# Kapitel 2

# Stand der Wissenschaft und Technik

Seit dem Ende der 80er Jahre steht die Gerätetechnologie und Grafikleistung zur Verfügung, auf deren Grundlage VR-Systeme realisiert werden können. Das erste System seiner Art, RB2 (Reality Built for 2), wurde von Blanchard et al. [BBHL90] 1990 vorgestellt. In der folgenden Dekade wurde eine ständig wachsende Zahl von VR-Systemen entwickelt. Bis heute ist die Virtuelle Realität ein offenes Feld der Forschung und daher konnten sich bisher keine verbindlichen Standards etablieren. Praktisch jede Gruppe, die auf diesem Gebiet forscht, entwickelt an ihrem eigenen VR-System. Die Systeme DIVE [CaHa93, Hags96], AVIARY [Snow94], dVS [Ghee95] und Avocado [Tram99] fokussieren auf die Entwicklung von verteilten Multi-User VR-Systemen. Sie bieten die üblichen Softwarekomponenten zur Handhabung von Interaktionsgeräten, geometrischen Objekten und zu deren grafischer Darstellung. Darauf basierend werden Anwendungen in C/C++ programmiert, die Gerätedaten verarbeiten und geometrische Objekte manipulieren. Das Avocado Framework bietet zusätzlich Scheme als Skriptsprache an.

In diesem Kapitel werden zunächst eine Auswahl von Systemen im Hinblick auf ihre Eignung für die Modellierung von dynamischem Objektverhalten und die Interaktion mit multidimensionalen Interaktionsgeräten untersucht. Es finden dabei sowohl Systeme aus der Forschung als auch kommerzielle Systeme Beachtung.

# 2.1 VR-Systeme

## 2.1.1 Simple Virtual Environment Library

Die Simple Virtual Environment Library (SVE) stammt aus dem Graphics, Visualization and Usability Center am Georgia Institute of Technology und wird dort gemeinsam mit der Leigh Universität in Pennsylvania weiterentwickelt, aktuell ist die Version 2.1 [KeBH00, SVE02].

SVE bildet ein Rahmensystem bestehend aus einer hierarchischen Objektdatenbank, einem grafischen Renderer und einer Bibliothek zur Anbindung von Interaktionsgeräten. Anwendungen bestehen aus Funktionen, die über einen Callback-Mechanismus in SVE integriert werden. Abbildung 2.1 gibt einen Überblick über den Aufbau des Systems.

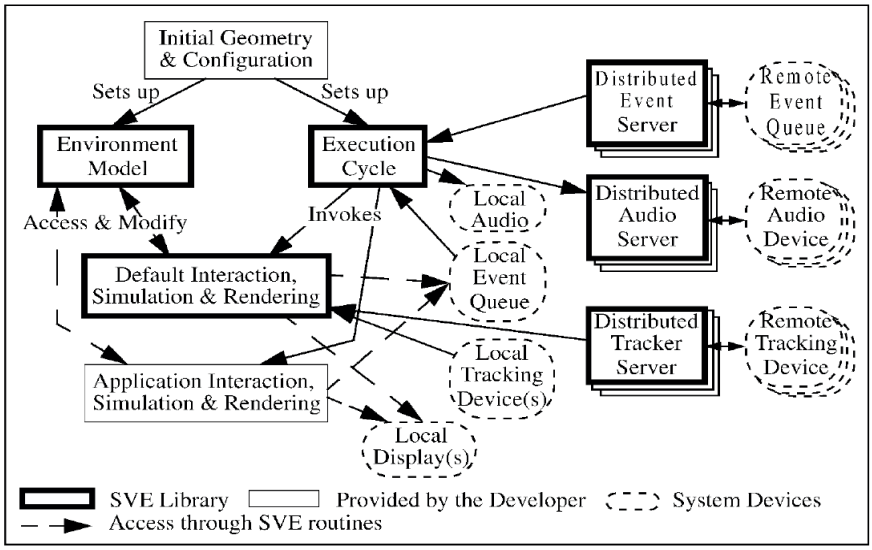


Abbildung 2.1: Architektur des SVE Frameworks [KeBH00]

SVE unterstützt eine Reihe von Trackingsystemen u.a. Ascension Bird, Polhemus Fastrak und den integrierten Tracker der Virtual I/O Glasses. Weiter kann der Cyberglove von Virtual Technologies sowie die lokale Tastatur und Maus verwendet werden.

SVE erlaubt den Zugriff auf Interaktionsgeräte, die an externe Rechner angeschlossen sind. Die Übertragung der Gerätedaten erfolgt über das verbindende Netzwerk.

Das Framework bietet zwei Möglichkeiten, Interaktionsgeräte anzubinden: Geräte können Objekten im Szenengraphen zugeordnet (*attached*) wer-

den, oder sie werden vom Programmcode der Applikation selbst abgefragt (*polled*).

Die erste Methode nutzt elegant die Szenenhierarchie um die Koordinatentransformationen abzubilden, die nötig sind um die Gerätedaten in das Koordinatensystem der Szene zu transformieren.

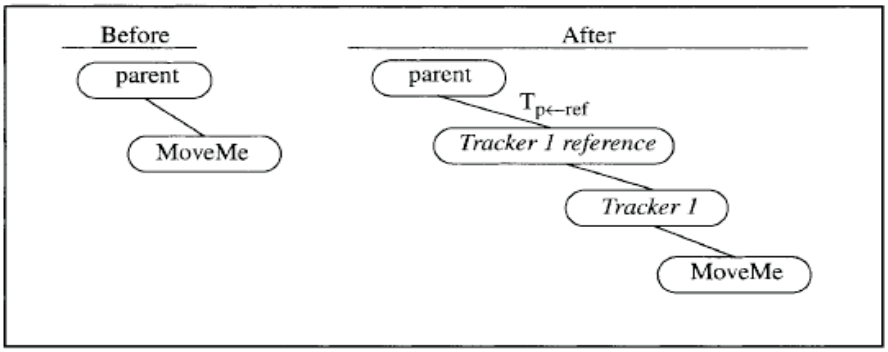


Abbildung 2.2: Abbildung eines Trackers in die SVE Szene [KeBH00]

Abbildung 2.2 illustriert das Verfahren. Die linke Seite stellt die Ausgangssituation dar. Ein grafisches Objekt (*MoveMe*) befindet sich als Kind des Knotens *Parent* im Szenengraphen. Wird *MoveMe* nun ein Tracker zugeordnet, werden zwei neue Transformationsknoten eingefügt. Der untere Knoten, *Tracker 1*, transformiert sein Kind *MoveMe* anhand der Gerätedaten des zugeordneten Trackingsystems. Die Transformation der Gerätedaten in das Welt-Koordinatensystem der Szene wird durch den statischen Transformationsknoten *Tracker 1 Reference* abgebildet.

Werden die Geräte von der Applikation selbst verwaltet, finden dazu Funktionen der Tracker Library von SVE Verwendung. Wie der Name bereits andeutet, unterstützt diese lediglich Trackingsysteme, die hardwareunabhängig angesprochen werden können. Zusätzlich können die Tasten des Polhemus Stylus oder die Gestenerkennung für den Cyberglove abgefragt werden. Selbstverständlich ist auch die Abfrage von Tastatur und Maus möglich. Die Handhabung von Tracker, Tasten, Gesten und Tastatur/Maus erfolgt jeweils über einen eignen Satz von C-Funktionen, ein einheitliches logisches Gerätekonzept existiert nicht.

