V.A.I.

Ein System zur intelligenten Navigation durch VRML Welten mittels graphischer Abstraktion

Diplomarbeit
Lehrstuhl Prof. Wahlster
Universität des Saarlanders

vorgelegt von

Christoph Stahl

Saarbrücken
April 2001

Hiermit versichere ich an Eides statt, daß ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Christoph Stahl

Danksagung

An dieser Stelle möchte ich allen Personen danken, die mir die angenehme Erstellung dieser Diplomarbeit ermöglicht haben. Dazu gehört vor allem Prof. Dr. Wolfgang Wahlster, an dessen Lehrstuhl optimale Bedingungen für ein kreatives und freies Arbeiten zu finden sind. Besonderer Dank gilt meinem Betreuer Antonio Krüger, der mir das interessante Thema angeboten und stets mit inspirierenden Gesprächen geholfen hat, tiefer in das Fachgebiet einzutauchen und dabei auftauchende Fragen zu klären. Die gemeinsame Vorstellung der Zwischenergebnisse unserer Arbeiten auf Konferenzen war ein schönes Erlebnis. Danken möchte ich auch den übrigen Mitarbeitern, Diplomanden und Hiwis für eine wohlwollende Atmosphäre am Lehrstuhl.

Auf persönlicher Seite möchte ich meinen Eltern für die finanzielle Grundlage meines Studiums danken, insbesondere meiner Mutter Brita.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren hat sich das Informationsangebot im Internet enorm gesteigert. Dabei blieb die Informationsdarstellung stets zweidimensional in Analogie zum gewohnten Papier. In Videoclips und Computerspielen wurde der Schritt in die dritte Dimension jedoch schon lange vollzogen. Es lag also nahe, Informationen im Internet auch dreidimensional zu präsentieren. Mit VRML[1] wurde 1997 ein internationaler Standard geschaffen, um 3D-Welten in HTML[2]-Dokumente einzubinden. VRML bietet dabei Möglichkeiten zur Animation, Interaktion und Hyperlinks. Obwohl VRML kaum für kommerzielle Anwendungen genutzt wurde, sind zahlreiche Welten von Privatleuten, Künstlern und Organisationen kreiert worden. Eine umfangreiche Sammlung solcher Welten ist im VRML-Repository [Con01] zu finden. Leider birgt die dritte Dimension auch einige Tücken für den Betrachter.

- **VRML Welten bieten oft vielfältige, interaktive Funktionalität an, die auf Anhieb jedoch nicht erkennbar ist.**

  Dies liegt daran, das es im Gegensatz zu HTML-Dokumenten noch keine allgemein gültigen Gestaltungsregeln für funktionale Elemente gibt. In Hypertextdokumenten führen bestimmte Begriffe den Leser zu relevanten Textpassagen; diese Begriffe sind dann durch eindeutige Stilelemente hervorgehoben (z.B. im Internet durch Unterstreichen). Analog dazu können in VRML beliebige Gegenstände verwendet werden, um den Betrachter an einen anderen Ort zu versetzen. Zusätzlich können Gegenstände die Möglichkeit zur Interaktion bieten, indem sie sich mit der Maus manipulieren lassen oder Veränderungen in der Welt auslösen. Gegenstände lassen sich im Gegensatz zu Schriften jedoch nicht unter Verwendung von klar definierten Stilen eindeutig hervorheben. In der Praxis ist es also recht schwierig, sich in einer 3D Umgebung zurechtzufinden.

  Abbildung 1.1 zeigt dazu als Beispiel eine interaktive VRML-Welt [Pla00], in der die Stadt Tokyo mit den wichtigsten Gebäuden modelliert wurde. Die Stadt läßt sich aus verschiedene Perspektiven betrachten (siehe auch Abbildung 1.2). Durch Anklicken von Gebäuden im Bild lassen sich Informationen abrufen, die dann unten rechts im Bildschirm angezeigt werden. Jedoch sind nicht zu allen Gebäuden Informationen verfügbar, und es ist nicht ersichtlich, welche das sind. Hier hilft nur ausprobieren.

  Das gleiche Problem stellt sich in ähnlicher Form bei dem Computertisch [Gra00], der in Abbildung 1.3 dargestellt ist. Dort sind verschiedene Objekte wie Monitor, Bücher, Lampe und Telefon dargestellt. Viele davon bieten die Möglichkeit zur Interaktion an. Die Lampe kann beispielsweise an- und ausgeschaltet werden. Das Telefon kann abgehoben, Schubladen und Bücher können herausgezogen werden. Die-

[1] Virtual Reality Modeling Language
[2] Hyper-Text Markup Language

se Funktionalität ist jedoch auf den ersten Blick nicht erkennbar, sondern kann nur experimentell herausgefunden werden.

Dabei unterstützt der verwendete VRML-Browser die Navigation, indem der Mauszeiger bei Berührung eines interaktiven Objektes dessen Funktionalität anzeigt. Dazu verwandelt er sich von einem Pfeil in ein anderes Symbol. Jedoch muß der gesamte Bildinhalt mit der Maus untersucht werden, was ein zeitaufwendiger Vorgang ist. Kleinere Objekte werden dabei leicht übersehen.

- **Die begrenzten technischen Ressourcen zur Bildgenerierung** stehen dem Wunsch des Betrachters nach einer möglichst real wirkenden Welt entgegen. Die dazu notwendigen Details gehen auf Kosten der Rechenkapazität und verlangsamen den Prozeß der Bildgenerierung. Die für das Auge wichtige Rate von mindestens 25 aufeinanderfolgenden Bildern pro Sekunde ist dann nicht zu erreichen, der Eindruck von Bewegung wie in einem Film geht veroren. Für den Betrachter äußert sich dies nachteilig in einer trägen, ruckartigen Navigation, die mit einer Maus ohnehin nur umständlich zu bewerkstelligen ist.

- Doch nicht nur die Rechnerkapazität, sondern **auch die Kognitiven Ressourcen des Betrachters sind limitiert**. Je mehr Gegenstände abgebildet sind, desto länger benötigt der Betrachter um diese zu erfassen. Es besteht die Gefahr, den Überblick zu verlieren und wichtige Objekte zu übersehen.

  Zusammenfassend erscheint eine **Navigationshilfe** wünschenswert, die den Betrachter virtueller Welten von den genannten Problemen befreit. Diese können durch die Anwendung graphischer Abstraktion gelöst werden.

Abbildung 1.1: Ein interaktives Modell der Stadt Tokyo, gesehen aus der Vogelperspektive.

Abbildung 1.2: Dasselbe Modell, gesehen aus der Egoperspektive.

Abbildung 1.3: Ein interaktives Modell von einem Computertisch.

1.1 Ziel und Gegenstand der Arbeit

Das Ziel der Arbeit besteht darin, eine Navigationshilfe für VRML 3D-Welten zu entwerfen. Dieses System soll es dem Betrachter einer Welt ermöglichen, den Abstraktionsgrad von einzelnen Objekten der Szene interaktiv zu manipulieren und seinem individuellen Navigationsziel anzupassen. So wird er durch Reduzieren von Details auf drei Wegen unterstützt: Die zur Grafikdarstellung benötigten Ressourcen werden entlastet, so daß die Bildwiederholrate zunimmt und die Bewegungssteuerung erleichtert wird. Gleichzeitig werden die kognitiven Ressourcen des Betrachters geschont und können gezielt auf funktionale Elemente der Szene fokussiert werden.

Um ein solches System tatsächlich implementieren zu können, sollen im Rahmen dieser Arbeit die folgenden Fragen beantwortet werden:

- **Wie kann der Betrachter durch Abstraktion bei der Navigation unterstützt werden ?** In der Einleitung wurden unsichtbare Funktionalität und Ressourcenbeschränkung des Betrachters und Computers als Probleme indentifiziert. Es ist zu zeigen, wie Abstraktion diese Probleme lösen kann.
- **Welche Techniken und Systeme zur graphischen Abstraktion sind bereits vorhanden und lassen sich auf VRML-Welten anwenden ?** Dazu werden Verfahren aus der Computergraphik zur Datenreduktion betrachtet.
- **Welche Informationen über mögliche Ziele des Betrachters lassen sich aus einer VRML-Welt ableiten ?** Da über die Ziele des Betrachters kein Wissen vorhanden ist, soll versucht werden, mögliche Ziele aus den interaktiven Möglichkeiten einer Szene abzuleiten.
- **Wie kann eine Benutzerschnittstelle zur Auswahl von Abstraktionsgraden gestaltet werden ?** Da sich bisher keine Standards für Benutzerschnittstellen in 3D Umgebungen etabliert haben, soll versucht werden, Konzepte zweidimensionaler Oberflächen zu übertragen.
- **Mit welchen Technologien kann ein Hilfssystem mit vertretbarem Aufwand implementiert werden ?** Welche Komponenten sind frei verfügbar und für VRML geeignet, und welche Anpassungen sind erforderlich ?

1.2 Aufbau der Arbeit

In Kapitel 2, das dieser Einleitung folgt, wird in Abschnitt 2.1 über graphische Abstraktion und in Abschnitt 2.2 über Navigation in virtuellen Welten diskutiert. Anschließend werden Grundlagen der Computergraphik (Abschnitt 2.3), insbesondere die Repräsentation von Modellen und Techniken zur graphischen Abstraktion vorgestellt (Abschnitt 2.4). In Abschnitt 2.5 werden Systeme zur automatischen Abstraktion ganzer Szenen beschrieben. Den Abschluß bildet Abschnitt 2.6 mit einer Einführung in die Szenenbeschreibungssprache VRML.

In Kapitel 3 werden in Abschnitt 3.1 die vorgestellten Techniken zur graphischen Abstraktion auf ihre Eignung in VRML hin überprüft. Abschnitt 3.2 beschreibt die Extraktion von Informationen aus der VRML-Welt, die für ein Abstraktionssystem relevant sind. In Abschnitt 3.3 werden Algorithmen zur Primitverkennung entwickelt, um weitere Informationen über die Struktur der Welt zu erhalten. Abschnitt 3.4 definiert Konzepte zur Interaktion des Benutzers mit dem Hilfssystem.

Kapitel 4 beschreibt in Abschnitt 4.1 den Entwurf der Navigationshilfe. Dabei werden in Abschnitt 4.2 zwei Betriebsmodi unterschieden und Varianten bezüglich Ablauf und Architektur spezifiziert. In Abschnitt 4.3 wird dazu ein User Interface entworfen.

Kapitel 5 behandelt in Abschnitt 5.1 die Implementation der im vorangegangenen Kapitel spezifizierten Varianten des Systems. Abschnitt 5.2 erläutert die technische Umsetzung der in Abschnitt 3.3 entworfenen Algorithmen zur Primitiverkennung. Die Funktionalität wird in Kapitel 6 anhand von Anwendungsbeispielen bewiesen (Abschnitt 6.1). Die weiteren Abschnitte 6.2 und 6.3 zeigen anhand verschiedener VRML- Welten, wie das entwickelte Hilfssystem den Betrachter bei seiner Navigationsaufgabe unterstützen kann.

Kapitel 7 bildet den Abschluß mit einer Zusammenfassung der Arbeit.

Kapitel 2

Grundlagen und Stand der Forschung

In der Einleitung sind die Schwierigkeiten, die sich für den Betrachter einer 3D-Welt bei der Navigation ergeben, veranschaulicht worden. Um diese Probleme zu lösen, wird in dieser Arbeit ein System entworfen, das den Benutzer bei seiner Navigationsaufgabe unterstützt. Dieses System soll die kognitiven Anforderungen an den Betrachter verringern, indem es dessen Aufmerksamkeit gezielt auf relevante Objekte lenkt.

Dieses Kapitel gibt einen Überblick über verschiedene Techniken und Grundlagen der Computergraphik, die zur Realisierung der Navigationshilfe eingesetzt werden. In Abschnitt 2.1 werden Techniken der graphischen Abstraktion zusammengefasst, die häufig bei Illustrationen und Bedienungsanleitungen Anwendung finden. In Abschnitt 2.2 werden Vorschläge und Anforderungen an eine Navigationshilfe aus der Fachliteratur zitiert. Abschnitt 2.3 gibt eine kurze Zusammenfassung der Grundlagen der Computergraphik, soweit sie zum Verständnis der Arbeit notwendig sind. In Abschnitt 2.4 liegt der Schwerpunkt auf den Vorarbeiten im Bereich der graphischen Abstraktion, danach werden in Abschnitt 2.5 verschiedene Abstraktionssysteme vorgestellt. Eine kurze Einführung in VRML folgt in Abschnitt 2.6. Abschnitt 2.7 ergänzt die vorgestellten Techniken zur Bildgenerierung um ein Verfahren zur Bilderkennung, das später im Abschnitt 3.2 zur Informationsextraktion aus einer Szene motiviert wird. Abschließend wird in Abschnitt 2.8 die Bedeutung der vorgestellten Grundlagen und Literatur für diese Arbeit diskutiert.

2.1 Fokussieren durch graphische Abstraktion

Es stellt sich nun die Frage, wie die Aufmerksamkeit des Betrachters einer Illustration fokussiert werden kann. In Bedienungsanleitungen und technischen Dokumentationen kommen verschiedene zeichnerische Methoden der graphischen Abstraktion zum Einsatz, um in Abbildungen wichtige Elemente hervorzuheben. Im einfachsten Fall werden unwesentliche Objekte einfach weggelassen (Beispiel: Einleitung Abbildung 2.6(b)). Wird jedoch zuviel entfernt, fehlt der Zusammenhang, und die dargestellten Details lassen sich nicht mehr lokalisieren. Um solche Fehler zu vermeiden, werden Objekte zwar gezeigt, jedoch nur als vereinfachte Kontur dargestellt (Beispiel: Abbildung 2.1). Dabei kann man noch einen Schritt weiter gehen und benachbarte ähnliche Objekte miteinander *verschmelzen* (Beispiel: Abbildung 2.2), um die Anzahl der unterscheidbaren Elemente zu verringern.

Durch eine Vereinheitlichung von Objektattributen wie der Farbe und Darstellungsart kann eine Abbildung in Bereiche aufgeteilt werden, denen unterschiedliche Aufmerksamkeit zuteil wird (Beispiel: Abbildung 2.4(b)). Hierbei sind die möglichen Stile des Hervorhebens durch die technischen Fähigkeiten des Ausgabemediums beschränkt. Insbesondere bei einem Wechsel des Mediums ergeben sich daher Probleme. Beispielsweise gehen

Einfärbungen bei einem Wechsel vom Farbdisplay auf Schwarzweißdruck verloren. Niedrigauflösende Displays von mobilen Geräten können keine Schraffuren und Strichstärken darstellen. Falls eine Manipulation der Attribute nicht in Frage kommt, kann eine Fokussierung durch *Vergrösserung* der relevanten Objekte erreicht werden.

Alle diese Methoden beruhen auf einer Veränderung des ursprünglichen Bildinhaltes und werden in der Regel durch den Einsatz von *Metagraphiken* verstärkt. Darunter versteht man das Einfügen von graphischen Elementen, die relevante Inhalte markieren. Beispiele sind Pfeile, Kreuze oder Kreise und auch Annotationen. Diese Methode ist in der Praxis häufig anzutreffen, da sie mit geringem technischen Aufwand verbunden ist. Metagraphische Elemente sind jedoch stets Fremdkörper, die ungewollt Aufmerksamkeit auf sich selbst ziehen.

Abbildung 2.3 zeigt ein Beispiel einer von Hand angefertigten Illustration, in der verschiedenexr Methoden der graphischen Abstraktion kombiniert angewendet worden sind. Die Abbildung verdeutlicht, wie ein Speichermodul in den dafür vorgesehenen Sockel der Graphikkarte eingesetzt wird. Es handelt sich nicht um eine photographische Abbildung, sondern um eine Skizze. Das Speichermodul ist als vereinfachte Kontur dargestellt, die RAM-Bausteine darauf sind nur durch Rechtecke angedeutet. Die Graphikkarte selbst wird nicht dargestellt, nur der Sockel ist als Umriss gezeichnet. Als Metagraphik sind Pfeile eingefügt worden, um die Richtung der Bewegung beim Einsetzen anzuzeigen. Durch die Vergrösserung wird besonders auf die Aussparung des Moduls hingewiesen. Ein Pfeil hilft bei der Lokalisiserung des Ausschnitts.

Die eleganteste Lösung der Illustration besteht in einer indirekten Fokussierung, die ohne Metagraphiken und Veränderung der Farbe auskommt. Stattdessen werden alle unwichtigen Objekte entfernt oder verschmolzen und in Form und Kontur vereinfacht dargestellt, so daß sie in den Hintergrund treten.

Die in diesem Abschnitt angesprochenen Methoden sowie weitere Bildbeispiele dazu werden detailliert in [Krü00] vorgestellt. In Abschnitt 2.4 werden verschiedene Techniken aus der Computergraphik zur graphischen Abstraktion beschrieben.

2.1.1 Designrichtlinien für Illustrationen

Der Psychologe Kosslyn hat in seinem Buch [Kos94] das bisherige Wissen über die Beschränkungen der menschlichen Wahrnehmung in Bezug auf das Verständnis von Abbildugen zusammengefasst. Kosslyn überträgt nun diese Erkenntnisse auf die Gestaltung von Illustrationen, so daß ihre vom Autor beabsichtige Aussage besonders schnell und leicht zu verstehen ist. Er formuliert anhand von Bildbeispielen zuerst Regeln für gutes Design von Graphen, die typischerweise Zahlen visualisieren, später aber auch von allgemeineren Illustrationen. Diese Beispiele lassen sich durchaus auch auf die Gestaltung virtueller Welten übertragen und sollen daher hier kurz vorgestellt werden.

Das erste Bildbeispiel ist eine Darstellung, die dem Betrachter die Position der Bedienungselemente eines Autos vermitteln soll. Abbildung 2.4(a) ist dazu wenig geeignet, da dort alle Elemente gezeigt werden. In Abbildung 2.4(b) sind daher irrelevante Komponenten entfernt und relevante hervorgehoben worden. Das zweite Beispiel ist ein Stadtplan. In Abbildung 2.5(a) sind Haupt- nicht von Nebenstrassen zu unterscheiden. Diese sind in Abbildung 2.5(b) hervorgehoben, so daß die Routenplanung vereinfacht wird. Im dritten Beispiel wird eine Karte gezeigt, die als Weginformation zur Feuerwache gedacht ist. Abbildung 2.6(a) zeigt mehr Details als für diesen Zweck notwendig sind. Die in Abbildung 2.6(b) dargestellte Karte ist leichter zu lesen, da hier nur die wichtigen Hauptstrassen sichtbar sind.

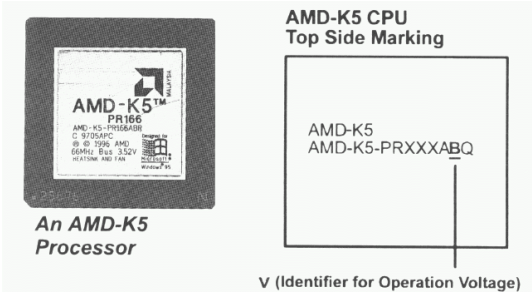


Abbildung 2.1: Quelle: AP5c/P User's Guide [Aut95]

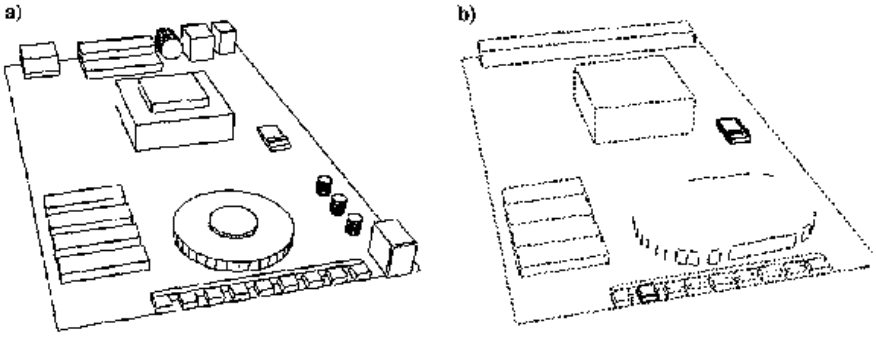


Abbildung 2.2: Verschmelzen von Bauteilen einer Modemplatine mit *PROXIMA* [Krü95]

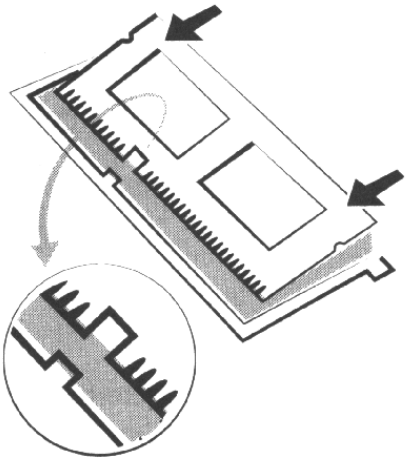
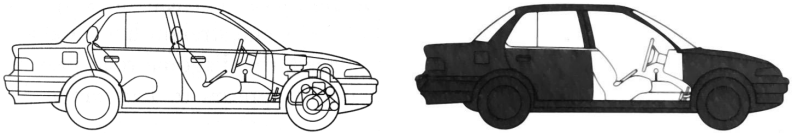
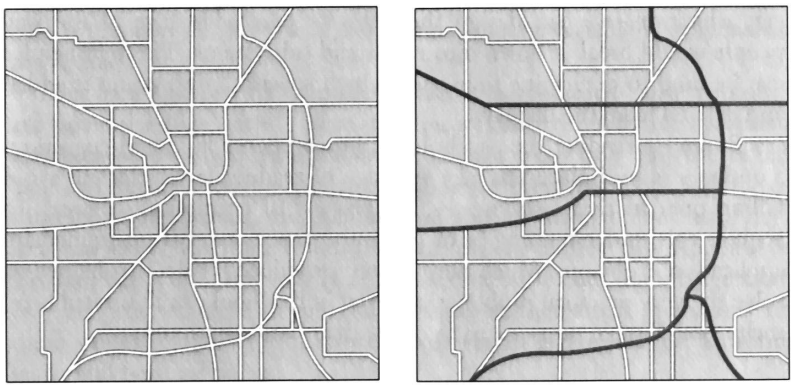


Abbildung 2.3: Quelle: Elsa Gloria Synergie Handbuch [Aut97]



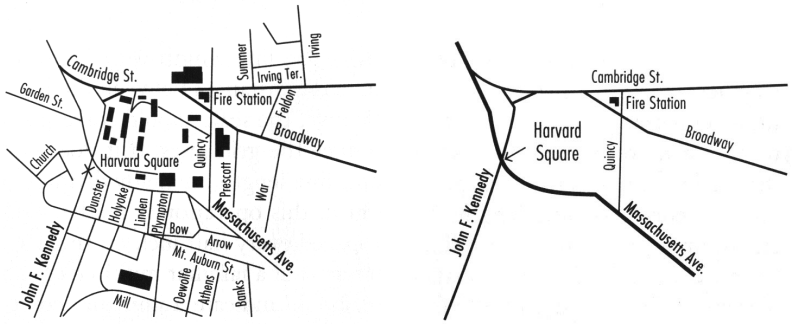
(a) Falsch (b) Richtig

Abbildung 2.4: Ein Diagramm, das die Bedienungselemente des Fahrers zeigen soll, ist einfacher zu verstehen, wenn die irrelevanten Komponenten entfernt und die relevanten hervorgehoben werden.



