

Entwicklung einer 3D-User-Interface-Bibliothek für technisch-wissenschaftliche VR-Anwendungen

Franz Maurer

Studienarbeit im Hauptfach Angewandte Informatik

Betreuer: Dipl.-Ing. Dirk Rantzau

August 1996

Interner Bericht Nr. 123

RECHENZENTRUM DER UNIVERSITÄT STUTTGART
INSTITUT FÜR COMPUTERANWENDUNGEN II
ANWENDUNGEN DER INFORMATIK IM MASCHINENWESEN
PROFESSOR DR.-ING. R. RÜHLE
RUS, ALLMANDRING 30, 70569 STUTTGART

Abstract

The design of interactive 3D graphic applications has proved in the past to be a time-consuming and difficult task which demands a high level of knowledge and experience from the participating developers. Especially the user interface for virtual reality applications should be intuitive and easy to learn for the user but on the other hand also being adjustable to the sometimes very specific application areas. This paper describes the design and implementation of a software library (based on the commercial WorldToolKit library) with some standard 3D interaction elements like 3D buttons and sliders. Later on we will introduce an alternative approach for direct object manipulation using a 'select and manipulate' paradigm.

Keywords: 3D user interface design, 3D interaction, direct manipulation, virtual reality, rapid prototyping.

Inhaltsverzeichnis

1	Einleitung	4
2	Virtuelle Realität	6
2.1	Begriffsdefinition und VR-Grundsysteme	6
2.1.1	Immersive Systeme	7
2.1.2	Telepresence	7
2.1.3	Mixed Reality (Augmented Reality)	8
2.1.4	Fish Tank Virtual Reality	8
2.2	Trackingsysteme	8
2.3	3D-Eingabegeräte	9
2.4	Ansprechen weiterer Sinnesorgane	9
2.5	Das RUS VR-System	10
3	WorldToolKit	12
3.1	Konzept	12
3.2	Die Simulationsschleife	13
3.3	Die Sichtgeometrie	14
3.4	Koordinatensysteme, Position und Orientierung	15
3.5	Klassen und Methoden	17
3.5.1	Das Universum	17
3.5.2	Der Standpunkt	18
3.5.3	Das Objekt	18
3.5.4	Der Sensor	18
3.5.5	Weitere Klassen	19
4	Gestaltung der 3D-Benutzerschnittstelle	20
4.1	Pflichtenheft	20
4.2	3D-Interaktoren	21
4.2.1	3D-Button	22
4.2.2	3D-Slider	23
4.3	3D-Manipulator	24
4.4	Aufbau der Bibliothek	26
4.4.1	WTIE_Manager	28

4.4.2	WTIE_Event	29
4.4.3	WTIE_Sensor	29
4.4.4	WTIE_Application	30
4.4.5	WTIE_SelObject	30
4.5	Beispielanwendung	31
5	Zusammenfassung	34
5.1	Probleme und Bewertung	34
5.2	Entwicklungsstand	35
5.3	Ausblick	35
A	Programmierung	37
A.1	WTK-Sensortreiber für den 5th Dimension-Datenhandschuh	37
A.1.1	Aufbau	37
A.1.2	Grafisches Handmodell	38
A.1.3	Handstellung und Handgeste	39
A.1.4	Einbettung des Eingabegerätes in die WTIE-Bibliothek	40
A.2	Programmierschnittstellen	43
A.2.1	Symbolische Konstanten	43
A.2.2	WTIE_Button	45
A.2.3	WTIE_ButtonBar	48
A.2.4	WTIE_Event	50
A.2.5	WTIE_Frame	52
A.2.6	WTIE_Manager	55
A.2.7	WTIE_Object	57
A.2.8	WTIE_SelObject	59
A.2.9	WTIE_Sensor	62
A.2.10	WTIE_Slider	64
A.2.11	WTIE_Text	67
B	Fehlermeldungen	68
B.1	Notation	68
B.2	Häufiger auftretende Fehlermeldungen	69
C	Hardware	70

Abbildungsverzeichnis

2.1	Das RUS VR-System	10
3.1	WorldToolKit Simulationsschleife	13
3.2	Monoskopische Sichtgeometrie und Bezugskoordinatensysteme . .	15
3.3	Rotation durch ein Quaternion	16
4.1	Verschiedene Zustände beim 3D-Button	22
4.2	3D-Slider-Ausrichtung	23
4.3	Anordnungsmöglichkeiten der 3D-Slider-Beschriftung	24
4.4	Modell des Manipulatorobjektes	25
4.5	OMT-Diagramm der WTIE-Bibliothek	27
4.6	Geänderte WTK-Simulationsschleife	28
4.7	Beispielanwendung	31
4.8	Datenhandschuh in Probing-Handstellung	32
A.1	Prinzipieller Aufbau des grafischen Handmodells	38
A.2	Grafisches Handmodell in Auswahlhaltung	41

Kapitel 1

Einleitung

Die Entwicklung interaktiver 3D-Grafikanwendungen hat sich in der Vergangenheit als äußerst zeitaufwendig und schwierig erwiesen und von den Programmierern einen hohen Grad an Wissen und Erfahrung vorausgesetzt. Insbesondere die Gestaltung der Benutzerschnittstelle in Anwendungen der virtuellen Realität (VR) erweist sich dabei als kritisch. Einerseits soll sie für den Benutzer intuitiv sein, andererseits aber dem jeweiligen, u. U. sehr speziellen, Anwendungsgebiet angepaßt sein. Die Lernphase für den Benutzer sollte dabei möglichst kurz gehalten werden. Während in VR-Anwendungen wie Architectural-Walkthroughs die Benutzerschnittstelle zur Interaktion mit den der physischen Welt nachempfundenen Objekten noch vergleichsweise einfach zu gestalten sind, erweist sich die Manipulation von abstrakten Objekten z. B. aus Simulationsrechnungen in 3D-Räumen aus verschiedenen Gründen als schwierig und stellt besondere Anforderungen an die Mensch-Maschine-Schnittstelle.

Im Rahmen dieser Arbeit soll aufsetzend auf ein vorhandenes Basis-Toolkit für VR-Anwendungen (WorldToolKit) zunächst eine kleine Bibliothek mit Standard-3D-Interaktoren wie 3D-Buttons und 3D-Sliders entwickelt werden. Danach soll anhand eines spezifischen Anwendungsfalles, nämlich der Berechnung und Darstellung von Schnittflächen in strukturierten Gittern, speziell hierfür angepaßte Manipulatoren entwickelt werden, um beispielsweise durch Verschieben und Drehen eine Rückkopplung in die Berechnung zu bekommen und das neu berechnete Ergebnis an der mit den 3D-Eingabegeräten eingestellten Position sichtbar zu machen. Diese Arbeit ist Teil des Sonderforschungsbereichs 374, welches sich mit dem Themenbereich Rapid Prototyping und im Teilprojekt 'Simulation und Visualisierung technisch physikalischer Vorgänge' vor allem mit der direkten Interaktion und Manipulation in VR-Umgebungen beschäftigt.

Im ersten Teil dieses Berichtes wird die Motivation der Arbeit dargelegt. Kapitel 2 bietet dazu eine kurze Übersicht über häufig verwendete VR-Grundsysteme und

die in diesem Zusammenhang verwendete Terminologie. Eine große Bedeutung in der Gestaltung der Benutzerschnittstelle haben die verwendeten 3D-Eingabegeräte. Sie bestimmen im wesentlichen die Möglichkeiten der Interaktion. Das hier am Rechenzentrum verwendete VR-System wird dabei ebenfalls kurz angesprochen. Als Basis für die eigene User-Interface-Bibliothek dient die Bibliothek WorldToolKit, deren grundsätzliches Konzept und speziell deren Eigenschaften in Hinsicht auf die Entwicklung von VR-Anwendungen kurz dargelegt werden. Kapitel 4 widmet sich der 3D-User-Interface-Bibliothek. Ausgehend von den Anforderungen wird das Design der Dialogelemente und die Konzeption der Bibliothek ausführlich erläutert. Die bei manchen 3D-Eingabegeräten auftretende Schwierigkeiten bei der Interaktion mit den bisher entworfenen Dialogelementen führte zu einem alternativen Ansatz der Interaktionsmöglichkeit. Das 'Auswählen und Manipulieren' selektierbarer Objekte in der VR-Umgebung ist vom Konzept her schon von grafischen Objekten in 2D-Zeichenprogramme bekannt. Aufgrund dieses Ansatzes wird die direkte Interaktion mit 3D-Eingabegeräten, wie z. B. dem Datenhandschuh, erheblich erleichtert. Der hier vorgestellte Ansatz ermöglicht desweiteren die Verwirklichung eines 'hybriden Interface', d. h. die Interaktion kann sowohl klassisch am Bildschirm sitzend (*Desktop Oriented*) als auch in der am RUS vorhandenen VR-Umgebung vorgenommen werden. Kapitel 5 faßt zusammen, bewertet die Ergebnisse und bietet einen Ausblick.

Der Anhang beschäftigt sich mit Aspekten zur Programmierung. Am Beispiel des implementierten Treibers für den 5D-Datenhandschuh wird der grundsätzliche Aufbau eines WorldToolKit-Gerätetreibers sowie die Einbettung in die hier vorgestellte 3D-User-Interface-Bibliothek dargelegt. Die Beschreibung der Programmierschnittstellen im zweiten Teil dieses Kapitels vermittelt einen detaillierten Einblick in die Bibliotheksstruktur. Desweiteren kommt im Anhang die in der Bibliothek verwendete Notation bei der Fehlerausgabe zur Sprache. Eine kurze Übersicht über häufiger auftretende Fehlermeldungen sowie deren möglichen Ursache und Beseitigung soll helfen selbige aus dem Weg zu räumen. Zum Schluß wird ein kurzer Überblick über die wichtigsten technischen Daten der eingesetzten Hardware gegeben.

Kapitel 2

Virtuelle Realität

Bevor ich auf Einzelheiten zum Design und zur Implementation der hier vorgestellten Bibliothek eingehen kann, möchte ich zunächst einmal die Voraussetzungen zu dieser Arbeit darlegen. In diesem Kapitel möchte ich deshalb ein paar Anmerkungen zum Begriff der virtuellen Realität machen. Mit der großen Verbreitung und Bekanntheit der VR in den letzten Jahren sind unterschiedliche Terminologien entstanden. Eine Definition des Begriffs der VR zu Beginn dieses Kapitels, soll einen gemeinsamen Nenner für die weiteren Ausführungen schaffen. Ein kurzer Abriss über die Eigenschaften und Einsatzbereiche der verschiedenen VR-Grundsysteme, sowie den dabei verwendbaren Trackingsystemen und 3D-Eingabegeräten, soll einen Überblick gegenwärtiger VR-Technologie geben. Zu diesem Thema existieren bereits zahlreiche Publikationen. Stellvertretend sei auf die Bücher [2], [15] und [6] verwiesen. Die Beschreibung des hier am Rechenzentrum eingesetzten VR-Systems am Ende des Kapitels zeigt mit dem zuvor Beschriebenen die Vor- und Nachteile eines solchen Systems. Es wird aber auch der primäre Verwendungszweck deutlich und läßt Aussagen zur Übertragbarkeit auf andere Systeme zu.

2.1 Begriffsdefinition und VR-Grundsysteme

Eine brauchbare Definition der virtuellen Realität liefert das Buch 'Silicon Mirage' der Autoren S. Aukstakalnis und D. Blatner. Darin heißt es: '*Computer übernehmen die Aufgabe, virtuelle Welten, die so nur dort existieren, für den Anwender mittels hochkomplexer Daten interaktiv zu visualisieren, zu simulieren und sensorisch erfaßbar zu machen.*' Ein entsprechendes Szenario kann aus einem CAD-Modell bestehen, etwa die realistische, begehbare Abbildung der Inneneinrichtung eines Gebäudes, der virtuellen Prototypenstellung von Produkten wie Autos oder eine wissenschaftliche Simulation der Vorgänge im Innern der Erd-

kruste durch die Darstellung des geophysikalischen Modells. Als Benutzer kann man mit diesen Modellen interagieren oder Objekte in der Kunstwelt verändern. Virtuelle Gegenstände bis zu 'Welten in Welten' lassen sich dabei von anderen Prozessen animieren oder etwa durch die physikalische Simulation beeinflussen. Die Interaktion erfolgt in Echtzeit und in Abhängigkeit vom Standpunkt (*View-point*) des Betrachters.

Da sich die unterschiedlichen VR-Systeme hinsichtlich ihres Einsatzgebietes stark unterscheiden möchte ich zunächst einen Überblick über die wichtigsten Systeme und ihre spezifischen Eigenschaften geben. Grundsätzlich lassen sich die wesentlichen VR-Systeme in vier Grundtypen unterteilen.

2.1.1 Immersive Systeme

Immersive Systeme weisen Eigenschaften auf, die der idealtypischen Definition von VR nahekommen. Dem Benutzer wird dabei der Eindruck vermittelt, von virtuellen Objekten umgeben zu sein. Durch die Benutzung eines HMD-Helms (*Head Mounted Display*) beispielsweise erhält der Benutzer das Gefühl, sich komplett in der Simulation zu befinden, regelrecht darin 'einzutauchen'. Neben einer Stereoptik verfügt solch ein Helm in der Regel auch über eine Möglichkeit der 3D-Audioausgabe und ein Sensorsystem (*Tracking System*), das dem Computer die Position und Blickrichtung des Betrachters zurückliefert.

2.1.2 Telepresence

Diese Technologie verbindet Remote-Sensoren in der realen Welt mit den Sinnen eines menschlichen Operators. Die Sensoren (wie Videokamera, Druck, Wärme) können etwa an einem entfernt arbeitenden Roboter angebracht sein. Der Operator kann den Roboter fernsteuern, weil er alle relevanten Daten aus dessen Umgebung zur Verfügung hat. Solche Telerobotics-Anwendungen werden in der Raumfahrt, aber auch beim Experimentieren mit gefährlichen Präparaten in der chemischen oder Atomindustrie praktiziert. Telepresence spielt auch im Bereich der invasiven Mikrochirurgie eine große Rolle, wo es darum geht, durch die Benutzung von sehr kleinen Instrumenten (Endoskope) den Eingriff beim Patienten minimal zu halten und dem Chirurgen dreidimensionale Orientierung zu ermöglichen.

2.1.3 Mixed Reality (Augmented Reality)

Als Mixed Reality oder auch Augmented Reality bezeichnet man die Kombination von VR und Telepresence. Dabei lassen sich computergenerierte Daten mit den Telepresence-Daten oder der Sicht des Benutzers auf die reale Welt mischen. Ein Beispiel dafür wäre etwa die Betrachtung einer Gehirnoperation durch eine Videokamera vor Ort. Dieses Bild könnte sich etwa mit vorher erstellten Computertomographie-Aufnahmen und einer aktuellen Ultraschallaufnahme überlagern lassen.

2.1.4 Fish Tank Virtual Reality

Der Begriff 'Fish Tank Virtual Reality' (die dabei auftretende räumliche 3D-Anwenderperspektive ähnelt der des Blickes in ein großes Aquarium) wird auf Systeme angewendet, die folgende Komponenten aufweisen: einen Stereomonitor oder ein Leinwandprojektionssystem (sorgt für die aus zwei Perspektiven überlagerte Bildausgabe), eine LCD-Shutterbrille (Spezialbrille für stereoskopische Sicht, zur Dekodierung der Monitorinformation) und ein Tracking-System zur Positionsbestimmung des Anwenders.

2.2 Trackingsysteme

Als Bewegungs- und Positionsrückmelder werden in Virtual-Reality-Anwendungen Trackingsysteme eingesetzt. Sie lassen sich in vier Kategorien unterteilen: in akustische, optische, magnetische sowie mechanische Systeme. **Akustische Systeme** (Ultraschall) - wie z. B. der Red Baron von Logitech - bestehen aus Sendern im Eingabegerät bzw. dem HMD-Helm und einem Empfänger, der auf dem Computermonitor plaziert wird. Die aktuelle Sensorposition wird mehrmals in der Sekunde übermittelt. Die Vorteile solcher Systeme liegen in der geringen Komplexität und den geringen Kosten. Die Nachteile sind das notwendige freie Sichtfeld zwischen Sender und Empfänger, der kurze Abstand und das stark verrauschte Signal. **Optische Systeme** integrieren Sender und Empfänger in einem Gerät, das auf dem Computermonitor plaziert wird. Optische Systeme liefern oft ein sehr gutes Signal. Sie benötigen aber ein freies Sichtfeld und sind meist sehr schwer zu installieren. **Elektromagnetische Systeme** eignen sich im Vergleich zu den beiden vorherigen Systemen vorzugsweise für den Aufbau von immersiven VR-Anlagen. Die Sender sind üblicherweise im (realen) Raum verteilt. Die Empfänger werden meist am Kopf oder an der Hand befestigt. Die Vorteile elektromagnetischer Systeme sind die ausgereifte Technologie, die Störunanfälligkeit gegenüber nichtmetallischen Objekten zwischen Sender und Empfänger sowie die kleinen,

und damit leichten, Empfangsgeräte. Die Nachteile liegen in der schlechten Genauigkeit, in den kurzen Abständen sowie in der Störanfälligkeit durch elektromagnetischer Strahlung (wie z. B. Monitore) und andere metallische Gegenstände in der VR-Umgebung. **Mechanische Kombinationssystem** kommen in den Arm Mounted Displays, z. B. dem BOOM von Fakespace, zum Zuge. Meistens stellen sie eine Kombination aus mechanischem Trackingsystem und Display dar, bei dem im Schwenkarm ein Positionssender integriert ist. Die Vorteile liegen in der hohen Genauigkeit sowie der Möglichkeit große Anzeigegeräte zu verwenden. Die Nachteile liegen in den hohen Kosten und einer verminderten Bewegungsfreiheit.

2.3 3D-Eingabegeräte

Für die Interaktion mit Objekten oder die Kontrolle von 3D-Dialogelementen werden 3D-Eingabegeräte verwendet. Oft kann der Benutzer sein reales Umfeld optisch nicht wahrnehmen oder kann aufgrund der räumlichen Entfernung keine Eingaben über die Tastatur des angeschlossenen Rechners tätigen. Zu diesem Zweck werden die 3D-Eingabegeräte eingesetzt, deren 'Cursor' (bei einem Datenhandschuh ist dies z. B. ein grafisches Modell einer Hand) in der virtuellen Szene sichtbar sind. Zukünftig wird wohl die Spracheingabe diese Aufgabe erledigen, die sich zur Zeit aber noch als sehr beschränkt einsetzbar erweist. Bewährt haben sich Eingabegeräte wie die Space Mouse bzw. der Spaceball, die 3D Mouse und der Datenhandschuh. Kennzeichnend für die meisten 3D-Eingabegeräte sind die sechs Freiheitsgrade der Bewegung, sowie die Möglichkeit diverse Funktionstasten zur Dateneingabe zu betätigen.

2.4 Ansprechen weiterer Sinnesorgane

Wenn ein Anwender in eine virtuelle Umgebung 'eintaucht', kann er sich in den bisher beschriebenen Systemen meist fortbewegen, Objekte berühren oder verändern. Wenn sich beispielsweise das grafische Modell des Eingabegerätes mit einem Simulationsobjekt schneidet kommt es zu einer virtuellen Berührung (*Collision Detection*). Wie sich die Simulation an diesem Punkt weiterverhält ist anwendungsspezifisch. Wie aber erfährt der Benutzer, daß er ein Objekt berührt? In dem bisher noch wenig erforschten Bereich der VR gibt es Ansätze, die versuchen, über eine Vibration an verschiedenen Stellen des Körpers für eine Rückmeldung zu sorgen. Hebt man virtuell einen Gegenstand an, weiß man dadurch allerdings immer noch nicht, wie schwer das virtuelle Objekt ist. Mit Hilfe von kleinen Kammern im Datenhandschuh, die sich schnell mit Luft füllen bzw. leeren lassen, versuchen VR-Gerätehersteller eine bessere Rückmeldung (*Force Feedback*)

zu vermitteln. Solche Systeme sind derzeit noch nicht marktreif oder sehr teuer. Virtual Reality versucht weitgehend, alle Sinne des Menschen miteinzubeziehen. Dazu zählt auch die Fähigkeit des Menschen, beim Hörvorgang auf die Richtung, Position und Art des Objektes, von dem ein Geräusch ausgeht, zu schließen. Für eine optimale Ausnutzung dieser Fähigkeit reicht allerdings konventioneller Stereoklang in der Regel nicht aus. Deshalb kommen hier Surround-Systeme zur akustischen Rückkopplung zum Einsatz.

