der Mauscursor über dem Objekt befindet, und liefert Koordinaten sowie den Zustand der Maus (Klick) zurück. Ein TouchSensor bezieht sich immer auf das ganze Objekt.
- Anchor-Knoten: Mit Hilfe des Anchors können Verknüpfungen zu anderen Objekten auf ein Objekt gesetzt werden. Beim Anklicken des Objekt wird die Verknüpfung zum anderen Objekt ausgeführt. Auch hier gilt, dass ein Anchor sich immer auf ein ganzes Objekt bezieht.
- Script-Knoten: Im Script-Knoten ist ein Programm enthalten, welches die Position der Maus auf dem Objekt untersucht. Befindet sich die Maus in einem der vordefinierten Perimeter, so wird der entsprechende Link an den Anchor-Knoten weitergegeben.

Mit Hilfe einer Kombination der drei Knoten ist es möglich, das verknüpfte Objekt abhängig von der Mausposition (und somit vom gewählten Ort im Modell) zu machen. Der TouchSensor liefert die Mausposition auf dem Objekt, anhand welcher der Script-Knoten entscheidet, in welchem Gebiet sich die Maus befindet. Je nach Gebiet wird ein anderer Link (Zieladresse) an den Anchor-Knoten weitergegeben, der bei einem Mausklick geladen wird. Ein weiteres Problem

taucht jedoch auf, da durch den Anchor die Navigierfähigkeit verloren geht, sobald man sich auf dem Objekt befindet, da der Cursor vom Navigier- zum Anchor-Modus wechselt. Dasselbe Problem besteht auch beim TouchSensor, jedoch kann man den TouchSensor ohne weiteres ausschalten, was beim Anchor jedoch nicht möglich ist. Um dieses Problem zu umgehen muss auf eine weiter Funktion von VRML zurückgegriffen werden:

- Switch: Der Switch-Knoten umfasst mehrere Objekte, aus denen dann jeweils eines ausgelesen werden kann.

Um sowohl die Navigier-Funktion auch im Gelände zu erhalten als auch die Verknüpfungen realisieren zu können, erstellten wir zwei Modi, zwischen denen durch Anklicken umgeschaltet werden kann:

- Der ChooseModus, wo auf die Mausposition reagiert wird und Links zu den Untermodellen vorhanden sind.
- Der NavigationModus, wo keine Interaktion zwischen Benutzer und Modell erfolgt.

Dies war aber nur möglich, indem zwei verschiedene Modelle implementiert wurden; das eine besteht wirklich nur aus dem Modell selber und ist deshalb für die Navigation geeignet, das zweite beinhaltet sowohl TouchSensor wie auch Anchor, und ermöglicht die Verknüpfungen zu den Untermodellen. Beim Wechsel zwischen Navigation- und Choose-Modus wird mit Hilfe des Switch-Knotens vom einen zum anderen Modell gewechselt.

4.3.2 Steuerung der Beleuchtung

Die aktuelle Beleuchtung kann ebenfalls von der Mausposition und somit vom Verhalten des Betrachters abhängig gemacht werden. Dies haben wir dazu verwendet, um dem Betrachter mitzuteilen, sobald er mit der Maus auf eine Stelle des Modells trifft, die für ihn interessant ist. Die Verknüpfungen zu den anderen Modellen sind solche Stellen.

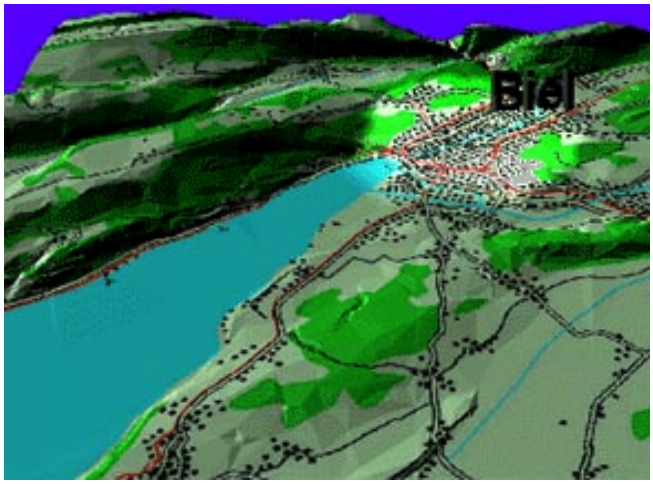


Abbildung 6: SpotLight-Beleuchtung interessanter Stellen

Zu diesem Zweck befinden sich über den interessanten Stellen jeweils ein Spotlight. Diese Spotlights sind fest installiert und allesamt ausgeschaltet, d.h. sie haben den Wert FALSE im Feld ‚on' des Spotlight-Knotens. Es wird die gleiche Kombination von TouchSensor- und Script-Knoten verwendet, wie sie schon bei der Verknüpfung der Modelle zum Einsatz kommt. Der TouchSensor gibt die aktuellen Mauskoordinaten an den Script-Knoten weiter, wo abgeklärt wird, in welchem Bereich des Modells die Maus gerade ist. Befindet sich die Maus in einem der interessanten Bereiche, so gibt der Script-Knoten den Wert TRUE weiter an das Feld ‚on' des entsprechenden Spotlight, wodurch das Licht eingeschaltet wird. Verlässt die Maus das Gebiet wieder, wird der Wert FALSE geschickt, was das Licht wieder ausschaltet.

4.3.3 Hilfsprogramme für die Bestimmung von Position und Orientierung

Für verschiedene Anwendungen werden die exakten Koordinaten innerhalb des Modells sowie die Orientierung benötigt, sei es für die Definition von Aussichtspunkten oder für das Gestalten von Flügen. Dafür standen uns von Heiko Mundle verschiedene Dateien zur Verfügung, welche wir vorübergehend in unsere Modell integrieren konnten, um Koordinaten und Orientierung an bestimmten Stellen zu erhalten. Um diese Dateien und deren Funktionen in unsere Modell zu integrieren, reichte es, sie der Datei des zu untersuchenden Modells anzuhängen. Der Aufbau dieser Dateien soll kurz erläutert werden.

OrientationKoord.wrl

In einem ersten Schritt wird ein ProximitySensor definiert, in dessen Bereich das Anzeigen der Orientierung erfolgt. Anschliessend wird ein Knoten namens ‚UserPos' definiert, der die angezeigte Schrift (MyText) beinhaltet. Der Script-Knoten wird dazu verwendet, die aktuellen Orientierungswerte, die der ProximitySensor liefert, in einen String umzuwandeln der später dem Text-Knoten ‚MyText' übergeben wird. Die beiden Ereignisse orientation_changed und position_changed des ProximitySensors werden an den Knoten UserPos weitergegeben, was bewirkt, dass die Schrift die gleichen Bewegungen wie der Betrachter durchführt und somit aus der Sicht des Betrachters immer am gleichen Ort bleibt.

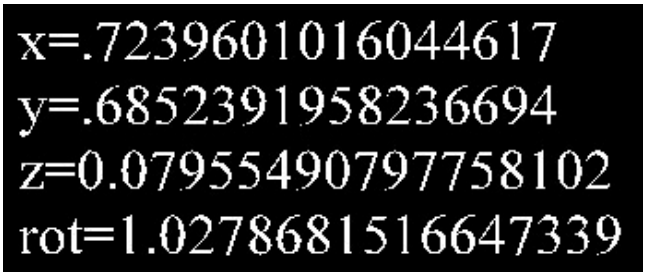


Abbildung 7: Angezeigte Orientierungsparameter

```
DEF User ProximitySensor {
    size 500 500 500
}
Group {
    children [
        DEF UserPos Transform {
            translation 0 0 14
            children [
                Transform {
                    translation  0.1 0.3 -1
                    children [
                        Shape {
                            geometry   DEF MyText Text {
                                string "Orientierung"
                                fontStyle FontStyle {
                                    size 0.06
                                }
                            }
                        }
                    ]
                }
            ]
```

```
        }
        DEF MyScript Script {
            eventIn SFRotation orientOfUser
            eventOut MFString orientString
            url ["javascript:
            function orientOfUser(value) {
                orientString = new MFString('x='+value[0],'y='+value[1],'z='+value[2],
                'rot='+value[3]);
            }",
            "vrmlscript:
            function orientOfUser(value) {
                orientString = new MFString('x='+value[0],'y='+value[1],'z='+value[2],
                'rot='+value[3]);
            }"]
        }
        Viewpoint {
            position 0 0 14
        }
    ]
}
ROUTE User.orientation_changed TO MyScript.orientOfUser
ROUTE User.position_changed TO UserPos.set_translation
ROUTE User.orientation_changed TO UserPos.set_rotation
ROUTE MyScript.orientString TO MyText.string
```

PositionKoord.wrl

Vom Aufbau her entspricht diese Datei der Datei ‚OrientationKoord', der einzige Unterschied besteht darin, dass nicht die Orientierungsparameter an den Script-Knoten weitergegeben werden, sondern die Positionswerte. Diese werden dort in Text umgewandelt, der dem Text-Knoten ‚MyText' zugewiesen und somit am Bildschirm dargestellt wird.

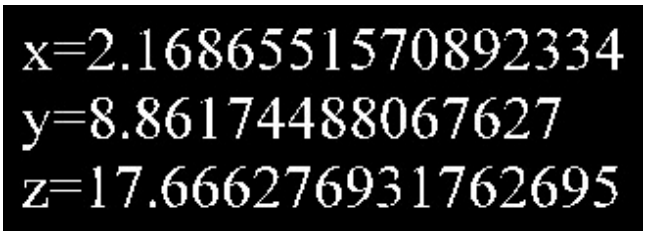


Abbildung 8: Angezeigte Positionsparameter

Die Änderungen betreffen somit nur den Script-Knoten sowie eine Zeile der ROUTE-Zuweisungen.

```
DEF MyScript Script {
    eventIn SFVec3f posOfUser
    eventOut MFString posString
    url ["javascript:
    function posOfUser(value) {
        posString = new MFString('x='+value[0],'y='+value[1],'z='+value[2]);
    }",
    "vrmlscript:
    function posOfUser(value) {
        posString = new MFString('x='+value[0],'y='+value[1],'z='+value[2]);
    }"]
}
```

Anstelle der Werte der geänderten Orientierung werden in dieser Datei die Werte der geänderten Position vom ProximitySensor an den Knoten ‚UserPos' weitergegeben, die Zeile

ROUTE User.orientation_changed TO MyScript.posOfUser
wird ersetzt durch die Zeile
ROUTE User.position_changed TO MyScript.posOfUser

VertexKoord

Die Gruppe bestehend aus dem Transform-Knoten ‚UserPos', dem Script-Knoten ‚MyScript' sowie einem Viewpoint-Knoten (siehe Quelltext der Datei OrientationKoord.wrl) wird in dieser Datei ergänzt mit einer Box, auf der ein TouchSensor definiert ist.

```
Transform {
    children [
        Shape {
            appearance Appearance {
                material Material {
                    diffuseColor  0.8 0.2 0.2
                }
            }
            geometry   Box { }
        },
        DEF TSensor TouchSensor {}
    ]
}
```

Der Script-Knoten entspricht dem der Datei ‚positionKoord.wrl', wobei zu beachten ist, dass dem Script-Knoten andere Werte zugewiesen werden und somit am Bildschirm dargestellt werden.

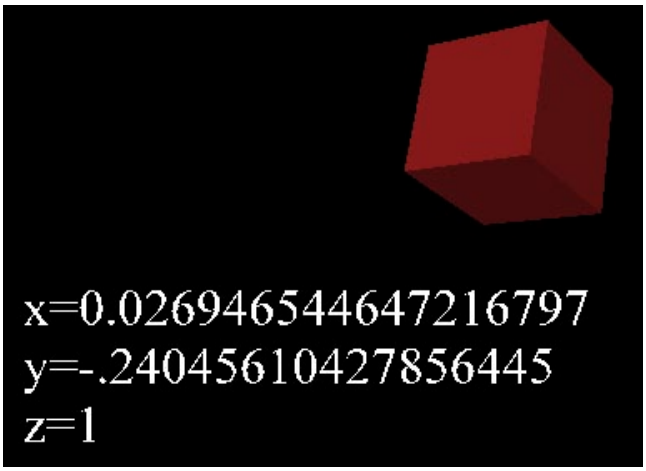


Abbildung 9: Koordinaten der Mausposition auf dem Quader

Es werden nicht die Positionswerte des ProximitySensors (und somit die Position des Betrachters) an den Script-Knoten weitergegeben, sondern die Koordinaten der Maus auf dem Quader, welche das Feld ‚hitPoint_changed' des TouchSensor erzeugt. Dies erfordert eine Änderung der ROUTE-Zuweisungen.
Die Zeile
ROUTE User.position_changed TO MyScript.posOfUser
wird ersetzt durch die Zeile
ROUTE User.orientation_changed TO MyScript.orientOfUser

4.3.4 Für den Betrachter fixe Objekte (Link-Box)

Besonders für das Starten der Flüge war das Bedürfnis vorhanden, Objekte im Bild plazieren zu können, deren Position sich nicht ändert im Bezug zum Betrachter, d.h. sie sollen im Blickfeld des Betrachters immer am gleichen Ort bleiben. Vorbild für diese Objekte waren die Schriften, wie sie zum Beispiel in der Datei VertexKoord (siehe Kapitel 4.3.3) vorkommen. Zur Veranschaulichung wird die Testdatei ‚StatBox.wlr' betrachtet.