(a) Falsch (b) Richtig

Abbildung 2.5: Wenn die hervorstechenden Linien den Hauptstrassen entsprechen, ist die Karte leichter zu lesen und schnelle Routen sind einfacher zu finden.



(a) Falsch (b) Richtig

Abbildung 2.6: In einer Karte sind nur solche Details nützlich, die dem eigentlichen Zweck der Karte entsprechen. Zur Beschreibung der Feuerwache sind nur Hauptstrassen nötig.

2.2 Navigationshilfen in 3D Welten

In diesem Abschnitt werden zwei Arbeiten vorgestellt, die grundlegende und wichtige Aussagen zum Thema Navigation gemacht haben. Im Anschluß wird das Open Inventor System vorgestellt, das die Anregungen umsetzt und dem Betrachter von Modellen die Navigation erleichtert. In Abschnitt 2.8 wird diskutiert, wie die formulierten Anforderungen in dieser Arbeit umgesetzt werden können.

2.2.1 Interaktive Illustrationen für elektronische Bücher

Andries van Dam beschreibt in seinem Artikel [vD91] anlässlich der Interactive Learning Konferenz 1992 seine Vorstellung von elektronischen Büchern, die insbesondere in der Lehre Einzug halten sollen.

- **Die einfache Handhabung gedruckter Bücher** soll dabei als Vorbild dienen. Die Vorteile liegen einerseits in der leichten Annotierbarkeit von Textpassagen, andererseits in der einfachen und vertrauten Navigation mittels Vor- und Zurückblättern der Seiten. Die Suchmöglichkeiten sind jedoch auf das Inhaltsverzeichnis und den Index begrenzt. Das Ziel bei der Entwicklung elektronischer Bücher muß es deshalb sein, die Vorteile von gedruckten Büchern mit der Dynamik digitaler Medien zu verbinden, die es erlaubt, Inhalte kontinuierlich zu aktualisieren und zu erweitern. Van Dam gliedert die dazu notwendigen Techniken in die Bereiche Hypermedia, KI basierte Wissensbasis und interaktive Animation auf.

- Die **Interaktive Illustration** bildet den Schwerpunkt von van Dams Tätigkeit. Er geht davon aus, dass in Lehrbüchern vom Autor gestaltete und vorgefertigte Videoclips zum Verständnis nicht ausreichen.

- **Der Betrachter einer Szene muß den Detaillierungsgrad im Modell frei wählen können**, um die Animation inhaltlich genau seinen Bedürfnissen anzupassen und sich gezielt auf die ihn interessierenden Objekte zu konzentrieren. Dazu gehört auch, die simulierten Abläufe über Parameter zu steuern und die virtuelle Kamera frei zu positionieren. Die größte Schwierigkeit liegt in den fehlenden Tools für Autoren um Animationen zu gestalten. Es gibt noch immer keine einfache Standardsoftware, die Editoren sind sehr komplex und erfordern viel Einarbeitungszeit. Eine physikalisch korrekte Simulation der Bewegung von Objekten könnte in einigen Fällen die Animation vereinfachen; dazu sind Techniken wie inverse Kinematik und Constraints erforderlich. Um die Modellierung zu beschleunigen, sind Bibliotheken von 3D Modellen unverzichtbar. Dabei schlägt van Dam für jedes Modell verschiedene Detailstufen je nach Zielgruppe vor und erläutert dies beispielhaft an einem Molekülmodell. Ein sehr einfaches Modell richtet sich an Schüler, für Studienanfänger gibt es eine komplexere und detailliertere Variante. Wissenschaftler bekommen ein Modell, das die Realität so gut wie möglich abbildet. Einem Schüler wird in seinem elektronischen Buch nun das einfache Modell angeboten, entsprechend seinem Alter und Wissen. Interessiert er sich für mehr Details, kann er das Modell für Studenten abrufen und erklären lassen. Der Schüler wird nicht auf einen seiner Schulklasse entsprechenden Inhalt beschränkt.

- **Ein geeignetes Eingabegerät** ist Voraussetzung für ein erfolgreiches Arbeiten mit interaktiven Illustrationen. Die Maus ist nur zweidimensional, so daß immer zwischen verschiedenen Freiheitsgraden umgeschaltet werden muß. Eine Tastatur kann keine natürliche Bewegung erfassen. Als Alternative schlägt van Dam einen Joystick mit Force-Feedback und 6 Freiheitsgraden vor.

Das System *Textillustrator* [Sch99] von Schlechtweg wird in dieser Arbeit in Abschnitt 2.5.3 vorgestellt. Es stellt einen Lösungsansatz für interaktive Illustrationen in elektronischen Büchern dar.

2.2.2 Ergonomischer Entwurf von Benutzerschnittstellen

In seinem Buch [Shn92] *Designing the Userinterface* definiert Shneiderman Eigenschaften, die ein Userinterface unbedingt erfüllten muß, um ergonomisch und benutzerfreundlich zu sein. Dabei geht er speziell auf Virtual-Reality Umgebungen ein.

- **Das Prinzip der direkten Manipulation** wird von Shneiderman als Schnittstellendesign empfohlen. Um Aktionen auszulösen, werden dabei Objekte direkt durch Zeigegesten, z.B. mit der Maus, selektiert, ohne vorher Funktionen in einem Menü auszuwählen. Die Auswirkungen der Aktion sind sofort und ohne Wartezeit sichtbar.
- **Ein Menü ermöglicht die Auswahl aus verschiedenen Aktionen.** Es sollte als eigenständiges Objekt in die VR-Szene integriert sein um ein ständiges Umschalten zwischen der 3D- Darstellung und einer 2D-Benutzeroberfläche zu vermeiden.
- **Die Bildschirmrate** des User Interface ist neben dem Design ebenfalls von hoher Bedeuting für die Navigation. Damit ist die Anzahl der vom Computer berechneten Bilder pro Sekunde gemeint. Um dem menschlichen Auge den Eindruck einer flüssigen Animation zu vermitteln, sind mindestens 25 Bilder pro Sekunde erforderlich. In komplexen Szenen kann es jedoch vorkommen, das nur ein einzelnes Bild erzeugt werden kann. Dann entsteht eine starke Verzögerung zwischen Mausbewegung und Reaktion der Kamera, so daß man kein Gefühl für die Steuerung entwickeln kann und sein Ziel entweder nicht erreicht oder darüber hinaus schießt.

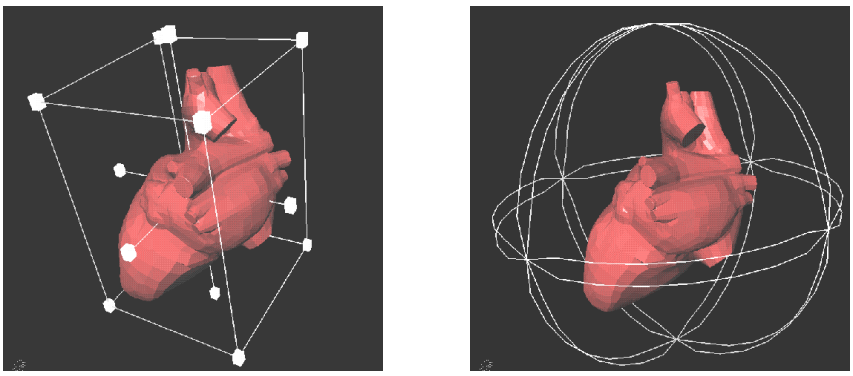
2.2.3 Die Open Inventor Umgebung

Das *Open Inventor* System stammt von der Firma *Silicon Graphics* und ist eine objektorientierte Klassenbibliothek, die den interaktiven Umgang mit 3D-Modellen ermöglicht.

- **Ein Szenegraph repräsentiert die 3D-Umgebung** und kann durch ein Programm beliebig manipuliert werden.
- **Sensoren** können dazu verwendet werden, auf Eingaben des Betrachters zu reagieren und Funktionen aufzurufen. Die Benutzerschnittstelle von *Open Inventor* realisiert Shneidermans Konzept der direkten Manipulation.
- Die Klassen des 3D-Toolkits ermöglichen mittels **3D-Widgets** die Manipulation der Modelle mit der Maus, wobei sofort eine visuelle Reaktion erfolgt. Ein virtueller Trackball (siehe Abbildung 2.7(a)) ermöglicht das Drehen, eine virtuelle Handlebox (siehe Abbildung 2.7(b)) das Skalieren des Modells. Allgemein ist die Steuerung über eine 2D-Eingabe mit der Maus schwierig, da das Modell in drei Achsen verändert werden kann. Daher gibt es für jede Achse ein eigenes *Handle* (deutsch Griff), das vor unkontrollierten Bewegungen durch eine Beschränkung der Freiheitsgrade schützt. (Für eine Beschreibung der Konzepte siehe [SC92]).

2.3 Grundlagen der Computergraphik

Ein System zur Navigationshilfe benötigt Informationen über Struktur und Funktionalität der Welt, um dem Benutzer intelligente Abstraktionen zu generieren. Bei der Abstraktion werden die Modelle der Welt geometrisch verändert und ihre Darstellung manipuliert. Das



(a) Trackball für Rotation; 6 Handles

(b) 6 Handles für Translation und 6 Handles für Skalierung

Abbildung 2.7: Interaktionselemente für Transformation von 3D-Objekten in *Open Inventor*

Verständnis der Repräsentation und der Darstellung von 3D-Welten ist also von hoher Bedeutung für diese Arbeit. Beide Themen sind Inhalt der Computergraphik und werden in [FvDFH96] ausführlich behandelt. Dieses Kapitel soll die Thematik in ihren Grundzügen vorstellen. In Abschnitt 2.3.1 gibt es einen kurzen Überblick über die Möglichkeiten zur Modellierung von Objekten. Abschnitt 2.3.2 beschreibt deren Anordnung zu einer komplexen Szene mittels einer Baumstruktur. Abschnitt 2.12 beschreibt den Vorgang der stufenweisen Abstraktion in Analogie zu den Stufen der Bilderzeugung.

2.3.1 Repräsentation der Daten

Grundsätzlich gibt es zwei Wege, um Objekte der realen Welt virtuell zu modellieren. Zum einen können geometrische Primitive wie Quader, Zylinder oder Kugeln verwendet werden. Diese sind mathematisch exakt definiert und benötigen nur wenige Parameter wie Position und Größe zu ihrer Repräsentation. Aus ihnen lassen sich durch Verknüpfungen wie Addition, Subtraktion oder Schnitt komplexere Körper zusammenbauen. Diese Methode der Modellierung wird als *CSG*[1] bezeichnet. Für technische Objekte wie Maschinen oder Möbel ist diese Methode vorteilhaft, da diese traditionell durch Verwendung geometrischer Grundformen am Reissbrett entstehen.

Unregelmässige organische Formen, wie z.B. ein menschlicher Körper, lassen sich so jedoch nur sehr grob modellieren. In solchen Fällen kommen approximative Verfahren wie *Voxel* oder *Polygonmeshes* zum Einsatz.

In der Medizintechnik werden gescannte Patientendaten ähnlich einer Bitmap gerastert. Dabei werden viele horizontale Schnitte zu einem Volumenmodell zusammengesetzt. So werden zweidimensionale Pixel zu Würfeln, sogenannten Voxeln. Je kleiner die Voxel, desto natürlicher wirkt das Modell. Der Nachteil dieser Methode liegt im exponentiellen Speicherplatzbedarf, bei einer Segmentierung der Kanten in n Einheiten werden n^3 Voxel benötigt.

Eine effizientere Modellierung beliebiger Formen ist durch Approximation mittels hinreichend kleiner Polygone möglich, dabei wird nur die sichtbare Oberfläche des Körpers gespeichert. Abbildung 2.8 zeigt ein solches Modell, das nur aus Dreiecken besteht. Benachbarte Dreiecke teilen eine gemeinsame Kante, so daß pro Dreick nur ein Eckpunkt gespeichert werden muß. Generell werden die Dreiecke, oder allgemeiner Polygone, durch

[1] Computational Solid Geometry

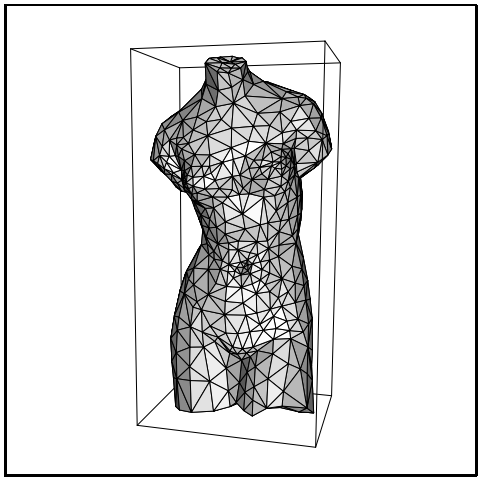


Abbildung 2.8: Mithilfe von Polygonen modellierte Venus

die Koordinaten ihrer Eckpunkte repräsentiert, üblicherweise in Gleitkommadarstellung. Darüberhinaus sind Dreiecke sehr leicht zu berechnen und darzustellen, so daß in der Regel auch Primitive wie Kugeln im letzen Schritt der Renderingpipeline durch Dreiecke approximiert und von Spezialhardware dargestellt werden.

Sogenannte *NURBS* erweitern die Möglichkeiten von Polygonmeshes durch gekrümmte Flächen, die durch Interpolation zwischen Kontrollpunkten entstehen. Diese Flächen gehen an ihren Schnittstellen nahtlos ohne Knick ineinander über und sind überall differenzierbar, also völlig glatt. Dafür ist der Aufwand zur Implementierung und Darstellung erheblich grösser als bei einfachen Polygonen, so daß sie eher in CSG basierter Software zu finden sind. Die Darstellung von Polygonen ist technisch einfacher zu realisieren und wird für Hardwarelösungen bevorzugt.

In dieser Arbeit wird für Polygonmodelle das OFF[2] Dateiformat [Geo96b] verwendet. Der Aufbau des Formats soll anhand der Beispieldatei *color_face_cube.off* (siehe Abbildung 2.9(a)) gezeigt werden, die einen farbigen Würfel beschreibt (Abb. 2.9(b)). Nach der OFF Typ-Kennung (Zeile 2) folgt die Anzahl der Knoten, Polygone und Kanten im Modell (Zeile 3). Darauf folgen die Koordinaten der Knoten (Zeilen 4-11). Im nächsten Abschnitt werden Polygone über ihre Knoten definiert (Zeilen 12-17). Jede Zeile entspricht einem Polygon. Die erste Zahl gibt die Anzahl der Eckpunkte des Polygons an, danach folgen entsprechend viele Knotenindizes. Am Ende folgt optional die Polygonfarbe, angegeben in Rot-, Grün-, Blau- und Transparenzanteilen.

Alternativ zu OFF kann das VECT Format verwendet werden, um Polygonzüge zu speichern. Dabei wird auf Indizierung verzichtet, und Polygonzüge werden direkt durch Koordinatenlisten definiert.

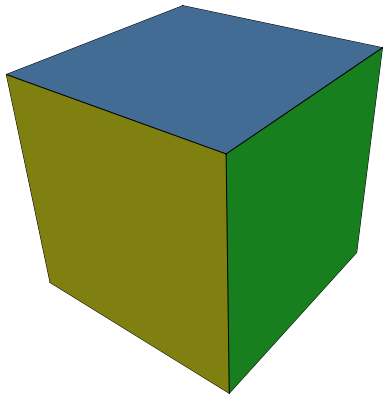
2.3.2 Anordnung von Objekten zu einer Szene

Theoretisch lassen sich mit dem gezeigten OFF-Format beliebig komplexe statische Modelle aus Polygonen repräsentieren. Um Animation und Interaktion zu realisieren, ist jedoch eine Strukturierung erforderlich. Typischerweise wird dazu eine Baumstruktur, der sogenannte *Szenegraph*, aufgebaut, um einzelne Modelle zu einer Szene zusammenzustellen. Die hierarchische Ordnung ermöglicht das Gruppieren von Objekten, so daß sie sich leichter anordnen lassen.

[2] object file format

```
1 # off file with per face color
2 OFF
3 8 6 12
4 1.0 1.0 1.0
5 1.0 1.0 -1.0
6 1.0 -1.0 1.0
7 1.0 -1.0 -1.0
8 -1.0 1.0 1.0
9 -1.0 1.0 -1.0
10 -1.0 -1.0 1.0
11 -1.0 -1.0 -1.0
12 4 0 2 3 1 .05 .8 .1 .75
13 4 4 5 7 6 .2 .05 .8 .75
14 4 0 4 6 2 .9 .9 .02 .75
15 4 1 3 7 5 .0 .7 .4 .75
16 4 0 1 5 4 .1 .4 .7 .75
17 4 2 6 7 3 .7 .7 0 .75
```

(a) Definition des Modells im OFF-Dateiformat



(b) Eine 3D-Ansicht des Modells

Abbildung 2.9: Definition eines farbigen Würfels durch Polygone

Ein Beispiel einer aus drei Primitiven bestehenden Szene ist in Abbildung 2.10 dargestellt. Die oberste Hierarchieebene wird durch die Wurzel des Baumes (A) definiert, der alle Bestandteile der Szene folgen. Um Objekte innerhalb der Welt zu plazieren, werden Transformationsknoten verwendet. Im Beispiel verschiebt der Knoten (B) alle seine Nachfolger, also den gesamten linken Zweig, gemeinsam relativ zum Ursprung. Zusätzlich werden alle Objekte von Knoten (C) gedreht. Innerhalb der Gruppe werden die Objekte Pyramide (E), Quader (F) und Kugel (G) noch einmal individuell positioniert (D). Neben der darzustellenden Geometrie enthält der Szenegraph auch eine Lichtquelle (H) zur Beleuchtung der Objekte sowie die Position und Ausrichtung einer virtuellen Kamera (J). Diese Kamera bestimmt die Perspektive der Abbildung der Szene mithilfe des Computers. Ein komplexer Szenegraph wird in Abschnitt 5.1 beschrieben.

Während für den Szenegraph die Objekte nur entsprechend technischer Gesichtspunkte wie Beleuchtung und Animation strukturiert werden, ist zusätzlich auch eine hierarchische Unterteilung der Modelle nach funktionalen Aspekten möglich. Eine solche Gliederung in logische Komponenten wird im folgenden als *Objekthierarchie* bezeichnet. Da diese zur Visualisierung der Szene nicht benötigt wird, fehlt den meisten heutigen Modellen eine solche Struktur. Für die intelligente Abstraktion einer Szene spielt sie jedoch eine wichtige Rolle. Ein Beispiel für eine Objekthierarchie findet sich auf Seite 71 in Abbildung 6.6. Dort ist ein Gebäude (das Capitol in Washington D.C.) in Haupt- und Nebenflügel unterteilt, die Nebenflügel sind weiter in drei Abschnitte gegliedert.

2.3.3 Abstraktionspipeline und Renderingpipeline

In der Computergraphik wird der Vorgang, aus dreidimensionalen Geometriedaten fotorealistische perspektivische Darstellungen zu berechnen, als *rendering*[3] bezeichnet. Die Berechnung geschieht in einzelnen Stufen, die verallgemeinert als *Rendering-Pipeline* beschrieben werden. Abbildung 2.11 zeigt diesen Vorgang. Zuerst werden die Koordinaten der Objekte aus dem Nullpunkt heraus an ihre Positionen innerhalb der Welt verschoben

[3] Englisch: Wiedergabe

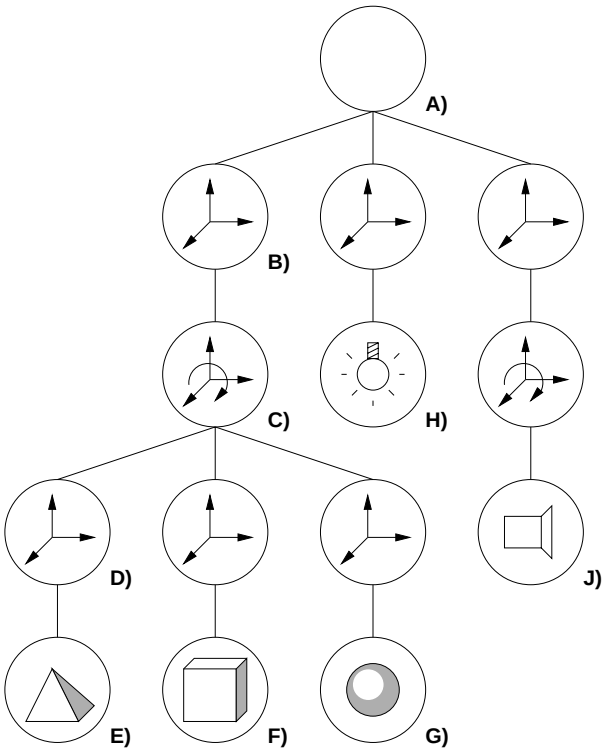


Abbildung 2.10: Beispiel eines Szenegraphen

und rotiert. Danach erfolgt die Projektion der 3D-Koordinaten auf die Bildschirmebene unter Berücksichtigung der Perspektive. Die Bildkoordinaten der projizierten Polygone werden dann gerastert. Ein typischer Computerbildschirm verwendet ein Raster aus 1024 horizontalen und 768 vertikalen Bildelementen, Pixel genannt.