2.5 Das RUS VR-System

Das derzeit am Rechenzentrum vorhandene VR-System kann man als 'Fish Tank Virtual Reality'-System klassifizieren. Abbildung 1.1 zeigt schematisch den Aufbau.

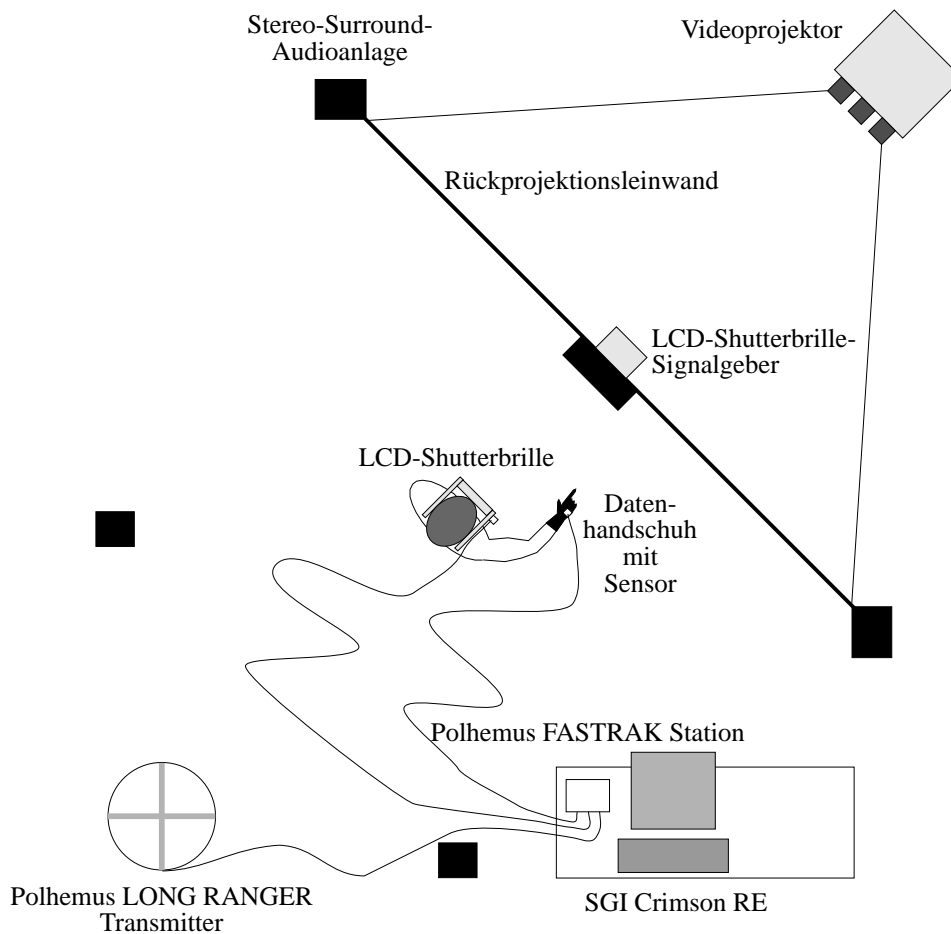


Abbildung 2.1: Das RUS VR-System

Die Ausgabe des Stereobildes auf dem Computermonitor wird mit einem Videoprojektor auf eine großflächige Projektionsleinwand projiziert. Zu einer weiteren Verstärkung des Realitätseindrucks könnten mehrere Leinwände zu einem sog. CAVE [5] kombiniert werden. Oberhalb der Projektionsleinwand befindet sich der Signalgeber zur Synchronisation der LCD-Shutterbrillen mit dem Monitorbild. Der Stereoeffekt wird dadurch erzeugt, daß zwei Bilder (eins für das linke, eins für das rechte Auge) abwechselnd dargestellt werden. Diese Bilder werden nach dem Prinzip des stereoskopischen, räumlichen Sehens berechnet (weiterführende Literatur zu diesem Thema findet man beispielsweise in [14]). Das abwechselnde Darstellen der Bilder findet bei diesem System 96 mal pro Sekunde statt, also 48 Bilder pro Sekunde für jedes Auge. Die Shutterbrille sorgt dafür, daß für jedes Auge nur das individuell berechnete Bild sichtbar ist. Dazu besitzt die Shutterbrille für jedes Auge ein Ein-Pixel-LCD-Display, die entweder transparent oder opak geschaltet werden können. Wird bei der Projektion des Bildes für das linke Auge die LCD für das rechte opak geschaltet und umgekehrt entsteht der visuell räumliche Eindruck. Aufgrund der Augenträgheit (flüssige Bewegung bei ca. 25 Bilder pro Sekunde) wird das schnelle Umschalten zwischen transparent und opak kaum wahrgenommen.

Mit dem Einsatz von LCD-Shutterbrillen und dem elektromagnetischem Trackingsystem Polhemus FASTRAK und LONG RANGER ergibt sich ein großer Arbeitsraum. Der große Vorteil dieses Systems liegt in der von mehreren Personen gleichzeitig beobachtbaren virtuellen Realität. So kann z. B. eine Person im System agieren und gleichzeitig mit anderen Personen das Dargestellte diskutieren. Diese Eigenschaft ist besonders wichtig bei Anwendungen des Rapid Prototyping. Oft ist nicht nur ein einzelner Ingenieur an der Entwicklung beteiligt, sondern ein komplettes Team. Mit diesem System ist eine kooperative Arbeit an einem Projekt möglich. Als 3D-Eingabegeräte kommen am RUS hauptsächlich ein Datenhandschuh sowie ein 3D-Joystick der Firma 5th Dimension zum Einsatz. Zur weiteren Verstärkung des Realitätseindrucks verfügt das Rechenzentrum über eine Stereo-Surround-Audioanlage.

Kapitel 3

WorldToolKit

In diesem Abschnitt soll der grundsätzliche Aufbau sowie die wichtigsten Eigenschaften zur VR-Bibliothek WorldToolKit erwähnt werden, da sie als Ausgangsbasis für die hier vorgestellte Arbeit diente. Dieses Kapitel ist deshalb für all die gedacht, die noch nicht viel bzw. gar nichts über die Bibliothek WorldToolKit wissen und sich einen kurzen Überblick verschaffen möchten. Im wesentlichen stammen diese Informationen aus dem Benutzerhandbuch [16] und beziehen sich auf die z. Zt. aktuelle Version 2.1 für SGI-Computer.

3.1 Konzept

Die Bibliothek WorldToolKit (im folgenden als WTK bezeichnet) der Sense8 Corporation ist eine Sammlung von über 400 C-Routinen zur Entwicklung von VR-Anwendungen. In der simulierten, virtuellen Welt können grafischen Objekten Eigenschaften und Verhalten aus der realen Welt zugewiesen werden. Die grafische Ausgabe auf dem Computermonitor bietet dabei einen Einblick in die virtuelle Welt (im Handbuch wird von einem beweglichen Fenster zur virtuellen Welt gesprochen). Mit Hilfe von verschiedenartigen Eingabesensoren (von der einfachen 2D-Maus bis hin zum Datenhandschuh CyberGlove der Firma VPL) kann man das Sichtfenster bewegen oder die in der virtuellen Welt vorhandenen Objekte manipulieren und kontrollieren. Objekte können dabei auf unterschiedliche Art und Weise generiert werden (siehe Abschnitt 3.5.3).

Anmerkung:

Im Abschnitt 3.5 wird gezeigt, daß WTK beim prinzipiellen Aufbau und der Namenskonvention ein objektorientiertes Konzept verfolgt. Da es in WTK aber die Klasse `WtObject` gibt und in der objektorientierten Terminologie ein Objekt die Instanz einer Klasse ist, werde ich im folgenden die Klasse `WtObject` als WTK-

Objekt bezeichnen und die Instanz einer Klasse als Objekt, um Zweideutigkeiten zu vermeiden.

3.2 Die Simulationsschleife

Die eigentliche Simulation beginnt in WTK mit dem Eintritt in die Simulationsschleife, die zunächst vorbereitet werden muß (`WTuniverse_ready()`). In der Simulationsschleife ist der Programmablauf von der Simulationsverwaltung bestimmt.

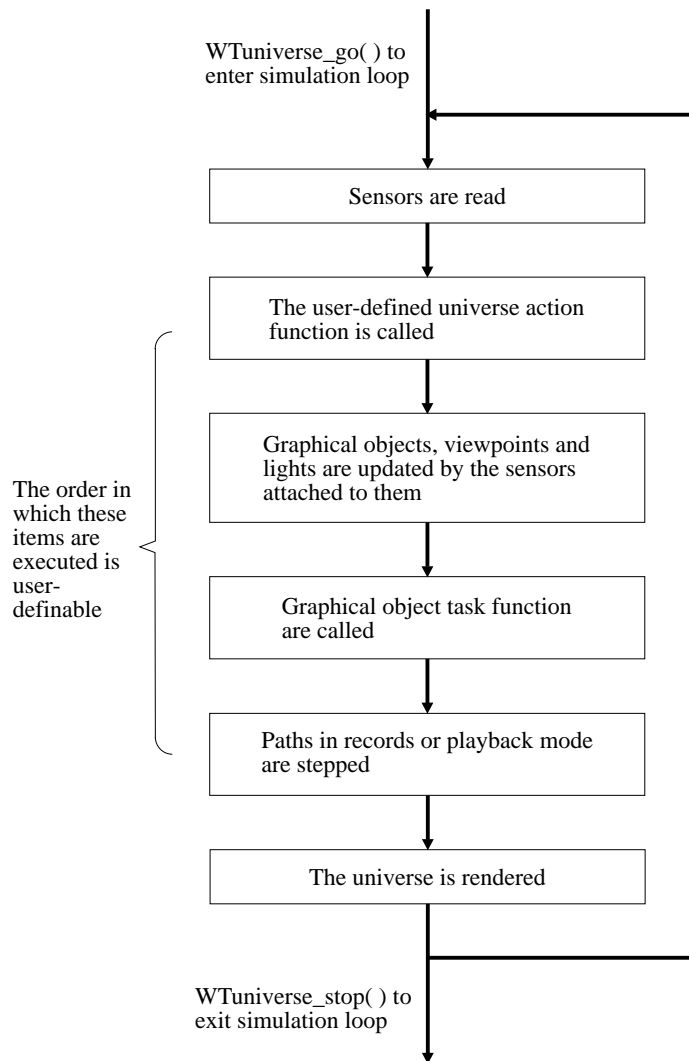


Abbildung 3.1: WorldToolKit Simulationsschleife

Bei einem Schleifendurchlauf werden hauptsächlich die Sensordaten ausgelesen, die Objekte aktualisiert und dargestellt. In einer vom Anwendungsprogrammierer bereitgestellten Funktion (`universe_action_function()`) können z. B. Tastatureingaben abgefragt werden. Wie in Abbildung 3.1 schematisch dargestellt, ist die Reihenfolge einzelner Arbeitspunkte in der Schleife frei wählbar. Im nächsten Kapitel werde ich darauf noch einmal zurückkommen, da die hier vorgestellte 3D-User-Interface-Bibliothek davon Gebrauch macht.

In der Simulationsschleife werden alle im Universum (der virtuellen Welt) enthaltenen Objekte verwaltet. Je mehr Objekte aber enthalten sind, desto langsamer ist die Ausführungsgeschwindigkeit in einem Schleifendurchgang. WTK bietet deshalb die Möglichkeit nicht benutzte Objekte aus der Verwaltung herauszunehmen (`WObject_remove()`) ohne sie zu zerstören (`WObject_delete()`). Bei Bedarf können diese Objekte dann der Simulationsverwaltung wieder hinzugefügt werden (`WObject_add()`). Nur selten benötigte Objekte können schon vor dem Eintritt in die Simulationsschleife erzeugt und aus der Simulationsverwaltung herausgenommen werden. Je nach Bedarf kann man sie dann wieder hinzufügen.

3.3 Die Sichtgeometrie

Wie am Anfang dieses Kapitels erwähnt, kann man sich den Computermonitor bzw. das von WTK erzeugte Computerbild als ein Fenster zur virtuellen Welt vorstellen. Der Standpunkt (*Viewpoint*) des Betrachters kann dabei durch verschiedene Sensoren, wie z. B. der Maus oder durch einen Positions-/Orientierungssensor am Kopf des Betrachters befestigt (*Head Tracking*), bewegt werden. Die im Bildschirmfenster dargestellte Szene ist bestimmt durch die augenblickliche Sichtgeometrie. Abbildung 3.2 zeigt die Verhältnisse für einen einäugigen Betrachter (monoskopische Sicht).

Vom Standpunkt des Betrachters (`WTFRAME_VPOINT`) aus wird der sichtbare Bereich begrenzt durch den horizontalen Öffnungswinkel der Sichtpyramide sowie dem Seitenverhältnis des Fensters (`WINDOW`) auf dem Computermonitor. WTK stellt dabei nur die Objekte dar, die sich vor einer hinteren Ebene (*Yon Plane*) und hinter einer vorderen Ebene (*Hither Plane*) befinden. Diese beiden Ebenen stehen senkrecht auf der *z*-Achse des Standpunktkoordinatensystems und die Abstände zum Standpunkt des Betrachters können verändert werden. Man bezeichnet diese beiden Ebenen als *Clipping Planes*, die die Darstellung von zu weit bzw. zu nahe (oder sogar dahinter) beim Projektionszentrum liegenden grafischen Objekten ausschließen, um so Rechenzeit einzusparen.

Alle bisher beschriebenen Ausführungen zur Sichtgeometrie beziehen sich auf die monoskopische Projektion. Bei der stereoskopischen Projektion kommt eine

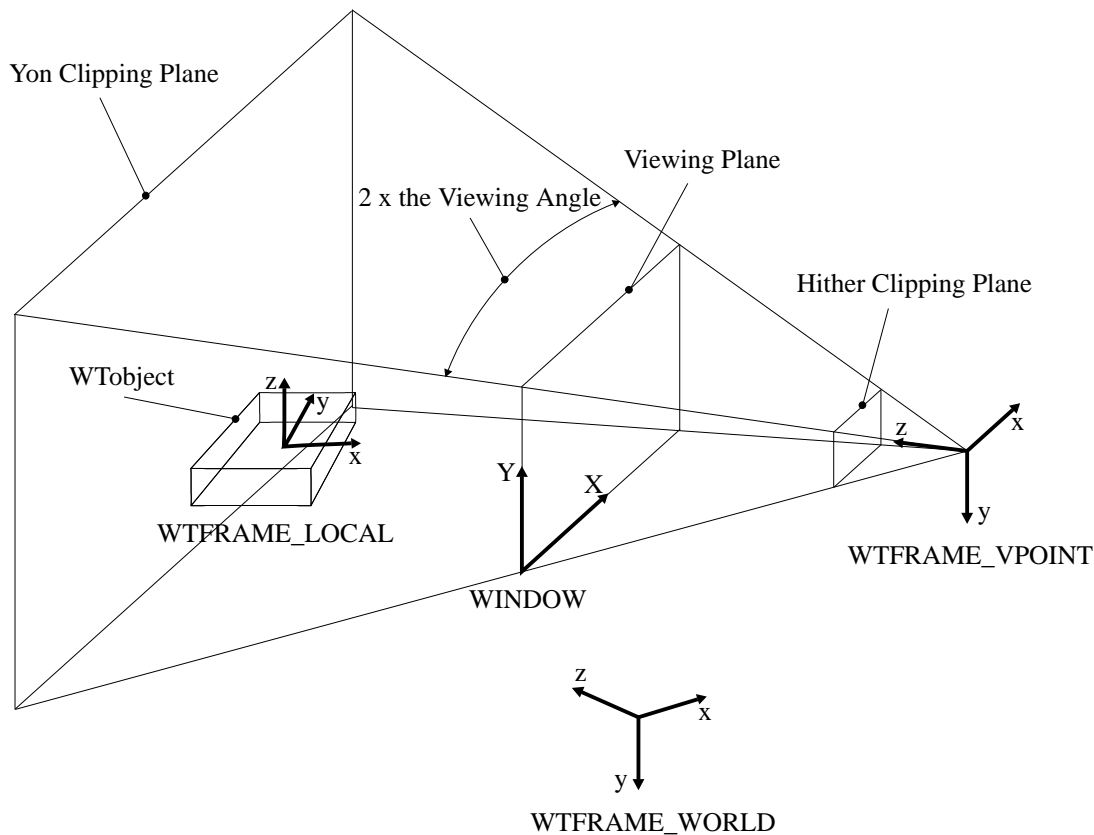


Abbildung 3.2: Monoskopische Sichtgeometrie und Bezugskoordinatensysteme

weitere Sichtpyramide für das zweite Betrachterauge hinzu. Die Sichtpyramiden besitzen einen gewissen Abstand und einen Konvergenzwinkel, ähnlich den stereoskopischen Sehverhältnissen beim menschlichen Auge. Werden dem Betrachter die beiden Sichtfenster getrennt für jedes Auge angeboten (z. B. durch einen HMD-Helm), so entsteht der für VR-Anwendungen kennzeichnende, dreidimensionale Eindruck. WTK bietet eine Reihe von Möglichkeiten der Stereoausgabe. Dazu zählen u. a. auch die Ausgabe des Stereobildes in einem Bildschirmfenster (WTDISPLAY_STEREOWINDOW bzw. WTDISPLAY_CRYSTALEYES), oder ein Rot/Blau-Stereobild (WTDISPLAY_RBSTEREO).

3.4 Koordinatensysteme, Position und Orientierung

WTK verwendet hauptsächlich drei Koordinatensysteme. Neben dem Weltkoordinatensystem (WTFRAME_WORLD) besitzt der Standpunkt des Betrachters ein Koordinatensystem (WTFRAME_VPOINT) sowie jedes Objekt ein lokales (WTFRAME_LO-

CAL). Bei allen Koordinatensystemen handelt es sich um Rechts-Systeme mit kartesischen Koordinaten. Ein weiteres Koordinatensystem, das seltener Anwendung findet, ist das Fensterkoordinatensystem. Wie in Abbildung 3.2 dargestellt handelt es sich um ein zweidimensionales Koordinatensystem dessen Ursprung in der linken unteren Ecke des sichtbaren WTK-Fensters auf dem Computerbildschirm ist, und dessen Einheit Bildschirmpixel sind.

Eine vollständige Beschreibung der Lage und Orientierung eines Objektes im virtuellen Raum wird in WTK in der Datenstruktur `WTpq` verwaltet. Diese setzt sich zusammen aus einem dreidimensionalen Feld für die Position `WTp3` und einem vierdimensionalen Feld für ein Quaternion `WTq`. Ein Quaternion ist eine effiziente Repräsentation für Orientierungen und Drehungen und kann als ein 3D-Vektor und eine Drehung um diesen Vektor angesehen werden. Nähere Informationen zur Mathematik der Quaternionen erhält man u. a. in [7], [12] und [10].