SVE Anwendungen bestehen aus einem Satz von Callback Funktionen, die in den Kontrollfluß des Systems eingefügt werden können. Wie im Falle der Interaktionsgeräte wird deutlich, daß SVE um den grafischen Renderer herum konstruiert wurde.

Die Hauptschleife durchläuft zwei Phasen, in der ersten werden die Da-

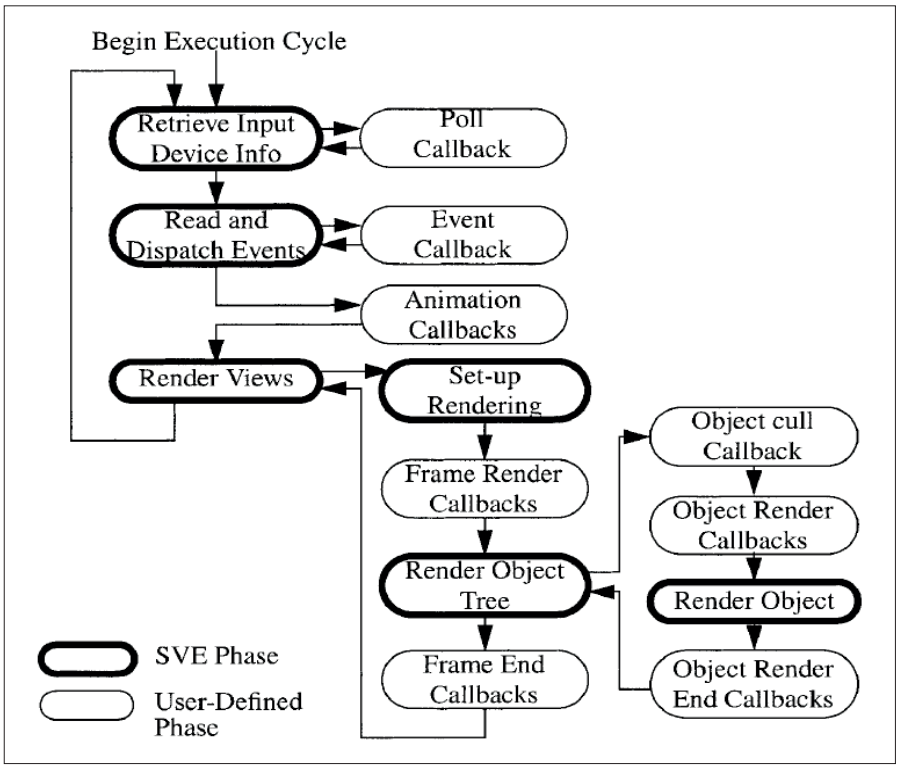


Abbildung 2.3: Kontrollfluß der Simulationsschleife von SVE [KeBH00]

ten der Interaktionsgeräte abgefragt, anschließend die grafische Darstellung durchgeführt. In beiden Phasen können an verschiedenen Stellen eigene Callback Funktionen eingefügt werden, um SVE um applikationsspezifisches Verhalten zu erweitern. Das Callback-Konzept von SVE hat einen gravierenden Nachteil: Die grafische Darstellung kann nicht von der Simulation getrennt werden, beides läuft streng sequentiell in einem einzigen Prozeß ab. Damit lässt sich die Leistung moderner Multi-Prozessor Hardware nicht nutzen.

Die Abstraktionsebene, auf der Anwendungen mit SVE entwickelt werden, niedrig. Vorteilhaft ist, daß die Anwendung praktisch überall in den Ablauf der Hauptschleife eingreifen kann. Allerdings wird die Beschreibung von Objektverhalten auf höherem Niveau nicht unterstützt, sodas die Entwicklung von komplexen dynamischen Systemen sehr aufwendig ist.

### 2.1.2 Minimal Reality

Das Minimal Reality Toolkit (MR) wird seit 1991 an der Universität von Alberta, Canada als Software-Toolkit für Forschung und Entwicklung im Bereich der Virtuellen Realität entwickelt [SLGS92, SGLS93, GrWh95]. Später

wurde es durch MRObjects, ein objektorientiertes Softwarepaket ergänzt [Gree01]. MR und MRObjects werden bis heute im VR-Labor "VizRoom" der Universität von Alberta als Grundlage für die Entwicklung von VR-Anwendungen verwendet. MR besitzt Interfaces zu den Programmiersprachen C und Fortran und ist unter UNIX auf SGI, IBM RISC System/6000 und DEC 5000 lauffähig.

MR besteht konzeptionell aus vier Komponenten, *Interaction* ist für die Benutzerinteraktion zuständig, *Computation* für die Simulation, *Geometric Model* enthält die Szenenbeschreibung und *Presentation* sorgt für die grafische, akustische und haptische Rückkopplung zum Benutzer. Die Komponenten können auf mehrere Prozesse verteilt werden. Die Ansteuerung der Interaktionsgeräte erfolgt über Serverprozesse, an die sich MR-Anwendungen dynamisch an- und abmelden können.

Der Funktionsumfang von MR beschränkt sich auf die Ansteuerung von Interaktionsgeräten, Interprozeß-Kommunikation, 2D-Interaktion und einigen weiteren Paketen. Funktionen zum Rendering eines Szenengraphen sind nicht enthalten, MR-Anwendungen basieren direkt auf OpenGL.

Diese Lücke füllt MRObjects. MRObjects bietet ein objektorientiertes C++ Interface für VR-Anwendungen und basiert auf MR und OpenGL. Sein Funktionsumfang umfaßt die grafische Darstellung, eine logische Geräteschnittstelle (siehe dazu auch Abschnitt 5.3) sowie fundamentale Interaktionstechniken.

Für die grafische Darstellung einer Szene bietet MRObjects eine Klassenbibliothek, die Klassen für Transformationen, geometrische Primitive, polygonale Objekte, Material und Beleuchtung beinhaltet. Aus diesen kann ein hierarchischer Szenengraph konstruiert werden.

Die Verwendung von Interaktionsgeräten erfolgt hardware-unabhängig über Klassen von logischen Geräten. Die Abbildung von logischen auf physische Geräte wird durch eine externe Konfigurationsdatei definiert.

Fundamentale Interaktionstechniken unterstützt MRObjects durch eine weitere Klassenbibliothek. Diese umfassen die Steuerung von Cursoren durch multidimensionale Interaktionsgeräte sowie Techniken zur Selektion von Objekten über Menüs und Cursoren.

### 2.1.3 Alice

Alice als interaktive Programmierumgebung für Grafische Anwendungen an der University of Virginia entstanden und wird seit einigen Jahren an der Carnegie Mellon University von der Stage 3 Research Group weiterentwickelt [Conw00, PaBu95].