Analog dazu beschreibt Krüger [Krü00] die möglichen Ebenen der Abstraktion in einer Abstraktions-Pipeline, die in Abbildung 2.12 dargestellt ist.

Die oberste Stufe bildet dabei die Modellebene, auf der die abzubildenden Objekte verändert werden. Dies geschieht durch Manipulation der dreidimensionalen Geometriedaten, die die darzustellenden Objekte mathematisch beschreiben und die dem Renderer als Eingabe dienen. In der Regel bestehen die Geometriedaten aus Listen von Polygonen, wie in Abschnitt 2.3.1 gezeigt wurde.

Die Generierungsebene umfaßt alle Veränderungen der Darstellungsparameter wie Beleuchtung und Oberflächeneigenschaften der Objekte. So lassen sich Objekte farblich oder durch Verwendung von Spotlights hervorheben. Umgekehrt ist es möglich, Objekte durch Transparenz oder durch Darstellung in einheitlichem Grau in den Hintergrund zu rücken.

Der Rendervorgang erzeugt eine zweidimensionale Abbildung, die in der letzten Stufe, der Bildebene, noch einmal mit Hilfe von pixelbasierten Methoden vereinfacht werden kann, wie sie z.B. das Bildbearbeitungsprogramm *GIMP* [SK00] bietet.

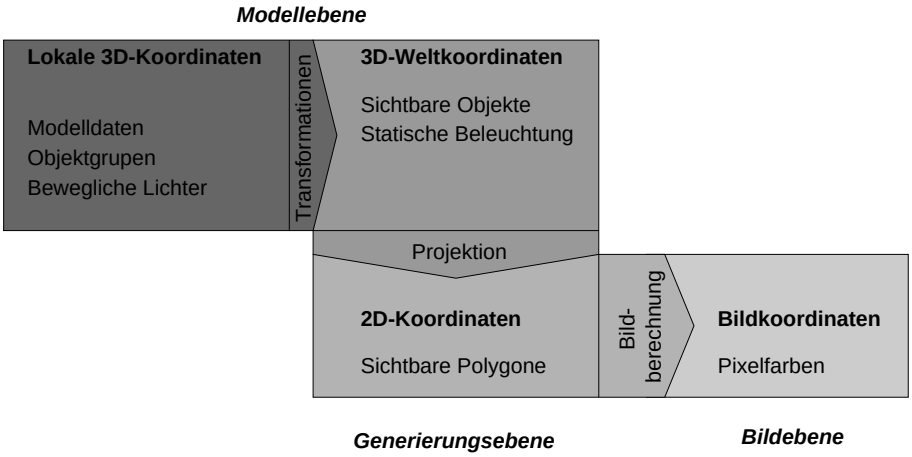


Abbildung 2.11: Darstellung der Rendering-Pipeline.

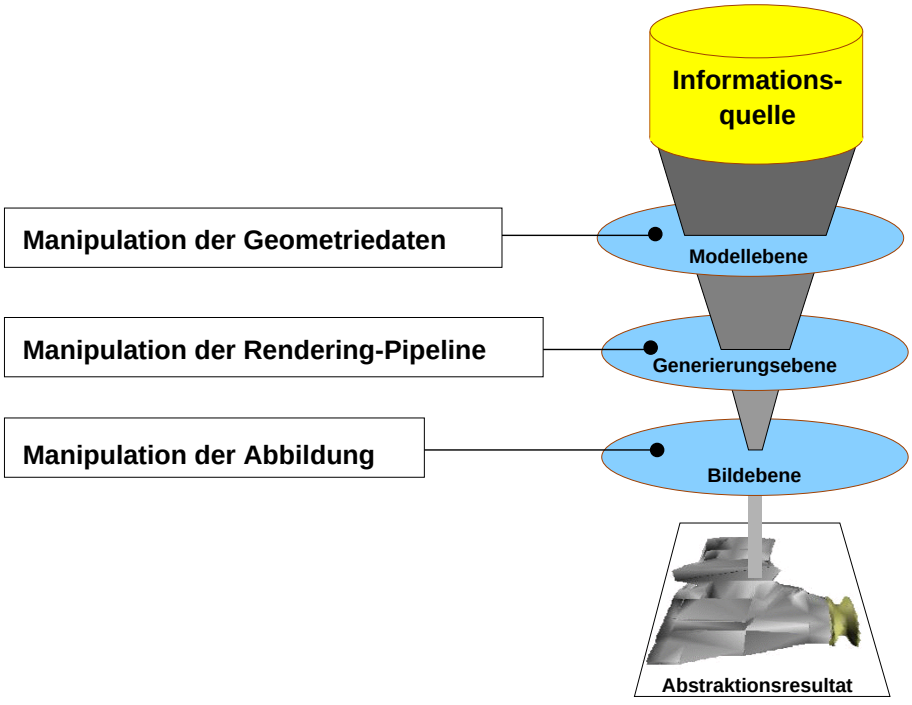


Abbildung 2.12: Darstellung der Abstraktions-Pipeline.

2.3.4 Das Level Of Detail Konzept

Mit der *Level Of Detail* Modellierung macht man sich die Eigenschaft zunutze, daß Objekte umso kleiner erscheinen, je weiter sie vom Betrachter entfernt sind. Je kleiner die darzustellenden Polygone werden, desto weniger fallen sie in einer Szene ins Gewicht. Um die Bildgenerierung zu beschleunigen, können Polygone ab einer gewissen Fläche approximiert werden, ohne die Bildqualität zu beeinträchtigen. Für diesen Ansatz bietet VRML die Möglichkeit, für ein Objekt mehrere Polygonmodelle in verschiedenen Detaillierungen anzugeben. Der VRML- Viewer wählt dann je nach Entfernung ein Modell mit so wenig Details wie nötig. Dabei kommt es zu sichtbaren Unstetigkeiten, wenn zwischen verschiedenen Modellen umgeschaltet wird. Einen geschickteren Ansatz bieten sogenannte *Geomorphs*. Das sind Datenstrukturen, die ein Modell mit allen Details beschreiben, aber gleichzeitig alle Abstraktionsgrade mit geringem Aufwand daraus ableiten lassen. Ein Beispiel dafür sind Hoppes *Progressive Meshes* [Hop96].

2.4 Abstraktionstechniken

In diesem Abschnitt werden einige Verfahren vorgestellt, die aus der Literatur bekannt sind und sich zur graphischen Abstraktion eignen. Dabei wird entsprechend der Definition der Abstraktionspipeline zwischen Techniken auf Modell-, Generierungs- und Bildebene unterschieden.

2.4.1 Abstraktion auf Modellebene

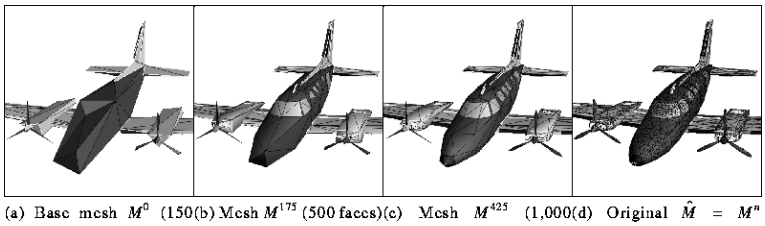
Im Bereich der Modellebene wurden verschiedene Verfahren zur Datenreduktion von Polygonmodellen entwickelt. Dabei ging es in erster Linie darum, die Anzahl der Polygone zu verringern, um dadurch den Rechenaufwand zur Darstellung der Modelle zu reduzieren und höhere Bildwiederholraten zugunsten der Interaktion zu erreichen. Obwohl die Zielsetzung bei der Entwicklung der Verfahren eine andere war, lassen sie sich gut zur Abstraktion einsetzen. Denn mit dem Entfernen von Polygonen gehen gerade die Details und individuellen Merkmale eines Modells verloren, während die Grundform beibehalten wird.

Das *Progressive Meshes* Verfahren von Hoppe [Hop96] erzeugt eine kompakte Darstellung beliebiger Reduktionsgrade. Hoppe definiert eine Energiefunktion, die sich aus der Knotenanzahl des reduzierten Modells und dem durch die Reduktion verursachten Fehler zusammensetzt. Ziel ist eine Minimierung dieser Funktion. Die Knoten werden durch eine Operation reduziert, die zwei benachbarte Knoten vereint und dadurch eine Kante entfernt. In einem Progressive Mesh wird das reduzierte Modell zusammen mit allen Kantenoperationen gespeichert. Durch inverses Ausführen der Operationen können alle Reduktionsgrade bis hin zum Original wiederhergestellt werden. Abbildung 2.13 zeigt ein Beispiel.

Rossignac [RB92] schlägt ein Filterverfahren vor, das aufgrund seiner Einfachheit ein hervorragendes Laufzeitverhalten aufweist. Das Polygonmodell wird durch ein räumliches Raster in kubische Zellen partitioniert. Rossignac bezeichnet diesen Vorgang als ”clustering”. So wird jeder Knoten eines Polygons einem Cluster (Zelle) zugeordnet, wobei in der Regel mehrere Knoten auf denselben Cluster abgebildet werden. Nun werden alle Knoten innerhalb einer Zelle auf ihren gemeinsamen Mittelpunkt abgebildet. Dabei kann eine Gewichtung der Knoten nach verschiedenen Kriterien vorgenommen werden, um das Ergebnis zu verbessern. Dreiecke, deren Eckpunkte vollständig in einer Zelle liegen, kollabieren zu einem Punkt und werden entfernt. Liegen nur zwei Eckpunkte innerhalb einer gemeinsamen Zelle, degeneriert das Dreieck zu einer Linie, die optional als solche repräsentiert oder entfernt wird. Je grösser die Cluster sind, desto kleiner ist die resultierende Auflösung, und das Ergebnis wird abstrakter, da verhältnismässig mehr Knoten pro Cluster vereint werden. Abbildung 2.14(a) zeigt die Partitionierung einer Birne in Cluster, das Ergebnis der Abstraktion ist in Abbildung 2.14(b) dargestellt.

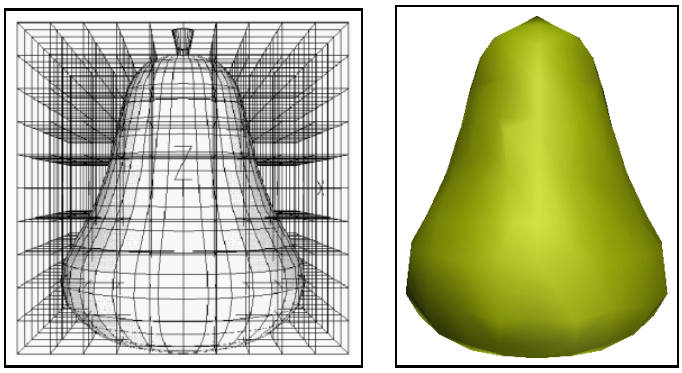
Das Abstraktionswerkzeug *BUNDY* [KS97] implementiert und verbessert den Rossignac-Algorithmus. Insbesondere ermöglicht *BUNDY* neben kubischem auch sphärisches Clustering, das bei kugelförmigen Objekten an den Polen bessere Ergebnisse liefert. Daneben kann über eine inhomogene Clusterdichte eine Fokussierung innerhalb der Abstraktion erreicht werden. *BUNDY* stellt eine Benutzeroberfläche zur Verfügung, um interaktiv Abstraktionen von Polygonmodellen zu erstellen und mit verschiedenen Parametern zu experimentieren. Abbildung 2.4.1 zeigt das Polygonmodell eines Hasen in verschiedenen Abstraktionsgraden, wobei die Anzahl der Polygone stets abnimmt. Bei Modellen mit einer Objekthierarchie können für verschiedene Komponenten unterschiedliche Abstraktionsgrade gewählt werden. Abbildung 2.16(a) zeigt das unveränderte Polygonmodell eines Motorblocks. In Abbildung 2.16(b) wird auf den Zylinderkopf fokussiert, indem der übrige Motor leicht abstrahiert wurde. Der Fokuseffekt wird im Vergleich mit Abbildung 2.16(c) besonders deutlich. In diesem Bild ist der gesamte Motor abstrakt dargestellt. Zusätzlich bietet *BUNDY* eine Verschmelzungsoperation, die Lücken zwischen benachbarten Objekten schliesst. Ein Beispiel zu dieser Operation ist in Abbildung 2.17(a) durch zwei Scheunen gegeben. Es wird der Raum zwischen den Scheunen bestimmt, das Ergebnis ist in 2.17(b) angedeutet. In Abbildung 2.17(c) ist der Zwischenraum gefüllt, und die Scheunen sind verschmolzen..

Preim verwendet in seinem System *Zoom Illustrator* [Pre98] die Technik des Fisheye-Zoomens. Der Begriff Fisheye ist in Anlehnung an Fisheye-Objektive in der Fotographie entstanden, die mittels nichtlinearer Verzerrungen Teile im Zentrum der Szene größer darstellen als am Rand. Preim vergrößert die Modellgeometrie der fokussierten Objekte, so daß Details deutlicher zu sehen sind.



(a) Base mesh M^0 (150 facets) (b) Mesh M^{175} (500 facets) (c) Mesh M^{425} (1,000 facets) (d) Original $\hat{M} = M^n$ (13,546 facets)

Abbildung 2.13: Ein Flugzeugmodell in verschiedenen Abstraktionsgraden. Quelle: [Hop96]



(a) Clustering (b) $9 \cdot 9 \cdot 9$ Cluster

Abbildung 2.14: Rossignac Algorithmus

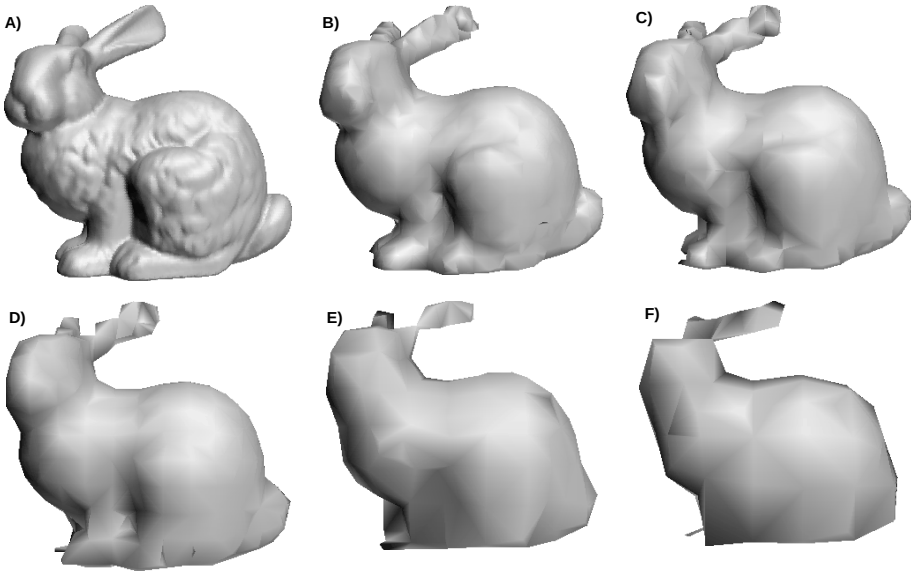
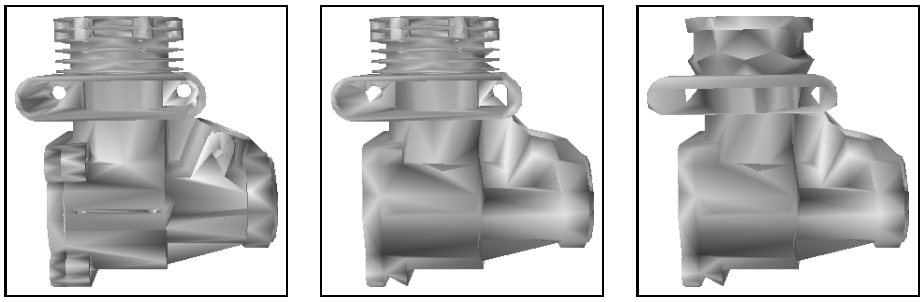
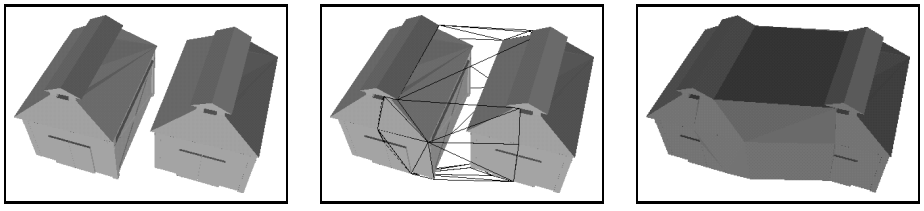


Abbildung 2.15: Verschiedene Abstraktionsgrade desselben Modells, erzeugt von *BUNDY* durch Variation der Clustergrößen. A) zeigt das Original-Modell. B) ist mit 8.000 Clustern, C) mit 4.096 Clustern, D) mit 1.728 Clustern, E) mit 512 Clustern und F) mit 216 Clustern berechnet worden.



(a) Detailliert (b) Fokus auf Zylinderkopf (c) Abstrakt

Abbildung 2.16: Abstraktion eines Motorblocks durch den Rossignac Algorithmus



(a) (b) (c)

Abbildung 2.17: Zwei Scheunen werden von BUNDY verschmolzen

2.4.2 Abstraktion auf Generierungsebene

Auf der Generierungsebene können spezielle, nicht-photorealistische Renderingtechniken zur Abstraktion verwendet werden. Diese Techniken versuchen illustrative Abbildungen zu erzeugen, die mehr Aussagekraft enthalten, als es eine reine Simulation der physikalischen Gesetze der Optik zulässt. Dazu werden klassische Rendering-Algorithmen erweitert. Man unterscheidet dabei zwischen Pixel- und Linienbasierten Verfahren. Pixelbasierte Verfahren bestimmen zu jedem Bildpunkt nicht nur Farbe und Helligkeit, sondern auch Informationen wie z-Koordinate, Winkel und Krümmung, welche als Zwischenergebnisse beim Rendering anfallen. In einem Nachbearbeitungsschritt wird die Farbe jedes Bildpunktes aus all diesen Informationen bestimmt. So können Oberflächen und Schatten durch Schraffuren ähnlich einem Kupferstich gebildet werden. Bei Linienbasierten Verfahren wird das Bild nicht durch Pixel, sondern durch sichtbare Kanten beschrieben. Diese Linien können in Zeichenstil und Strichstärke variiert werden, um Effekte wie Beleuchtung oder Bewegung ähnlich einem menschlichen Zeichner umzusetzen. Durch das gezielte Entfernen von Kanten können Skizzendarstellungen wie in Abbildung 2.18 erzeugt werden. Dabei sollen nur diejenigen Kanten eines Polygonmeshes gezeichnet werden, die auf der Silhouette liegen oder markante Details wiedergeben. Hierzu werden zwei Verfahren kombiniert, wie es im zweiten Bild von links zu sehen ist. Erstens werden abhängig von der Perspektive nur solche Kanten gewählt, deren Skalarprodukt von Normalenvektor und Blickrichtungsvektor größer als Null ist. Zweitens wird davon ausgegangen, daß Kanten relevant sind, die einen spitzen Winkel zwischen benachbarten Polygonen bilden.

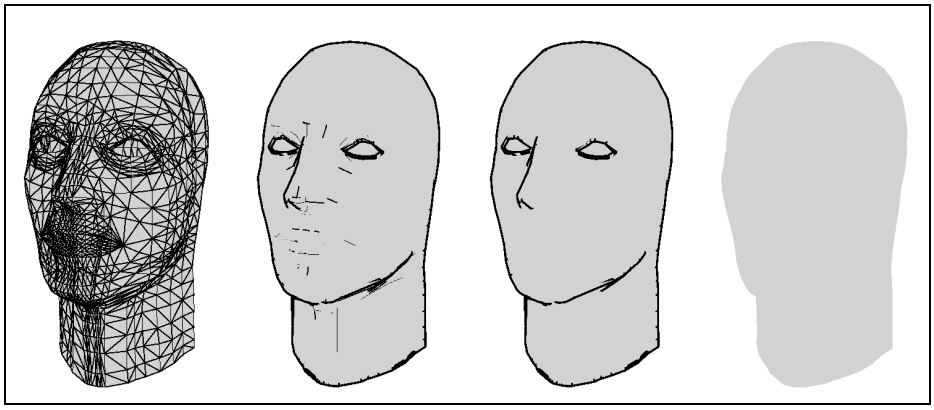
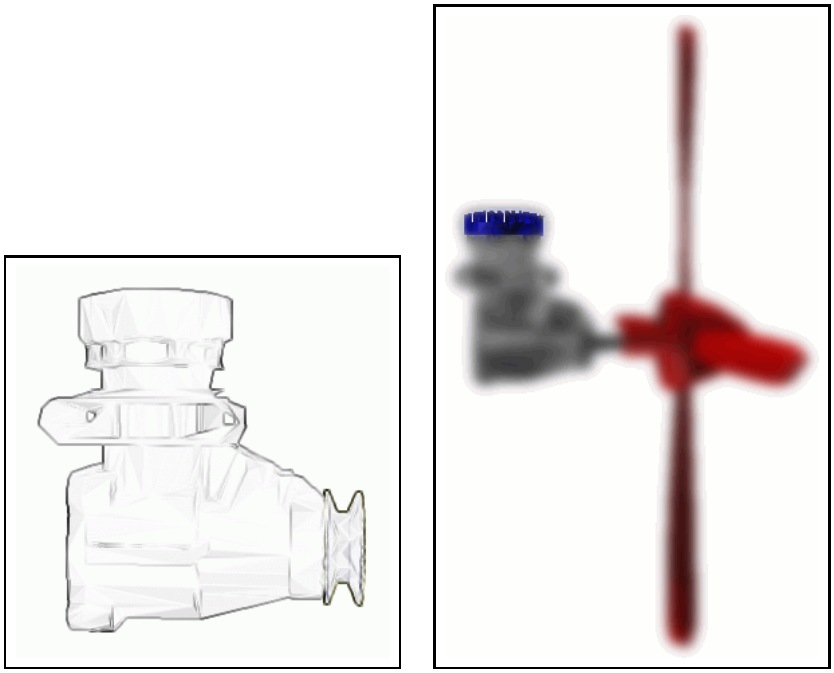


Abbildung 2.18: Skizzendarstellung durch Entfernen von Kanten [Krü00]

2.4.3 Abstraktion auf Bildebene

Das Ergebnis der Rendering-Pipeline ist eine Abbildung der Szene in Form einer Bitmap, die sich aus einzelnen Bildpunkten zusammensetzt. Auf dieser Bitmap lassen sich über eine einfache Matrix verschiedene Filter realisieren, die zur Abstraktion eingesetzt werden können. Dabei wird jeder Bildpunkt durch die gewichtete Summe der Pixelfarben seiner Umgebung ersetzt. Solche matrixbasierten Operationen sind typischer Bestandteil von Bildbearbeitungsprogrammen.

