

Entwicklung und Realisierung eines Prototyps zur Darstellung interaktiver 3D-Welten in räumlich immersiven Displayumgebungen

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

an der
Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Betreuer : Prof. Dr. Thomas Jung
2. Betreuer : Dipl.-Ing. Rolf Kretschmer

Eingereicht von Alex Habermann
Berlin, 12.11.2002

Inhaltsverzeichnis

1. Einleitung.....	6
2. Virtual Reality - Begriffserklärung und Analyse bestehender Systeme.....	7
2.1 Begriff und Entwicklung der VR.....	7
2.2 Eingabegeräte für VR-Welten.....	8
2.2.1 Tracking.....	8
2.2.2 Navigation.....	9
2.2.3 Sensorik.....	10
2.2.3.1 Boolesche Sensoren.....	10
2.2.3.1.1 Bewegungsmelder.....	11
2.2.3.1.2 Lichtschranke.....	11
2.2.3.1.3 Trittmatten.....	11
2.2.3.2 Werteliefernde Sensoren.....	12
2.2.3.2.1 Drucksensoren.....	12
2.2.3.2.2 Umweltsensoren.....	12
2.2.3.2.3 Abstandssensoren.....	12
2.3 Bestehende immersive VR-Systeme.....	13
2.3.1 CAVE.....	13
2.3.2 X-Rooms.....	14
2.3.3 CaveUT.....	15
2.4 Zuverlässigkeit und Wartbarkeit von Multimediasystemen.....	16
2.4.1 Komponentenwahl.....	16
2.4.2 Stromversorgung.....	16
2.4.3 Redundanz.....	17
3. Grundlegende Erklärungen zu den verwandten Technologien.....	18
3.1 Blender.....	18
3.2 SNMP.....	18
3.3 MIDI.....	19
3.4 Python.....	20
3.5 Java.....	20

4. Anforderungen an das zu entwickelnde System.....	21
4.1 Einsatzgebiete.....	21
4.2 Interaktion.....	22
4.3 Administration und Stabilität.....	22
4.4 Skalierbarkeit.....	22
4.5 Kosten.....	23
4.6 Inhalte.....	23
5. Design.....	24
5.1 Gesamtüberblick über Cube4.....	24
5.2 Beschreibung der virtuellen Welten.....	25
5.3 Kostenreduzierung durch Hardwarewahl.....	25
5.4 Echtzeit durch Blender.....	26
5.5 Beschreibung des Javatools Cube4J.....	27
5.6 Interaktionskonzepte für die 3D-Welten.....	27
5.6.1 Webcam.....	27
5.6.2 Abstandssensor.....	28
5.6.3 Trittmatten.....	28
5.7 Administration.....	29
5.7.1 Starten von Cube4.....	29
5.7.2 Filewechsel.....	30
5.8 Konzept zur Sicherung der Stabilität von Cube4.....	30
5.9 Bildgenerierung.....	31
5.9.1 Möglichkeiten der geometrischen Anordnung.....	31
5.9.2 Darstellung und Synchronisation der einzelnen Projektionsflächen.....	31
5.9.3 Objektkontrolle.....	32
5.9.4 Verteilung der erzeugten Interaktionsevents.....	32
5.10 Ausgabe.....	32
5.10.1 Erzeugung von MIDI-Daten durch Blender.....	32
5.10.2 Datenbankbindung.....	32
5.10.3 Steuerung von E-Motoren.....	32

6. Implementierung.....	34
6.1 Überblick über die verwendeten Technologien.....	34
6.1.1 Blender.....	34
6.1.1.1 Modellierungs- und Animationstool.....	34
6.1.1.2 Gameengine.....	34
6.1.1.2.1 Game Logik (Sensor, Controller, Actuator).....	35
6.1.1.2.2 Texturierung und Farbgebung.....	36
6.1.1.2.3 Dynamisches System.....	37
6.1.2 Python.....	37
6.1.2.1 Bildverarbeitung.....	37
6.1.2.2 Webcam.....	37
6.1.2.3 Netzwerkfunktionalität.....	38
6.1.2.4 Blenderbibliotheken.....	38
6.1.2.4.1 Das Modul Blender.....	38
6.1.2.4.2 Das Modul GameLogic.....	38
6.1.2.4.3 Das Modul Rasterizer.....	39
6.1.3 Java.....	39
6.1.3.1 serielle Schnittstelle.....	39
6.1.3.2 SNMP.....	39
6.1.3.3 Multifunktionalität.....	40
6.2 Eingabegeräte.....	40
6.2.1 Realisierung der Schnittstelle und Interaktion für Trittmatten.....	
(Boolesche Sensoren).....	40
6.2.2 Abstandsmessung.....	41
6.2.2.1 I ² C-Bus.....	41
6.2.2.1.1 Eigenschaften des I ² C-Buses.....	41
6.2.2.1.2 Aufbau von I ² C-Nachrichten.....	42
6.2.2.2 Beschreibung des verwendeten Sensormoduls.....	43
6.2.2.3 Interface seriell/I ² C-Bus.....	44
6.2.3 Webcam.....	44
6.3 Pythonskripte innerhalb Blenders zur Bildgenerierung.....	45
6.3.1 Synchronisation der Clients.....	45
6.3.2 Verteilung der Tastenevents.....	47
6.3.3 Objektkontrolle.....	49
6.4 Die Java-Anwendung Cube4J.....	49
6.4.1 GUI	49

6.4.2 Basisklassen.....	50
6.4.2.1 Rechnerinfo.....	50
6.4.2.2 Client.....	50
6.4.2.3 Server.....	50
6.4.2.4 Port.....	50
6.4.2.5 I ² C.....	50
6.4.2.6 SNMP.....	51
6.4.2.7 Midi.....	51
6.4.3 Starten von Cube4.....	51
6.4.4 Filewechsel.....	52
6.4.5 Softwareseitige Prozesskontrolle durch SNMP.....	52
6.4.6 Starten und Kalibrieren des Sensors.....	53
6.4.7 MIDI.....	54
6.5 Stabilität.....	54
6.5.1 Hardwareüberwachung durch die PCI-Karte Watchdog.....	54
6.5.2 Ersatz bei Ausfall.....	54
6.6 Administration.....	55
6.6.1 Automatischer Start.....	55
6.6.2 Filewechsel durch Blender.....	56
7. Test und Einsatzberichte.....	57
7.1 Objektkontrolle.....	57
7.2 Abstandssensor.....	57
7.3 Autostart.....	58
7.4 Webcam.....	58
7.5 Trittmatten.....	58
7.6 Watchdog.....	59
8. Resümee und Ausblick.....	60
Anhang A : Literaturverzeichnis.....	61
Anhang B : Informationsquellen im Internet.....	62
Anhang C : Abbildungsverzeichnis.....	63
Anhang D : Eigenständigkeitserklärung	

1. Einleitung

Die Echtzeitvisualisierung von dreidimensionalen Welten ist ein gutes und effektives Werkzeug zum Darstellen komplexer Sachverhalte oder noch nicht oder nicht mehr existierender Objekte.

Durch die raschen Entwicklungen auf dem Soft- und Hardwaremarkt sind heute Anwendungen möglich und somit für einen großen Kundenkreis interessant, die vor einigen Jahren Budgets im Millionenbereich erforderten.

Das Entwickeln von interaktiven Mediensystemen und das Erstellen und Konzeptionieren von Inhalten für VR-Umgebungen sind ein Standbein der Datenflug GmbH, bei der diese Diplomarbeit angefertigt wurde.

Ziel dieser Diplomarbeit war die Entwicklung eines Systems zur Echtzeitdarstellung interaktiver 3D-Welten in immersiven Displayumgebungen.

Hierbei sollte das System sehr flexibel und mobil einsetzbar und auch für die Zukunft gut erweiterbar sein.

2. Virtual Reality - Begriffserklärung und Analyse bestehender Systeme

2.1 Begriff und Entwicklung der VR

„Virtual Reality ist eine vom Computer geschaffene, interaktive, dreidimensionale Umwelt, in die eine Person eintaucht“ [Aukstakalnis 94]

Der Begriff der Virtuellen Realität (VR) wurde das erste Mal 1989 durch Jaron Lanier eingeführt, „um verschiedene Richtungen der 3D-Computertechnologie unter einem marktfähigen Label zusammenzufassen“ [Däßler 98].

Der Computer generiert Bilder bzw. Blicke auf eine 3D-Welt die nicht real, sondern lediglich virtuell existiert. Die Inhalte der Welt können äußerst unterschiedlich sein. Hierbei kann man aber zwei Hauptunterscheidungen treffen, die gleichzeitig auch Existenzberechtigungen der VR sind.

Es gibt zum einen den Versuch die Welt möglichst detailgetreu und realistisch abzubilden, wie dies in vielen Spielen, Architektur- oder Industrievisualisierungen geschieht, um entweder Entwicklungskosten für Produkte zu sparen, Eindrücke von erst in der Zukunft entstehenden Objekten zu vermitteln oder den User lediglich zu erfreuen.

Zum anderen gibt es den Blick auf komplizierte Sachverhalte, wobei durch verschiedene Blickwinkel und Abstraktionsstufen versucht wird, Zusammenhänge klarer und schneller zu begreifen. Hierzu gehört z.B. die Visualisierung von Produktionsabläufen oder Darstellung von Molekülen oder Kristallstrukturen.

Ein weiterer wichtiger Punkt der VR ist die Möglichkeit des Benutzers zur Interaktion. Dies bedeutet, der User kann durch sein reales Verhalten die virtuelle Welt beeinflussen und verändern. Dies kann zum einen in der Möglichkeit sich in der 3D-Welt zu bewegen bestehen, zum anderen aber auch im konkreten Verändern der Welt, z.B. durch Generieren oder Verändern von Objekten in der 3D-Szene.

Innerhalb der VR haben sich verschiedene Teilgebiete gebildet, die entweder durch technische Besonderheiten oder inhaltliche Ausrichtung gekennzeichnet sind.

Hierzu zählen wie in [Isdale 98] beschrieben unter anderem :

Desktop-VR-Systeme : Bei diesen Systemen wird ein konventioneller Monitor zur Darstellung der virtuellen Welt verwendet. Der Monitor fungiert als Fenster in diese Welt, weswegen diese Systeme auch WoW (Window on a World) genannt werden. Dieses Prinzip basiert auf der Veröffentlichung „The Ultimate Display“ von Ivan Sutherland.

Immersive Systeme : „Die weitestentwickelten VR Systeme verschieben den Sichtpunkt (Viewpoint) des Benutzers komplett in das Innere einer virtuellen Welt. Diese immersiven VR Systeme sind häufig mit einem Visualisierungshelm (Head Mounted Display) ausgerüstet. Eine Variation von immersiven Systemen stellt der CAVE dar. Ein CAVE ist ein Raum, an dessen Wänden virtuelle Bilder projiziert werden. Innerhalb dieses Raumes hat man das Gefühl sich in der realen Welt zu befinden.“ [Isdale 98]

Telepräsenz : Hierbei handelt es sich um eine Weiterführung, in der die Virtuelle Realität sozusagen als Medium dient.

Sensoren, die z.B. an einem Roboter angebracht sind, übermitteln ihre Daten. Diese Daten werden in einer 3D-Welt dargestellt. Nun kann ein Operator aufgrund dieser Darstellung Aktionen ausführen, die der Roboter interpretiert und ebenfalls ausführt. Einsatzgebiete für die Telepräsenz sind z.B. Operationen, wobei der operierende Arzt überall auf der Welt sein kann, oder Einsätze von Robotern in für den Menschen ungeeigneten Lebensräumen.

2.2 Eingabegeräte für VR-Welten

Die Interaktion mit den VR-Welten erfolgt über unterschiedliche Eingabegeräte.

Hierbei kann man im wesentlichen zwischen zwei Themengebieten unterscheiden : dem Tracking, d.h. der Positionsbestimmung des Users, um z.B. eine korrekte Stereodarstellung zu ermöglichen, und der Navigation in den 3D-Welten durch unterschiedliche Eingabegeräte.

2.2.1 Tracking

Tracking ist für die Stereodarstellung von großer Wichtigkeit, da dort die perspektivische Ansicht auf die 3D-Welt vom Blickpunkt und somit der Position des Betrachters abhängt. Die Stereoskopie ist kein Bestandteil des praktischen Teils dieser Arbeit, weswegen das Thema hier nicht erschöpfend behandelt wird.

Es gibt mehrere Methoden, die genaue Position des Users im Raum zu bestimmen : Ultraschall, Inertial Tracking, Mechanik, Magnetismus und Bildverarbeitung.

Ultraschall : Der User bekommt ein Ultraschallgerät, welches mit jeweils drei Lautsprechern und Mikrofonen ausgestattet ist. Das Gerät ist zusätzlich mit einer elektronischen Einheit versehen, die die Daten der Mikrophone aufnimmt und daraus Positions- und Orientierungsdaten berechnet. Diese werden anschließend zum Rechner gesendet.

Inertial Tracking (Trägheit) : Beim Inertial Tracking wird die relative Veränderung der Position und Orientierung des Betrachters im Raum bestimmt. Diese Veränderung wird mit Hilfe eines Trägheitssensors, der Beschleunigung und Winkelgeschwindigkeit bestimmt, ermittelt.

Durch diese zwei Größen und die absolute Startposition des Betrachters kann die aktuelle Position berechnet werden.

Mechanik : Beim mechanischen Tracking wird das zu trackende Objekt mit einer mechanischen, mit Gelenken versehenen Konstruktion verbunden. Nun wird über an den Gelenken angebrachten Potentiometern die jeweilige Beugung des Gelenks ermittelt, wodurch die Position des Objekts bestimmt werden kann.

Magnetismus : Beim magnetischen Tracking wird der User mit einem magnetischen Sender ausgestattet. Der ihn umgebende Raum ist mit 6 Empfängersensoren ausgestattet, die jeweils die Intensität der ausgesandten Magnetstrahlen messen. Mit Hilfe dieser Daten kann zentral die Position des Users bestimmt werden. Magnetisches Tracking ist äußerst empfindlich und schwer kalibrierbar, da Stahlkonstruktionen jeglicher Art, wie z.B. Stahlträger in den Wänden von Räumen, große Abweichungen hervorrufen können.

Bildverarbeitung : Bei dieser Trackingmethode werden in den Ecken des zu trackenden Raums Infrarotkameras montiert. Der User ist mit meist kugelförmigen Markern versehen, die reflektierende Eigenschaften haben. Die Trackingsoftware kann nun aus den gewonnenen 2D-Daten jeder Kamera die einzelnen Trackingpunkte zuordnen und die Positionen der Marker bestimmen.

2.2.2 Navigation

Für VR-Systeme gibt es mehrere unterschiedliche Eingabegeräte, die der Interaktion dienen. Hierzu zählen zum einen die gängigen 2D-Eingabegeräte, wie z.B.: Tastatur, Maus und Joystick. Zum anderen sind aber auch einige spezifische VR-Eingabegeräte entwickelt worden, von denen vier im folgenden beschrieben werden : die Cubic Mouse, Locomotion Interfaces, der Spaceball und der Datenhandschuh.

Cubic Mouse : Die Cubic Mouse wurde am Fraunhofer Institut für Medienkommunikation entwickelt und wird mittlerweile von der amerikanischen Firma Fakespace Systems produziert und vertrieben.

Bei der Cubic Mouse handelt es sich um ein „Eingabegerät, welches die intuitive Bestimmung von präzisen 3D-Koordinaten erlaubt“ [CUBIC]. Die Cubic Mouse besteht aus einem würfelförmigen, handtellergrößen Grundkörper, der den virtuellen Raum repräsentiert. Dies bedeutet es wird die Orientierung und Position der Cubic Mouse bestimmt und in der 3D-Welt interpretiert. Wird das Eingabegerät vom User gekippt, kippt auch das virtuelle 3D-Modell. Zusätzlich kann man über Buttons ein- oder auszoomen und die Trackingfunktionalität an- oder ausschalten. Ein weiteres Feature sind drei Stäbe, die durch das Gerät gesteckt sind und die einzelnen Achsen repräsentieren. Diese Stäbe können rotiert oder verschoben werden, wodurch eine „genaue Kontrolle über Orientierung und Position von Punkten und Flächen“ [CUBIC] in der 3D-Welt möglich ist.



Abb. 1 : Cubic Mouse detailliert

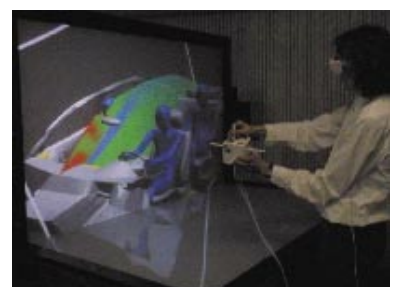


Abb. 2 : Cubic Mouse

Locomotion Interfaces : Diese Art von Interaktionsgeräten versucht natürliche Bewegungsmuster des Menschen aufzunehmen und in eine virtuelle Fortbewegung umzusetzen. Hierzu gehören z.B. das VRBike, das am MPI Tübingen entwickelt wurde. Es ermöglicht es dem Betrachter sich mit einem realen Fahrrad (Hometrainer) durch eine virtuelle Welt fortzubewegen, wobei er durch die Intensität des Tretens der Pedale die Geschwindigkeit regulieren kann.



Abb. 3 : Datenhandschuh

Spaceball : Auch der Spaceball ist ein 3D-Eingabegerät. Er ermöglicht Navigation in drei Achsen über eine handgroße, kugelförmige Kappe, die so gelagert ist, dass sie in alle Richtungen bewegt werden kann. Dies bedeutet, zieht man an der Kugel nach oben, wird das 3D-Modell nach oben verschoben, schiebt man sie leicht nach vorne, vollzieht das 3D-Modell diese Bewegung nach. Außerdem stellt das Eingabegerät einige frei programmierbare Buttons für weitere Funktionalitäten zur Verfügung.

Datenhandschuh : Datenhandschuhe messen zum einen die Krümmung der einzelnen Finger und verfügen zum anderen über ein Trackingsystem, wodurch die Position der Hand im Raum festgestellt werden kann. Diese Trackingsysteme basieren meist auf Ultraschallsensoren. Ein Problem ist die „Ermüdung des Nutzers“, da „das Heben des Arms über länger Zeit kraftraubend“ [Zachow] ist.

2.2.3 Sensorik

Die Interaktion mit den 3D-Welten erfolgt mittels Sensoren. Diese Sensoren sollen dem User eine einfach zu handhabende Interaktion ermöglichen und ihn ermuntern, die dargestellte 3D-Welt ohne Berührungängste zu erforschen und zu verändern. Dies wird besonders durch non-taktile Sensoren unterstützt, da die Bedenken, etwas zu zerstören, genommen werden. Falls die Sensorik berührt wird, muss sie einen robusten und stabilen Eindruck vermitteln.

Da man mit den Sensoren möglichst in Echtzeit arbeiten möchte, werden hohe Anforderungen an sie gestellt. Es sind mindestens 15 Werte in der Sekunde notwendig, um eine möglichst flüssige Darstellung zu erreichen. Dies soll natürlich mit geringem Performanceaufwand gewährleistet sein.