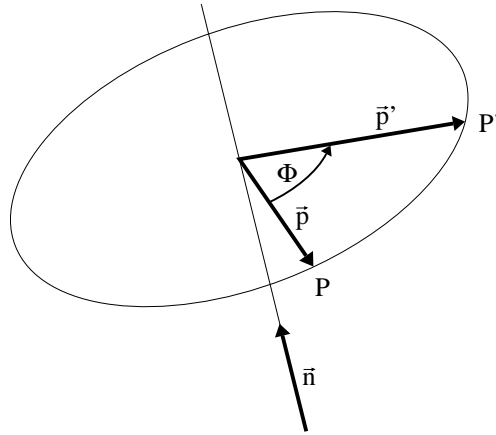


Abbildung 3.3: Rotation durch ein Quaternion

Wenn ein Quaternion \vec{q} einen Einheitsvektor (Länge = 1) darstellt, dann wird ein Vektor \vec{p} gemäß Abbildung 3.3 um die Drehachse $\vec{n} = [n_x \ n_y \ n_z]^T$ und den Winkel Φ in den Vektor \vec{p}' nach folgender mathematischer Beziehung gedreht:

$$\vec{q} = \cos\left(\frac{\Phi}{2}\right) + \sin\left(\frac{\Phi}{2}\right)\vec{n} \quad (3.1)$$

In WTK ist ein Quaternion \vec{Q} ein vierdimensionaler, normierter Vektor und wird in folgender Reihenfolge gespeichert:

$$\vec{Q} = \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ Q_w \end{pmatrix} = \begin{pmatrix} n_x \sin\left(\frac{\Phi}{2}\right) \\ n_y \sin\left(\frac{\Phi}{2}\right) \\ n_z \sin\left(\frac{\Phi}{2}\right) \\ \cos\left(\frac{\Phi}{2}\right) \end{pmatrix} \quad \text{mit} \quad |\vec{Q}| = 1 \quad (3.2)$$

Mit Hilfe von ebenfalls in der Bibliothek enthaltenen mathematischen Funktionen können Quaternionen z. B. in Eulerwinkel oder Drehmatrizen transformiert werden. Bei den meisten Funktionsaufrufen wird allerdings ein Quaternion für die Orientierung bzw. Drehung erwartet.

3.5 Klassen und Methoden

Obwohl die WTK-Bibliothek in der Programmiersprache C geschrieben wurde, verfolgt man ein objektorientiertes Konzept, soweit dies die prozedurale Programmiersprache zuläßt. Neben einer entsprechenden Namenskonvention werden zusammengehörende Daten in C-Strukturen (quasi den Klassen) verwaltet, die dem Benutzer verborgen sind und nur durch entsprechende Funktionen (Klassenmethoden) abrufbar bzw. veränderbar sind. Man möchte mit diesem Ansatz nicht nur solche Gesichtspunkte wie die Datenkapselung und das Verbergen der inneren Struktur (*Encapsulation* und *Information Hiding*) erreichen, sondern gleichzeitig die Übersichtlichkeit der Bibliothek erhöhen. Dem Präfix *WT* für WorldToolkit folgt der Name der Klasse gefolgt von einem Unterstrich und dem Namen der auf dieser Klasse angewendeten Methode.

Beispiel: `WObject_getposition` (Klasse `object`, Methode `getposition`)

Für alle Klassen gibt es dabei einen Objektkonstruktor (Methode `_new`) und einen Objektdestruktor (Methode `_delete`). Alle anderen Methoden einer Klasse haben als erstes Argument den Zeiger auf das Objekt, das die Methode empfangen soll.

3.5.1 Das Universum

Das Universum ist eine besondere Klasse in WTK, denn es gibt nur eine Instanz. Das Universum ist sozusagen der Behälter für alle generierten Klasseninstanzen, wie WTK-Objekte, Sensoren, Beleuchtungsobjekte, u.s.w. Es beinhaltet die Simulationsverwaltung (siehe Abbildung 3.1). Aus diesem Grund kann die Angabe eines Objekt-Zeigers im Argument von Universumsfunktionen entfallen. Im Konstruktor wird dabei festgelegt, wie die Ausgabe auf dem Monitor erfolgen soll (monoskopisch, stereoskopisch, 2 Fenster, ...). Außerdem wird ein Standard-Standpunkt-Objekt für die Darstellung erzeugt. Zusätzlich kann noch ein grafisches Modell angegeben werden, das das Universum grafisch repräsentiert. Es ist allerdings unbeweglich und immer ein sichtbarer Teil der Simulation.

3.5.2 Der Standpunkt

Der Standpunkt (*Viewpoint*) definiert die Position und Orientierung von dem aus das Universum auf den Computerbildschirm projiziert und wiedergegeben wird. Zusammen mit dem horizontalen Öffnungswinkel und dem Seitenverhältnis des Bildschirmfensters definiert er den sichtbaren Bereich der Simulation. Ein Standpunkt kann als eine Art Kameraposition angesehen werden, von der aus die virtuelle Welt betrachtet wird (siehe Abbildung 3.2). Normalerweise kommt eine WTK-Applikation mit einem Standpunkt-Objekt aus. Mehrere Objekte können aber dazu benutzt werden, um unterschiedliche Kamerapositionen leicht umzuschalten, bzw. unterschiedliche Kamerapositionen in mehreren Fenstern darzustellen.

3.5.3 Das Objekt

WTK-Objekte sind der eigentliche Gegenstand der Simulation. Sie repräsentieren grafisch der physischen Welt nachempfundene Objekte. In WTK können sie auf unterschiedliche Art und Weise erzeugt werden: Eine Möglichkeit besteht in der Verwendung von speziellen Funktionen zur Generierung von grafischen Grundobjekten, wie z. B. einem Zylinder einem Würfel oder einer Kugel. Eine andere Möglichkeit ist der Aufbau eines WTK-Objekts auf der untersten Ebene, nämlich der Eingabe von Eckpunkten eines Objekts, die dann zu Polygone miteinander verbunden werden. Neben der dynamischen Objekterzeugung kann man in WTK verschiedene 3D-Grafikformate lesen und schreiben. Unterstützt werden eine Reihe von Dateiformaten wie z. B. das Neutral File Format (*.NFF) oder das 3D Studio Format (*.3DS). Dazu existieren bereits mehrere kommerzielle bzw. frei verfügbare Programme (vom 3D-CAD-Programm bis hin zu speziellen Modellierprogrammen), mit denen dreidimensionale Objekte entworfen und in einem WTK bekannten Dateiformat abgespeichert werden können. WTK-Objekte sind beweglich und veränderbar und können, wie schon erwähnt, zur Steigerung der Geschwindigkeit aus der Simulationsverwaltung herausgenommen werden.

3.5.4 Der Sensor

Ein Sensor-Objekt erfasst in WTK Positions- und/oder Orientierungsdaten, die von einem Eingabegerät geliefert werden. Ein Sensorobjekt kann dabei z. B. an einen Standpunkt geknüpft werden (*Head Tracking*), oder an Objekte wie WTK-Objekte und Beleuchtungsobjekte. Die WTK-Bibliothek beinhaltet schon eine Reihe von Gerätetreibern. Dazu zählen u. a. die Positions-/Orientierungssensoren Polhemus FASTRAK oder Logitech RED BARON. Diese speziellen Treiber leiten

sich alle von einer Basisklasse `WtSensor` ab, die außerdem dazu benutzt werden kann, eigene Gerätetreiber für die Bibliothek zu schreiben.

3.5.5 Weitere Klassen

Neben den oben erwähnten Klassen gibt es in WTK noch einige andere, die ich hier noch kurz erwähnen möchte. Die Klasse `Polygon` (`WtPoly`) unterstützt das dynamische Erzeugen von WTK-Objekten. Terrains repräsentieren Landschaften. Oberflächen von WTK-Objekten können mit Texturen versehen werden. Zur Beleuchtung der virtuellen Welt wird die Beleuchtungsklasse eingesetzt. Eine Animationsklasse verhilft WTK-Objekten zu definierten Bewegungsabläufen. Die Klasse `Portale` beschreibt einen 3D-Hypertext-Link um z. B. virtuelle Türen zu öffnen und zu schliessen. Außerdem gibt es eine Klasse für das Bildschirmfenster, zur Arbeit in einem Netzwerk, zur Benutzung von seriellen Schnittstellen und zur Anzeige von 3D-Text.

Neben den bisher beschriebenen Klassenmethoden gibt es noch ein paar wenige Funktionen, die nicht dem Klassenkonzept folgen. Dazu gehören mathematische Funktionen und Funktionen zur Ausgabe von Fehlermeldungen.

Kapitel 4

Gestaltung der 3D-Benutzerschnittstelle

Ausgehend von den Anforderungen an die Bibliothek werden in diesem Kapitel der grundsätzliche Aufbau der Bibliothek sowie die Eigenschaften der einzelnen Interaktoren beschrieben. Zu Beginn dieser Arbeit war es notwendig einen Arbeitsnamen für die zu entwickelnde Bibliothek einzuführen. Da es sich bei der hier vorgestellten Arbeit um eine Bibliothek mit 3D-Interaktionselementen handelt, die auf dem VR-Paket WorldToolkit aufbaut, habe ich der Bibliothek den Arbeitsnamen *WTIE* (für WorldToolkit Interaction Elements) gegeben. Der Präfix *WTIE_* erscheint deshalb sowohl in den Klassen- als auch in den Dateinamen. Die Bibliothek hat den Namen *libWTIE.a* bekommen und die dazugehörige Headerdatei heißt *wtie.h*. Aufgrund der Eigenschaften von objektorientierten Programmiersprachen wie der Identität, der Klassifikation, dem Polymorphismus und der Vererbung wurde die Bibliothek in der Programmiersprache C++ implementiert.

4.1 Pflichtenheft

Bevor mit dem Entwurf der Interaktionselemente begonnen wurde, haben wir uns neben dem funktionalen Aspekt noch folgende Ziele gesetzt:

- Leichte Bedienbarkeit. Dem Anwender sollte sofort klar sein, wie der Interaktor zu bedienen ist. Aus diesem Grund habe ich mich immer wieder an den Prinzipien der 2D-Benutzerschnittstellen orientiert, da deren Handhabung schon sehr geläufig ist.
- Jeder Interaktor sollte mit den zur Verfügung stehenden Eingabegeräten bedient werden können (hybrides Interface). Dies schließt sowohl das 2D-

Eingabegerät Maus als auch den vielfältig einsetzbaren Datenhandschuh mit ein.

- Die Komplexität der Interaktoren sollte möglichst gering gehalten werden, um nicht unnötig Rechenzeit zu beanspruchen. Dies insbesondere in Hinblick auf das Anwendungsgebiet, nämlich der direkten Manipulation und Interaktion mit Visualisierungsdaten aus Simulationsrechnungen.

4.2 3D-Interaktoren

Unter einem Interaktor versteht man ein grafisches Element, das zur Interaktion mit der Applikation benutzt werden kann. Die Benutzung der Interaktoren erfolgt mit Eingabegeräten. Sowohl bei dem anschließend vorgestellten 3D-Button als auch beim folgenden 3D-Slider handelt es sich um Dialogelemente, die aus zweidimensionalen Benutzerschnittstellen bekannt sind. Die 3D-Eigenschaften der beiden Interaktionselemente bestehen im wesentlichen darin, daß sie frei im virtuellen Raum plaziert und orientiert werden können. Die Positionierung und Orientierung ist dabei relativ zum Betrachterstandpunkt und wird bei Veränderungen des Standpunkts beibehalten, d. h. die Interaktoren sind ortsfest zum Betrachterstandpunkt. Bisher erschien es nicht sinnvoll die Interaktoren ortsfest zum WTK-Weltkoordinatensystem zu halten, so daß man sich zu den Interaktoren hin bzw. von ihnen weg bewegen könnte. Die Elemente sollten immer im Sichtbereich des Benutzers liegen, weswegen die Angabe der Position und Orientierung zum Betrachterstandpunkt sinnvoll erschien. Oftmals werden die Interaktionselemente nur zur Interaktion in die Simulation eingeblendet und anschließend sofort wieder entfernt, um einen möglichst großen Ausschnitt der Simulation betrachten zu können.

Wie im Abschnitt Pflichtenheft beschrieben, wurde auch Wert darauf gelegt, die Interaktoren möglichst einfach zu halten. Einfach bezieht sich in diesem Zusammenhang in erster Linie auf die Komplexität der 3D-Objekte, d. h. je weniger Polygone ein Objekt enthält, um so weniger Rechenzeit wird für die Darstellung benötigt. WTK kann grafische Objekte mit Texturen versehen. In Verbindung mit der hier eingesetzten SGI RealityEngine kann damit eine sehr schnelle Grafikausgabe erreicht werden. Aus diesem Grund wurde auf das Erzeugen 'echter' 3D-Elemente zunächst verzichtet. Das Aussehen der Interaktoren kann aber jederzeit verändert werden, ohne daß davon der grundsätzliche Aufbau der Bibliothek betroffen wäre.

4.2.1 3D-Button

Buttons sind aus der 2D-Welt bekannte Dialogelemente, die nach dem Auslösen eine festlegbare Aktion anstoßen. Beim 3D-Button der *WTIE*-Bibliothek handelt es sich um zwei übereinander angeordnete Rahmen, die einen kleinen Abstand in Blickrichtung (Δz in *WTFRAME_VPOINT*) zueinander haben. Der vordere Rahmen trägt dabei die Button-Textur, der hintere Rahmen ist eingefärbt und gibt je nach Farbe den aktuellen Zustand des 3D-Buttons wieder.




Hintergrundfarbe		Buttonzustand
grün		normal
rot		hervorgehoben
blau		ausgewählt

Abbildung 4.1: Verschiedene Zustände beim 3D-Button

Abbildung 4.1 zeigt die verschiedenen möglichen Buttonzustände anhand eines Beispiels. Neben dem normalen Zustand, zeigt der hervorgehobene Zustand an, daß sich ein der *WTIE*-Bibliothek bekannter Sensor über dem Button befindet. Der 3D-Button befindet sich dann im Fokus des Sensors. Beim 2D-Sensor Maus muß dazu beispielsweise der Mauszeiger über die sichtbare Fläche des 3D-Buttons positioniert werden. In diesem Zustand löst dann das Drücken der mittleren Maustaste die dem 3D-Button zugeordnete Aktion aus. Obwohl es sich in der Praxis gezeigt hat, daß der ausgewählte Zustand nur sehr kurz sichtbar ist, kann dennoch eine Zustandsveränderung wahrgenommen werden, die das Auswählen visuell dokumentiert. Aufgrund der Anordnung des 3D-Buttons in zwei übereinander angeordnete Rahmen, ist es nicht sinnvoll, Buttons sehr schräg zum Betrachter anzuordnen. Momentan erscheint es mir auch nicht sinnvoll einen Button so anzuordnen, da man damit die Oberfläche sehr stark verkleinert. Es wäre umso schwieriger den Button auszuwählen und ihn gar als solchen zu erkennen. Das Ausrichten der Interaktoren aber auf die durch sie manipulierbaren grafischen Objekte assoziiert die bestehende Verbindung.

In vielen Fällen ist es notwendig, mehr als einen 3D-Button anzuzeigen. Zur Vereinfachung dieser Aufgabe wird die Klasse *WTIE_ButtonBar* eingeführt. Mehrere Buttons werden dabei untereinander (Parameter *WTIE_VERTICAL*) bzw. nebeneinander (Parameter *WTIE_HORIZONTAL*) angeordnet. Buttons können der Buttonleiste hinzugefügt und wieder entfernt werden. Die in der Buttonleiste enthaltenen Buttons können gemeinsam plaziert und orientiert werden, wobei sich die Angaben auf den ersten Button in der Liste beziehen. Die Bedienung einer Buttonlei-

ste unterscheidet sich nicht vom Auswählen aus einer Liste, weswegen auf einen Listen-Interaktor verzichtet wurde.

4.2.2 3D-Slider

Der 3D-Slider ist ebenfalls ein 2D-Element, das auch als Scrollbar bezeichnet wird. Einem mechanischen Schieber gleich können innerhalb eines Intervalls beliebige Werte angesteuert werden, die in der gegenwärtigen Implementierung aber diskrete Werte sind. Wie beim 3D-Button besteht der 3D-Slider aus Rahmen, in diesem Fall vier, die zum Teil (bis auf den Hauptrahmen im Hintergrund) mit Texturen überzogen sind. 3D-Slider können verschiedene Ausrichtungen besitzen und tragen eine Beschriftung, die den minimalen, den augenblicklichen und den maximalen Wert anzeigen. Abbildung 4.2 zeigt die Möglichkeiten für die Ausrichtung von 3D-Slidern.

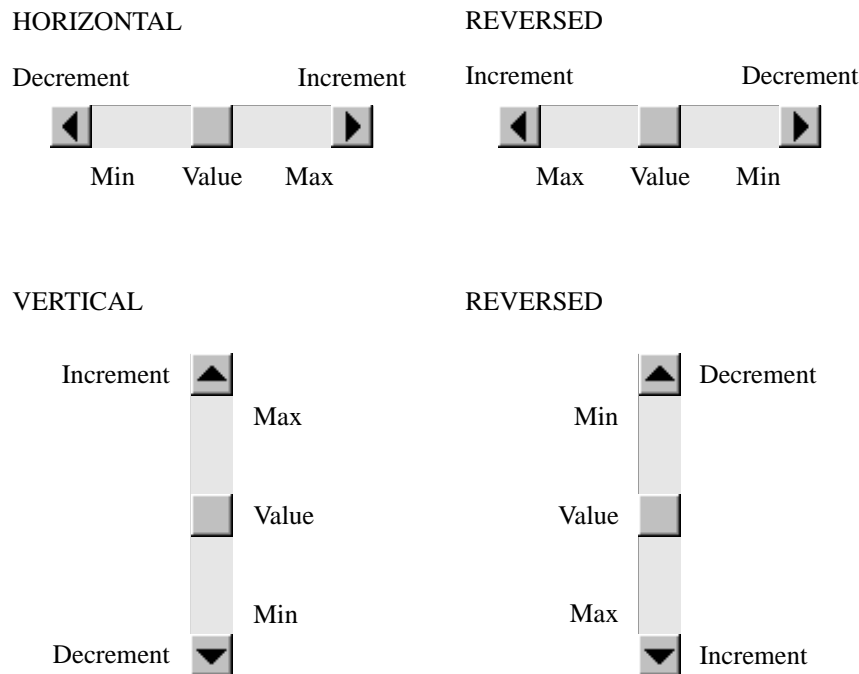


Abbildung 4.2: 3D-Slider-Ausrichtung

Wird der Inkrementierrahmen angewählt, so wird der Wert des 3D-Sliders um einen einstellbaren Betrag (`step_value`) erhöht, beim Dekrementierrahmen entsprechend erniedrigt. Wird der 3D-Slider zwischen dem Rahmen, der den im Augenblick eingestellten Wert darstellt, und dem Inkrementierrahmen bzw. dem Dekrementierrahmen angewählt, so wird der Wert des Sliders um einen i. d. R. höheren Wert (`big_step_value`) erhöht bzw. erniedrigt. Die Bedienung entspricht damit den Slidern aus 2D-Benutzerschnittstellen.

3D-Slider werden mit einem 3D-Text beschriftet, da WTK die Möglichkeit bietet 3D-Textobjekte zu erzeugen. Zur einfacheren Handhabung des 3D-Textes wurde die Klasse `WTIE_Text` implementiert. Bei der ersten Instanziierung wird eine im Lieferumfang von WTK befindliche Fontdatei `rcfont3d.nff` geladen, die für alle weiteren `WTIE_Text`-Objekte benutzt wird. Die Beschriftung des 3D-Sliders kann abhängig von seiner Ausrichtung unterschiedlich angeordnet werden. Abbildung 4.3 zeigt die möglichen Kombinationen.