Sind alle Gewichte in der Matrix gleichmässig verteilt, ergibt sich ein Unschärfefilter. Ein Beispiel dafür zeigt Abbildung 2.19(b). Motorblock und Propeller sind unscharf dargestellt, während der Zylinderkopf unverändert scharf gezeigt wird. Auf diese Weise wird in der Abbildung eine Fokussierung auf den Zylinderkopf erzielt. Verwendet man eine Matrix, die anstatt des Durchschnitts die Differenz benachbarter Pixel bestimmt, werden Kanten besonders hervorgehoben. Das Ergebnis dieser Operation ist in Abbildung 2.19(a) zu sehen. Diese und weitere Filter werden in [Umb98] beschrieben.



(a) Kanten hervorheben (b) Unschärfe

Abbildung 2.19: Bitmap Filter

2.5 Abstraktionssysteme

Im vorangegangenen Abschnitt wurden Techniken zur graphischen Abstraktion eingeführt, die auf Polygonmodelle anwendbar sind. In diesem Abschnitt werden Abstraktionssysteme vorgestellt, die mithilfe dieser Techniken aus komplexen Szenen Illustrationen mit einem kommunikativen Ziel erzeugen.

2.5.1 APEX

APEX [Fei85] wurde 1985 von Feiner im Rahmen seiner Doktorarbeit veröffentlicht und war das erste System zum automatischen Erzeugen von abstrakten Illustrationen.

Der Abstraktionsvorgang basiert vollständig auf der Objekthierarchie der Einzelkomponenten eines Modells, die einer Baumstruktur entsprechen muß. Um das System leicht implementieren zu können, sind die Teilobjekte auf achsenparallele Quader beschränkt.

Der Abstraktionsalgorithmus erzeugt eine Abstraktionshierarchie. Er beginnt damit, die Blätter des Baumes zu einer Abstraktion zusammenzufassen und durch diese zu ersetzen. Dabei werden benachbarte und farblich ähnliche Quader vereinigt, und besonders kleine Quader werden entfernt. Diese Operation wird rekursiv bottom-up durchgeführt, bis jeder Knoten des Baumes seine Abstraktion enthält. Je nach kommunikativem Ziel wird nun für jeden Zweig entweder die detaillierte Originalgeometrie oder die Abstraktion verwendet und daraus die Gesamtabstraktion des Objektes zusammengesetzt. Es ist leicht zu sehen, das bei dieser Vorgehensweise das Ergebnis vor allem von der Objekthierarchie des Modells abhängt. Die darzustellenden Objekte müssen also mit besonderer Sorgfalt modelliert und in einem strukturerhaltenden Format gespeichert werden.

APEX arbeitet dabei nur mit syntaktischem Geometriewissen. Informationen über die Semantik bezüglich der Aussagekraft der Einzelteile innerhalb der Hierarchie sind nicht

vorhanden. Der Prozeß des Verschmelzens kann also nicht gesteuert werden, um einzelne Objekte davon auszuschliessen. Daher kann nicht auf einzelne Blätter, sondern nur auf Knoten innerhalb der Hierarchie fokussiert werden. Durch die Einschränkung auf Quader ist das System in der Praxis heutzutage nicht anwendbar, da in der Regel Polygonmodelle vorliegen.

2.5.2 ARP

Das Abstraktionssystem ARP entstand 1999 im Rahmen der Dissertation von Krüger [Krü00]. Das Navigationshilfesystem VAI entstand parallel zu ARP, zum Nutzen beider Systeme. VAI ergab eine Beispielanwendung für die Fähigkeiten von ARP, umgekehrt wurde bei der Implementation von ARP auf technische Kompatibilität zu VAI geachtet.

ARP generiert automatisch abstrakte Abbildungen von 3D-Modellen anhand eines gegebenen Abstraktionsziels. Bei der Generierung wird eine Abstraktionshierarchie aufgebaut, ähnlich der von APEX. Neu an ARP ist die Fähigkeit, beliebige Polygonmodelle zu verschmelzen und zu abstrahieren, während die früheren Systeme nur geometrische Primitive wie Quader und Zylinder verarbeiten konnten. ARP verwendet den Rossignac-Algorithmus zur Abstraktion auf Modellebene.

Ähnlich APEX benötigt auch ARP eine detaillierte Objekthierarchie für gute Ergebnisse. Heute wird die Struktur der Modelle üblicherweise nur durch den Konstruktionsvorgang und den Szenegraphen bestimmt. Daher bietet ARP die Möglichkeit, die Objekthierarchie von Hand zu verfeinern und propositionales Wissen über den logisch-funktionalen Zusammenhang der Objekte hinzuzufügen.

Das Abstraktionsziel wird formuliert, indem jedes Teilobjekt aus der Hierarchie einer Darstellungsklasse zugeordnet wird. Die Zuordnung erfolgt mit Hilfe von signifikanten Attributen, die während des Abstraktionsvorganges darüber entscheiden, ob das Teilobjekt verschmolzen und abstrahiert werden darf oder detailliert dargestellt werden muß. Der Abstraktionsalgorithmus wird dazu durch Regelwissen gesteuert. So ist es möglich einen Fokus gezielt auf einzelne Bestandteile des darzustellenden Objektes zu setzen.

Ein wichtiges Ziel bei der Entwicklung von ARP war die Fähigkeit, bei der Bilderzeugung sowohl technische Ressourcenbeschränkungen als auch unterschiedliche Benutzerprofile zu berücksichtigen. So kann ARP dazu verwendet werden, aus denselben Modelldaten Graphiken für verschiedene Ausgabemedien zu erzeugen. Bei Displays mit einer geringeren Auflösung wird stärker abstrahiert. Sind weder Farben noch Graustufen darstellbar, werden einfache Skizzen ohne Schattierung erzeugt. Für Laien werden detailliertere Abbildungen generiert als für Experten, denen zusammen mit ihrem Hintergrundwissen einfache Skizzen genügen.

Zum Erzeugen graphischer Abstraktionen auf Modellebene steht ARP mit dem Programm BUNDY [KS97] eine Implementation des Algorithmus von Rossignac zur Verfügung. Dieser Algorithmus stellt einen Filter dar, der die Anzahl der Knoten und Polygone in einem 3D-Modell beliebig reduziert und dabei sehr stabil und schnell ist. Zusätzlich wurde ein Algorithmus zum Verschmelzen von Polygonmodellen in BUNDY implementiert.

Abbildung 2.20 zeigt als Beispielanwendung von ARP eine Betriebsanleitung für einen Videorecorder. Die Illustrationen heben die im jeweiligen Schritt der Anleitung relevanten Bedienelemente hervor. Als Basis liegt ein Polygonmodell vor, das um propositionales Wissen über die Funktionalität der Tasten ergänzt worden ist. Zur Fokussierung werden die entsprechenden Tasten nicht markiert, sondern die im jeweiligen Kontext irrelevanten Tasten abstrahiert. Dazu werden diese entweder vollständig entfernt oder zu einer Einheit verschmolzen, um die Anzahl der dargestellten Elemente zu verringern.

A)

B)

"Starten Sie die Wiedergabe...."

C)

"Drücken Sie die Standbildtaste, um die Wiedergabe anzuhalten."

D)

"Benutzen Sie die Justiertasten, um die Bildqualität zu verbessern."

Abbildung 2.20: Beispiel einer automatisch generierten Betriebsanleitung für Videorecorder unter Verwendung von graphischer Abstraktion

2.5.3 Textillustrator

Die Aufgabe eines Illustrators besteht darin, zu einem gegebenen Text spezielle skizzenhafte Abbildungen anzufertigen, um dem Leser die Zusammenhänge zu veranschaulichen. Dabei ist es besonders wichtig, die Aufmerksamkeit des Betrachters auf die zum Verständnis relevanten Teile der Abbildung zu lenken. Dazu werden diese Teile möglichst detailgetreu dargestellt und eventuell zusätzlich durch besondere stilistische Merkmale hervorgehoben. Umgekehrt sollten die übrigen Elemente so weit wie möglich in den Hintergrund treten. Dies wird oft durch die Technik der graphischen Abstraktion erreicht.

Schlechtweg stellt in seiner Dissertation [Sch99] Verfahren und Techniken vor, um den Vorgang des Illustrierens durch den Computer zu unterstützen. Dazu untersucht er den Begriff der Illustration und die Möglichkeiten der graphischen Abstraktion sowie nicht-photorealistische Renderingverfahren zum Erzeugen von skizzenhaften Darstellungen. Ergebnis seiner theoretischen Untersuchungen ist das prototypische System TextIllustrator.

Das Programm TextIllustrator wird vom Benutzer interaktiv gesteuert. Auf der linken Bildschirmhälfte befindet sich eine dreidimensionale Ansicht der Objekte, auf der rechten Seite der zu illustrierende Text. Wird im Text gescrollt, verändert sich automatisch die Grafik und zeigt detailliert die Objekte, von denen in der gerade sichtbaren Textpassage gesprochen wird. Die Objekte werden zusätzlich mit Beschriftungen versehen. Umgekehrt führt das Selektieren eines bestimmten Objekts in der Illustration dazu, daß relevante Textpassagen angezeigt werden. Wird hingegen ein Wort, das in Relation zu einem bestimmten geometrischen Objekt steht, im Text ausgewählt, so wird das Objekt detaillierter dargestellt oder durch eine besondere Darstellungsweise (wie z.B. Farbveränderung) hervorgehoben.

Das Programm kann einerseits vom Leser zum interaktiven browsing in einer Multimediaumgebung verwendet werden. Schlechtweg schlägt hierzu als Anwendungsdomäne einen Anatomieatlas für Medizinstudenten vor. Andererseits kann das System von Illustratoren genutzt werden, um Abbildungen zu Texten in Papierform zu erzeugen. Als Beispiel hat Schlechtweg aus einem Polygonmodell Skizzen für die Bedienungsanleitung einer Kamera angefertigt.

Ein vollautomatisches Generieren von Skizzen wurde nicht angestrebt, da die künstlerische Erfahrung eines menschlichen Illustrators kaum in ein Programm zu fassen ist.

Möglich wurde die Kopplung von Text und 3D-Graphik durch die Erweiterung der jeweiligen Datenstrukturen. Zu Stichworten wurden Referenzen auf entsprechende Textpassagen gespeichert. Umgekehrt wurden die Geometrie-Informationen mit Referenzen auf Stichworte ergänzt.

Zum Erzeugen von aussagekräftigen Illustrationen verwendet Schlechtweg die Technik der graphischen Abtraktion. Dabei gleichen sich die Ziele der Illustration und der Navigationshilfe. In beiden Fällen soll die Aufmerksamkeit auf das Wesentliche fokussiert werden, indem unbedeutende Details reduziert werden. Ein entscheidender Unterschied zu meiner Arbeit besteht in der Abstraktionspipeline, zur Begriffserklärung siehe Abschnitt 2.12. Bei Schlechtweg findet die Abstraktion auf der Generierungsebene und vor allem auf der Präsentationsebene statt. Bei VAI dagegen wird die Abstraktion auf der Modellebene angewandt. Somit unterscheiden sich auch die Abstraktionsmethoden. Bei VAI werden die 3D Geometrie Informationen abstrahiert, während die nicht-photorealistischen Verfahren (wie der Skizzenrenderer) auf einer 2D- Projektion als Eingabe basieren.

2.5.4 VisDok

Bei *VISDOK* [HHS98] handelt es sich um ein System zur intelligenten Generierung und Nutzung von technischer Dokumentation. Ausgehend von Konstruktionsdaten und formalen Funktionsbeschreibungen werden für den Benutzer zur Laufzeit textuelle und visuelle Informationen erzeugt und als animierte Präsentation zur Verfügung gestellt. Sowohl der beschreibende Text als auch die kontextsensitiven Menüs in der Graphik bieten Hyperlinks an, mit denen der Benutzer das dokumentierte Produkt interaktiv erkunden kann.

Als zusätzlicher Zugriffsmechanismus steht dem Betrachter ein Index aller Objektklassen zur Verfügung. Über ihre Namen können Instanzen dieser Klassen selektiert werden. Die selektierten Objekte werden hervorgehoben, indem alle übrigen Teile auf der Generierungsebene abstrahiert werden. Als Technik wird dazu eine Skizzendarstellung verwendet. Dabei werden nur scharfe Objektkanten und Silhouetten gezeichnet, wobei abhängig vom Blickpunkt stets eine neue Berechnung erforderlich ist.

Abbildung 2.21 zeigt dazu ein Beispiel, in dem Wechselstrommotoren hervorgehoben sind. Die gesamte Szene mit Ausnahme der vier Motoren ist als Liniengraphik dargestellt.

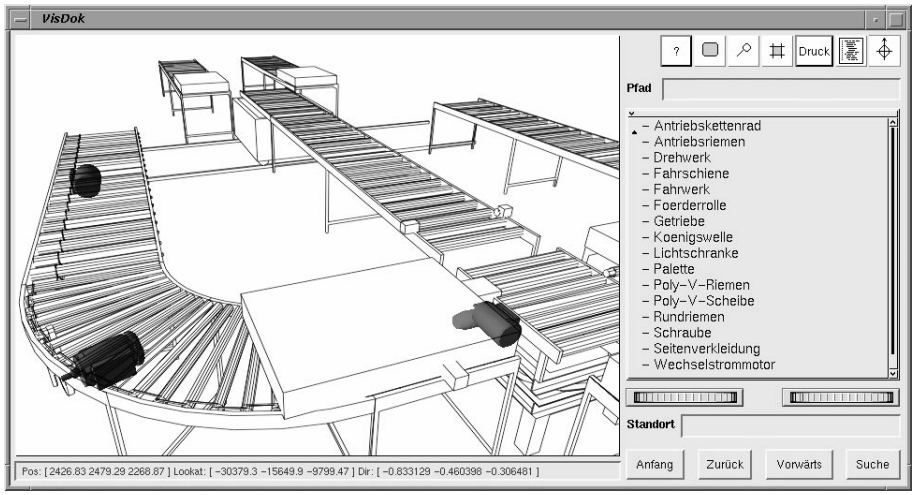


Abbildung 2.21: Hervorhebung in Visdok durch Abstraktion mittels Skizzenrendering

2.6 Die 3D-Web Beschreibungssprache VRML

Als Anwendungsdomäne der Navigationshilfe wurden Welten gewählt, die mit Hilfe von VRML97 modelliert wurden. Der folgende Abschnitt soll die Möglichkeiten dieser Sprache kurz aufzählen und dann einen kleinen Einblick in die technischen Konzepte geben. Eine umfassende Dokumentation zu VRML findet sich in [CB97].

2.6.1 Entstehungsgeschichte

Die Sprache VRML[4] wurde vom web3D Konsortium ins Leben gerufen. Das web3D Konsortium wurde 1994 gegründet mit dem Ziel einen offenen Standard für 3D Inhalte im WWW[5] zu schaffen. Beispiele für die Verwendung von 3D-Darstellung sind Visualisierung von Daten, Multimedia-Präsentationen, Lehre und Unterhaltung. Zahlreiche Firmen sind an web3D beteiligt und erhoffen sich einen neuen Markt durch ein schnelles Wachstum von 3D-Anwendungen im Internet.

Als erster Schritt wurde die VRML1.0 Spezifikation entwickelt. Damit lassen sich Geometrie und äussere Erscheinung der 3D-Objekte sowie die Anordnung der Objekte zu komplexen Szenen beschreiben. Dabei wurden bei der Sprachentwicklung folgende Designziele realisiert: Objekte lassen sich leicht wiederverwenden, und Szenen sind durch Vernetzung über Hyperlinks beliebig skalierbar. Die Sprache ist erweiterbar und systemunabhängig.

In der nächsten Stufe 2.0 kamen die Fähigkeiten Animation und Dynamik hinzu. Es lassen sich verschiedene Medien wie animierte Graphik, Sound und HTML Dokumente

[4] Virtual Reality Modeling Language
[5] World Wide Web

integrieren. Durch die Unterstützung von Scriptsprachen wie JavaScript kann man komplexes Verhalten von Objekten programmieren. Zusätzlich besteht die Möglichkeit die VRML Szene durch Java Applets zu manipulieren. 1997 wurde die Spezifikation VRML2.0 unter dem Namen VRML97 [VRM97] zum Standard ISO/IEC 14772 international anerkannt. In naher Zukunft wird VRML durch X3D[6] abgelöst werden. X3D repräsentiert die Geometrie-Informationen und Animationsmöglichkeiten von VRML mittels XML[7]. Inzwischen sind auch VRML- Importmodule für Java3D erhältlich.

2.6.2 Einführung in das VRML-Konzept

Alle Bestandteile der virtuellen Welt werden durch Knoten eines Baumes repräsentiert, dem sogenannten Szenegraphen. Es gibt spezielle Knoten für Geometrien, Lichtquellen, Sensoren und so fort. Die verschiedenen Informationen eines Knotens werden in entsprechenden typisierten Datenfeldern gespeichert. Geometrie-Informationen werden beispielsweise in Form von Polygonlisten in sogenannten *Shape* Knoten gespeichert, und die dazugehörigen Farbinformationen werden in *Appearance* Knoten abgelegt. Objekte können in *Group* Knoten gruppiert werden und durch *Transform* Knoten gedreht und positioniert werden. Dynamik in Szenen wird durch ein *Event* Modell realisiert. Knoten können entsprechend ihrem Typ sowohl Ereignisse (Events) erzeugen als auch empfangen. Ein Ereignis ist ein Wert beliebigen Typs wie z.B. eine Farbe, eine Zeit oder ein Koordinatenvektor. Ereignisse werden immer exakt in der Reihenfolge ihrer Entstehung verarbeitet. *Routen* stellen beliebige Verbindungen zwischen Sendern und Empfängern von Ereignissen innerhalb des Szenegraphen her. Eine einfache Animation wie ein sich drehender Würfel kann folgendermassen realisiert werden: Ein *Timer* Knoten erzeugt in regelmässigen Abständen Zeit-Ereignisse, die an einen *Interpolator* Knoten geroutet werden. Dieser Interpolator Knoten erzeugt nun Winkel über einem Intervall von 0 bis 360 Grad. Diese Winkel werden nun als Gleitkommazahl-Ereignis zu einem Transform Knoten gesendet. Dieser bewirkt die Drehung aller seiner Söhne, insbesondere des Würfels, um den empfangenen Winkel. Zur Interaktion mit dem Betrachter der Szene stehen verschiedene Sensoren zur Verfügung. Ein *TouchSensor* kann einer Geometrie, z.B. einem aus Polygonen modellierten Schalter, zugeordnet werden und erzeugt jedesmal ein Ereignis, wenn der Betrachter mit der Maus auf diese Geometrie klickt. Ein JavaScript Programm könnte diese Ereignisse empfangen und darauf reagieren, indem es ein Licht abwechselnd ein- und ausschaltet. Mit Hilfe von ähnlichen Sensoren lassen sich auch Objekte mit der Maus hin- und herbewegen. *Proximity* Sensoren reagieren, wenn der Betrachter sich mit seinem *Avatar*, seinem virtuellen Charakter innerhalb der 3D-Welt, einer bestimmten Position nähert.

Ein komplexes Anwendungsbeispiel der hier vorgestellten VRML Konzepte bietet der Szenegraph des Online-VAI Systems. Dieser wird in Abschnitt 5.1 im Detail vorgestellt.

2.6.3 External Authoring Interface

Das External Authoring Interface, im Folgenden kurz EAI, ermöglicht eine externe Steuerung des VRML Browsers. Die Schnittstelle wird in [Aut99] spezifiziert, und es sind entsprechende Klassen für Java verfügbar. Allgemein bietet das EAI einfache Funktionen, um beispielsweise VRML Szenen in den Browser zu laden. Daneben ermöglicht es die Anbindung von Funktionen an das Ereigniskonzept von VRML. Mit dem EAI kann man auf einen Knoten über seinen Namen zugreifen und Ereignisse an Felder dieses Knotens senden. Umgekehrt kann man Java-Methoden als Empfänger von Ereignissen registrieren, um auf solche zu reagieren. So kann eine VRML-Welt durch ein Java Programm kontrolliert und animiert werden. Das Programm kann auf Bewegungen und Mausklicks des Benutzers reagieren und Objekte manipulieren, hinzufügen oder entfernen.

[6] Next-Generation Extensible 3D
[7] Extensible Markup Language

2.7 Bilderkennung

Die klassische Aufgabe der Computergraphik liegt in der photorealistischen Abbildung (Projektion) von mathematisch definierten Modellen, wie sie in Abschnitt 2.3 beschrieben ist. Von einem Photo eines Gegenstandes auf dessen Form zu schliessen und ein numerisches Modell abzuleiten ist weitaus schwieriger; eine Zusammenfassung der Probleme und Lösungsansätze finden sich in [RN95]. Prinzipiell geht bei der Projektion des Objektes auf eine Ebene die räumliche Information verloren und kann nur durch eine Veränderung der Perspektive (entweder des Objektes oder des Betrachters) rekonstruiert werden. Eine Möglichkeit liegt in der Auswertung stereoskopischer Aufnahmen. Aber auch ohne räumliche Informationen können Menschen in einem Photo eindeutig von Helligkeitsverläufen und Musterverzerrungen her auf Formen schliessen. Oft reichen sogar Konturen von Objekten aus, um eine Vorstellung von der Struktur des abgebildeten Objektes zu entwickeln. Schwierig ist dabei die Interpretation der Kanten, die entweder aus der Silhouette von gekrümmten Oberflächen (z.B. Zylinder) oder aus Knicken der Oberfläche entstehen. Bei letzteren muß gedeutet werden, ob es sich um einen konvexen oder konkaven Knick handelt.