Im folgenden werden die Sensoren in boolesche und werteliefernde Sensoren unterteilt.

2.2.3.1 Boolesche Sensoren

Boolesche Sensoren können per Definition nur zwei Werte liefern : TRUE oder FALSE. Ein weiteres Unterscheidungsmerkmal ist die Art der Zustände, die ein boolescher Sensor annehmen kann. Dies kann z.B. in einem einmaligen Auslösen oder einem permanenter Zustand des an- oder ausgeschaltet sein bestehen.

2.2.3.1.1 Bewegungsmelder

Der Bewegungsmelder schließt aufgrund einer detektierten Bewegung einen Schalter um z.B. eine Alarmanlage auszulösen. Es gibt jedoch auch softwarebasierte Lösungen, wie z.B. erweiterte Funktionalität einer Webcam, die die aufgezeichneten Bilder eines Raumes miteinander vergleicht und somit aufgrund der Bildunterschiede feststellen kann, ob sich der aufgenommene Raum durch Bewegung verändert hat.

Daraufhin können softwareseitig Events ausgelöst oder andere Programme gestartet werden. Der Bewegungsmelder hat allerdings den Nachteil, dass eine detektierte Bewegung den Schalter für eine längere Zeit mittels eines Relais schließt, dies heißt er ist für Installationen höchstens zum Anschalten dieser geeignet, da die einzelnen Zustände sich nicht häufig genug ändern.

2.2.3.1.2 Lichtschranke

Ein weiterer Sensor ist die Lichtschranke. Die Lichtschranke sendet einen sehr dünnen Lichtstrahl aus, der auf eine Photozelle trifft. Diese Photozelle erhält ein gewisses Widerstandsniveau aufrecht und ändert dieses, wenn der Strahl unterbrochen wird. Hierdurch wird z.B. über ein Relais oder einen Kondensator ein weiterer Stromkreis geschlossen.

Man unterscheidet zwischen Ein- und Zwei-Weg-Lichtschranken, wobei bei den Ein-Weg-Lichtschranken Sender und Photozelle räumlich getrennt sind, also gegenüber angebracht werden und bei den Zwei-Weg-Lichtschranken diese beiden Komponenten sich in einem Modul befinden und der Lichtstrahl über einen Reflektor wieder zurückgeleitet wird.

2.2.3.1.3 Trittmatten

Auch Trittmatten, die in der Alarmtechnik zum Einsatz kommen, sind boolesche Sensoren. Hierbei handelt es sich um Plastikmatten, die es in unterschiedlichen Formaten gibt. Gängige Größen sind 18 cm x 61 cm oder 40 cm x 71 cm.

Die Matten verfügen im Inneren über zwei leitende Schichten, die durch den in der Matte vorherrschenden Überdruck voneinander fern gehalten werden. Die Matten können nun über zwei Anschlußkabel an einen Stromkreis angeschlossen werden und fungieren somit als Schalter. Werden die Matten mit ca. 30 kg belastet, schließen sie den Stromkreis.

2.2.3.2 Werteliefernde Sensoren

Werteliefernde Sensoren ermöglichen es, eine numerische Aussage über die Intensität der Interaktion zu treffen. Des weiteren ist mittels dieser Sensoren eine Interpretation der Umwelt möglich, was später anhand der einzeln vorstellbaren Sensoren näher erläutert wird.

2.2.3.2.1 Drucksensoren

Drucksensoren sind für eine Reihe von Einsatzmöglichkeiten vorstellbar. Hierzu gehören z.B. Interaktionsboards, auf denen der User stehen kann und durch Gewichtsverlagerung Steuerimpulse für die 3D-Welten generieren kann, die je nach Stärke umgesetzt werden können.

Auch sind einzelne Eingabegeräte, wie z.B. mit Sensoren ausgestattete Bälle denkbar, die durch die Stärke des auf sie ausgeübten Druckes Reaktionen in der 3D-Welt nach sich ziehen, wie z.B. Rotationen, Translationen oder Skalierungen.

2.2.3.2.2 Umweltsensoren

Unter Umweltsensoren kann man im allgemeinen Sensoren verstehen, die den umgebenden Raum bewerten und nach gewissen Aspekten erfassen.

Die hierdurch gewonnenen Daten können zur visuellen Interpretation in den 3D-Welten herangezogen werden. Mögliche Umwelteinflüsse können u.a. Temperatur, Luftdruck oder Luftfeuchtigkeit sein.

Temperatursensoren können sich sowohl auf die sich verändernde Umgebungstemperatur in Räumen beziehen als auch direkt durch die einzelnen User z.B. durch Berühren verändert werden. Für Feuchtigkeitssensoren gilt im wesentlichen dasselbe.

Im ZKM in Karlsruhe wurden z.B. Feuchtigkeitssensoren an Pflanzen angebracht, um bei Berührung durch Hände Events auszulösen, in diesem Falle wuchsen auf der vor den Pflanzen aufgestellten Leinwand virtuelle Pendants dieser.

2.2.3.2.3 Abstandssensoren

Man kann im wesentlichen zwischen drei verschiedenen Arten der Abstandsermittlung unterscheiden : Laser, Infrarot und Ultraschall.

Die Verfahren zur Ermittlung des Abstandes von Objekten zum Sensor unterscheiden sich prinzipiell relativ wenig.

Das Grundprinzip liegt im Senden eines Signals, welches durch ein Objekt im Raum reflektiert und zurückgesendet wird. Nun wird die Laufzeit des zurückkommenden Signals bestimmt und somit kann der Abstand ermittelt werden.

Je nach Einsatzgebiet werden unterschiedliche Sensorarten verwendet. Es kommt zum einen auf die Genauigkeit, die Reichweite, das reflektierende Objekt, die Umweltbedingungen und die Kosten an.

Abstandssensoren werden in vielen unterschiedlichen Gebieten eingesetzt : in der Robotik zum Bestimmen von Hindernissen, in der Automobilbranche als Parkhilfe oder in der Landwirtschaft zum Bestimmen von Füllständen in Silos, um nur wenige zu nennen.

Lasersensoren arbeiten meist mit dem Prinzip der Triangulation. Hiermit können äußerst genaue Ergebnisse erzielt werden und Oberflächen mit Genauigkeiten im Mikrometerbereich vermessen werden. Laserabstandsmessung ist jedoch im Vergleich zu anderen Methoden sehr teuer und kam somit in diesem System nicht zum Einsatz.

Infrarotsensoren werden häufig im Bereich der Robotik eingesetzt. Diese Sensormodule sind auch als Eingabegeräte interessant, haben aber meist eine äußerst geringe Reichweite (z.B. bis zu 80 cm).

Ultraschallsensoren kommen ebenso im Bereich der Robotik zum Einsatz, haben aber den Vorteil, dass sie größere Reichweiten erzielen können.

2.3 Bestehende immersive VR-Systeme

Seit Anfang der 90er Jahre hat die Wichtigkeit von immersiven VR-Systemen stark zugenommen.

Dies ist hauptsächlich auf den technischen Fortschritt und die damit verbundenen neuen Möglichkeiten zurückzuführen.

Dadurch dringt die VR immer mehr in den Consumerbereich vor, da aufgrund der großen Performancesteigerung dieser Systeme, nun auch aufwändige Medieninstallationen, die noch vor 10 Jahren nur mit erheblichen Entwicklungs- und monetärem Aufwand möglich gewesen wären, mit diesen Geräten realisierbar sind.

Dies führte zu vielen Entwicklungen im VR-Bereich, von denen ich im folgenden drei vorstellen möchte, die in Teilaspekten in engem Bezug zu dieser Arbeit stehen.

2.3.1 CAVE

Die CAVE ist ein im Electronic Visualization Laboratory (EVL) der Universität von Illinois entwickeltes VR-System. Der Name ist ein rekursives Akronym und steht für CAVE Automatic Virtual Environment. Die CAVE wurde „als nützliches Werkzeug für wissenschaftliche Visualisierung entworfen“ [Cruz-Neira 93] und das erste Mal auf der SIGGRAPH im Jahre 1992 vorgestellt.

Die ursprüngliche Implementierung der CAVE - wie in [Pape 97] beschrieben - besteht aus 4 Projektionsflächen die in einem gedachten Würfel vier Seiten (vorne, links, rechts, unten) darstellen. Die Projektionen erfolgen über Rück- und Aufprojektion, sodass der User sich innerhalb der Wände frei bewegen kann, ohne Schatten zu werfen. Um Platz zu sparen werden die Projektionen jeweils über Spiegel abgelenkt.

Der User hat nun die Möglichkeit eine 3D-Welt in dieser Projektionsumgebung zu betrachten, wobei ihm der virtuelle Blick nach vorne, rechts, links und unten lückenlos und gleichzeitig geliefert wird. Dies erzeugt beim User eine Art der Immersion, des Hineingezogenwerdens.

Dieser Punkt wird beim User noch durch eine stereoskopische Projektion der 3D-Welt, also einem echten dreidimensionalen Eindruck, gesteigert. Um diesen Effekt zu erzielen, muss der User eine Stereobrille tragen, die in der Originalkonfiguration der CAVE eine Shutterbrille der Firma Stereographics ist. Die Synchronisation der einzelnen Bilder für die Shutterbrille wurde über Stereoemitter erreicht.

Hierbei handelt es sich um „kleine, weiße Boxen, die an den Ecken der CAVE aufgestellt sind“ [Pape 97]. Der User kann mit einer sogenannten 3D-Maus mit den Welten interagieren.

Dieses Eingabegerät besteht laut [Pape 97] aus einem Joystick mit drei Tasten, der an einem separaten Rechner angeschlossen ist, welcher dem Steuerrechner die Eingabebefehle übergibt.

Die Hardware der CAVE besteht aus einer SGI Onyx mit vier Reality Engine 2 Graphikkarten, die jeweils einer Projektionsfläche zugeordnet sind.

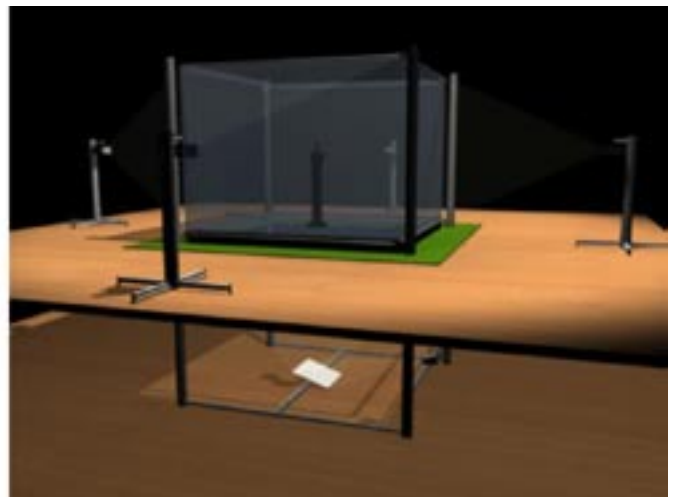
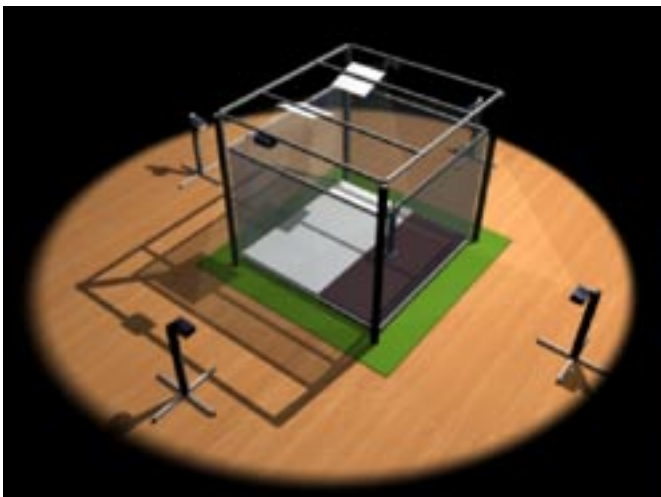
Als Betriebssystem wird das Unixderivat IRIX verwendet. Die Daten der dargestellten 3D-Welten basieren auf VRML-Dateien.

2.3.2 X-Rooms

Die X-Rooms wurden am Fraunhofer Institut entwickelt und legen ihr Hauptaugenmerk - wie unter [X-Rooms] beschrieben - auf eine kostengünstige Realisierung der CAVE, um sie einem weitem Anwenderkreis zugänglich zu machen.

Hierzu greifen sie auf marktübliche PC-Cluster anstatt der teuren SGI-Graphikrechner zurück, die unter Windows 2000 betrieben werden. Dies hat zum einen den Vorteil der geringen Wartungskosten sowohl der Software als auch der Hardware, zum anderen kann man auch von der rasanten Entwicklung der PC-Hardware profitieren, die den hochprofessionellen

Abb. 4 : 5-Wand-X-Room von oben
Abb. 5 : 5-Wand-X-Room in Seitenansicht



Graphikrechnern schon in einigen Teilgebieten überlegen ist.

Auch die X-Rooms implementieren die Stereoprojektion in ihrem System. Diese erfolgt jedoch nicht wie bei der CAVE über Shutterbrillen sondern wird mithilfe passiver Technik über mit Polarisierungsfilter versehenen Brillen realisiert.

2.3.3 CaveUT

Die CaveUT ist eine weitere Entwicklung im Umfeld der VR-Systeme.

Hierbei wird auf eine bestehende Software zurückgegriffen, dem Sharewarespiel und Egoshooter Unreal Tournament der Firma Epic.

Dieses Spiel verfügt über eine sehr leistungsstarke Spiele- und Renderengine, die den weiteren Vorteil hat, dass sich der Consumergraphikkartenmarkt an solchen Spielen orientiert.

Dies bedeutet, die großen Weiterentwicklungen von Graphikkarten werden in diesem Marktsegment erzielt, da hier ein großer Kundenkreis vorhanden ist.

Die Software ist für handelsübliche PCs (sowohl Windows als auch Linux) entwickelt und laut [Jacobson] bereits auf 300MHz-Systemen mit einer guten OpenGL-fähigen Graphikkarte (z.B. GeForce 4-Chip) lauffähig.

Das Programm liegt bis auf die Renderengine als Open Source-Projekt im Quelltext vor und darf für nicht kommerzielle Zwecke modifiziert und eingesetzt werden.

Unreal Tournament ist ein Netzwerk-Egoshooter. Dies bedeutet, der Spieler bewegt sich durch eine virtuelle Welt und ist währenddessen mit einem Server verbunden. Auf diesem Server befinden sich die Daten sämtlicher anderer Spieler, die sich in dieser Welt befinden und werden in Echtzeit aktualisiert. Somit weiß jeder Client permanent, wo sich alle Spieler befinden, da sie miteinander interagieren und sich gegenseitig sehen müssen. Es können sich laut [Jacobson] bis zu 32 Spieler gleichzeitig in einer Welt befinden.

Diese Netzwerkfähigkeit macht sich nun CaveUT zunutze. Hierbei ist in der Welt nur ein Spieler. Durch dessen Augen betrachtet man als sogenannter „Zuschauer“ [Jacobson] die Welt aus unterschiedlichen Blickwinkeln. Bei der einfachsten Konfiguration der CaveUT (2-wändig) wird z.B. nach vorne und auf eine Seite geschaut.

Die Betrachtungswinkel können so gewählt werden, dass sie nahtlos aneinander gereiht werden können. Der User bewegt sich nun durch die Welt. Dies zieht ein Ändern der Ansicht der Welt nach sich, wodurch die beiden Projektionen aktualisiert werden müssen.

Da die Software und das Netzwerk relativ leistungsfähig sind und sich laut [CavUT] mit ca. 30 Frames pro Sekunde aktualisieren, hat man aufgrund der Trägheit des Auges den Eindruck, sich ruckelfrei durch die Szenerie bewegen.

Abb. 6 : CaveUT in Eckprojektion



Die CaveUT hat einen relativ geringen Grad der Immersion, da bisher lediglich die Mehrwandprojektion und keine Stereoskopie implementiert sind, außerdem ist noch kein Soundsystem vorhanden.

2.4 Zuverlässigkeit und Wartbarkeit von Multimediasystemen

Die Sicherheit und Verfügbarkeit von permanent installierten Mediensystemen in Öffentlichkeit ist sehr wichtig.

Um eine höchstmögliche Verfügbarkeit dieser Systeme zu erreichen gibt es unterschiedliche Hard- und Softwarelösungen sowie Strategien, die dies unterstützen sollen.

Im folgenden soll kurz auf gängige Sicherheitsaspekte eingegangen werden.

2.4.1 Komponentenwahl

Der Komponentenwahl der Rechnersysteme muss große Aufmerksamkeit eingeräumt werden. Hierbei ist zum einen auf die hohe Qualität der einzelnen Komponenten und deren Zusammenspiel zu achten.

Außerdem ist eine Ausgewogenheit des Systems notwendig, da meist die schwächste Komponente die Verfügbarkeit bestimmt. Im Vorfeld müssen die Anforderungen an das System genau bestimmt werden und die Auswahl an Komponenten mit genügend Leistungsspielraum erfolgen.

2.4.2 Stromversorgung

Die Stromversorgung spielt eine große Rolle bei der Verfügbarkeit von Rechnersystemen. Einerseits brauchen diese Strom, um funktionieren zu können, andererseits spielt auch die Qualität des Stroms eine Rolle, da Spannungsspitzen im „unreinen“ Strom Hardwareschäden nach sich ziehen und somit die Verfügbarkeit der Systeme beeinträchtigen können.

Um diesem vorzubeugen kommen sogenannte USV-Anlagen zum Einsatz. USV steht hierbei für „unterbrechungsfreie Stromversorgung“. Diese Anlagen sorgen für die Filterung des Stroms, gleichen also die Spannungsspitzen aus.

Des weiteren übernehmen sie bei Stromausfall für eine gewisse Zeit über eigene Batterien die Stromversorgung der Systeme. Nun kann das betreute System durch die USV heruntergefahren oder bis zum Wiedereinsetzen des Stroms weiterbetrieben werden.

2.4.3 Redundanz

Unter Redundanz versteht man laut [Deboeser 02] das „mehrfache Vorkommen von Komponenten in einem System. Redundante Komponenten können die Services einer ausgefallenen Komponente sofort übernehmen, um ununterbrochenen Betrieb des Systems zu gewährleisten.“

Hierbei werden die Aufgaben von sogenannten Clustern übernommen. Dies bedeutet mehrere Rechner übernehmen eine Aufgabe, sind aber nur unter einer Adresse ansprechbar. „Im Fehlerfall gibt es zwei Strategien, nach denen ein Cluster den Betrieb weiterführt.

Entweder werden die Anwendungen auf dem funktionsfähigen Server neu gestartet (Failover), oder es erfolgt eine Übernahme im laufenden Betrieb (Takeover)“ [Raepple 01]. Das Takeover hat laut [Raepple 01] den großen Vorteil, dass der User den Ausfall kaum bemerkt, ist jedoch wesentlich komplexer zu implementieren, da sich beide Rechner permanent abgleichen müssen.

