Jolanda

Technical Report

Jakob.Beetz@archit.uni-weimar.de Mai 2001
Professur InfAR Prof.Donath

Einleitung

Jolanda stellt den kompletten Leistungsumfang eines ausgewachsenen VRML97-Browsers für die Benutzung in einer virtual-reality-Umgebung zur Verfügung. Die derzeit implementierte Version ist dabei für die Verwendung mit HMD und Stylus und der Verwendung eines Polhemus Longrangers zugeschnitten. Die Implementierung weiterer Ein- und Ausgabegeräte ist grundsätzlich möglich und vorgesehen (Fastrack FOB, Stereo-Projektion).
Der entscheidende Vorteil des Systems gegenüber Speziallösungen (VRAM) liegt in der Trennung von Kernteilen des Systems wie Grafikausgabe, Eingabverwaltung, Netzwerkinterfaces usw. von der funktionalen und formalen Gestaltbarkeit der Interaktionsmöglichkeiten. Das enthaltene Scripting-Interface auf Basis von JavaScript bietet eine leicht erlernbare, flexible und dennoch mächtige Schnittstelle zur Realisierung unterschiedlichster Anwendungsmöglichkeiten, die selbst engagierten Laien die Modellierung und Durchführung komplexer interaktiver Anwendungen, z.B. Aus dem architektonischen Umfeld gestatten.
Im Folgenden werden nach einer kurzen Übersicht der zugrundeliegenden Software-Architektur anhand eines einfachen Beispiels (die Erstellung und Löschung von "Wänden") einige grundlegende Verfahren dargestellt, mit deren Hilfe komplexe Aufgaben in Virtuellen Umgebungen durch einfaches scripting auch für Nichtprogrammierer (z.B. Architekturstudenten) zu bewältigen sind. Den Abschluß bilden mögliche Anwendungsfelder in der Lehre / rapid Prototyping / der weiteren Entwicklung auf Grundlage der existierenden Software, sowie Ausblicke auf mögliche Erweiterungen des Basisprogramms durch Spezialisten.

Software-Architektur

Jolanda ist eine Erweiterung des im CommunitySource frei verfügbaren GlView (Holger Grahn, http://www.snafu.de/~hg). GlView bildet ebenfalls die Grundlage des kostenlosen Browser-Plugins "blaxxun Contact" sowie dessen Multiuser-Erweiterungen (Virtual Worlds, Virtual Community). Als Grafikkern kommen wahlweise OpenGL (Win32, Linux) oder Direct3D (Win32) zum Einsatz. Die interne Organisation des SceneGraphs ist eng an derjenigen von VRML97 angelehnt. Das Event-Modell ist ebenfalls eine direkte Abbildung der in der ISO-zertifizierten Spezifikation (ISO/IEC 14772-1:1997) festgeschriebenen Verwaltung von Ereignissen.

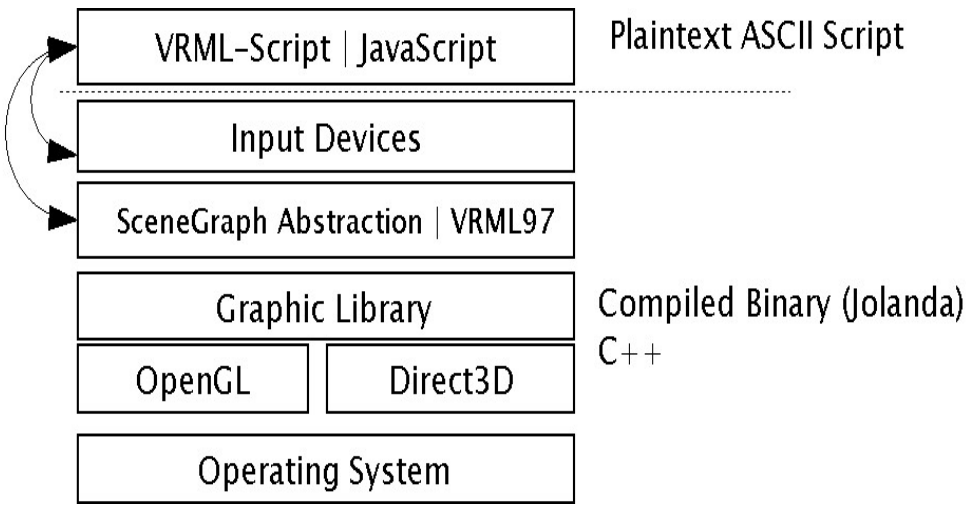


Abb. 1
Basisarchitektur von Jolanda

Die Erweiterung von Jolanda ist stellt im wesentlichen die Anbindung an ein Tracking-System und dessen Kopplung an Kamera-Steuerung und Eingabegerät sowie die Umsetzung der Sensoren (Touch-, Plane-, Cylinder-, Sphere-, Visibility-, Proximity-Sensor) im Raum dar. Die direkte Abbildung der Sensoren in einer echten räumlichen Umgebung hat zur Folge, das prinzipiell alle Interaktionen des Nutzers an einem einfachen, d.h. ungetrackten Arbeitsplatz erstellt und getestet werden können, da sich die Skripte und Sensoren an Bildschirm/Maus genauso verhalten wie mit HMD/Stylus. Dadurch ist es beispielsweise Architekturstudenten möglich komplexe Interaktionen zu Hause zu entwickeln und sie dann "unter der Mütze" einzusetzen. Technisch ist das identische Verhalten von Mauszeiger und Stylus durch die Verschiebung und Drehung des im 2D-Fall ortogonal zur Projektionsebene stehenden Strahles (zur Kollisionsüberprüfung mit Sensoren) entsprechend der Trackerkoordinaten gelöst.

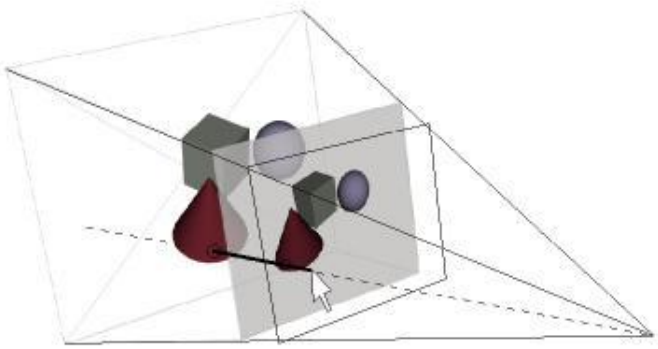


Abb. 2:
Screen-/Mousebasierter VRML-Client:
Viewpointachse enstpricht Selektions-/Kollisionsachse

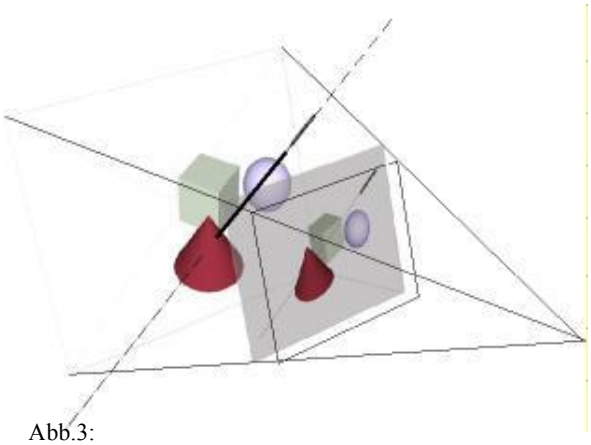


Abb.3:
Screenunabhängiger VRML-Client:
Viewpointachse unabhängig von Selektions-/Kollisionsachse