Zuerst wird ein ProximitySensor definiert, der immer Ereignisse mit der aktuellen Position und der aktuellen Orientierung des Betrachter erzeugt. Der Knoten ‚UserPos' beinhaltet eine Box sowie den Text ‚Video', der innerhalb dieser Box plaziert ist. Jede Bewegung des Betrachters erzeugt im ProximitySensor zwei Ereignisse mit den geänderten Positions- und Orientierungswerten, welche an den Knoten ‚UserPos' weitergegeben werden und dessen Position und Orientierung laufend aktualisieren, so dass er relativ zum Betrachter immer an der selben Stelle bleibt.

```
DEF User ProximitySensor {
    Size  100 100 100
}
Group {
    children [
        DEF UserPos Transform {
            translation 0 0 14
            children [
                Transform {
                    translation  -0.65 0.3 -1
                    children [
                        DEF BoxxTest Shape {
                            geometry Box {
                                size 0.12 0.05 0.01
                            }
                        }
                        DEF VertexText Transform {
                            translation -0.04 -0.01 0.01
                            children [
                                Shape {
                                    appearance Appearance {
                                        material Material {
                                            diffuseColor 0 0 0
                                        }
                                    }
                                    geometry Text {
                                        string "Video1"
                                        fontStyle FontStyle {
                                            size  0.035
                                            family "SANS"
                                            style "BOLD"
                                        }
                                    }
                                }
                        ]}
                ]}
        ]}
]}
ROUTE User.position_changed TO UserPos.set_translation
ROUTE User.orientation_changed TO UserPos.set_rotation
```

4.3.5 Flüge und Animationen

Ein Flug ist eigenltich nichts anderes als eine Aneinanderreihung von Viewpoints, wie sie im VRML-Aufbau erklährt wurden. Dabei können verschiedene Parameter definiert werden:

- Positionen als Stützstellen während des Flugs
- Zu jeder Position gehört eine Sichtrichtung

Diese beiden Parameter können durch die in Kapitel 4.3.3 beschriebenen Hilfsprogramme bestimmt werden. Das Hilfsprogramm wird in das VRML-Dokuments eingesetzt. Dies ermöglicht das Notieren der Position sowie der Orientierung während der Navigation durch das Modell. Weitere Parameter:

- Dauer des Fluges
- Variation der Fluggeschwindigkeit zwischen den Stützpunkten
- Variation der Schwenkgeschwindigkeit zwischen den Sichtrichtungen

Format der Parameter:

Flugdauer CycleInterval definiert die Flugdauer in Sekunden	`DEF FlugTime TimeSensor {` `  Loop FALSE` `  Enabled TRUE` `  CycleInterval 60` `}`
Position Der PositionInterpolator definiert Fluglinie: • Key vgl. Flug-geschwindigkeit • KeyValue: Hier müssen die Stützpunktkoordinaten eingegeben werden die mit dem Hilfsprogramm bestimmt wurden	`DEF FlugPInterpol PositionInterpolator {` `  key  [`*vgl. Fluggeschwindigkeit*`]` `  keyValue [` `  0 400 600,` `  -178 324 509,` `  -207 137 379,` `  -118 -2.2 1.2` `  -82 -2.4 –78` `  0 400 600` `  ]` `}`
Orientierung Der OrientatinInterpolator definiert die Sichtrichtung: • Key vgl. Schwenk-geschwindigkeit • KeyValue: Hier müssen die Orientierungen eingegeben werden die mit dem Hilfsprogramm bestimmt wurden	`DEF FlugOInterpol OrientationInterpolator {` `  Key  [`*vgl Schwenkgeschwindigkeit*`]` `  KeyValue [` `  1 0 0 -0.7,` `  -0.93 0.32 0.11 0.74,` `  -0.98 -0.19 -0.02 0.53,` `  -0.4 -0.9 -0.07 0.35` `  -0.2 -0.97 -0.07 0.73` `  1 0 0 -0.7` `  ]` `}`
Fluggeschwindigkeit Hier kann über Werte zwischen 0 und 1 die Fluggeschwindigkeit variiert werden. Achtung: Es müssen immer gleich viele Werte vorhanden sein wie es im Feld KeyValue Koordinaten hat.	`  Key  [` `  0,0.2,0.4,0.6,0.8,1` `  ]` *#= konstante Geschwindigkeit* `  Key  [` `  0,0.4,0.7,0.9,0.95,1` `  ]` *#= Anfang schnell Schluss langsam*

Schwenkgeschwindigkeit Hier kann über Werte zwischen 0 und 1 die Schwenkgeschwindigkeit variiert werden. Achtung: Es müssen immer gleich viele Werte vorhanden sein wie es im Feld KeyValue Orientierungen hat.	`Key [` `0,0.2,0.4,0.6,0.8,1` `]` *#= konstante Geschwindigkeit* `Key [` `0,0.4,0.7,0.9,0.95,1` `]` *#= Anfang schnell Schluss langsam*

Zudem muss ein TouchSensor als Flugstart definiert werden. Vorteilhaft ist auch die Definition eines Viewpoints, damit der Flug immer am gleichen Ort startet. Somit lauten die ROUTE-Anweisungen wie folgt:

```
ROUTE FuehrungsTime.isActive TO FuehrerVP.set_bind
ROUTE FuehrerStart.touchTime TO FuehrungsTime.startTime
ROUTE FuehrungsTime.fraction_changed TO FuehrungPInterpol.set_fraction
ROUTE FuehrungsTime.fraction_changed TO FuehrungOInterpol.set_fraction
ROUTE FuehrungPInterpol.value_changed TO Fuehrer.set_translation
ROUTE FuehrungOInterpol.value_changed TO Fuehrer.set_rotation
```

Wichtiges Detail: Damit man, wenn man über das Modell fliegt, nicht „auf dass Modell fällt“, sollte als Navigationsart der Typ FLY im Feld type des Knotens NavigationInfo definiert sein.

```
NavigationInfo {type ["FLY"]}
```

Animation

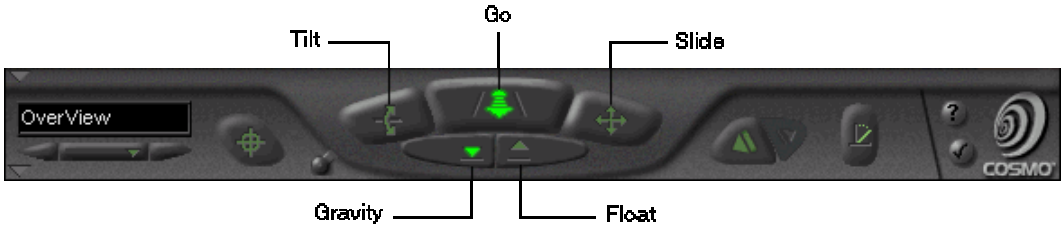
Eine einfache Art von Animation erreicht man durch animierte Bilder die als Textur auf Objekte gelegt werden. Weiter kann sich in einer Szene die Position, die Grösse oder die Oberflächenbeschaffenheit von Objekten ändern; Lichter können aufleuchten, verlöschen oder die Farbe wechseln; Geräusche können ertönen, an- und abschwellen oder enden.
Hinter all diesen Veränderungen stecken Ereignisse, die von den beteiligten Objekten empfangen oder gesendet werden können.

Echte Animation bzw. bewegte Objekte können ähnlich wie der Flug organisiert werden. Vektoren, welche die Koordinaten von Punkten enthalten, können in Abhängigkeit von der Zeit geändert werden. Als Motor, der die Animation antreibt, wird ebenfalls der TimeSensor-Knoten verwendet; er generiert kontinuierliche Folgen von Zeitwerten in Form von Ereignissen. Mit Hilfe dieser Zeitwerte können aus vorgegebenen Vektoren laufend neue Vektoren berechnet werden. Hierzu werden die Zeitwerte als Argumente zur Interpolation zwischen Vektoren an vorgegebenen Stützstellen verwendet. Die Interpolationsknoten sind die gleichen wie beim Flug. Erweiterte Möglichkeiten bieten eingebettete Scirpts oder Programme.

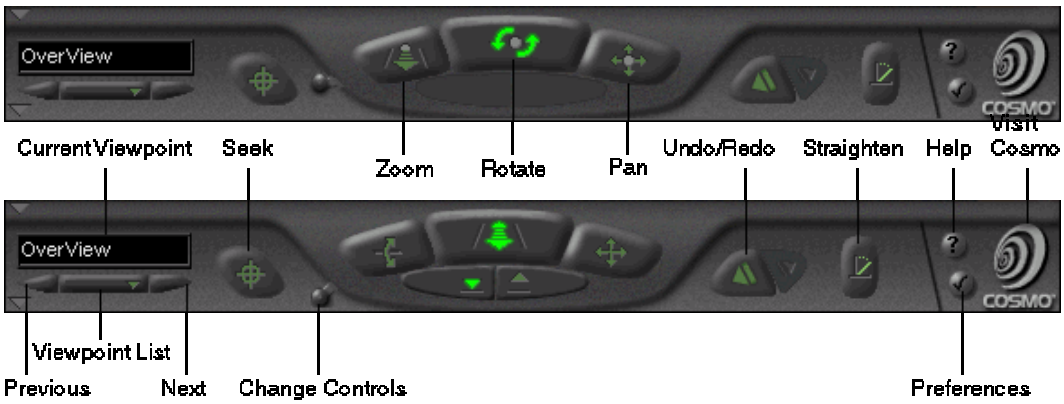
4.4 CosmoPlayer 2.0

Der Plug-In CosmoPlayer von SiliconGraphics ermöglicht das Navigieren in 3D-Welten, die in VRML erstellt sind. Diese 3D-Welten können auch andere Media-Elemente wie Ton, Bild und Filme enthalten. Es existieren zwei Bedienungsoberflächen, zwischen denen hin und her geschaltet werden kann: Einerseits zum Navigieren (Movement Controls) sowie zum Untersuchen eines Objektes (Examine Controls). In diesem Kapitel sollen die Navigierfunktionen kurz beschrieben werden.

Die Navigieroberfläche sieht wie folgt aus:



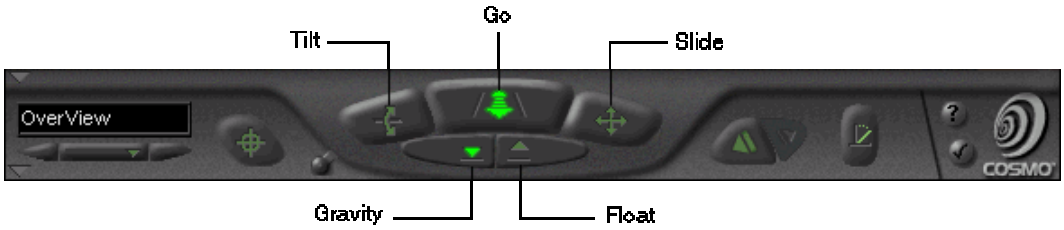
Durch Umschalten erscheint die Untersuchungsoberfläche:

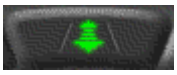
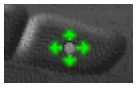
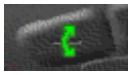
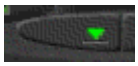
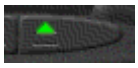


4.4.1 Common Controls

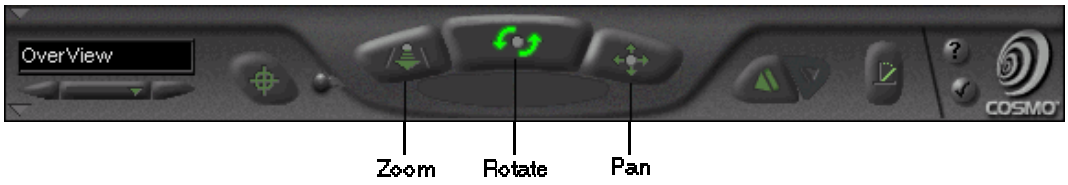
- Seek — Durch Anklicken eines Objektes geht man direkt zum gewählten Objekt.
- Straighten — Anklicken führt dazu, dass die Sicht wieder horizontal ausgerichtet wird bezüglich der Welt, in der man sich bewegt.
- Undo Move — Kehrt zur letzten Position zurück.
- Redo Move — Kehrt zurück zur Position, wo man sich vor dem Drücken von Undo Move befand.
- Current Viewpoint — Kehrt zum aktuellen ViewPoint zurück.
- Viewpoint List — Zeigt eine Liste aller vorhandenen Viewpoints an.
- Previous Viewpoint — Geht zum vorhergehenden Viewpoint in der Liste.
- Next Viewpoint — Geht zum nächsten Viewpoint in der Liste.

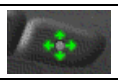
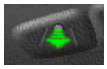
4.4.2 Movement Controls



	**Go**	Klicke **Go** und ziehe den Cursor, um in der Welt herumzugehen. Dabei wird ein Aufwärtsziehen als Vorwärstbewegung, ein Abwärts-ziehen als Rückwärtsbewegung interpretiert. Ein Ziehen nach links oder rechts ergibt eine Drehung in die entsprechende Richtung. Gleichzeitiges Drücken von **Shift** beschleunigt die Bewegung. **Go** dreht den Blick in die Richtung der Bewegung.
	**Slide**	Durch Ziehen des Cursor erfolgt eine Bewegung nach oben/unten oder links/rechts. Dabei bleibt die Blickrichtung gleich.
	**Tilt**	Mit Hilfe von **Tilt** wird die Blickrichtung geändert, die Position ändert sich nicht.
	**Gravity**	Durch Anklicken wird die Schwerkraft wirksam, das heisst man bleibt während des Navigierens auf dem Boden der Welt.
	**Float**	Es wirkt keine Schwerkraft. Dies ermöglicht das Navigieren in der Luft.

4.4.3 Examine Controls



	**Rotate**	Ein Ziehen des Mauscursors führt zu einer Rotationsbewegung des Objektes, das sich vor einem befindet. Das gleichzeitige Drücken von **Shift** beschleunigt die Rotation.
	**Pan**	Das Bild schwenkt nach links, rechts, oben oder unten, je nachdem in welche Richtung die Maus gezogen wurde.
	**Zoom**	Ein Ziehen der Maus nach oben ermöglicht ein Einzoomen, eine Bewegung nach unten ein Rauszoomen, wobei **Shift** ein schnelleres Zoomen ermöglicht

5. Theoretischer Aufbau des 3D Informationssystems Expo.01

Das Grundelement unseres Systems ist eine Website, die sowohl die erstellten dreidimensionalen Modelle als auch Links auf weitere Expo-01-Informationssysteme besitzt. Die traditionelle HTML- und JAVA-Ebenen bieten eine ideale Umgebung für die VRML-Modelle und können diese zum Teil auch unterstützen. Wichtig ist, dass die verschiedenen Elemente klar strukturiert sind, insbesondere innerhalb der 3D-Modelle. Die Bestandteile unserer 3D-Modelle sind zusammengefügte 3D-Daten sowie Rasterdaten. Wünschenswert wäre ein grosses Modell bei dem von der Übersicht über die Städte bis zu den Arteplages alles integriert ist und man sich so frei und kontinuierlich bewegen kann. Dies ist aber wegen des grossen Speicherbedarfs der 3D-Modelle nicht sinnvoll. Man muss sich bewusst sein, dass dieses 3D-Informationssystem für das Internet konzipiert ist und grosse Datenmengen unakzeptierbare Ladezeiten hervorrufen.