3. Grundlegende Erklärungen zu den verwandten Technologien

3.1 Blender

Die 3D-Software Blender ist eine Inhouse-Entwicklung der niederländischen Firma „NeoGeo“. 1998 wurde, nachdem „NeoGeo“ seine Geschäftstätigkeit aufgab, die erste Version von Blender im Internet frei zugänglich gemacht.

Zeitgleich gründete Ton Roosendaal, der Hauptprogrammierer von Blender, die Firma „Not a Number“, mit dem Ziel Blender weiterzuentwickeln.

Die Firma NaN hat im Frühjahr 2002 Insolvenz anmelden müssen. Nach zähem halbjährigen Ringen mit den Aktionären NaNs und einem Kraftakt der äußerst lebendigen Community von Blender, ist es Anfang Oktober 2002 gelungen, die Quelltexte der Software durch von der Community gespendete 100.000 Euro freizukaufen. Dies bedeutet eine weitere Entwicklung Blenders durch die Community als Open Source-Projekt.

Die Funktionalität Blenders kann man grob in zwei Bereiche gliedern : Modellier- und Animationsbereich und die Echtzeitengine.

Blender als Modellierungstool hat ähnliche Funktionalität wie kommerzielle Anbieter, wie z.B. 3D Studio Max von Discreet oder Maya von Alias Wavefront.

Die Echtzeitengine erlaubt es erstellte 3D-Szenen mit Navigation und Interaktion zu versehen und im Echtzeitmodus zu erleben.

3.2 SNMP

SNMP ist ein Akronym und steht für Simple Network Management Protocol. SNMP ist für das Verwalten und Überwachen von Geräten in TCP/IP-Netzwerken konzipiert.

Auf den Geräten wird im Hintergrund ein Softwareagent gestartet, der Informationen über das eigene System bereitstellt.

Diese Informationen können dann zentral überwacht und verändert werden. Die Informationen sind als Objekte in der Management Information Base (MIB) abgelegt und können von dort ausgelesen werden.

Es gibt eine standardisierte MIB, die für alle Geräte gültig ist, und herstellerspezifische Bibliotheken mit Zusatzinformationen.

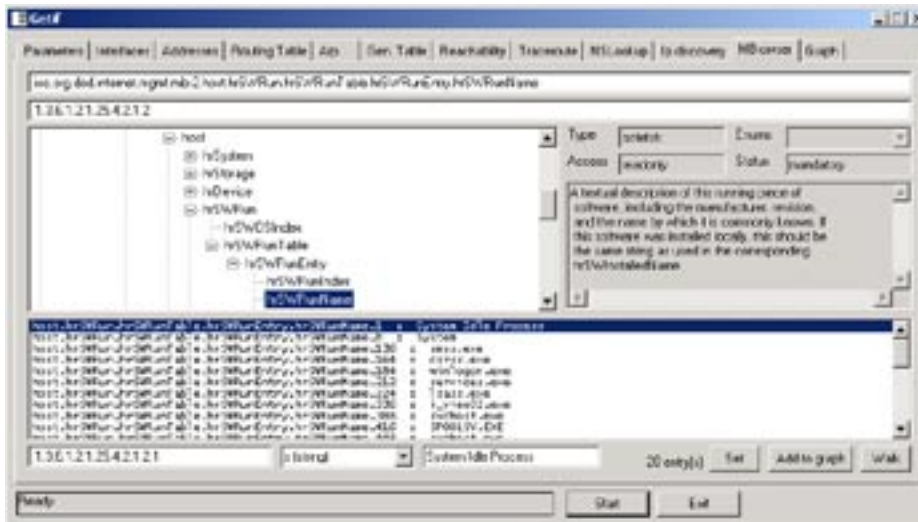


Abb. 7 : Screenshot des Programmes „Getf“, zeigt einen Teil der MIB-Struktur und den Eintrag für die laufenden Prozesse

Auszulesende Informationen können z.B. die Auslastung des Prozessors, Drehzahl des Lüfters, laufende Prozesse, Status des Netzteils oder freier Speicherplatz auf der Festplatte oder des RAMs sein.

Diese Informationen sind in einer tief verzweigten Baumstruktur abgelegt.

3.3 MIDI

MIDI steht für Musical Instruments Digital Interface und wurde als erstes auf der 1st North American Music Manufacturers Show 1983 in Los Angeles vorgestellt.

Die Idee hinter dem MIDI-Konzept liegt im Beschreiben von der Erzeugung von Musik und nicht der Musik selbst. Das heißt, es wird nicht der resultierende Ton, der eine sehr große Datenmenge beinhalten würde, sondern lediglich der erzeugende Impuls beschrieben.

MIDI ist somit ein Kommunikationsprotokoll und beschreibt die Klang- oder Tonerzeugung anhand von Begriffen, die ansonsten bei Tasteninstrumenten üblich sind.

Typische Nachrichten sind z.B. NoteOn oder NoteOff. In einer MIDI-Nachricht kann man die zu spielende Note festlegen und wählt hierzu aus 128 verschiedenen Tonhöhen, wobei die Tonhöhen 21-108 der normalen Klaviatur entsprechen.

Außerdem kann man die Stärke des Anschlags (velocity) und den sogenannte Pitch, also die Modulation der Note, festlegen. MIDI verfügt über 16 Kanäle somit sind 16 einzelne Stimmen gleichzeitig abspielbar.

Die MIDI-Nachrichten können nun in unterschiedliche Klänge synthetisiert werden. Dies wird auf Computerebene durch einen Softwaresynthesizer vorgenommen.

Hierbei kann man meist aus einer Soundbank mehrere verschiedene Instrumente auswählen, die die einzelnen Noten in sich definiert haben müssen.

Als zweite Möglichkeit kann man MIDI-Nachrichten zur Steuerung von Softwaresynthesizern, wie z.B. Reaktor von Native Instrument nutzen, um in Echtzeit Klänge zu erzeugen und einige Parameter der Klangerzeugung über MIDI zu steuern.

3.4 Python

Python ist wie in [von Löwis 01] beschrieben eine objektorientierte, interpretierte Skriptsprache, die ursprünglich von Guido van Rossum zum Testen eines von ihm entwickelten Betriebssystems namens „Amoeba“ entwickelt wurde.

Im Zusammenhang mit der 3D-Software Blender kann Python in zwei Bereichen zum Einsatz kommen : während der Entstehung der 3D-Welten und zur Laufzeit der Gameengine. Man muss hierbei zusätzlich zwischen blendertypischen Modulen und den allgemeinen Fähigkeiten von Python unterscheiden.

3.5 Java

Java wurde 1991 von der Firma Sun Microsystems ins Leben gerufen. In Java geschriebene Programme „sollten beispielsweise über Fernsehkanäle auf Fernseher geladen werden können oder aber als Steuerung anderer technischer Haushaltsgeräte dienen“ [Schütte 99].

Java entwickelte sich jedoch schnell zu einer Hochsprache und steht somit in direkter Konkurrenz zu C++ .

Java ist dabei streng objektorientiert konzipiert und integriert wie in [Schütte 99] beschrieben einige Konzepte und die Syntax von C++, ohne missverständliche Teile, wie z.B. Zeiger und Mehrfachvererbung zu übernehmen.

4. Anforderungen an das zu entwickelnde System

Das im Laufe dieser Diplomarbeit entstandene System hat den Arbeitstitel Cube4, der im folgenden verwendet wird.

Einige Prinzipien, denen Cube4 zugrunde liegt, sind im letzten Punkt unter X-Rooms und CaveUT bereits angesprochen worden, sollen hier aber nochmals detailliert erläutert werden.

Mit Cube4 soll ein leicht zu administrierendes, günstiges und vor allem gut skalierbares Gesamtsystem zur Darstellung von 3D-Welten in immersiven Displayumgebungen erstellt werden.

Besonderes Augenmerk wird zusätzlich auf die einfache Erstellung und Einbindung von Content in dieses System gelegt.

Außerdem sollen einige neue Wege der Interaktion getestet und implementiert werden, die dem User einen intuitiven Zugang zu den 3D-Welten ermöglichen.

4.1 Einsatzgebiete

Das VR-System Cube4 kommt hauptsächlich im öffentlichen Raum zum Einsatz. Dies sind zum einen Galerien und Museen, aber auch Foyers großer Gebäude, Bahnhöfe und Flughäfen. Auch auf Plätzen sind derartige Projektionsumgebungen denkbar. Des weiteren soll Cube4 auf Messen, Kunst- oder Musikevents zum Einsatz kommen.

Aus diesen örtlichen Begebenheiten ergeben sich einige Konsequenzen für das System.

Zum einen ist die Stereoskopie nicht vorgesehen, da das System extrem flexibel einsetzbar sein muss und bei Stereoskopie ein erheblicher Kalibrierungs- und Wartungsaufwand zu erwarten ist. Außerdem bringt die Stereoskopie einen Mehraufwand an Hardware und Softwareentwicklung mit sich.

Ein weiterer Punkt ist das notwendige Tragen von Brillen (Shutter- oder Polarisierungsbrillen), um die Stereoskopie erlebbar zu machen. Dies ist für Museen und Galerien sicherlich interessant und organisierbar, jedoch bei permanenten, öffentlichen Installationen nicht oder nur schwer realisierbar und außerdem mit einem zusätzlichen monetären Aufwand verbunden.

Ein weiterer Unterschied zur CAVE soll die Zugänglichkeit sein. Die CAVE ist ein in sich geschlossener Raum, der lediglich einer geringen Anzahl von Personen Zutritt erlaubt, da er meist Ausmaße von ca. 3 x 3 m hat.

Cube4 soll relativ flexibel auf die räumlichen Begebenheiten reagieren können und somit freier in der Positionierung und Anzahl der Projektion sein und mehreren Usern gleichzeitig zugänglich sein.

4.2 Interaktion

Mit dem Einsatzgebiet sind auch eng die Anforderungen an die Interaktion verbunden.

Hierbei muss man zwischen einer festen Installation ohne Betreuung und dem Erzeugen visueller Effekte auf Veranstaltungen unterscheiden.

Für letzteres kann man auf die gängigen Eingabegeräte wie Maus, Tastatur oder Joystick zurückgreifen, da lediglich ein Artist die Steuerung durch die 3D-Welten innehat. Handelt es sich jedoch um permanente Installationen, die interaktiv vom Publikum gesteuert werden, sind diese Eingabegeräte eher ungeeignet, da ungeübte Zuschauer Berührungssängste haben und diese Geräte nur von einer Person zu bedienen sind. Dies ist auch ein Unterschied zur herkömmlichen CAVE, die meist für eine oder sehr wenige Personen ausgelegt ist.

Dieses kann und soll bei diesem System nicht vorausgesetzt werden, da das System aufgrund der nicht vorhersehbaren räumlichen Situation flexibel sein muss.

4.3 Administration und Stabilität

Es soll von Anfang an großer Wert auf eine möglichst leichte Administrierbarkeit des Systems gelegt werden, welche die Mobilität und leichte Veränderbarkeit von Cube4 unterstützen soll.

Des weiteren ist es Ziel das Gesamtsystem leicht und sicher starten und stoppen zu können oder, wie bei permanenten Installationen erforderlich, automatisch hochzufahren und zu beenden. Im laufenden Betrieb ist ein Wechseln der 3D-Welten oder zumindest der einzelnen Szenen ebenso notwendig, um die Darstellung zu variieren, wie die Erreichbarkeit und Administrierbarkeit der Rechner nach Aufbau der Installationen, um im Fall eines Zusammenbrechens des ganzen Systems oder einzelner Rechner schnellen Zugriff zu haben.

4.4 Skalierbarkeit

Cube4 soll sehr auf modulhaften Aufbau und Skalierbarkeit ausgelegt sein. Einzeln entwickelte Softwaremodule zur Interaktion und Mehrwandprojektion sollen lückenlos miteinander kombinierbar sein und sozusagen eine Art Bausteinsystem bilden, das schnell und einfach kombinierbar ist.

Hierdurch wird auch die Skalierbarkeit unterstützt.

Das System muss mit geringem zeitlichen Aufwand aus einer nur für eine Wand erstellten 3D-Welt für den Einsatz auf beliebig vielen Projektionsflächen mit nahezu derselben Funktionalität ausbaufähig sein.

Dies gilt auch für die Sensoren, die prinzipiell in beliebig großer Anzahl eingesetzt werden sollen.

Die Berechnung und Abfrage der einzelnen Sensoren kann unter Umständen sehr aufwändig werden. Hierfür ist eine dezentrale Gewinnung der Sensordaten notwendig. Dies soll über die Netzwerkfähigkeit der Module möglich gemacht werden.

4.5 Kosten

Die Gesamtkosten des Systems sollen äußerst gering gehalten werden um das System einem möglichst großen Kundenkreis anbieten zu können. Die Skalierbarkeit des Systems und die Variierbarkeit der Module sollen zudem dem Kundenanspruch entsprechen.

4.6 Inhalte

Die Welten, an denen dieses System getestet und entwickelt wird, zeichnen sich durch eine große Dynamik und darstellende Varianz aus.

Man bewegt sich nicht durch stehende Architekturen, sondern durch dynamische, sich stets verändernde, räumliche Konstrukte.

Hierbei wird dem User ein Werkzeug gegeben, mit dem er durch Interaktion die Räume selbst generieren und verändern kann. Es gilt also nicht vorgefertigte Architekturen zu erkunden, sondern durch die eigene Bewegung und das Erzeugen von Objekten virtuelle Architekturen zu kreieren.

Dieses interaktive Einbeziehen des Betrachters und die daraus resultierende, stetige Neuerschaffung virtueller Welten sollen die Aufmerksamkeit des Users stark und andauernd auf sich ziehen.



Abb. 8 : Screenshots von Datenflügen

5. Design

5.1 Gesamtüberblick über Cube4

Das Gesamtsystem Cube4 umschließt im wesentlichen Konzepte und Realisierungen für fünf Bereiche : Interaktion, Administration, Stabilität, Bildgenerierung und Ausgabe eines immersiven VR-Systems.

Im Bereich Interaktion werden verschiedene Eingabegeräte entwickelt und realisiert, um ein gewisses Spektrum an Interaktionsmöglichkeiten zu bieten.

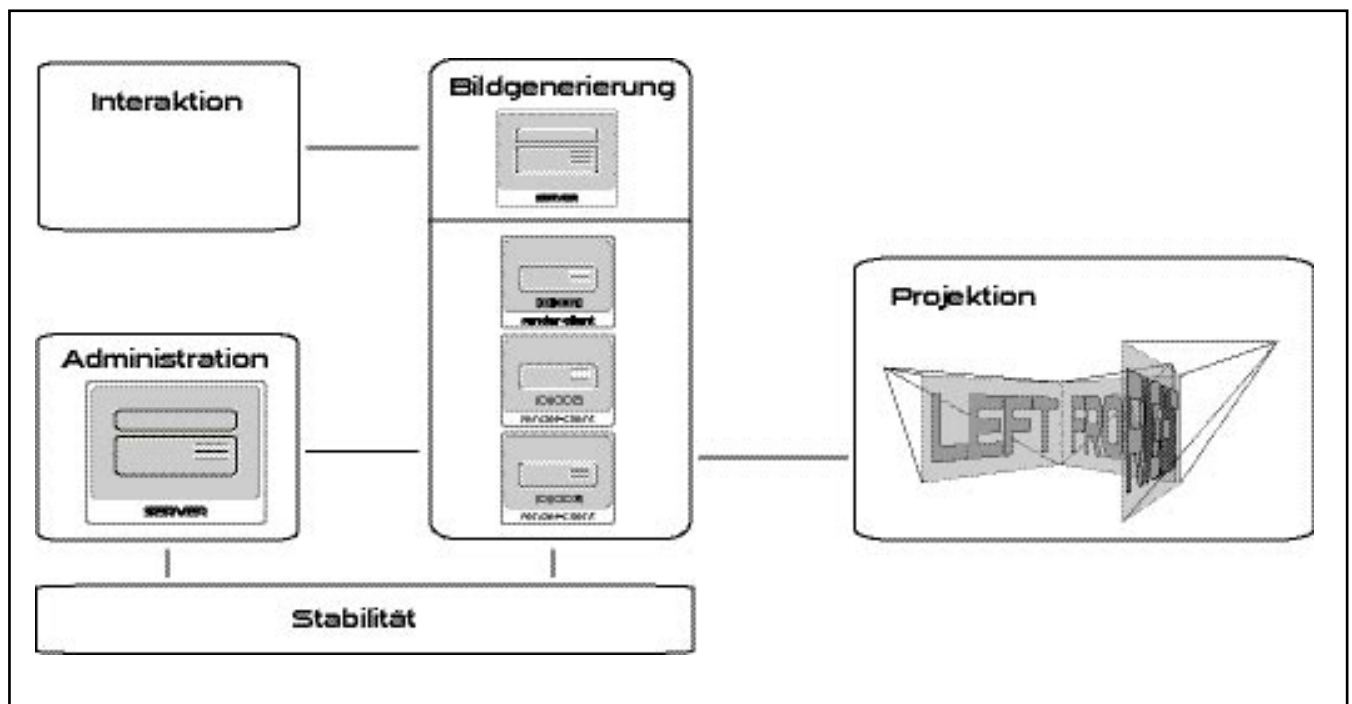
Der Bereich Administration umfasst die Ablaufsteuerung der VR-Welten.

Die Stabilität des Gesamtsystems wird durch Überwachung der Einzelkomponenten sowohl durch Hardware- als auch durch Softwarelösungen gewährleistet.

Im Bereich Bildgenerierung wird die korrekte Darstellung der VR-Welten durch Verarbeitung und Verteilung der Interaktionsevents erreicht.

Der Bereich Ausgabe enthält unterschiedliche Strategien zur Projektion und Darstellung der VR-Welten und Konzepte zur weiteren Ausgabe und Beeinflussung anderer Medien.

Abb. 9 : Überblick über Cube4



5.2 Beschreibung der virtuellen Welten

Die Welten, an denen dieses System getestet und entwickelt wird, zeichnen sich durch eine große Dynamik und darstellende Varianz aus.

Der User bewegt sich grundsätzlich durch einen schwarzen Raum ohne Begrenzungen und erzeugt mit einer bestimmten Frequenz Objekte. Diese können beliebiger Gestalt sein, wobei natürlich auf die Komplexität der Geometrien zu achten ist. Als Basisobjekte werden einfache, rechteckige Flächen verwendet die mit einer Textur bezogen sind.

Diese Texturen bestehen im wesentlichen aus durchsichtigen Bügeln unterschiedlicher Farbe, die vor dem sich in Bewegung befindenden User erzeugt werden. Hierdurch entsteht eine Art Tunnelwahrnehmung.

Nun hat der User die Möglichkeit mit dem System zu interagieren. Er kann zum einen die Art seines Fluges verändern, d.h. er kann seine Geschwindigkeit erhöhen oder verringern, Vorwärts- und Rückwärtsloopings fliegen oder sich seitlich abrollen.