Die Positionierung von Kamera (HMD) und Kollisionsstrahl (Stylus) ist auf dem derzeitigen Stand der Implementierung "hartverdratet" (Class GvView, Renderloop) und wird über den Menüpunkt "`Options->enable fastrak`" aktiviert[1]. Eleganter und sauberer wäre an dieser Stelle die Einführung abstrackter TrackerNodes, die aus einem Skript heraus an den View oder die Kollisionsüberprüfung gebunden werden können ("binded Viewport" Konzept aus der VRML-Spec, dieses wäre auf die Eingabe noch zu übertragen).
Die Konfiguration des Trackers ist zum derzeitigen Stand ebenfalls nur durch einen neuen Compile zu verändern. Die voreingestellten Werte sind: COM2:, 38000 Baud, Longranger, Receiver 1 HMD, Receiver 2 Stylus. In zukünftigen Versionen sollte eine saubere Konfiguration der entsprechenden Parameter über eine Dialogbox vorgenommen werden können.

Known bugs & limitations:

- irgendwas merkwürdiges passiert mit dem Kamerablickwinkel bei bestimmten Bewegungnen mit dem Stylus (Magnetfeld?)
- Hängt sich bei nicht angeschlossenem Tracker auf, wenn "enable fastrak" aktiviert wird
- Einbindung der Stylusgeometrie unsauber
- Stürzt bei fehlender Stylusgeometrie ab
- autoOffset von Plansensor SPEC-konform in getrackter Variante?

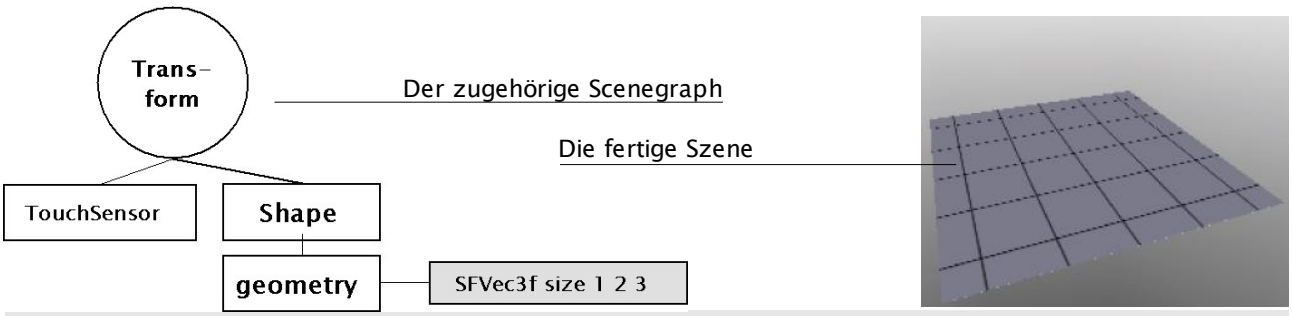
1 eine Datei "stylus.wrl" muß im selben Verzeichnis wie die .exe-Datei liegen. Die Welt(en) müssen vor der Aktivierung dieses Menüpunktes geladen sein. Mit "F4" kann anschliessend der fullscreen-Modus aktiviert werden. Der Menüpunkt "start tracker at startup" sollte nur aktiviert werden, wenn die entsprechenden Einstellungen des Trackers überprüft worden sind und die zu ladenden Dateien per Kommandozeile übergeben werden. Versehentliches aktivieren kann nur in der Registry wieder geändert werden.

Anwendungsbeispiel

Anhand des Zeichnens einer einfachen Geometrie (z.B. einer Wand) und der weiteren Interaktionsmöglichkeiten mit dieser (z.B. dem "Schneiden von Fenstern" oder die Löschung) soll im Folgenden die Einfachheit aber auch die Mächtigkeit der Verwendungsmöglichkeiten der Applikation veranschaulicht werden.
Zum Zeichnen einer einfachen Geometrie reicht zunächst ein einzeler .wrl-file: Die Grundlage bildet eine "Bodenplatte", also eine große flache Box, die mit einem PlaneSensor versehen wird. Dies kann mithilfe geeigneter Modellingprogramme (3DS Max, CosmoWorlds) vorgenommen werden. Drückt der Nutzer nun die Maus- oder Stylustaste wärend sich das Zeigegerät über der Bodenplatte befindet wird ein Feld des PlaneSensors (SFVec3f hitPoint) mit den lokalen Koordinaten des gedachten Schnittpunktes des Zeigegerätes und der Bodenplatte gefüllt. Um diese Koordinaten zum ersten Eckpunkt des Grundrisses der späteren Wand zu machen muß dieses Feld ausgelesen und gespeichert werden. Dies geschieht mittels des sogenannten Routings.

Routing

Geometrien, Materialien und andere (teilweise unsichbare) Bestandteile von VRML-Welten werden durch einzelne, hirarchisch verschachtelte Knoten (Nodes) beschrieben. Die Eigenschaften dieser Knoten werden in Feldern unterschiedlichster Datentypen festgelegt. Die Kantenlänge einer Box wird beispielsweise im Feld "size" festgelegt, das vom Typ SFVec3f ist, also drei Gleitkommawerte aufnehmen kann (Länge, Breite, Höhe. SingelFieldVector3floats).



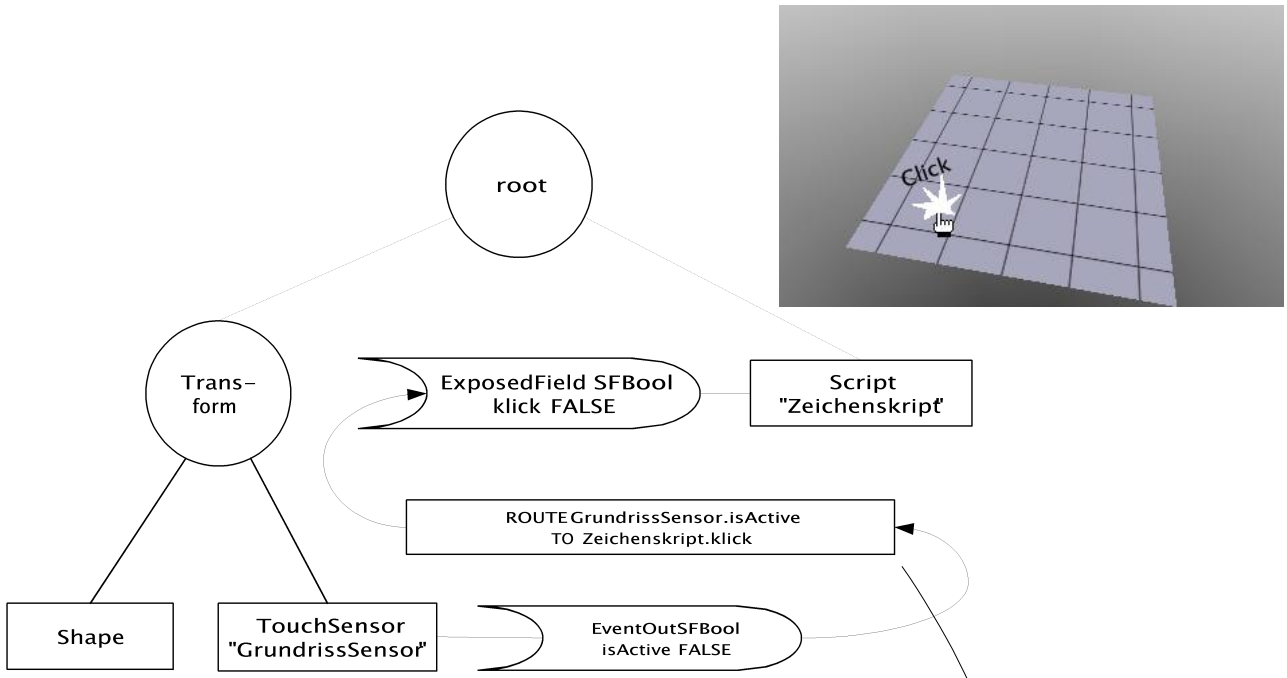
Zugehöriger Quelltext

```
DEF scene Transform
{
  children
  [
      DEF GrundrissSensor TouchSensor {},
      DEF Bodenplatte Transform
      {
      geometry Box
      {
                  size 100 1 100
            }
      }
  ]
}
```