Die Schwierigkeit besteht darin eine Aufteilung in verschiedene Modellebenen zu finden die folgenden Ansprüchen genügt:

- Logisches Plazieren von Links und Zoomfunktionen auf dem Modell
- Vertretbare Datenmengen (3D Daten und Rasterdaten)
- Expostädte, Hafenanlage und Arteplage detailliert darstellen

Die Aufteilung erfolgt wie folgt:

- erste Stufe als Expo-Gebietsübersicht
- zweite Stufen als Aufteilung auf die 3 Seen
- dritte Stufen als Aufteilung auf die 4 Stadtgebiete
- vierte Stufen als Aufteilung auf die 5 Arteplage

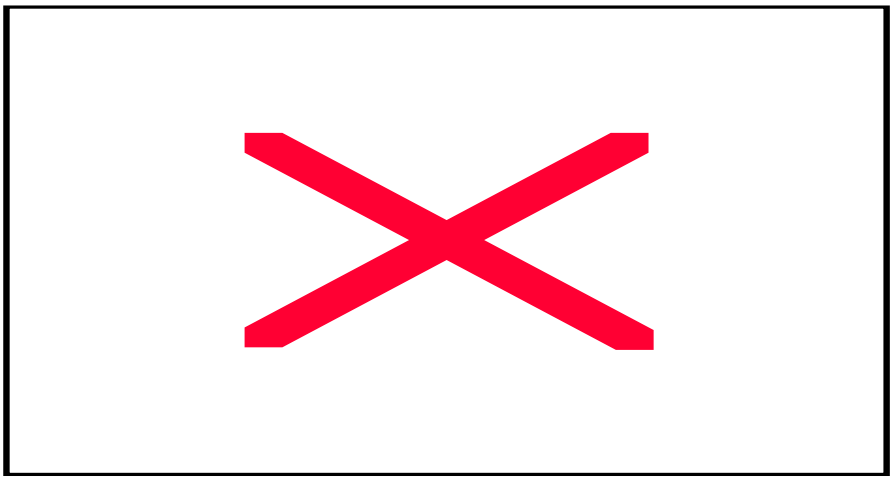


Abbildung 10: Stukturierung der Modellebenen

Von grosser Bedeutung ist die Textur. Das gewählte Rasterbild verleiht dem Höhenmodell durch die Farbgebung topologische Informationen.
Speziell ist die 3D Beschriftung der Städte. Sie bleiben immer auf den Benutzer ausgerichtet und enthalten Links zu je einer Infoseite, die ins Nebenfenster geladen werden. Darin können Informationen und Links zu weiteren Modellen bzw. Daten aus dem Internet integriert werden. Damit kann nun der grosse Vorteil von VRML gegenüber anderen 3D Visualisierungsprogrammen ausgespielt werden, nämlich die Integration von Informationen aus dem World Wide Web.

5.1 Website

Anforderung an die Website sind:

- Ansprechendes Layout
- 3D Modelle direkt aufrufbar
- Implementierung von Links zum Thema Expo-01
- Hilfestellung für unerfahrene Benutzer

Abbildung 11: Website

Im Grundgerüst besteht die Website aus folgenden Frames:

<table>
<tr><td colspan="2">Kopfzeile</td></tr>
<tr><td>Links 1.Ordnung
(name="inhalt")</td><td rowspan="3">Hauptfenster
(name="vrml")</td></tr>
<tr><td>Zwischenbalken</td></tr>
<tr><td>Links
2.Ordnung
(name="info")</td></tr>
</table>

- Kopfzeile: Sie dient nur dem Layout
- Links 1. Ordnung beinhaltet die Themenlinks:
  - Expo Info: Lädt die Expolinkliste in das Frame ‚Links 2. Ordnung'
  - 3D Modelle: Lädt die Modelllinkliste in das Frame ‚Links 2. Ordnung'
  - Hilfe: Lädt die Helplinkliste in das Frame ‚Links 2. Ordnung'
- Zwischenbalken: Dient nur dem Layout
- Links 2. Ordnung: Laden Informations- bzw. Modelldaten ins Hauptfenster
- Hauptfenster: Informations-Frame

Expo Info

In diesem Unterverzeichnis sind Links zu verschiedenen Informationen über die EXPO.01. Wir machten vor allem Gebrauch von den Informationen auf der offiziellen EXPO.01 Homepage (www.expo-01.ch). Dort sind unter anderem auch Studien über die geplanten Arteplage zu finden.

3D-Modelle

Die 3D-Modelle können alle direkt über Links angewählt werden.

Hilfe

Über ein Hilfeverzeichnis wird Unterstützung zu verschiedensten Themen geboten.

5.2 Modell Seen

Abbildung 12: Ausschnitt Modell Seen

Ausdehnung des Modells: (176'000 / 528'000) (227'000 / 594‘000)

Dieses Grundmodell dient der Übersicht und Orientierung. Die Benutzer sollen das Gebiet erkennen und sich mit der Ansicht vertraut machen. Zudem wird man sich in diesem Modell ein Ziel setzen, das man sich näher ansehen will. Um dies zu erreichen sind alle Gebiete, die in einem kleineren, genaueren Modell verfügbar sind mit Links zu diesen Modellen versehen. Diese Links werden dem Benutzer durch ein Spotlicht symbolisiert. Damit man jedoch weiterhin Navigieren kann ist ein Wechsel zwischen NavigationMode und ChooseMode möglich.

Bestandteile des Modells:

- Als Grundlage dient ein Höhenmodell (Abweichung < 200m zu DHM25, 1700KB)
- Textur: Rasterbild (Teile der L+T, 1:200'000, 224KB)
- Zudem sind Die Namen der Expo.01 Städte eingebunden.
- Symbolisch die Jura Insel

In diesem Modell konnten wir der Forderung nach vertretbarer Datenmenge nicht nachkommen, aus diesem Grund sind die Ladezeiten bei langsamen Computern sehr gross. Das Modell müsste wohl mit einem Grunddatensatz von 50 oder 100m Maschenweite Konvertiert werden.

5.3 Modell Bielersee

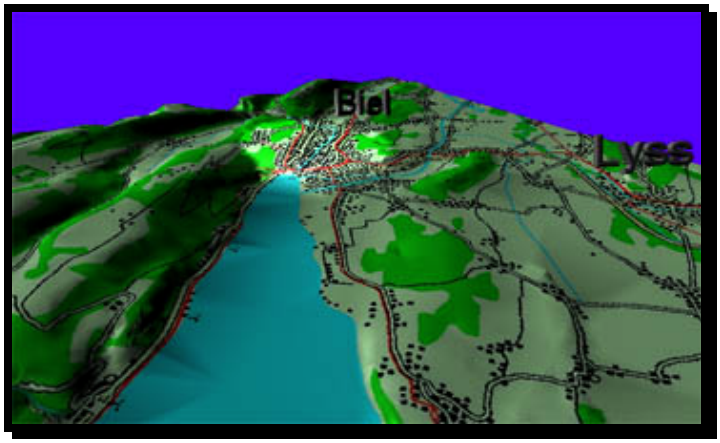


Abbildung 13: Auschnitt Modell Bielersee

Ausdehnung des Modells: (210'000 / 577‘500) (225'000 / 592‘500)

Das Modell Bielersee wurde exemplarisch für alle Modelle der zweiten Stufe programmiert. Dieser Ausschnitt von 15km auf 15km ergibt eine angenehme Dateigrösse von unter 350KB, womit die Navigation angenehmer ist als im Modell Seen. Dies ist ideal um mit fest programmierten Flügen dem Benutzer das Gebiet näher zu bringen. Für das Auslösen der Flüge sind kleine, stationäre Linktasten nötig. Zudem sollte auch hier, gleich wie im Modell Seen, zwischen NavigationMode und Choose-Mode gewählt werden können. Zusätzlich wird dem Benutzer im ChooseMode die Links mit Spots symbolisiert. Gleich wie in allen Modellen sind auch im Bielerseemodell die Städte mit Schriften bezeichnet, derren Link zu einer Informationsseite führen.

Bestandteile des Modells:

- Als Grundlage dient ein Höhenmodell (Abweichung<20m zu DHM25,339KB)
- Textur: Rasterbild (Teile der L+T 1:200'000, 267KB)
- Zudem sind Die Namen der Expo.01 Städte als 3D Schrift eingebunden.

5.4 Modell Biel

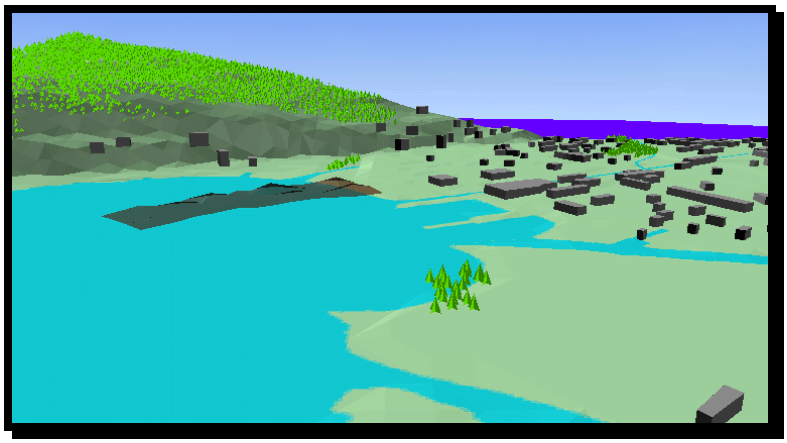


Abbildung 14: Auschnitt Modell Biel

Ausdehnung des Modells: (220'800 / 583'680) (219'425 / 585'575)

Das Modell Biel wurde exemplarisch für alle Modelle der dritten Stufe programmiert. In dieser Stufe ist es wichtig, Details nicht nur mit Rasterdaten auf das Höhenmodell zu legen, sondern diese als dreidimensionale Objekte ins Modell zu integrieren. Für diese Detaildaten kann der Verkor25 Datensatz der L+T verwendet werden. Allerdings muss man dann diesen Objekten noch durch Handarbeit oder mit Hilfe von Programmen eine dritte Dimension geben. Hier besteht wieder die Gefahr von grossen Datenmengen, so dass die verwendeten Daten zum Teil generalisiert werden müssen. Wichtig ist auch eine erste Grundform der Arteplage einzufügen, um den Link auf die vierte Stufe zu ermöglichen. Es gilt also Höhenmodell-, Haus- und Walddaten zusammenzufügen.

Bestandteile des Modells:

- Als Grundlage dient ein Höhenmodell (Abweichung ~ 0m zu DHM25, 157KB)
- Integration von 3D-Stadt (749KB)
- Integration von Wald (628KB)
- Integration von Arteplage-Studie (Eigenkonstruktion)
- Textur: Rasterbild (Teile der L+T 1:200'000, 13KB)

Die Möglichkeiten zur Integration von mehreren 3D Datensätzen ist hier gut gelungen, allerdings gibt es einige Ungenauigkeiten, wie zum Beispiel leicht abgehobene Häuser, die noch mit Handarbeit in einem CAD Programm korrigiert werden müssten. Zudem muss eine Methode gefunden werden die den Wald speicherfreundlicher einbinden kann.

5.5 Arteplage Biel

Ideal wären 3D Daten von den Expo-Architekten zu integrieren. Aber Notfalls können auch selber konstruiert Daten nützlich sein. Ein grosses Arteplage-Gebäude selber zu konstruieren ist sehr aufwendig, denn es soll ja auch ein Innenleben haben. Als Ersatz zur Ebene Arteplage haben wir ein sehr einfaches Arteplage-Dach konstruiert, das wir in das Bieler Stadtmodell integriert haben.

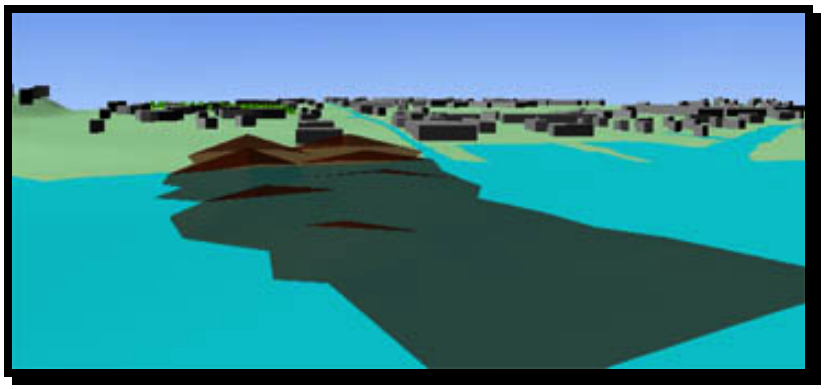


Abbildung 15: Nahaufnahme der Arteplage Biel

6. Literaturverzeichnis

[1] Hase, Hans-Lothar: Dynamische Virtuelle Welten mit VRML 2.0, dpunkt Verlag, 1997

[2] VRML97 X ISO/IEC 14772-1:1997

[3] Mintert, Stefan: JavaScript: Grundlagen und Einführung, addison-wesley Verlag, 1996

[4] Divisek, Werner: HTML: Publizieren im World Wide Web 1996

Anhang I Programmbeschreibungen der einzelnen Modelle

1. Modell Seen

1.1 Dateistruktur

Das Modell Seen besteht aus folgenden Dateien:

- seen.wrl
- region200.wrl
- seen_flach.wrl
- test_region.wrl
- region.jpg
- CC024232.gif
- CC024402.gif
- jura_versuch.wrl
- text_Biel2.wrl
- text_Jura2.wrl
- text_Murten2.wrl
- text_Neuchatel.wrl
- text_Yverdon2.wrl

Die eigentliche Strukturierung des Modells erfolgt in der Datei seen.wrl, von wo aus alle anderen wrl-Files aufgerufen und im Modell positioniert werden. Das eigentliche Gelände (Koordinaten der Punkte sowie Defintion der Polygonflächen) befindet sich in der Datei *region200.wrl*. Dort wird die Bilddatei *region.jpg* aufgerufen wird, welche dem Gelände überlagert wird. Um die Position der Arteplage des Kantons Jura im Modell anzugeben, wird ein im Programm Simply3D erzeugtes Objekt (*jura_versuch.wrl*) eingefügt, welches die beiden Graphik-Dateien *CC024232.gif* sowie *CC024402.gif* verwendet. Die Schriften, welche über den Standorten der verschiedenen Arteplages plaziert werden, sind als eigene wrl-Files vorhanden (*text_Biel2.wrl* bis *text_Yverdon2.wrl*) und werden in das Modell eingefügt.

1.2 Beschreibung der wrl-Files

1.2.1 seen.wrl

Die Datei *seen.wrl* beinhaltet alle Grundeinstellungen, welche für die Darstellung der Szene benötigt werden (Beleuchtung, Viewpoints, Hintergrund, etc.). Die einzelnen Teile des eigentlichen Modells (Geländemodell, 3D-Schriften, andere Objekte) werden aus anderen Dateien geladen. Zusätzlich wird die Verknüpfung zu Informationen im WWW sowie die Verknüpfung zum nächsten Modell sichergestellt. Dies geschieht mit Hilfe von JavaScript-Programmierung.