Diese menschliche Fähigkeit erfolgreich nachzubilden gelang erstmals Huffman, Clowes und Waltz [Wal75]. Sie beschränkten sich auf eine vereinfachte Welt, die nur aus trihedralen Körpern besteht und auf gekrümmte Oberflächen verzichtet. So eine Welt entsteht z.B. durch Kombination von Quadern.

In der ersten Stufe der Bildverarbeitung versucht man in der photographischen Abbildung Kanten und Knoten zu erkennen. Dabei kommen verschiedene Filter zum Einsatz, die auch schon in Abschnitt 2.3 gezeigt wurden. In der zweiten Stufe werden die gefundenen Kanten interpretiert. Diese Aufgabe wird durch die besondere Eigenschaft trihedraler Körper erleichtert, daß an jeder Ecke drei ebene Flächen orthogonal aufeinanderstoßen. So sind nur vier Eckentypen möglich, die in Abbildung 2.22 dargestellt sind. Die Analyse geschieht mithilfe einer von Huffman-Clowes aufgestellten Tabelle. Abbildung 2.23 zeigt auszugsweise alle Interpretationsmöglichkeiten der Kanten zum ersten Eckentyp, die durch verschiedene Perspektiven entstehen. Ein + bzw. - steht für konvexe bzw. konkave Kanten zwischen zwei sichtbaren Flächen. Ein Pfeil bezeichnet eine konvexe Kante, die ihre benachbarte Fläche verdeckt. Betrachtet man die Gesamtheit der Kanten, muß deren Interpretation konsistent sein, denn eine Kante kann nicht sowohl konvex als auch konkav sein. Mit einem Constraint-Solving-Algorithmus lässt sich eine solche konsistente Lösung ermitteln.

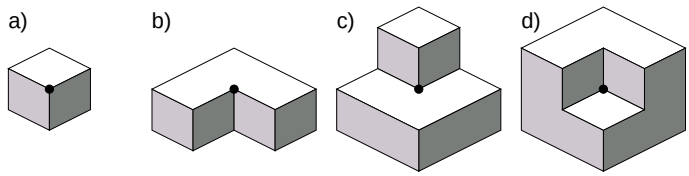


Abbildung 2.22: Die vier möglichen trihedralen Eckentypen

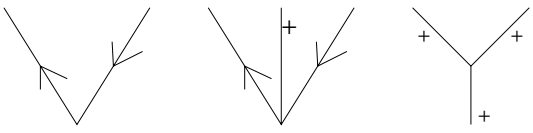


Abbildung 2.23: Interpretationsmöglichkeiten der Kanten von Ecke a)

2.8 Diskussion

Als offener Standard eignet sich VRML hervorragend für wissenschaftliches Arbeiten, da keine Urheberrechte bei der Verwendung beachtet werden müssen und umfangreiche Dokumentation und Tools frei verfügbar sind. Darüber hinaus ist VRML plattformunabhängig und überlässt dem Betrachter die Wahl seines Systems. Das Event-Prinzip ist in sich abgeschlossen und leicht analysierbar, gleichzeitig ist es mächtig genug, um eine komfortable Benutzeroberfläche zur Navigation zu schaffen. Es sind Java-Klassen des Open-Source VRML-Viewers *VRwave* [Ins00] verfügbar, um VRML-Knoten zu parsen und auf Java Objekte abzubilden. Somit ist es sehr leicht, den VRML-Szenegraphen im Rahmen einer Java-Applikation zu bearbeiten. Nachteilig an VRML erweist sich die Navigation mit der Maus, dem derzeit einzigen unterstützten Eingabegerät. Die Freiheitsgrade werden über Ikonen in der VRML- Konsole ausgewählt, der Benutzer hat dabei die Wahl zwischen Laufen, Fliegen, Rotieren und Zoomen. Bewegungen können in Kombination mit den Maustasten in verschiedenen Achsen erfolgen.

Im Vergleich zu *Java3D* [SRD00] Anwendungen steckt die gesamte Welt inklusive ihrer Dynamik im Szenegraphen. Dieser lässt sich ohne weiteres parsen, um die Hierarchie und Funktionalität der enthaltenen Objekte abzuleiten. Bei *Java3D* Programmcode ist dies nahezu unmöglich, da der Szenegraph erst zur Laufzeit aus Java-Objekten dynamisch aufgebaut und manipuliert wird. Sowohl VRML Viewer als auch *Java3D* setzen auf dem Graphikstandard *Open GL* [Shr99] auf. So lassen sich Grafikkarten mit Hardwarebeschleunigung über entsprechende *OpenGL* Treiber des Betriebssystems für Effekte wie Lichtquellen und Texturen nutzen. Alternativ stehen meist auch für bestimmte Prozessorentypen optimierte Softwarerenderer zur Verfügung.

Krüger und Kosslyn geben in Abschnitt 2.1 gute Beispiele dafür, wie graphische Abstraktion die kognitiven Ressourcen des Betrachters durch das Entfernen unnötiger Details und das Hervorheben von relevanten Merkmalen schonen kann. Die vorgeschlagenen Methoden lassen sich leicht von statischen Abbildungen auf 3D-Welten übertragen. Dann findet die Abstraktion nicht erst auf der Bildebene, sondern schon auf der Modell- und Generierungsebene statt. Der Effekt im erzeugten Bild ist jedoch der gleiche, somit wird auch dieselbe Wirkung beim Betrachter erzielt.

Im folgenden sollen die in Abschnitt 2.2 zitierten Vorschläge zur Gestaltung von Benutzerschnittstellen diskutiert werden. Für eine Navigationshilfe im Sinne dieser Diplomarbeit können einige Anforderungen von van Dam übernommen werden. Der Benutzer soll den Abstraktionsgrad der Objekte direkt erhöhen oder verringern können. Zusätzlich sind Sichten wünschenswert, die vom System vorbereitet werden. Im Fall von VRML-Welten liegt es nahe, auf funktionale Elemente der Szene zu fokussieren, da diese höchstwahrscheinlich zum Erreichen des Navigationszieles notwendig sind.

Es empfiehlt sich, bei der Gestaltung der Benutzerschnittstelle die von Shneiderman formulierten Anforderungen zu berücksichtigen. So weit wie möglich sollten Objekte direkt anklickbar sein. Um zwischen verschiedenen Operationen auszuwählen, sind 3D-Widgets in die Szene zu integrieren. Als nützlicher Nebeneffekt reduziert die graphische Abstraktion die Geometriedaten, so kann die Bildrate erhöht und die Navigation erleichtert werden.

Von den in Abschnitt 2.5 beschriebenen Abstraktionssystemen eignet sich *ARP* besonders gut für VRML-Welten, da es zur automatischen Abstraktion von ganzen Szenen entworfen wurde.

Die Grundlagen der Computergraphik bieten in Abschnitt 2.3 eine Terminologie an, um die Anforderunen an eine Navigationshilfe im folgenden zu beschreiben. Zur Repräsentation der Modelle verwenden sowohl VRML als auch *ARP* Polygonmeshes. Während in VRML die Modelle in einem Szenegraphen angeordnet sind, benötigt *ARP* eine Objekthierarchie. Beide Strukturen sind sich ähnlich und können ineinander überführt werden. Die allgemeine Definition der Abstraktionspipeline ermöglicht eine Klassifizierung der vorhandenen Abstraktionstechniken nach den Stufen, auf denen sie anwendbar sind.

Kapitel 3

Verwendung Graphischer Abstraktion in VRML-Welten

In diesem Kapitel sollen die Möglichkeiten zur graphischen Abstraktion von VRML-Welten untersucht werden. Dazu werden in Abschnitt 3.1 die im vorherigen Kapitel vorgestellten Verfahrensweisen auf ihre Eignung hin überprüft, wobei zwischen Modell-, Generierungs- und Bildebene unterschieden wird. Im Anschluss wird in Abschnitt 3.2 gezeigt, welche Informationen ein Hilfssystem zur Navigation der VRML-Welt entnehmen kann, um für den Benutzer sinnvolle Abstraktionen zu erzeugen. Dazu wird der Szenegraph geparst, um eine Objekthierarchie aus ihm abzuleiten. Ergänzend wird die Funktionalität der Geometrieknoten analysiert. Ist die Objekthierarchie der Szene zu flach, kann diese durch das Erkennen von Primitiven verfeinert werden. Dazu wird in Abschnitt 3.3 ein im Rahmen dieser Arbeit entwickeltes Verfahren vorgestellt.

3.1 Fokussieren durch graphische Abstraktion in VRML-Welten

In diesem Kapitel soll untersucht werden, welche Abstraktionstechniken im speziellen Fall von VRML-Welten anwendbar sind. Dabei werden die einzelnen Ebenen der Abstraktionspipeline (siehe Abschnitt 2.12) der Reihe nach betrachtet.

Den Anfang der Abstraktionspipeline bildet die Modellebene. In VRML werden die Objekte einer Szene durch Primitive wie Quader und Kugeln oder Polygonmengen modelliert und durch Geometriedaten im Szenegraphen repräsentiert. In der Abstraktion auf Modellebene gehen Details und spezielle Merkmale eines Gegenstandes verloren, während die Grundform und die Farbe und Oberfläche beibehalten werden. Dazu bieten sich bestehende Systeme wie PROXIMA (Primitivbasiert) oder ARP (Polygonbasiert) an. Diese Systeme sind jedoch nicht in der Lage, eine VRML-Datei einzulesen. Um eine Szene mit ARP zu abstrahieren wird ein spezieller Parser benötigt, der die Szenenhierarchie in einer für ARP verständlichen Listenform ausgibt und die Geometrien als Dateien im OFF-Format herausschreibt. Bei der Umwandlung der Geometrien in OFF-Dateien sind die VRML-typischen Fähigkeiten der Objektwiederverwendung und Transformation besonders zu behandeln. Referenzierte Geometrien sind erneut zu instanzieren und alle Transformationen wie Verschiebung, Skalierung und Rotation entlang des Szenegraphen so auf die Koordinaten anzuwenden, wie es ein VRML-Browser intern implementiert. Im Fall von ARP sind auch Primitive und komplexe Objekte wie Extrusionskörper durch Polygone zu approximieren.

Auf der Generierungsebene ist man bei VRML auf den Browser-eigenen Renderer angewiesen, der nicht austauschbar ist. Daher können keine Skizzenrenderer verwendet werden, die in Abschnitt 2.4.2 beschrieben werden. Es ist jedoch ohne weiteres möglich, zur

Laufzeit die Objektattribute Farbe und Transparenz zu manipulieren. So können Objekte ausgeblendet oder durchscheinend dargestellt werden. Ein Effekt ähnlich einer skizzenhaften Abstraktion kann durch Verwendung von einheitlichem, unschattiertem Grau für einzelne Objekte erreicht werden. Dadurch bleibt nur die Silhouette erkennbar.

In der letzten Ebene der Abstraktionspipeline ist kein Eingriff mehr möglich, da VRML rein dreidimensional orientiert ist und keine Operationen auf dem gerenderten Bild ermöglicht. So ist die Verwendung von Bitmapfiltern wie z.B. Unschärfe ausgeschlossen.

3.2 Informationsextraktion aus der VRML-Szenenbeschreibung

Das Ziel der Navigationshilfe besteht darin, dem Betrachter von VRML-Welten verschiedene Abstraktionsgrade der Objekte zur Verfügung zu stellen und nach funktionalen Eigenschaften in Sichten zu gliedern. Es erscheint nicht sinnvoll, einen eigenen Abstraktionsalgorithmus für die Navigationshilfe zu implementieren. Vielmehr sollte auf ein existierendes Abstraktionssystem wie z.B. ARP (siehe Abschnitt 2.5.2) zurückgegriffen werden. Dazu muß Information über Hierarchie und Funktionalität der Objekte aus der VRML-Welt extrahiert werden. Diese Aufgabe erfordert aufgrund der Komplexität und Anzahl der von VRML verwendeten Datenstrukturen einen spezialisierten Parser, der die vom Abstraktionssystem benötigten Informationen aus der Szene extrahiert.

Die Informationsextraktion erfolgt in zwei Schritten. Zuerst werden der Szenegraph geparst und die Objekthierarchie abgeleitet, danach werden die Geometrieknoten auf ihre Funktionalität hin analysiert.

Die VRML-Szenenbeschreibung besteht aus einem Baum von Knoten, in der sämtliche Information enthalten ist. Dabei repräsentiert nur ein Teil der Knoten echte Geometrie, während viele andere Knoten die Information über Kamera, Beleuchtung und Animation enthalten. Somit muß als erster Schritt nach dem Parsen der VRML-Datei eine Objekthierarchie aus dem Szenegraphen abgeleitet werden, die reine Geometrieknoten enthält und die Basis für die Abstraktionshierarchie bildet.

Im Fall der Navigation in VRML-Welten ist kein Wissen über die Ziele des Betrachters vorhanden. Auch ist über die Bedeutung der Objekte innerhalb der Welt für den Betrachter nichts bekannt. Um den Betrachter dennoch bei seiner Aufgabe zu unterstützen, kann man prinzipiell davon ausgehen, daß für ihn genau die Objekte relevant sind, die durch Interaktion Funktionen auslösen oder animiert werden.

Im Rahmen einer funktionalen Analyse betrachtet man den VRML-Kontext, in den die Geometrieknoten eingebettet sind und durch den sie ihre Funktionalität erhalten. Anhand der Struktur von VRML lassen sich funktionale Geometrieknoten in drei Klassen einteilen. Die erste Klasse enthält alle Objekte, die als Hyperlink dienen. Im VRML Szenegraph sind das die Söhne eines *Anchor* Knotens. Diese Objekte reagieren auf Mausklicks und bringen den Betrachter in andere VRML Welten oder veranlassen den Web-Browser dazu, HTML-Dokumente zu öffnen. Diese Hyperlink-Objekte sind das dreidimensionale Äquivalent zu HTML-Links.

Die zweite und dritte Klasse ergeben sich aus der Eventrouten-Technik von *VRML97*, die detailliert in dem Kapitel 2.6 beschrieben ist. Die zweite Klasse umfasst alle Objekte, die Ereignisse (Events) erzeugen und an andere Objekte senden. In der Regel sind dies Sensoren, die auf Mausklicks oder Bewegungen des Avatars reagieren. Daneben können auch Scripte oder Timer nach Ablauf einer bestimmten Zeitspanne Ereignisse erzeugen. Die dritte Klasse beinhaltet die Empfänger solcher Ereignisse. Beziehungen zwischen Sender und Empfänger werden über *Route* Knoten definiert. Üblicherweise ergeben sich Ereignisketten: Sensoren aktivieren Skripte, die über eine Programmlogik Timer steuern, die wiederum Animationen starten und stoppen.

Da sich alle funktionalen Objekte einer Szene den drei gerade definierten Klassen zu-

ordnen lassen, liegt es nahe, diese Klassen für den Betrachter der VRML-Welt mithilfe graphischer Abstraktion zu visualisieren. Dies lässt sich umsetzen, indem alle in der zu visualisierenden Klasse enthaltenen Objekte detailliert und die übrigen abstrakt dargestellt werden. Das Konzept lässt auch die Vereinigung von Klassen zu.

Im Folgenden soll unter einer *Sicht* auf die Welt eine Abbildung verstanden werden, die jedem Objekt entsprechend seiner Klassenzugehörigkeit einen Abstraktionsgrad zuweist. Der Benutzer des Hilfesystems kann so vorgegebene Sichten auf die drei funktionalen Klassen Hyperlink, Sensor und Animation abrufen. Zusätzlich sind alle Kombinationen von Eigenschaften zulässig, so daß sich insgesamt sechs mögliche funktionale Sichten ergeben. Sucht er in der virtuellen Welt nach weiterführenden Wegen oder einem Ausgang, kann er die Hyperlink- Sicht aktivieren. Alle Geometrien, die mit Hyperlinks assoziert sind, werden detailliert dargestellt, die übrige Szene wird abstrahiert. Diese Sicht kann auch nützlich sein, wenn der Betrachter sich für Informationen interessiert, die in Form von HTML-Dokumenten über symbolische Gegenstände abgerufen werden. Durch die Fokussierung treten solche Objekte deutlich hervor. Die Ereignisorientierte Sicht offenbart alle verborgenen Mechanismen. Ein einfaches Beispiel ist ein Lichtschalter, der eine Lampe steuert oder eine Türklinke, die angeklickt werden kann, um eine Tür zu öffnen. Der Benutzer kann nun sofort gezielt mit den interaktiven Elementen experimentieren, ohne sie erst suchen zu müssen oder zu übersehen.

3.3 Primitiverkennung

Abstraktionssysteme wie APEX oder ARP (Abschnitt 2.5) verschmelzen Teilobjekte einer Szene miteinander. Dabei halten sich die Abstraktionsalgorithmen strikt an die Objekthierarchie, es werden alle Söhne eines Knotens untereinander verschmolzen. Es wird davon ausgegangen, daß die Hierarchie dem Aufbau der Objekte durch ihre Teile entspricht, so daß die Verschmelzungsoperationen sinnvoll ausgeführt werden. Im Fall von VRML- Welten folgt die Objekthierarchie direkt aus der Struktur des Szenegraphen.

Falls der Autor die VRML-Datei von Hand selbst entworfen hat, ist von einem hohen Strukturierungsgrad auszugehen. Die VRML-Konzepte der Wiederverwendung von Objekten und der Gruppierung durch Transformationsknoten legen eine andere Vorgehensweise nahe.

Sind die Objekte jedoch mit CAD Modeling-Software wie 3D Studio [3dm00] konstruiert und anschliessend durch einen Exportfilter automatisch in VRML konvertiert worden, geht in der Regel viel von der Struktur verloren. Schlimmstenfalls wird die gesamte Szene durch wenige Geometrieknoten repräsentiert, die nur ungeordnete Polygone enthalten. Die Verschmelzung findet dann nur zwischen komplexen Objekten und nicht zwischen deren Bestandteilen statt und verfehlt ihre Wirkung. Genauso unsinnige Ergebnisse entstehen, wenn alle Einzelteile auf derselben Hierarchiestufe stehen und rein willkürlich miteinander verschmolzen werden. Das Abstraktionssystem ARP versucht natürlich, passende Kandidaten zur Verschmelzung anhand von Farbe und Nähe zu finden, um fehlerhafte Kombinationen zu vermeiden. Grundsätzlich geht ARP jedoch von einer sinnvollen Objektstruktur aus, die unlogische Verschmelzungen nicht zulässt.

Will man dennoch in beliebigen VRML-Welten sinnvolle Abstraktionen zu erzeugen, ist ein Verfahren notwendig, das Strukturen in deren 3D-Geometrie erkennt und durch eine Verfeinerung der Hierarchie ausdrückt. Einfache Strukturen sind zum Beispiel durch geometrische Primitive gegeben, die entsprechend den Gestaltgesetzen Nähe und Ähnlichkeit angeordnet sind.

Ein wichtiger Schritt besteht also darin, Primitive wie Quader und allgemeine Rotationskörper in einer Polygonmenge zu erkennen und zur weiteren Analyse symbolisch zu repräsentieren. Als Rotationskörper werden hier alle Körper bezeichnet, die durch Rotation eines Profils um eine Achse beschrieben werden können. Darunter fallen alle Objekte, die in der realen Welt auf einer Drehbank hergestellt werden könnten. Diese Definition

schliesst die Primitive Kegel(stumpf), Zylinder und Ellipsoid (mit Spezialfall Kugel) ein. In einer symbolischen Repräsentation werden Primitive, wie z.B. ein Zylinder, nicht approximativ durch Flächen und deren Koordinaten dargestellt, sondern durch ihren Mittelpunkt, Radius, ihre Länge und Orientierung beschrieben. Allgemeine Rotationskörper können durch ihr Profil als zweidimensionaler Polygonzug eindeutig repräsentiert werden. Dieser Wechsel von einer approximativen zur symbolischen Beschreibung der Objekte kann als Abstraktion auf der Datenstrukturebene aufgefasst werden.

Die symbolische Repräsentation bietet neben der Verfeinerung der Objektstruktur noch einige zusätzliche Vorteile. Es existieren spezielle Algorithmen zum Verschmelzen von Primitiven, z.B. PROXIMA [Krü95], die bessere Ergebnisse liefern können als polygonorientierte Verfahren wie der Algorithmus von Rossignac (siehe Abschnitt 2.4).

Durch die Profilrepräsentation können Rotationskörper wesentlich einfacher abstrahiert werden als Polygonmeshes. Profile lassen sich gut durch Splines darstellen und mit Hilfe von Wavelet-Kompression [SDS96] vereinfachen und glätten.

Die Darstellungsqualität kann gegenüber dem Polygonmodell verbessert werden. Die Primitive können anhand der symbolischen Repräsentation neu erzeugt und dabei durch feinere Polygone besser approximiert werden. Für eine perfekte Darstellung der Primitive mit völlig runden und glatten Oberflächen kann ein CSG Renderer wie *Povray* [PT99] eingesetzt werden.

Im Rahmen dieser Diplomarbeit wurden zwei eigenständige Algorithmen für Quader und Rotationskörper entworfen, deren Konzeption im folgenden vorgestellt wird. Die Implementation wird in Kapitel 5.3 beschrieben und Anwendungsbeispiele befinden sich in Kapitel 6.1.

3.3.1 Erkennen von Quadern

Bei dem hier vorgestellten Ansatz geht man statt von einem Bild von einem 3D-Polygonmodell als Eingabe aus, so daß Bildverarbeitungsschritte entfallen. Es gibt keine verdeckten Flächen und die Orientierung von Ecken (konvex oder konkav) ist klar definiert. Dennoch ist die Definition eines Quaders auf Polygonebene komplex und der Suchraum relativ groß. Daher ist es wichtig, den Algorithmus und die verwendeten Hilfsdatenstrukturen so zu gestalten, daß die Eigenschaften, nach denen gesucht wird, den Suchraum so weit wie möglich reduzieren, um exponentielle Laufzeiten zu vermeiden. Die spezifische Eigenschaft ist der Zusammenhang von Knoten (in Meshes 4 Nachbarn), danach kommen die Orientierung der Ecken (je Koordinatensystem 8 Klassen) und die Parallelität (je Koordinatensystem 3 Klassen).