Hier sind sämtliche Bewegungsarten in unterschiedlichen Intensitäten vorstellbar, was später noch deutlicher erörtert wird.

Die zweite entscheidende Interaktionsmöglichkeit ist das eigenhändige Erzeugen von Objekten. Auch hier ist wieder jede Art von Objekt vorstellbar.

Diesen Objekten können mit unterschiedlichen Verhaltensweisen, wie z.B. Kollisionsfähigkeit, Animationen, usw. versehen werden, die entweder zufällig, stets oder auf Interaktion durch den User ausgelöst werden.

Durch die dynamische Generierung der Objekte und somit der Welten entsteht eine nahezu unendlich große Vielfalt des Aussehens der Welten, da lediglich die Rahmenbedingungen der Welten festgelegt sind, das eigentliche Aussehen der Welten aber durch die Interaktion der User generiert wird.

5.3 Kostenreduzierung durch Hard- und Softwarewahl

Es wird versucht, ähnlich wie bei den X-Rooms, die Kosten für das System sehr gering zu halten.

Hierfür ist in erster Linie der geplante Einsatz von Consumer-PCs zu erwähnen, die um ein Vielfaches günstiger als die High-End-Graphikrechner, wie z.B. Rechner der Firma SGI, sind.

Als Betriebssystem wird derzeit das Microsoft Betriebssystem Windows 2000 verwendet, da dies relativ einfach zu administrieren ist und die notwendige Stabilität mit sich bringt. Eine Portierung auf Linux ist allerdings für die nahe Zukunft geplant, welche durch die Plattformunabhängigkeit der eingesetzten Technologien Blender, Python und Java mit relativ geringem Aufwand zu realisieren ist und



Abb. 10 : Skizzierung der Auswirkung der interaktiven Änderung der Bewegung im Raum

somit die Softwarekosten des Systems weiter senken wird.

Die eingesetzte Software ist Freeware. Blender ist mittlerweile ein Open Source-Projekt und auch die restlichen Softwarekomponenten wie das Java SDK, Python, die Pythonerweiterungen und die Java Entwicklungsumgebung Forte sind frei verfügbar.

Des weiteren profitiert man von der rasanten Entwicklung des Graphikkartenmarktes für Spiele, die sehr hohe Anforderungen an die Graphikleistung stellen. Somit können die Erneuerungszyklen für die Hardware wesentlich kürzer gestaltet werden

5.4 Echtzeit durch Blender

Die meisten CAVE-Anwendungen basieren auf VRML.

VRML hat als Echtzeit-3D-Format auch einige Vorteile gegenüber Blender. Es gibt unter anderem leistungstärkere Renderer, was sich zum einen in der besseren Lichtberechnung, zum anderen im sehr guten und effizienten Umgehen mit detaillierten, hochpolygonalen Modellen äußert. Des weiteren bietet VRML einige Features, die Blender nicht zur Verfügung stellt. Man kann z.B. bei manchen Playern ein Linienrendering von einzelnen Objekten einstellen, wodurch sehr feine Strukturen möglich sind, die wenig Rechenaufwand erfordern.

VRML hat aber auch große Nachteile. Hierzu gehört, dass der VRML-Standard seit 1997 nicht mehr weiterentwickelt wird und es kaum komfortable VRML-Editoren gibt.

Man muss für das Modelling meist auf externe Tools, wie z.B. Maya oder 3D Studio Max zurückgreifen, die über einen VRML-Export verfügen, da selbst das Positionieren von Objekten in Editoren, wie z.B. „Cosmo Worlds“ oft sehr unkomfortabel ist. Auch der Funktionsumfang von VRML ist relativ gering. Die Steuerung durch die Welten ist im wesentlichen vorgegeben und die Anzahl der Sensoren und Aktuatoren in VRML ist relativ gering.

VRML lässt sich durch Hochsprachen wie C++ oder Java erweitern. Dies ermöglicht auch komplexe Anwendungen wie z.B. eine CAVE, erfordert aber immensen Entwicklungsaufwand.

Blenders Hauptstärke liegt darin, dass alle Produktionsschritte in einer Software erfolgen können.

Blender ist ein mächtiges Modellier- und Animationstool, welches den Vergleich mit vielen kommerziellen Produkten nicht scheuen muss.

Die Gameengine ist im Softwarepaket integriert und verfügt über wesentlich mehr Funktionalität als VRML standardmäßig bietet. Außerdem kann man die Funktionalität von Blender sehr leicht über die voll integrierte Skriptsprache Python erweitern.

Das in der Gameengine enthaltene dynamische System erlaubt es sehr komplexe Steuerungen und Verhaltensweisen von Objekten schon mit geringen Programmierkenntnissen zu implementieren.

VRML ist besser für statische und hochaufgelöste Modelle geeignet, während Blender bei dynamischen, sich verändernden Welten klar überlegen ist.

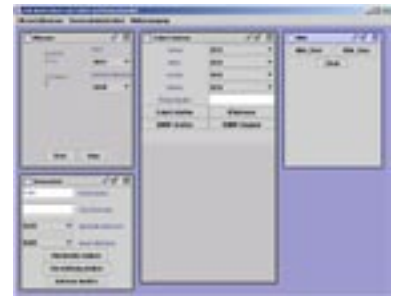


Abb. 11 : Screenshot von Cube4J

5.5 Beschreibung des Javatools Cube4J

Das Programm Cube4J stellt Funktionalität für das Starten und Stoppen eines Teils der Interaktionsgeräte und die Verwaltung von Cube4 und seinen einzelnen Komponenten zur Verfügung und fungiert somit als Steuerzentrale des Systems.

Dies beinhaltet das Starten von Cube4, das Auswählen der Rechner im Netzwerk für die mehrwandige Darstellung, den Filewechsel zur Laufzeit und die Überwachung der Funktionalität von Cube4.

Außerdem übernimmt Cube4J einige Aufgaben der Interaktion und der Ausgabe. Hierbei kann z.B. ein Teil der Sensorik gestartet und kalibriert werden. Außerdem ist die MIDI-Ausgabe des Systems, die später noch näher beschrieben wird, durch Cube4J zu starten.

Cube4J ist komplett in Java programmiert. Dies hat mehrere Gründe.

Zum einen bietet Java gute Möglichkeiten zum schnellen Implementieren von Oberflächen und zum anderen eine Vielzahl von zusätzlichen übersichtlichen APIs mit klarem Umfang, die die notwendige Funktionalität meist abdecken.

Die teilweise geringere Geschwindigkeit im Vergleich zu C++ ist zu vernachlässigen, da die meisten Funktionen, wie z.B. das Starten von Cube4, nicht auf große Performance angewiesen sind.

Abb. 12 : drei Beispiele für 3D-Interpretationen der Webcambilder



5.6 Interaktionskonzepte für die 3D-Welten

5.6.1 Webcam

Durch die Kopplung einer Webcam mit dem System, werden einige konkrete Interaktionen möglich.

Der Vorteil der Webcam liegt hierbei in dem großen Informationsgewinn (Bildinformationen) über den Raum, der durch die Webcam beobachtet wird. Da die Webcam einen großen Ausschnitt des Raumes abdecken kann, können sich mehrere Personen auf dem von der Webcam erzeugten Bildern wiederfinden und durch ihre bloße Anwesenheit und Repräsentierung als Pixelwerte Einfluss auf die Veränderung der 3D-Welten nehmen.

Dieser Einfluss kann sich z.B. in einer Veränderung oder Neupositionierung der dargestellten Geometrien, z.B. Kuben, anhand der einzelnen Pixelwerte äußern. Hierbei wird das Bild Pixel für Pixel ausgewertet und die Entsprechung in der 3D-Welt neu positioniert.

Um ein Wiedererkennen des Raums zu erreichen, muss eine große Anzahl an Geometrien das Bild repräsentieren, was eine große Anforderung an die Leistung des Systems darstellt.

5.6.2 Abstandssensor

Das Messen des Abstandes und die Interpretation des Wertes in der 3D-Welt, ist ein weiteres Interaktionskonzept.

Der User kann mit der 3D-Welt über seine Entfernung zum Sensor mit der Installation interagieren und somit die Darstellung der 3D-Welt verändern.

Der Abstandssensor kann auf verschiedene Art und Weise eingesetzt werden. Er kann zum einen den ermittelten Wert numerisch über UDP an Blender weitergeben, wo er dann z.B. in eine Translation von Objekten oder des Betrachterstandpunkts werden kann.

Andererseits kann je nach Entfernungsbereich, in dem der User sich befindet, ein Tastenevent erzeugt werden, das anschließend von Blender interpretiert wird. Diese Interpretation kann z.B. das Erzeugen unterschiedlicher Objekte oder Aktivieren und Deaktivieren von Bereichen in der 3D-Welt sein.

5.6.3 Trittmatten

Trittmatten können zwei unterschiedliche Interaktionsaufgaben in Cube4 übernehmen.

Zum einen kann man die Navigation durch die virtuellen Welten und das Erzeugen von Interaktionsevents über Trittmatten vornehmen. Jede Trittmatte repräsentiert eine Änderung der Bewegung in eine gewisse Richtung oder löst bei Berührung ein gewisses Event aus, z.B. das Generieren neuer Objekte.

Eine weitere Einsatzmöglichkeit besteht darin, die Trittmatten zur räumlichen Bestimmung von Usern im Raum zu verwenden.

Hierbei kann der Raum vor der Medieninstallation flächendeckend mit Trittmatten ausgelegt sein, die alle über den Tastaturkontroller mit dem Rechner verbunden sind.

Nun wird über den erzeugten Tastendruck jeweils die Position bestimmt. Hierdurch ist zwar keine genaue Bestimmung der Position der User möglich, man kann jedoch mehrere User gleichzeitig mit einbeziehen, da diese sich z.B. nicht wie bei einem Abstandssensor im Schatten des Abtastkegels befinden können.

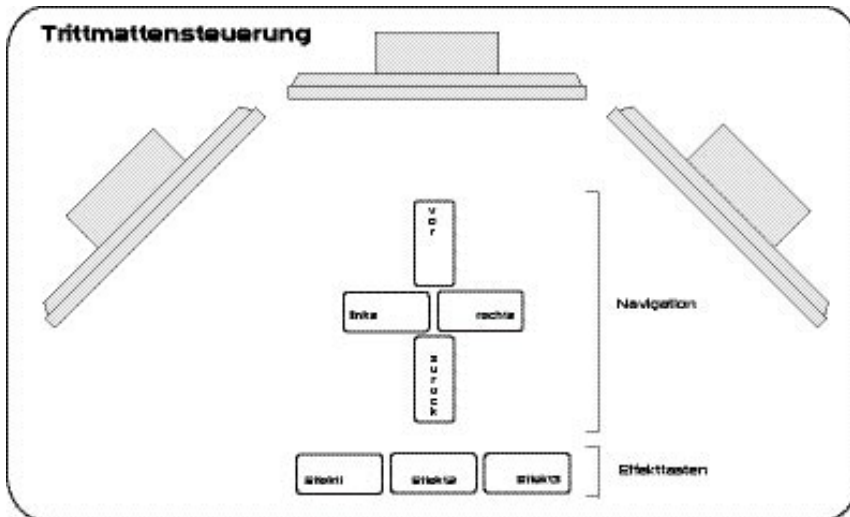


Abb. 13 : Trittmattensteuerung

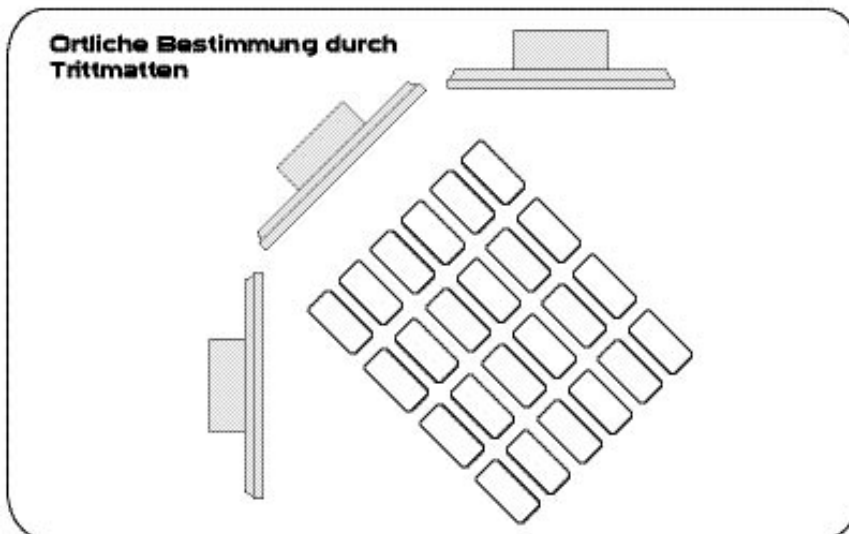


Abb. 14 : Räumliche Bestimmung durch Trittmatten

5.7 Administration

5.7.1 Starten von Cube4

Das Starten von Cube4 ist in zwei unterschiedlichen Szenarios denkbar. Zum einen beim Starten der einzelnen Rechner zum anderen mittels des Administrationstools.

Durch das Starten mit Hilfe des Administrationstool entsteht ein geringerer Aufwand beim Vorbereiten der Installation.

Man kann hierbei auswählen welcher Rechner, welche Seite der Projektion übernehmen soll. Außerdem kann das darzustellende File ausgewählt werden.

Der Nachteil hierbei ist, dass ein Operator das Starten ausführen muss und es nicht automatisch vonstatten geht.

Das automatische Starten ist für Festinstallationen in Museen von großer Bedeutung, da dort selten ein Operator permanent vor Ort ist.

Für einen automatischen Start müssen die darstellenden Rechner bereits bekannt und sämtliche Files vorbereitet sein. Der Start kann nun durch unterschiedliche BIOS- und Autostarteinstellungen automatisch vorgenommen werden.

5.7.2 Filewechsel

Der Filewechsel zur Laufzeit von Cube4 ist besonders für Liveauftritte wichtig, da man hier Blender nicht auf jedem Rechner einzeln beenden und mit dem neuen File starten kann.

Aus diesem Grund muss man Strategien entwickeln, dies zentral und mit möglichst wenig Administrationsaufwand zu gewährleisten. Um den Filewechsel zu ermöglichen gibt es momentan zwei Wege. Zum einen durch das Javatool Cube4J und zum anderen direkt durch ein Tastenevent, das innerhalb Blenders interpretiert wird.

Erstere Methode ist sehr komfortabel und vom Operator einfach durchzuführen, hat jedoch den Nachteil, dass Blender automatisch geschlossen und neu gestartet wird.

Zweiteres ist stets neu vorzubereiten und relativ aufwendig, bietet aber den Vorteil, dass Blender nicht verlassen wird und die grundsätzliche Funktionalität des Verschickens von Interaktionsevents, was später näher beschrieben wird, bereits vorhanden ist.

5.8 Konzept zur Sicherung der Stabilität von Cube4

Eine größtmögliche Verfügbarkeit für Cube4 als interaktive Medieninstallation wird über das skizzierte 3-Stufen-Konzept erreicht, um einen hohen Grad an Lauffähigkeit und Stabilität sicherzustellen.

Hierbei wird der Server als zentrales Stück des Systems durch eine eigene Kontrollhardware geschützt, die dessen Verfügbarkeit stets kontrolliert und gegebenenfalls den Server neu startet.

Die Kontrolle der Clients wird vom Server über SNMP, also durch Softwareüberwachung, realisiert. Dies hat den Vorteil der geringeren

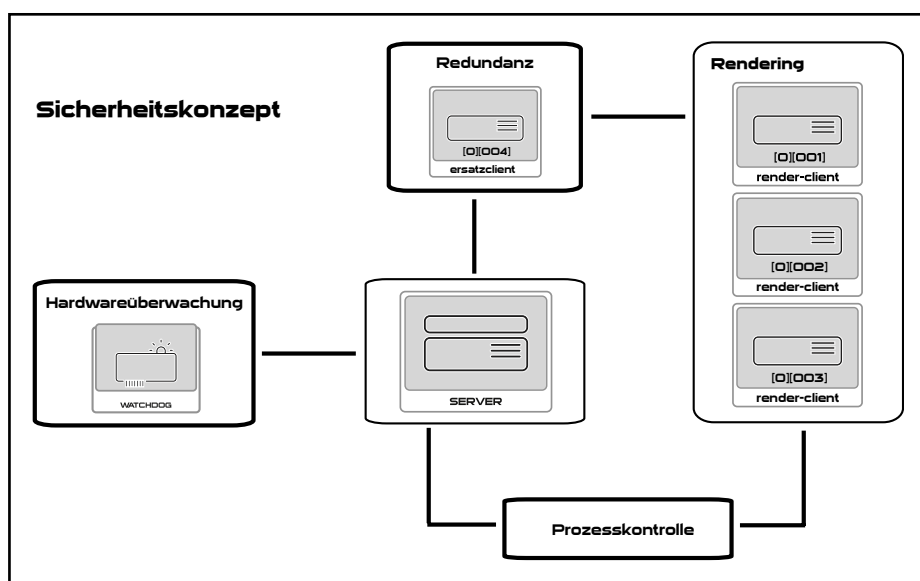


Abb. 15 : Sicherheitskonzept

Kosten, da evtl. eine große Anzahl von Clients zu überwachen ist und man über dieses Protokoll sehr viele Parameter der Rechner auslesen kann.

Falls ein Client ausfällt, gibt es zwei Möglichkeiten : Entweder wird der hängende Prozess beendet und neu gestartet oder ein Ersatzrechner übernimmt dessen Funktion, während der gestörte Rechner heruntergefahren und neu gestartet wird. Dieser übernimmt anschließend wieder die Position des Ersatzrechners.

5.9 Bildgenerierung

5.9.1 Möglichkeiten der geometrischen Anordnung

Das Prinzip von Cube4 basiert auf dem Darstellen einer einzigen 3D-Welt auf mehreren Projektionsflächen. Hierbei ist für jede Projektionsfläche ein eigener Rechner zuständig.

Bei der CAVE handelt es sich um 5 Flächen, die wie die Flächen eines Würfels angeordnet sind (ausgenommen der Rückwand). Die richtige Projektion wird durch ortogonales Setzen der Kameras in der 3D-Welt auf jede Projektionsfläche erreicht.

Dies bedeutet, jede Kamera hat denselben Standort, ist aber dann um einen bestimmten Winkel (bei der CAVE 90 Grad) gedreht.

Hierzu muss die Brennweite der einzelnen Kameras korrekt gesetzt werden um Überschneidungen zu verhindern.

Vorstellbar sind unterschiedliche Szenarios. Das bedeutet, man kann den Winkel und die Brennweite der einzelnen Kameras in Blender beliebig wählen und somit z.B. die Projektionsflächen halbkreis- oder kugelförmig positionieren.