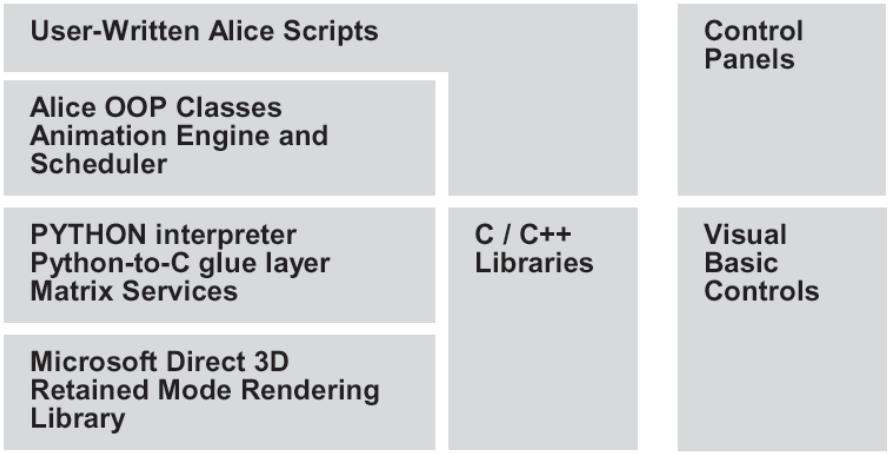


Abbildung 2.4: Alice Software Architektur[Conw00]

Wie Abbildung 2.4 darstellt, basiert Alice auf einem Microsofts Direct3D Renderer und der objektorientierten interpretierten Programmiersprache Python [Beaz01]. Python organisiert die Szenenbeschreibung, wie die anderen untersuchten Systeme in einem hierarchischen Szenengraphen. Im Unterschied zu diesen gibt es in Alice aber keinen Unterschied zwischen den Knoten des Szenengraphen und anderen Objekten der Programmiersprache. Kinder eines Gruppenknotens in der Szenenhierarchie werden auf die gleiche Weise angesprochen wie aggregierte Objekte. Die folgende Anweisung dreht den Kopf eines Hasen (*bunny*) geradeaus.

```
Bunny.Head.Turn(Forward)
```

Anweisungen können sequentiell und parallel ausgeführt werden. Da in Python Funktionen ebenfalls Objekte sind, ergeben sich in Verbindung mit Listen ein mächtige Ausdrucksmöglichkeiten um Verhalten zu beschreiben. Ein Beispiel aus [Conw00]:

```
ArmsOut = DoTogether(Bunny.Body.LeftArm.Turn(Left, 1/8),
                     Bunny.Body.RightArm.Turn(Right,1/8) )
ArmsIn = DoTogether(Bunny.Body.LeftArm.Turn(Right, 1/8),
                     Bunny.Body.RightArm.Turn(Left,1/8) )
BangTheDrumSlowly = DoInOrder(
                     ArmsOut, ArmsIn, Bunny.PlaySound('bang'))
BangTheDrumSlowly.Loop()
```

Aktuelle Versionen von Alice unterstützen offenbar keine multidimensionalen Interaktionsgeräte mehr, während frühere Versionen diese Fähigkeit

noch besaßen [PaBu95]. Der Grund liegt vermutlich in der Entwicklung von Alice zu rein PC-basierten System für den "Hausgebrauch" und ein Laienpublikum. Allerdings besteht die Möglichkeit, eine Reihe von solchen Geräten über die Microsoft-Schnittstelle DirectInput in Anwendungen zu integrieren (siehe dazu auch Abschnitt 2.2.1).

### 2.1.4 Sense8 World Tool Kit

Bei World Toolkit (WTK) handelt es sich um eine Bibliothek von C-Funktionen zur Entwicklung von virtuellen Umgebungen. Es umfaßt die u.A. Bereiche Rendering, Handhabung von Interaktionsgeräten und den Import von Geometriemodellen [WTK94, WTK99]. WTK ist als kommerzielles Produkt von der Firma Sense8, Mill Valley, Kalifornien in der Version 9 verfügbar und unter Microsoft Windows und UNIX lauffähig.

WTK verwaltet die grafische Szenenbeschreibung in Form von hierarchischen Szenengraphen. Mehrere Szenengraphen können gleichzeitig existieren. Attribute von Knoten des Szenengraphen können von der Applikation direkt gelesen und geschrieben werden. Darüber hinaus steht seit Version 8 ein Zugriffsmechanismus über sogenannte *Properties* zur Verfügung. Für jede Property und jedem Knoten können Callback Funktionen angemeldet werden, die aufgerufen werden, sobald sich der Wert der Property ändert[1]. Neben den vom System vorgegebenen Satz von Properties ist es auch möglich, Knoten selbst definierte Properties zu assoziieren und so den Szenengraphen mit Applikationsdaten zu anzureichern. Wie Properties, können jedem Knoten auch sogenannte *action* Funktionen zugeordnet werden. Die Reihenfolge ihres Aufrufs kann über die Vergabe Prioritäten von der Anwendung kontrolliert werden.

Logische Geräte werden im WTK-Sprachgebrauch Sensoren genannt. Das Konzept der Sensoren von WTK umfasst ausschließlich logische Geräte, die Position und Orientierungen von grafischen Objekten steuern. Sensoren können Interaktionsgeräten beliebig zugeordnet werden. Für Geräte mit weniger als 6 Freiheitsgraden, wie der Maus werden Metaphern zur Interaktion im dreidimensionalen Raum geboten. Etwa kann ein Maus-Sensor eine Rotation liefern, wenn der Mauszeiger den Bildschirmrand berührt oder wenn er von der Bildschimmitte entfernt wird. Die Abbildung von logischen auf physische Geräte erfolgt durch die Applikation. Eine Konfiguration durch eine externe

[1]Dieser Mechanismus funktioniert natürlich nur dann, wenn zum Ändern des Wertes Methoden des Property Objektes benutzt werden. Wird der Wert direkt im Knoten geändert, was nach wie vor möglich ist, bleibt die Änderung unerkannt.

Datei ist in WTK nicht vorgesehen, kann aber durch die Applikation realisiert werden.

Sensoren können darüberhinaus Knoten im Szenengraphen zugeordnet werden, sodas Wertänderungen automatisch in den Szenengraphen propagiert werden können. Das Konzept der Sensoren ist auf 6D-Positionsgeber beschränkt. Um beispielsweise auf die Tasten des Spaceballs zugreifen zu können, muß eine geräteabhängige Datenstruktur ausgewertet werden. Dadurch geht der Vorteil des Konzepts logischer Geräte, nämlich die Hardwareunabhängige Verwendung von Interaktionsgeräten teilweise verloren.

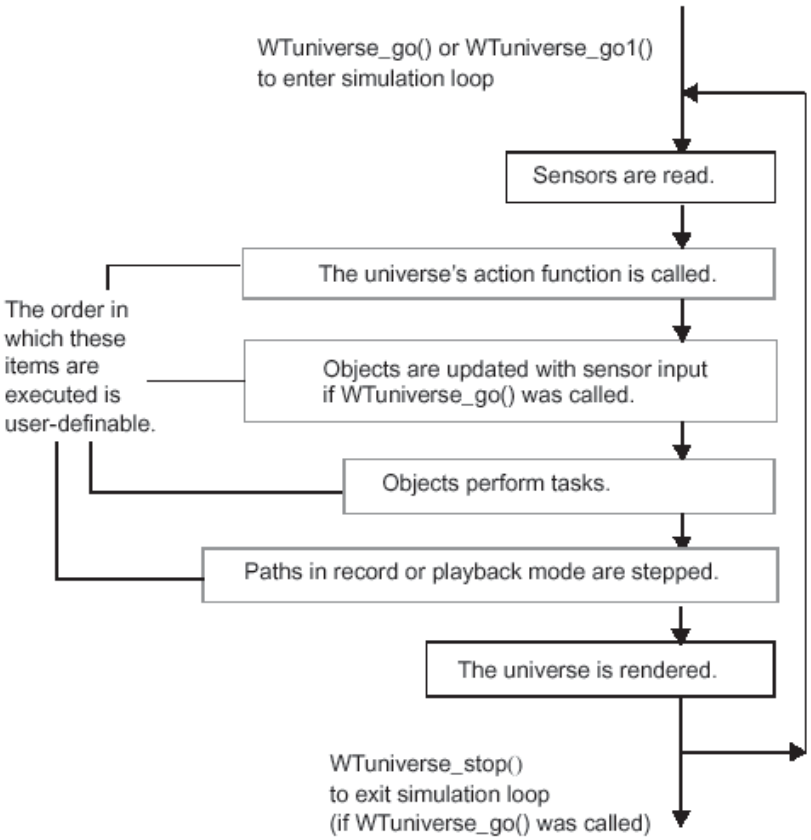


Abbildung 2.5: Simulationsschleife von WTK [WTK99]