Abb. 4:
einfache Szene zur Erstellung einer Interaktion

Die meisten dieser Felder können zur Laufzeit (also nach dem ersten Laden und der Betrachtung der Welt) verändert werden. Umgekehrt lösen veränderte Werte eines Feldes durch den Browser Nachrichten (Events) aus, die weitervarbeitet werden können. Routing bezeichnet nun das Verknüpfen der Veränderung eines Wertes mit der Veränderung eines anderen Wertes des gleichen Datentyps. In der Philosophie von VRML stellt ein Skript ebenfalls (wie die Geometrien) einen Knoten dar.

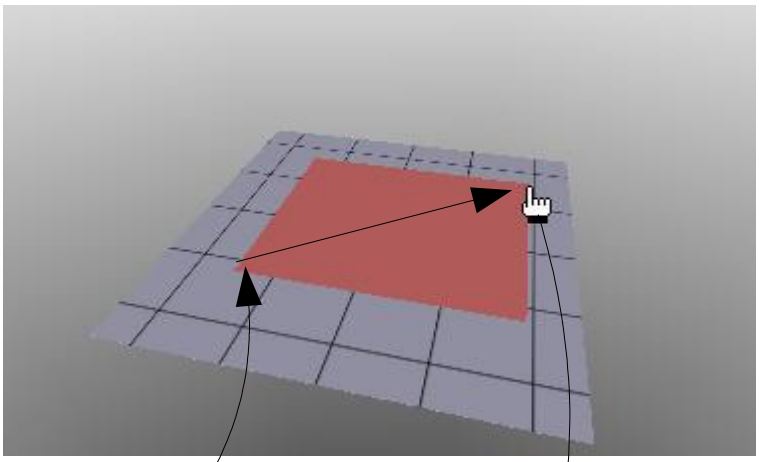
In diesem Fall wird das auftretende Ereignis (Benutzer drückt Knopf über dem PlaneSensor der Bodenplatte, die Veränderung des Wahrheitswertes SFBool [TouchSensor].isActive von FALSE nach TRUE) in eine Funktion eines Skriptes geleitet, das zusammen mit den Geometriedaten im .wrl-file der Welt steht.
Um das auftretende Ereignis auszuwerten wird zunächst ein SkriptNode erstellt, ein Eingangsfeld eventIn vom Wahrheitstyp SFBool als "Funktionskopf" erstellt und vermittels der Zeile ROUTE [TouchSensor].isActive TO [SkriptNode].[meineFunktion] mit der Behandlungsroutine verknüpft.




```
DEF temporaerWand Transform
{
     children Shape
     {
             geometry    Box{}
     }
}
DEF Zeichenskript
{
eventIn SFBool klick
url  [ "javascript:"
function klick (value, timestamp)
{
     active = value;          // Aktviert-Status speichern
     if (value == false)
     {
          scalenew[1] = 1;  // die Hilfsgeometrie sichbar machen
     }
}"]}
ROUTE GrundrissSensor.isActive TO Zeichnungsskript.klick
```

Abb. 5
Implementierung eines einfachen Sensors

Im Rumpf der Funktion wird nun die erste (lokale) Koordinate als Eckpunkt des Grundrisses der neuen Geometrie gespeichert und jede weitere Veränderung dieses Schnittpunktes ([TouchSensor].translation_changed) durch o.a. Verfahren in eine weitere Funktion des Skriptes geleitet. Solange der Knopf gedrückt gehalten wird liegen von nun an zunächst die beiden Eckpunkte der Grundrissfläche der neuen Wand vor. Eine vorher in die Szene eingefügte Hilfsgeometrie, hier eine Box, wird in der zweiten Funktion nun ständig auf diese Größe skaliert und in der Welt als visuelles Feedback gezeichnet (ROUTE [SkriptNode].[neueSkalierung] TO [Hilfsgeometrie].[neueSkalierung] und ROUTE [SkriptNode].[neuePosition] TO [Hilfsgeometrie].[neuePosition]).



```
function touchChange (value, time)
{
     if (active == false)
     {
        startvec = value;

    }
     else
     {
        // skalierung : erster Punkt - zweiter Punkt /2
        neueGroesse[0] = Math.abs((startvec[0] - value[0])/2);
        neueGroesse[2] = Math.abs((startvec[2] - value[2])/2);
        // Bewegung: Zentrum in erster Punkt - zweiter Punkt /2
        Zentrum [1] = 6;
        Zentrum[0] = (value[0] + startvec[0]) / 2;
        Zentrum[2] = (startvec[2] + value [2]) /2;
        endvec = value;
    }
}
ROUTE Zeichnungsskript.neueGroesse TO temporaerWand.scale
ROUTE Zeichnungsskript.Zentrum TO temporaerWand.translation
```

Abb. 6:
Skript zum Erstellen des Grundrisses der Wand

Die Hilfsgeometrie ist ihrerseits ebenfalls mit einem senkrecht angeordneten PlaneSensor versehen, der "scharfgeschaltet" wird, sobald der Benutzer durch loslassen der Maustaste den Grundriss der neuen Wand engültig festgelegt hat. Ein gehaltener Mausklick über dem Grundriss der Wand verändert nun nach dem selben Muster die Höhe der Wand. Lässt der Nutzer die Maustaste ein weiteres Mal los, so stehen die engültigen Maße der Wand fest. Um beliebig viele