5.9.2 Darstellung und Synchronisation der einzelnen Projektionsflächen

Der Aufbau des Cube4 funktioniert prinzipiell ohne Synchronisation, solange sich weder die Position des Betrachters noch die Position der dargestellten Objekte verändert.

Ist dies jedoch der Fall muss jedem Rechner jede Veränderung in der Welt mitgeteilt werden, da sich einerseits die Betrachterposition verändern kann und andererseits Objekte von einer Darstellungsfläche zur nächsten wandern können.

Hierzu wird ein eigener Server verwendet, der den darstellenden Rechnern, im folgenden Clients genannt, die Veränderungen der Welt mitteilt. Die Position des Betrachters, der in der 3D-Welt durch eine Kamera repräsentiert wird, ist durch die Positionskoordinaten und die Orientierung, also die Neigung im Raum definiert.

Diese Daten werden vom Server permanent erzeugt und allen Clients framegenau mitgeteilt, um eine korrekte Darstellung zu erreichen.

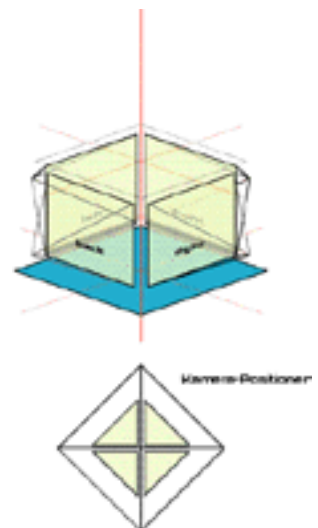


Abb. 16 : 4-Wand-Projektion

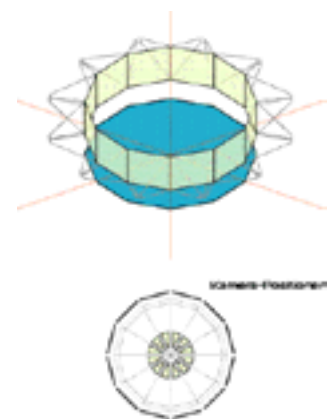


Abb. 17 : 360-Grad-Projektion

5.9.3 Objektkontrolle

Um ein Objekt richtig kontrollieren zu können, müssen dieselben Aspekte erfüllt sein, wie bei der Synchronisation der Darstellung.

Da jedes Objekt noch nach der Generierung bewegt und verändert werden kann, muss zu jeder Zeit die Veränderung mitgeteilt werden.

Diese Veränderung umfasst die Position und die Orientierung, wodurch die genaue Lage des Objektes definiert ist. Beide Komponenten werden verändert, sobald ein Objekt nach der Erzeugung transliert oder rotiert werden kann.

Fällt eine dieser Operationen weg, muss die Änderung der zugehörigen Komponente nicht weiter kontrolliert werden. Dies bedeutet z.B. wird ein Objekt nach Generierung nicht mehr rotiert, ändert sich auch die Orientierung nicht mehr und muss somit nicht aktualisiert werden.

5.9.4 Verteilung der erzeugten Interaktionsevents

Die einzelnen Interaktionsevents, die z.B. ein Hinzufügen von einem Objekt auslösen, müssen an die Clients übertragen werden. Dies ist erforderlich um die gleiche Anzahl von Objekten bei jedem Client zu garantieren.

5.10 Ausgabe

5.10.1 Erzeugung von MIDI-Daten durch Blender

Ein zusätzliches akustisches Erleben der 3D-Welten kann die Immersion bedeutend steigern. Hierfür steht in Cube4 prototypisch ein MIDI-Generator zur Verfügung.

Dieses Modul ist mit Java realisiert und interpretiert die versendeten Positions- und Orientierungsdaten. Hierbei werden Notenevents entweder an systemeigene Synthesizer geschickt und dort in Töne umgewandelt oder um Steuersignale für angebunden Softwaresynthesizer, wie z.B. Reaktor von Native Instruments, weiterzuleiten. Dies bietet die Möglichkeit bei Liveauftritten mit Musikern zusammenzuarbeiten, die die erzeugten Klänge in Echtzeit bearbeiten können.

5.10.2 Datenbankbindung

Die Datenbankbindung von Blender erfolgt mittels Python. Hierzu kann man auf mehrere Pythonmodule zurückgreifen, die im Internet frei erhältlich sind und sehr komfortable Schnittstellen zu den meisten Datenbanksystemen bereitstellen.

Für Cube4 ist noch keine Datenbankbindung realisiert jedoch für die Zukunft angedacht.

Implementiert wurde eine Datenbankbindung für Blender jedoch bereits als einwandige Applikation, die durchaus auch in der immersiven Displayumgebung lauffähig wäre.

Hierbei wurde symbolisch ein Verkehrsfluss visualisiert.

Die Daten für das Verkehrsaufkommen einer Stadt wurden in einer MySQL-Datenbank abgelegt, wobei es sich um eine Tabelle mit dem jeweilige Verkehrsaufkommen an einer Kreuzung zu verschiedenen Uhrzeiten handelte.

Diese Daten wurden innerhalb Blenders ausgelesen und interpretiert. Hierfür wurde je nach Verkehrsaufkommen eine unterschiedliche Anzahl von Autos auf den Straßen generiert, die ab Überschreiten eines gewissen Grenzwertes rot gefärbt werden um einen Verkehrsstau anzuzeigen.

Man kann mit diesem System interagieren, indem man z.B. das Verkehrsaufkommen erhöht. Veränderten Werte können nun in die Datenbank zurückgeschrieben werden.

Dieses Beispiel steht exemplarisch für eine Reihe von vorstellbaren Anwendungen. Man kann Kundendaten jeglicher Art durch 3D-Welten visualisieren und die Datenstämme durch sinnvolle Interaktion verändern und bewerten.

5.10.3 Steuerung von E-Motoren

Um die Funktionalität von Cube4 zu erhalten wird permanent die Orientierung und Position der Kamera gesendet.

Diese Daten repräsentieren den Blick des Users in die 3D Welt. Nun können diese Daten nicht nur für die Interpretation der Kamera in der virtuellen Welt verwendet werden, sondern auch Veränderungen in der realen Welt nach sich ziehen.

Denkbar sind hier z.B. Mediensysteme bei denen die Interaktion des Users nicht nur auf einen starren Bildschirm beschränkt ist, sondern sich auch das Darstellungsmedium bewegt.

Für diesen Zweck ist die Ansteuerung von Elektromotoren durch die Ausgabeparameter der Welt denkbar. Man kann sich den Monitor als Rahmen oder Fenster in die 3D-Welt vorstellen, das sich nun durch Interaktion des Users bewegt.

Bewegt sich der User virtuell nach rechts, folgt der Bildschirm seiner Bewegung und verschiebt auch die Ansicht der 3D-Welt, also die Kamera, nach rechts.

Hierfür ist lediglich ein Abgleich der mechanischen Bewegung des Bildschirms mit der virtuellen Bewegung notwendig. Gut geeignet sind für diese Anwendung Schrittmotoren, die sich immer um eine gewisse Gradzahl pro Schritt drehen. Kameradrehungen lassen sich somit gezielt in die reale Welt übertragen.

6. Implementierung

6.1 Überblick über die verwendeten Technologien

Bevor auf die genaue Implementierung der einzelnen Komponenten eingegangen wird, soll ein Überblick über die verwendeten Technologien, im Besonderen über Blender und seine für diese Arbeit wichtigen Teile, gegeben werden.

Des weiteren schließt dieser Überblick die verwendeten und zusätzlich installierten Softwarekomponenten ein.

6.1.1 Blender

6.1.1.1 Modellierungs- und Animationstool

Blender ist ein voll funktionsfähiges, ausgereiftes Animations- und Modellierungstool, welches über die marktüblichen Features kommerzieller Softwarepakete, wie z.B. Maya von Alias Wavefront oder 3DStudio Max von Discreet, verfügt.

Die Geometrieerzeugung ist durch polygonale, Subdivision Surface- und NURBS-Modellierung möglich.

Die resultierenden Objekte können mit Materialien und Texturen (prozedural und auf Bildern basierend) versehen werden.

Blender bietet die gängigen Arten von Lichtern zur Beleuchtung der Welten an und ermöglicht Radiosityrendering.

Die Animation der Objekte erfolgt über Keyframe- und Pfadanimation, wobei die resultierenden Animationskurven weiterhin beeinflusst werden können. Zusätzlich kann die Animierbarkeit der Objekte über Skelette und Latticeboxen gezielt gesteuert werden.

6.1.1.2 Gameengine

In Blender ist die Gameengine Ketsji integriert, die es ermöglicht 3D-Szenen in Echtzeit darzustellen und begehbar zu machen.

Dies ist auch ein immenser Vorteil gegenüber kommerziellen Softwarepaketen, die solche Funktionalität gar nicht oder nur in Teilen anbieten.

Die nachträgliche Integration der Gameengine in die Blenderumgebung hat zur Folge, dass nicht alle Funktionalitäten des Editiermodus zur Verfügung stehen.

Hierzu gehört z.B. die Texturierung, die für die Echtzeit separat vorgenommen werden muss.

Dennoch gibt es viele Möglichkeiten die Welten für die Darstellung in Echtzeit vorzubereiten, die ich im Folgenden kurz beschreiben möchte.



6.1.1.2.1 Game Logik (Sensor, Controller, Actuator)

Abb. 18 : Beispiel für einen Logic Brick

In der Game Logik wird das Verhalten jedes einzelnen Objektes im Echtzeitmodus festgelegt.

Hierzu wird eine Kette von drei Komponenten, sogenannte Logic Bricks, verwendet.

Diese Komponenten sind : Sensoren, Controller und Aktuatoren. Der Sensor oder eine Kombination von Sensoren dient als Auslöser für eine gewisse Aktion.

Es gibt insgesamt 11 unterschiedliche Sensoren, die auf unterschiedliche Reize reagieren. Einige Sensoren reagieren auf Objekte und deren Entfernung, wie z.B. der Touch-, Collision-, Near-, Radar- oder Raysensor.

Zwei Sensoren reagieren direkt auf Eingaben von außen : der Keyboard- und der Mousesensor.

Der Always-Sensor erzeugt mit einer festgelegten, frameabhängigen Frequenz Impulse, während der Random-Sensor diese Impulse auf zufällige Art und Weise abfeuert.

Der Property-Sensor spricht auf eine dem Objekt eigene Variable an, während der Message-Sensor auf Nachrichten von anderen Objekten reagiert.

Die von den Sensoren erzeugten Impulse werden durch den zugehörigen Controller verbunden und weitergeleitet.

Es gibt vier unterschiedliche Möglichkeiten. OR und AND interpretieren mehrere Sensoren gleichzeitig und lösen je nach Ergebnis den oder die zugehörigen Aktuatoren aus.

Außerdem kann man als Controller ein Pythonskript ausführen, welches z.B. einen oder mehrere angefügte Aktuatoren auslöst oder andere das Objekt betreffende Aktionen ausführt.

Eine letzte Controllerart ist der Expression-Controller, mit dem man einzeilige Codestücke als Entscheidung ausführen kann.

Die eigentliche Aktion wird dann durch den Aktuator ausgeführt.

Es gibt 12 unterschiedliche Aktuatoren.

Der Motion-Aktuator erlaubt es die Bewegung eines Objektes zu beeinflussen. Dies kann sich einerseits auf die aktuelle Geschwindigkeit

und Rotation eines Objekts in allen Achsen beziehen, andererseits kann man auch Rotations- und Translationskräfte auf die Objekte wirken lassen.

Der Constraint-Aktuator bezieht sich auf die Reaktionseigenschaften eines Objektes auf bestimmte Kräfte.

Über den IPO-Aktuator können dem Objekt zugewiesen Animationen auf unterschiedliche Art und Weise abgespielt werden.

Der Camera-Aktuator ermöglicht es die Kamera zu veranlassen, einem bestimmten Objekt mit einer gewissen Verhaltensweise zu folgen.

Auf den verwendeten Sound in den einzelnen Spielszenen kann man über den Sound- und den CD-Aktuator Einfluss nehmen. Der Sound-Aktuator spielt auf ähnliche Art und Weise wie der IPO-Aktuator WAV-Sound ab, während der CD-Aktuator als CD-Spieler fungiert.

Der Property-Aktuator dient dem Zuweisen von Werten für objekteneigene Variablen, während der Message-Aktuator es ermöglicht Nachrichten oder Variablenwerte an andere Objekte zu schicken.

Ein extrem effektvoller und wichtiger Aktuator ist der Edit-Object-Aktuator. Dieser Aktuator ermöglicht das Generieren und Löschen von Objekten. Außerdem kann man den neu erzeugten Objekten eine achsenspezifische Geschwindigkeit zuweisen.

Ein weiterer mächtiger Aktuator ist der Scene-Aktuator, der es ermöglicht einzelne Szenen, die komplett unterschiedliches Verhalten und Objekte besitzen, zu überlagern, zu beenden, zu starten oder in den Anfangszustand zu bringen.

Dieser wird durch den Game-Aktuator ergänzt, der das Beenden des aktuellen Spiels und Starten eines neuen Spiels ermöglicht.

Der Random-Aktuator ermöglicht es, Zufallswerte für objekteneigene Properties zu erzeugen.

6.1.1.2.2 Texturierung und Farbgebung

Da die Gameengine erst nachträglich in Blender integriert wurde, sind für den Gamemodus einige Schritte separat zu machen. Hierzu gehört die Texturierung und Farbgebung. Dies erfolgt nicht über die Materialgebung, sondern getrennt für den Gamemodus.

Hier kann man festlegen, wie die einzelnen Faces eines Objektes auszusehen haben, jedem einzelnen Vertex eine Farbe zuweisen, das Verhalten der Faces in Bezug auf Licht festlegen und die einzelnen Faces texturieren.

6.1.1.2.3 Dynamisches System

Die Gameengine basiert auf einem physikalischen, dynamischen System, welches Blender auch für Simulationszwecke qualifiziert.

Man kann einerseits die Eigenschaften der gesamten Welt setzen, wie z.B. Gravitation und Farbgestaltung der Welt. Es können auch jedem Objekt dynamische Eigenschaften zugewiesen werden.

Hierzu gehören die Reaktionen auf Rotations- und Translationskräfte, aber auch eine generelle Masse des Objekts, welche sich in der Reaktion auf Gravitation widerspiegelt.

Das dynamische System Blenders ist bis zur Version 2.25 in das Programm integriert. Seitdem Blender ein OpenSource-Projekt ist, ist dieser Teil vorerst nicht mehr in den Quellen verfügbar, da es nicht NaN-proprietäre Software war.

Es wird aber momentan daran gearbeitet das externe OpenSource-System ODE in Blender einzufügen, welches dem alten dynamischen System an Funktionalität und Performance überlegen ist.

6.1.2 Python

Python ist komplett in Blender integriert.

Leider wird nicht die aktuellste Version Pythons unterstützt, sondern lediglich die Version 2.0.1. Dies bedeutet, man kann mit gewissen Einschränkungen den vollen Funktionsumfang von Python innerhalb Blenders nutzen, also externe, blenderfremde Pythonmodule nutzen.

Hierzu gehören unter anderem die einfache Einbindung von Datenbanken über Pythonbibliotheken, wie z.B. MySQLDB, oder Nutzung von Netzwerkbibliotheken. Außerdem kann man Module zur Bildverarbeitung einbinden.

6.1.2.1 Bildverarbeitung

Die Bildverarbeitung ist innerhalb Pythons nicht direkt implementiert.

Hierfür ist das zusätzlich zu installierende Pythonmodul PIL (Python Image Library) der schwedischen Firma Secret Labs AB notwendig, welches auf deren Seite (www.pythonware.com) gratis zum Download zur Verfügung steht.

Die Bibliothek beinhaltet, wie unter [PIL] beschrieben, einige grundsätzliche Bildberechnungsfunktionalitäten, wie z.B. Punktoperationen, verschiedene Filter, Skalierung, Rotation und Histogrammfunktion.

6.1.2.2 Webcam

Über das Modul VideoCapture von Markus Gritsch (<http://stud4.tuwien.ac.at/~e9326522/VideoCapture/>) kann man komfortabel Zugang zu einer USB-Webcam erlangen. Hierfür ist zum einen die Installation des Moduls VideoCapture notwendig, zum anderen die im

vorigen Punkt beschriebene Bibliothek PIL für die Bildverarbeitung, also vor allem das Abspeichern der erlangten Bilder.

VideoCapture bietet nun verschiedene Funktionalitäten zur Konfiguration des USB-Kameratreibers und zum Abfragen des Kamerastatus. Kameraeigenschaften und Speicherort der aufgenommenen Bilder können ebenso definiert werden.

6.1.2.3 Netzwerkfunktionalität

Python bietet das Modul „socket“ für die Kommunikation im Netzwerk an. Hierdurch ist es möglich, Sockets auf Client- und Serverseite zu öffnen und über diese Daten per TCP oder UDP zu übertragen.

6.1.2.4 Blenderbibliotheken

Bei den blendereigenen Modulen handelt es sich um Pythonklassen, die von Blender zur Verfügung gestellt werden und Funktionalitäten des Programms zugänglich machen, ähnlich wie MEL für Maya oder MaxScript für 3DStudio Max.

Diese Bibliotheken beziehen sich sowohl auf den Modellierungs- und Animationsteil Blenders als auch auf die Echtzeitengine, sind jedoch in der derzeitigen Version 2.25 voneinander getrennt.

Dies bedeutet man kann zur Laufzeit eines Blenderspiels nicht oder nur gering auf die Funktionalität des Blendermoduls zugreifen. Es ist zwar prinzipiell möglich, führt allerdings bei vielen Funktionalitäten zum Absturz oder zu nicht vorhersehbaren Ergebnissen.

Im folgenden wird etwas näher auf die Module Blender, GameLogic und Rasterizer und deren Funktionalität eingegangen.

6.1.2.4.1 Das Modul Blender

Das Modul Blender erlaubt es dem User einen Großteil der Funktionalität Blenders per Skript nachzubilden und zu erweitern.

Der User kann aktiv auf die GUI einwirken, also per Skript neue Buttons und Slider definieren und diese mit Funktionen belegen.

Außerdem kann aufwändiges Erzeugen von Objekten automatisiert werden. Hierzu zählt zum Beispiel das Generieren und Kopieren von Objekten, aber auch der Aufbau einzelner Objekte selbst kann verändert werden.

Über dieses Modul besteht Zugang auf die einzelnen Meshes jedes Objektes. Somit kann jeder Vertex einzeln angefasst und verändert werden. Ebenso können Texturen geladen und zugewiesen werden.

6.1.2.4.2 Das Modul GameLogic

Das Modul GameLogic kann nur geladen werden, wenn sich Blender im Gamemodus befindet. Hier bietet das Modul Zugang auf die gesamte Funktionalität der Game Logic.

Über den Controller, der das Pythonskript ausführt, wird das Objekt zurückgegeben, welches diesen Controller besitzt. Hierdurch ist es

möglich, das Objekt selbst zu bewegen oder dessen Orientierung zu ändern.