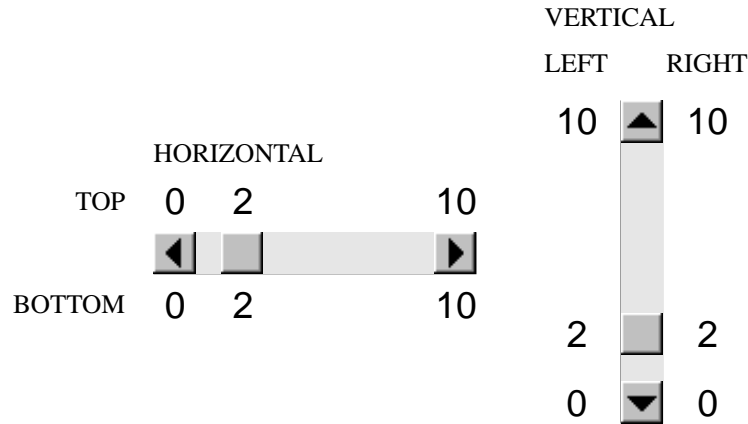


Abbildung 4.3: Anordnungsmöglichkeiten der 3D-Slider-Beschriftung

4.3 3D-Manipulator

Bei den Tests zur Implementierung der Bibliothek fiel auf, daß nicht alle Elemente mit den verschiedenen Eingabegeräten im gleichen Maße gut bedient werden können. So ist z. B. die Bedienung des 3D-Sliders mit der Maus sehr gut handhabbar, wohingegen der 3D-Slider mit dem Datenhandschuh wesentlich schwerer zu bedienen ist. Dies ist nicht weiter verwunderlich, wenn man sich die Vielzahl von Eingabegeräten und deren Unterschiede in der Art und Funktionsweise vor Augen hält. Aus diesem Grund wurde ein alternativer Ansatz zur Interaktion und Manipulation eingeführt, der in diesem Abschnitt beschrieben wird. Dieser Ansatz stellt eine zusätzliche Möglichkeit dar und kann mit allen Eingabegeräten erfolgen.

Der 3D-Manipulator ist weniger ein Element zur Interaktion als ein Konzept zur Manipulation von grafischen Objekten. Manipulierbare Objekte können mit den Eingabegeräten selektiert und anschließend manipuliert werden. Das Prinzip des Auswählens und Manipulierens ist schon bekannt aus den zur Zeit gebräuchlichen Computer-Applikationen, wie z. B. Zeichenprogrammen. Die Selektion erfolgt dabei äquivalent zum Anwählen von Interaktoren. Da die manipulierbaren Objekte

aber oft nur beschränkt manipulierbar sind, etwa die Translationsmöglichkeit entlang einer Achse, muß der Anwender auf diese Beschränkungen hingewiesen werden. Aus diesem Grund wird nach einer erfolgten Selektion eines manipulierbaren Objekts ein grafisches Manipulator-Objekt angezeigt, das die Möglichkeiten der Manipulation visuell dokumentiert. Abbildung 4.6 zeigt das grafische Modell des 3D-Manipulatorobjekts der *WTIE*-Bibliothek.

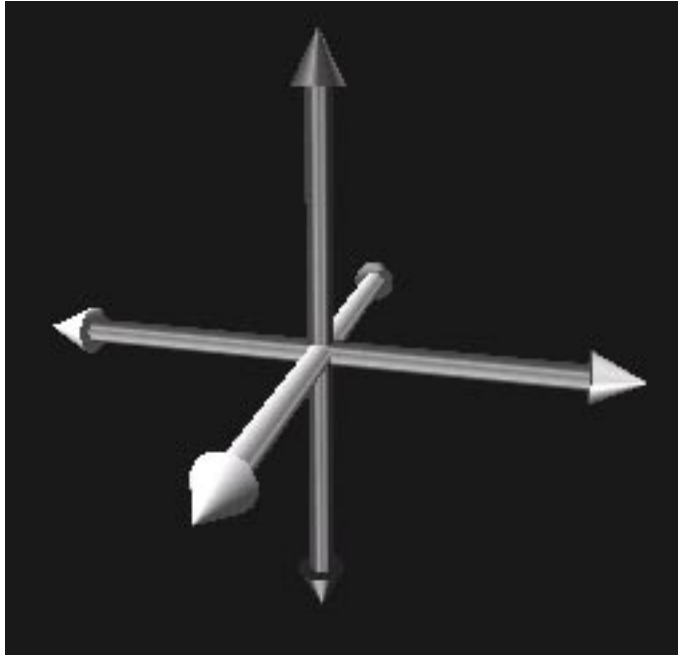


Abbildung 4.4: Modell des Manipulatorobjektes

Das Modell besteht aus drei Zylindern und sechs Kegeln. Ein Zylinder symbolisiert eine Rotation um diese Achse, ein Kegel eine Translation in Richtung der Kegelspitze. Die Visualisierung der Manipulationsmöglichkeiten erfolgt mit einer unterschiedlichen Einfärbung der Manipulatorelemente. Die Farbe Grün (helle Farbe in der Abbildung) bedeutet die Möglichkeit der Manipulation, die Farbe Rot (dunkle Farbe in der Abbildung) schließt diese aus. Für die Abbildung 4.6 ist also die Translation nach links und rechts, sowie in die Zeichenebene und aus ihr heraus, sowie die Rotation um die Zylinderachse, die in die Zeichenebene zeigt, möglich. Wird ein manipulierbares Objekt selektiert, so wird der Manipulator in der rechten oberen Ecke des Bildschirmfensters eingeblendet und entsprechend der Orientierung des selektierten Objekts ausgerichtet.

Ein manipulierbares Objekt kann ein einzelnes WTK-Objekt sein oder eine zusammengehörende WTK-Gruppe, dessen Wurzelobjekt als Referenz dient. Nach einer erfolgten Selektion wird um das Objekt oder die Gruppe ein Selektionsrahmen gezeichnet (siehe Abbildung 4.7). Die anschließenden Eingaben über die der *WTIE*-Bibliothek bekannten Eingabegeräte beziehen sich dann auf das se-

lektierte Objekt. Die Implementierung der Manipulationsmöglichkeiten erfolgt beim selektierbaren Objekt. Von der *WTIE*-Simulationsverwaltung kommt lediglich die Benachrichtigung in Form einer Aktion (Bsp. *WTIE_EVENT_MOVE*) und die Daten der Veränderung der Position und Orientierung des beteiligten Eingabegeräts. Selektierbare Objekte können eine Reihe von Beschränkungen unterworfen sein. Deshalb muß die Implementierung der Manipulationsmöglichkeiten in deren Umfeld erfolgen. Das Anzeigen des Manipulatorobjekts von der *WTIE*-Simulationsverwaltung soll lediglich für eine einheitliche Darstellung der Manipulationsmöglichkeiten sorgen.

4.4 Aufbau der Bibliothek

Wie schon am Anfang des Kapitels erwähnt, wurde die Bibliothek objektorientiert in der Programmiersprache C++ entworfen. Die zentrale Klasse der *WTIE*-Bibliothek ist die *WTIE_Manager*-Klasse. Sie verwaltet alle Interaktoren und regelt die Kommunikation mit den Eingabegeräten (Sensoren). Es stellt quasi eine Simulationsverwaltung für die *WTIE*-Bibliothek innerhalb der *WTK*-Simulationsverwaltung dar. Eingabegeräte, die für die Interaktion bzw. Manipulation benutzt werden, müssen eine Instanz der Klasse *WTIE_Sensor* bilden. Damit werden die verschiedenartigen Sensoren auf einen gemeinsamen Nenner gebracht und in die *WTIE*-Simulationsverwaltung aufgenommen. Auf diese Klassen wird später noch etwas genauer eingehen.

Die Bibliothek besitzt keine ausgeprägte Klassenhierarchie, da die meisten Klassen nur eine ganz spezielle Entität beschreiben. Dennoch konnte für grafische Objektklassen eine Basisklasse eingeführt werden. Abbildung A.3 zeigt mit Hilfe der OMT-Notation [11] die Beziehungen der Klassen untereinander. Weitere grafische Objektklassen können von der Basisklasse *WTIE_Object* abgeleitet werden. Die grafischen Objektklassen werden zur Erzeugung der Interaktoren (*WTIE_Slider* und *WTIE_Button*) verwendet. Von der *WTIE_Manager*-Klasse gibt es nur eine Instanz. Sie verwaltet Objekte der Klassen *WTIE_Button*, *WTIE_Slider*, *WTIE_Sensor* und *WTIE_SelObject*. Der *WTIE_Manager* besitzt eine *WTIE_Event*-Instanz zur Kommunikation. Die Klasse *WTIE_ButtonBar* beherbergt mehrere Objekte der Klasse *WTIE_Button*.

Zur Einbettung der *WTIE*-Simulationsverwaltung in die *WTK*-Simulationsverwaltung mußte die Reihenfolge der Simulationsschleife (Abbildung 3.1) verändert werden. Normalerweise kann *WTK*-Objekten eine sogenannte Task-Funktion zugeordnet werden. Diese Funktion wird bei jedem Simulationsschleifendurchlauf von der *WTK*-Simulationsverwaltung aufgerufen. Bei den hier implementierten Interaktionselementen ist die Task-Funktion für das Beibehalten der Position und Orientierung zum Standpunkt des Betrachters zuständig. Der *WTK*-Funktion

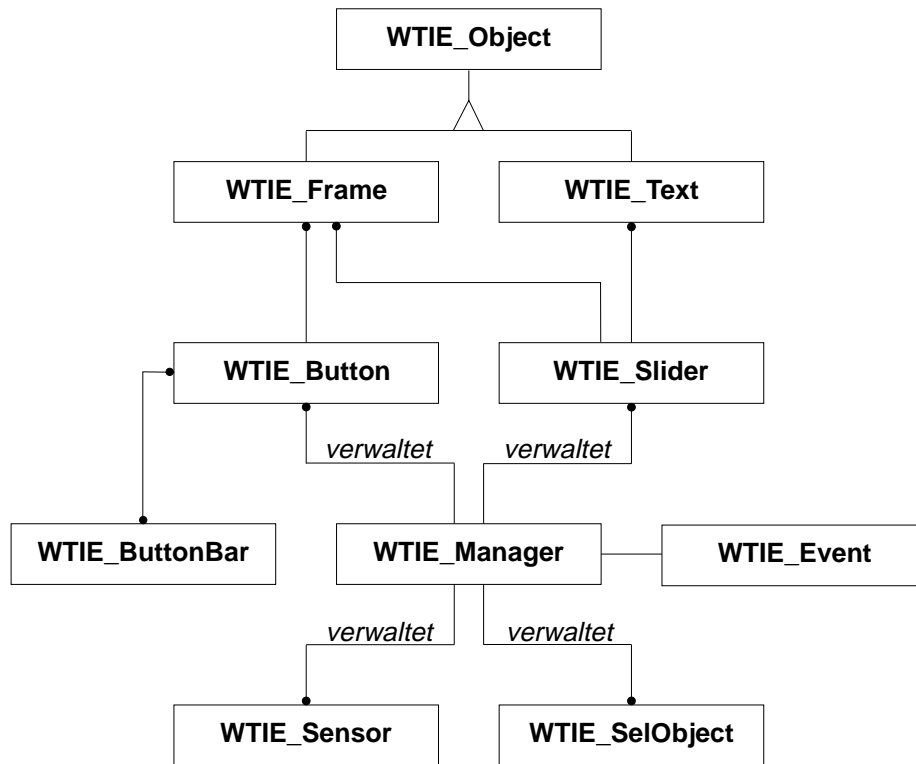


Abbildung 4.5: OMT-Diagramm der WTIE-Bibliothek

`WObject.settask()` wird ein Funktionszeiger mit einem `WObject`-Argument ohne Rückgabewert übergeben. Da die Bibliothek *WTIE* aber in C++ implementiert wurde und diese Task-Funktion im Kontext der Klasse `WTIE_Object` sein sollte, wurde die Task-Funktion zu einer Methode der Klasse `WTIE_Object` gemacht. Dies führte aber zu dem Problem, daß in C++ keine Funktionszeiger, die Teil einer Klasse sind, an eine C-Funktion (`WObject.settask()`) übergeben werden können. Um diesen Konflikt zu vermeiden, wird die Methode `task()` der Klasse `WTIE_Object` von der *WTIE*-Simulationsverwaltung aufgerufen. Zum korrekten Aufruf der Task-Funktionen muß die *WTIE*-Simulationsverwaltung in jedem WTK-Simulationsschleifendurchlauf benachrichtigt werden. Deshalb müssen *WTIE*-Anwendungen eine `WTK-user_action_function()` bereitstellen und als erste Anweisung die Methode `update()` der Klasseninstanz `wm` (`WTIE_Manager`) aufrufen. Zur Gewährleistung des korrekten Ablaufs der Aktualisierung der Interaktoren und Eingabegeräte muß noch die Reihenfolge der WTK-Simulationsschleife (Abbildung 3.1) entsprechend der Abbildung 4.6 abgeändert werden. Dies erfolgt automatisch bei der Initialisierung der Bibliothek.

Da die Methode `update()` der globalen Klasseninstanz `wm` (`WTIE_Manager`) in der `user_action_function()` aufgerufen wird, wurde die Reihenfolge der WTK-Simulationsschleife so abgeändert, daß diese Funktion erst nach dem Aktualisie-

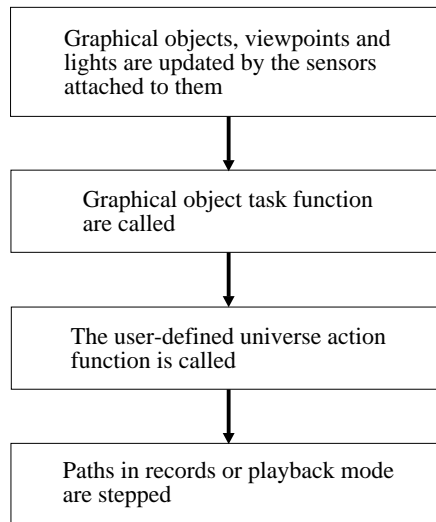


Abbildung 4.6: Geänderte WTK-Simulationsschleife

ren der Sensoren und der im Universum enthaltenen Objekte aufgerufen wird. Für den korrekten Programmablauf einer *WTIE*-Anwendung ist also die oben dargestellte Reihenfolge der WTK-Simulationsschleife zwingend erforderlich.

In den folgenden Abschnitten werden für die elementaren Klassen der *WTIE*-Bibliothek weitere grundsätzliche Anmerkungen gemacht. Sie fassen ebenfalls das zuvor schon Angesprochene noch einmal kurz zusammen.

4.4.1 **WTIE_Manager**

Der *WTIE_Manager* ist, wie schon erwähnt, eine Verwaltungsklasse innerhalb der WTK-Simulationsverwaltung. Es gibt wie beim WTK-Universum nur eine Instanz (`wm`), die automatisch beim Programmstart von der Bibliothek selbst angelegt wird. Über die Einbettung der *WTIE*-Simulationsverwaltung in die WTK-Simulationsverwaltung wurde in den vorhergehenden Abschnitten schon berichtet. Alle Interaktoren, Sensoren und selektierbare Objekt müssen dem *WTIE_Manager* bekannt gemacht werden. Für die Interaktoren und Sensorobjekte erfolgt dies automatisch bei der Instanziierung. Sensorobjekte werden zudem von der *WTIE_Application*-Klasse entsprechend einer Konfigurationsdatei angelegt, sofern diese Klasse verwendet wird. Die selektierbaren Objekte müssen vom Anwendungsprogrammierer bei Bedarf angemeldet werden. Dem *WTIE_Manager* kann eine Rückruffunktion von der Anwendung übergeben werden, um die vom *WTIE_Manager* erzeugten Aktionen (Events) auszuwerten. Der *WTIE_Manager* ist ebenfalls für den Selektionsvorgang, der Darstellung des Selektionsrahmens und dem Anzeigen des Manipulators verantwortlich.

4.4.2 WTIE_Event

Die Klasse `WTIE_Event` wurde zur Vereinheitlichung der Kommunikation zwischen der *WTIE*-Simulationsverwaltung und der VR-Applikation sowie den selektierbaren Objekten eingeführt. Sie bündelt die relevanten Daten einer von den Eingabegeräten ausgelösten Aktion und wird von der *WTIE*-Simulationsverwaltung generiert und zur Applikation bzw. den selektierbaren Objekten geleitet. Dort können dann anhand der Daten des Aktionsobjekts die entsprechenden Folgeaktionen ausgelöst werden.

4.4.3 WTIE_Sensor

Die Klasse `WTIE_Sensor` bringt die Vielzahl der Eingabegeräte auf einen gemeinsamen Nenner. Die spezifischen Möglichkeiten der Interaktion der einzelnen Eingabegeräte werden zu allgemeine Aktionen transformiert, die an den `WTIE_Manager` zur Weiterverarbeitung weitergeleitet werden. Von der *WTIE*-Simulationsverwaltung werden die Zustände der `WTIE_Sensor`-Objekte abgefragt, nachdem die Interaktoren aktualisiert wurden. Die einzelnen Sensoren werden dabei entsprechend der Reihenfolge der Anmeldung abgearbeitet.

Unterschieden wird allgemein zwischen einem 2D-Sensor (der Computermaus) und verschiedenen 3D-Sensoren (Datenhandschuh, 3D-Joystick, usw.). Wie schon im Abschnitt 3D-Button erwähnt, muß bei der Maus der Mauszeiger über die sichtbare Fläche des Interaktionselementes auf dem Bildschirmfenster plaziert werden, um dann durch Betätigen der Maustaste die zugeordnete Aktion auszulösen. Da die Computermaus in der Regel ebenfalls zum Verändern des Sichtpunkts benutzt wird (linke und rechte Maustaste), wird für die Interaktion mit den *WTIE*-Elementen die mittlere Maustaste benutzt. Ganz anders sind die Verhältnisse bei den 3D-Eingabegeräten. Ausgangspunkt des Auswahlvorgangs sind hier das Interaktionselement und die grafische Repräsentation des Eingabegerätes in der virtuellen Welt. Beide besitzen eine Position und Orientierung im Raum. Analog zur Computermaus muß mit dem Sensor auf das Interaktionselement gezeigt werden, und durch eine anschließende Sensoraktion die Auswahl erfolgen. Zur Vereinfachung des Zeigevorgangs wird bei einem 3D-Sensor daher ein zylindrischer Hilfsstrahl (Auswahlstrahl) verwendet, der entlang der Orientierung des Sensors von einem festgelegten Sensorpunkt aus emittiert. Als Beispiel sei hier der Datenhandschuh erwähnt. Durch eine bestimmte Auswahl-Handstellung wird vom Zeigefinger der grafischen Repräsentation des Datenhandschuhs aus ein Auswahlstrahl angezeigt (siehe Abbildung A.2). Dieser Auswahlstrahl muß dann durch Verdrehen der Hand mit dem jeweiligen Interaktionselement zum Schnitt gebracht werden. Schneidet sich Auswahlstrahl und Interaktionselement wird durch das Vollführen der Auswahlgeste auf dem Element die zugehörige

Elementaktion aufgerufen. Genauso gut wäre es denkbar gewesen, die grafische Repräsentation des Datenhandschuhs mit dem Interaktionselement zu schneiden, um anschließend die Auswahlgeste zu vollführen. Mit Hilfe des Auswahlstrahls erfolgt dieser Vorgang aber schneller, durch den geringen Durchmesser des Auswahlstrahls auch präziser und für alle 3D-Eingabegeräte unabhängig von ihrer grafischen Repräsentation gleich.

4.4.4 WTIE_Application

Die Klasse `WTIE_Application` dient zur einfacheren Erstellung von VR-Applikationen, die die *WTIE*-Bibliothek benutzen. Es bietet hauptsächlich die Möglichkeit entsprechend den Angaben in einer Gerätekonfigurationsdatei `device.config` Sensorobjekte anzulegen. Die *WTIE*-Anwendung muß dazu lediglich die Headerdatei `WTIE_Application.h` mit einbinden und mit der Objektdatei `WTIE_Application.o` und der Bibliothek `libWTIE.a` gelinkt werden. Eine *WTIE*-Anwendung bildet eine `WTIE_Application`-Instanz. Ähnlich den WTK-Funktionen für das Universum wird die Methode `start()` aufgerufen zur Vorbereitung der Simulation. Die Methode `go()` startet die Simulationsschleife und die Methode `stop()` verläßt sie wieder. Der Aufwand zur Erstellung einer minimalen *WTIE*-Anwendung wird durch die Verwendung der `WTIE_Application`-Klasse erheblich reduziert.