Wände zeichnen zu können, wird nun die Hilfsgeometrie gegen eine echte neue Geometrie ausgetauscht. Die Hilfsgeometrie mit ihrer PlaneSensor-Funktionalität wird zu späteren Verwendung durch eine Skalierung auf 0 0 0 "unsichtbar" gemacht.
Um neue Geometrien zur Laufzeit zu einer VRML-Szene hinzufügen zu können bietet der VRML-Standard zwei Verfahren an: Der erste, einfache Weg besteht darin mittels "`CreateVRMLfromString`" eine Zeichenkette mit einer gewöhnlichen VRML-Geometrie-Beschreibung zu übergeben. Für den vorgestellten Fall wird jedoch zusätzlich die eindeutige Identifizierbarkeit der neuen Geomtrie innerhalb der Szene sowie eine Verknüpung der in ihr enthaltenen Sensoren benötigt, die durch dieses Verfahren nicht gewährleistet werden kann. Das Mittel der Wahl ist hier eine Kombination von "CreateVRMLfromURL" und dem sog. Prototyping.
Mithilfe des Befehls "CreateVrmlFromURL" können beliebige .wrl-Dateien lokal oder via http zur Laufzeit zu einer vorhandenen Szene hinzugefügt werden. Auf diese Weise könnten beispielsweise aus einer großen Anzahl in seperaten Datein vorgehaltene Möbelstücke aus einem Katalog heraus in einen fertigen Rohbau gestellt werden. Ein Problem das bei der weiteren Arbeit mit diesen neuen Elementen in der Szene in dieser Grundvariante besteht ist, daß einzelne Objekte (Nodes) nachträglich nicht mehr identifizier- und damit modifizierbar sind, da VRML keinen direkten Zugriff auf dynamisch hinzugeladene Objekte vorsieht. Der Mechanismus, der zum Tragen kommt um dennoch weitere Operationen mit diesen neuen Bestandteilen vornehmen zu können läßt sich relativ einfach mit Prototyping und dynamischem routen bewerkstelligen. Ein Prototyp (PROTO-Node) ist die abstrakte Schablone eines Objektes dessen beliebig konfigurierbare Parameter beim Laden (der Instanzierung) bestimmt werden können. Auf diese Weise kann beispielsweise jeder Instanz ein eindeutiger Name / eine Eindeutige Nummer (ID) zugeordnet werden, über den/die später angesprochen werden kann. Die einfache Anwendung dieser Technik ist im vorliegenden Fall das erstellen eines Popupmenüs mit einem Menüeintrag zum Löschen der neu erzeugten Wand. Hierfür wird die Schablone einer Wand, also einer Box die die Außenmaße der im vorhergehenden Schritt verwendeten Hilfsgeometrie erhält, mit einem Eigenschaftsfeld "Außmaße" versehen und dieses gefüllt, sobald die neue Geometrie (Instanz) vollständig geladen ist. Zusätzlich ist auf jeder Wand ein TouchSensor angebracht, der signalisiert daß der Benutzer mit der Wand interagieren will. Der Befehl "`CreateVrmlFromURL`" ist zweigeteilt: Einmal in den Aufruf einer entsprechenden Datei und danach in der Behandlung des Ereignises "vollständig geladen" in einer sogenannten Callback-Funktion, die vom Autoren der Welt frei definiert werden kann. Die Callback-Funktion erhält als Parameter ein Feld aller Nodes, die aus der enstsprechenden Datei geladen wurde. Im Falle eines Prototypen können nun alle seine Eigenschaften, so z.B. das Namens-/ID-Feld nachträglich gefüllt werden.. So kann bspw. ein Menü an der Stelle eingeblendet werden, an der ein "`isOver`" des Mauszeigers/Stylus auftritt. Um dies zu realisieren muß der jeweilige TouchSensor seine zugehörige Wand an eine zentrale Stelle in der Hauptszene melden können. Hierfür wird ein EventOut vom Typ MFString (unbegrenzte Anzahl von Zeichenketten) in der Schablone (dem Prototypen) definiert. Als Reaktion auf das Ereignis "Zeigegerät über Wand" kann diese Zeichenkette dann mit dem Namen des Objektes gefüllt werden. Um diese Zeichenkette im Zentralskript behandeln zu können wird an dieser Stelle eine letzte Technik, das sog. dynamicRouting vorgestellt. Dabei werden, (wie beim oben beschriebenen "statische", d.h. einmaligen routen des Events) Ereignisse einer dyamisch hinzugeladenen Instanz mit einer bestehenden Funktion verknüpft, also im vorliegenden Fall z.B. durch "`Browser.addRoute([neueWand],'Nachrichtenevent', [Zentralskript], 'Behandlungsroutine');`" . Aus dem Zentralskript heraus kann dann jeder Bestandteil der Szene dahin überprüft werden, ob der in der Zeichenkette enthaltene Name mit dem des entsprechenden Objektes identisch ist. (Dies erscheint zunächst ein wenig umständlich läßt sich aber auf sehr viele Anwendungsfelder übertragen). Ist ersteinmal die eindeutige Zuorordnung hergestellt lassen sich beliebige Operationen ausführen.

Möglichkeitsraum

Wie das obige, einfache Beispiel zeigt, ist mit der leicht zu erlernenden Skriptsprache JavaScript eine sehr grosse Bandbreite möglicher Anwendungen realisierbar. Zu möglichen Anwendungen, die "im Prinzip" nur noch Fleißaufgaben darstellen zählen beispielsweise:

- Entwicklung eines Sets von Funktionen zur Manipulation mit den drei Grundtransformationen (Translation, Rotation, Skalierung)
- Abbildung aller in VRML möglichen Erscheinungs- und Materialparameter auf ein Menue
- Snaping und Alignment und Grids
- Set von Navigationsmethoden
- Aufbau einer Sammlung mit architektonischen Grundelementen (Stützen, Treppen, Träger etc.) aus denen Gestalter durch die o.a. Manipulationen Gebäude zusammenstellen kann (in einem späteren Schritt auch in eine netzbasierten Datenbank (z.B. SQL mit entsprechenden Serverseitigem Interface) einpflegbar)
- Einblendung von dynamischen Zusatzinformationen (Maße, k-Werte, Preise etc.)
- Kamerafahrtplaner für Renderanimationen durch Stylus.
- Landschaftsmodell-Modellierer

Beinahe alle der obenstehenden Erweiterungsmöglichkeiten können von Studenten selbständig erstellt werden. Vorstellbar ist z.B. ein zweistufiger VRML-Kurs, der erste davon wie bislang in den Modulen abgehalten und ein zusätzlicher, fortgeschrittener Kurs (Modul) mit Belegaufgaben aus dem oben stehenden Feld. Hierzu müßte ein gut durchdachtes Grundgerüst erstellt werden, Namenskonventionen festgeschrieben und Teilaufgaben definiert werden. Es könnte so über die Semester hinweg ein langsam anwachsender Funktionsumfang aus einzelnen Komponenten zusammengestellt werden. Der Anforderungsgrad ist in etwa auf Höhe des Mindstorm-Moduls anzusiedeln.

Erweiterungen

Von der Erweiterung des Grundprogrammes her, also dem Teil, der ausschließlich einem kleinen Kreis Programmierender vorbehalten ist wären folgende Erweiterungen denkbar:

- Verknüpfung / Anbindung an die Multiuserkomponenten von TAP
- Erweiterung der Graphikfähigkeit (z.B. Schatten für interaktiven Sonnenstands-Simulator in Echtzeit)
- Zusätzliche Implementierung von Ein-/und Ausgabegeräten (Stereo, Stereo-Projektion, Flock of Birds)
- Anbindung an Webray (selbe Codebasis): Mapping von Windows-Anwendungen auf Geometrien (z.B. Html-Browser in 3D, Photoshop als Zeichenwerkzeug für Texturen "unter der Mütze") etc.
- Plattform-Port ???
- Zusätzliche Schaffung einzelner Eingabegeräte-Nodes
- Anbindung an Spracheingabe

Der komplette, kommentiert Quelltext des Wandmodelierers mit der Funktionalität zum Erzeugen von Öffnungen ist unter http://www.uni-weimar.de/~beetz/jolanda/beispiele/wandgenerator/ zu finden