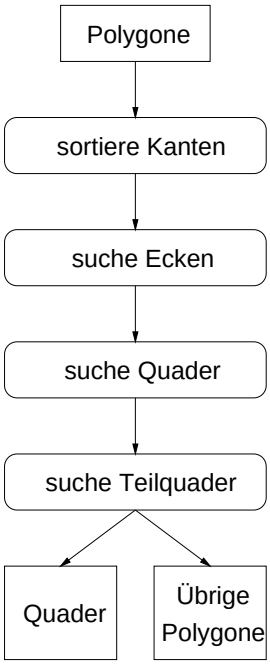


Abbildung 3.1: Suche nach Quadern

Die Quaderdefinition wurde daher so formuliert, daß sie auf dem Datentyp der Ecke basiert. Eine Ecke wird von einem Knoten gebildet, von dem drei Kanten im Rechten Winkel zueinander ausgehen. Ein Quader besteht aus 8 Ecken, von denen jeweils zwei über eine gemeinsame Kante verfügen und spiegelsymmetrisch zusammen passen. Um die Spiegelsymmetrie leicht prüfen zu können, werden die Ecken nach ihrem Koordinatensystem klassifiziert. Jede Ecke, die nicht achsenparallel zu einer bisherigen ist, wird einem eigenen neuen Koordinatensystem zugeteilt. Alle Ecken innerhalb eines gemeinsamen Koordinatensystems können nach Oktanten klassifiziert werden, die den acht

möglichen Kombinationen entsprechen, die sich aus drei Kantenrichtungen ergeben.

Der Ablauf der Suche nach Quadern ist als Datenflußdiagramm in Abbildung 3.1 dargestellt. Die Eingabe bildet eine Menge von beliebigen Polygonen. Die Ausgabe teilt sich auf in eine symbolische Beschreibung der gefundenen Quader und den nicht zuordbaren Restpolygonen. In einem ersten vorbereitenden Schritt wird ein Index erstellt, über den später zu jedem Knoten alle anliegenden Kanten gefunden werden können. Das Erstellen des Index erfolgt am schnellsten durch Sortieren der Kanten mit Quicksort. Im zweiten Schritt werden alle Ecken gesucht. Dazu werden je drei ausgehende Kanten eines Knotens auf Rechtwinkligkeit zueinander untersucht und gegebenenfalls als Ecke gespeichert. Die Datenbank der Ecken bildet zusammen mit dem Kantenindex die Grundlage für die Suche nach Quadern.

Die Suche ist in Abbildung 3.2 dargestellt und beginnt bei einer beliebigen Ecke A. Nun wird an den Enden der drei Kanten nach passenden Ecken gesucht. Dabei ist auf Spiegelsymmetrie zu achten. Falls drei solche Ecken B,C,D gefunden wurden, wird nach drei weiteren Ecken E,F,G gesucht. Bei diesen Ecken sind nun zwei Kanten zu prüfen, z.B. muß E sowohl zu B als auch zu C passen. Anschliessend wird die A diagonal gegenüberliegende Ecke H gesucht, die zu E,F und G passen muß. Konnten acht Ecken gefunden werden, wird eine symbolische Repräsentation des Quaders ausgegeben. Diese enthält Größe, Position und Orientierung des Quaders und wird aus den Koordinaten der Eckpunkte berechnet. Die entsprechenden Polygone werden nun aus der Eingabedatei gefiltert.

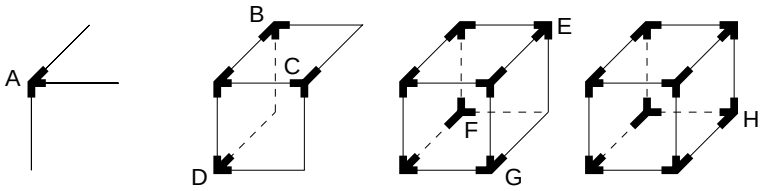


Abbildung 3.2: Auf der Suche nach einem Quader

In weiteren Schritten wird nun nach Teilquadern gesucht, wie sie in Abbildung 3.3 dargestellt sind. Der erste Typ von Teilquader wird als Schachtel bezeichnet und besteht aus 6 Ecken und 4 Seitenflächen. Solche Teilquader ergeben sich, wenn verdeckte Flächen innerhalb von Modellen aus Effizienzgründen entfernt wurden. Ein zweiter Typ mit 4 Ecken und 3 Flächen wird als sogenannte Brücke klassifiziert.

Probleme bereiten Quader mit segmentierten Kanten. Dies ist der Fall wenn eine Seitenfläche aus mehreren Teilen modelliert wurde, so daß zwei Ecken nicht direkt durch eine Kante verbunden sind. Daher wurde der Algorithmus verfeinert. Wird zu Ecke A eine passende Ecke B benötigt, wird die von A ausgehende Kante betrachtet und an ihrem Endknoten nach einer Ecke B gesucht. Findet sich dort keine Ecke, aber dafür eine parallele Kante die eine Verlängerung der ursprünglichen Kante darstellt, wird die Suche an dem Endknoten der Verlängerung fortgesetzt. Es wird solange weitergesucht, bis eine Ecke gefunden wird oder keine Verlängerung mehr existiert.

Die Laufzeit des Verfahrens hängt von einigen Eigenschaften des Eingabemodells ab, die zuerst definiert werden sollen: Sei k die Anzahl der Knoten, sei n die Anzahl der Kanten und n_i die Anzahl der Kanten an Knoten i, sei e die Anzahl der Ecken im Modell und m die mittlere Anzahl der Ecken pro Knoten. Der erste Schritt, das Sortieren der Kanten, benötigt mit Quicksort [Sed90] im Durchschnitt $O(nlogn)$ und im Worstcase $O(n^2)$ Operationen. Die Suche nach Ecken an einem Knoten i erfordert den Test von $\binom{n_i}{3}$ Kombinationen von Kanten. Im besten Fall besteht die Eingabe nur aus Quadern, dann ist $n_i = 3$ und $\binom{3}{3} = 1$. Im Fall eines triangulisieren Meshes sind $\binom{6}{3} = 20$ Kombinationen zu prüfen. Im Idealfall sind pro Knoten also konstant viele Kombinationen zu prüfen, dazu werden

$O(k)$ Operationen benötigt. Im ungünstigsten Fall sind alle Knoten miteinander verbunden, also $n_i = k - 1$ und $O(k(\binom{k-1}{3}))$ Kombinationen zu prüfen. Im dritten und vierten Schritt werden Quader und Teilquader gesucht. Beide Schritte sind sich sehr ähnlich und können gleich behandelt werden. Es wird eine beliebige Ecke gewählt und nach benachbarten Ecken gesucht. Das erfordert für jede der drei Kanten durchschnittlich das Überprüfen von m Ecken. In einem Polygonmesh gilt typischerweise $m < 1$. In einem Modell, das nur aus Quadern besteht, gilt $m = 1$. Da e aufgrund der komplexen Anforderungen an eine Ecke keineswegs überproportional zu k wachsen kann, ist $m = \frac{e}{k}$ als Konstante anzusehen. Es ergibt sich daher eine Laufzeit von $O(e)$ bzw. $O(k)$. Zusammenfassend ergibt sich eine günstige Laufzeit für eine Eingabe bestehend aus Polygonmeshes und Quadern. Dann gilt $3 <= n_i <= 6$, $m <= 1$, $e <= k$ und $n = 2k$. Daraus folgt als Laufzeit $O(klogk) + O(k) + O(k)$, also kurz $O(klogk)$. Im Worstcase gilt $n = \binom{k}{2}$, zusammen mit $O(n^2)$ ergibt sich $O(k^4) + O(k(\binom{k}{3})) + O(k)$. Zusammengefasst bleibt der stärkste Exponent $O(k^4)$ stehen.

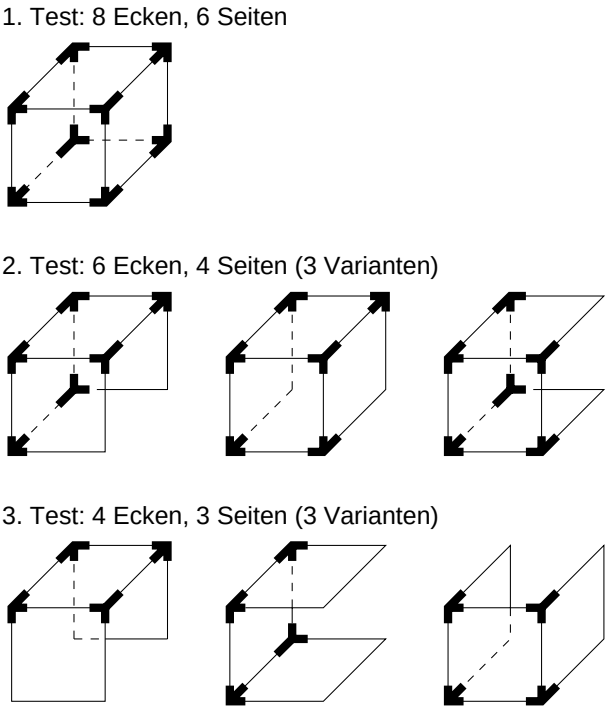


Abbildung 3.3: Es werden sieben mögliche Quader und Teilquader erkannt

3.3.2 Erkennen von Rotationskörpern

Neben Quadern bestehen Modelle häufig aus Zylindern, Kegeln und Kugeln. Im allgemeinen werden diese Primitive durch Polygone approximativ modelliert, indem ein Profil als Polygonzug in n Schritten um eine Achse gedreht wird und die Punkte benachbarter Schritte durch Polygonflächen verbunden werden. Ausser den genannten Primitiven lassen sich auf diese Weise auch komplexere Objekte wie Vasen oder Gläser leicht modellieren. Allen auf diese Weise erzeugten Modellen sind Ringe gemeinsam, die durch das Rotieren und Verbinden des Profils entstehen. Genauer gesagt sind es n-Ecke mit drei Eigenschaften. Alle Winkel sind gleich groß und kleiner als 180 Grad, alle Kanten sind gleich lang und alle Winkel liegen innerhalb einer Ebene. Diese Eigenschaften lassen sich leicht prüfen und sind so speziell, daß die Suche nach Ringen effizient durchführbar ist. In der Regel findet sich entweder ein Zyklus von n Winkeln, oder die Suche terminiert bereits nach dem ersten Winkel, da Kreissegmente relativ selten in Modellen vorkommen.

Ein Datenflußdiagramm der Suche nach Rotationskörpern ist in Abbildung 3.4 dargestellt. Für den Suchalgorithmus ist ein schneller Zugriff auf Kanten über ihre Knoten erforderlich. Da die Datenstruktur des Polygonmodells Polygone als Aufzählung von Knoten definiert, muß ein neuer Index angelegt werden. Der schnellste Weg besteht in einer Sortierung der Kanten nach Knoten. Als Vorbereitung zur Suche nach Winkeln wird in einem ersten Schritt zu allen Kantenpaaren eines Knotens ihr Winkel errechnet und gespeichert, dabei kommen natürlich nur Winkel mit Kanten gleicher Länge in Betracht. Jeder Winkel wird um seinen Normalenvektor ergänzt und über seinen Knoten indexiert.

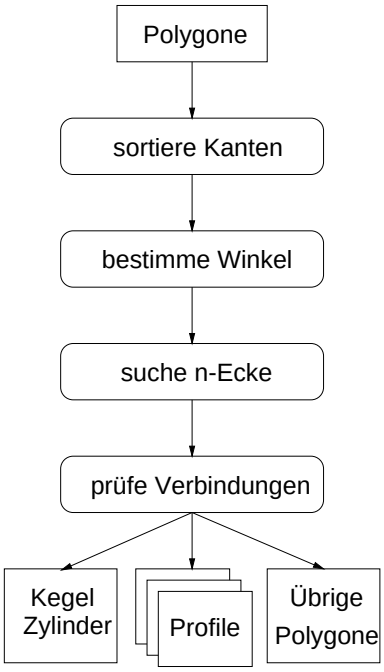


Abbildung 3.4: Suche nach Rotationskörpern

Auf diesen Hilfsdatenstrukturen basiert die Suche nach Ringen, die im nächsten Absatz im Detail beschrieben wird. Anschliessend werden alle gefundenen Ringe untereinander auf Zusammenhang und eine gemeinsame Achse geprüft. Miteinander verbundene Ringe werden zu Objekten zusammengefasst und je nach Anzahl und Radius in verschiedene Primitive klassifiziert. Den einfachsten Fall bilden Scheiben. Sind alle Eckpunkte eines Ringes mit einem einzigen Punkt verbunden, handelt es sich um einen Kegel. Sind zwei Ringe miteinander verbunden und haben den gleichen Radius, wird das Objekt als Zylinder klassifiziert, ansonsten als Kegelstumpf. Diese Primitive werden symbolisch ausgegeben und durch ihre Höhe, Radien und Achse beschrieben. Sind mehr als zwei Ringe entlang einer gemeinsamen Achse miteinander verknüpft, wird ein Profil aus Höhe und Radius der Ringe erstellt und gespeichert.

Die Suche nach einem Ring, also einem gleichseitigen und gleichwinkligen ebenen n-Eck, ist in Abbildung 3.5 dargestellt. Dort sind 4 Knoten I,J,K und L und die an J und K liegenden Winkel alpha und beta abgebildet. Der Algorithmus beginnt bei einem beliebigen Winkel alpha (IJK) am Knoten J. Nun wird nach einem passenden Winkel beta (JKL) am Knoten K gesucht, der eine gemeinsame Kante, den gleichen Winkel und einen parallelen Normalenvektor zu alpha hat. Ist ein solcher Winkel vorhanden, wird die Suche solange fortgesetzt, bis der Startknoten J erreicht wird und der Ring dann gespeichert. Ansonsten werden alle besuchten Winkel gelöscht, da sie nicht Teil eines anderen Ringes sein können. Auch eine Suche in Richtung des Knoten J hätte keinen Ring gefunden, da die Kanten des

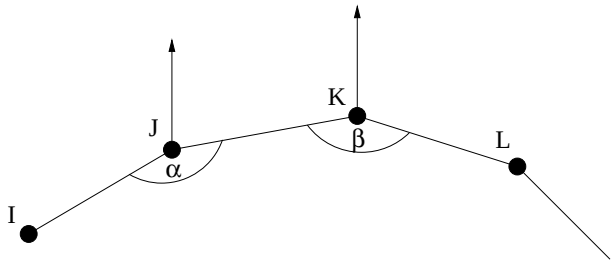


Abbildung 3.5: Die Suche nach Ringen orientiert sich an gleichen Winkeln und parallelen Normalenvektoren

Modells ungerichtet sind.

Die Laufzeitanalyse der Rotationskörpererkennung soll für Polygonmeshes durchgeführt werden, da sie die typische Eingabe für den Algorithmus darstellen und einige Vereinfachungen der Parameter ermöglichen. Sei k die Anzahl der Knoten, n die Anzahl der Kanten, w die Anzahl der Winkel im Modell und m die mittlere Anzahl der Winkel an einem Knoten. In einem Polygonmesh entspricht m der Anzahl der 2-elementigen ungeordneten Teilmengen der Kanten pro Knoten. Bei einem Polygonmesh aus Vierecken liegen an jedem Knoten 4 Kanten an, daraus ergeben sich $m = \binom{4}{2} = 6$ Winkel und es gilt $n = 2k$. Bei einem triangulisierten Mesh ergeben sich aus 6 Kanten $m = \binom{6}{2} = 15$ Winkel und es gilt $n = 3k$. Beschränkt man die Suche auf mindestens 5-Ecke, kann man durch Festlegen einer Untergrenze von $180 - \frac{360}{5} = 108$ Grad die meisten Winkel herausfiltern. Für das Sortieren der Kanten ergibt sich unter Verwendung von Quicksort eine Laufzeit von $O(nlogn)$ und bei einem Polygonmesh $O(klogk)$. Die worstcase Laufzeit liegt bei $O(k^2)$. Das Bestimmen der insgesamt $w = km$ Winkel benötigt für jeden Knoten $O(m(m-1))$ Operationen, und da m konstant ist, $O(k)$. Die Suche nach einem n-Eck beginnt bei einem beliebigen der insgesamt w Winkel und vergleicht ihn dann mit den benachbarten m Winkeln. Die Suche terminiert, sobald kein passender Winkel mehr gefunden wird oder der Startpunkt wieder erreicht ist. In jedem Fall werden alle besuchten Winkel gelöscht. So beträgt die Laufzeit $O(wm)$, und da $w = km$ gilt und m konstant ist, hängt die Laufzeit effektiv nur von k ab, also $O(k)$. Insgesamt beträgt die Laufzeit $O(klogk) + O(k) + O(k)$, zusammengefasst $O(klogk)$. Für das worstcase Szenario gilt $O(k^2)$. Der Speicherbedarf ergibt sich aus der Anzahl der Kanten n für den Kantenindex und der Anzahl der Winkel w. Beide Faktoren hängen im Fall von Polygonmeshes von der Anzahl der Knoten k ab, so daß für den Speicher $RAM = O(k)$ gilt.

3.4 Interaktionskonzepte

Das Ziel der vorliegenden Arbeit ist eine Navigationshilfe, die dem Benutzer das Explorieren von VRML-Welten erleichtern soll. Die Technik der graphischen Abstraktion soll zum einen die Aufmerksamkeit des Betrachters auf Wichtiges konzentrieren, zum anderen die Rechnerressourcen schonen, um höhere Bildraten zu ermöglichen. Da das Navigationshilfssytem die individuellen Ziele des Betrachters in der Regel nicht kennt, muß der Benutzer selbst interaktiv die Abstraktionsgrade einzelner Objekte der Szene variieren können.

Welche Konzepte zum Entwurf einer Benutzerschnittstelle gibt es, und welche Auswahlmöglichkeiten lassen sich damit für den Benutzer in einer VRML-Umgebung realisieren ?

3.4.1 Direkte Manipulation

Unter direkter Manipulation versteht man das direkte Auswählen eines Objekts innerhalb der Szene mittels einer Zeigegeste, in der Regel durch die Maus. Dadurch wird eine fest mit dem Objekt assoziierte Aktion ausgelöst. Die Vorteile der direkten Manipulation wurden bereits in Kapitel 2.2 dargestellt. In VRML wird dieses Konzept durch Sensoren unterstützt, die auf Berühren und auf Anklicken von Objekten mit der Maus reagieren und je nach Objekt verschiedene Scripte starten können. Es wäre so möglich, jedes Objekt durch Anklicken zu abstrahieren oder mit seinen Nachbarn zu verschmelzen. Da VRML als Eingabe nur eine Maustaste unterstützt, kann jedoch keine weitere Operation neben der Abstraktion direkt ausgeführt werden. So gäbe es keine Möglichkeit mehr, eine Abstraktion rückgängig zu machen. In klassischen graphischen Benutzeroberflächen werden in solchen Fällen Ikonen und Menüs verwendet, um den Benutzer aus allen möglichen Funktionen wählen zu lassen. Es liegt nahe, in VRML ebenfalls Menüs zu verwenden.

3.4.2 Menü-Manipulation

Java-Applets bieten die typischen Elemente einer Benutzerschnittstelle wie Buttons, Schalter und Textfelder und können zur Steuerung des VRML-Viewers eingesetzt werden. Ihre Verwendung fordert jedoch eine Teilung des Bildchirms in zwei Zonen: Zum einen die 3D-Ansicht der VRML-Welt, zum anderen die 2D-Oberfläche des Applets. So kann nicht der gesamte Bildschirm für VRML genutzt werden. In einer echten Virtual-Reality-Umgebung, in der der Benutzer eine 3D-Brille und einen Handschuh zur Navigation verwendet, sind 2D-Menüs ohnehin unangemessen. Daher sollen die Menüs als 3D- Geometrie in die Szene integriert werden. Weil die Darstellung von Text durch Polygone recht teuer im Sinne von Ressourcen ist, sollen vorwiegend Symbole/Ikonen verwendet werden.

Über Menüs lassen sich verschiedene Möglichkeiten zur Auswahl von Abstraktionsgraden realisieren. Grundsätzlich soll der Abstraktionsgrad jedes einzelnen Objektes direkt manipuliert werden können. Dazu sind Ikonen zum Entfernen und Hinzufügen von Details erforderlich. Das Entfernen von Details kann auch eine Verschmelzung mit benachbarten Objekten bedeuten. Entweder man bindet an jedes Objekt eigene Ikonen zur Manipulation, oder man verwendet nur zwei globale Ikonen. In diesem Fall muß ein Auswahlmechanismus realisiert werden, um festzulegen, welches Objekt durch die Ikonen manipuliert werden soll.

Neben der individuellen Abstraktion sollen funktionale Sichten der Szene abgerufen werden können. Diese sind in Abschnitt 3.2 definiert und weisen allen Objekten in der Szene einen Abstraktionsgrad zu, um ihre Funktion zu visualisieren. Zur Auswahl dieser Sichten eignen sich Ikonen, die in einem Menü zusammengefasst werden können.

3.4.3 Gruppenbildung

Im vorigen Abschnitt sind zwei Möglichkeiten zur Abstraktion über Menüs vorgestellt worden. Darüber hinaus sind noch zwei weitere Methoden denkbar: Erstens das Zusammenfassen von Objekten, um diese gezielt zu verschmelzen und gemeinsam zu abstrahieren. Zweitens das Herausnehmen eines einzelnen Objekts aus einer solchen Gruppe, um es so indirekt hervorzuheben und die Aufmerksamkeit darauf zu fokussieren. Für diese beiden Methoden ist eine Mehrfachauswahl von Objekten erforderlich. Diese kann durch ein einfaches Anklicken der einzelnen Teilobjekte erfolgen. Da jedoch im Gegensatz zum Konzept der direkten Manipulation keine unmittelbare Aktion ausgelöst wird, ist ein visuelles Feedback der Auswahl notwendig. Die selektierten Objekte können durch veränderte Darstellungsparameter, wie z.B. Helligkeit oder Farbe, markiert werden. In VRML kann dies einfach und wirkungsvoll durch Einfügen von Spotlights realisiert werden. Nachdem alle Elemente ausgewählt sind, kann eine Gruppenoperation aus einem Menü ausgewählt und angewendet werden.

3.5 Anforderungen an ein System zur intelligenten Navigation

An dieser Stelle werden kurz die Anforderungen an das zu entwerfende Hilfssystem zusammenfassend formuliert, soweit sie sich aus den vorherigen Abschnitten dieses Kapitels ableiten lassen.