DEF User ProximitySensor { size 2000 2000 2000 }	Definition des ProximitySensors
Background { groundColor [0.4 0 1, 0.4 0 1] groundAngle [1.5] skyColor [0.3 0.5 0.9, 0.3 0.5 0.9, 0.8 0.9 1, 0.8 0.9 1] skyAngle [1.3, 1.7, 2] }	Definition des Hintergrundes
NavigationInfo { headlight FALSE }	Aussschalten der Standardbeleuchtung
DEF Licht1 DirectionalLight { direction 1 -1 1 on TRUE }	Definition eines gerichteten Lichtes
DEF Spot1 SpotLight { direction 0 -1 0 location 300 60 -230 intensity 0.5 on FALSE } DEF Spot2 SpotLight { direction 0 -1 0 location 170 60 40 intensity 0.5 on FALSE } DEF Spot3 SpotLight { direction 0 -1 0 location -265 60 245 intensity 0.5 on FALSE } DEF Spot4 SpotLight { direction 0 -1 0 location 0 60 -45 intensity 0.5 on FALSE }	Definintion der Spotlights 1, 2, 3 und 4
DEF sued Viewpoint { position 0 400 600 orientation 1 0 0 -0.7 description "Sicht von Sued" } DEF ost Viewpoint { position 600 400 0 orientation -0.32 0.8949 0.32 1.6814 description "Sicht von Ost" } DEF nord Viewpoint { position 0 400 -600 orientation 0 0.9317 0.3631 3.1498 description "Sicht von Norden" } DEF west Viewpoint { position -600 400 0 orientation -0.32 -0.8949 -0.32 1.6814 description "Sicht von Westen" }	Definition der Standardblickpunkte Süd, Ost, Nord und West

```
DEF linkScript Script {                                                    Beginn des Script-Knotens
      url "javascript:                                                     Bezeichnung der verwendenten Sprache

            function linkKoord(value) {                                    Definition einer Funktion

            var bedingung1 = ((value[0] > 0.83) && (value[0] < 0.91) && (value[1] >  Definition von Bedingungen (entspricht dem
0.05) && (value[1] < 0.13)) ? TRUE : FALSE;                                  Abklären, ob sich die Maus in einem
            var bedingung2 = ((value[0] > 0.68) && (value[0] < 0.74) && (value[1] >  bestimmten Perimeter des Geländes
0.41) && (value[1] < 0.47)) ? TRUE : FALSE;                                  befindet)
            var bedingung3 = ((value[0] > 0.14) && (value[0] < 0.19) && (value[1] >
0.65) && (value[1] < 0.74)) ? TRUE : FALSE;
            var bedingung4 = ((value[0] > 0.41) && (value[0] < 0.60) && (value[1] >
0.29) && (value[1] < 0.36)) ? TRUE : FALSE;
            if (bedingung1 == TRUE)

                  { Desc = 'Biel';                                         Auszuführende Anweisungen, falls
                  State1 = TRUE;                                             Bedingung 1 (Maus in Perimeter 1)
                  State2 = FALSE;                                            zutrifft
                  State3 = FALSE;
                  State4 = FALSE;
                  Link = new MFString('../bielersee/bielersee.wrl');
                  para = new MFString('');
                  Ort = new MFString('Bielersee');}

            else

             { if (bedingung2 == TRUE)

                  { Desc = 'Murten';                                       Auszuführende Anweisungen, falls
                  State1 = FALSE;                                            Bedingung 2 (Maus in Perimeter 2)
                  State2 = TRUE;                                             zutrifft
                  State3 = FALSE;
                  State4 = FALSE;
                  Link = new MFString('../../html/sorry.html');
                  para = new MFString('target=info');
                  Ort = new MFString('Murten'); }

               else

                  { if (bedingung3 == TRUE)

                    { Desc = 'Yverdon';                                    Auszuführende Anweisungen, falls
                    State1 = FALSE;                                          Bedingung 3 (Maus in Perimeter 3)
                    State2 = FALSE;                                          zutrifft
                    State3 = TRUE;
                    State4 = FALSE;
                    Link = new MFString('../../html/sorry.html');
                    para = new MFString('target=info');
                    Ort = new MFString('Yverdon'); }

                  else

                    { if (bedingung4 == TRUE)

                        { Desc = 'Neuchatel';                              Auszuführende Anweisungen, falls
                        State1 = FALSE;                                      Bedingung 1 (Maus in Perimeter 1)
                        State2 = FALSE;                                      zutrifft
                        State3 = FALSE;
                        State4 = TRUE;
                        Link = new MFString('../../html/sorry.html');
                        para = new MFString('target=info');
                        Ort = new MFString('Neuchatel'); }

                    else

                        { Desc = 'Übersicht';                              Auszuführende Anweisungen, falls keine
                        State1 = FALSE;                                      der oben aufgeführten  Bedingungen
                        State2 = FALSE;                                      zutrifft
                        State3 = FALSE;
                        State4 = FALSE;
                        Link = new MFString('');
                        para = new MFString('');
                        Ort = new MFString(''); }
                    }
                  }
            }
```

```
    }"

    eventIn      SFVec2f    linkKoord                    eventIn: Ein Ereignis kann empfangen
    eventOut     SFString   Desc                           werden, der aktuelle Wert des Feldes
    eventOut     MFString   Link                           wird durch ein empfangenes Ereignis neu
    eventOut     MFString   Ort                            besetzt.
    eventOut     MFString   para
    eventOut     SFBool     State1                       eventOut: Ein Ereignis kann erzeugt
    eventOut     SFBool     State2                         werden; bei einer Änderung des
    eventOut     SFBool     State3                         aktuellen Feldwertes wird ein Ereignis mit
    eventOut     SFBool     State4                         diesem Wert erzeugt.
}                                                         Ende des Script-Knotens


DEF OBJEKTE Group {                                       Beginn des Gruppenknotens, der alle zu
    children [                                              ladenenden Text-Objekte umgibt.
        DEF Anchor1 Anchor {                              Beginn des Anchor-Knotens (Schrift Biel)
            description "Biel"                            Beschreibung des Links
            parameter "target=info"                       Zielfenster zum Öffnen des Links
            url "../../html/biel.html"                    URL des Links
            children [
                DEF TEXT_BIEL Transform {                 Beginn Transform-Knoten (Schrift Biel)
                    translation 300 35 -230
                    rotation 3 .0 .1 1
                    scale 2 2 2
                    children [
                        Billboard {                       Objekt immer zum Betrachter ausrichten
                            axisOfRotation 0 0 0
                            children [
                                Inline {url "text_Biel2.wrl"}   Verknüpfung mit der Schrift-Datei
                            ]
                        }
                    ]
                }                                         Ende des Transform-Knotens
            ]
        }                                                 Ende des Anchor-Knotens (Schirft Biel)
        DEF Anchor2 Anchor {                              Beginn des Anchor-Knotens (Murten)
            description "Murten"                          Beschreibung des Links
            parameter "target=info"                       Zielfenster zum Öffnen des Links
            url "../../html/murten.html"                  URL des Links
            children [
                DEF TEXT_MURTEN Transform {               Beginn Transform-Knoten (Murten)
                    translation 170 35 40
                    rotation 3 .0 .1 1
                    scale 2 2 2
                    children [
                        Billboard {                       Objekt immer zum Betrachter ausrichten
                            axisOfRotation 0 0 0
                            children [
                                Inline {url "text_Murten2.wrl"}   Verknüpfung mit der Schrift-Datei
                            ]
                        }
                    ]
                }                                         Ende des Transform-Knotens (Murten)
            ]
        }                                                 Ende des Anchor-Knotens (Murten)
        DEF Anchor3 Anchor {                              Beginn des Anchor-Knotens (Yverdon)
            description "Yverdon"                         Beschreibung des Links
            parameter "target=info"                       Zielfenster zum Öffnen des Links
            url "../../html/yverdon.html"                 URL des Links
            children [
                DEF TEXT_YVERDON Transform {              Beginn Transform-Knoten (Yverdon)
                    translation -265 35 245
                    rotation 3 .0 .1 1
                    scale 2 2 2
                    children [
                        Billboard {                       Objekt immer zum Betrachter ausrichten
                            axisOfRotation 0 0 0
                            children [
                                Inline {url "text_Yverdon2.wrl"}   Verknüpfung mit der Schrift-Datei
                            ]
                        }
                    ]
                }                                         Ende des Transform-Knotens (Yverdon)
            ]
        }                                                 Ende des Anchor-Knotens (Yverdon)
        DEF Anchor4 Anchor {                              Beginn des Anchor-Knotens (Neuchâtel)
```

```
                description "Neuchatel"                                    Beschreibung des Links
                parameter "target=info"                                     Zielfenster zum Öffnen des Links
                url "../../html/neuenburg.html"                             URL des Links
                children [
                    DEF TEXT_NEUCHATEL Transform {                          Beginn Transform-Knoten (Neuchâtel)
                        translation 0 35 -45
                        rotation 3 .0 .1 1
                        scale 2 2 2
                        children [
                            Billboard {                                     Objekt immer zum Betrachter ausrichten
                                axisOfRotation 0 0 0
                                children [
                                    Inline {url "text_Neuchatel2.wrl"}      Verknüpfung mit der Schrift-Datei
                                ]
                            }
                        ]
                    }                                                       Ende des Transform-Knotens (Neuchâtel)
                ]
            }                                                               Ende des Anchor-Knotens (Neuchâtel)
            DEF Anchor5 Anchor {                                            Beginn des Anchor-Knotens (Jura)
                description "Jura"                                          Beschreibung des Links
                parameter "target=info"                                     Zielfenster zum Öffnen des Links
                url "../../html/jura.html"                                  URL des Links
                children [
                    DEF JURA Transform {                                    Beginn Transform-Knoten (Modell Jura)
                        translation 0 3 40
                        scale 0.5 0.5 0.5
                        children [ Inline {url "jura_versuch.wrl"} ]
                    }                                                       Ende des Transform-Knotens (Modell Jura)
                    DEF TEXT_JURA Transform {                               Beginn Transform-Knoten (Schrift Jura)
                        translation 0 35 40
                        rotation 3 .0 .1 1
                        scale 2 2 2
                        children [
                            Billboard {                                     Objekt immer zum Betrachter ausrichten
                                axisOfRotation 0 0 0
                                children [
                                    Inline {url "text_Jura2.wrl"}           Verknüpfung mit der Schrift-Datei
                                ]
                            }
                        ]
                    }                                                       Ende des Transform-Knotens (Schrift Jura)
                ]
            }                                                               Ende des Anchor-Knotens (Jura)
        ]
    }                                                                       Ende des Gruppen-Knotens

    DEF GelSwitch Switch {                                                  Beginn Switch-Knoten, der die Wahl
        choice [                                                              zwischen zwei Modellen ermöglicht
            Group {                                                         Beginn Gruppenknoten (Modell 1)
                children [
                    DEF ANCHOR3 Anchor {                                    Beginn Anchor-Knoten
                        description "Gelände"
                        url ""
                        children [
                            DEF GELAENDE Inline {url "Region200.wrl"}       Verknüpfung mit Gelände-Datei
                            DEF TSensor TouchSensor {                       Definition TouchSensor auf Gelände
                                enabled TRUE
                            }
                        ]
                    }                                                       Ende des Anchor-Knotens
                ]
            }                                                               Ende des Gruppenknotens (Modell 1)
            Group {                                                         Beginn Gruppenknoten (Modell 2)
                children [
                    DEF Testtest Transform {                                Beginn Transform-Knoten
                        children [
                            DEF GELAENDE Inline {url "Region200.wrl"}       Verknüpfung mit Gelände-Datei
                        ]
                    }                                                       Ende des Transform-Knotens
                ]
            }                                                               Ende des Gruppenknotens (Modell 2)
        ]
        whichChoice 0                                                       Aktuelle Auswahl
    }                                                                       Ende des Choice-Knotens

    Group {                                                                 Beginn Gruppenknoten (Schrift zur Modus-
```

```
    children [                                                          Wahl, mit TouchSensor)
        DEF UserPos Transform {                                         Transform #1
            translation 0 0 14
            children [
                Transform {                                             Transform #2
                    translation  -0.71 0.35 -1
                    children [
                        Shape {                                         Beginn der Definition des geometr. Objekts
                            geometry DEF VertexText Text {              Beginn Text-Knoten
                                string "ChooseMode"
                                fontStyle FontStyle {
                                    size 0.06
                                }
                            }                                           Ende des Text-Knotens
                        }                                               Ende des Shape-Knotens
                    ]
                }                                                       Ende Transform #2
            ]
        }                                                               Ende Transform #1
        DEF TextTouch TouchSensor {}                                    Definition des TouchSensors auf Schrift
    ]
}

DEF UserPos2 Transform {                                                Transform #1
    translation 0 0 14
    children [
        Transform {                                                     Transform #2
            translation  -0.71 0.29 -1
            children [
                Shape {                                                 Beginn der Definition des geometr. Objekts
                    geometry DEF PosText Text {                         Beginn Text-Knoten
                        string ""
                        fontStyle FontStyle {
                            size 0.06
                        }
                    }                                                   Ende des Text-Knotens
                }                                                       Ende des Shape-Knotens
            ]
        }                                                               Ende Transform #2
    ]
}                                                                       Ende Transform #1

DEF pushScript Script {                                                 Beginn Script-Knoten
    url "javascript:                                                    Bezeichnung der verwendenten Sprache

        function pushButton() {                                         Definition einer Funktion

        if (pushPop == TRUE)                                            Abfrage, ob Bedingung erfüllt ist

            {pushPop = FALSE;                                           Anweisungen für den Fall, dass die
            mode = new MFString('NavigationMode');                        Bedingung erfüllt ist
            pushPop2 = 1; }

        else                                                            Else-Zuweisung

            {pushPop = TRUE;                                            Anweisungen für den Fall, dass die
            mode = new MFString('ChooseMode');                            Bedingung nicht erfüllt ist
            pushPop2 = 0;}
    }"
    eventIn      SFTime      pushButton                                 Definition der Eingabe- und Ausgabefelder
    eventOut     SFBool      pushPop
    eventOut     MFString    mode
    eventOut     SFInt32     pushPop2
}                                                                       Zuweisungen:
ROUTE User.position_changed TO UserPos.set_translation                  Die Position sowie die Orientierung des
ROUTE User.orientation_changed TO UserPos.set_rotation                    Betrachters wird an die Knoten UserPos
ROUTE User.position_changed TO UserPos2.set_translation                   sowie UserPos2 weitergegeben.
ROUTE User.orientation_changed TO UserPos2.set_rotation
ROUTE TSensor.hitTexCoord_changed TO linkScript.linkKoord               Mausposition an den Knoten LinkScript
ROUTE linkScript.Link TO ANCHOR3.url                                    LinkScript gibt den Link sowie zusätzliche
ROUTE linkScript.para TO ANCHOR3.parameter                                Parameter an den ANCHOR3 weiter
ROUTE TextTouch.touchTime TO pushScript.pushButton                      Anklicken der Schrift VertexText wird an
                                                                          den Knoten pushScript weitergegeben
ROUTE pushScript.pushPop TO TSensor.enabled                             Der Wert PushPop wird an den
                                                                          TouchSensor Tsensor weitergegeben
ROUTE pushScript.mode TO VertexText.string                              Der Text im Feld ‚mode' ersetzt den Text im
                                                                          Feld ‚string' von VertexText.
```