4.4.5 WTIE_SelObject

Die Klasse `WTIE_SelObject` beherbergt die Daten und Methoden, um aus einem einzelnen WTK-Objekt bzw. aus einer zusammengehörenden Gruppe von WTK-Objekten manipulierbare Objekte zu machen. Eine WTK-Gruppe ist eine Anordnung von WTK-Objekten, die hierarchisch miteinander verknüpft sind. Das Bezugsobjekt ist dabei das Wurzelobjekt der Gruppe. Durch den Aufruf der Methode `appendSelObject()` des `WTIE_Managers` werden eine `WTIE_SelObject`-Instanz vom `WTIE_Manager` angelegt und in die *WTIE*-Simulationsverwaltung aufgenommen. Das selektierbare Objekt kann dann von den *WTIE*-Eingabegeräten manipuliert werden. Als Parameter müssen dabei die Möglichkeiten der Manipulation mit angegeben werden (Kombination aus den symbolischen Konstanten für die Manipulation im Anhang A.2.1), anhand dessen die Einfärbung des Manipulators erfolgt.

4.5 Beispielanwendung

Anhand eines praxisorientierten Beispiels sollte die Eignung der Interaktoren getestet werden. Zur Darstellung und Berechnung der Schnittflächen in strukturierten Gittern sollten die zuvor entwickelten Interaktoren eingesetzt werden, um eine Rückkopplung in die Berechnung zu bekommen und die Schnittflächen an den mit den 3D-Eingabegeräten eingestellten Positionen darzustellen. Bei den hier verwendeten Daten handelt es sich um Daten auf einem nichtäquidistantem, orthogonalem Gitter und sie beschreiben einen einzelnen Zeitpunkt aus einer Strömungssimulation (nähere Informationen dazu findet man in [8] und [9]). Als Simulationsergebnisse stehen die Geschwindigkeitskomponenten und der Druck zur Verfügung. Abbildung 4.7 zeigt einen Bildschirmabzug der Beispielanwendung mit der Visualisierung der Geschwindigkeitskomponente in z-Richtung der Strömung.

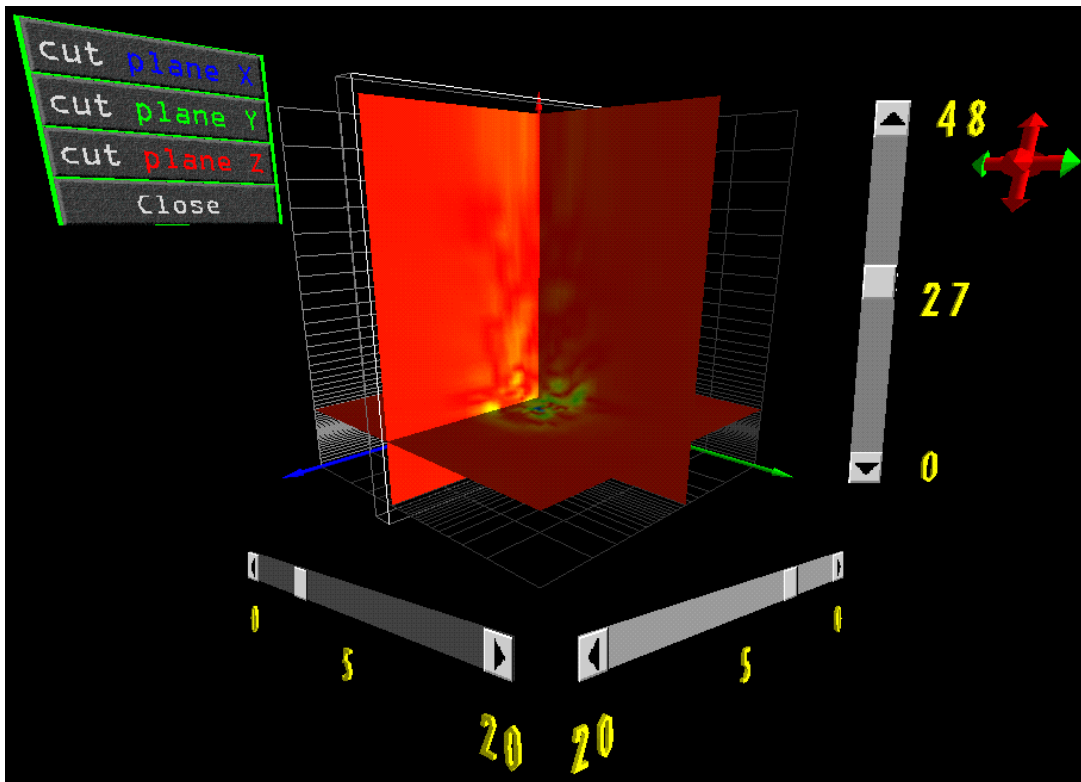


Abbildung 4.7: Beispielanwendung

Wie in der Abbildung zu sehen ist, werden sowohl der 3D-Button als auch der 3D-Slider zur Interaktion eingesetzt. Mit Hilfe der 3D-Buttons werden die Schnittflächen für die einzelnen Achsen ein- und ausgeblendet. Mit dem 3D-Slider werden die jeweiligen Schnittflächen entlang der Achse verschoben. Alle drei Schnittflächen sind sichtbar und eine davon ist selektiert, weshalb in der oberen rech-

ten Ecke der Manipulator eingeblendet ist, der die Translationsmöglichkeit der Schnittfläche entlang der Achse visualisiert. Im Hintergrund wird das nichtäquidistante, orthogonale Gitter sowie die Koordinatenachsen zur Orientierung im Datensatz gezeichnet.

Für die Darstellung des Datensatzes wurde die Klasse `WTplot3D` angelegt. Sie implementiert die drei Schnittflächen als selektierbare Objekte. Die Methode `handleEvent()` empfängt dabei die Aktionsobjekte von der *WTIE*-Simulationsverwaltung und implementiert die Translationsmöglichkeiten der Schnittflächen. Zusätzlich kann noch die zur Darstellung des Datensatzes benutzte Farbpalette angezeigt werden.

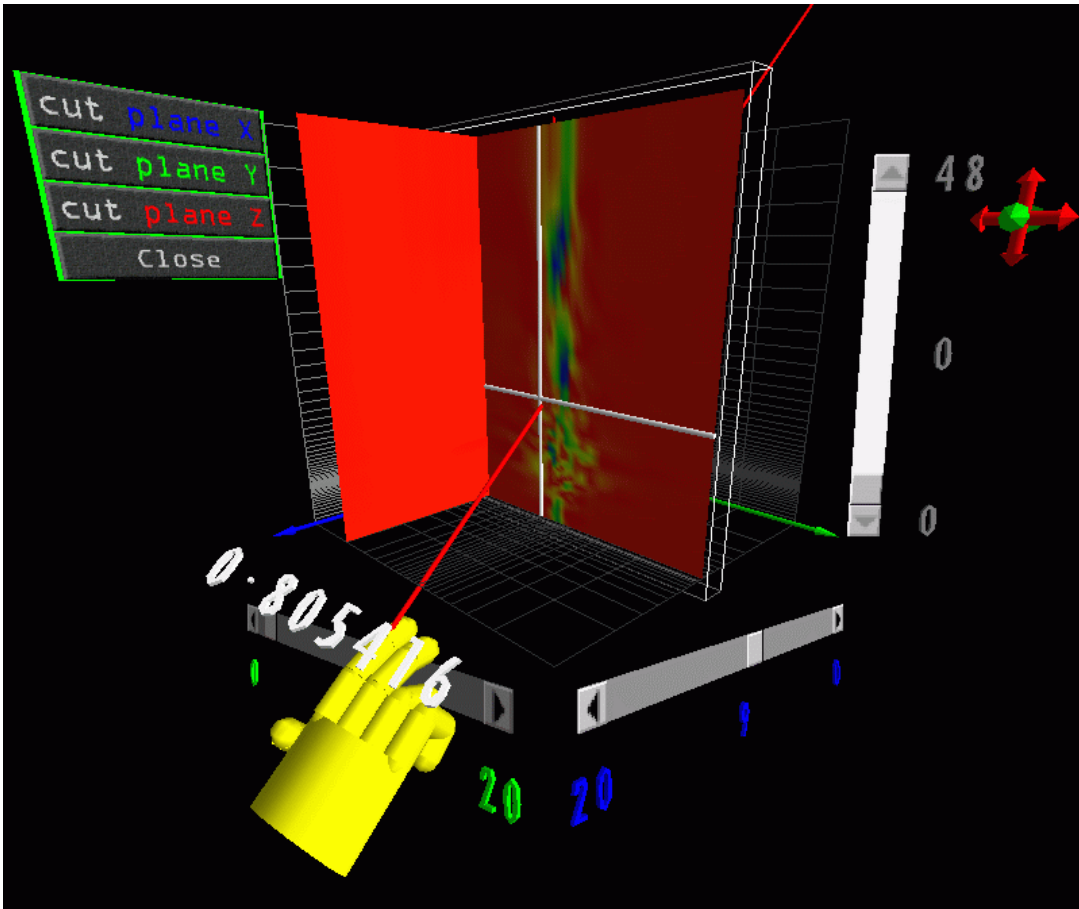


Abbildung 4.8: Datenhandschuh in Probing-Handstellung

Abbildung 4.8 zeigt den Datenhandschuh in einer benutzerdefinierten Handstellung. In der Beispielanwendung wird dieser Handstellung eine Probingfunktion zugeordnet. Beim Probing wird der Auswahlstrahl verwendet, um einen Skalarwert der selektierten Schnittfläche mit Hilfe eines 3D-Textobjektes zu visualisieren. Vom Schnittpunkt des Hilfsstrahls mit der selektierten Schnittfläche aus

wird der nächstgelegene Gitterpunkt bestimmt und durch einen Kreuzcursor angezeigt. Das Skalarwert-Textobjekt wird dabei relativ zum verwendeten Datenhandschuh positioniert. So kann unabhängig vom Sichtpunkt eine optimale Sicht auf die Schnittfläche sowie dem Textobjekt durch entsprechendes Positionieren des Datenhandschuhs erreicht werden.

Kapitel 5

Zusammenfassung

5.1 Probleme und Bewertung

Die aufgetretenen Probleme wurden teilweise schon im vorigen Kapitel angesprochen und sie seien an dieser Stelle noch einmal zusammengefaßt. An erster Stelle muß man die unterschiedliche Eignung der Interaktoren für die verschiedenen Eingabegeräte erwähnen. Während der 3D-Slider mit der Maus sehr leicht zu bedienen ist, ist die Bedienung mit dem Datenhandschuh zwar möglich, aber etwas schwerer. Aufgrund der unterschiedlichen Bauarten der Eingabegeräte und deren Möglichkeiten der Interaktion ist dies nicht zu vermeiden. Dieses Problem führte zum Manipulatorkonzept, das eine Alternative zur Interaktion mit den Standardelementen darstellt. Als Anwender kann man nun die einfachere Art der Manipulation für das jeweilige Eingabegerät auswählen. Das eingangs angesprochene Problem reduziert sich daher auf das geeignete Auswählen der Interaktionsmöglichkeit für ein bestimmtes Eingabegerät und einem bestimmten Interaktor. Eine Anwendung sollte deshalb beide Möglichkeiten bereitstellen. Zu einem Problem könnte auch die Festlegung der Reihenfolge der WTK-Simulationsschleife werden. Im Augenblick stellt dies allerdings kein Problem dar. Es wäre jedoch denkbar, daß es für bestimmte Anwendungen wünschenswert wäre, die Reihenfolge verändern zu können. Auf Geschwindigkeitstests wurde bisher verzichtet, weil keine Vergleichsdaten vorliegen. Entsprechende Messungen können lediglich Vergleichswerte für zukünftige Weiterentwicklungen liefern. Deutlich ist allerdings, daß ein zusätzliches Eingabegerät in der Simulationsschleife die Performance der Anwendung viel stärker reduziert, als zusätzlich angezeigte Interaktoren. Dies zeigt die weitaus geringere Bedeutung der Anzahl grafischer Elemente in der Simulation bei der Gesamtperformance gegenüber der Anzahl der eingesetzten Sensoren.

Insgesamt würde ich die hier vorgestellte 3D-User-Interface-Bibliothek als praxistauglich bewerten, trotz der zuvor beschriebenen Probleme. Die Möglichkeit Objekte direkt zu manipulieren und direkt mit der Anwendung zu interagieren steigert den Gebrauchswert der VR-Applikation erheblich. Die Fähigkeiten der hier vorgestellten Bibliothek erfüllen diese Aufgabe gut. Als ebenfalls sehr positiv hat sich das hybride Interface für die verschiedenen Eingabegeräte herausgestellt, das sowohl die Bedienung der Interaktoren an der Workstation als auch direkt vor der Projektionsleinwand zuläßt. Die Anwendung ist dadurch universeller einsetzbar und kann zur Demonstration ebenfalls auf Workstations ausgeführt werden, die nicht primär als VR-Workstations konzipiert sind. Die Bibliothek wurde so angelegt, daß das Einbinden neuer Interaktoren und Eingabegeräte keine große Schwierigkeit darstellt. Die Funktionalität und der Programmierkomfort der einzelnen Klassen der Bibliothek sind in vielen Fällen noch rudimentär. Im Vordergrund stand bei dieser Arbeit die grundlegende Entwicklung der Konzepte.

5.2 Entwicklungsstand

Ausgehend vom VR-Softwarepaket WorldToolKit und dem am Rechenzentrum vorhandenem VR-System wurden die Verfahren für ein 3D-User-Interface entwickelt und in einer Bibliothek implementiert. Als Basis für weitere Interaktoren beinhaltet die Bibliothek zwei 3D-Standardinteraktoren: der 3D-Button und der 3D-Slider. Ebenfalls implementiert wurde die Möglichkeit der Interaktion durch das Selektieren und anschließende Manipulieren von Objekten als Alternative zur Benutzung der Standard-Interaktionselemente. Als Eingabegeräte fanden die Computermaus und der Datenhandschuh Eingang in die Bibliothek. Der Treiber für den 3D-Joystick steht kurz vor der Vollendung. Als praxisorientierter Test wurde eine Anwendung entwickelt, die Schnittflächen in einem orthogonalem Gitter darstellt.

5.3 Ausblick

Zukünftig soll die Eignung der Interaktions- und Manipulationsmöglichkeiten der Bibliothek an der Beispielanwendung und anderen Anwendungen weiter getestet werden. Die dabei erzielten Erfahrungswerte können dazu dienen die Eigenschaften und Bedienung der einzelnen Interaktoren anzupassen. Die Verfeinerung des 'Handlings' der einzelnen Interaktoren ist nur durch ausgiebiges Testen möglich. Weitere elementare Interaktoren sind bereits angedacht. Die Veränderung und der Ausbau der einzelnen Klassen vollzieht sich fortlaufend. Weiterhin soll der Einsatz von Klangeffekten (3D-Audio) als zusätzliches Mittel zur Informationsrück-

kopplung den Benutzer beim Interagieren in der virtuellen Welt unterstützen. Die Möglichkeit mittels Spracheingabe mit Anwendungen zu interagieren wird in der Zukunft wohl eine sehr große Bedeutung erlangen. Die grundlegenden Voraussetzungen dazu könnten ebenfalls Eingang in die hier vorgestellte Bibliothek finden.

Anhang A

Programmierung

Über den grundsätzlichen Aufbau der Bibliothek wurde schon in Kapitel 4 eingegangen. In diesem Kapitel soll gezeigt werden, wie der WTK-Gerätetreiber für den 5th Dimension (5D) Datenhandschuh implementiert wurde, sowie die Integration dieses Eingabegerätes in die hier vorgestellte *WTIE*-Bibliothek. Im zweiten Teil dieses Kapitels werden die Programmierschnittstellen der *WTIE*-Bibliotheksklassen beschrieben. Sie dienen als Orientierung und Nachschlagewerk für Entwickler von VR-Anwendungen, die die *WTIE*-Bibliothek benutzen wollen.

A.1 WTK-Sensortreiber für den 5th Dimension-Datenhandschuh

A.1.1 Aufbau

Da der hier am Rechenzentrum verfügbare 5D-Datenhandschuh z. Zt. nicht zum Geträtetreiberumfang von WTK gehört, mußte ein WTK-Treiber geschrieben werden. Beim 5D-Datenhandschuh handelt es sich um ein relativ simples Gerät, das eigentlich nur für den Einsatz am PC vorgesehen ist. Deshalb mußten für den Betrieb an einer SGI-Workstation kleine Änderungen in der Verkabelung vorgenommen werden. Über eine serielle Schnittstelle werden die Sensordaten des 5D-Datenhandschuhs eingelesen. Es werden je ein Wert für die Krümmung jedes Fingers sowie zwei Daten für die Verdrehung des Handgelenks zum Unterarm eingelesen, die in der derzeitigen Implementierung allerdings nicht ausgewertet werden (das graphische Modell der Hand enthält kein Objekt für den Unterarm). Für jeden Finger erhält man dabei einen Wert zwischen 0 und 255 für einen geöffneten bzw. geschlossenen Finger. Für die Bestimmung der Position und Orientierung der Hand wird ein Polhemus FASTRAK-Sensor am Handgelenk des

Datenhandschuhs befestigt. Der Treiber für den 5D-Datenhandschuh wurde ähnlich angelegt, wie der im Lieferumfang von WTK enthaltene Treiber für den VPL Cyberglove. Es werden nur die Krümmungssensoren der Finger ausgewertet. Für die Positions- und Orientierungsbestimmung des Datenhandschuhs wird ein Polhemus FASTRAK-Sensor eingesetzt, für den schon ein Treiber in WTK existiert. Wie in Kapitel 4 schon kurz angedeutet werden eigene WTK-Gerätetreiber von der `WTsensor`-Klasse abgeleitet. Dem Konstruktor müssen dazu eine Funktion zum Öffnen, zum Schließen und zum Aktualisieren des Gerätes übergeben werden. Außerdem muß ein serielles Schnittstellenobjekt generiert werden, welches die Verbindung zum Eingabegerät definiert. Nachdem ein neues `WTsensor`-Objekt angelegt wurde, übernimmt die WTK-Simulation das Öffnen, Aktualisieren und Schließen des neuen Sensors, entsprechend der Reihenfolge der WTK-Simulationsschleife (s. Abbildung 3.1 bzw. Abbildung 4.5).

A.1.2 Grafisches Handmodell

Um die reale Hand mit dem Datenhandschuh in der virtuellen Welt sichtbar zu machen, kann ein einfaches grafisches Modell benutzt werden. Abbildung A.1 zeigt schematisch den Aufbau.

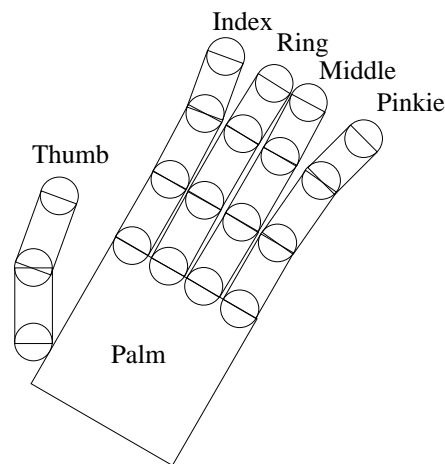


Abbildung A.1: Prinzipieller Aufbau des grafischen Handmodells

Das grafische Handmodell besteht aus dynamisch erzeugten Zylindern und Kugeln. Der Zylinder der Handfläche (*Palm*) wurde dabei zu einem Zylinder mit elliptischer Grundfläche gestreckt. Im Gegensatz zu den restlichen Fingern mit drei Gelenken, besitzt der Daumen (*Thumb*) nur zwei Gelenke. In jedem Simulationsschleifendurchlauf werden die Sensordaten ausgelesen und das grafische Modell entsprechend in der virtuellen Welt plaziert und orientiert. Die Krümmungswerte der Sensoren der Finger werden vom Gerätetreiber gleichmäßig auf die

Fingerelemente des grafischen Modells verteilt, linear zwischen einem geöffneten und geschlossenen Finger. Diese Vorgehensweise ist konstruktionsbedingt, da die Krümmungssensoren nur einen Wert pro Finger liefern.