Das System soll die Navigation in VRML Welten erleichtern, indem graphische Abstraktion eingesetzt wird, um die Ressourcen des Betrachters und des Computers zu schonen. Der Benutzer muß dazu interaktiv Abstraktionsgrade einzelner Objekte variieren können, um so eine persönliche Sicht auf die 3D-Welt zu erhalten, in der wichtige Details dargestellt und unwichtige unterdrückt werden. Um die Aufmerksamkeit auf funktionale Elemente zu fokussieren, sind abrufbare vorkonfigurierte Sichten sinnvoll. Dazu ist eine Benutzerschnittstelle zu entwerfen, die sich an den in Abschnitt 3.4 beschriebenen Konzepten orientiert.

Der eleganteste und effizienteste Weg der Implementierung führt über die Verwendung von möglichst vielen fertigen Komponenten. Die Basis bildet ein VRML-Plugin oder eigenständiger Viewer, der mittels der EAI- Technik (siehe Abschnitt 2.6.3) von einer Java Applikation gesteuert werden kann. Von dem EAI wird ein Web-Browser mit einer Virtual-Machine für die Ausführung von Java-Bytecode vorausgesetzt. Zur Abstraktion auf Modellebene eignet sich der robuste Filter-Algorithmus von Rossignac [RB92]. Um eine Neuimplementierung zu vermeiden , empfiehlt sich die Verwendung eines fertigen Systems. Dazu wurde das Abstraktionssystem ARP ausgewählt, denn es ermöglicht das automatische Abstrahieren und Verschmelzen ganzer Szenen. Dabei läßt es sich durch Spezifikation von Darstellungsparametern steuern. Die Abstraktion auf Generierungsebene kann innerhalb von VRML durch Veränderung der Darstellungsparameter mittels Javascript-Programmierung erfolgen.

Kapitel 4

Entwurf eines Systems zur interaktiven Auswahl von Abstraktionsgraden

4.1 Systembeschreibung

Dieses Kapitel beschreibt das VAI[1] System. Es unterstützt den Betrachter von virtuellen Welten bei seiner individuellen Navigationsaufgabe. Durch die Verwendung der Technik der graphischen Abstraktion werden seine kognitiven Ressourcen entlastet, indem die Objektanzahl und -details reduziert werden. Über eine intelligente Auswahl von Abstraktionsgraden kann die Aufmerksamkeit des Benutzers gezielt auf die Objekte fokussiert werden, die für sein Navigationsziel relevant sind (siehe Abschnitt 2.1.1). Zusätzlich werden durch die Abstraktion der Umfang der Modellgeometrie verringert und die zur Visualisierung benötigten Rechnerressourcen geschont. Dadurch kann die Bildgenerierungsgeschwindigkeit des Displays deutlich erhöht und die Interaktion verbessert werden (siehe Abschnitt 2.2.2).

Das VAI-System fügt der VRML-Welt eine Steuerungskonsole als Benutzerschnittstelle hinzu, die in Kapitel 4.3 detailliert beschrieben wird. Mit Hilfe dieser Konsole kann der Benutzer die Abstraktionsgrade aller Objekte verändern, wie es in Abschnitt 2.2.1 gefordert wird. Eine stärkere Abstraktion beinhaltet das Verschmelzen des Objekts mit seinen unmittelbaren Nachbarn. Die mathematischen Berechnungen der Abstraktionen und Verschmelzungen auf der Modellebene erfolgen extern durch das ARP-System im Client-Server-Betrieb. Auf Generierungsebene können Objektattribute wie Farbe und Transparenz VRML-intern unmittelbar zur Laufzeit manipuliert werden.

Es wurden zwei Varianten von VAI entworfen und implementiert, die sich im Zeitpunkt der Abstraktionsanforderung an ARP unterscheiden. Dadurch bedingt unterscheidet sich auch der Navigationsablauf, der in den Abschnitten 4.2.1.2 und 4.2.2.2 beschrieben wird. Im ersten Entwurf finden die Berechnungen der Abstraktion erst zur Laufzeit statt, ausgelöst durch individuelle Anfragen des Benutzers. In der zweiten Version werden alle Abstraktionen vor der Navigation erzeugt und der VRML-Welt hinzugefügt. Diese können dann über die VAI-Konsole abgerufen und aktiviert werden.

[1] **V**RML **A**bstraction **I**nterface

4.2 Betriebsmodi

Bei dem Entwurf des Systems stellt sich die grundlegende Frage, zu welchem Zeitpunkt die Abstraktionen von ARP berechnet werden sollen. Dabei kommen zwei Varianten in Betracht, deren Antwortzeiten auf Benutzeranfragen in Abbildung 4.1 durch Pfeile dargestellt sind. Im Ablauf des gezeigten Beispiels wählt der Benutzer eine VRML-Welt und navigiert darin. Dabei fordert er nacheinander über die Konsole die Abstraktionen A1 und A2 an. Graue Pfeile symbolisieren in der Abbildung die Berechnung von Geometrie.

Die erste Variante besteht darin, Abstraktionen genau nach den Wünschen des Benutzers während der Navigation dynamisch zu erzeugen. So hat er unmittelbar nach dem Laden und Analysieren der Welt *(a)* die Möglichkeit, beliebige Abstraktionsgrade und Verschmelzungen frei zu wählen. Der Nachteil dieser Lösung sind unvermeidbare Wartezeiten, die durch die Berechnung und den Netzwerktransport entstehen *(b,c)*. Die zweite Variante vermeidet diese Problematik, indem sinnvolle Abstraktionsgrade und Verschmelzungen vorberechnet und zu Beginn der Navigation geladen werden *(d)*. Während der Navigation wird nur zwischen bereits existierenden Abstraktionsgraden umgeschaltet *(e,f)*. Beide Entwurfsmöglichkeiten wurden getestet. Zuerst wurde ein Prototyp des Online-Konzepts implementiert, wobei sich jedoch bereits in einer frühen Entwicklungsphase die Wartezeiten bei komplexen Modellen als störend erwiesen. Daher wurde eine Offline-Version entworfen und vollständig implementiert. Im folgenden werden die Systemarchitektur und der Ablauf beider Konzepte miteinander verglichen. Anschliessend werden die Benutzerschnittstellen beider Systeme vorgestellt.

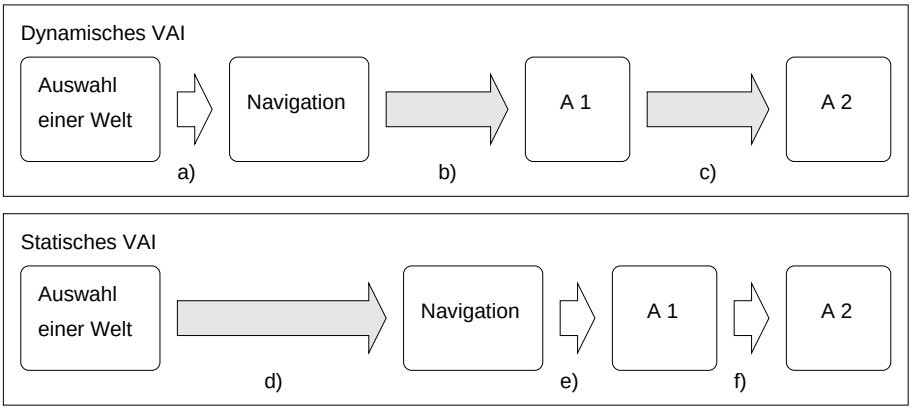


Abbildung 4.1: Antwortzeiten der dynamischen und statischen VAI Variante

4.2.1 Dynamisches VAI

4.2.1.1 Architektur

Der Umfang und die Komponenten des Online-Systems sind in der Abbildung 4.2 zu sehen. Das Abstraktionssystem ARP berechnet auf Anfrage von VAI Abstraktionen mit Hilfe des Abstraktionsservers BUNDY (siehe Abschnitt 2.5.2). Beide Programme können in einer verteilten Umgebung auf eigenen Servern laufen und kommunizieren über eine Socketverbindung, der Austausch der Geometriedateien erfolgt über Dateien.

VRML-Welten werden in der Regel aus dem Internet geladen und mit einem geeigneten Plugin in einem Web-Browser betrachtet. In der Entwicklungsumgebung des Prototyps handelt es sich dabei um das Cosmoplayer-Plugin [Sil98] der Firma *Silicon Graphics* und den *Netscape Communicator* [Net00]. Da VRML jedoch ein systemunabhängiger Standard ist, können auch andere Produkte verwendet werden. Das eigentliche VRML Abstraction Interface wurde als Java-Applet implementiert und läuft innerhalb der Java Virtual Machine des Browsers. Die VRML-Welt und das Applet sind gemeinsam in einem HTML Dokument eingebunden und sichtbar. Das Applet hat nun die Möglichkeit, über das sogenannte External Authoring Interface EAI [Aut99] (siehe Abschnitt 2.6.3) innerhalb des Browsers eine Verbindung mit dem Cosmoplayer Plugin herzustellen. Diese Schnittstelle bietet dem Applet die Möglichkeit, auf Ereignisse wie Mausklicks innerhalb der VRML-Welt zu reagieren und den Szenegraphen direkt zu manipulieren. So ist es möglich, die Steuerung des Applets nicht nur über Java-Menüs sondern auch durch eine Konsole innerhalb der VRML-Welt zu realisieren. Die Möglichkeiten der Manipulation sind so umfassend, daß Teile der VRML-Welt komplett durch Abstraktionen ersetzt werden können. Zur Berechnung von Abstraktionen kann das Applet über eine Socket-Verbindung Kontakt zum Abstraktionssystem ARP herstellen und beliebige Abstraktionen und Verschmelzungen anfordern. Im Prototyp wurde auf eine Socket-Verbindung zu ARP zunächst verzichtet, stattdessen werden mögliche Abstraktionen mit Bundy vorbereitet und auf dem Server hinterlegt. Das Applet kann dann die entsprechenden Dateien herunterladen. Während der Implementierung hat sich gezeigt, daß die EAI Schnittstelle ein proprietärer Standard und mit technischen Mängeln behaftet ist. Diese äusserten sich in Speicherkonflikten beim gemeinsamen Zugriff auf VRML-Referenzen durch das Plugin und die Java Virtual Machine des Applets.

4.2.1.2 Ablauf

Der typische Ablauf der Navigation mit der Online-Implementierung des VAI Systems ist in Abbildung 4.3 dargestellt.

Als erstes wird über den HTML-Browser, in der Prototypen-Umgebung der Netscape Communicator, die VAI Navigationshilfe geöffnet. Diese VRML- Welt wird im Cosmoplayer dargestellt und enthält eine Steuerungskonsole, mit deren Hilfe der Benutzer über ein Icon eine beliebige VRML-Welt auswählen und aus dem Internet herunterladen kann.

Im zweiten Schritt wird die VRML-Datei dem Analyse-Modul übergeben. Das Modul parst die Datei und den darin enthaltenen Szenegraphen. Dabei wird programmintern eine Baumstruktur aufgebaut, die die Szene repräsentiert. Nun können die Geometrieknoten des Baums entsprechend ihrer Hierarchie und Funktionalität klassifiziert werden. Die Hierarchie ergibt sich aus Gruppierungs- und Transformationsknoten im Szenegraphen. Die Funktionalität geht aus dem Kontext hervor, wobei Knoten in gleicher oder höherer Ebene beachtet werden.

Dann folgt in Schritt 3 die Navigation. Der VRML-Browser zeigt die VRML-Welt an, wobei die Steuerkonsole des Hilfssystems stets im Sichtfeld der Kamera bleibt. Der Benutzer kann sich nun frei durch die Welt bewegen und nach seinen Wünschen Abstraktionsgrade mit Hilfe der Konsole verändern. Das Hilfssystem stellt dazu eine Abstraktionsanforderung an den ARP Server. Das ARP-System berechnet nun eine neue abstrakte Geometrie anhand der Vorgaben des Benutzers und liefert sie zurück. Dabei kann die Netzwerkübertragung bei umfangreichen Geometrien in der Grössenordnung von mehrerern hundert Ki-

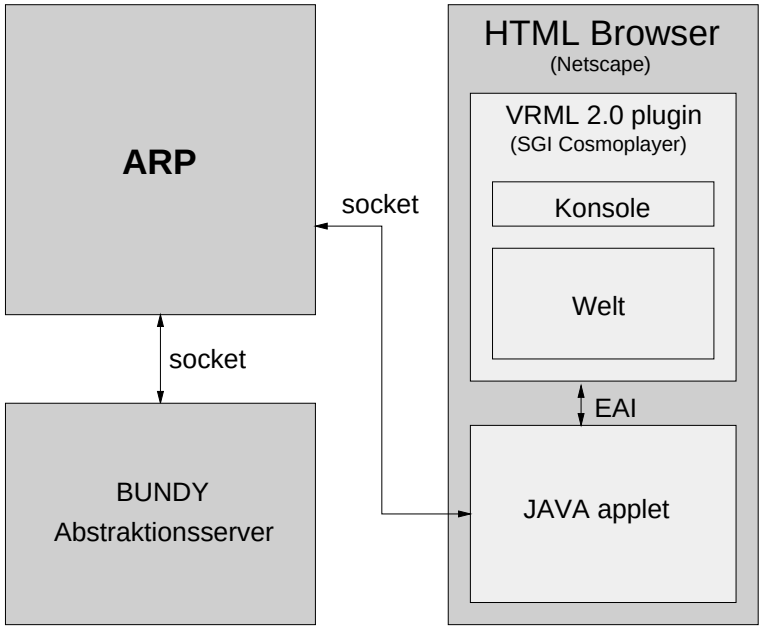


Abbildung 4.2: Architektur der dynamischen VAI Variante

lobytes zu deutlichen Verzögerungen führen. Das Hilfssystem tauscht nun die Geometrien innerhalb der VRML-Welt aus, so daß die Abstraktion für den Betrachter direkt sichtbar wird. Der Benutzer kann nun in der manipulierten Welt navigieren und weitere Änderungen der Abstraktionsgrade vornehmen.

Es ist zu überlegen, wie der initiale Zustand der Welt sein sollte. Falls die Welt dem Benutzer noch unbekannt ist, sollte sie zunächst ohne Abstraktion präsentiert werden. Denn eine starke Abstraktion kann dazu führen, daß die dargestellten Objekte vom Betrachter nicht mehr erkannt werden. Betritt der Benutzer jedoch eine Welt, die er bereits kennt, ist eine anfängliche Darstellung auf hohem Abstraktionsniveau sinnvoll. Er kann gezielt darin navigieren und erst bei Bedarf auf eine detaillierte Ansicht umschalten. Im Idealfall ist das Navigationsziel des Benutzers bekannt oder zumindest wahrscheinlich. Dann kann mittels Auswahl geeigneter Abstraktionsgrade direkt auf relevante Objekte fokussiert werden.

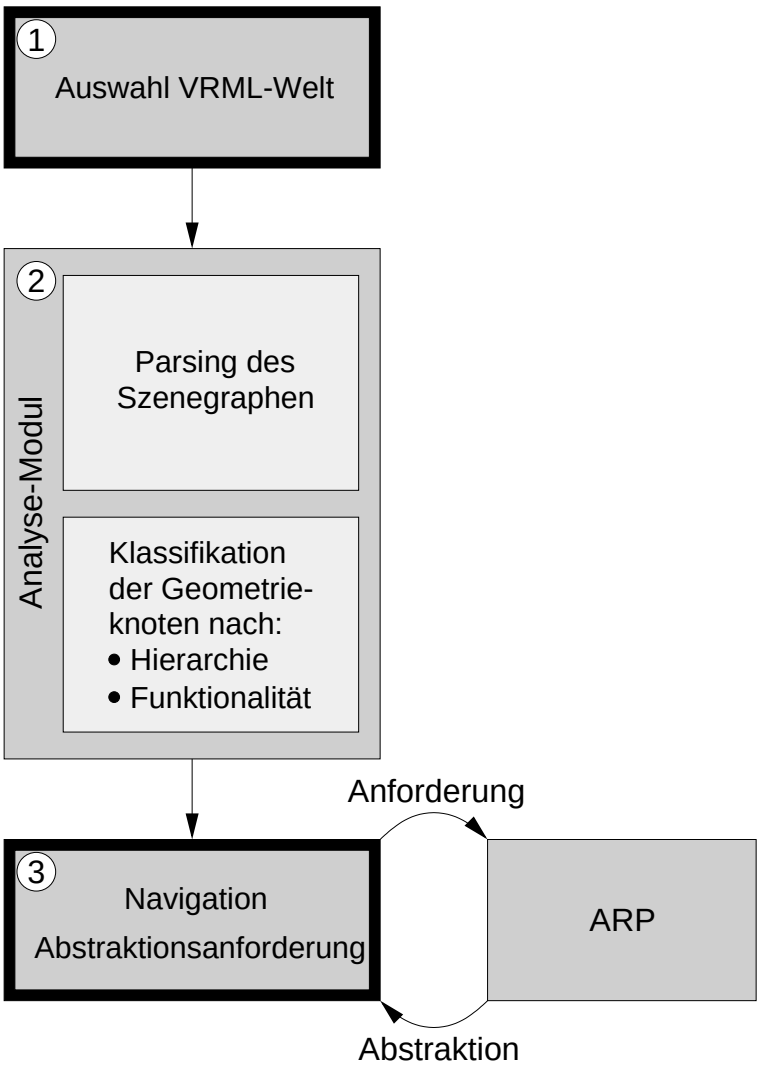


Abbildung 4.3: Ablauf der dynamischen VAI-Variante: Abstraktionen werden zur Laufzeit erzeugt

4.2.2 Statisches VAI

4.2.2.1 Architektur

Die Architektur des statischen VAI-Systems mit Vorberechnung der Abstraktionen ist in Abbildung 4.4 dargestellt. Sie ähnelt der dynamischen Version, da ebenfalls die Komponenten ARP, BUNDY und der Comsoplayer verwendet werden. Anstelle des Java-Applets kommt jedoch eine Java-Applikation zum Einsatz. Dieses Programm erhält die VRML-Welt als Eingabe und übergibt die Geometrien in Form von separaten Dateien an ARP.

Laufen VAI und ARP auf verschiedenen Rechnern, muß ein gemeinsames Netzwerkfilesystem vorhanden sein. Das VAI erweitert den VRML-Szenegraphen um die Abstraktionen und erzeugt eine neue VRML-Datei. Diese wird über den HTML-Browser geladen und im Cosmoplayer dargestellt.

4.2.2.2 Ablauf

Die Anwendung des statischen VAI-Systems läßt sich in 5 Schritten beschreiben, die in Abbildung 4.5 dargestellt sind. Sie stimmen am Anfang mit dem Ablauf der Online-Version überein. Als erstes werden die VRML-Datei ausgewählt und die VAI-Applikation gestartet. Das statische VAI enthält ebenfalls ein Analyse-Modul, welches im zweiten Schritt ausgeführt wird und die Geometrieknoten nach Hierarchie und Funktionalität klassifiziert.

Im Unterschied zur dynamischen Variante beinhaltet die Navigation keine Iterativen Aufrufe von ARP. Das Abstraktionssystem wird nur einmal im dritten Schritt aufgerufen, um alle möglichen Verschmelzungen entlang der Objekthierarchie von unten nach oben zu erzeugen. Dazu wird die Hierarchie der Geometrieknoten an ARP in Form einer in LISP-Stil geklammerten Liste übergeben. Diese Liste enthält zu jedem Geometrieknoten seinen Namen, seine Funktionalität und alle Nachfolger.

Das Abstraktionssystem erzeugt nun alle Abstraktionsgrade, die sich direkt aus der Objekthierarchie ableiten lassen. Dazu beginnt ARP auf der untersten Hierarchieebene damit, benachbarte Teilobjekte paarweise miteinander zu verschmelzen und anschliessend zu abstrahieren. Die Objekte müssen dazu ähnliche Farbe haben und dürfen sich nicht schneiden. Sind keine solchen Objekte mehr vorhanden, wird das Verfahren auf der nächsthöheren Ebene fortgesetzt. Dabei werden nun in einer Rekursion die bereits erzeugten Abstraktionen der benachbarten Ebenen noch einmal verschmolzen und abstrahiert. Ist schliesslich an der Wurzel der Objekthierarchie eine Abstraktion der gesamten Szene berechnet worden, terminiert ARP und liefert die entstandene Abstraktionshierarchie an VAI zurück. Die Geometrien der Zwischenergebnisse werden in einzelnen Dateien abgelegt.

Im vierten Schritt liest das VAI-System nun diese Liste ein und fügt mit ihrer Hilfe die entstandenen Abstraktionen an den entsprechenden Stellen des VRML-Szenegraphen ein. Dazu wird die Szene um die Konsole und Skripte erweitert, um bei der Navigation interaktiv zwischen Original und Abstraktion wechseln zu können. Der erweiterte Szenegraph wird nun in Form einer VRML-Datei ausgegeben.

Jetzt kann im letzten Schritt die neue VRML-Datei über den HTML-Browser geöffnet werden. Dieser startet das entsprechende Plugin für VRML Daten. In der prototypischen Umgebung ist dies der Cosmoplayer. Der Benutzer kann sich nun in der Virtuellen Realität bewegen und mit Hilfe der Konsole Objekte verschmelzen oder detailliert darstellen.