Dies ist von großer Wichtigkeit für die Mehrwandfunktionalität von Cube4, was später noch näher erläutert wird. Man kann aber nicht nur einige Eigenschaften des Objektes verändern, sondern hat Zugang auf die dem Objekt zugewiesenen Variablen, um Ergebnisstände zu ermitteln oder Statusänderungen an den einzelnen Objekten vorzunehmen.

Auch die einzelnen Eigenschaften der dem Controller zugewiesenen Aktuatoren können über Python gesetzt werden.

Ein weiterer Punkt ist die Geschwindigkeit der GameEngine, die sich durch das Ersetzen von LogicBricks durch Pythonskripte erhöht.

6.1.2.4.3 Das Modul Rasterizer

Das Rasterizermodul ist ebenso wie das Modul GameLogic nur im Gamemodus verfügbar.

Mit Hilfe dieses Moduls können einige Einstellungen für das Rendering vorgenommen werden.

Zum einen wird dort die Funktion des Nebels in einer Szene eingestellt, außerdem kann man das Anzeigen der Maus unterdrücken oder zulassen.

Ein weiteres Feature ist das Anfertigen von Screenshots, welches es dem User - jedoch auf Kosten der Performance - erlaubt Bilder des Spiels aufzunehmen.

Lässt man diese mit einer hohen Frequenz anfertigen, kann man mit dem integrierten Schnittsystem Filmsequenzen der Spiele erstellen.

6.1.3 Java

6.1.3.1 serielle Schnittstelle

Für Java gibt es eine zusätzlich zu installierende API zur Ansteuerung der seriellen und parallelen Schnittstelle : die Communications API.

Diese API stellt sämtliche Werkzeuge zur einfache Kontrolle und Handhabung der seriellen und parallelen Schnittstelle zur Verfügung. Hierbei kann man auf die Schnittstelle sehr hardwarenah zugreifen und sogar die einzelne Leitungen (sowohl In- als auch Outputleitungen) an und ausschalten.

Diese Funktionalität ist auch für die Implementierung des Ultraschall-Abstands-Sensors notwendig.

6.1.3.2 SNMP

Zum Auslesen der SNMP-Dienste wird auf das Java SNMP Package von Jonathan Sevy, der als Softwareingenieur an der Drexel Universität arbeitet, zurückgegriffen.

Das Package bietet eine einfache Kommunikationsschnittstelle zu einem SNMP-Dienst. Das SNMP Package stellt Klassen zum Setzen und Erhalten von Werten in der MIB (Management Information Base) zur Verfügung und bietet diese als SNMP-Datentypen an.

Hierbei kann man die Werte einzeln abfragen, oder in einem Zweig der MIB rekursiv, um Wertelisten zu erhalten.

6.1.3.3 Midifunktionalität

Java besitzt eine eigene Sound API. Diese API unterteilt sich in javax.sound.midi und javax.sound.sampled.

Mittels dieser API lassen sich vorgefertigte Sounds abspielen und verändern. Außerdem kann man über die MIDI-API MIDI-Signale erzeugen und durch einen von Java gelieferten Synthesizer interpretieren lassen.

Dieser Synthesizer greift auf ebenfalls javaeigene Soundbänke zurück um aus den MIDI-Signalen Töne zu generieren. Die Soundbänke haben einen Satz an unterschiedlichen Instrumenten, die ausgewählt werden können.

Man hat aber auch die Möglichkeit, die MIDI-Signale nicht von einem Javasynthesizer interpretieren zu lassen, sondern sie zur weiteren Verarbeitung an einen anderen Softwaresynthesizer zu schicken, wodurch sich zahlreiche Möglichkeiten zum Generieren und Verändern von Sound ergeben.

6.2 Eingabegeräte

6.2.1 Realisierung der Schnittstelle und Interaktion für Trittmatten (Boolesche Sensoren)

Die Gameengine von Blender kann auf externe Eingabegeräte, wie z.B. Tastatur und Maus, über die Sensoren Mouse und Keyboard reagieren.

Als weitere Möglichkeit ist realisierbar, Nachrichten und Informationen über TCP/IP zu schicken, die dann innerhalb Blenders interpretiert werden.

Hier hat man den entscheidenden Nachteil, dass Blender für jede offene Socketverbindung die Performance, also die Anzahl der pro Sekunde dargestellten Frames, halbiert, weswegen es sinnvoll ist, nur einen Port zu öffnen.

Daher muss man versuchen, die Interaktion direkt auszuführen, was mittels Tastendrucke möglich ist. Sämtliche Booleschen Sensoren funktionieren wie Tasten, da sie nur zwei Zustände kennen, nämlich Stromkreis geschlossen oder offen.

So lag es nahe, auch die Schnittstelle der Tastatur, den Tastaturcontroller, zu nutzen. Hierzu wurde ein Tastaturcontroller einer handelsübliche PS2-Tastatur ausgewählt und auf seine Funktionsweise getestet.

Abb. 19 : Auszug einer selbstbestimmten Tastaturmatrix; hervorgehoben ist eine der Tastenreihen, in der alle Tasten gleichzeitig gedrückt werden können

	16	17	18	19	20	21	22	23	24
1									
2									
3			.		k	+	a	8	
4		n	m	h	j	=	u	7	6
5		b	v	c	f	t	r	4	5
6			e		d		e	3	
7			z	<		Sh	w		1
8			y		a	ft	o	2	
9									
10									
11									
12		*			em		+		
13		-	*		3	6	9		
14				0	1	4	7		
15			/	0	2	5	8		

Abb. 20 : Aufbau des Prototyps der Trittmattensteuerung



Der Tastaturkontroller verfügt über 25 Kontakte durch die eine Matrix aufgebaut werden kann. Dies bedeutet, schließt man zwei Kontakte kurz, was in der Praxis durch einen Tastendruck geschieht, wird dieses über einen Interrupt dem System als ASCII-Zeichen übergeben.

Jetzt kann man die Tasten durch jegliche Art von Booleschen Sensor ersetzen, um neue Interaktionsschnittstellen zu schaffen.

Hierzu verbindet man die Kontakte des Controllers mit einer Platine auf der eine Matrix aufgebaut ist, die der Tastaturmatrix entspricht.

Ein Problem hierbei ist, dass dem gleichzeitigen Drücken mehrerer Tasten eine Grenze gesetzt ist. Dies kann man beobachten, indem man z.B. drei Cursortasten gleichzeitig drückt, was den Rechner zur Abgabe eines Warnsignals veranlasst und zum anderen dazu führt, dass die Tastendrucke nicht weiter interpretiert werden. Dies hat mit der Abstimmung der Tastaturmatrix zu tun.

Durch weiteres Ausprobieren ließ sich erkennen, dass Tasten die in einer Zeile der Tastaturmatrix liegen gleichzeitig gedrückt werden dürfen und auch interpretiert werden. Dies ermöglicht es bis zu 8 Tasten gleichzeitig zu drücken.

Eine mögliche Kombination sind hier die Tasten 4,5,R,T,F,G,V und B, die wie man auf der Tastatur erkennen kann auch einen räumlichen Bezug zueinander haben. 8 Tasten sind nun meist für das erste Einsatzgebiet, eine Steuerung der 3D-Welten, ausreichend, genügen aber für eine Positionsbestimmung mehrerer User in großen Räumen nicht.

Ein möglicher Lösungsansatz hierfür sind USB-Tastaturen, da diese parallel zur PS2-Tastatur betrieben werden können und der Abfragemechanismus der Tastaturmatrix anders ist, da diese kein Signal des Rechners beim gleichzeitigen Drücken mehrerer Tasten auslösen.

Diese ist jedoch zum gegenwärtigen Zeitpunkt nicht realisiert.

6.2.2 Abstandsmessung

6.2.2.1 I²C-Bus

Der I²C-Bus wurde - wie in [Götz 01] beschrieben - von Philips zur einfachen Kommunikation zwischen Unterhaltungsgeräten, wie z.B. Fernsehgeräten und CD-Playern, entwickelt.

Hierbei soll er konkret die Kommunikation zwischen ICs (integrierten Schaltkreisen) vereinfachen, die diesem Bus auch den Namen gegeben hat : Inter IC = I²C.

6.2.2.1.1 Eigenschaften des I²C-Buses

Der I²C-Bus ist ein „bidirektionaler 2-Draht-Bus, der eine serielle, synchrone Datenübertragung in 8-Bit-Blöcken mit Geschwindigkeit bis zu 100 kBit/s im Standardmodus (und 3,4 MBit/s im High-speed mode) über Kabellängen von mehreren Metern zulässt“ [Götz 01].

Über den I²C-Bus kann man bis zu 16 verschiedene Geräte gleichzeitig über die Broadcastadresse oder einzeln direkt ansprechen.

Die Geräte müssen jedoch unterschiedliche Adressen besitzen, da es sonst zu Kollisionen kommt. Wie schon angedeutet, gibt es eine klare Rollenverteilung im I²C-Bus, den Master und den Slave.

Master und Slave können jedoch unterschiedliche Rollen einnehmen, nämlich die des Senders und des Empfängers, was im folgenden näher erläutert wird.

Der I²C-Bus besitzt zwei Leitungen : SDA, die Leitung, auf der die Daten geschickt werden und SCL, die Leitung, die dem Synchronisieren der einzelnen Teile der Kommunikation dient.

6.2.2.1.2 Aufbau von I²C-Nachrichten

Es gibt zwei unterschiedlich Arten von I²C-Botschaften : den lesenden und den schreibenden Zugriff des Masters auf den Slave. Beide Arten werden im folgenden Schritt für Schritt nach [Götz 01] erläutert :

Der Schreibvorgang :

1. Das Herstellen der Start-Bedingung durch den Master
2. Der Master sendet die Adresse des Slaves (7 Bit) für den die Nachricht bestimmt ist. Das letzte Bit dient der Bestimmung der Zugriffsart und wird auf 0 gesetzt für schreibend.
3. Der Slave sendet ein Acknowledgement-Bit mit dem er seine Bereitschaft Daten zu empfangen signalisiert.
4. Der Master sendet jeweils ein Byte, das jeweils vom Slave durch ein Acknowledgement-Bit quittiert wird.
5. Sind alle Daten übertragen beendet der Master die Kommunikation durch das Herstellen der Stop-Bedingung.

Der Lesevorgang :

1. Der Master stellt die Start-Bedingung her
2. Der Master sendet die Adresse des Slaves (7 Bit) für den die Nachricht bestimmt ist. Das letzte Bit dient der Bestimmung der Zugriffsart und wird auf 1 gesetzt für lesend.
3. Der Slave sendet ein Acknowledgement-Bit mit dem er seine Bereitschaft Daten zu senden signalisiert.
4. Der Slave sendet jeweils ein Byte, das jeweils vom Master durch ein Acknowledgement-Bit quittiert wird.
5. Abschließend sendet der Master ein Not-Acknowledgement-Bit, um zu signalisieren, dass er keine weiteren Daten empfangen möchte, und beendet die Kommunikation durch das Herstellen der Stop-Bedingung.

6.2.2.2 Beschreibung des verwendeten Sensormoduls

Der SRF08 ist ein äußerst hoch entwickelter Ultraschallsensor, der über das I²C-Protokoll angesprochen wird. Er kommt in der Robotik z.B. als Hinderniswarner zum Einsatz.

Das Sensormodul wird auf [US-Sensor] wie folgt beschrieben.

Das Sensormodul hat eine theoretische Reichweite von 11 m, ist jedoch aufgrund der Komponenten auf ca. 6 m begrenzt.

Zusätzlich ist ein Helligkeitssensor auf dem Modul angebracht, welcher Werte zwischen 0 und 255 liefert.

Jedes Sensormodul verfügt über eine eigene Adresse, die bei jeder Anfrage verwendet wird, da man über das I²C-Protokoll mehrere Sensormodule über einen Bus ansprechen kann.

Der Sensor verfügt über 36 Register, von denen 3 beschreib- und lesbar sind. Die restlichen Register sind lediglich lesbar.

Im Schreibmodus legt man im Register 0 die Befehle ab, die Verstärkung des ausgesandten Signals kann über das Register 1, die Reichweite über das Register 2 konfiguriert werden.

Im Befehlsregister kann man u.a. eine Messung mit verschiedenen Ergebnisangaben (Zoll, Zentimeter oder Mikrosekunden) auslösen oder die Adresse des Sensors ändern.

Im Register 1 kann im Lesemodus das Ergebnis der Lichtmessung ausgelesen werden. Die Abstandsmessergebnisse befinden sich im Register 2 und 3.

Das Ergebnis wird durch einen vorzeichenlosen 16bit-Wert repräsentiert, wobei sich im Register 2 das höherwertige Byte und in Register 3 das niederwertige Byte befindet.



Abb. 21 : Ultraschallsensor mit Interfacekarte

Die übrigen Register enthalten weitere Messwerte, die durch Mehrfachechos entfernterer Objekte entstehen.

Das Sensormodul verfügt über eine an der Oberseite angebrachte LED, die zum einen beim Einschalten des Sensors dessen Adresse durch einen Blinkcode aus langen und kurzen Impulsen zeigt und zum anderen bei jeder Messung einen kurzen Lichtimpuls schickt.

6.2.2.3 Interface seriell/I²C-Bus

Um den Sensor durch den Computer steuern und ansprechen zu können, müssen beide miteinander verbunden sein.

Dies erfolgt durch eine Interfacekarte, die den Rechner auf der einen Seite über die serielle Schnittstelle mit dem Sensor auf der anderen Seite über die zwei Leitungen SDA und SCL verbindet. Als Interfacekarte wurde hier ein Nachbau einer in der Technikzeitschrift „Elektor“ vorgestellten Karte verwendet.

Das Interface hat im wesentlichen zwei Funktionen [Gerlach 01]. Zum einen das Gewährleisten des korrekten Pegels des Signals, da die serielle Schnittstelle mit +/- 9 Volt, der I²C-Bus jedoch mit lediglich 5 Volt betrieben wird und zum anderen das Durchschleifen der beiden Leitungen (SDA und SCL) zur Datenübertragung.

Das Interface verfügt über eine eigene Stromversorgung, kann aber auch durch die I²C-Applikation gespeist werden.

6.2.3 Webcam

Die Webcam wird durch ein Pythonskript ausgelesen. Dies wurde notwendig, da die gängigen Webcam-Applikationen meist das Erstellen von Bildern nur einmal pro Sekunde vorsehen und nicht wie benötigt mindestens 15 mal pro Sekunde.

Zuerst wird das Modul VideoCapture geladen. Nun wird der USB-Anschluss und somit die Webcam für das Pythonskript reserviert und ist somit nicht mehr für andere Applikationen zugänglich.

Die Kamera ist jetzt ansprechbar und fertigt durch einen Befehl Bilder in einer Endlosschleife an. Diese Bild können nun innerhalb Blenders durch die Pythonerweiterung PIL (Python Image Library) der Firma Pythonware geladen und interpretiert werden.

Abb. 22 : Webcam als Interaktionsgerät



Die Webcam kam bereits als Interaktionsgerät auf dem fünftägigen Kongress „Urban Drift“ zum Einsatz. Hierbei wurde ein Mediensystem mit fünf Monitoren installiert. Dies ist der jüngste Beitrag der „Instant Architecture“-Reihe.

Ein Monitor zeigte permanent das Bild, das von der Webcam aufgenommen wurde, die anderen vier Monitore interpretierten das Bild in einer Blenderwelt.

Hierbei wurde das Bild innerhalb Blenders mit Hilfe des PIL-Moduls geladen, anschließend verkleinert und in Graustufen umgewandelt. Nun hatte jeder Pixel des Bildes einen Repräsentanten in der 3D-Welt. Dieser Repräsentant war ein zweifarbiger Kubus.

Nun wurde jeder Kubus anhand des zugehörigen Pixelwertes skaliert und verschoben, wodurch eine Art Nadelkisseneffekt (siehe Abbildung 12) entstand. Diese Welt wurde aus verschiedenen Blickwinkeln gezeigt. Der User konnte also durch seine Anwesenheit die Kubenwelt beeinflussen und gestalten.

6.3 Pythonskripte innerhalb Blenders zur Bildgenerierung

6.3.1 Synchronisation der Clients

Die Kameras der einzelnen Clients, durch die die virtuelle Welt betrachtet wird, müssen sich stets an derselben Positionskoordinate in der 3D-Welt befinden, damit eine lückenlose Darstellung gewährleistet wird.

Um dies zu erreichen wird ein Nullobjekt, ein sogenanntes Empty, generiert, an welches die Kameras angehängt werden. Bewegt sich das Empty, wird auch der Blick auf die 3D-Welt verändert. Die Bewegung, also die Veränderung der Position des Emptys muss nun permanent allen Clients mitgeteilt werden.

Hierzu öffnet der Server einen Socket, über den er die Position und die Orientierung des Kameraemptys durch Broadcasting verschickt.

Jeder Client besitzt dieses Kamerempty, an dem die jeweilige Kamera mit dem richtigen Blickwinkel angehängt ist. Die Position umfasst jeweils die Raumkoordinate (x, y, z) des Punktes.

Somit ist gewährleistet, dass das Kameraempty immer an der richtigen Position ist. Diese Daten werden vom Server jeden Frame gebroadcastet.

Man verwendet Broadcasting um die höchstmögliche Geschwindigkeit zu gewährleisten und eine permanente Aktualisierung zu erreichen.

Dies bedeutet es müssen nicht alle Pakete der Reihe nach abgearbeitet werden, sondern es wird immer nur das aktuellste verwendet. Somit fällt ein eventueller Verlust von einem oder mehreren Paketen nicht sehr ins Gewicht, da das System automatisch wieder die aktuellste Information bekommt.