Den Ablauf der Simulationsschleife von WTK zeigt Abbildung 2.5 noch einmal im Überblick.

WTK wird seit 1997 durch das WorldUp ergänzt, welches vom Hersteller als Werkzeug für Autoren von virtuellen Umgebungen vermarktet wird. Ne-

ben einer grafischen Benutzungsschnittstelle bietet WorldUp Scripting mit einer Basic-artigen Programmiersprache, einen Modellierer sowie zahlreiche Werkzeuge für die Konfiguration des Systems und der Interaktionsgeräte.

Zusätzlich führt WorldUp eine Klasse von Objekten ein, die Trigger (*trigger*) genannt werden. Trigger haben bis zu drei Eingänge und können über ihre Ausgänge mit weiteren Triggern oder *actions* (siehe oben) verdrahtet werden. Solche Systeme können als *behaviors* gekapselt und in einer Bibliothek verwaltet werden. Abbildung 2.6 gibt ein Beispiel.

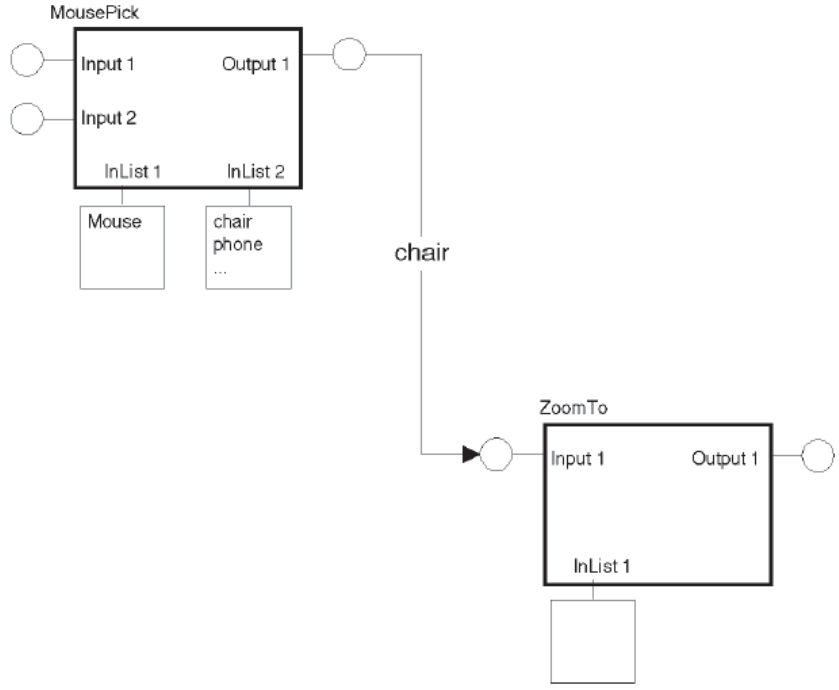


Abbildung 2.6: WorldUp Trigger und Action [WUP00]

Ein Trigger `MousePick` wird mit einer Action `ZoomTo` so verbunden, daß sobald ein Objekt gepickt wird, die Kamera darauf zu schwenkt. Dazu enthält der Trigger eine Liste aller selektierbaren Objekte und sendet, sobald eines dieser Objekte mit der Maus selektiert wurde, dieses an die Aktion welche die Kamera steuert.

### 2.1.5 VRML-basierte Systeme

VRML hat sich in den letzten Jahren zu einem Standard für den Austausch von dreidimensionalen Modellen und zur Darstellung von Verhalten in virtu-

ellen Umgebungen entwickelt [Iso97]. Insbesondere Internet-orientierte Anwendungen verwenden VRML. VRML ist inzwischen weit verbreitet und hat sich als ein Standardformat zum Austausch von dreidimensionalen grafischen Modellen etabliert. Da die logischen Konzepte von VRML inzwischen allgemein geläufig sind, werden diese hier nicht ausführlich wiederholt, sondern die lediglich die Schlüsselkonzepte diskutiert.

VRML beschreibt Geometrie der Szene durch einen hierarchischen Szenengraphen. Zusätzlich zu den Knotenklassen, die zur grafischen Darstellung der Szene und zum Aufbau des Szenengraphen benötigt werden, existieren um vier weitere Klassen von Knoten: Sensor-Knoten (*sensor*) für die Benutzerinteraktion [2]), Interpolator-Knoten für Animationen (*interpolators*), Inline-Knoten *inline* für Hyperlinks auf andere Szenen und Knoten die Skripte enthalten (*script node*).

Jeder VRML-Knoten besitzt Felder (*fields*) die seine Attribute beschreiben. Felder können über gerichtete Routen (*routes*) miteinander verbunden werden. Ändert sich der Wert es Feldes am Anfang der Route wird der Wert vor dem nächsten Zyklus in das Feld am Ende der Route propagiert. Als Quellen für diese Änderungen kommen Sensoren und Skriptknoten in Frage.

Routen werden typischerweise innerhalb der Szene definiert. Zur Laufzeit können aber durch Skriptknoten neue Routen erzeugt oder gelöscht werden.

Teile von Szenen können über Prototypen-Knoten (*prototypes*) zusammengefasst und mit einer eigenen Schnittstelle aus Feldern versehen werden.

Verhalten, welches über die Fähigkeiten der eingebauten Sensor- und Interpolator-Knoten hinausgeht wird über Skriptknoten realisiert. Diese können in JAVA, JAVA Script oder VRML Script verfasst werden. Ebenso wie Prototypen können Skriptknoten beliebige Interfaces aus Feldern besitzen.

## 2.2 Systeme zur Anbindung von Interaktionsgeräten

### 2.2.1 Microsoft DirectInput

DirectInput [DiIn99] stellt eine Komponente von DirectX dar, Microsofts Multimedia Betriebsystemerweiterung für MS-Windows. Aufgabe von DirectInput ist die hardwareunabhängige Ansteuerung von Spielecontrollern wie Joysticks, Lenkrädern und Pedalen für Fahrsimulationen, aber auch Tastatur

[2] Zur Klasse der Sensoren werden hier auch Zeitgeber gezählt.

und Maus. Relevant für diese Arbeit ist DirectInput aus mehreren Gründen. Erstens unterstützt als einziges der untersuchten Systeme auch haptische Ausgabegeräte. Zweitens kommen wesentliche Impulse für die Computergrafik im Allgemeinen und die Virtuelle Realität im Besonderen aus der Unterhaltungsbranche. Schließlich ist DirectInput als Systemkomponente von Microsoft Windows das mit Abstand am weitesten verbreitete System dieser Art überhaupt. DirectInput unterstützt die folgenden Geräte:

- Tastatur und Maus
- Spielecontroller aller Art
- Force Feedback Geräte

Neben den aktuell unterstützten Spielgeräten würde DirectInput aber auch die Anbindung der in Abschnitt 5.2 genannten Geräte erlauben. DirectInput umgeht aus Gründen der Effizienz das Message System von Windows und greift direkt über die Gerätetreiber auf die Gerätehardware zu.