A.1.3 Handstellung und Handgeste

Bis jetzt wird lediglich das grafische Modell der Hand entsprechend den Sensordaten in der virtuellen Welt aktualisiert. Zur Interaktion mit der VR-Applikation werden verschiedene Handstellungen und Handgesten herangezogen.

Posture ID ¹	Thumb	Index	Middle	Ring	Pinkie
FLAT	O	O	O	O	O
FIST	C	C	C	C	C
SINGLEPOINT	C	O	C	C	C
DOUBLEPOINT	C	O	O	C	C
TRIPLEPOINT	C	O	O	O	C
QUADPOINT	C	O	O	O	O
MIDDLEPOINT	C	C	O	C	C
FEATHERS	C	C	C	C	O
TWOFEATHERS	C	C	C	O	O
THREEFEATHERS	C	C	O	O	O
SINGLESHOOT	O	O	C	C	C
DOUBLESHOOT	O	O	O	C	C
TRIPLESHOOT	O	O	O	O	C
HORNS	C	O	C	C	O
OK	O	C	C	C	C
O ... OPENED, C ... CLOSED					

Tabelle A.1: Handstellungen

In jedem Durchlauf der Simulationsschleife wird eine Handstellung, je nach Krümmung der einzelnen Finger, identifiziert. Unterschieden wird im hier vorgestellten Gerätetreiber zwischen einer Handstellung und einer Handgeste, wobei letzteres nichts anderes ist, als das unmittelbare Aufeinanderfolgen zweier Handstellungen. Dieser Ansatz weicht von anderen mir bekannten Ansätzen ab, die lediglich die Stellung der Hand als Handgeste bezeichnen. Die Unterscheidung in Handstellung und Handgeste bietet eine größere Flexibilität in der Interpretation der Eingabe und eine größere Sicherheit gegenüber unbeabsichtigten Eingaben. Der

¹Aus Platzgründen sind die Haltungssymbole abgekürzt. Alle Haltungssymbole besitzen das Präfix WT5GLOVE_POSTURE_.

Einsatz von Handgesten als Auslöser für Aktionen (*Events*) erfordert vom Benutzer das bewußte Ausführen einer definierten Bewegung. Die Wahrscheinlichkeit einen definierten Bewegungsablauf zufällig auszuführen ist wesentlich geringer als die Wahrscheinlichkeit eine definierte Handstellung einzunehmen. Der hier vorgestellte Treiber identifiziert nur vollständig geöffnete bzw. geschlossene Finger zur Bildung der Handstellung und nicht teilweise geöffnete Finger, da das stark verrauschte Eingangssignal der Krümmungssensoren der Finger dies nicht zuläßt. Tabelle A.1 zeigt den derzeitigen Stand der identifizierbaren Handstellungen und die entsprechenden Stellungen der Finger.

Gesture ID ²	Previous Posture ID ¹	Current Posture ID ¹
DOWN	SINGLESHOOT	SINGLEPOINT
HOLD	SINGLEPOINT	SINGLEPOINT
UP	SINGLEPOINT	SINGLESHOOT
GRAB	SINGLEPOINT	FIST
MOVE	FIST	FIST
RELEASE	FIST	FLAT

Tabelle A.2: Handgesten

Tabelle A.2 zeigt einen Auszug von Handgesten. Wenn der Handstellung 1 unmittelbar die Handstellung 2 folgt, dann wird die in der ersten Spalte angeführte Handgeste identifiziert. Eine Handstellung könnte man vergleichen mit den physikalischen Zuständen anderer Eingabegeräte, wie z. B. Button gedrückt bei der Maus. Die Handgeste dagegen beschreibt eine Aktion. Beim Beispiel der Maus wäre das der Übergangsvorgang der Maus von einem nicht gedrückten in einen gedrückten Zustand.

A.1.4 Einbettung des Eingabegerätes in die WTIE-Bibliothek

Die Einbettung der 3D-Eingabegeräte in die *WTIE*-Bibliothek erfolgt in der allgemeinen Sensorklasse *WTIE_Sensor* (siehe Abschnitt 4.3.2). Wie schon erwähnt stellt diese Klasse eine Art Sammelklasse für alle Sensoren dar, die für die Interaktion mit der *WTIE*-Bibliothek geeignet sind. Zur Benutzung des 5D-Datenhandschuhs muß zunächst mit dem Sensorkonstruktor *WT5glove_new()* ein neues Sensorobjekt angelegt werden. Mit dem Befehl *WT5glove_usehandmodel()*

²Aus Platzgründen sind die Gestensymbole abgekürzt. Alle Gestensymbole besitzen das Präfix *WT5GLOVE_GESTURE_*.

wird für das Sensorobjekt ein grafisches Handmodell gemäß Abbildung A.1 erzeugt. Mit dem Befehl `WT5glove_usetracker()` wird dem grafischen Handmodell ein Positions-/Orientierungssensor zugeordnet. Bis hierhin kann der WTK-Sensortreiber für den 5D-Datenhandschuh unabhängig von der *WTIE*-Bibliothek benutzt werden. Das grafische Handmodell wird entsprechend den Sensordaten des Datenhandschuhs und des Tracking-Sensors aktualisiert. Um den Datenhandschuh in die *WTIE*-Verwaltung einzubinden, muß ein *WTIE_Sensor*-Objekt angelegt werden. Aus den identifizierten Handgesten werden während des Programmablaufs allgemeine Aktionen generiert, die zum *WTIE_Manager* weitergeleitet werden. Die Interpretation der Handgesten erfolgt also in der Klasse *WTIE_Sensor*. In ihr werden die unterschiedlichen Zustände und Aktionen der Sensoren auf eine gemeinsame Basis gebracht.

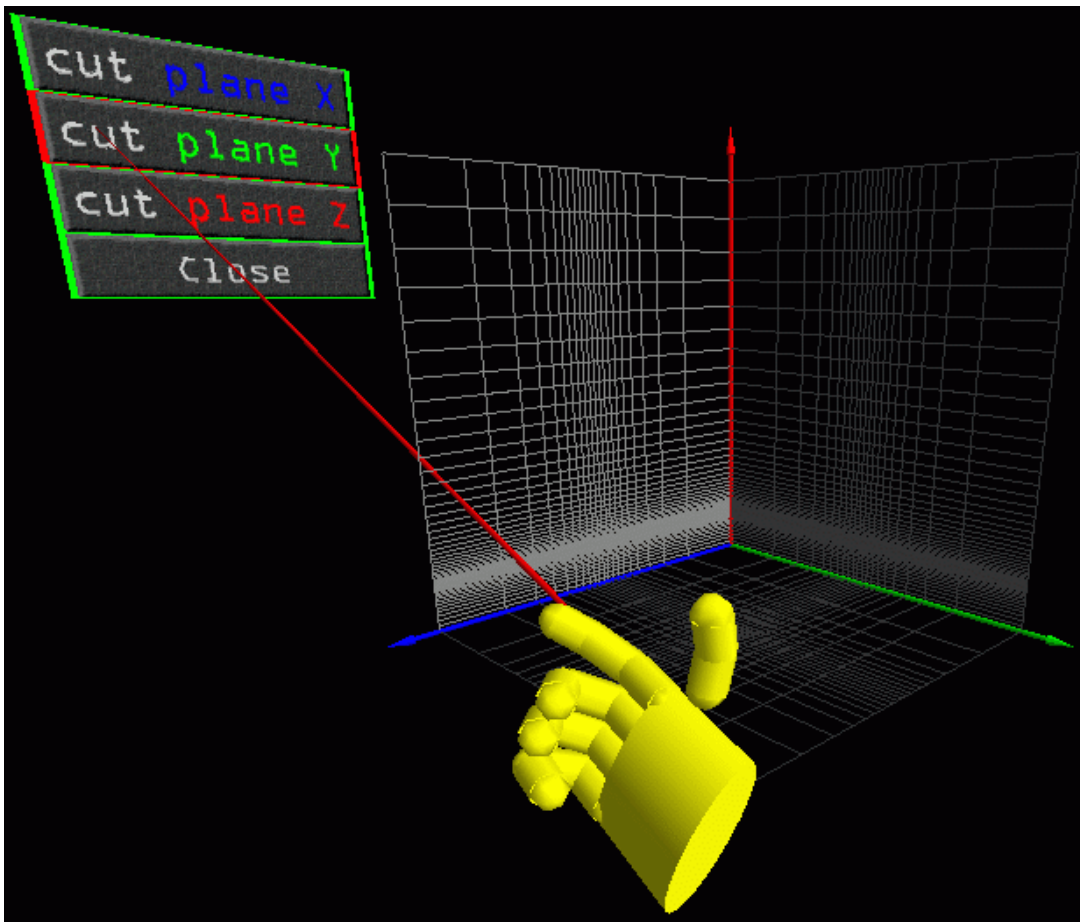


Abbildung A.2: Grafisches Handmodell in Auswahlhaltung

Als Beispiel für eine Handgeste und eine Handhaltung sei hier die Auswahlprozedur (Selektion) angeführt. Sobald die Handstellung `WT5GLOVE_POSTURE_SINGLE-SHOOT` identifiziert wird, erscheint ein grafischer Strahl aus dem Zeigefinger des

Handmodells. Der Auswahlstrahl besitzt die gleiche Orientierung wie die Handfläche. Schneidet der Auswahlstrahl selektierbare bzw. manipulierbare Elemente und wird danach die Handgeste `WT5GLOVE_GESTURE_DOWN` auf diesem Element ausgeführt (ohne mit dem Auswahlstrahl das Element zu verlassen), so wird das Element an- bzw. ausgewählt. Abbildung A.2 zeigt einen Bildschirmausdruck des grafischen Handmodells in der Handstellung `WT5GLOVE_POSTURE_SINGLESHOOT` und dem Auswahlstrahl. Da sich der Auswahlstrahl über einem 3D-Button befindet ist der Button visuell hervorgehoben. Der unmittelbare Übergang in die Handstellung `WT5GLOVE_POSTURE_SINGLEPOINT` ohne die Position und Orientierung zu verändern löst die dem 3D-Button zugeordnete Rückruffunktion aus. Das Anzeigen des Auswahlstrahls ist der Handstellung `WT5GLOVE_POSTURE_SINGLESHOOT` zugeordnet. Zum Auslösen von Aktionen wird dagegen die Handgeste herangezogen. Dieses Beispiel verdeutlicht die angesprochene größere Flexibilität durch den Einsatz von Handstellungen und Handgesten.

A.2 Programmierschnittstellen

In diesem Abschnitt werden die Schnittstellen der einzelnen Klassen der *WTIE*-Bibliothek beschrieben. Über den grundsätzlichen Aufbau wurde in Kapitel 4 schon eingegangen. Abbildung 4.4 verdeutlicht dabei beispielsweise die gegenseitigen Beziehungen der Klassen der *WTIE*-Bibliothek. Bei den im Anschluß angeführten Klassenmethoden handelt es sich ausschließlich um öffentliche (*Public*) Methoden, die vom Anwendungsprogrammierer aufgerufen werden können. Dieser Abschnitt ist also hauptsächlich für all diejenigen gedacht, die die Bibliothek verwenden möchten um VR-Anwendungen zu entwickeln und kann als Nachschlagewerk benutzt werden. Es muß aber darauf hingewiesen werden, daß sich die Beschreibung auf den Zeitpunkt der Erstellung dieses Dokuments bezieht. Da weiter an der Bibliothek gearbeitet wird, ändern sich die Schnittstellen laufend. Für diejenigen, die die *WTIE*-Bibliothek weiterentwickeln möchten, ist es sowieso unumgänglich sich mit den Quellen genauer zu befassen. Die einzelnen Klassen sowie die dazugehörigen Klassenmethoden sind in alphabetischer Reihenfolge angeordnet. Am Anfang jeder Klassenbeschreibung steht zur schnellen Orientierung eine Kurzübersicht mit den Methoden der Klasse. Daran schließen sich die ausführlichen Beschreibungen der Schnittstellen an.

Im weiteren Verlauf dieses Kapitels werden typografische Konventionen verwendet, um die Beschreibung der Schnittstellen übersichtlicher zu gestalten. Diese sind im einzelnen:

Bold Font	Kennzeichnet Methodennamen.
<i>Slanted Font</i>	Kennzeichnet Dateinamen, Querverweise und stehende Begriffe.
<i>Italic Font</i>	Kennzeichnet Datentypen.
Typewriter Font	Kennzeichnet Variablen-, Funktions- und Klassennamen.

A.2.1 Symbolische Konstanten

Um die Lesbarkeit des Quelltextes zu erhöhen, werden für einige Parameter symbolische Konstanten verwendet. Zur Vermeidung von Namenskonflikten wurde den Konstanten der Präfix *WTIE_* vorangestellt. Bei der Beschreibung der Schnittstellen wird ausführlich darauf hingewiesen, wenn für einen Parameter eine symbolische Konstante (oder eine Kombination daraus) erwartet wird. Nachfolgend sind auszugsweise häufig verwendete symbolische Konstanten aus verschiedenen Bereichen angeführt.

Aktionen

WTIE_EVENT_NULL
WTIE_EVENT_DOWN
WTIE_EVENT_HOLD
WTIE_EVENT_LONGHOLD
WTIE_EVENT_UP
WTIE_EVENT_SELECT
WTIE_EVENT_GRAB
WTIE_EVENT_MOVE
WTIE_EVENT_RELEASE
WTIE_EVENT_DESELECT

Fehler

WTIE_OK
WTIE_INFO
WTIE_WARNING
WTIE_ERROR
WTIE_FATAL

Manipulation

WTIE_TRANSLATE_X
WTIE_TRANSLATE_Y
WTIE_TRANSLATE_Z
WTIE_ROTATE_X
WTIE_ROTATE_Y
WTIE_ROTATE_Z
WTIE_SIZE_X
WTIE_SIZE_Y
WTIE_SIZE_Z

Orientierung und Platzierung

WTIE_HORIZONTAL
WTIE_VERTICAL

WTIE_LEFT
 WTIE_RIGHT
 WTIE_TOP
 WTIE_BOTTOM
 WTIE_FRONT
 WTIE_BACK
 WTIE_REVERSE

A.2.2 WTIE_Button

Die Klasse `WTIE_Button` implementiert einen einzelnen 3D-Button. Wird der Button ausgewählt wird eine zuweisbare Rückruffunktion aufgerufen. Für die Verwendung mehrerer 3D-Buttons sollte die Klasse `WTIE_ButtonBar` benutzt werden.

Kurzübersicht

<code>display</code>	Anzeigen bzw. Nichtanzeigen des Buttons.
<code>intersectsWith</code>	Prüft ob sich der Auswahlstrahl mit einem WTK-Objekt des Buttons schneidet.
<code>isButton</code>	Prüft, ob ein gegebenes WTK-Objekt ein WTK-Objekt des Buttons ist.
<code>isVisible</code>	Abfragen des Sichtbarkeitszustandes des Buttons.
<code>moveTo</code>	Setzt die Position und Orientierung des Buttons.
<code>setAction</code>	Setzt die Rückruffunktion des Buttons.
<code>setOrientation</code>	Setzt die Orientierung des Buttons.
<code>setPosition</code>	Setzt die Position des Buttons.
<code>translate</code>	Verschiebt den Button.

WTIE_Button::display

Beschreibung:	Anzeigen bzw. Nichtanzeigen des <code>WTIE_Button</code> -Objektes. Die grafischen WTK-Objekte werden dabei aus der Simulationsschleife herausgenommen bzw. wieder hinzugefügt.
Interface:	<code>void display(FLAG show)</code>
Parameter:	
<code>show</code>	Anzeigeoption [TRUE, FALSE].
Rückgabewert:	Keinen.

WTIE_Button::intersectsWith

Beschreibung: Prüft, ob sich der Auswahlstrahl mit einem WTK-Objekt des Buttons schneidet (*Intersection Test*).

Interface: *FLAG intersectsWith(WObject * ray)*

Parameter:
ray WTK-Objekt des Auswahlstrahls.

Rückgabewert:
TRUE Auswahlstrahl schneidet den Button.
FALSE Auswahlstrahl schneidet den Button nicht.

WTIE_Button::isButton

Beschreibung: Prüft, ob ein gegebenes WTK-Objekt ein WTK-Objekt des WTIE_Button-Objektes ist.

Interface: *FLAG isButton(WObject * object_to_look_for)*

Parameter:
object_to_look_for WTK-Objekt mit dem verglichen werden soll.

Rückgabewert:
TRUE Bei dem WTK-Objekt handelt es sich um ein WTK-Objekt des Buttons.
FALSE Bei dem WTK-Objekt handelt es sich nicht um ein WTK-Objekt des Buttons.

WTIE_Button::isVisible

Beschreibung: Gibt den Sichtbarkeitszustand des WTIE_Button-Objektes zurück.

Interface: *FLAG isVisible(void)*

Parameter: Keine.

Rückgabewert:
TRUE WTIE_Button-Objekt ist sichtbar.
FALSE WTIE_Button-Objekt ist nicht sichtbar.

WTIE_Button::moveTo

Beschreibung: Setzt die Position und Orientierung des WTIE_Button-Objektes in Bezug zu **WTFRAME_VPOINT**.

WTIE_Button::moveTo, Fortsetzung

Interface: *void moveTo(WTpq new_pq)*
Parameter:
new_pq Neue Position/Orientierung.
Rückgabewert: Keinen.

WTIE_Button::setAction

Beschreibung: Setzt die Rückruffunktion des WTIE_Button-Objektes.
Interface: *void setAction(PVFV new_action)*
Parameter:
new_action Neue Rückruffunktion.
Rückgabewert: Keinen.

WTIE_Button::setOrientation

Beschreibung: Setzt die Orientierung des WTIE_Button-Objektes in Bezug zu WTFRAME_VPOINT.
Interface: *void setOrientation(WTq new_q)*
Parameter:
new_q Neue Orientierung.
Rückgabewert: Keinen.

WTIE_Button::setPosition

Beschreibung: Setzt die Position des WTIE_Button-Objektes in Bezug zu WTFRAME_VPOINT.
Interface: *void setPosition(WTp3 new_p)*
Parameter:
new_p Neue Position.
Rückgabewert: Keinen.

WTIE_Button::translate

Beschreibung: Verschiebt das WTIE_Button-Objekt um den Vektor *dp3* in Bezug zum Referenzkoordinatensystem *frame*.
Interface: *void translate(WTp3 dp3, short frame)*

WTIE_Button::translate, Fortsetzung

Parameter:

`dp3` Verschiebungsvektor.
`frame` Bezugskoordinatensystem (`WTFRAME_WORLD`, `WTFRAME_VPOINT` oder `WTFRAME_LOCAL`).

Rückgabewert: Keinen.

WTIE_Button::WTIE_Button

Beschreibung: `WTIE_Button`-Konstruktor.

Interface: `void WTIE_Button(float new_x, float new_y, float new_z, float new_width, float new_height, PVFV new_action, char * new_texture = NULL)`

Parameter:

`new_x` x-Koordinate des Buttons.
`new_y` y-Koordinate des Buttons.
`new_z` z-Koordinate des Buttons.
`new_width` Breite des Buttons.
`new_height` Höhe des Buttons.
`new_action` Zeiger auf eine Rückruffunktion, die beim Auslösen des Buttons aufgerufen wird.
`new_texture` Dateiname der Buttontextur.

Rückgabewert: Keinen.

A.2.3 WTIE_ButtonBar

Die Klasse `WTIE_ButtonBar` ist für die Benutzung von mehreren `WTIE_Buttons` gedacht, da in der Regel mehr als ein Button benötigt werden. Es vereinfacht die Generierung und Verwaltung der Buttons.