ROUTE linkScript.Ort TO PosText.string	Der Text im Feld ‚Ort' ersetzt den Text im Feld ‚string' von PosText
ROUTE pushScript.pushPop2 TO GelSwitch.whichChoice	Der Wert im Feld ‚pushPop2' bestimmt, welches Modell ausgewählt wird
ROUTE linkScript.State1 TO Spot1.on ROUTE linkScript.State2 TO Spot2.on ROUTE linkScript.State3 TO Spot3.on ROUTE linkScript.State4 TO Spot4.on	Die Mausposition bestimmt den Zustand der Felder ‚State1' bis ‚State4' und somit auch den Zustand der vier Spotlichter

1.2.2 region200.wrl

Die Datei *region200.wrl* beinhaltet das eigentliche Geländemodell sowie alle Eigenschaften desselben, wie die geometrische Form und das äussere Erscheinungsbild. Ebenfalls in dieser Datei beschrieben wird die Überlagerung des Geländes mit einem Bild (*region.jpg*).

```
NavigationInfo { headlight FALSE }                          Ausschalten der Standardbeleuchtung
Transform {                                                 Aufruf des Transform-Knotens
    children [
        DEF TIN_200_MediumSpringGreen Transform {           Aufruf eines zweiten Transform-Knotens
            children [
                Shape {                                     Beginn der Definition des geometr. Objekts
                    appearance Appearance {                 Aufruf des Appearance-Knotens
                        material Material {                 Aufruf des Material-Knotens
                            ambientIntensity 0.0499347      Zuweisen der Material-Eigenschaften
                            diffuseColor     0.498039 1 0
                            specularColor    0.8 0.8 0.8
                            shininess        0.1
                        }                                   Ende der Material-Knotens
                        texture ImageTexture {              Aufruf des ImageTexture-Knotens
                        url ["region.jpg" "biel.jpg"]       Verweis auf Bild-Datei
                        }                                   Ende des Image-Texture-Knotens
                        textureTransform TextureTransform { Aufruf des TextureTransform-Knotens
                        scale 1 -1.28
                        }                                   Ende des TextureTransform-Knotens
                    }                                       Ende des Appearance-Knotens
                    geometry IndexedFaceSet {               Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                         Beginn der Koordinatenliste
                                -395.57  -2.09173    293.98,
                                -387.479  -2.49928   305.668,
                                -395.57  -2.27153   305.668,
                                -383.883    -2.763    293.98,
                                -383.883   -2.8469   305.668,
                                ...
                                ...                         Liste mit 18176 Geländepunkten
                                ...
                                387.479  -1.57629  -304.469,
                                388.378  -1.31257   -303.87,
                                393.472  -1.43244  -302.671,
                                389.277  -1.68417  -305.668,
                                395.57  -1.54033  -305.668
                            ]                               Ende der Koordinatenliste
                        }
                        coordIndex [                        Zuweisung der Punkte an Polygonseiten
                            0, 1, 2, -1,
                            1, 0, 3, -1,
                            1, 3, 4, -1,
                            5, 6, 4, -1,
                            7, 8, 9, -1,
                            ...
                            ...                             Liste aller Polygonseiten (Geländeflächen)
                            ...
                            18401, 18403, 18181, -1,
                            18401, 18171, 18400, -1,
                            18401, 18400, 18403, -1,
                            18181, 18402, 18176, -1,
                            18402, 18177, 18176, -1
                        ]
                        creaseAngle 1.57                    Glätten der Oberfläche
                        normalPerVertex FALSE               Zuordnung der Normalen an Eckpunkte
                        ccw FALSE                           Polygonecken im Uhrzeigersinn definiert
                    }                                       Ende des IndexedFaceSet-Knotens
                }                                           Ende des Shape-Knotens
            ]
        }                                                   Ende des 2. Transform-Knotens
    ]
}                                                           Ende des 1. Transform-Knotens
```

1.2.3 seen_flach.wrl

Die einzige Änderung zum Modell ‚Seen' besteht darin, dass das 3D-Gelände durch eine flache Platte ersetzt wird. Der Aufbau der Datei bleibt somit gleich, mit Ausnahme des Switch-Knotens, der ein anderes Geländemodell enthält. Aus diesem Grund wird nur der Aufbau des Switch-Knoten kurz erklärt, die restlichen Informationen sind der Beschreibung der Datei seen.wrl zu entnehmen.

```
DEF GelSwitch Switch {                                  Beginn Switch-Knoten, der die Wahl
    choice [                                              zwischen zwei Modellen ermöglicht
        Group {                                         Beginn Gruppenknoten (Modell 1)
            children [
                DEF ANCHOR3 Anchor {                    Beginn Anchor-Knoten
                    description "Gelände"
                    url ""
                    children [
                        DEF GELAENDE Inline {url "test_region.wrl"}   Verknüpfung mit Gelände-Datei
                        DEF TSensor TouchSensor {       Definition TouchSensor auf Gelände
                            enabled TRUE
                        }
                    ]
                }                                       Ende des Anchor-Knotens
            ]
        }                                               Ende des Gruppenknotens (Modell 1)
        Group {                                         Beginn Gruppenknoten (Modell 2)
            children [
                DEF Testtest Transform {                Beginn Transform-Knoten
                    children [
                        DEF GELAENDE Inline {url "test_region.wrl"}   Verknüpfung mit Gelände-Datei
                    ]
                }                                       Ende des Transform-Knotens
            ]
        }                                               Ende des Gruppenknotens (Modell 2)
    ]
    whichChoice 0                                       Aktuelle Auswahl
}                                                       Ende des Choice-Knotens
```

1.2.4 test_region.wrl

Anstelle des wirklichen Geländemodelles wird bei der Datei *seen_flach.wrl* eine Ebene eingeführt, welche in dieser Datei beschrieben ist. Die Darstellung der Geometrie des Objektes ist im Vergleich zum Geländemodell stark vereinfacht; das äussere Erscheinungsbild und die Überlagerung mit dem Bild *region.jpg* bleiben jedoch gleich.

```
NavigationInfo { headlight FALSE }                          Ausschalten der Standardbeleuchtunge
Transform {                                                 Transform #1
    children [
        DEF TIN_200_MediumSpringGreen Transform {           Transform #2
            children [
                Shape {                                     Beginn der Definition des geometr. Objekts
                    appearance Appearance {                 Aufruf des Appearance-Knotens
                        material Material {                 Aufruf des Material-Knotens
                            ambientIntensity 1              Zuweisen der Material-Eigenschaften
                            diffuseColor    0.5 1 0
                            specularColor   0.2 0.2 0.2
                            shininess       0.1
                        }                                   Ende des Material-Knotens
                        texture ImageTexture {              Aufruf des ImageTexture-Knotens
                        url ["region.jpg" "biel.jpg"]       Verweis auf Bild-Datei
                        }                                   Ende des Image-Texture-Knotens
                        textureTransform TextureTransform { Aufruf des TextureTransform-Knotens
                        scale 1 1
                        }                                   Ende des TextureTransform-Knotens
                    }                                       Ende des Appearance-Knotens
                    geometry   Box {                        Geometrischer Knoten Box
                    size 800 1 600
                    }                                       Ende des Box-Knotens
                }                                           Ende des Shape-Knotens
            ]
        }                                                   Ende Transform #2
    ]
}                                                           Ende Transform #1
```

1.2.5 text_Biel2.wrl

Die Datei *text_Biel2.wrl* sowie alle anderen 3D-Text-Dateien sind im Programm Simply3D entstanden und somit vom Computer erstellt worden, welcher keinen Wert auf Übersichtlichkeit der Daten legt. Der unten aufgeführte Ausschnitt der Datei ist zuerst von Hand geordnet worden, um wenigstens einen kleinen Einblick in den Aufbau dieser Datei zu ermöglichen. Abgesehen vom Ausschalten der Standardbeleuchtung hatten wir an diesen Dateien keine Änderungen vorzunehmen, so dass auf eine ausführlichere Schilderung verzichtet wird.

DEF Root Group { children [DEF Renderize WorldInfo { info ["Renderize Project File: text_Biel2.wrl Thu Nov 12 11:33:22 1998 "] title "text_Biel2.wrl" } NavigationInfo { headlight FALSE }	Aussschalten der Standardbeleuchtung
DEF view_view Viewpoint { position 0 20 100 orientation -1 0 0 0.174533 fieldOfView 0.392699 description "view" } DEF _object_root Transform { #start of objects children [Transform{ # Standardvorderseite_layer tranBack translation 0 0 0 children [Transform { # Standardvorderseite_layer_layer translation -100 100 100 children[DEF Standardvorderseite_layer_rot_y Transform{ children[DEF Standardvorderseite_layer_rot_x Transform{ children[DEF Standardvorderseite_layer_rot_z Transform{ scale 1 1 1 children[Transform{ # Standardvorderseite_layer tranCent translation 0 0 0 children [ ] } # from Center 'tranCent'] } # rot_z] } # rot_x] } # rot_y] } #end tranform for Standardvorderseite_layer_layer] } # to Center 'tranBack' etc.....	

1.2.6 jura_versuch.wrl

Das Objekt, welches die ungefähre Position der Arteplage des Kantons Jura im Neuenburgersee anzeigen soll, wurde ebenfalls im Programm Simply3D erstellt, und es wird hier nicht genauer auf den Dateiinhalt eingegangen.

2. Modell Bielersee

2.1 Dateistruktur

Das Modell Bielersee besteht aus folgenden Dateien:

- bielersee.wrl
- Koord.wrl
- text_Biel2.wrl
- text_Lyss.wrl
- biel-flaechen.gif
- biel-flaechen.jpg

Die eigentliche Strukturierung des Modells erfolgt in der Datei *bielersee.wrl*. Dort werden alle anderen wrl-Files aufgerufen und im Modell positioniert. Das eigentliche Gelände (Koordinaten der Punkte sowie Defintion der Polygonflächen) befindet sich in der Datei *Koord.wrl*, von wo aus die Bilddatei (*biel-flaechen.jpg* oder *biel-flaechen.gif*) aufgerufen wird. Diese wird dem Gelände überlagert. Die Schriften, welche über den verschiedenen Ortschaften plaziert werden, sind als eigene wrl-Files vorhanden (*text_Biel2.wrl* und *text_Lyss.wrl*) und werden in das Modell eingefügt.

2.2 Beschreibung der wrl-Files

2.2.1 bielersee.wrl

Die Datei *bielersee.wrl* beinhaltet alle Grundeinstellungen, welche für die Darstellung der Szene benötigt werden (Beleuchtung, Viewpoints, Hintergrund, etc.). Die einzelnen Teile des eigentlichen Modells (Geländemodell, 3D-Schriften, andere Objekte) werden aus anderen Dateien geladen. Zusätzlich wird auch die Verknüpfung zu Informationen im WWW sowie die Verknüpfung zum nächsten Modell sichergestellt. Dies geschieht mit Hilfe von JavaScript-Programmierung.

DEF User ProximitySensor { size 2000 2000 2000 }	Definition des ProximitySensors
Background { groundColor [0.4 0 1, 0.4 0 1] groundAngle [1.5] skyColor [0.3 0.5 0.9, 0.3 0.5 0.9, 0.8 0.9 1, 0.8 0.9 1] skyAngle [1.3, 1.7, 2] }	Definition des Hintergrundes
NavigationInfo { headlight FALSE type ["FLY","WALK","EXAMINE"] }	Beginn NavigationInfo-Knoten Aussschalten der Standardbeleuchtung Definition der Navigationsarten Ende des NavigationInfo-Knotens
DEF Licht1 DirectionalLight { direction 1 -1 1 on TRUE }	Definition eines gerichteten Lichtes
DEF Spot1 SpotLight { direction 0 -1 0 location 40 80 -170 on FALSE } DEF Spot2 SpotLight { direction 0 -1 0 location 220 80 170 on FALSE }	Definintion der Spotlights 1 und 2
DEF sued Viewpoint { position 0 400 600 orientation 1 0 0 -0.7 description "Sicht von Sued" } DEF ost Viewpoint { position 600 400 0 orientation -0.32 0.8949 0.32 1.6814 description "Sicht von Ost" } DEF nord Viewpoint { position 0 400 -600 orientation 0 0.9317 0.3631 3.1498 description "Sicht von Norden" } DEF west Viewpoint { position -600 400 0 orientation -0.32 -0.8949 -0.32 1.6814 description "Sicht von Westen" }	Definition der Standardblickpunkte Süd, Ost, Nord und West, sowie eines Blickpunktes auf der St. Petersinsel
DEF insel Viewpoint { position -344 -20 182 orientation 0 -1 0 0.7 description "Insel" }	
DEF linkScript Script { url "javascript: function linkKoord(value) {	Beginn des Script-Knotens Bezeichnung der verwendenten Sprache Definition einer Funktion