### Initialisierung eines Gerätes

Eine Anwendung, die an den Daten eines bestimmten Gerätes interessiert ist, muß sich zunächst über eine COM-Schnittstelle eine Referenz auf ein DirectInput Device Object verschaffen. Dazu wird ein sogenannter Enumeration-Vorgang durchgeführt, der für die gewünschte Gerätekategorie parametriert werden kann. Referenzen auf Device Objects für Tastatur und Maus stehen standardmäßig zur Verfügung.

Im nächsten Schritt wird die Verfügbarkeit des Gerätes und dessen spezifische Eigenschaften ermittelt. Etwa kann festgestellt werden, wieviele Tasten ein Joystick besitzt, oder ob an einer Maus ein zusätzliches Stellrad zur Verfügung steht. Auf jede einzelne Eigenschaft kann später durch eine sogenannte Object Instance zugegriffen werden. Danach werden verschiedene Einstellungen für das Gerät vorgenommen. Zunächst wird definiert, ob das Gerät exklusiv von der Applikation genutzt werden soll, oder ob andere Anwendungen darauf ebenfalls zugreifen dürfen. Für Force-Feedback Geräte ist der exklusive Zugriff aber vorgeschrieben. Anschließend wird das für ein Gerät zu verwendende Datenformat bestimmt. Es wird, bestimmt welche Device Instance Daten liefern soll, ob diese absolut oder relativ sind [3] Für das

[3]Microsoft benutzt in [DiIn99] für die Begriffe "absolut" und "relativ" eine andere Bedeutung als die in dieser Arbeit verwendete. Bei DirectInput meint "absolut" den aktuellen Wert einer Object Instance (etwa die Auslenkung eines Joysticks in X-Richtung), während

Gerät kann weiter bestimmt werden, ob die Daten gepuffert oder ungepuffert geliefert werden, und wie groß dieser Puffer sein soll. Damit ist die Initialisierungssequenz abgeschlossen und es können Daten vom Gerät gelesen werden.

#### Lesen von Gerätedaten

In dieser Phase können Daten vom Gerät gelesen und für Force-Feedback Geräte auch geschrieben werden. DirectInput kennt zwei Möglichkeiten, Gerätedaten abzufragen. Im Pollingmodus führt die Applikation eine Abfrage per Funktionsaufruf durch, im Eventmodus wird eine vorher angemeldete Funktion der Applikation (Callback) aufgerufen, wenn neue Daten verfügbar sind (vergl. Abschnitt 2.2.2). Neben den eigentlichen Nutzdaten kann DirectInput auch Zeitstempel und Sequenznummern liefern. Letztere sind wichtig, festzustellen, ob Daten verlorengegangen sind.

### 2.2.2 Fraunhofer-IGD INTO

Das Interaktionstoolkit INTO wurde im Fraunhofer-IGD als Komponente des VR-Systems Virtual Design II entwickelt [Felg95, FSZ96]. INTO bildet Interaktionsgeräte auf eine Reihe von logischen Geräten ab. Zusätzlich kennt INTO auch sogenannte virtuelle Geräte, die wie physikalische Interaktionsgeräte auf logische Geräte abgebildet werden können. Ein virtuelles Gerät bezeichnet im Gegensatz zum physikalischen Gerät keine tatsächlich vorhandene Hardware, sondern lediglich ein grafisches Objekt, durch das ein physikalisches Gerät nachgebildet wird (vergl. [Iso86]). Dies können beispielsweise Schieberegler in einer grafischen Benutzungsoberfläche oder Schalter in einer virtuellen Umgebung sein, die mit der Maus oder anderen grafischen Objekten betätigt werden können. INTO kennt insgesamt acht logische Geräteklassen: Taste, Wertgeber, Auswähler, Lokalisierer, Identifizierer, Orientierung, Raum und Hand. Da diese zum größten Teil in das IDEAL -System übernommen wurden und in Abschnitt 5.3 diskutiert werden, wird an dieser Stelle auf eine genaue Beschreibung verzichtet und auf ([Felg95] Seite 102-104) verwiesen.

Jedes logische Gerät kann im Polling- oder Eventmodus betrieben werden. Im Pollingmodus werden Geräte von der VR-Applikation zyklisch abgefragt, im Eventmodus wird bei einer Wertänderung eine vordefinierte Funktion (Callback) aufgerufen.

"relativ" die Veränderung seit dem letzten Auslesen des Wertes bedeutet.

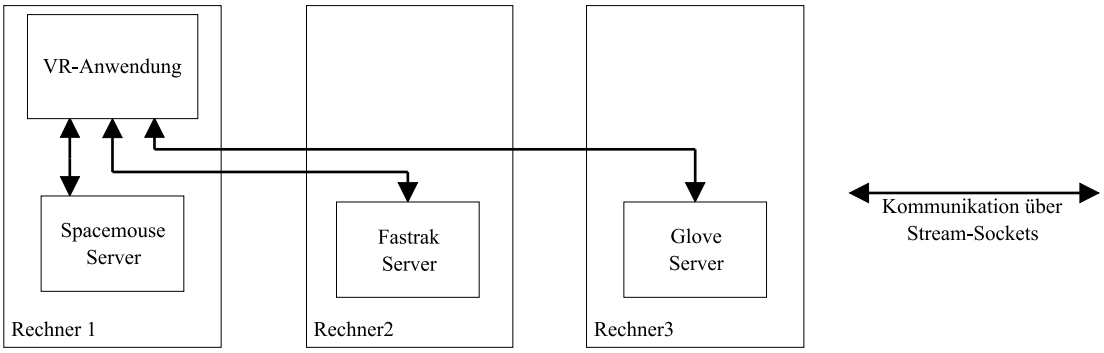


Abbildung 2.7: Applikation und Server mit INTO

INTO identifiziert physikalische Geräte über einen Aufzählungstyp. Für jedes logische Gerät existiert ein Verweis auf das zugeordnete physikalische Gerät unter Verwendung dieses Aufzählungstyps. Daher ist es nicht möglich, zwei oder mehr physikalische Geräte eines Typs zu verwenden.

Mit Hilfe einer Konfigurationsdatei bildet INTO physikalische Geräte auf logische Geräte ab. Die Konfigurationsdatei enthält auch die Information, welcher Server (siehe unten) das Gerät verwaltet, an welchen Rechner es angeschlossen ist sowie die Initialisierungsdaten, wie Baudrate und die Schnittstelle.