Kurzübersicht

`appendButton` Fügt einen neu zu erstellenden Button der Buttonleiste hinzu.
`display` Anzeigen bzw. Nichtanzeigen der Buttonleiste. (Schnittstellenbeschreibung siehe *WTIE_Button::display*).
`isVisible` Abfragen des Sichtbarkeitszustandes der Buttonleiste (Schnittstellenbeschreibung siehe *WTIE_Button::isVisible*).

Kurzübersicht WTIE_ButtonBar, Fortsetzung

moveTo	Setzt die Position und Orientierung der Buttonleiste (Schnittstellenbeschreibung siehe <i>WTIE_Button::moveTo</i>).
removeButton	Entfernt einen Button aus der Buttonleiste und löscht die grafischen Objekte.
setOrientation	Setzt die Orientierung der Buttonleiste (Schnittstellenbeschreibung siehe <i>WTIE_Button::setOrientation</i>).
setPosition	Setzt die Position der Buttonleiste (Schnittstellenbeschreibung siehe <i>WTIE_Button::setPosition</i>).

WTIE_ButtonBar::appendButton

Beschreibung:	Fügt einen neuen Button der Buttonleiste hinzu.
Interface:	<i>WTIE_Button</i> appendButton(<i>PFVF</i> new_action, <i>char *</i> new_texture = NULL)
Parameter:	
new_action	Rückruffunktion des Buttons.
new_texture	Texturdatei des Buttons.
Rückgabewert:	Zeiger auf das <i>WTIE_Button</i> -Objekt.

WTIE_ButtonBar::removeButton

Beschreibung:	Entfernt einen Button aus der Buttonleiste.
Interface:	<i>WTIE_Button</i> removeButton(<i>WTIE_Button *</i> button_to_remove)
Parameter:	
button_to_remove	Button, der entfernt werden soll.
Rückgabewert:	Keinen.

WTIE_ButtonBar::WTIE_ButtonBar

Beschreibung:	<i>WTIE_ButtonBar</i> -Konstruktor.
Interface:	<i>WTIE_ButtonBar</i> (<i>WTpq</i> new_pq, <i>float</i> width, <i>float</i> height, <i>short</i> align = <i>WTIE_VERTICAL</i>)
Parameter:	
new_pq	Neue Position und Orientierung des ersten Buttons in der Buttonleiste.
width	Breite eines einzelnen Buttons.

WTIE_ButtonBar::WTIE_ButtonBar, Fortsetzung

<code>height</code>	Höhe eines einzelnen Buttons.
<code>align</code>	Anordnung der Buttonleiste [<code>WTIE_VERTICAL</code> , <code>WTIE_HORIZONTAL</code>].
Rückgabewert:	Keinen.

Beispiel:

Das folgende Beispiel erzeugt eine vertikale Buttonleiste, die senkrecht zur Betrachtungsrichtung orientiert und 50.0 Einheiten vor dem Standpunkt des Betrachters plaziert wird. Buttons der Buttonleiste sind 40.0 Einheiten breit und 10.0 Einheiten hoch. Anschließend werden der Buttonleiste zwei Buttons hinzugefügt. Der erste Button besitzt dabei die Rückruffunktion `open_button_action()` und trägt die Texturdatei `open_button.rgb`. Als erster Button in der Buttonleiste beziehen sich die Angaben der Position und Orientierung auf ihn. Der zweite Button (`close_button`) wird bei einer vertikalen Buttonleiste unterhalb des ersten angeordnet.

```
WTIE_ButtonBar *button_bar;
WTIE_Button *open_button, *close_button;
WTpq bb_pq;

WTpq_init( &bb_pq );
bb_pq.p[Z] = 50.0;
button_bar = new WTIE_ButtonBar( bb_pq, 40.0, 10.0, WTIE_VERTICAL );
open_button =
    button_bar->appendButton( open_button_action, "open_button" );
close_button =
    button_bar->appendButton( close_button_action, "close_button" );
```

A.2.4 WTIE_Event

Die Klasse `WTIE_Event` dient zur Kommunikation der `WTIE`-Simulationsverwaltung mit der Anwendung und den selektierbaren Objekten. Deshalb beschränken sich die Methoden auf das Auslesen der Aktionsobjektdaten.

Kurzübersicht

<code>getID</code>	Gibt die symbolische Aktionskonstante des Aktionsobjekts zurück.
<code>getObject</code>	Gibt das WTK-Objekt des Aktionsobjekts zurück.
<code>getOrientation</code>	Gibt die Orientierung des Aktionsobjekts zurück.
<code>getPosition</code>	Gibt die Position des Aktionsobjekts zurück.
<code>getSelectionRay</code>	Gibt das WTK-Objekt des Auswahlstrahls des Aktionsobjekts zurück.
<code>getSensor</code>	Gibt das Sensorobjekt des Aktionsobjekts zurück.

WTIE_Event::getID

Beschreibung:	Gibt die symbolische Aktionskonstante des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>long getID(void)</code>
Parameter:	Keine.
Rückgabewert:	Aktionskonstante des <code>WTIE_Event</code> -Objektes.

WTIE_Event::getObject

Beschreibung:	Gibt das WTK-Objekt des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>WObject * getObject(void)</code>
Parameter:	Keine.
Rückgabewert:	WTK-Objekt des <code>WTIE_Event</code> -Objektes.

WTIE_Event::getOrientation

Beschreibung:	Gibt die Orientierung oder eine Veränderung der Orientierung des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>void getOrientation(WTq rq)</code>
Parameter:	
<code>rq</code>	<code>WTq</code> -Zeiger, der die Orientierung empfangen soll.
Rückgabewert:	Keinen.

WTIE_Event::getPosition

Beschreibung:	Gibt die Position oder eine Veränderung der Position des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>void getPosition(WTp3 rp3)</code>
Parameter:	
<code>rp3</code>	<code>WTp3</code> -Zeiger, der die Position empfangen soll.
Rückgabewert:	Keinen.

WTIE_Event::getSelectionRay

Beschreibung:	Gibt das WTK-Objekt des Auswahlstrahls des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>WTKObject * getSelectionRay(void)</code>
Parameter:	Keine.
Rückgabewert:	WTK-Objekt des Auswahlstrahls des <code>WTIE_Event</code> -Objektes.

WTIE_Event::getSensor

Beschreibung:	Gibt das Sensorobjekt des <code>WTIE_Event</code> -Objektes zurück.
Interface:	<code>WTIE_Sensor * getSensor(void)</code>
Parameter:	Keine.
Rückgabewert:	Sensorobjekt des <code>WTIE_Event</code> -Objektes.

A.2.5 WTIE_Frame

Bei der Klasse `WTIE_Frame` handelt es sich um eine grafische Elementklasse, die die Daten und Methoden der Klasse `WTIE_Object` erbt. Es wird ein rechteckiges, zweidimensionales Rahmenelment erzeugt, das mit einer Textur überzogen werden kann.

Kurzübersicht

<code>intersectsWith- Mouse</code>	Prüft, ob sich der Rahmen mit dem virtuellen 3D-Strahl der Maus schneidet.
<code>intersectsWith- Ray</code>	Prüft, ob sich der Rahmen mit dem Auswahlstrahl schneidet.
<code>sizeTo</code>	Verändert die Größe des Rahmens.

Kurzübersicht WTIE_Frame, Fortsetzung

setTexture Setzt die Textur des Rahmens.

WTIE_Frame::intersectsWithMouse

Beschreibung: Prüft, ob sich der virtuelle Auswahlstrahl der Maus mit dem Rahmen schneidet (*Intersection Test*). Diese Methode ist eine Alternative zur WTK-Funktion `WTuniverse_pickobject()`. Ausgehend vom Standpunkt des Betrachters und den aktuellen Mauskoordinaten im Bildschirmfenster wird ein virtueller Auswahlstrahl berechnet, der mit dem Rahmen zum Schnitt gebracht wird.

Interface: *FLAG intersectsWithMouse(WTp2 * mpos, WTp2 ip)*

Parameter:

mpos Mausposition im WINDOW-Koordinatensystem.

ip 2D-Schnittpunkt in `WTFRAME_LOCAL` (für $z = 0.0$).

Rückgabewert:

TRUE Virtueller Auswahlstrahl schneidet den Rahmen.

FALSE Virtueller Auswahlstrahl schneidet den Rahmen nicht.

WTIE_Frame::intersectsWithRay

Beschreibung: Prüft, ob sich der Auswahlstrahl mit dem Rahmen schneidet (*Intersection Test*).

Interface: *FLAG intersectsWithRay(WTobject * ray, WTp2 ip)*

Parameter:

ray WTK-Objekt des Auswahlstrahls.

ip 2D-Schnittpunkt in `WTFRAME_LOCAL` (für $z = 0.0$).

Rückgabewert:

TRUE Auswahlstrahl schneidet den Rahmen.

FALSE Auswahlstrahl schneidet den Rahmen nicht.

WTIE_Frame::sizeTo

Beschreibung: Skaliert das `WTIE_Frame`-Objekt.

Interface: *void sizeTo(WTp3 new_extent)*

WTIE_Frame::sizeTo, Fortsetzung

Parameter:

new_extent Neue Größe des Rahmens.

Rückgabewert: Keinen.

WTIE_Frame::setTexture

Beschreibung: Setzt die Textur des **WTIE_Frame**-Objektes.

Interface: *short setTexture(char *new_texture)*

Parameter:

new_texture Dateiname der Textur.

Rückgabewert:

WTIE_WARNING Textur-Datei konnte nicht gefunden werden.

WTIE_OK Textur konnte gesetzt werden.

WTIE_Frame::WTIE_Frame

Beschreibung: **WTIE_Frame**-Konstruktor für einen senkrecht zur Betrachtungsrichtung angeordneten Rahmen.

Interface: *WTIE_Frame(float x = 0.0, float y = 0.0, float z = Z_DEFAULT, float width = 1.0, float height = 1.0, char *new_texture = NULL)*

Parameter:

x x-Koordinate des Rahmens (**WTFRAME_VPOINT**).

y y-Koordinate des Rahmens (**WTFRAME_VPOINT**).

z z-Koordinate des Rahmens (**WTFRAME_VPOINT**).

width Neue Breite des Rahmens.

height Neue Höhe des Rahmens.

new_texture Dateiname der Textur oder **NULL**.

Rückgabewert: Keinen.

WTIE_Frame::WTIE_Frame

Beschreibung: **WTIE_Frame**-Konstruktor.

Interface: *WTIE_Frame(*WTpq* new_pq, *WTp3* new_extent, char *new_texture = NULL)*

Parameter:

new_pq Neue Position und Orientierung des Rahmens.

WTIE_Frame::WTIE_Frame, Fortsetzung

<code>new_extent</code>	Ausmaße des Rahmens ($z = 0.0$).
<code>new_texture</code>	Dateiname der Textur oder NULL.
Rückgabewert:	Keinen.

A.2.6 WTIE_Manager

Die Klasse `WTIE_Manager` enthält die *WTIE*-Simulationsverwaltung. Sie regelt die Kommunikation der Sensoren mit den manipulierbaren Objekten.

Kurzübersicht

<code>appendSelObject</code>	Anlegen eines <code>WTIE_SelObject</code> -Objekts und Aufnahme in die <i>WTIE</i> -Simulationsverwaltung.
<code>deselectObject</code>	Hebt die Selektion auf.
<code>getSelectedObject</code>	Gibt den WTK-Objektzeiger des selektierten Objektes zurück.
<code>moveSelectionFrame</code>	Aktualisiert die Position und Orientierung des Auswahlrahmens auf das selektierte Objekt.
<code>removeSelObject</code>	Entfernen eines <code>WTIE_SelObject</code> -Objekts aus der <i>WTIE</i> -Simulationsverwaltung.
<code>setEventHandler</code>	Setzt die Rückruffunktion der <i>WTIE</i> -Simulationsverwaltung.
<code>setManipulatorOrientation</code>	Setzt die Orientierung des Manipulators auf die angegebene Orientierung.
<code>update</code>	Diese Methode sollte als erster Aufruf in der benutzerdefinierten <code>user_action_function()</code> der WTK-Anwendung stehen.

`WTIE_Manager::appendSelObject`

Beschreibung:	Anlegen eines <code>WTIE_SelObject</code> -Objekts und Aufnahme in die <i>WTIE</i> -Simulationsverwaltung.
Interface:	<code>void appendSelObject(WTKObject * object_to_append, int changeable_flags)</code>
Parameter:	
<code>object_to_append</code>	WTK-Objekt das selektierbar gemacht werden soll.
<code>changeable_flags</code>	Kombination aus den symbolischen Manipulationskonstanten.
Rückgabewert:	Keinen.

WTIE_Manager::deselectObject

Beschreibung: Hebt die Selektion auf.
Interface: *void* deselectObject(*void*)
Parameter: Keine.
Rückgabewert: Keinen.

WTIE_Manager::getSelectedObject

Beschreibung: Gibt das WTK-Objekt eines selektierten WTIE_SelObject-Objekts zurück.
Interface: *WObject *getSelectedObject(void)*
Parameter: Keine.
Rückgabewert: WTK-Objekt des selektierten Objekts.

WTIE_Manager::moveSelectionFrame

Beschreibung: Aktualisiert den Selektionsrahmen eines WTIE_SelObject-Objekts.
Interface: *void* moveSelectionFrame(*void*)
Parameter: Keine.
Rückgabewert: Keinen.

WTIE_Manager::removeSelObject

Beschreibung: Entfernen eines WTIE_SelObject-Objekts aus der WTIE-Simulationsverwaltung.
Interface: *void* removeSelObject(*WObject *object_to_remove*)
Parameter:
object_to_remove WTK-Objekt das entfernt werden soll.
Rückgabewert: Keinen.

WTIE_Manager::setEventHandler

Beschreibung: Setzt die Rückruffunktion der WTIE-Simulationsverwaltung.
Interface: *void* setEventHandler(*PWEHF new_event_handler*)

WTIE_Manager::setEventHandler, Fortsetzung

Parameter:

new_event_handler Rückruffunktion der Anwendung.

Rückgabewert: Keinen.

WTIE_Manager::setManipulatorOrientation

Beschreibung: Setzt die Manipulatororientierung. Diese Methode wird vom selektierten Objekt aufgerufen, nachdem es ein Aktionsobjekt mit der Konstante `WTIE_EVENT_SELECT` von der *WTIE*-Simulationsverwaltung empfangen hat.

Interface: *void setManipulatorOrientation(WTPq new_pq)*

Parameter:

new_pq Orientierung des selektierten Objekts.

Rückgabewert: Keinen.

WTIE_Manager::update

Beschreibung: Benachrichtigungsfunktion der *WTIE*-Simulationsverwaltung von der `user_action_function()` der Applikation.

Interface: *void update(void)*

Parameter: Keine.

Rückgabewert: Keinen.

A.2.7 WTIE_Object

Die Klasse `WTIE_Object` stellt die Basisklasse für alle grafischen Objektklassen der Bibliothek *WTIE* dar. Sie kapselt die WTK-Klasse `WtObject` und sollte nicht instanziiert sondern nur abgeleitet werden.

Kurzübersicht

display Anzeigen bzw. Nichtanzeigen des `WTIE_Object`-Objekts. (Schnittstellenbeschreibung siehe `WTIE_Button::display`).

getObject Liefert das WTK-Objekt des `WTIE_Object`-Objekts.

Kurzübersicht WTIE_Object, Fortsetzung

isVisible	Abfragen des Sichtbarkeitszustandes des WTIE_Object -Objekts. (Schnittstellenbeschreibung siehe <i>WTIE_Button::isVisible</i>).
moveTo	Setzt die Position und Orientierung des WTIE_Object -Objekts. (Schnittstellenbeschreibung siehe <i>WTIE_Button::moveTo</i>).
rotate	Rotiert das WTIE_Object -Objekt.
setColor	Setzt die Farbe des WTIE_Object -Objekts.
setOrientation	Setzt die Orientierung des WTIE_Object -Objekts. (Schnittstellenbeschreibung siehe <i>WTIE_Button::setOrientation</i>).
setPosition	Setzt die Position des WTIE_Object -Objekts. (Schnittstellenbeschreibung siehe <i>WTIE_Button::setPosition</i>).
translate	Verschiebt das WTIE_Object -Objekt. (Schnittstellenbeschreibung siehe <i>WTIE_Button::translate</i>).

WTIE_Object::getObject

Beschreibung:	Gibt den WObject -Pointer des WTIE -Objektes zurück.
Interface:	<i>WObject * getObject(void)</i>
Parameter:	Keine.
Rückgabewert:	WObject -Pointer des WTIE -Objektes.

WTIE_Object::rotate

Beschreibung:	Dreht das WTK -Objekt um die Achse axis und dem Winkel rad im Bezugskordinatensystem frame .
Interface:	<i>void rotate(short axis float rad, short frame)</i>
Parameter:	
axis	Achse [X, Y, Z].
rad	Winkel im Bogenmaß.
axis	Bezugskordinatensystem [WTFRAME_WORLD , WTFRAME_VPOINT , WTFRAME_LOCAL].
Rückgabewert:	Keinen.

WTIE_Object::setColor

Beschreibung:	Setzt die Farbe des WTIE -Objektes.
---------------	--

WTIE_Object::setColor, Fortsetzung

Interface: `void setColor(byte new_red = 255, byte new_green = 255, byte new_blue = 255)`

Parameter:

`new_red` Neuer Rotwert der Farbe.
`new_green` Neuer Grünwert der Farbe.
`new_blue` Neuer Blauwert der Farbe.

Rückgabewert: Keinen.

A.2.8 WTIE_SelectObject

Die Klasse `WTIE_SelectObject` (*Selectable Object*) implementiert die Methoden für auswählbare Objekte (s. Abschnitt 4.3.3). Ein auswählbares Objekt kann dabei aus einem einzelnen WTK-Objekt oder aus einer Gruppe von zusammengehörenden WTK-Objekten bestehen.

Kurzübersicht

<code>containsObject</code>	Prüft, ob ein gegebenes WTK-Objekt im <code>WTIE_SelectObject</code> -Objekt enthalten ist.
<code>getChangeableFlags</code>	Gibt die Möglichkeiten der Manipulation eines <code>WTIE_SelectObject</code> -Objekts zurück.
<code>getExtent</code>	Liefert die Ausmaße eines <code>WTIE_SelectObject</code> -Objekts.
<code>getPQ</code>	Gibt die Position und Orientierung des <code>WTIE_SelectObject</code> -Objekts zurück.
<code>getTopLevelObject</code>	Gibt das erste WTK-Objekt des <code>WTIE_SelectObject</code> -Objekts zurück.
<code>intersectsObject</code>	Prüft, ob sich ein gegebenes WTK-Objekt mit einem WTK-Objekt des <code>WTIE_SelectObject</code> -Objekts schneidet.
<code>isSingleObject</code>	Prüft, ob das <code>WTIE_SelectObject</code> -Objekt nur ein einzelnes WTK-Objekt enthält.

`WTIE_SelectObject::containsObject`

Beschreibung: Sucht nach einem WTK-Objekt im `WTIE_SelectObject`-Objekt.

Interface: `FLAG containsObject(WTKObject * object_to_look_for)`

Parameter:

WTIE_SelObject::containsObject, Fortsetzung

`object_to_look_` WTK-Objekt, nach dem gesucht werden soll.
`for`
Rückgabewert:
`TRUE` WTK-Objekt ist im `WTIE_SelObject`-Objekt enthalten.
`FALSE` WTK-Objekt ist nicht im `WTIE_SelObject`-Objekt enthalten.

WTIE_SelObject::getChangeableFlags

Beschreibung: Gibt die Möglichkeiten der Manipulation des `WTIE_SelObject`-Objekts zurück
Interface: *int* `getChangeableFlags(void)`
Parameter: Keine.
Rückgabewert: Möglichkeiten der Manipulation des `WTIE_SelObject`-Objekts.