Jolanda

Technical Report

Jakob.Beetz@archit.uni-weimar.de Mai 2001
Professur InfAR Prof.Donath

Einleitung

Jolanda stellt den kompletten Leistungsumfang eines ausgewachsenen VRML97-Browsers für die Benutzung in einer virtual-reality-Umgebung zur Verfügung. Die derzeit implementierte Version ist dabei für die Verwendung mit HMD und Stylus und der Verwendung eines Polhemus Longrangers zugeschnitten. Die Implementierung weiterer Ein- und Ausgabegeräte ist grundsätzlich möglich und vorgesehen (Fastrack FOB, Stereo-Projektion).
Der entscheidende Vorteil des Systems gegenüber Speziallösungen (VRAM) liegt in der Trennung von Kernteilen des Systems wie Grafikausgabe, Eingabverwaltung, Netzwerkinterfaces usw. von der funktionalen und formalen Gestaltbarkeit der Interaktionsmöglichkeiten. Das enthaltene Scripting-Interface auf Basis von JavaScript bietet eine leicht erlernbare, flexible und dennoch mächtige Schnittstelle zur Realisierung unterschiedlichster Anwendungsmöglichkeiten, die selbst engagierten Laien die Modellierung und Durchführung komplexer interaktiver Anwendungen, z.B. Aus dem architektonischen Umfeld gestatten.
Im Folgenden werden nach einer kurzen Übersicht der zugrundeliegenden Software-Architektur anhand eines einfachen Beispiels (die Erstellung und Löschung von "Wänden") einige grundlegende Verfahren dargestellt, mit deren Hilfe komplexe Aufgaben in Virtuellen Umgebungen durch einfaches scripting auch für Nichtprogrammierer (z.B. Architekturstudenten) zu bewältigen sind. Den Abschluß bilden mögliche Anwendungsfelder in der Lehre / rapid Prototyping / der weiteren Entwicklung auf Grundlage der existierenden Software, sowie Ausblicke auf mögliche Erweiterungen des Basisprogramms durch Spezialisten.

Software-Architektur

Jolanda ist eine Erweiterung des im CommunitySource frei verfügbaren GlView (Holger Grahn, http://www.snafu.de/~hg). GlView bildet ebenfalls die Grundlage des kostenlosen Browser-Plugins "blaxxun Contact" sowie dessen Multiuser-Erweiterungen (Virtual Worlds, Virtual Community). Als Grafikkern kommen wahlweise OpenGL (Win32, Linux) oder Direct3D (Win32) zum Einsatz. Die interne Organisation des SceneGraphs ist eng an derjenigen von VRML97 angelehnt. Das Event-Modell ist ebenfalls eine direkte Abbildung der in der ISO-zertifizierten Spezifikation (ISO/IEC 14772-1:1997) festgeschriebenen Verwaltung von Ereignissen.

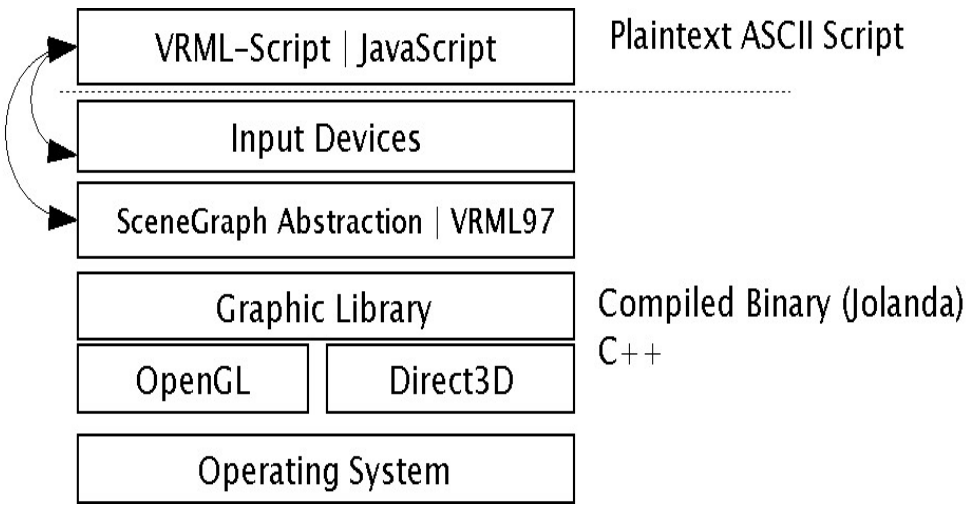


Abb. 1
Basisarchitektur von Jolanda

Die Erweiterung von Jolanda ist stellt im wesentlichen die Anbindung an ein Tracking-System und dessen Kopplung an Kamera-Steuerung und Eingabegerät sowie die Umsetzung der Sensoren (Touch-, Plane-, Cylinder-, Sphere-, Visibility-, Proximity-Sensor) im Raum dar. Die direkte Abbildung der Sensoren in einer echten räumlichen Umgebung hat zur Folge, das prinzipiell alle Interaktionen des Nutzers an einem einfachen, d.h. ungetrackten Arbeitsplatz erstellt und getestet werden können, da sich die Skripte und Sensoren an Bildschirm/Maus genauso verhalten wie mit HMD/Stylus. Dadurch ist es beispielsweise Architekturstudenten möglich komplexe Interaktionen zu Hause zu entwickeln und sie dann "unter der Mütze" einzusetzen. Technisch ist das identische Verhalten von Mauszeiger und Stylus durch die Verschiebung und Drehung des im 2D-Fall ortogonal zur Projektionsebene stehenden Strahles (zur Kollisionsüberprüfung mit Sensoren) entsprechend der Trackerkoordinaten gelöst.

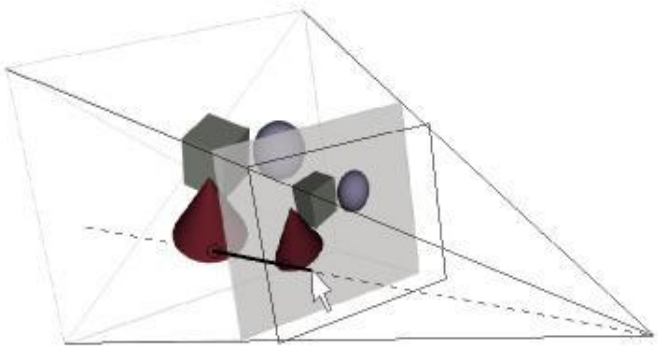


Abb. 2:
Screen-/Mousebasierter VRML-Client:
Viewpointachse enstpricht Selektions-/Kollisionsachse

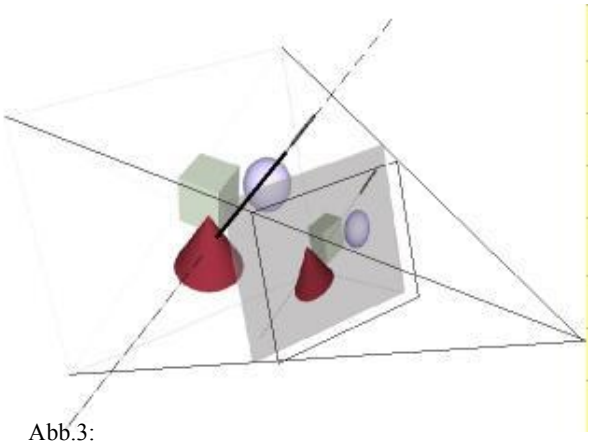


Abb.3:
Screenunabhängiger VRML-Client:
Viewpointachse unabhängig von Selektions-/Kollisionsachse