var bedingung1 = ((value[0] > 0.4) && (value[0] < 0.7) && (value[1] > 0.2) && (value[1] < 0.4)) ? TRUE : FALSE;	Definition von Bedingungen (entspricht dem Abklären, ob sich die Maus in einem bestimmten Perimeter des Geländes befindet)
var bedingung2 = ((value[0] > 0.75) && (value[0] < 0.9) && (value[1] > 0.6) && (value[1] < 0.8)) ? TRUE : FALSE;	
if (bedingung1 == TRUE)	
{ Desc = 'Biel';	Auszuführende Anweisungen, falls Bedingung 1 (Maus in Perimeter 1) zutrifft
State1 = TRUE;	
State2 = FALSE;	
SpotState = TRUE;	
Link = new MFString('../biel/biel.wrl');	
para = new MFString('');	
Ort = new MFString('Biel');}	
else	
{ if (bedingung2 == TRUE)	
{ Desc = 'Lyss';	Auszuführende Anweisungen, falls Bedingung 2 (Maus in Perimeter 2) zutrifft
State1 = FALSE;	
State2 = TRUE;	
SpotState = TRUE;	
Link = new MFString('../../html/sorry.html');	
para = new MFString('target=info');	
Ort = new MFString('Lyss'); }	
else	
{ Desc = 'Übersicht';	Auszuführende Anweisungen, falls keine der oben aufgeführten Bedingungen zutrifft
State1 = FALSE;	
State2 = FALSE;	
SpotState = FALSE;	
Link = new MFString('');	
para = new MFString('');	
Ort = new MFString(''); }	
}	
}"	
eventIn SFVec2f linkKoord	eventIn: Ein Ereignis kann empfangen werden, der aktuelle Wert des Feldes wird durch ein empfangenes Ereignis neu besetzt.
eventOut SFString Desc	
eventOut MFString Link	
eventOut MFString Ort	
eventOut MFString para	eventOut: Ein Ereignis kann erzeugt werden; bei einer Änderung des aktuellen Feldwertes wird ein Ereignis mit diesem Wert erzeugt.
eventOut SFBool State1	
eventOut SFBool State2	
eventOut SFBool SpotState	
}	Ende des Script-Knotens
DEF OBJEKTE Group {	Beginn des Gruppenknotens, der alle zu ladenenden Text-Objekte umgibt.
children [	
DEF Anchor1 Anchor {	Beginn des Anchor-Knotens (Schrift Biel)
description "Biel"	Beschreibung des Links
parameter "target=info"	Zielfenster zum Öffnen des Links
url "../../html/biel.html"	URL des Links
children [	
DEF TEXT_BIEL Transform {	Beginn Transform-Knoten (Schrift Biel)
translation 90 35 -130	
rotation 3 .0 .1 1	
scale 2 2 2	
children [	
Billboard {	Objekt immer zum Betrachter ausrichten
axisOfRotation 0 0 0	
children [	
Inline {url "text_Biel2.wrl"}	Verknüpfung mit der Schrift-Datei
]	
}	
]	
}	Ende des Transform-Knotens
]	
}	Ende des Anchor-Knotens (Schirft Biel)
DEF Anchor2 Anchor {	Beginn des Anchor-Knotens (Lyss)
description "Lyss"	Beschreibung des Links
parameter "target=info"	Zielfenster zum Öffnen des Links
url "../../html/sorry.html"	URL des Links
children [	
DEF TEXT_LYSS Transform {	Beginn Transform-Knoten (Lyss)

```
                translation 200 35 160
                rotation 3 .0 .1 1
                scale 2 2 2
                children [
                    Billboard {                                   Objekt immer zum Betrachter ausrichten
                        axisOfRotation 0 0 0
                        children [
                            Inline {url "text_Lyss.wrl"}          Verknüpfung mit der Schrift-Datei
                        ]
                    }
                ]
            }                                                     Ende des Transform-Knotens (Lyss)
        ]
    }                                                             Ende des Anchor-Knotens (Lyss)
]
}                                                                 Ende des Gruppen-Knotens

DEF GelSwitch Switch {                                            Beginn Switch-Knoten, der die Wahl
    choice [                                                        zwischen zwei Modellen ermöglicht
        Group {                                                   Beginn Gruppenknoten (Modell 1)
            children [
                DEF Testtest Transform {                          Beginn Transform-Knoten
                    children [
                        DEF GELAENDE Inline {url "Koord.wrl"}     Verknüpfung mit Gelände-Datei
                    ]
                }                                                 Ende des Transform-Knotens
            ]
        }                                                         Ende des Gruppenknotens (Modell 1)
        Group {                                                   Beginn Gruppenknoten (Modell 2)
            children [
                DEF ANCHOR3 Anchor {                              Beginn Anchor-Knoten
                    description "Gelände"
                    url ""
                    parameter ""
                    children [
                        DEF GELAENDE Inline {url "Koord.wrl"}     Verknüpfung mit Gelände-Datei
                        DEF TSensor TouchSensor {                 Definition TouchSensor auf Gelände
                            enabled FALSE
                        }
                    ]
                }                                                 Ende des Anchor-Knotens
            ]
        }                                                         Ende des Gruppenknotens (Modell 2)
    ]
    whichChoice 0                                                 Aktuelle Auswahl
}                                                                 Ende des Choice-Knotens

Group {                                                           Beginn Gruppenknoten (Schrift zur Modus-
    children [                                                      Wahl, mit TouchSensor)
        DEF UserPos Transform {                                   Transform #1
            translation 0 0 14
            children [
                Transform {                                       Transform #2
                    translation  -0.71 0.35 -1
                    children [
                        Shape {                                   Beginn der Definition des geometr. Objekts
                            geometry DEF VertexText Text {        Beginn Text-Knoten
                                string "NavigationMode"
                                fontStyle FontStyle {
                                    size 0.06
                                }
                            }                                     Ende des Text-Knotens
                        }                                         Ende des Shape-Knotens
                    ]
                }                                                 Ende Transform #2
            ]
        }                                                         Ende Transform #1
        DEF TextTouch TouchSensor {}                              Definition des TouchSensors auf Schrift
    ]
}

DEF UserPos2 Transform {                                          Transform #1
    translation 0 0 14
    children [
        Transform {                                               Transform #2
            translation  -0.40 0.35 -1
            children [
```

```
                    Shape {                                                 Beginn der Definition des geometr. Objekts
                        geometry DEF PosText Text {                         Beginn Text-Knoten
                            string ""
                            fontStyle FontStyle {
                                size 0.06
                            }
                        }                                                   Ende des Text-Knotens
                    }                                                       Ende des Shape-Knotens
                ]
            }                                                               Ende Transform #2
        ]
}                                                                           Ende Transform #1

DEF pushScript Script {                                                     Beginn Script-Knoten
    url "javascript:                                                        Bezeichnung der verwendenten Sprache

    function pushButton() {                                                 Definition einer Funktion

        if (pushPop == TRUE)                                                Abfrage, ob Bedingung erfüllt ist

            {pushPop = FALSE;                                               Anweisungen für den Fall, dass die
            mode = new MFString('NavigationMode');                            Bedingung erfüllt ist
            pushPop2 = 0; }

        else                                                                Else-Zuweisung

            {pushPop = TRUE;                                                Anweisungen für den Fall, dass die
            mode = new MFString('ChooseMode');                                Bedingung nicht erfüllt ist
            pushPop2 = 1;}
    }"
    eventIn    SFTime   pushButton
    eventOut   SFBool   pushPop                                             Definition der Eingabe- und Ausgabefelder
    eventOut   MFString mode
    eventOut   SFInt32  pushPop2
}

                                                                            Zuweisungen:
ROUTE User.position_changed TO UserPos.set_translation                      Die Position sowie die Orientierung des
ROUTE User.orientation_changed TO UserPos.set_rotation                        Betrachters wird an die Knoten UserPos
ROUTE User.position_changed TO UserPos2.set_translation                       sowie UserPos2 weitergegeben.
ROUTE User.orientation_changed TO UserPos2.set_rotation
ROUTE TSensor.hitTexCoord_changed TO linkScript.linkKoord                   Mausposition an den Knoten LinkScript
ROUTE linkScript.Link TO ANCHOR3.url                                        LinkScript gibt den Link sowie zusätzliche
ROUTE linkScript.para TO ANCHOR3.parameter                                    Parameter an den ANCHOR3 weiter
ROUTE TextTouch.touchTime TO pushScript.pushButton                          Anklicken der Schrift VertexText wird an
                                                                              den Knoten pushScript weitergegeben
ROUTE pushScript.pushPop TO TSensor.enabled                                 Der Wert PushPop wird an den
                                                                              TouchSensor Tsensor weitergegeben
ROUTE pushScript.mode TO VertexText.string                                  Der Text im Feld ‚mode' ersetzt den Text im
                                                                              Feld ‚string' von VertexText.
ROUTE linkScript.Ort TO PosText.string                                      Der Text im Feld ‚Ort' ersetzt den Text im
                                                                              Feld ‚string' von PosText
ROUTE pushScript.pushPop2 TO GelSwitch.whichChoice                          Der Wert im Feld ‚pushPop2' bestimmt,
                                                                              welches Modell ausgewählt wird
ROUTE linkScript.State1 TO Spot1.on                                         Die Mausposition bestimmt den Zustand
ROUTE linkScript.State2 TO Spot2.on                                           der Felder ‚State1' und ‚State2' und somit
                                                                              auch den Zustand der beiden Spotlichter


######################## Flüge ##########################                   Beginn der Beschreibung der Flüge
#
#boxlink Flug1
#

Group {                                                                     Beginn Gruppen-Knoten
    children [
        DEF box Transform {                                                 Transform #1
            translation 0 0 14
            children [
                Transform {                                                 Transform #2
                    translation  -0.65 0.3 -1
                    children [
                        DEF BoxxTest Shape {                                Beginn der Definition des geometr. Objekts
                            geometry Box {                                  Beginn Box-Knoten
                                size 0.12 0.05 0.01
                            }                                               Ende des Box-Knotens
                        }                                                   Ende des Shape-Knotens
                        DEF VertexText Transform {                          Transform #3
                            translation -0.04 -0.01 0.01
```

```
                                    children [
                                        Shape {                                 Beginn der Definition des geometr. Objekts
                                            appearance Appearance {             Aufruf des Appearance-Knotens
                                                material Material {             Aufruf des Material-Knotens
                                                    diffuseColor 0 0 0          Zuweisen der Material-Eigenschaften
                                                }                               Ende des Material-Knotens
                                            }                                   Ende des Appearance-Knotens
                                            geometry Text {                     Beginn Text-Knoten
                                                string "Flug 1"
                                                fontStyle FontStyle {           Aufruf des FontStyle-Knotens
                                                    size 0.035
                                                    family "SANS"
                                                    style "BOLD"
                                                }                               Ende des Font-Style-Knotens
                                            }                                   Ende des Text-Knotens
                                        }                                       Ende des Shape-Knotens
                                    ]
                                }                                               Ende Transform #3
                            ]
                        }                                                       Ende Transform #2
                    ]
                }                                                               Ende Transform #1
                DEF FlugStart TouchSensor {}                                    Definition TouchSensor auf Text
        ]
}

ROUTE User.position_changed TO box.set_translation                              Position und Orientierung des Benutzers
ROUTE User.orientation_changed TO box.set_rotation                                wird an den Knoten User weitergegeben

# Flug1                                                                         Beginn der Beschreibung von Flug 1

DEF Guide Transform {                                                           Transform #1
        translation 1000 1000 1000
        children [
                Transform {                                                     Transform #2
                        translation 1000 1000 1000
                        children [
                                Shape {                                         Beginn Shape-Knoten
                                        geometry Sphere {radius 1}              Definition einer Kugel mit Radius 1
                                }                                               Ende des Shape-Knotens
                                DEF FlugtestStart TouchSensor {}                Definition TouchSensor auf der Kugel
                        ]
                }                                                               Ende Transform #2
                DEF FlugVP Viewpoint {                                          Definition eines Viewpoints
                        position 0 0 0
                        orientation 0 0 0 0
                        description "Flug"
                }
        ]
}                                                                               Ende Transform #1

DEF FlugTime TimeSensor {                                                       Definition eines TimeSensors
        loop FALSE
        enabled TRUE
        cycleInterval 60
}
DEF FlugPInterpol PositionInterpolator {                                        Beginn des Knotens PositionInterpolator
        key [
                0,0.02,0.05,0.1,0.15,0.2,0.25,0.3,0.35,                         Angabe von Zeitmarken in einem Intervall
                0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,                              von 0 bis 1, an welchen der nächste
                0.8,0.9,0.95,0.98,0.99,1                                          Stützwert erreicht werden sollte
        ]
        keyValue [
                0 400 600,                                                      Liste aller Stützwerte, an denen man
                -11 400 503.2,                                                    während des Fluges vorbeikommt
                -47 381 381,
                -144 213 336.
                -172 132 221,
                -184 39 68,
                -99 -9 -59,
                -50 -15 -108,
                -30 -15 -125,
                45 -4 -172,#biel
                62 -4.3 -197,
                76 -1.76 -227.5,
                99 7.9 -293,
                104 7.9 -296,#ausicht
```