Die Verwaltung von physikalischen Geräten wird durch Server durchgeführt. Diese laufen auf dem Rechner, an den das Gerät angeschlossen ist. Die Kommunikation zwischen Applikation und Server wird über Stream-Sockets realisiert. Es ist die Aufgabe der Anwendung alle benötigen Server zu starten und, falls diese nicht mehr benötigt werden, zu beenden. Aus diesem Grund kann ein physikalisches Geräte jeweils nur von einer Anwendung verwendet werden. Wie die Verteilung von Applikation und Servern mit INTO aussehen könnte, stellt Abbildung 2.7 beispielhaft dar.

## 2.3 Vergleich und Bewertung

### 2.3.1 Systemarchitektur

Die untersuchten Systeme können in zwei Gruppen unterteilt werden, in Programmbibliotheken und Skript-basierte Systeme. Zu den Programmbibliotheken gehören SVE, WTK und MR. Sie bestehen im Wesentlichen aus einer Sammlung von Funktionen, die die Bereiche grafische Darstellung, Anbin-

dung von Interaktionsgeräten, das Laden von Szenen in gängigen Datenformaten und mathematischen Funktionen bestehen. Während es sich bei MR um ein Toolkit handelt, also die Applikation die Hauptschleife enthält, bilden SVE und WTK Mischformen aus Rahmensystem (Framework) und Toolkit. Hier liegt zwar die Hauptschleife im System (Framework-Ansatz), große Teile des Systems bestehen jedoch aus Funktionen, die von der Anwendung aufgerufen werden; nach Beendigung der Aufgabe erhält die Anwendung die Kontrolle sofort zurück.

Anwendungen nutzen diese Bibliotheken, in dem sie die zur Verfügung gestellten Funktionen aufrufen diese mit eigenem Programmcode kombinieren. Über Callbacks kann die Anwendung in diese Systeme eigene Funktionen in dessen Kontrollfluß integrieren. Soll die das Verhalten eines Teilsystems einer bestehenden Anwendung geändert oder erweitert werden, erfordert dies eine neue Übersetzung des Programmcodes. Dies führt zu monolithischen Anwendungen und erhöht die Turnaround Zeiten[4], was solche Systeme schwer optimierbar und wartbar macht. Andererseits sind Systeme, in denen das Verhalten einer virtuellen Umgebung in Maschinensprache übersetzt werden kann, sehr performant.

Anders ist dies bei den der zweiten Gruppe, den Skript-basierten Systemen. Zu diesen sind Alice, WorldUp und die VRML-basierten Systeme zu rechnen. Da diese über Konfigurationsdateien und Skripte programmiert werden, ist ein Neustart des Systems bei einer Änderung des Verhaltens nicht prinzipiell nötig. Diese Systeme sind als Frameworks gestaltet, in die der Programmierer seinen Skriptcode einfügt. Der Programmablauf wird durch das Framework kontrolliert. Im Vergleich zu den Programmbibliotheken ist die Performanz wesentlich geringer, da Skriptsprachen interpretiert werden. Für aufwendige Simulationen und komplexe Virtuelle Umgebungen ist dieser Ansatz daher kaum geeignet.

### 2.3.2 Struktur der Datenbasis

Alle untersuchten Systeme verwenden einen Szenengraphen zur Darstellung von virtuellen Umgebungen. Bei hierarchischen Szenengraphen handelt es sich um gerichtete azyklische Graphen, also Bäume. Die Blätter dieses Baumes bilden die Teilgeometrien der Szene. Deren Hüllvolumen bilden das Ordnungskriterium des Baumes. Die Vereinigungsmenge der Hüllvolumen aller Kindknoten bildet das Hüllvolumen des Elternknotens. Bei der graphischen

[4]Zeit, die von dem Erkennen eines Fehlers, über dessen Beseitigung bis zum erneuten Test vergeht.

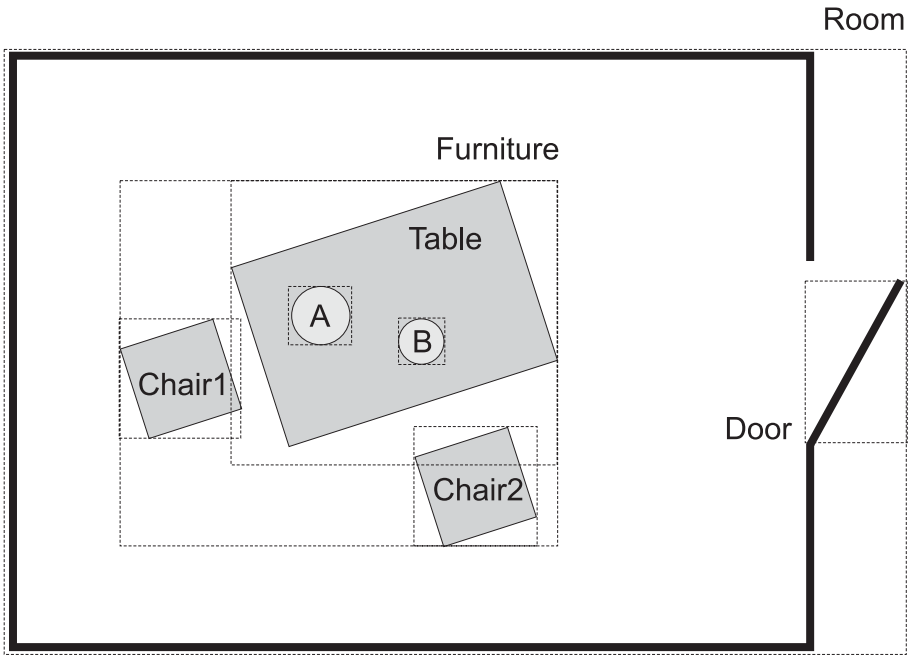


Abbildung 2.8: Raum mit Möbeln in der Draufsicht

Darstellung der Szene kann anhand der Hüllvolumen schnell erkannt werden, welche Teile der Szene nicht dargestellt werden müssen.

Diese Darstellung ist zwar für die effiziente Darstellung einer Szene gut geeignet, nicht aber alleinige Grundlage für die Beschreibung einer komplexen dynamischen virtuellen Welt.

Betrachten wir zunächst eine einfache Beispielszene, die einen Raum mit Möbeln und den Gegenständen A und B. Abbildung 2.8 stellt diese in der Draufsicht dar.

Handelt es sich um eine statische Szene, in der kein Objekt bewegt werden soll, ergibt sich ein Szenengraph wie sie in Abbildung 2.9 links dargestellt ist. Als Ordnungskriterium für den Aufbau des Szenengraphen wird eine Hierarchie der Hüllvolumen (in Abbildung 2.8 gestrichelt dargestellt) herangezogen. Das Hüllvolumen jedes Elternknotens umschließt das seiner Kinder.

Die Situation ändert sich nun, wenn die Objekte A und B bewegt werden sollen. Sie dürfen nun nicht länger Teil der statischen Szenenhierarchie bleiben, da diese entarten würde. In unserem Beispiel würden, wenn A oder B bewegt werden, die Hüllvolumen aller übergeordneten Knoten (Table, Furniture usw.) so anwachsen müssen, daß sie A und B weiterhin enthalten.