Die Positionierung von Kamera (HMD) und Kollisionsstrahl (Stylus) ist auf dem derzeitigen Stand der Implementierung "hartverdratet" (Class GvView, Renderloop) und wird über den Menüpunkt "`Options->enable fastrak`" aktiviert[1]. Eleganter und sauberer wäre an dieser Stelle die Einführung abstrackter TrackerNodes, die aus einem Skript heraus an den View oder die Kollisionsüberprüfung gebunden werden können ("binded Viewport" Konzept aus der VRML-Spec, dieses wäre auf die Eingabe noch zu übertragen).
Die Konfiguration des Trackers ist zum derzeitigen Stand ebenfalls nur durch einen neuen Compile zu verändern. Die voreingestellten Werte sind: COM2:, 38000 Baud, Longranger, Receiver 1 HMD, Receiver 2 Stylus. In zukünftigen Versionen sollte eine saubere Konfiguration der entsprechenden Parameter über eine Dialogbox vorgenommen werden können.

Known bugs & limitations:

- irgendwas merkwürdiges passiert mit dem Kamerablickwinkel bei bestimmten Bewegungnen mit dem Stylus (Magnetfeld?)
- Hängt sich bei nicht angeschlossenem Tracker auf, wenn "enable fastrak" aktiviert wird
- Einbindung der Stylusgeometrie unsauber
- Stürzt bei fehlender Stylusgeometrie ab
- autoOffset von Plansensor SPEC-konform in getrackter Variante?

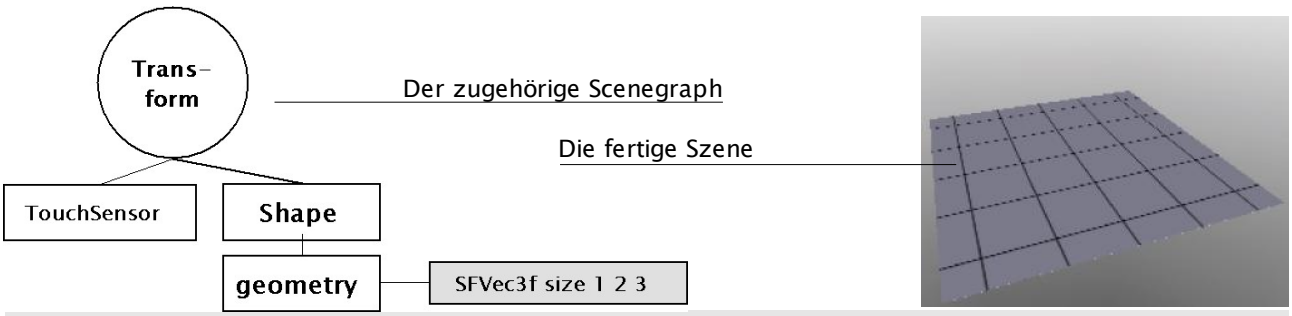
1 eine Datei "stylus.wrl" muß im selben Verzeichnis wie die .exe-Datei liegen. Die Welt(en) müssen vor der Aktivierung dieses Menüpunktes geladen sein. Mit "F4" kann anschliessend der fullscreen-Modus aktiviert werden. Der Menüpunkt "start tracker at startup" sollte nur aktiviert werden, wenn die entsprechenden Einstellungen des Trackers überprüft worden sind und die zu ladenden Dateien per Kommandozeile übergeben werden. Versehentliches aktivieren kann nur in der Registry wieder geändert werden.

Anwendungsbeispiel

Anhand des Zeichnens einer einfachen Geometrie (z.B. einer Wand) und der weiteren Interaktionsmöglichkeiten mit dieser (z.B. dem "Schneiden von Fenstern" oder die Löschung) soll im Folgenden die Einfachheit aber auch die Mächtigkeit der Verwendungsmöglichkeiten der Applikation veranschaulicht werden.
Zum Zeichnen einer einfachen Geometrie reicht zunächst ein einzeler .wrl-file: Die Grundlage bildet eine "Bodenplatte", also eine große flache Box, die mit einem PlaneSensor versehen wird. Dies kann mithilfe geeigneter Modellingprogramme (3DS Max, CosmoWorlds) vorgenommen werden. Drückt der Nutzer nun die Maus- oder Stylustaste wärend sich das Zeigegerät über der Bodenplatte befindet wird ein Feld des PlaneSensors (SFVec3f hitPoint) mit den lokalen Koordinaten des gedachten Schnittpunktes des Zeigegerätes und der Bodenplatte gefüllt. Um diese Koordinaten zum ersten Eckpunkt des Grundrisses der späteren Wand zu machen muß dieses Feld ausgelesen und gespeichert werden. Dies geschieht mittels des sogenannten Routings.

Routing

Geometrien, Materialien und andere (teilweise unsichbare) Bestandteile von VRML-Welten werden durch einzelne, hirarchisch verschachtelte Knoten (Nodes) beschrieben. Die Eigenschaften dieser Knoten werden in Feldern unterschiedlichster Datentypen festgelegt. Die Kantenlänge einer Box wird beispielsweise im Feld "size" festgelegt, das vom Typ SFVec3f ist, also drei Gleitkommawerte aufnehmen kann (Länge, Breite, Höhe. SingelFieldVector3floats).



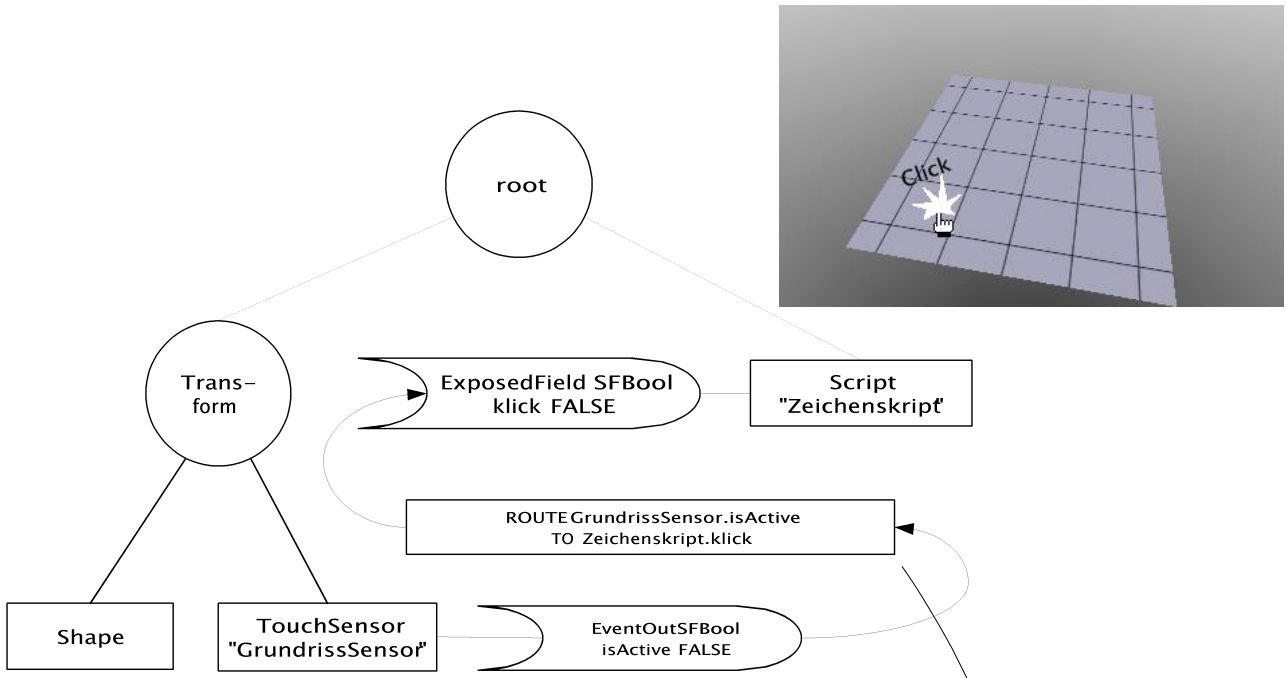
Zugehöriger Quelltext

```
DEF scene Transform
{
  children
  [
      DEF GrundrissSensor TouchSensor {},
      DEF Bodenplatte Transform
      {
      geometry Box
      {
                  size 100 1 100
            }
      }
  ]
}
```