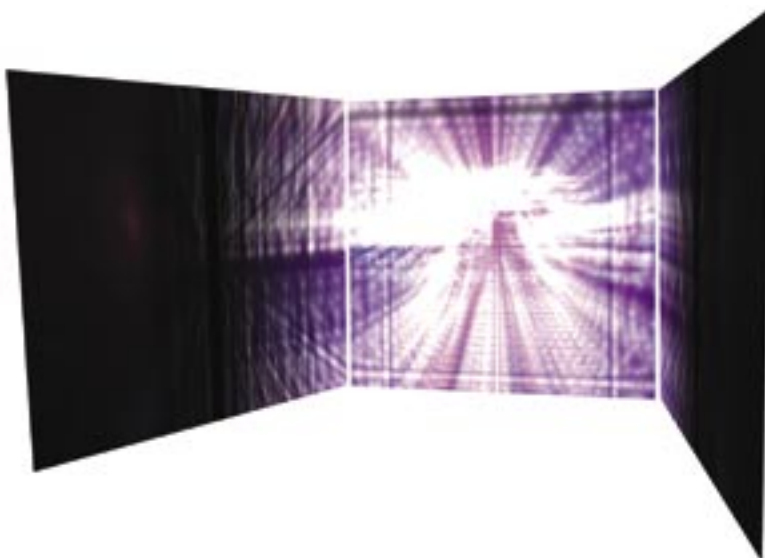
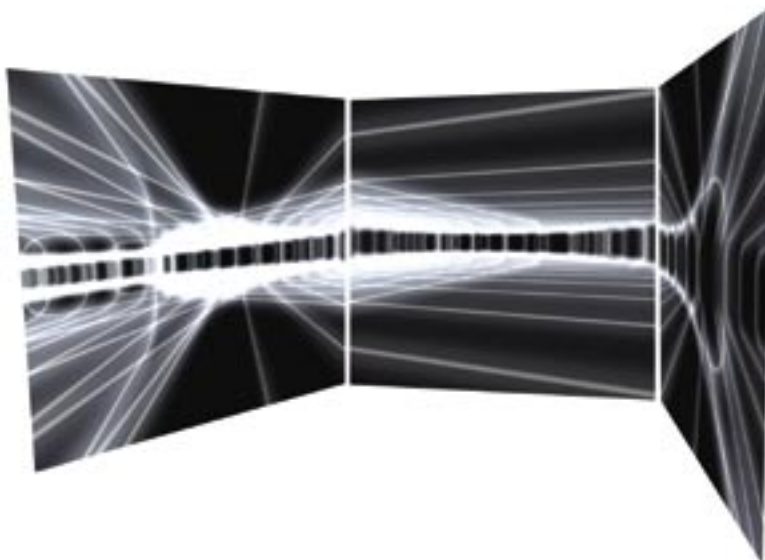


Abb. 23 : jeweils 3 Screenshots der Renderclients von Cube4 perspektivisch dargestellt

Ein weiterer wichtiger Punkt ist die Übertragung der Orientierung. Die Orientierungsvariable des Kameraempty besteht aus einer 3 x 3 Matrix, die die räumliche Lage des Emptys in allen drei Achsen beschreibt.

Diese Information wird z.B. bei einem seitlichen Wegkippen im Raum benötigt.

Ein Problem bei der Synchronisation ist die Echtzeitberechnung. Dies bedeutet die einzelnen Clients berechnen jeweils einen Frame und stellen diesen dar.

Hierbei hat man keinen Einfluss auf die Dauer der Berechnung, also kann es sein, dass ein Client äußerst viele Objekte mit komplexen Texturen darstellen muss und die anderen Rechner durch ihre Kamera keine Objekte sehen.

Dies führt zu unterschiedlichen Berechnungszeiten bei den einzelnen Clients. Hierauf hat man leider keinen direkten Einfluss und muss dies in der Erstellung der Welten berücksichtigen, damit eine möglichst gleichmäßige Darstellung gewährleistet ist.

Eine Möglichkeit die framebasierte Darstellungs-geschwindigkeit zu deckeln gibt es. Dies erfolgt durch Einstellung der Herzzahl der Graphikkarte über den Graphikkartentreiber. Es ist also speziell darauf zu achten, dass bei allen Clients und dem Server die gleiche Herzzahl verwendet wird.

Außerdem wählt man grundsätzlich eine geringe Herzzahl (meist 60 Hz), da dies zum einen Ressourcen der Clients schont, also einem Überlaufen oder Verzögern entgegenwirkt, und zum anderen die Darstellungsmedien, wie z.B. Beamer oder Plasmabildschirme, meist mit geringen Herzzahlen getaktet sind.

6.3.2 Verteilung der Tastenevents

Zusätzlich zu der Position und der Ausrichtung der Kamera müssen auch die einzelnen Events, die Einfluss auf die Darstellung der Welt haben, übertragen werden.

Hierzu gehören u.a. das Erzeugen von neuen Objekten und jegliche Art von Bewegung schon bestehender Objekte, die nicht bei Entstehung dieser schon vorgegeben ist und nicht mehr verändert wird.



ADD property					
Del	Bool ▾	Name:InitS	True	False	D
Del	String ▾	Name:state			D
Del	String ▾	Name:keys	0000000000		D

Abb. 24 : Stringproperty zum Versenden von Tastenevents

Die Darstellung der Welt wird stets entweder über den Aufruf eines Aktuators oder ein Pythonskript verändert. Bei der Verteilung der Events bedient man sich dieses Mechanismus, indem man stets nur das auslösende Event und nicht die resultierende Aktion verschickt.

Dies möchte ich am folgenden Beispiel erläutern : Man kann in der Echtzeitengine mittels eines Tastendrucks durch den Edit Object-Aktuator ein Objekt mit einer gewissen Frequenz erzeugen.

Dies bedeutet solange eine Taste gedrückt bleibt wird z.B. alle 20 Frames ein weiteres Objekt erzeugt.

Nun hat man in einer Welt z.B. 10 verschiedene Tasten zum Erzeugen ebenso vieler unterschiedlicher Objekte definiert. Nun kann entweder die Information, dass ein Objekt erzeugt wurde an die Clients weitergegeben werden oder dass eine Taste gedrückt ist.

Der erste Weg hat den entscheidenden Nachteil, dass die Weitergabe der Information des Erzeugens eines Objektes in durch Broadcasting erfolgt, also das Ankommen des Paketes nicht garantiert ist. Wenn diese Information nicht den Client erreicht, wird das Objekt nicht dargestellt.

Alternativ hierzu wird nicht die Information „Objekt generiert“, sondern die Information „Taste gedrückt / nicht gedrückt“ verschickt. Auf Seite des Clients werden nun die Pakete empfangen und mit einer gewissen Frequenz neue Objekte erzeugt.

Dies bedeutet bei einer Frequenz von 20 müssen 20 Datenpakete mit der Information „Taste gedrückt“ ankommen, dann wird das Objekt generiert.

Dies verringert die Problematik des Nicht-Generierens von Objekten, da das Objekt bei eventuellen Paketverlust einen oder zwei Frames später generiert wird, was dem Betrachter nicht auffällt.

Für das Versenden der Information „Taste gedrückt/nicht gedrückt“ wird nun serverseitig ein String mit ebenso vielen Stellen wie zu übertragende Interaktionsevents erstellt. Sind z.B. drei Tasten definiert, lautet der String „000“. Die erste Stelle entspricht nun z.B. der Taste A, die zweite der Taste S, usw.

Nun wird das zugehörige Skript serverseitig zu jedem Frame aufgerufen, prüft welche Taste gerade gedrückt ist und setzt die dementsprechende Stelle des Strings auf 1.

Dieser String wird nun dem Datenpaket, in dem die Position und die Orientierung gespeichert sind, hinzugefügt und an die Clients geschickt.

Bei den Clients wird nun das Paket entpackt und interpretiert. Es müssen hier für jedes Event zwei Variablen zugewiesen werden, um das Event mit der richtigen Frequenz auslösen zu können : eine mit der benötigten Frequenz und eine mit dem aktuellen Zählerstand.

Hat der Zählerstand die Frequenz erreicht, löst das Skript den zugehörigen Aktuator aus.

6.3.3 Objektkontrolle

Das im vorigen Punkt erwähnte Erzeugen von Objekten ist dann sinnvoll, wenn sich die Objekte nach der Generierung nicht mehr bewegen und nicht mehr mit Ihnen interagiert werden kann. Dies ist bei den „klassischen“ Datenflügen der Fall.

Möchte man jedoch weiterhin Einfluss auf diese Objekt haben, reicht das Erzeugen des Objektes nicht aus.

Nun ist es notwendig stets die Position und Orientierung von jedem Objekt in der Szene zu kennen und zu beeinflussen. Hierfür melden sich die Objekte bei der Kreierung bei einer globalen Liste der GameLogic an und geben ihren Namen bekannt.

Über diesen Namen kann nun jederzeit der Ort und die Orientierung des Objektes erfragt werden und in einer Liste abgespeichert werden.

Diese Liste wird nun vom Server an die Clients verschickt, die nach dieser Liste zum einen kontrollieren können, ob die Objektanzahl mit der Zahl der vom Client dargestellten Objekten übereinstimmt und zum anderen die Position und Orientierung der Objekte im Bedarfsfall korrigieren können.

Stellt sich heraus, dass ein Objekt hinzugekommen ist, wird dies bei den Clients erzeugt und an die richtige Position gesetzt.

Hierüber kann man in etwa 40 Objekte mit einer angenehmen Performance steuern und kontrollieren.

Bei mehr als 40 Objekten nimmt die Anzahl der Frames pro Sekunde stark ab und wird inakzeptabel.

6.4 Die Java-Anwendung Cube4J

6.4.1 GUI

Für die GUI-Programmierung des Administrationstools wurde auf die Swingklassen zurückgegriffen. Die GUI besteht im wesentlichen aus drei Themengebieten.

Im ersten Menüpunkt kann man den Ultraschallsensor starten und kalibrieren. Zum Kalibrieren gehört die Einstellung der Reichweite, der Verstärkung und der Adresse des Sensors.

Das zweite Themengebiet ist die Administration von Cube4 und die Überwachung der Stabilität. Jeder Rechner im Netzwerk, der eine Freigabe besitzt, kann ausgewählt und eine Position in Cube4 zugewiesen werden. Daraufhin kann in dieser Konfiguration Cube4 mit dem eingegebenen File gestartet oder später ein Filewechsel veranlasst werden.

Die Überwachung der Stabilität erfolgt automatisch und wird in einem Textfeld angezeigt.

Im dritten Themengebiet kann die MIDI-Ausgabe gestartet und angehalten werden.

6.4.2 Basisklassen

6.4.2.1 Rechnerinfo

Die Klasse „Rechnerinfo“ dient dem Auffinden der verfügbaren Rechner im Netzwerk. Hierzu wird über die Methode „netview“ eine Liste der im Netzwerk vorhandenen Rechner zur Verfügung gestellt.

Das Auffinden der Rechner im Netzwerk wird über ein DOS-Kommando realisiert, das durch Java aufgerufen und interpretiert wird.

Der Befehl NET VIEW gibt den Namen der Rechner zurück, die über eine Freigabe verfügen. Die Ausgabe von NET VIEW wird nun innerhalb der Methode netview bereinigt und in eine Liste abgelegt.

Dies ist eine relativ komfortable Lösung, da man nun die Namen aller erreichbaren Rechner über einen sehr performanten Befehl verfügbar hat.

6.4.2.2 Client

Die Klasse „Client“ bietet Funktionalität zum Empfangen von Daten.

Durch die Klasse „Client“ wird ein Datagramm-Socket auf einem bestimmten Port geöffnet. Über die Methode „recData“ können Daten empfangen werden. Die Methode „close“ schließt den Socket.

6.4.2.3 Server

Durch die Klasse „Server“ können Daten verschickt werden.

Die Klasse „Server“ generiert einen DatagramSocket der über die Methode „connectSocket“ mit einem Port verbunden werden kann. Die Methode „disconnectSocket“ kann diese Verbindung wieder lösen. Durch „sendData“ verschickt das Server-Objekt Integerdaten.

6.4.2.4 Port

Die Klasse „Port“ bietet Funktionalität zum Ansprechen der seriellen Schnittstelle. Hierzu wird zuerst die serielle Schnittstelle reserviert und kann anschließend durch die Methoden „open“ und „close“ geöffnet und geschlossen werden.

6.4.2.5 I²C

Die Klasse „I²C“ bietet Funktionalität zum Ansprechen des Ultraschallsensors über die serielle Schnittstelle durch das I²C-Protokoll.

Methodenübersicht :

close : schließt den seriellen Port

open : öffnet den seriellen Port

iicMessenCM : löst eine Messung aus

iicAdresseAendern : ändert die Adresse des Ultraschallsensor

iicVerstaerkungAendern : ändert die Verstärkung des vom Ultraschallsensor gesendete Signals

iicMessergebnisDistanz : liefert das Abstandsmessergebnis

iicMessergebnisLicht : liefert den ermittelten Lichtwert

iicReadByte : liest ein Byte vom I²C-Bus

iicWriteByte : schreibt ein Byte auf den I²C-Bus

iicStart : stellt die Startbedingung für die I²C-Nachricht her

iicStop : stellt die Stopbedingung für die I²C-Nachricht her

6.4.2.6 SNMP

Die Klasse SNMP ermöglicht es Verbindung mit einem SNMP-Dienst auf einen entfernten Rechner aufzunehmen und von diesem durch die Methode „readMIBItem“ Informationen über den Rechner durch Auslesen des Eintrages in der Management Information Base zu beschaffen.

6.4.2.7 Midi

Die Klasse „Midi“ lädt den Defaultsynthesizer des Systems und wählt daraus ein Instrument, auf dem anschließend gespielt werden kann.

Zum Spielen einer Note wird die Methode „spielNote“ aufgerufen, der man den Wert der Note und die Anschlagstärke übergibt.

6.4.3 Starten von Cube4

Cube4J ermöglicht auch den Start von Cube4. Hierbei werden aus der bereits ermittelten Liste von verfügbaren Rechnern im Netzwerk, die einzusetzenden ausgewählt und der richtigen Position zugewiesen.

Dies bedeutet jeder Rechner muss das korrekte File laden, da sich die Kameraeinstellungen in den einzelnen Files unterscheiden.

Die Clients starten nach dem Bootvorgang automatisch ein kleines Javaprogramm im Hintergrund, welches lediglich einen Port öffnet und auf die Anweisungen des Servers wartet, welches Blenderfile zu laden ist.

Der User wählt nun im Administrationstool für jeden Rechner das richtige File aus. Der Name und Ort des Files wird dem dazugehörigen Client geschickt.

Dieser kann nun das File über das Filesystem mit der eigenen Blenderpublisher-Version starten, was über einen DOS-Konsolenaufruf durch Java erfolgt.

Es gibt einige Aspekte in der Vorbereitung zu beachten.

Der Ordner, der die zu verwendenden Files enthält muss für alle Clients freigegeben sein. Auf den Clientrechnern muss das Javaprogramm gestartet und Blenderpublisher, Python und alle benötigten Pythonzusatzmodule installiert sein.

6.4.4 Filewechsel

Das Wechseln der Files ist dem Ablauf des Startens von Cube4 sehr ähnlich, die Auswahl der Rechner entfällt jedoch, da dies bereits im Vorfeld geschehen ist. Es ist also lediglich die Angabe des neu zu ladenden Files notwendig.

Der Ort des neuen Files wird den einzelnen Clients bekannt gegeben. Diese müssen nun zuerst für das Beenden des laufenden Blenderfiles sorgen. Normalerweise wird ein Spiel durch das Drücken der Escape-Taste beendet.

Da dies nicht physisch möglich ist wird diese Aufgabe von Java übernommen. Über die Robot-Klasse lassen sich Tastendrücke simulieren. Daher wird für einen gewissen Zeitraum, z.B. eine halbe Sekunde, die Escape-Taste gedrückt, was das Beenden des laufenden Blenderspiels nach sich zieht.

Anschließend wird durch den DOS-Systemaufruf das neue Blenderfile gestartet und der Filewechsel ist vollzogen.

6.4.5 Softwareseitige Prozesskontrolle durch SNMP

Mit dem Starten von Cube4 wird auch die Prozesskontrolle aktiviert, die in einem eigenen Thread abläuft. Beim Starten wurde festgelegt, welche Rechner die Darstellung der 3D-Welten übernehmen.

Auf diesen Rechnern muss der SNMP-Dienst installiert und aktiv sein.

Über die IP-Adresse, die in Java automatisch aufgelöst wird, können nun die vom SNMP-Dienst bereitgestellten Daten ausgelesen werden. Dies wird zentral vom Server für alle Rechner vorgenommen, ohne dass weitere Software auf den Clients läuft.

Zusätzlich zur IP-Adresse ist zum Zustandekommen der Verbindung eine Art Passwort, der sogenannte community-Parameter anzugeben. Dieser Parameter muss für jeden einzelnen Rechner bekannt sein, ist aber standardmäßig für den Lesemodus auf den Wert „public“ gesetzt.

Ist die Verbindung hergestellt, kann man Informationen über den aktuellen Status des Clients beziehen, indem man den zugehörigen Wert aus der Management Information Base (MIB) ausliest. In der MIB sind die einzelnen Werte in einer großen Zweigstruktur abgelegt. In dem Zweig „iso.org.dod.internet.mgmt.mib-2.host“, der dem numerischen Äquivalent „1.3.6.1.2.1.25“ entspricht (siehe Abbildung 7), finden sich zahlreiche Informationen zum Hostsystem.

Man kann Details über das installierte System, den Speicher und die betriebenen Geräte abfragen. Zusätzlich sind Informationen über die installierte Software und deren Status zu erhalten.

Wichtig für Cube4 ist der Zustand der laufenden Prozesse. Hierfür muss erst unter „iso.org.dod.internet.mgmt.mib-2.host.hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunName“ (.1.3.6.1.2.1.25.4.2.1.2) die ID des laufenden Blenderprozesses ermittelt werden.

Der Status dessen kann nun unter „iso.org.dod.internet.mgmt.mib-2.host.hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunStatus“ (.1.3.6.1.2.1.25.4.2.1.7) kontrolliert werden.

Wird der Prozess nun nicht mehr ausgeführt, muss man ihn entfernen und das richtige Blenderfile nochmals starten.

Falls der SNMP-Dienst auf eine Anfrage nicht mehr verfügbar ist, ist anzunehmen, dass das ganze System eingefroren ist, was einen Neustart des Rechners nach sich ziehen muss.

6.4.6 Starten und Kalibrieren des Sensors

Für das Starten des Sensors wird zuerst ein Port ausgewählt, über den die Messdaten an Blender verschickt werden, oder es wird auf das Erzeugen von Tastenevents für Blender geschaltet.

Anschließend wird die Adresse des Sensors ausgewählt. Durch Druck des Start-Buttons beginnt die Messung.

Die Messergebnisse werden in einem eigenen Thread ermittelt und durch einen Balken angezeigt.

Diese Messergebnisse werden durch Broadcasting an Blender geschickt und könne dort z.B. durch Translation oder Rotation von Objekten interpretiert werden. Eine weitere Möglichkeit der Interaktionsweitergabe ist das Erzeugen von abstandsabhängigen Tastendrücken durch die Robotklasse. Diese Tastendrücke könne dann ebenso von Blender als Eingabe interpretiert werden.

Des weiteren werden die Messergebnisse für Abstand und Lichtwert innerhalb von Cube4J numerisch dargestellt.

In einem weiteren Fenster kann man den Sensor testen und Kalibrieren. Hier kann man die Adresse, über die der Sensor angesprochen wird, permanent verändern, was beim Einsatz mehrerer Sensoren wichtig ist, um sie unterscheiden zu können.

Außerdem kann man temporär die Einstellungen für die Verstärkung und die Reichweite des Sensors verändern. Diese Einstellungen gehen aber nach einem Neustart des Sensors verloren.



Abb. 25 : Fenster zum Starten und Kalibrieren des Ultraschallsensors

6.4.7 MIDI

Im Bereich MIDI ist derzeit nur das Starten und Stoppen vorgesehen.

Hierfür wird der Standardsynthesizer geladen und ein Instrument ausgewählt.

Außerdem wird ein Clientobjekt erzeugt, das auf Daten, die von Blender (Position, Orientierung und Tastenevents) geliefert werden, wartet. Diese Daten werden dann interpretiert und in Notenwerte konvertiert, die durch das Midi-Instrument gespielt werden.