104 7.9 -296, 104 7.9 -296, 68 16 -187, 30 -5 -9, 30 27 -9, 72 49 74, 41 62 291, 234 331 585, 0 400 600]	
}	Ende des Knotens PositionInterpolator
DEF FlugOInterpol OrientationInterpolator {	Beginn des Knotens OrientationInterpolator
key [0,0.02,0.05,0.1,0.15,0.2,0.25,0.3,0.35, 0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75, 0.8,0.9,0.95,0.98,0.99,1]	Angabe von Zeitmarken in einem Intervall von 0 bis 1, an welchen die nächste Orientierung der folgenden Liste aktiv sein sollte
keyValue [1 0 0 -0.7, -0.92 0.36 0.13 0.75, -0.94 0.3 0.13 0.89, -0.9 0.38 0.13 0.7, -0.96 0.25 0.07 0.61, -0.9 -0.39 -0.08 0.46, -0.33 -0.93 -0.12 0.75, 0 -1 0 0.8, 0 -1 0 1, 0 -1 0 1,#biel 0 -1 0 0.57, 0 -1 0 0.24, 0 -1 0 0.49, 0 -1 0 1.4,#ausicht 0 1 0 3.5, 0 1 0 2.85, 0 1 0 2.85, 0 1 0 3, 0 1 0 2, 0 1 0 1.4, 0 1 0 0.57, -0.64 0.74 0.17 0.68, 1 0 0 -0.7]	Liste der Orientierungen, die an den jeweiligen Stützstellen aktiv sind
}	Ende des Knotens OrientationInterpolator
	Zuweisungen:
ROUTE FlugTime.isActive TO FlugVP.set_bind	Aufruf des vordefinierten Viewpoints
ROUTE FlugStart.touchTime TO FlugTime.startTime	Zustand des TouchSensors wird an Knoten Flug3Time weitergegeben (Start!)
ROUTE FlugTime.fraction_changed TO FlugPInterpol.set_fraction ROUTE FlugTime.fraction_changed TO FlugOInterpol.set_fraction	Verstrichene Zeit (in Zeitintervall zw.ischen 0 und 1) wird an die beiden Interpolations-Knoten weitergegeben
ROUTE FlugPInterpol.value_changed TO Guide.set_translation ROUTE FlugOInterpol.value_changed TO Guide.set_rotation	Interpolierte Werte für Position und Orientierung werden an den Flug (Guide3) weitergegeben
#	
#linkbox flug2###	
#	
Group {	Beginn Gruppen-Knoten
children [	
DEF box2 Transform {	Transform #1
translation 0 0 14	
children [	
Transform {	Transform #2
translation -0.65 0.2 -1	
children [	
DEF BoxxTest Shape {	Beginn der Definition des geometr. Objekts
geometry Box {	Beginn Box-Knoten
size 0.12 0.05 0.01	
}	Ende des Box-Knotens
}	Ende des Shape-Knotens
DEF VertexText Transform {	Transform #3
translation -0.04 -0.01 0.01	
children [	
Shape {	Beginn der Definition des geometr. Objekts
appearance Appearance {	Aufruf des Appearance-Knotens
material Material {	Aufruf des Material-Knotens
diffuseColor 0 0 0	Zuweisen der Material-Eigenschaften
}	Ende des Material-Knotens

```
                                        }                                   Ende des Appearance-Knotens
                                        geometry Text {                     Beginn Text-Knoten
                                            string "Flug 2"
                                            fontStyle FontStyle {           Aufruf des FontStyle-Knotens
                                                size 0.035
                                                family "SANS"
                                                style "BOLD"
                                            }                               Ende des Font-Style-Knotens
                                        }                                   Ende des Text-Knotens
                                    }                                       Ende des Shape-Knotens
                                ]
                            }                                               Ende Transform #3
                        ]
                    }                                                       Ende Transform #2
                ]
            }                                                               Ende Transform #1
            DEF Flug2Start TouchSensor {}                                   Definition TouchSensor auf Text
        ]
}

ROUTE User.position_changed TO box2.set_translation                         Position und Orientierung des Benutzers
ROUTE User.orientation_changed TO box2.set_rotation                            wird an den Knoten User weitergegeben

#flug2                                                                      Beginn der Beschreibung von Flug 2

DEF Guide2 Transform {                                                      Transform #1
    translation 1000 1000 1000
    children [
        Transform {                                                         Transform #2
            translation 1000 1000 1000
            children [
                Shape {                                                     Beginn Shape-Knoten
                    geometry Sphere {radius 1}                              Definition einer Kugel mit Radius 1
                }                                                           Ende des Shape-Knotens
                DEF FlugtestStart TouchSensor {}                            Definition TouchSensor auf der Kugel
            ]
        }                                                                   Ende Transform #2
        DEF Flug2VP Viewpoint {                                             Definition eines Viewpoints
            position 0 0 0
            orientation 0 0 0 0
            description "Flug2"
        }
    ]
}                                                                           Ende Transform #1

DEF Flug2Time TimeSensor {                                                  Definition eines TimeSensors
    loop FALSE
    enabled TRUE
    cycleInterval 60
}
DEF Flug2PInterpol PositionInterpolator {                                   Beginn des Knotens PositionInterpolator
    key [
        0,0.3,0.5,0.8,1                                                     Angabe von Zeitmarken in einem Intervall
    ]                                                                          von 0 bis 1, an welchen der nächste
    keyValue [                                                                 Stützwert erreicht werden sollte
        0 400 600,                                                          Liste aller Stützwerte, an denen man
        600 400 0,                                                             während des Fluges vorbeikommt
        0 400 -600,
        -600 400 0,
        0 400 600
    ]
}                                                                           Ende des Knotens PositionInterpolator
DEF Flug2OInterpol OrientationInterpolator {                                Beginn des Knotens OrientationInterpolator
    key [                                                                   Angabe von Zeitmarken in einem Intervall
        0,0.3,0.5,0.8,1                                                        von 0 bis 1, an welchen die nächste
    ]                                                                          Orientierung der folgenden Liste aktiv
    keyValue [                                                                 sein sollte
        1 0 0 -0.7,
        -0.32 0.8949 0.32 1.6814,                                           Liste der Orientierungen, die an den
        0 0.9317 0.3631 3.1498,                                                jeweiligen Stützstellen aktiv sind
        -0.32 -0.8949 -0.32 1.6814,
        1 0 0 -0.7
    ]
}                                                                           Ende des Knotens OrientationInterpolator
                                                                            Zuweisungen:
ROUTE Flug2Time.isActive TO Flug2VP.set_bind                                Aufruf des vordefinierten Viewpoints
ROUTE Flug2Start.touchTime TO Flug2Time.startTime                           Zustand des TouchSensors wird an Knoten
```

```
                                                                    Flug2Time weitergegeben (Start!)
ROUTE Flug2Time.fraction_changed TO Flug2PInterpol.set_fraction     Verstrichene Zeit (in Zeitintervall zw.ischen
ROUTE Flug2Time.fraction_changed TO Flug2OInterpol.set_fraction         0 und 1) wird an die beiden
                                                                        Interpolations-Knoten weitergegeben
ROUTE Flug2PInterpol.value_changed TO Guide2.set_translation        Interpolierte Werte für Position und
ROUTE Flug2OInterpol.value_changed TO Guide2.set_rotation               Orientierung werden an den Flug
                                                                        (Guide2) weitergegeben
#
#linkbox flug3##########################################
#
Group {                                                             Beginn Gruppen-Knoten
    children [
        DEF box3 Transform {                                        Transform #1
            translation 0 0 14
            children [
                Transform {                                         Transform #2
                    translation  -0.65 0.1 -1
                    children [
                        DEF BoxxTest Shape {                        Beginn der Definition des geometr. Objekts
                            geometry Box {                          Beginn Box-Knoten
                                size 0.12 0.05 0.01
                            }                                       Ende des Box-Knotens
                        }                                           Ende des Shape-Knotens
                        DEF VertexText Transform {                  Transform #3
                            translation -0.04 -0.01 0.01
                            children [
                                Shape {                             Beginn der Definition des geometr. Objekts
                                    appearance Appearance {         Aufruf des Appearance-Knotens
                                        material Material {         Aufruf des Material-Knotens
                                            diffuseColor 0 0 0      Zuweisen der Material-Eigenschaften
                                        }                           Ende des Material-Knotens
                                    }                               Ende des Appearance-Knotens
                                    geometry Text {                 Beginn Text-Knoten
                                        string "Flug 3"
                                        fontStyle FontStyle {       Aufruf des FontStyle-Knotens
                                            size 0.035
                                            family "SANS"
                                            style "BOLD"
                                        }                           Ende des Font-Style-Knotens
                                    }                               Ende des Text-Knotens
                                }                                   Ende des Shape-Knotens
                            ]
                        }                                           Ende Transform #3
                    ]
                }                                                   Ende Transform #2
            ]
        }                                                           Ende Transform #1
        DEF Flug3Start TouchSensor {}                               Definition TouchSensor auf Text
    ]
}

ROUTE User.position_changed TO box3.set_translation                 Position und Orientierung des Benutzers
ROUTE User.orientation_changed TO box3.set_rotation                     wird an den Knoten User weitergegeben

#flug3                                                             Beginn der Beschreibung von Flug 3

DEF Guide3 Transform {                                              Transform #1
    translation 1000 1000 1000
    children [
        Transform {                                                 Transform #2
            translation 1000 1000 1000
            children [
                Shape {                                             Beginn Shape-Knoten
                    geometry Sphere {radius 1}                      Definition einer Kugel mit Radius 1
                }                                                   Ende des Shape-Knotens
                DEF FlugtestStart TouchSensor {}                    Definition TouchSensor auf der Kugel
            ]
        }                                                           Ende Transform #2
        DEF Flug3VP Viewpoint {                                     Definition eines Viewpoints
            position 0 0 0
            orientation 0 0 0 0
            description "Flug2"
        }
    ]
}                                                                   Ende Transform #1

DEF Flug3Time TimeSensor {                                          Definition eines TimeSensors
```

loop FALSE enabled TRUE cycleInterval 60 }	
DEF Flug3PInterpol PositionInterpolator {	Beginn des Knotens PositionInterpolator
key [0,0.1,0.2,0.3,0.4,0.5,0.6,0.8,0.9,1]	Angabe von Zeitmarken in einem Intervall von 0 bis 1, an welchen der nächste Stützwert erreicht werden sollte
keyValue [0 400 600, -22 128 743, -480 146 697, -727 166 13, -507 184 -497, -4.9 178 -739, 417 130 -619, 727 166 4.6, 507 177 496, 0 400 600]	Liste aller Stützwerte, an denen man während des Fluges vorbeikommt
}	Ende des Knotens PositionInterpolator
DEF Flug3OInterpol OrientationInterpolator {	Beginn des Knotens OrientationInterpolator
key [0,0.1,0.2,0.3,0.4,0.5,0.6,0.8,0.9,1]	Angabe von Zeitmarken in einem Intervall von 0 bis 1, an welchen die nächste Orientierung der folgenden Liste aktiv sein sollte
keyValue [1 0 0 -0.7, -1 -0.1 0 0.27, -0.58 -0.8 -0.17 0.47, -0.15 -0.97 -0.17 1.57, 0 0.98 0.17 3.97, 0 0.98 0.17 3.15, 0 0.98 0.17 2.57, 0 0.98 0.17 1.58, 0 0.98 0.17 0.9, 1 0 0 -0.7]	Liste der Orientierungen, die an den jeweiligen Stützstellen aktiv sind
}	Ende des Knotens OrientationInterpolator
	Zuweisungen:
ROUTE Flug3Time.isActive TO Flug3VP.set_bind	Aufruf des vordefinierten Viewpoints
ROUTE Flug3Start.touchTime TO Flug3Time.startTime	Zustand des TouchSensors wird an Knoten Flug3Time weitergegeben (Start!)
ROUTE Flug3Time.fraction_changed TO Flug3PInterpol.set_fraction ROUTE Flug3Time.fraction_changed TO Flug3OInterpol.set_fraction	Verstrichene Zeit (in Zeitintervall zw.ischen 0 und 1) wird an die beiden Interpolations-Knoten weitergegeben
ROUTE Flug3PInterpol.value_changed TO Guide3.set_translation ROUTE Flug3OInterpol.value_changed TO Guide3.set_rotation	Interpolierte Werte für Position und Orientierung werden an den Flug (Guide3) weitergegeben

2.2.2 Koord.wrl

Die Datei *Koord.wrl* beinhaltet das eigentliche Geländemodell sowie alle Eigenschaften desselben, wie die geometrische Form und das äussere Erscheinungsbild. Ebenfalls in dieser Datei beschrieben wird die Überlagerung des Geländes mit einem Bild (*biel-flaechen.jpg* oder *biel-flaechen.gif*).

```
NavigationInfo { headlight FALSE }                                  Ausschalten der Standardbeleuchtung
Transform {                                                         Transform #1
    children [
        DEF AdTIN_MediumSpringGreen Transform {                     Transform #2
            children [
                Shape {                                             Beginn der Definition des geometr. Objekts
                    appearance Appearance {                         Aufruf des Appearance-Knotens
                        material Material {                         Aufruf des Material-Knotens
                            ambientIntensity 0.0499347              Zuweisen der Material-Eigenschaften
                            diffuseColor     0.498039 1 0
                            specularColor    0.3 0.3 0.3
                            shininess        0.1
                        }                                           Ende der Material-Knotens
                        texture ImageTexture {                      Aufruf des ImageTexture-Knotens
                        url ["biel-flaechen.jpg","biel-flaechen.gif"] Verweis auf Bild-Datei
                        }                                           Ende des Image-Texture-Knotens
                        textureTransform TextureTransform {         Aufruf des TextureTransform-Knotens
                            scale 1 -1
                        }                                           Ende des TextureTransform-Knotens
                    }                                               Ende des Appearance-Knotens
                    geometry IndexedFaceSet {                       Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                                 Beginn der Koordinatenliste
                                -353.239  -18.1565   350.884,
                                -353.239  -17.8268   353.239,
                                -348.529  -18.4862   348.529,
                                -236.67   -21.4534   321.448,
                                -236.67   -20.2288   295.543,
                                ...
                                ...                                 Liste mit 3834 Geländepunkten
                                ...
                                273.172   -17.1674  -353.239,
                                279.059   -18.5333  -352.062,
                                281.414   -18.4391  -353.239,
                                287.301   -19.7578  -348.529,
                                353.239   -21.0766  -353.239
                            ]                                       Ende der Koordinatenliste
                        }
                        coordIndex [                                Zuweisung der Punkte an Polygonseiten
                            0, 1, 2, -1,
                            3, 4, 5, -1,
                            6, 7, 8, -1,
                            7, 6, 9, -1,
                            10, 11, 12, -1,
                            ...
                            ...                                     Liste aller Polygonseiten (Geländeflächen)
                            ...
                            3835, 3811, 3832, -1,
                            3841, 3838, 3842, -1,
                            3841, 3575, 3838, -1,
                            3846, 3524, 3646, -1,
                            3524, 3846, 3833, -1
                        ]
                        normalPerVertex FALSE                       Zuordnung der Normalen an Eckpunkte
                        creaseAngle 3.14                            Polygonecken im Uhrzeigersinn definiert
                    }                                               Ende des IndexedFaceSet-Knotens
                }                                                   Ende des Shape-Knotens
            ]
        }                                                           Ende des 2. Transform-Knotens
    ]
}                                                                   Ende des 1. Transform-Knotens
```

2.2.3 text_Biel2.wrl

Die 3D-Schriften wurden allesamt im Programm Simply3D erstellt; für eine kurze Beschreibung siehe Datei *text_Biel2.wrl* im Modell *seen.wrl* (Anhang I, Kapitel 1.2.5).

3. Modell Biel

3.1 Dateistruktur

Das Modell Seen besteht aus folgenden Dateien:

- biel.wrl
- Biel_Gel.wrl
- Biel_Geb.wrl
- Biel_Wald.wrl
- arteplage.wrl
- see.gif

Die eigentliche Strukturierung des Modells erfolgt in der Datei *biel.wrl*, von wo aus alle anderen wrl-Files aufgerufen und im Modell positioniert werden. Das eigentliche Gelände (Koordinaten der Punkte sowie Definition der Polygonflächen) befindet sich in der Datei *Biel_Gel.wrl*. Dort wird auch die Bilddatei *see.gif* aufgerufen, welche dem Gelände überlagert wird. In diesem Modell werden zusätzlich noch andere 3D-Modelle integriert (Häuser und Wald), welche in den Dateien *Biel_Geb.wrl* und *Biel_Wald.wrl* enthalten sind. Ausserdem wird ein rudimentäres Modell der Arteplage von Biel (*arteplage.wrl*) in das Modell eingefügt.