Abb. 4:
einfache Szene zur Erstellung einer Interaktion

Die meisten dieser Felder können zur Laufzeit (also nach dem ersten Laden und der Betrachtung der Welt) verändert werden. Umgekehrt lösen veränderte Werte eines Feldes durch den Browser Nachrichten (Events) aus, die weitervarbeitet werden können. Routing bezeichnet nun das Verknüpfen der Veränderung eines Wertes mit der Veränderung eines anderen Wertes des gleichen Datentyps. In der Philosophie von VRML stellt ein Skript ebenfalls (wie die Geometrien) einen Knoten dar.

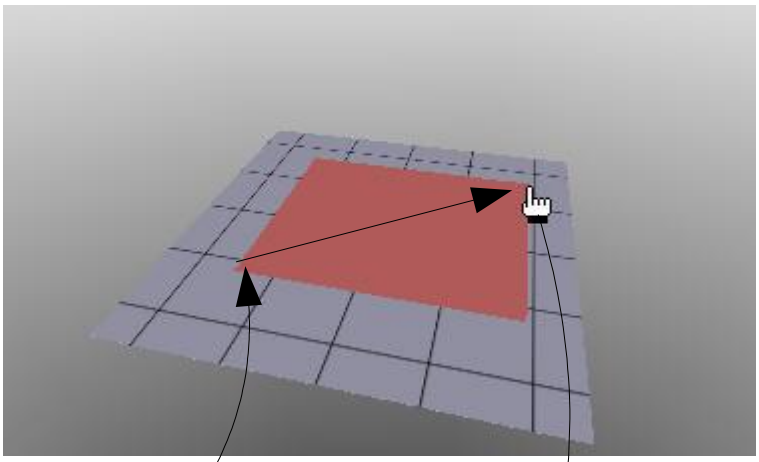
In diesem Fall wird das auftretende Ereignis (Benutzer drückt Knopf über dem PlaneSensor der Bodenplatte, die Veränderung des Wahrheitswertes SFBool [TouchSensor].isActive von FALSE nach TRUE) in eine Funktion eines Skriptes geleitet, das zusammen mit den Geometriedaten im .wrl-file der Welt steht.
Um das auftretende Ereignis auszuwerten wird zunächst ein SkriptNode erstellt, ein Eingangsfeld eventIn vom Wahrheitstyp SFBool als "Funktionskopf" erstellt und vermittels der Zeile ROUTE [TouchSensor].isActive TO [SkriptNode].[meineFunktion] mit der Behandlungsroutine verknüpft.




```
DEF temporaerWand Transform
{
     children Shape
     {
             geometry    Box{}
     }
}
DEF Zeichenskript
{
eventIn SFBool klick
url  [ "javascript:"
function klick (value, timestamp)
{
     active = value;          // Aktviert-Status speichern
     if (value == false)
     {
          scalenew[1] = 1;  // die Hilfsgeometrie sichbar machen
     }
}"]}
ROUTE GrundrissSensor.isActive TO Zeichnungsskript.klick
```

Abb. 5
Implementierung eines einfachen Sensors

Im Rumpf der Funktion wird nun die erste (lokale) Koordinate als Eckpunkt des Grundrisses der neuen Geometrie gespeichert und jede weitere Veränderung dieses Schnittpunktes ([TouchSensor].translation_changed) durch o.a. Verfahren in eine weitere Funktion des Skriptes geleitet. Solange der Knopf gedrückt gehalten wird liegen von nun an zunächst die beiden Eckpunkte der Grundrissfläche der neuen Wand vor. Eine vorher in die Szene eingefügte Hilfsgeometrie, hier eine Box, wird in der zweiten Funktion nun ständig auf diese Größe skaliert und in der Welt als visuelles Feedback gezeichnet (ROUTE [SkriptNode].[neueSkalierung] TO [Hilfsgeometrie].[neueSkalierung] und ROUTE [SkriptNode].[neuePosition] TO [Hilfsgeometrie].[neuePosition]).



```
function touchChange (value, time)
{
     if (active == false)
     {
        startvec = value;

    }
     else
     {
        // skalierung : erster Punkt - zweiter Punkt /2
        neueGroesse[0] = Math.abs((startvec[0] - value[0])/2);
        neueGroesse[2] = Math.abs((startvec[2] - value[2])/2);
        // Bewegung: Zentrum in erster Punkt - zweiter Punkt /2
        Zentrum [1] = 6;
        Zentrum[0] = (value[0] + startvec[0]) / 2;
        Zentrum[2] = (startvec[2] + value [2]) /2;
        endvec = value;
    }
}
ROUTE Zeichnungsskript.neueGroesse TO temporaerWand.scale
ROUTE Zeichnungsskript.Zentrum TO temporaerWand.translation
```

Abb. 6:
Skript zum Erstellen des Grundrisses der Wand

Die Hilfsgeometrie ist ihrerseits ebenfalls mit einem senkrecht angeordneten PlaneSensor versehen, der "scharfgeschaltet" wird, sobald der Benutzer durch loslassen der Maustaste den Grundriss der neuen Wand engültig festgelegt hat. Ein gehaltener Mausklick über dem Grundriss der Wand verändert nun nach dem selben Muster die Höhe der Wand. Lässt der Nutzer die Maustaste ein weiteres Mal los, so stehen die engültigen Maße der Wand fest. Um beliebig viele