6.5 Stabilität

6.5.1 Hardwareüberwachung durch die PCI-Karte Watchdog

Die PCI-Karte Watchdog der Firma Berkshire Products Inc. ist eine Hardwarelösung zur Überwachung der Verfügbarkeit von PCs.

Sie verfügt über einen eigenen Prozessor und kommuniziert mit einer auf dem Rechner installierten Software.

Man kann unter anderem die Temperatur des Prozessors und die Verfügbarkeit des Systems überwachen. Steigt die Temperatur über einen eingestellten Schwellwert oder sendet der PC über eine bestimmte Zeit keine Signale, löst die Watchdog-Karte einen Neustart des Systems aus.

Dies wird durch ein Durchschleifen des Reset-Buttons des PCs realisiert, der dann über ein Relais geschlossen wird. Liegt eine Temperaturüberschreitung vor, wird das System solange im Reset-Status gehalten bis die Prozessortemperatur wieder signifikant unterhalb des Schwellenwertes liegt.

Durch die Watchdog-Karte ist somit sichergestellt, dass ein Einfrieren des Computers auf Ausstellungen nicht von Dauer ist, sondern durch einen automatischen Neustart behoben wird.

Dies ist insbesondere bei großen Plasmabildschirmen wichtig, da sich lange statische, kontrastreiche Bilder einbrennen können, was den Plasmabildschirm unbrauchbar macht.

6.5.2 Ersatz bei Ausfall

Bei großen permanenten Installationen, kann man einen oder mehrere zusätzliche Rechner bereithalten, um die Betriebssicherheit und Verfügbarkeit des Systems zu erhöhen.

Diese Rechner können relativ nahtlos an die Stelle ausgefallener Rechner treten. Ein bestimmter Rechner wird entweder durch die Watchdog-Karte oder die SNMP-Überwachung als ausgefallen detektiert.

Dadurch ist die Funktion, welche Seite der ausgefallene Rechner mit welcher Welt darstellt, sehr leicht nachzuvollziehen. Nun wird das korrekte File auf dem neuen Rechner gestartet. Diese Funktionalität kann leicht durch die Startroutine von Cube4 abgedeckt werden.

Ein größeres Problem stellt die Hardware dar, da nun auch der neue Rechner physisch mit dem Monitor oder dem Beamer des ausgefallenen Rechners verbunden werden muss.

Hierzu kann man eine sogenannte Kreuzschiene verwenden. Diese ermöglicht es das von den Rechnern erstellte Graphiksignal variabel auf die verschiedenen Ausgabegeräten umzuschalten.

Hierbei können die Kanäle per Knopfdruck verändert und die einzelnen Signale ausgetauscht werden.

Manche Kreuzschienen verfügen außerdem über eine RS232-Schnittstelle, die es möglich macht, das Zuweisen der Signale zu den Ausgabegeräten softwareseitig zu regeln, wodurch nun der Austausch des Rechners gewährleistet werden kann.

6.6 Administration

6.6.1 Automatischer Start

Der automatische Start von Cube4 erfolgt direkt und automatisch, ohne dass ein Operator die einzelnen Files auf jedem Client und dem Server von Hand starten muss.

Hierfür wird jedem Rechner im Autostartordner das zu startende Blenderfile und die dazugehörige Version des Blenderpublishers mitgeteilt. Hierdurch wird nach dem Hochfahren des Systems das korrekte Blenderfile gestartet.

Das Blenderfile selbst muss so konfiguriert sein, dass es automatisch im Gamemodus startet. Hierfür wird die Option „Autostart“ im Menuepunkt „Game“ aktiviert und das File abgespeichert. Dies bewirkt das direkte Starten des Files.

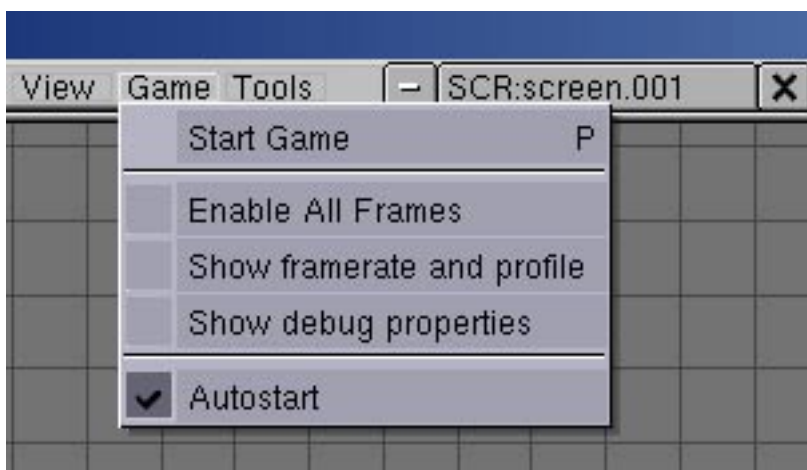


Abb. 26 : Autostarteinstellungen in Blender

Die Methode des automatischen Startens ist auch für das ganze Clustersystem möglich, da die Clients nach dem Start in eine Wartestellung gehen und auf Eingaben des Servers warten. In diesem Status sind die Clients blockiert und erst wieder reaktionsfähig, wenn der Server die ersten Datenpakete geschickt hat.

Ein weiterer Punkt ist das Anschalten der Rechner. Gerade auf Ausstellungen sind die Rechner nicht gut zugänglich, weswegen ein Anschalten jedes Rechners einzeln per Hand oft nicht möglich oder sehr schwierig ist und einen großen Aufwand bedeutet. Um dies einfacher zu gestalten, kann man im Bios das automatische Booten beim Erhalten von Strom aktivieren.

6.6.2 Filewechsel durch Blender

Der Filewechsel auf Client und Server kann relativ einfach mittels des Game-Aktuators vollzogen werden.

Hierzu wird der Aktuator sowohl auf den einzelnen Clients als auch auf dem Server verwendet. Der Aktuator bekommt lediglich das neu zu startende Blenderfile.

Nun muss noch gewährleistet sein, dass sämtliche Clients über den Filewechsel informiert werden.

Hierbei wird die Information des Filewechsel auslösenden Tastendrucks an die Clients weitergeleitet, die dann denselben Filewechsel nachvollziehen können wie der Server.

Vorteil dieser Implementierung gegenüber der Lösung durch Java ist das Zurückgreifen auf schon bestehende Funktionalität, nämlich das Verschicken der Tastenevents.

Nachteil ist das relativ umständliche Implementieren der einzelnen Aktuatoren. Außerdem muss man genau im Vorfeld wissen, welche Blenderfiles man verwenden möchte und kann nicht frei wählen. Daher wurde auch die Javavariante implementiert, die es ermöglicht die zu ladenden Files per Namen anzugeben und zu starten.

7. Test und Einsatzberichte

7.1 Objektkontrolle

Dieses Prinzip kam bereits auf dem Internationalen Architektur Symposium Pontresina vom 12. bis zum 14. September 2002 zum Einsatz. Hierbei wurde eine dreiwandige, interaktive Medieninstallation geschaffen, die es dem User erlaubte mit eigenem Verhalten ausgestattete, dreidimensionale Bauelemente im virtuellen Raum zu platzieren und somit eigene Raumstrukturen zu erschaffen. Diese Installation war Teil 2 der Ausstellungsreihe „Instant Architecture“.

Hierbei wurde die Objektkontrolle in zwei verschiedenen 3D-Welten eingesetzt. In der ersten Welt wurde eine volle Kontrolle über alle erzeugten Objekte gewährleistet. Die Objektanzahl wurde jedoch auf 40 Stück begrenzt, da ab dieser Zahl ein deutlich sehbarer Performanceverlust mit Frameraten unter 25 Bildern pro Sekunde eintritt.

Bei der zweiten Welt konnten ca. 200 Objekte generiert werden, bevor die Framerate rapide sank. Dies wurde durch eine Art „Kurzzeitgedächtnis“ erreicht.

Es wurden immer 10 Objekte erzeugt, auf die noch Einfluss genommen werden konnte. Waren diese 10 Objekte generiert, wurden sie nicht mehr jeden Frame zentral durch den Server positioniert und ausgerichtet. Ein neues Feld von 10 kontrollierten Objekten wurde erzeugt und manipulierbar gehalten.

Die beiden Welten liefen trotz der relativ hohen Anforderungen stabil.

7.2 Abstandssensor

Der Abstandssensor kam als Interaktionsgerät auf der „trans-sphaere“-Ausstellung im Rahmen des Architektur-Weltkongresses UIA vom 18. bis 28.7.2002 zum Einsatz. Hierbei wurden je nach Entfernung des Users unterschiedlich 3D-Objekte auf einem 40 Zoll-Plasmabildschirm erzeugt.

Der Abstandssensor stellt nach Kalibrierung ca. 15 mal pro Sekunde Messwerte zur Verfügung. Dies ist leider zu wenig, um eine ruckelfreie Interaktion zu gewährleisten und daher noch optimierungsbedürftig. Zum Erzeugen von Tastenevents ist dies jedoch ausreichend.

Das System lief 10 Tage ohne Ausfälle.

7.3 Autostart

Das Prinzip des automatischen Starten ist bei einer einwandigen Installation seit 10. Juli 2002 in der Deutschen Arbeitsschutzausstellung in Dortmund ohne Zwischenfälle im Einsatz.

Auf dieser Ausstellung wird jeden Abend aus feuerrechtlichen Gründen der Strom ab- und morgens wieder angeschaltet, was das Starten des Systems nach sich zieht.

7.4 Webcam

Die Webcam kam, wie bereits erwähnt, auf dem fünftägigen Symposium „Urban Drift“ vom 9. - 13. Oktober 2002 zum Einsatz. Als Interaktionsgerät ist diese Art sehr direkt und gut nachvollziehbar. Technisch lief die Anwendung stabil.

Es wurde eine Frameanzahl von ca. 15 Frames pro Sekunde erreicht. Dies wirkte etwas ruckelnd, was aber nicht als sehr störend empfunden wurde, da keine flüssigen Bewegungen durch die 3D-Welt eingeplant waren.

Ein störender Aspekt war, dass beim Anzeigen der aufgenommenen Bilder manchmal Zugriffsfehler auftraten. Dies bedeutet das Bild konnte nicht geladen werden, wenn gleichzeitig geschrieben wurde, wodurch der Bildschirm kurzzeitig schwarz erschien.

7.5 Trittmatten

Diese Art der Interaktion ist bei der bereits beschriebenen Medieninstallation in Dortmund und in einer 3-Wand-Installation auf der „trans-sphaere“-Ausstellung zum Einsatz gekommen.

Hierbei wurde jeweils die Navigation durch die 3D-Welten durch 4 Matten und die Erzeugung von Objekten durch 2-4 weitere Matten erreicht.

Die Trittmatten haben sich als äußerst robust und zuverlässig erwiesen. Selten trat ein Hängenbleiben der Trittmatten aus. Dies bewirkte ein



Abb. 27 : Aufbau des Exponats in Dortmund



Abb. 28 : Trittmattensteuerung in einer 3-Wand-Umgebung zur UIA

weiteres Drücken der Taste, wodurch die resultierende Aktion weiter ausgeführt wurde, obwohl die Trittmatte nicht mehr belastet war.

Um diesen Effekt wieder aufzulösen, musste man nochmals auf die Trittmatte treten.

Dieses Problem trat jedoch bisher nur auf der „trans-sphaere“-Ausstellung auf und nicht auf der permanenten Ausstellung in Dortmund.

7.6 Watchdog

Auch die Watchdog-Karte ist in der Installation auf der Deutschen Arbeitsschutzausstellung in Dortmund als Sicherheit in das Mediensystem, welches seit vier Monaten problemlos läuft, eingebaut.

Es kam bisher einmal zum Einfrieren des Systems, worauf die Watchdogkarte einen Neustart veranlasste.

8. Resümee und Ausblick

Cube4 hat sich bisher als äußerst stabiles und flexibles System bewiesen. Die Wahl von Blender sowohl als Modellierungstool als auch als Gameengine hat sich aufgrund der offenen Architektur und der damit verbundenen Flexibilität als sehr vielversprechend erwiesen und bietet auch für die Zukunft ein großes Entwicklungspotential. Dieser Aspekt wird noch durch die Freigabe der Quelltexte von Blender im Oktober 2002 verstärkt. Die Kombination von Blender und Python und der Aufbau der Gameengine unterstützen ein äußerst rasches Implementieren von Konzepten für VR-Welten mit einem großen Spektrum an Möglichkeiten.

Die in dieser Arbeit eingesetzten und entwickelten Eingabegeräte haben sich bisher in Praxistests und auf den bereits gelaufenen Ausstellungen bewährt und sind gut angenommen worden.

Die Ausgabe der 3D-Welten ist bisher lediglich im frühen Prototypenstadium, wird aber in Zukunft weiterentwickelt.

Das gesamte System wird auch in Zukunft erweitert und verbessert. Ein großer neuer Bereich wird die Einbindung von externen Datenquellen sein. Hierzu gehören vor allem Datenbanken, aber auch andere Quellen, wie z.B. Wetterstationen, Telefonanlagen, u.a. sind als Lieferanten zur Datenvisualisierung in Echtzeit denkbar.

Auch die Überwachung der Stabilität des Systems wird weiterhin verbessert. Hierfür wird vor allem die Javaüberwachung durch SNMP detaillierter gestaltet werden.

Anhang A : Literaturverzeichnis

- [Aukstakalnis 94] Cyberspace : Die Entdeckung künstlicher Welten
S. Aukstakalnis, D. Blatner
VGS Verlagsgesellschaft, Köln, 1994
- [Cruz-Neira 93] Surround-Screen Projection-Based Virtual Reality : The Design and
Implementation of the CAVE
Carolina Cruz-Neira, Daniel J. Sandin, et al.
Siggraph, Anaheim, 1.-6.8.1993, S 135-142
- [Däßler 98] Virtuelle Informationsräume mit VRML
Rolf Däßler, Hartmut Palm
dpunkt-Verlag, Heidelberg, 1998, S. 101
- [Gerlach 01] Serielles I²C-Interface
Ingo Gerlach
Elektor, Ausgabe 06/2001, Seite 76 ff.
- [Götz 01] Messen, Steuern, Regeln mit Java
S. Götz, R. Mende
Franzis' Verlag GmbH, 2001
- [Raepple 01] Sicherheitskonzepte für das Internet
Martin Raepple
dpunkt-Verlag, Heidelberg, 2001, S271 ff.
- [Schütte 99] Java
Lothar Schütte
RRZN, Uni Hannover, 1999, S. 9
- [von Löwis 01] Python 2
Martin von Löwis, Nils Fischbeck
Addison-Wesley-Verlag, München, S. 1

Anhang B : Informationsquellen im Internet

- [CUBIC] The Cubic Mouse
http://www.fakespacesystems.com/pdfs/FS_ss_Cmouse.pdf
zuletzt besucht am 10.11.2002
- [Deboeser 02] Konzeption und Umsetzung von Disaster Recovery-Strategien
Achim Deboeser
http://www.itseccity.de/?url=/content/fachbeitraege/grundlagen/020315_fac_gru_veritas.html
zuletzt besucht am 10.11.2002
- [Isdale 98] What is VR
Jerry Isdale
<http://vr.isdale.com/WhatIsVR.html>
zuletzt besucht am 10.11.2002
- [Jacobson] CaveUT
Jeffrey Jacobson
<http://planetjeff.net/ut/CaveUT.html>
zuletzt besucht am 10.11.2002
- [Pape 97] Cave User's Guide
Dave Pape, Carolina Cruz-Neira, et al.
<http://www.evl.uic.edu/pape/CAVE/prog/CAVEGuide.html>
zuletzt besucht am 10.11.2002
- [PIL] Python Imaging Library
<http://www.pythonware.com/library/pil/handbook/preface.htm>
zuletzt besucht am 10.11.2002
- [US-Sensor] SRF08 Ultraschall-Abstandsmesser
<http://www.robot-electronics.co.uk/html/srf08tech.shtml>
zuletzt besucht am 10.11.2002
- [X-Rooms] X-Rooms
http://www.first.gmd.de/vista/X-Rooms-Dateien/d_jb00_erg_pt2.html
zuletzt besucht am 10.11.2002
- [Zachow] VR Eingabegeräte
Stefan Zachow, Johann-Habakuk Israel, et al.
http://cg.cs.tu-berlin.de/~kai/tablet/vr_in.html
zuletzt besucht am 10.11.2002

Anhang C : Abbildungsverzeichnis

Abbildung 1 :	Cubic Mouse detailliert http://computer.org/cga/cg2000/pdf/g4012.pdf
Abbildung 2 :	Cubic Mouse http://computer.org/cga/cg2000/pdf/g4012.pdf
Abbildung 3 :	Datenhandschuh http://cg.cs.tu-berlin.de/%7Ekai/tablet/vr_in.html
Abbildung 4 :	5-Wand-X-Room von oben http://www.x-rooms.de/images/pic_fuenfwand_big.jpg
Abbildung 5 :	X-Room in Seitenansicht http://www.x-rooms.de/images/pic_fuenfwand2_big.jpg
Abbildung 6 :	CaveUT in Eckprojektion http://usl.sis.pitt.edu/PlanetJeff/ut/CaveUT/Images/CornerCaveBIG.jpg
Abbildung 7 :	Screenshots des Programmes „Getif“
Abbildung 8 :	Screenshots von Datenflügen
Abbildung 9 :	Überblick über Cube4
Abbildung 10 :	Skizzierung der Auswirkung der interaktiven Änderung der Bewegung im Raum
Abbildung 11 :	Screenshot von Cube4
Abbildung 12 :	3D-Interpretationen der Webcambilder
Abbildung 13 :	Trittmattensteuerung
Abbildung 14 :	Örtliche Bestimmung durch Trittmatten
Abbildung 15 :	Sicherheitskonzept
Abbildung 16 :	4-Wand-Projektion
Abbildung 17 :	360-Grad-Projektion
Abbildung 18 :	Beispiel für einen Logic Brick
Abbildung 19 :	Auszug aus einer Tastaturmatrix
Abbildung 20 :	Aufbau des Prototyps der Trittmattensteuerung
Abbildung 21 :	Ultraschallsensor mit Interfacekarte

- Abbildung 22 : Webcam als Interaktionsgerät
- Abbildung 23 : jeweils 3 Screenshots der Renderclients von Cube4 perspektivisch dargestellt
- Abbildung 24 : Stringproperty zum Verschicken von Tastenevents
- Abbildung 25 : Fenster zum Starten und Kalibrieren des Ultraschallsensors
- Abbildung 26 : Autostarteinstellungen in Blender
- Abbildung 27 : Aufbau des Exponats in Dortmund
- Abbildung 28 : Trittmattensteuerung in einer 3-Wand-Umgebung zur UIA

Anhang D : Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner Prüfungsbehörde vorgelegt.

Alex Habermann
Berlin, den 12.11.2002