WTIE_SelObject::getExtent

Beschreibung: Gibt die Ausmaße des `WTIE_SelObject`-Objekts zurück.
Interface: *void* `getExtent(WTp3 extent)`
Parameter: `extent` *WTp3*-Zeiger, der die Ausmaße empfangen soll.
Rückgabewert: Ausmaße des `WTIE_SelObject`-Objekts.

WTIE_SelObject::getPQ

Beschreibung: Gibt die Position und Orientierung des `WTIE_SelObject`-Objekts zurück
Interface: *WTpq* `getPQ(void)`
Parameter: Keine.
Rückgabewert: Position und Orientierung des `WTIE_SelObject`-Objekts.

WTIE_SelObject::getTopLevelObject

Beschreibung: Gibt das erste WTK-Objekt des `WTIE_SelObject`-Objekts zurück

WTIE_SelObject::getTopLevelObject, Fortsetzung

Interface: *WObject * getTopLevelObject(void)*

Parameter: Keine.

Rückgabewert: Erstes WTK-Objekt des *WTIE_SelObject*-Objekts.

WTIE_SelObject::isSingleObject

Beschreibung: Prüft, ob das *WTIE_SelObject*-Objekt ein einzelnes WTK-Objekt enthält.

Interface: *FLAG isSingleObject(void)*

Parameter: Keine.

Rückgabewert:

TRUE *WTIE_SelObject*-Objekt enthält einzelnes WTK-Objekt.

FALSE *WTIE_SelObject*-Objekt enthält eine WTK-Objektgruppe.

WTIE_SelObject::intersectsObject

Beschreibung: Prüft, ob ein WTK-Objekt die WTK-Objekte eines *WTIE_SelObject*-Objektes schneidet.

Interface: *FLAG intersectsObject(WObject * object_to_intersect)*

Parameter:

object_to_intersect WTK-Objekt, mit dem der Schnittest durchgeführt werden soll.

Rückgabewert:

TRUE WTK-Objekt schneidet mindestens eins der WTK-Objekte des *WTIE_SelObject*-Objekts.

FALSE WTK-Objekt schneidet keines der WTK-Objekte des *WTIE_SelObject*-Objekts.

WTIE_SelObject::WTIE_SelObject

Beschreibung: *WTIE_SelObject*-Konstruktor für ein einzelnes WTK-Objekt.

Interface: *void WTIE_SelObject(WObject * object, int new_changeable_flags)*

WTIE_SelObject::WTIE_SelObject, Fortsetzung

Parameter:

object WTK-Objekt, das veränderbar ist.
new_changeable_- Möglichkeiten der Veränderung.
flags
Rückgabewert: Keinen.

WTIE_SelObject::WTIE_SelObject

Beschreibung: WTIE_SelObject-Konstruktor für eine Gruppe von WTK-Objekten.

Interface: *void WTIE_SelObject(WObject ** objects, int new_num_of_objs, int new_changeable_flags, WTpq new_pq, WTp3 new_extent)*

Parameter:

objects Gruppe von WTK-Objekten, die veränderbar sind.
new_num_of_objs Anzahl der WTK-Objekte in der Gruppe.
new_changeable_- Möglichkeiten der Veränderung.
flags
new_pq Position und Orientierung der Gruppe.
new_extent Ausdehnung der WTK-Gruppe.
Rückgabewert: Keinen.

A.2.9 WTIE_Sensor

Die Klasse `WTIE_Sensor` ist die allgemeine Sammelklasse für alle Eingabegeräte, die mit der `WTIE`-Bibliothek interagieren können.

Kurzübersicht

getPQ Gibt die augenblickliche Position und Orientierung des Sensors zurück.
getPQChange Gibt die Veränderung der Position und Orientierung des Sensors gegenüber den Werten im vorherigen Schleifendurchgang zurück.

WTIE_Sensor::getPQ

Beschreibung: Liefert die augenblickliche Position und Orientierung des Sensors zurück.

<i>WTIE_Sensor::getPQ, Fortsetzung</i>	
Interface:	<i>WTpq getPQ(void)</i>
Parameter:	Keine.
Rückgabewert:	Augenblickliche Position und Orientierung des Sensors.

WTIE_Sensor::getPQChange	
Beschreibung:	Liefert die Veränderung der Position und Orientierung des Sensors gegenüber dem vorherigen Simulations-schleifendurchlauf zurück.
Interface:	<i>WTpq getPQChange(void)</i>
Parameter:	Keine.
Rückgabewert:	Veränderung der Position und Orientierung des Sensors.

WTIE_Sensor::WTIE_Sensor	
Beschreibung:	WTIE_Sensor-Konstruktor.
Interface:	<i>WTIE_Sensor(WTsensor *new_sensor, short new_sensor_type)</i>
Parameter:	
<i>new_sensor</i>	Zeiger auf ein WTK-Sensorobjekt.
<i>new_sensor_type</i>	Typ des Sensors. Zur Zeit zulässig sind MOUSE und WT5-JOYSTICK. Für den 5D-Datenhandschuh sollte der folgende Konstruktor benutzt werden.
Rückgabewert:	Keinen.

WTIE_Sensor::WTIE_Sensor	
Beschreibung:	WTIE_Sensor-Konstruktor für den 5D-Datenhandschuh.
Interface:	<i>WTIE_Sensor(WT5glove *new_glove)</i>
Parameter:	
<i>new_glove</i>	Zeiger auf ein WT5glove-Sensorobjekt.
Rückgabewert:	Keinen.

A.2.10 WTIE_Slider

Die Klasse `WTIE_Slider` implementiert einen 3D-Slider-Interaktor. Abbildung 4.2 zeigt die verschiedenen Anordnungsmöglichkeiten des 3D-Sliders und Abbildung 4.3 die zulässigen Möglichkeiten für die Beschriftung.

Kurzübersicht	
<code>display</code>	Anzeigen bzw. Nichtanzeigen des Sliders. (Schnittstellenbeschreibung siehe <code>WTIE_Button::display</code>).
<code>getValue</code>	Gibt den aktuellen Wert des Sliders zurück.
<code>intersectsWith</code>	Prüft, ob sich der Auswahlstrahl mit einem WTK-Objekt des Sliders schneidet. (Schnittstellenbeschreibung siehe <code>WTIE_Button::intersectsWith</code>).
<code>isSlider</code>	Prüft, ob ein gegebenes WTK-Objekt ein WTK-Objekt des Sliders ist. (Schnittstellenbeschreibung siehe <code>WTIE_Button::isButton</code>).
<code>isVisible</code>	Abfragen des Sichtbarkeitszustandes des Sliders. (Schnittstellenbeschreibung siehe <code>WTIE_Button::isVisible</code>).
<code>setBigStepValue</code>	Setzt den 'großen' Inkrementier-/Dekrementierwert des Sliders.
<code>setCallbackFunction</code>	Setzt die Rückruffunktion des <code>WTIE_Slider</code> -Objekts.
<code>setValue</code>	Setzt den aktuell angezeigten Wert des Sliders.
<code>setStepValue</code>	Setzt den 'kleinen' Inkrementier-/Dekrementierwert des Buttons.

WTIE_Slider::getValue	
Beschreibung:	Liefert den aktuellen Wert des <code>WTIE_Slider</code> -Objektes zurück.
Interface:	<code>int getValue(void)</code>
Parameter:	Keinen.
Rückgabewert:	Aktueller Wert des 3D-Sliders.

WTIE_Slider::setBigStepValue	
Beschreibung:	Setzt den 'großen' Inkrementier-/Dekrementierwert des <code>WTIE_Slider</code> -Objektes.
Interface:	<code>void setBigStepValue(int new_value)</code>

WTIE_Slider::setBigStepValue, Fortsetzung

Parameter:

new_value Neuer Wert des 'großen' Inkrementier-/Dekrementierwertes.

Rückgabewert: Keinen.

WTIE_Slider::setCallbackFunction

Beschreibung: Setzt die Rückruffunktion des WTIE_Slider-Objektes.

Interface: *void setCallbackFunction(PVFI new_callback_function)*

Parameter:

new_callback_function Neue Rückruffunktion.

Rückgabewert: Keinen.

WTIE_Slider::setStepValue

Beschreibung: Setzt den 'kleinen' Inkrementier-/Dekrementierwert des WTIE_Slider-Objektes.

Interface: *void setStepValue(int new_value)*

Parameter:

new_value Neuer Wert des 'kleinen' Inkrementier-/Dekrementierwertes.

Rückgabewert: Keinen.

WTIE_Slider::setValue

Beschreibung: Setzt den aktuell angezeigten Wert des WTIE_Slider-Objektes.

Interface: *void setValue(int new_value)*

Parameter:

new_value Neuer Wert des Sliders.

Rückgabewert: Keinen.

WTIE_Slider::WTIE_Slider

Beschreibung:	WTIE_Slider-Konstruktor.
Interface:	WTIE_Slider(<i>PVFI</i> callback_func, <i>int</i> val, <i>int</i> min_val, <i>int</i> max_val, <i>WTpq</i> new_pq, <i>float</i> width, <i>float</i> height, <i>int</i> style, <i>int</i> text_align, <i>FLAG</i> reverse = FALSE)
Parameter:	
callback_func	Rückruffunktion.
val	Startwert des Sliders.
min_val	Minimaler Wert des Sliders.
max_val	Maximaler Wert des Sliders.
new_pq	Position und Orientierung.
width	Breite des Sliders.
height	Höhe des Sliders.
style	Stil des Sliders.
text_align	Anordnung der Sliderbeschriftung.
reverse	Slider normal oder umgedreht anzeigen.
Rückgabewert:	Keinen.

Beispiel:

Das folgende Beispiel erzeugt einen horizontalen 3D-Slider, der 20.0 Einheiten vor dem Standpunkt des Betrachters plaziert und 15 Grad um die x-Achse (WTFRAME.VPOINT) gedreht ist. Der Slider ist 75.0 Einheiten breit und 5.0 Einheiten hoch. Dem Slider wird die Rückruffunktion `x_slider_callback()` zugewiesen. Die Sliderwerte laufen von 0 bis 20, beginnend bei 0. Die Beschriftung erfolgt unterhalb des Sliders.

```
void x_slider_callback( int new_value);

WTIE_Slider *x_slider = NULL;
WTpq spq;

WTp3_set( spq.p, 0.0, 0.0, 20.0 );
WTeuler_2q( WT2dr( 15.0 ), 0.0, 0.0, spq.q );
x_slider = new WTIE_Slider( x_slider_callback, 0, 0, 20, spq, 75.0,
                           5.0, WTIE_HORIZONTAL, WTIE_BOTTOM );
```

A.2.11 WTIE_Text

Die Klasse `WTIE_Text` dient zur Ausgabe eines einfachen Textstrings mit Hilfe eines im Lieferumfang von WTK enthaltenen 3D-Textfont. Das Verzeichnis mit der Datei `rcfont3d.nff` muß sich deshalb im WTK-Modellpfad befinden. Der einmal geladene Font wird für alle weiteren `WTIE_Text`-Objekte verwendet. Bei der Klasse `WTIE_Text` handelt es sich um eine grafische Elementklasse, die die Daten und Methoden der Klasse `WTIE_Object` erbt.

Kurzübersicht

<code>sizeToHeight</code>	Skaliert das <code>WTIE_Text</code> -Objekt proportional zu einer angegebenen Höhe.
---------------------------	---

`WTIE_Text::sizeToHeight`

Beschreibung:	Skaliert das Textobjekt proportional zu der angegebenen Höhe <code>new_height</code> .
Interface:	<code>void sizeToHeight(float new_height)</code>
Parameter:	
<code>new_height</code>	Höhe, auf die das Textobjekt skaliert werden soll.
Rückgabewert:	Keinen.

`WTIE_Text::WTIE_Text`

Beschreibung:	<code>WTIE_Text</code> -Konstruktor.
Interface:	<code>void WTIE_Text(WTpq new_pq, char * string)</code>
Parameter:	
<code>new_pq</code>	Position und Orientierung in <code>WTFRAME_VPOINT</code> .
<code>string</code>	Textstring, der angezeigt werden soll.
Rückgabewert:	Keinen.

Anhang B

Fehlermeldungen

B.1 Notation

In diesem Abschnitt werden die Fehlermeldungen beschrieben, die während eines Programmlaufs auftreten können. Unterschieden werden vier Kategorien von Fehlermeldungen: die Information, die Warnung, der Fehler und der fatalen Fehler. Tritt eine Fehlermeldung der Kategorie Fehler und fataler Fehler auf, wird das Programm beendet. Um die Ausführung des Programms nicht unnötigerweise zu verlangsamen, wurde auf die Ausgabe von Informationsmeldungen weitgehend verzichtet. Zusätzliche Informationsmeldungen können aber durch das Definieren der symbolischen Konstante `WTIE_DEBUG` in der Datei `WTIE.h` bei der Suche von Fehlern im Programm eingeschaltet werden.

Eine Fehlermeldung ist nach folgendem Schema aufgebaut:

*'WTIE Fehlerkategorie in module Dateiname method Klassenmethode:
Fehlernachricht'*

Beispiel:

WTIE ERROR in module WTIE_Application method setupSensors():
Failed to setup Spaceball device!

B.2 Häufiger auftretende Fehlermeldungen

Fehlermeldung	Failed to setup ... device!
Bemerkung	Die wohl am häufigsten auftretende Fehlermeldung ist das Scheitern der Initialisierung eines Eingabegerätes (s. Notationsbeispiel).
Ursache	Falsche Angabe der Schnittstellenummer in der Gerätekonfigurationsdatei <i>device.config</i> . Stromversorgung des Gerätes ist nicht eingeschaltet. Fehlende oder gestörte Kabelverbindung (Netz und Schnittstelle). Defektes Gerät.
Beseitigung	Schnittstellenummern in der Gerätekonfigurationsdatei <i>device.config</i> überprüfen. Stromversorgung und die Steckverbindungen der Kabel überprüfen. Speziell beim Polhemus FASTRAK kann es notwendig sein, das Gerät nur einmal kurz aus- und anzuschalten. Nur die FASTRAK-Empfänger einschalten (über DIP-Schalter an der Polhemus-Station) die auch benötigt werden. WTK akzeptiert nur Polhemus-Empfänger, die von der Einheit 1 aufsteigend verkabelt sind. Das Ansprechen der Einheiten 1 und 3 quittiert WTK mit einem Fehler bei der Initialisierung des Sensors (1 und 2 verwenden!).

Fehlermeldung	Unable to apply texture to the WTIE_Frame!
Bemerkung	Befehle, die auf die Umgebungsvariablen WTIMAGES und WTMODELS zurückgreifen.
Ursache	Falsche Angabe des Dateinamens. Verzeichnis mit den Texturdateien bzw. Modelldateien nicht in der Umgebungsvariable WTIMAGES bzw. WTMODELS.
Beseitigung	Dateinamen und korrekte Umgebungsvariablen (normalerweise in <i>.cshrc</i>) prüfen.

Fehlermeldung	Unable to load config file!
Bemerkung	Die Sensor-Konfigurationsdatei wird im gleichen Verzeichnis wie die ausführbare Datei erwartet.
Ursache	Falsche Angabe des Dateinamens. Konfigurationsdatei nicht im gleichen Verzeichnis wie die ausführbare Datei.
Beseitigung	Dateinamen prüfen. Konfigurationsdateien in selbes Verzeichnis wie die ausführbare Datei kopieren.

Anhang C

Hardware

An dieser Stelle stehen technische Daten der zur Verfügung stehenden Hardware. Sie sind nicht umfassend und sollen lediglich einen Eindruck über die verwendete Gerätekonfiguration geben.

Silicon Graphics Crimson

CPU	MIPS R4400
FPU	MIPS R4010
Data cache size	16 Kbytes
Instruction cache size	16 Kbytes
Secondary unified data/instruction cache size	1 Mbyte
Main memory size	256 MB
On board serial ports	4

Silicon Graphics RealityEngine ¹

8 GE
1 RM 4 board
Small pixel depth
10-bit RGB pixel
not using multi-channel option

Rückprojektionsleinwand

Höhe	2,20 m
Breite	4 m

¹Weitere Informationen zur SGI RealityEngine findet man in [1].

Polhemus FASTRAK, Polhemus LONG RANGER

Position Coverage	The system will provide the specified performance when the receivers are within 30" of the transmitter. Operation over a range of up to 10 feet is possible with slightly reduced performance.
Latency	4 milliseconds
Update Rate	120 updates/sec. number of receivers.
Interface	RS-232 with selectable baud rates up to 115.2 Kbaud (optional RS-422). IEEE-488 at up to 100 Kbytes/sec., ASCII or Binary format.
Static Accuracy	0.0" RMS for the X, Y, or Z position, 0.15 RMS for receiver orientation.
Resolution	0.0002"/" of transmitter and receiver separation, 0.025 orientation.
Range	Up to 10 feet with standard transmitter. Up to 30 feet with LONG RANGER transmitter.
Multiple Systems	Up to 8 systems can be frequency multiplexed with no change in update rate.

Literaturverzeichnis

- [1] **Akeley, K.** *RealityEngine Graphics*, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993) in Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pp. 109 - 116, 1993.
- [2] **Aukstakalnis, S.; Blatner, D.:** *Silicon Mirage: The Art & Science of Virtual Reality*, Peachpit Press, Berkeley USA, 1992.
- [3] **Bryson, S.; Feiner, S. K.:** *Virtual Reality for Visualization*, Visualization '94, Tutorial 6, Washington, D. C. USA, October 1994.
- [4] **Bryson, S.; Levit, C.:** *The virtual windtunnel: An environment for the exploration of threedimensional unsteady flows*, Proceedings of Visualization '91, pp. 17 - 24, 1991.
- [5] **Cruz-Neira, C.; Sandin, D. J.; DeFanti, T. A.:** *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993) in Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pp. 135 - 142, 1993.
- [6] **Earnshaw, R. A.; Gigante, M. A.; Jones, H.** *Virtual Reality Systems*, Academic Press Inc., San Diego USA, 1993.
- [7] **Foley, J. D.; van Dam, A.; Feiner, S. K.:** *Computer graphics: principles and practice*, Second Edition, Addison-Wesley Verlag, New York, 1992.
- [8] **Grosso, R; Lang, U.; Ryan, J.:** *CFD experiments in a distributed simulation and visualization system*, Proceedings 1993 European Simulation Symposium, October 1993, Delft, the Netherlands, 1993.
- [9] **Le, T. H.; Ryan, J.; Dang Tran, K.:** *Direct simulation of incompressible viscous flow through a rotating square channel*, 9th Gamm Conference, September 1991, Lausanne Switzerland, 1991.

- [10] **Maillot, P.-G.:** *Using Quaternions for Coding 3D Transformations*, Graphic Gems, A. S. Glassner, ed., Academic Press, pp. 498 - 515, 1990.
- [11] **Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorenson, W.:** *Objektorientiertes Modellieren und Entwerfen*, Carl Hanser Verlag München Wien und Prentice Hall International London, 1993.
- [12] **Shoemake, K.:** *Animating Rotation with Quaternion Curves*, ACM SIGGRAPH 1985, San Francisco, pp. 245 - 254, 1985.
- [13] **ACM SIGGRAPH:** *Practical 3D User Interface Design*, SIGGRAPH '95 Course Notes 23, 22nd International Conference on Computer Graphics and Interactive Techniques, Los Angeles, California USA, August 1995.
- [14] **Spektrum der Wissenschaft:** *Wahrnehmung und visuelles System*, 2. Auflage, Spektrum-der-Wissenschaft-Verlagsgesellschaft, Heidelberg, 1986.
- [15] **Wexelblat, A.:** *Virtual Reality Applications and Explorations*, Academic Press Inc., San Diego USA, 1993.
- [16] **Sense 8 Corp.:** *WorldToolkit Version 2.1 Reference Manual* und *SGI Hardware Guide*, Sense8 Corporation, 100 Shoreline Highway Suite 282, Mill Valley, CA 94941, USA.