Wände zeichnen zu können, wird nun die Hilfsgeometrie gegen eine echte neue Geometrie ausgetauscht. Die Hilfsgeometrie mit ihrer PlaneSensor-Funktionalität wird zu späteren Verwendung durch eine Skalierung auf 0 0 0 "unsichtbar" gemacht.
Um neue Geometrien zur Laufzeit zu einer VRML-Szene hinzufügen zu können bietet der VRML-Standard zwei Verfahren an: Der erste, einfache Weg besteht darin mittels "`CreateVRMLfromString`" eine Zeichenkette mit einer gewöhnlichen VRML-Geometrie-Beschreibung zu übergeben. Für den vorgestellten Fall wird jedoch zusätzlich die eindeutige Identifizierbarkeit der neuen Geomtrie innerhalb der Szene sowie eine Verknüpung der in ihr enthaltenen Sensoren benötigt, die durch dieses Verfahren nicht gewährleistet werden kann. Das Mittel der Wahl ist hier eine Kombination von "CreateVRMLfromURL" und dem sog. Prototyping.
Mithilfe des Befehls "CreateVrmlFromURL" können beliebige .wrl-Dateien lokal oder via http zur Laufzeit zu einer vorhandenen Szene hinzugefügt werden. Auf diese Weise könnten beispielsweise aus einer großen Anzahl in seperaten Datein vorgehaltene Möbelstücke aus einem Katalog heraus in einen fertigen Rohbau gestellt werden. Ein Problem das bei der weiteren Arbeit mit diesen neuen Elementen in der Szene in dieser Grundvariante besteht ist, daß einzelne Objekte (Nodes) nachträglich nicht mehr identifizier- und damit modifizierbar sind, da VRML keinen direkten Zugriff auf dynamisch hinzugeladene Objekte vorsieht. Der Mechanismus, der zum Tragen kommt um dennoch weitere Operationen mit diesen neuen Bestandteilen vornehmen zu können läßt sich relativ einfach mit Prototyping und dynamischem routen bewerkstelligen. Ein Prototyp (PROTO-Node) ist die abstrakte Schablone eines Objektes dessen beliebig konfigurierbare Parameter beim Laden (der Instanzierung) bestimmt werden können. Auf diese Weise kann beispielsweise jeder Instanz ein eindeutiger Name / eine Eindeutige Nummer (ID) zugeordnet werden, über den/die später angesprochen werden kann. Die einfache Anwendung dieser Technik ist im vorliegenden Fall das erstellen eines Popupmenüs mit einem Menüeintrag zum Löschen der neu erzeugten Wand. Hierfür wird die Schablone einer Wand, also einer Box die die Außenmaße der im vorhergehenden Schritt verwendeten Hilfsgeometrie erhält, mit einem Eigenschaftsfeld "Außmaße" versehen und dieses gefüllt, sobald die neue Geometrie (Instanz) vollständig geladen ist. Zusätzlich ist auf jeder Wand ein TouchSensor angebracht, der signalisiert daß der Benutzer mit der Wand interagieren will. Der Befehl "`CreateVrmlFromURL`" ist zweigeteilt: Einmal in den Aufruf einer entsprechenden Datei und danach in der Behandlung des Ereignises "vollständig geladen" in einer sogenannten Callback-Funktion, die vom Autoren der Welt frei definiert werden kann. Die Callback-Funktion erhält als Parameter ein Feld aller Nodes, die aus der enstsprechenden Datei geladen wurde. Im Falle eines Prototypen können nun alle seine Eigenschaften, so z.B. das Namens-/ID-Feld nachträglich gefüllt werden.. So kann bspw. ein Menü an der Stelle eingeblendet werden, an der ein "`isOver`" des Mauszeigers/Stylus auftritt. Um dies zu realisieren muß der jeweilige TouchSensor seine zugehörige Wand an eine zentrale Stelle in der Hauptszene melden können. Hierfür wird ein EventOut vom Typ MFString (unbegrenzte Anzahl von Zeichenketten) in der Schablone (dem Prototypen) definiert. Als Reaktion auf das Ereignis "Zeigegerät über Wand" kann diese Zeichenkette dann mit dem Namen des Objektes gefüllt werden. Um diese Zeichenkette im Zentralskript behandeln zu können wird an dieser Stelle eine letzte Technik, das sog. dynamicRouting vorgestellt. Dabei werden, (wie beim oben beschriebenen "statische", d.h. einmaligen routen des Events) Ereignisse einer dyamisch hinzugeladenen Instanz mit einer bestehenden Funktion verknüpft, also im vorliegenden Fall z.B. durch "`Browser.addRoute([neueWand],'Nachrichtenevent', [Zentralskript], 'Behandlungsroutine');`" . Aus dem Zentralskript heraus kann dann jeder Bestandteil der Szene dahin überprüft werden, ob der in der Zeichenkette enthaltene Name mit dem des entsprechenden Objektes identisch ist. (Dies erscheint zunächst ein wenig umständlich läßt sich aber auf sehr viele Anwendungsfelder übertragen). Ist ersteinmal die eindeutige Zuorordnung hergestellt lassen sich beliebige Operationen ausführen.

Möglichkeitsraum

Wie das obige, einfache Beispiel zeigt, ist mit der leicht zu erlernenden Skriptsprache JavaScript eine sehr grosse Bandbreite möglicher Anwendungen realisierbar. Zu möglichen Anwendungen, die "im Prinzip" nur noch Fleißaufgaben darstellen zählen beispielsweise:

- Entwicklung eines Sets von Funktionen zur Manipulation mit den drei Grundtransformationen (Translation, Rotation, Skalierung)
- Abbildung aller in VRML möglichen Erscheinungs- und Materialparameter auf ein Menue
- Snaping und Alignment und Grids
- Set von Navigationsmethoden
- Aufbau einer Sammlung mit architektonischen Grundelementen (Stützen, Treppen, Träger etc.) aus denen Gestalter durch die o.a. Manipulationen Gebäude zusammenstellen kann (in einem späteren Schritt auch in eine netzbasierten Datenbank (z.B. SQL mit entsprechenden Serverseitigem Interface) einpflegbar)
- Einblendung von dynamischen Zusatzinformationen (Maße, k-Werte, Preise etc.)
- Kamerafahrtplaner für Renderanimationen durch Stylus.
- Landschaftsmodell-Modellierer

Beinahe alle der obenstehenden Erweiterungsmöglichkeiten können von Studenten selbständig erstellt werden. Vorstellbar ist z.B. ein zweistufiger VRML-Kurs, der erste davon wie bislang in den Modulen abgehalten und ein zusätzlicher, fortgeschrittener Kurs (Modul) mit Belegaufgaben aus dem oben stehenden Feld. Hierzu müßte ein gut durchdachtes Grundgerüst erstellt werden, Namenskonventionen festgeschrieben und Teilaufgaben definiert werden. Es könnte so über die Semester hinweg ein langsam anwachsender Funktionsumfang aus einzelnen Komponenten zusammengestellt werden. Der Anforderungsgrad ist in etwa auf Höhe des Mindstorm-Moduls anzusiedeln.

Erweiterungen

Von der Erweiterung des Grundprogrammes her, also dem Teil, der ausschließlich einem kleinen Kreis Programmierender vorbehalten ist wären folgende Erweiterungen denkbar:

- Verknüpfung / Anbindung an die Multiuserkomponenten von TAP
- Erweiterung der Graphikfähigkeit (z.B. Schatten für interaktiven Sonnenstands-Simulator in Echtzeit)
- Zusätzliche Implementierung von Ein-/und Ausgabegeräten (Stereo, Stereo-Projektion, Flock of Birds)
- Anbindung an Webray (selbe Codebasis): Mapping von Windows-Anwendungen auf Geometrien (z.B. Html-Browser in 3D, Photoshop als Zeichenwerkzeug für Texturen "unter der Mütze") etc.
- Plattform-Port ???
- Zusätzliche Schaffung einzelner Eingabegeräte-Nodes
- Anbindung an Spracheingabe

Der komplette, kommentiert Quelltext des Wandmodelierers mit der Funktionalität zum Erzeugen von Öffnungen ist unter http://www.uni-weimar.de/~beetz/jolanda/beispiele/wandgenerator/ zu finden