Daher werden bewegliche, also dynamische Objekte in der Praxis typischerweise getrennt von der statischen Szenenhierarchie direkt unter der Wur-

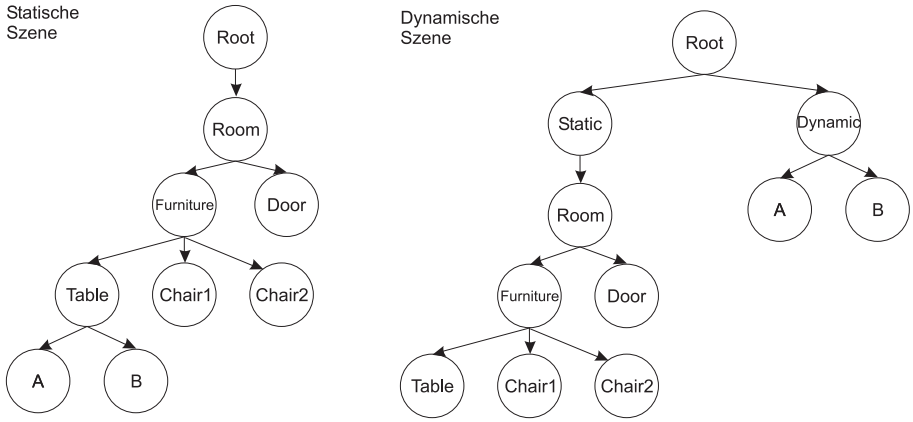


Abbildung 2.9: Statische und dynamische Szenenhierarchie der Beispielszene

zel des Szenengraphen eingeordnet, wie auf der rechten Seite in Abbildung 2.9 dargestellt. Auf diese Weise vermeidet man ein Entarten der Hierarchie der Hüllvolumen.

Dynamische Virtuelle Umgebungen enthalten nun aber nicht nur einige wenige, sondern eine große Zahl von beweglichen Objekten. Wird die Szene alleine durch den Szenengraphen dargestellt, entsteht für die Suche nach einem bestimmten Objekt ein Aufwand, der linear mit der Zahl der beweglichen Objekte wächst. Oft muß beispielsweise nach Objekten gesucht werden, die sich in einem bestimmten Volumen im Raum befinden, oder eine andere Eigenschaft besitzen.

Noch problematischer wird die Situation, wenn zwischen beweglichen Objekten bestimmte Relationen berechnet werden sollen, etwa wenn für alle Objekte der Abstand zum nächsten Objekt ermittelt werden soll. In diesem Fall steigt der Aufwand quadratisch mit der Zahl der Objekte.

Es stellt sich also heraus, daß der Szenengraph allein nicht die geeignete Datenstruktur ist um komplexe, dynamische Szenen darzustellen. Strukturen für eine Suche im Raum oder nach Objekten mit bestimmten Eigenschaften müssen ergänzt werden. Dies geschieht in der Praxis natürlich, etwa mit Octrees [Tamm84] oder Gittern [Froh00, Froh97], muß aber für jede Anwendung neu implementiert werden. Dieses Vorgehen ist umständlich und aufwendig, daher werden solche Strukturen in die Welt-Datenbasis integriert die Teil des im Rahmen dieser Arbeit entwickelten Simulationssystem ist. So stehen diese für allen Anwendungen zur Verfügung.

### 2.3.3 Parallelisierung von Darstellung und Simulation

Werden grafische Darstellung und Simulation in nebenläufigen Prozessen oder Threads durchgeführt, kann für komplexe dynamische Welten moderne Multiprozessor-Hardware genutzt werden. Daher ist es wünschenswert, wenn ein VR-System eine solche Möglichkeit bietet. Allerdings enthält lediglich Minimal Reality solche Mechanismen [SGLS93], während diese in SVE explizit ausgeschlossen wird [SVE02]. Alle anderen untersuchten Systeme bieten keine Unterstützung für die Parallelisierung; diese bleibt Aufgabe der Anwendung.

Soll die Parallelisierung von Rendering und Simulation nun durch die Anwendung realisiert werden, erweist sich wiederum die Vermischung von graphischen und simulierten Objekten im Szenengraphen als Nachteil. Arbeiten beide Prozesse parallel auf den selben Datenstrukturen, müssen alle Zugriffe synchronisiert werden. Keines der Systeme, die ihr Objektmodell von grafischen Objekten herleiten, bietet die nötigen Mechanismen für diese Synchronisation.

### 2.3.4 Objektmodell

Die untersuchten Systeme verwenden Objektmodelle, die nicht mehr sind als grafische Objekte oder leiten eigene Modelle direkt von diesen ab. Dieser Ansatz ist für die Simulation von dynamischem Verhalten problematisch, wie im Folgenden ausgeführt wird.

Viele Komponenten der Simulation besitzen keine direkte grafische Repräsentierung, sondern wirken sich bestenfalls indirekt auf die grafische Darstellung aus, beispielsweise, Zeitgeber oder Interpolatoren. Daher werden sie, beispielsweise in VRML völlig willkürlich in den Szenengraphen eingefügt[5]. Objekte ohne grafische Darstellung gehören nicht in den Szenengraphen.

Über konzeptionelle Schönheitsfehler hinaus, sind grafische Objekte für die Darstellung einer nicht-trivialen Simulation meist nicht mächtig genug. Ist das Objektmodell nicht erweiterbar, wie bei MR, SVE und WTK, muß die Simulation auf Schattenobjekten durchgeführt werden. Nach Beendigung eines Simulationsschrittes erfolgt die Aktualisierung der grafischen Szenenbeschreibung. Diese Trennung von Simulation und grafischer Darstellung ist sinnvoll und konzeptionell sauber, aber die untersuchten Systeme bieten solche Datenstrukturen nicht, sodaß diese für jede Anwendung neu implementiert werden müssen.

[5]Wie bereits erwähnt, ist das Ordnungskriterium für den Szenengraphen das Hüllvolumen. Welches Hüllvolumen haben Objekte ohne grafische Darstellung?

VRML und Alice bieten ein erweiterbares Objektmodell. Allerdings ist die Menge der Datentypen und -Strukturen im VRML-Standard festgelegt [Iso97] und nicht notwendigerweise für alle Simulationsdaten ausreichend. Alice ist hier flexibler, verwendet aber die interpretierte Skriptsprache Python [Beaz01], was die Komplexität der Simulation von vornherein begrenzt.

Keines der untersuchten Systeme bietet über die vorgegebene Klassenhierarchie hinaus die Möglichkeit, Objekte in anwendungsspezifische logische Klassen einzuteilen. So kann zwar ein Schalter oder ein Fahrzeug durch eine bestimmte Geometrie dargestellt werden, die Information darüber, daß es sich bei der Geometrie um einen Schalter, ein Fahrzeug bestimmten Typs, etc. handelt muß die Anwendung selbst bereithalten. Damit ist es ebenfalls nicht möglich, nach Objekten zu suchen, die zu bestimmten Klassen (außer den eingebauten) gehören was wiederum die Modellierung von dynamischem Verhalten erschwert.

### 2.3.5 Statisches und dynamisches Verhalten

Die semantischen Beziehungen zwischen Objekten werden in den untersuchten Systemen entweder per Programmcode, bei VRML-basierten Systemen über Routes und bei WorldUp über Verbindungen zwischen *trigger* und *action* Objekten festgelegt.

Damit lassen sich die statischen Verbindungen zwischen Objekten (z.B. Schalter und Lampe, Kommando und Aktion) adäquat ausdrücken. Solche Beziehungen kommen in allen virtuellen Umgebungen bei der Modellierung der Benutzerinteraktion und bei Maschinen-artigem Verhalten von Objekten vor. Jedes System muß also die Möglichkeit von statischen Verbindungen zwischen Objekten zulassen.