3.2 Beschreibung der wrl-Files

3.2.1 biel.wrl

Die Datei *biel.wrl* beinhaltet alle Grundeinstellungen, welche für die Darstellung der Szene benötigt werden (Beleuchtung, Viewpoints, Hintergrund, etc.). Die einzelnen Teile des eigentlichen Modells (Geländemodell und andere Objekte) werden aus anderen Dateien geladen. Zusätzlich wird auch die Verknüpfung zu Informationen im WWW sichergestellt.

```
DEF User ProximitySensor {                          Definition des ProximitySensors
    size 2000 2000 2000
}
Background {                                        Definition des Hintergrundes
    groundColor [0.4 0 1, 0.4 0 1]
    groundAngle [1.5]
    skyColor [ 0.3 0.5 0.9, 0.3 0.5 0.9, 0.8 0.9 1, 0.8 0.9 1]
    skyAngle [1.3, 1.7, 2]
}
NavigationInfo {                                    Ausschalten der Standardbeleuchtung
    headlight FALSE
}
DEF Licht1 DirectionalLight {                       Definiton eines gerichteten Lichtes
    direction 1 -1 1
    on TRUE
}
DEF Licht2 DirectionalLight {                       Definition eines 2. gerichteten Lichtes aus
    direction -1 -1 -1                              einer etwas anderen Richung, um die Szene
    intensity 0.4                                   besser auszuleuchten
}
DEF sued Viewpoint {                                Definition derStandardblickpunkte aus den
    position 0 400 600                              Richtungen Süd, Ost, Nord und West
    orientation 1 0 0 -0.7
    description "Sicht von Sued"
}
DEF ost Viewpoint {
    position 600 400 0
    orientation -0.32 0.8949 0.32 1.6814
    description "Sicht von Ost"
}
DEF nord Viewpoint {
    position 0 400 -600
    orientation 0 0.9317 0.3631 3.1498
    description "Sicht von Norden"
}
DEF west Viewpoint {
    position -600 400 0
    orientation -0.32 -0.8949 -0.32 1.6814
    description "Sicht von Westen"
}
DEF OBJEKTE Group {                                 Gruppen-Knoten, der alle zu ladenden
    children [                                      Objekte des Modells umfasst
        DEF Gebaeude Transform {                    Transform-Knoten (Gebäude)
            translation 12 0 12
            scale 0.97 1 0.97
            children [
                Inline {url "Biel_Geb.wrl"}         Verknüpfung mit der Gebäude-Datei
            ]
        }                                           Ende des Transform-Knotens (Gebäude)

        DEF Wald Transform {                        Transform-Knoten (Wald)
            translation -27 38.1380 0
            children [
                Inline {url "Biel_Wald.wrl"}        Verknüpfung mit der Wald-Datei
            ]
        }                                           Ende des Transform-Knotens (Wald)
        DEF Gelaende Transform {                    Transform-Knoten (Gelände)
            translation -15 35 -5
            children [
                Inline {url "Biel_Gel.wrl"}         Verknüpfung mit der Gelände-Datei
```

```
            ]
        }                                                   Ende des Transform-Knotens (Gelände)
        DEF Arteplage Anchor {                              Beginn des Anchor-Knoten (Arteplage)
            description "Arteplage Biel"                    Beschreibung des Links
            url "http://www.expo-pk.ch/de/art/biel/index.html"  URL zu Datei auf dem WWW
            children [
                DEF Arte_Biel Transform {                   Transform-Knoten (Arteplage)
                    translation -78 -16 -7
                    scale 0.3 0.3 0.3
                    rotation 1 0 0 1.57
                    children Transform {                    Zweiter Transform-Knoten für Arteplage
                        rotation 0 0 1 2.6
                        children [
                            Inline {url "arteplage.wrl"}    Verknüpfung mit der Arteplage-Datei
                        ]
                    }                                       Ende 2. Transform-Knoten (Arteplage)
                }                                           Ende 1. Transform-Knoten (Arteplage)
            ]
        }                                                   Ende des Anchor-Knotens (Arteplage)
    ]
}                                                           Ende des Gruppen-Knotens
```

3.2.2 Biel_Gel.wrl

Die Datei *Biel_Gel.wrl* beinhaltet vor allem das eigentliche Geländemodell sowie alle Eigenschaften desselben, wie die geometrische Form und das äussere Erscheinungsbild. Ebenfalls in dieser Datei beschrieben wird die Überlagerung des Geländes mit einem Bild (*see.gif*).

```
NavigationInfo { headlight FALSE }                          Ausschalten der Standardbeleuchtung
Transform {                                                 Aufruf des Transform-Knotens
    children [
        DEF TIN___0_Bronze Transform {                      Aufruf eines zweiten Transform-Knotens
            children [
                Shape {                                     Beginn der Definition des geometr. Objekts
                    appearance Appearance {                 Aufruf des Appearance-Knotens
                        material Material {                 Aufruf des Material-Knotens
                            ambientIntensity 0.0386667      Zuweisen der Material-Eigenschaften
                            diffuseColor    0.55 0.47 0.14
                            specularColor   0.3 0.3 0.3
                            shininess       0.1
                        }                                   Ende der Material-Knotens
                        texture ImageTexture {              Aufruf des ImageTexture-Knotens
                        url "see.gif"                       Verweis auf Bild-Datei
                        }                                   Ende des Image-Texture-Knotens
                        textureTransform TextureTransform { Aufruf des TextureTransform-Knotens
                        scale 0.985 -1.35
                        }                                   Ende des TextureTransform-Knotens
                    }                                       Ende des Appearance-Knotens
                    geometry IndexedFaceSet {               Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                         Beginn der Koordinatenliste
                                -400.686 -55.3485 283.151,
                                -400.686 -55.3485 293.836,
                                -390.001 -55.3485 293.836,
                                16.0274 -53.211 -112.192,
                                16.0274 -53.2109 -122.877,
                                ...
                                ...                         Liste mit 1733 Geländepunkten
                                ...
                                347.261 -51.9287 -186.987,
                                379.316 -51.9287 -176.302,
                                112.192 -51.5011 -272.467,
                                58.7673 -51.9286 -197.672,
                                58.7673 -52.3561 -176.302
                            ]                               Ende der Koordinatenliste
                        }
                        coordIndex [                        Zuweisung der Punkte an Polygonseiten
                            0, 1, 2, -1,
                            3, 4, 5, -1,
                            6, 7, 8, -1,
                            9, 7, 6, -1,
                            10, 9, 6, -1,
                            ...
                            ...                             Liste aller Polygonseiten (Geländeflächen)
                            ...
                            637, 1550, 1712, -1,
                            1690, 1585, 1702, -1,
                            1701, 1690, 1702, -1,
                            1692, 1730, 1732, -1,
                            1608, 1692, 1732, -1
                        ]
                        normalPerVertex FALSE               Zuordnung der Normalen an Eckpunkte
                        creaseAngle 3.14                    Polygonecken im Uhrzeigersinn definiert
                    }                                       Ende des IndexedFaceSet-Knotens
                }                                           Ende des Shape-Knotens
            ]
        }                                                   Ende des 2. Transform-Knotens
    ]
}                                                           Ende des 1. Transform-Knotens
```

3.2.3 Biel_Geb.wrl

Die Datei *Biel_Geb.wrl* beinhaltet Häuser der Stadt Biel in 3D-Form. Ähnlich wie beim Geländemodell sind die Häuser durch die Angabe der Eckpunkte sowie die Definition der Polygonflächen bestimmt.

```
NavigationInfo { headlight FALSE }                          Ausschalten der Standardbeleuchtung
Transform {                                                 Aufruf des Transform-Knotens
    children [
        DEF Layer0_LightGray Transform {                    Aufruf eines zweiten Transform-Knotens
            children [
                Shape {                                     Beginn der Definition des geometr. Objekts
                    appearance Appearance {                 Aufruf des Appearance-Knotens
                        material Material {                 Aufruf des Material-Knotens
                            ambientIntensity 0.065882       Zuweisen der Material-Eigenschaften
                            diffuseColor     0.658824 0.658824 0.658824
                            specularColor    0.3 0.3 0.3
                            shininess        0.1
                        }                                   Ende der Material-Knotens
                    }                                       Ende des Appearance-Knotens
                    geometry IndexedFaceSet {               Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                         Beginn der Koordinatenliste
                                -105.039 -15.8097 202.465,
                                -84.3093 -15.8098 212.278,
                                -90.1932 -15.8098 226.75,
                                -111.441 -15.8098 216.814,
                                -64.5327 -15.8098 250.912,
                                ...
                                ...                         Liste mit 2216 Gebäudepunkten
                                ...
                                291.364 -17.1158 -299.611,
                                208.09 -17.3413 -214.424,
                                205.284 -17.3413 -218.448,
                                178.017 -17.3413 -195.798,
                                174.938 -17.3413 -199.577
                            ]                               Ende der Koordinatenliste
                        }
                        coordIndex [                        Zuweisung der Punkte an Polygonseiten
                            0, 1, 2, -1,
                            3, 0, 2, -1,
                            4, 5, 6, -1,
                            7, 4, 6, -1,
                            8, 9, 10, -1,
                            ...
                            ...                             Liste aller Polygonseiten (Hausseiten)
                            ...
                            1103, 2214, 2212, -1,
                            1104, 1105, 2215, -1,
                            1104, 2215, 2214, -1,
                            1105, 1106, 2213, -1,
                            1105, 2213, 2215, -1
                        ]
                        solid FALSE                         Dreiecke von beiden Seiten sichtbar
                        normalPerVertex FALSE               Zuordnung der Normalen an Eckpunkte
                        ccw            TRUE                 Polygonecken im Uhrzeigersinn definiert
                    }                                       Ende des IndexedFaceSet-Knotens
                }                                           Ende des Shape-Knotens
            ]
        }                                                   Ende des 2. Transform-Knotens
    ]
}                                                           Ende des 1. Transform-Knotens
```

3.2.4 Biel_Wald.wrl

Die Datei *Biel_Wald.wrl* enthält die Definition sowie die Position jedes einzelnen Baumes, d.h. die Koordinaten der Baumpunkte sowie die Polygonseiten, aus denen die Bäume bestehen.

```
NavigationInfo { headlight FALSE }                          Ausschalten der Standardbeleuchtung
Transform {                                                 Aufruf des Transform-Knotens
    children [
        DEF Bosco_MediumSpringGreen Transform {             Aufruf eines zweiten Transform-Knotens
            children [
                Shape {                                     Beginn der Definition des geometr. Objekts
                    appearance Appearance {                 Aufruf des Appearance-Knotens
                        material Material {                 Aufruf des Material-Knotens
                            ambientIntensity 0.0499347      Zuweisen der Material-Eigenschaften
                            emissiveColor  0.2 0.5 0
                            diffuseColor     0.498039 1 0
                            specularColor    0.3 0.3 0.3
                            shininess        0.1
                        }                                   Ende der Material-Knotens
                    }                                       Ende des Appearance-Knotens
                    geometry IndexedFaceSet {               Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                         Beginn der Koordinatenliste
                                -399.724 -37.8625 -72.8524,
                                -396.193 -37.8625 -76.3841,
                                -397.959 -32.5649 -74.6182,
                                -399.724 -37.8625 -76.3841,
                                -396.193 -37.8625 -72.8524,
                                ...
                                ...                         Liste mit 13353 Baumpunkten
                                ...
                                396.204 -56.3278 174.235,
                                399.735 -56.3278 170.703,
                                397.969 -51.0302 172.469,
                                396.204 -56.3278 170.703,
                                399.735 -56.3278 174.235
                            ]                               Ende der Koordinatenliste
                        }
                        coordIndex [                        Zuweisung der Punkte an Polygonseiten
                            0, 1, 2, -1,
                            3, 4, 2, -1,
                            5, 6, 7, -1,
                            8, 9, 7, -1,
                            10, 11, 12, -1,
                            ...
                            ...                             Liste aller Polygonseiten (Bäume)
                            ...
                            13343, 13344, 13342, -1,
                            13345, 13346, 13347, -1,
                            13348, 13349, 13347, -1,
                            13350, 13351, 13352, -1,
                            13353, 13354, 13352, -1
                        ]
                        solid FALSE                         Dreiecke von beiden Seiten sichtbar
                        normalPerVertex FALSE               Zuordnung der Normalen an Eckpunkte
                        ccw TRUE                            Polygonecken im Uhrzeigersinn definiert
                    }                                       Ende des IndexedFaceSet-Knotens
                }                                           Ende des Shape-Knotens
            ]
        }                                                   Ende des 2. Transform-Knotens
    ]
}                                                           Ende des 1. Transform-Knotens
```

3.2.5 arteplage.wrl

Dieses Modell der Arteplage von Biel wurde anhand von Daten erstellt, welche selber aus Bildern erfasst wurden. Die so erhaltenen Daten bilden die Grundlage, auf welcher ein geometrischer Knoten (IndexedFaceSet) erstellt wurde.

```
Transform {                                                  Ausschalten der Standardbeleuchtung
    children [                                               Aufruf des Transform-Knotens
        DEF AdTIN_MediumSpringGreen Transform {
            children [                                       Aufruf eines zweiten Transform-Knotens
                Shape {
                    appearance Appearance {                  Beginn der Definition des geometr. Objekts
                        material Material {                  Aufruf des Appearance-Knotens
                            ambientIntensity 0.0499347       Aufruf des Material-Knotens
                            diffuseColor     0.4 0.1 0       Zuweisen der Material-Eigenschaften
                            specularColor    0.3 0.3 0.3
                            shininess        0.1
                            transparency     0.4
                        }                                    Ende der Material-Knotens
                    }                                        Ende des Appearance-Knotens
                    geometry IndexedFaceSet {                Geometrischer Knoten IndexedFaceSet
                        coord Coordinate {
                            point [                          Beginn der Koordinatenliste
                                21 60 1,
                                18 107 0,
                                9 133 2,
                                38 161 0,
                                21 182 1,
                                ...
                                ...                          Liste mit 58 Arteplagepunkten
                                ...
                                446 18 0,
                                424 63 -7,
                                461 63 0,
                                479 112 0,
                                492 141 0
                            ]                                Ende der Koordinatenliste
                        }
                        coordIndex [                         Zuweisung der Punkte an Polygonseiten
                            0, 9, 1, -1,
                            0, 10, 9, -1,
                            1, 9, 2, -1,
                            2, 9, 3, -1,
                            3, 9, 8, -1,
                            ...
                            ...                              Liste aller Polygonseiten (Arteplage)
                            ...
                            51, 57, 50, -1,
                            51, 50, 48, -1,
                            48, 50, 49, -1,
                            50, 57, 58, -1,
                            49, 50, 58, -1
                        ]
                        solid FALSE                          Dreiecke von beiden Seiten sichtbar
                        normalPerVertex FALSE                Zuordnung der Normalen an Eckpunkte
                        ccw FALSE                            Polygonecken im Uhrzeigersinn definiert
                        creaseAngle 0.45                     Glätten der Oberfläche
                    }                                        Ende des IndexedFaceSet-Knotens
                }                                            Ende des Shape-Knotens
            ]
        }                                                    Ende des 2. Transform-Knotens
    ]                                                        Ende des 1. Transform-Knotens
}
```