In komplexen Simulationen beispielsweise von Straßenverkehr, physikalischen oder biologischen Systemen ändern sich die Beziehungen zwischen den Objekten ständig. Hier interagieren Objekte die einen bestimmten Abstand voneinander unterschreiten oder abhängig von einem wechselnden inneren Zustand. Hier reicht die statische Verbindung von Objekten untereinander nicht mehr aus. Gleiches gilt für hoch-interaktive Anwendungen, mit einer Großen Zahl von Objekten, die ihre Beziehung zum Benutzer ändern.

Durch die ungünstige Struktur der Datenbasis wird dieses Problem noch erschwert, da diese keine Informationen bereitstellt, die die veränderlichen Verbindungen zwischen Objekten darstellen.

### 2.3.6 Integration von multidimensionalen Interaktionsgeräten

Alle untersuchten Systeme mit der Ausnahme von Alice weisen Mechanismen zur Geräteabstraktion auf. Über logische Geräte können physikalische Geräte damit theoretisch hardwareunabhängig angesprochen werden. Die Geräteabstraktion bei World Toolkit beschränkt sich aber ausschließlich auf 6D-Geräte und Tasten. Geräte mit mehr Freiheitsgraden oder anderen Funktionen (etwa Beugungssensoren eines Datenhandschuhs) werden nicht berücksichtigt. Um solche Geräte zu integrieren, müssen deren Daten über geräteabhängige externe Treiber verfügbar gemacht werden. Dieses Systeme erlaubt daher keine allgemeine hardwareunabhängige Integration für die in Abschnitt 5.2 diskutierten Gerätetypen.

DirectInput und INTO lösen dieses Problem dagegen zufriedenstellend. INTO bietet eine ganze Reihe von logischen Sensoren an, die die Bandbreite der diskutierten Systeme gut abdeckt. DirectInput bildet jeden Freiheitsgrad eines Gerätes einzeln auf sogenannte Object Instances ab. Damit steht ein sehr flexibler Abstraktionsmechanismus zur Verfügung, der der großen Zahl verschiedener Spielecontroller gerecht wird. Nachteilig ist allerdings der daraus folgende komplizierte und fehlerträchtige Initialisierungsvorgang zu bewerten. Besonders zu erwähnen ist, daß DirectInput als einziges der untersuchten Systeme haptische Interaktionsgeräte unterstützt.

INTO und MR Toolkit erlauben als einzige Systeme, die Abbildung logischer auf physische Geräte über eine externe Konfigurationsdatei vorzunehmen. Die Tatsache, daß bei INTO physische Geräte durch einen Aufzählungstyp identifiziert werden und nur eine globales Geräteverzeichnis existiert, bedingt aber daß immer nur ein Gerät eines Typs gleichzeitig von einer Applikation benutzt werden kann.

Alle untersuchten Systeme bieten die Möglichkeit, mit Hilfe von Maus und Tastatur im dreidimensionalen Raum zu interagieren. Werden diese Geräte auf logische 6D-Geräte abgebildet, werden die verfügbaren Freiheitsgrade in den dreidimensionalen Raum projiziert und die fehlenden Freiheitsgrade auf verschiedene Art und Weisen simuliert.

Zu den Systemen, welche netzwerkweiten Zugriff auf Interaktionsgeräte zulassen, gehören SVE, INTO und MR Toolkit. DirectInput, Alice, WTK und WorldUp können nur die am lokalen Rechner angeschlossenen Geräte verwenden.

Keine System erlaubt es, im Fehlerfall Geräte umzukonfigurieren oder gar auszutauschen. Diese Aufgabe fällt der Applikation oder aber den Gerätetreibern zu.

| | **SVE** | **MR** | **Alice** | **WTK** | **WorldUp** | **VRML** |
|---|---|---|---|---|---|---|
| Architektur | Framework | Toolkit | Framework | Mischform | Framework | Framework |
| Szenenbeschreibung | Szenen-hierachie | Szenen-hierachie | Szenen-hierachie | Szenen-hierachie | Szenen-hierachie | Szenen-hierachie |
| Objektmodell | grafisch | grafisch | grafisch, Python | grafisch | grafisch, trigger-action | grafisch, Routes, Skript |
| Parallelisierbar | nein | ja | nein | nein | nein | nein |
| Programmierung | Callback | Callback | Skript | Callback | Skript, Routes | Skript, Routes |
| Geräteabstraktion | ext. Konfig., nur 6D | ext. Konfig. | keine | nur 6D | nur 6D, Tastatur u. Maus | Tastatur u. Maus |

Tabelle 2.1: Vergleich der untersuchten VR-Systeme

SVE, DirectInput und World Toolkit erlauben es, mehrere Geräte der selben Art gleichzeitig in einer Applikation zu verwenden. INTO und Superscape gehen davon aus, daß von jedem unterstützten Gerät maximal eines verwendet wird. Ausnahmen sind bei INTO lediglich rechter und linker Datenhandschuh und bei Superscape zwei Joysticks.

DirectInput und MR Toolkit sind in der Lage, den Zugriff mehrerer Applikationen auf ein und dasselbe Interaktionsgerät zu regeln. Alle anderen Systeme gehen davon aus, exklusiven Zugriff zu besitzen und funktionieren andernfalls fehlerhaft, was bei in der Regel zum Neustart der Anwendung führt. Bei DirectInput kann eine Applikation den exklusiven oder gemeinsamen Zugriff auf ein Gerät anfordern. Ist das Gerät bereits von einer anderen Applikation belegt, wird diese Anforderung abgelehnt. Auf haptische Geräte wird grundsätzlich exklusiv zugegriffen.

## 2.4 Zusammenfassung

Im Rahmen dieser Arbeit wurden sechs VR-Systeme und zwei Systeme zur Integration von multidimensionalen Interaktionsgeräten untersucht und bewertet. Die Ergebnisse der Untersuchung der VR-Systeme ist in Tabelle 2.1 zusammengefasst[6]

Alle untersuchten Systeme betrachten Objektverhalten nur im Kontext

[6]World Toolkit und dessen Erweiterung WorldUp sind in getrennten Spalten dargestellt.

von dessen grafischer Darstellung. Die Szenenhierarchie, eine Datenstruktur, die eigentlich für effizientes Echtzeit-Rendering entwickelt wurde, wird um Skriptknoten, Callbacks oder Programmierschnittstellen erweitert. Neben konzeptionellen Unsauberkeiten macht dies die Parallelisierung von Simulation und grafischer Darstellung schwierig. Daher überrascht es nicht, daß nur ein System diese Möglichkeit bietet.

Die Darstellung von nicht-grafischen, semantischen Inhalten wird nur ungenügend unterstützt. Damit ist die Modellierung von komplexen virtuellen Umgebungen mit dynamischem Objektverhalten nur mit großem Entwicklungsaufwand möglich.

Bezüglich der Integration von multidimensionalen Interaktionsgeräten ist zunächst zu sagen, das keines der untersuchten Systeme alle Anforderungen erfüllt. Die Geräteabstraktion ist bei den meisten Systemen nicht vorhanden oder deckt nicht den gesamten Bereich ab, die robuste und fehlertolerante Handhabung dieser Geräte wird nicht thematisiert.

Damit sind die beiden wichtigsten Gebiete genannt, auf denen die vorliegende Arbeit Beiträge leisten möchte.