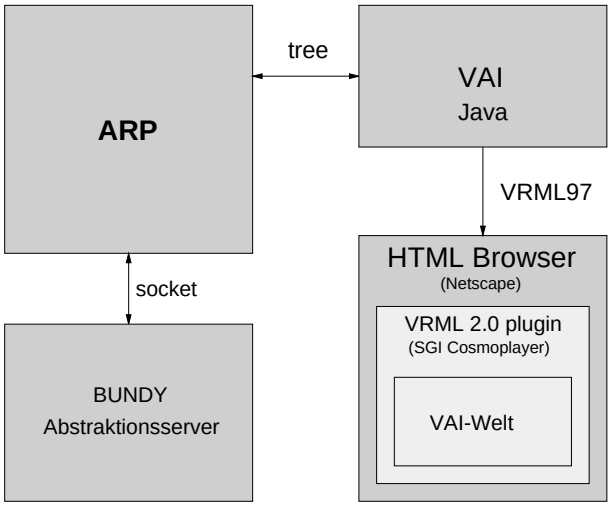


Abbildung 4.4: Architektur der statischen Verarbeitung mit Vorberechnung

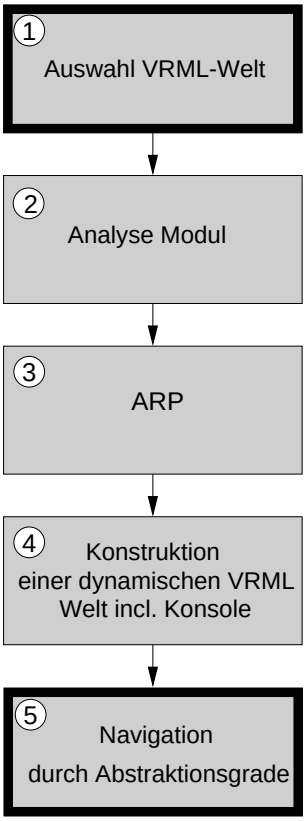


Abbildung 4.5: Ablauf bei dynamischer Verarbeitung mit online-Verbindung

4.3 Benutzerschnittstelle

In diesem Abschnitt wird die Benutzerschnittstelle des VAI-Systems vorgestellt. Sie ermöglicht es dem Benutzer, den Abstraktionsgrad der Szene seinen individuellen Bedürfnissen anzupassen. Die Gestaltung der Schnittstelle spielt eine wichtige Rolle, damit das System leicht verständlich und praktisch einsetzbar ist. Es handelt sich ja um eine System, das die Navigation erleichtern und nicht erschweren soll. Um dieses Designziel zu erreichen, wurden das Ergebnis der Diskussion des Literaturteils beim Entwurf berücksichtigt (siehe Abschnitte 2.2.2 und 2.2.1 sowie 2.8). Optimal ist demnach ein in die Szene integriertes Menü in Form einer Konsole, da eine direkte Manipulation alleine nicht ausreichen würde. In der statischen Variante kommen jedoch zusätzliche 3D-Widgets zum Einsatz, die eine direkte Manipulation des Abstraktionsgrades ermöglichen.

Die Benutzerschnittstelle zur dynamischen VAI-Version wird in Abchnitt 4.3.1 vorgestellt, die der statischen Variante in Abschnitt 4.3.2. Zum Abschluss folgt in Abschnitt 4.3.3 eine Zusammenfassung der jeweiligen Vor- und Nachteile.

4.3.1 Dynamische VAI-Version

Das VAI lässt sich in der dynamischen Version sowohl über eine in der VRML- Welt integrierte Steuerkonsole als auch parallel dazu über Java-Menüs im Applet bedienen. Die Konsole ist in Abbildung 4.6 dargestellt. Über die Disketten-Ikone kann durch einen Dateiauswahl-Dialog eine VRML-Welt geöffnet und geladen werden. In der geladenen Welt können nun die Abstraktionsgrade von Objekten manipuliert werden. Dazu wird als erstes das Objekt mit der Maus berührt. Dabei gibt es ein optisches Feedback, indem das Objekt durch ein Spotlight aufgehellt wird. Ein Beispiel wird in Abbildung 4.7 gezeigt. So lassen sich gleichfarbige, aneinander angrenzende Objekte leichter voneinander unterscheiden. Nun klickt man das Objekt an, um es auszuwählen. Dabei ist prinzipiell auch eine Mehrfachauswahl von Objekten möglich. Jetzt können die ausgewählten Objekte über die **-** Ikone abstrahiert und verschmolzen werden, dabei gehen Details verloren. Umgekehrt führt die **+** Ikone von einer Verschmelzung zu deren ursprünglichen Objekten mit allen Details zurück. Die **+** und **-** Symbole stehen dabei als Metapher für den Detaillierungsgrad (und in umgekehrter Relation zum Abstraktionsgrad).

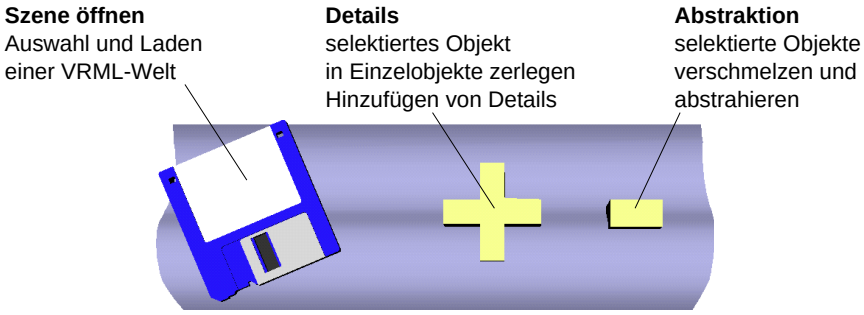


Abbildung 4.6: Konsole der dynamischen Variante

4.3.2 Statische VAI-Version

Die Benutzeroberfläche der statischen Version wird durch VAI im Rahmen der Vorberechnung der Abstraktionsgrade fest in die VRML-Welt integriert. Die Auswahl einer Welt wie bei der dynamischen Version entfällt daher in der Benutzeroberfläche. Die Steuerung der Abstraktion findet in verschiedenen Ebenen statt; Es gibt eine Konsole für globale Einstellungen und es gibt ein objektbezogenes Menü.

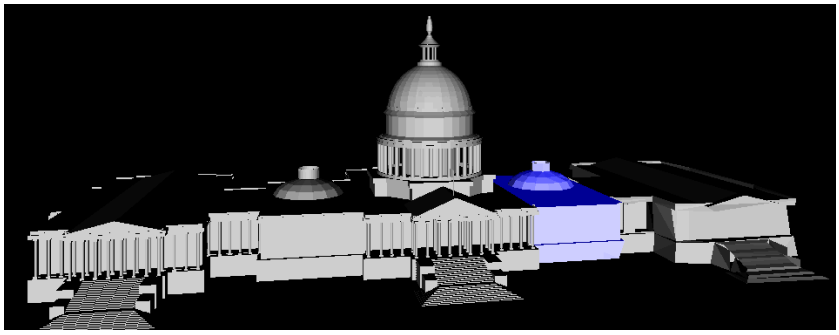


Abbildung 4.7: Auswahl eines Teilobjektes mit Visualisierung durch Spotlight

4.3.2.1 Konsole

Die Konsole ist in Abbildung 4.8 dargestellt. Sie beeinflusst die Abstraktionsgrade aller Objekte in der Szene und ist in drei Ebenen gegliedert. In der obersten Ebene kann über die **+** Ikone die Abstraktion abgeschaltet und die Welt in ihrem vollen Detailreichtum angezeigt werden. Umgekehrt schaltet die **-** Ikone auf die höchstmögliche Stufe der Abstraktion, in der alle Objekte so weit wie möglich miteinander verschmolzen sind. Zwischen beiden Extremen liegt die Fokus-Sicht, die die Aufmerksamkeit des Betrachters auf Objekte bestimmter Funktionalität fokussieren soll, und mit der **F** Ikone aktiviert wird. Dabei werden alle Objekte mit den gewünschten Eigenschaften detailliert dargestellt, die übrigen werden miteinander verschmolzen. Die Fokus-Sicht wird in der zweiten Ebene der Konsole konfiguriert.

Die Funktionalitäten leiten sich direkt aus dem VRML-Ereignismodell ab. Dabei stellen Routen eine Verbindung zwischen Objekten her, die Ereignisse erzeugen oder empfangen. Eine detaillierte Beschreibung des Modells findet sich in den Abschnitten 2.6 und 3.2. Es kann auf die Klassen der Sender und Empfänger von Ereignissen fokussiert werden. Eine dritte Klasse bilden Hyperlink-Objekte, die zwar unabhängig von Ereignissen arbeiten, aber dennoch von hoher Bedeutung für die Navigation sind. Sie können den Betrachter in andere Welten transportieren oder HTML-Dokumente mit Informationen aufrufen. Bei der Auswahl der Funktionalitäten können alle drei Klassen in beliebiger Kombination über Ikonen aktiviert werden. Die Ikonen der zweiten Ebene, von links nach rechts ausgezählt, symbolisieren Sender, Empfänger sowie Hyperlinks und leuchten grün auf, wenn ihre Funktion aktiv ist.

Über die dritte Ikonenreihe kann die Abstraktion auf Generierungsebene eingestellt werden. Sie beinflußt alle Objekte der Szene, die auch schon auf Modellebene abstrakt dargestellt werden. Dazu gehören bei der Fokus-Sicht alle Objekte, die nicht im Fokus liegen. Dabei werden die Darstellungsattribute der Objekte modifiziert. Die Abstraktion kann durch Auswahl der linken Kugel ausgeschaltet werden, dadurch werden alle Objekte in ihrer natürlichen Farbe dargestellt. Die mittlere Kugel schaltet auf eine abstrakte Darstellung in einfarbigem, unschattiertem Grau. Die rechte Kugel blendet die Objekte ganz aus. Die aktive Einstellung wird durch eine vergrösserte Kugel verdeutlicht.

Die Konsole wurde der Optik und Funktionalität von bekannten 2D-Benutzeroberflächen wie dem *FVWM* Fenstermanager [fvw96] für das Unix X-Windows System nachempfunden. Dabei stellt sich der bei 2D-Oberflächen durch helle und dunkle Ränder simulierte 3D-Effekt durch eine wirklich vorhandene Tiefe der Konsole naturgemäss von selbst ein. Die Konsole hat eine Titelleiste, mit der sie frei im Bildschirm plaziert werden kann. Über die rechte obere Ikone kann die Konsole ausgeblendet und als Symbol dargestellt werden, falls sie als störend empfunden wird.

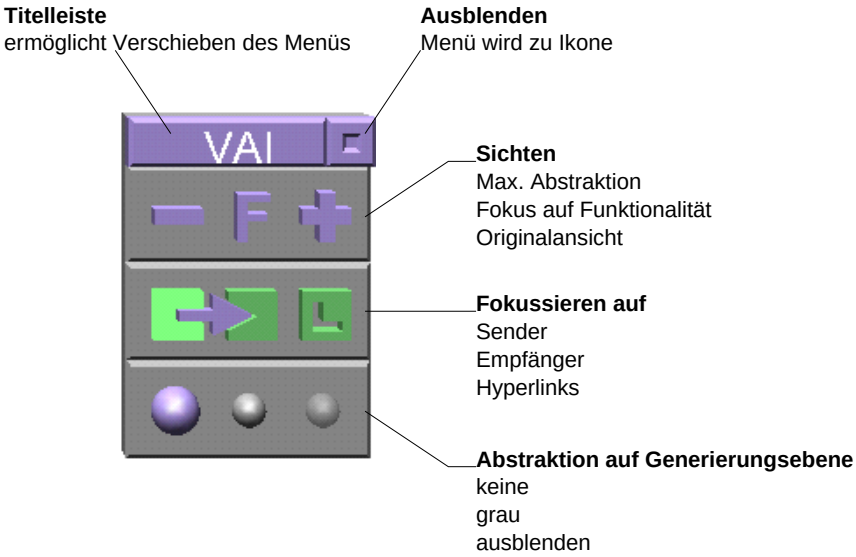


Abbildung 4.8: Konsole der statischen VAI-Version

4.3.2.2 Objektmenü

Hat man über die Konsole eine Sicht ausgewählt, die den Abstraktionsgrad für alle Objekte voreingestellt hat, können die Objekte nun individuell manipuliert werden. Dazu bewegt man die Maus über das Objekt. Daraufhin erscheint dort ein Menü mit einer**+** und **-** Ikone. Die Abbildung 4.9 zeigt als Beispiel einen Ausschnitt von Abbildung 6.15(b). Klickt man auf die **+** Ikone, werden dem Objekt Details hinzugefügt. Handelt es sich bereits um ein volldetailliertes Objekt, ist diese Ikone deaktiviert und transparent dargestellt. Ansonsten zerfällt die Verschmelzung in einzelne Objekte. Diese können wiederum berührt und über die **+** Ikone zerlegt werden, bis hin zu den volldetaillierten Einzelobjekten. Umgekehrt lassen sich Objekte über ihre **-** Ikone abstrahieren und mit den Nachbarn der gleichen Hierarchiestufe verschmelzen. Dazu wird die Abstraktion auf Generierungsebene entsprechend den Einstellungen der Konsole wirksam.

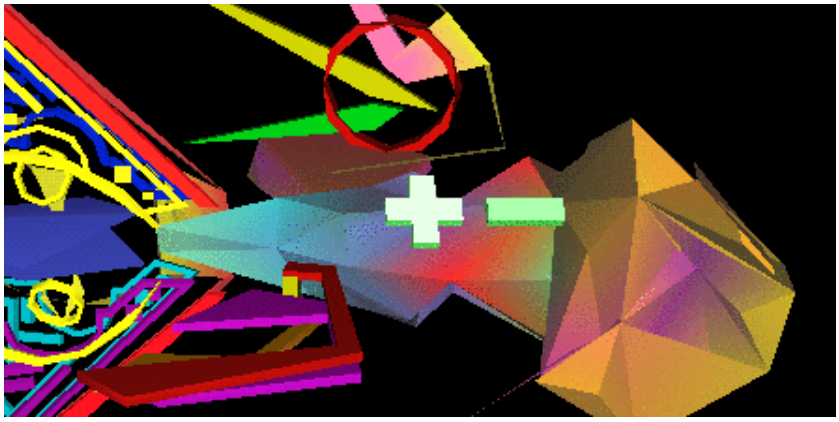


Abbildung 4.9: Objektmenü zur direkten Manipulation des Abstraktionsgrades

4.3.3 Vergleich beider VAI-Varianten

	Eigenschaft	dynamisches VAI	statisches VAI
Abstraktion	Berechnung der Geometrie	zur Laufzeit	vor Navigation
	Abstraktion mit Verschmelzen	beliebige Gruppen selektierbar	Söhne eines Knotens der Objekthierarchie
GUI	Szene öffnen	über Konsole	n. a.
	Funktionale Sichten abrufen	n. a.	über Konsole
	Abstraktionsgrad manipulieren	Auswahl über Konsole	direkt, über Objektmenü

Abbildung 4.10: Ein tabellarischer Vergleich der Möglichkeiten beider VAI-Varianten.

Die in Abbildung 4.10 dargestellte Tabelle zeigt die wesentlichen Möglichkeiten beider VAI-Varianten. Die Eigenschaften bzw. Fähigkeiten sind in der linken Spalte aufgeführt und in die Bereiche Abstraktion und Benutzerschnittstelle gegliedert. In der mittleren und rechten Spalte ist angegeben, ob und wie die jeweilige VAI-Variante diese unterstützt.

Kapitel 5

Implementation des VAI Systems

Dieses Kapitel beschreibt die Implementierung des *VRML Abstraction Interface*-Systems. In Kapitel 4.1 wurden alternative Betriebsmodi diskutiert und dementsprechend zwei verschiedene Systeme entworfen. Die beiden Varianten unterscheiden sich vor allem in ihrem Ablauf, aber auch in ihrer Architektur. In der dynamischen Version werden die Abstraktionen während der Navigation auf Anfrage des Betrachters generiert und in die VRML-Welt eingefügt. Im Gegensatz dazu werden bei der statischen Version alle Abstraktionsgrade vorberechnet. Die erzeugte VRML-Welt enthält neben dem User Interface alle Abstraktionsgrade. In dieser Variante muß der Betrachter auf individuelle Abstraktionen verzichten. Dafür entfallen Wartezeiten für das Berechnen und Laden von neuen Geometrien.

Zur Implementierung stand als Hardware ein PC mit 333 Mhz *Intel Pentium II* CPU und 128 MB Hauptspeicher zur Verfügung. Der PC war mit einer *GLORIA SYNERGY* Grafikkarte der Firma *ELSA* zur 3D-Beschleunigung ausgestattet. Zuerst wurde die Online-Version unter dem *Microsoft Windows NT 4.0* Betriebssystem entwickelt. Programmiert wurde in der Sprache *Java* [JSGB00] mit dem JDK[1] der Firma *Sun Microsystems*. Die Konsole wurde mit Hilfe der Software *Lightwave* [New01] modelliert.

Nach der dynamischen wurde die statische VAI-Variante unter dem *Linux* Betriebssystem entwickelt. Ein bereits vorhandenes Java-Klassenpaket zum Parsing von VRML-Szenen legte die vollständige Programmierung in Java nahe; dabei kam ebenfalls das JDK zum Einsatz. Die Geometrie des User Interface wurde mit einem Texteditor von Hand modelliert, die Funktionalität in Javascript programmiert. Die erzeugten VRML Szenen wurden unter dem *Microsoft Windows* Betriebssystem mit dem *Cosmoplayer* [Sil98] betrachtet.

Die Implementation der dynamischen Variante wird in Abschnitt 5.1 beschrieben, die der statischen Variante folgt in Abschnitt 5.2. Die Primitiverkennung wird in Abschnitt 5.3 anhand von Datenflußdiagrammen in Verbindung mit Pseudocode beschrieben.

5.1 Dynamischer Betriebsmodus mittels EAI-Technik

Dieser Abschnitt beschreibt die Implementierung der dynamischen Version des VRML Abstraction Interface. Es wurde ein Prototyp entwickelt, um das Konzept zu evaluieren. Dabei wurde auf einige der im Entwurf festgelegten Eigenschaften verzichtet, um den Aufwand der Implementierung zu verringern. So wurde keine echte online-Verbindung zum Abstraktionssystem ARP realisiert, sondern die VRML-Modelle müssen von Hand mit einer Software wie *Amapi 3D* [ama98] hierarchisch in einzelne Komponenten und Dateien zerlegt werden. Die entstandenen Teilobjekte werden mittels der Benutzeroberfläche von BUNDY abstrahiert. Die Objekthierarchie wird als Baum von Dateinamen in einer Strukturinformationsdatei gespeichert. Auf eine Auswahl von funktionalen Sichten wird im Prototyp

[1] Java Development Kit

ebenfalls verzichtet. Die Abstraktionen werden jedoch während der Navigation online über das Netzwerk geladen und in die VRML-Szene eingefügt.

Die dynamische Version des VAI-Systems besteht, wie schon in Abschnitt 4.2.1.1 beschrieben, aus einem Java-Applet und einer VRML-Welt, die über ein HTML-Dokument verknüpft sind, um miteinander zu interagieren. Die VRML-Welt enthält eine Konsole, die es dem Benutzer erlaubt, Abstraktionsgrade zu manipulieren. Das Applet enthält den dazu notwendigen Java-Programmcode.

5.1.1 Beschreibung des Szenegraphen

Im diesem Absatz werden der zugrundeliegende VRML-Szenegraph an einer Abbildung erläutert und ein kurzer Einblick in das Java-Applet gegeben. Eine kurze Einführung in die verwendeten Techniken ist in Abschnitt 2.6 zu finden.

Die VRML-Szene verbindet die Steuerkonsole, die zu abstrahierenden Objekte und das Applet über Ereignis-Routen. Abbildung 5.1 zeigt den VRML-Szenegraphen mit den wichtigsten Knoten. Das Java-Applet ist kein VRML-Bestandteil, interagiert jedoch mithilfe des EAI über Ereignisse mit der Welt. Die Knoten des Szenegraphen sind in der Abbildung als Rechtecke dargestellt. Die oberste Zeile in blau enthält eine selbstgewählte Bezeichnung, über die der Knoten referenziert wird. Im folgenden Text werden diese Bezeichnungen in Anführungszeichen gesetzt. Die zweite Zeile enthält in weiß den Knotentyp, der im Text kursiv gesetzt ist. Darunter folgen je nach Knotentyp mehrere Datenfelder, dargestellt in gelb und im Text ebenfalls kursiv gesetzt.

Der *Group* Knoten ”Root” (oben links im Bild) enthält die Modellkomponenten der zu abstrahierenden Welt als *Children*. Diese Komponenten enthalten jeweils einen *TouchSensor* Knoten mit dem Feld *isActive changed*, der ein Anklicken des Modells registriert und dieses Ereignis an das Java-Applet sendet. Diese Verbindung wird durch den Pfeil in der Abbildung symbolisiert.

Der übrige Teil des Szenegraphen ist alleine für die Steuerungskonsole mit ihren umfangreichen interaktiven Möglichkeiten zuständig. Die wichtigste Funktion besteht darin, die Konsole stets den Bewegungen des Betrachters nachzuführen. Dazu wird ein *ProximitySensor* eingesetzt, der jede Veränderung der Kameraposition und Orientierung registriert und an den *Transform* Knoten ”TRANS” als Ereignis weiterleitet. Dieser Transformationsknoten positioniert die Konsole, die am *Children* Feld hängt. Ein zweiter Transformationsknoten ”ConTrans” dient zur Positionierung innerhalb des Bildschirms, relativ zur Kameraposition. So kann die Konsole vom Benutzer in eine beliebige Ecke des Bildschirms plaziert werden. Ein *PlaneSensor* erfasst Mausbewegungen auf der Konsole und sendet diese an den Transformationsknoten, der die Konsole der Maus folgen lässt.

Im unteren Bildteil sind die Geometrien der Konsole dargestellt. Von links nach rechts der Konsolenhintergrund, die Diskettenikone und die Plus- und Minus-Ikonen. Die Ikonen enthalten jeweils einen *TouchSensor*, der auf die Maus reagiert. Bei Berührung wird die Ikone durch die Lichtquelle ”LStart” beleuchtet. Klickt der Benutzer die Ikone an, wird dieses Ereignis direkt über eine EAI-Route an das Java-Applet geschickt.

5.1.2 Funktionsweise des Java-Applets

Das Applet reagiert auf Mausklicks des Benutzers und nimmt entsprechende Veränderungen im Szenegraphen vor. Wird eine neue Welt über die Diskettenikone geöffnet, wird zuerst die dazugehörende Strukturinformation geladen. Diese enthält eine Abstraktionshierarchie, an deren Wurzel sich eine Verschmelzung der gesamten Welt befindet. Diese wird über das *addChildren* Feld des ”Root” Knoten in den Szenegraph eingefügt. Klickt der Benutzer auf die **+** Ikone, wird über das EAI die entsprechende Methode des Applets aufgerufen. Diese entfernt die Geometrie des Knotens über das *removeChildren* aus dem Szenegraphen und fügt dafür die Nachfolger der Abstraktionshierarchie neu ein. Der Vorgang der Abstraktion mit der **-** Ikone verläuft analog.

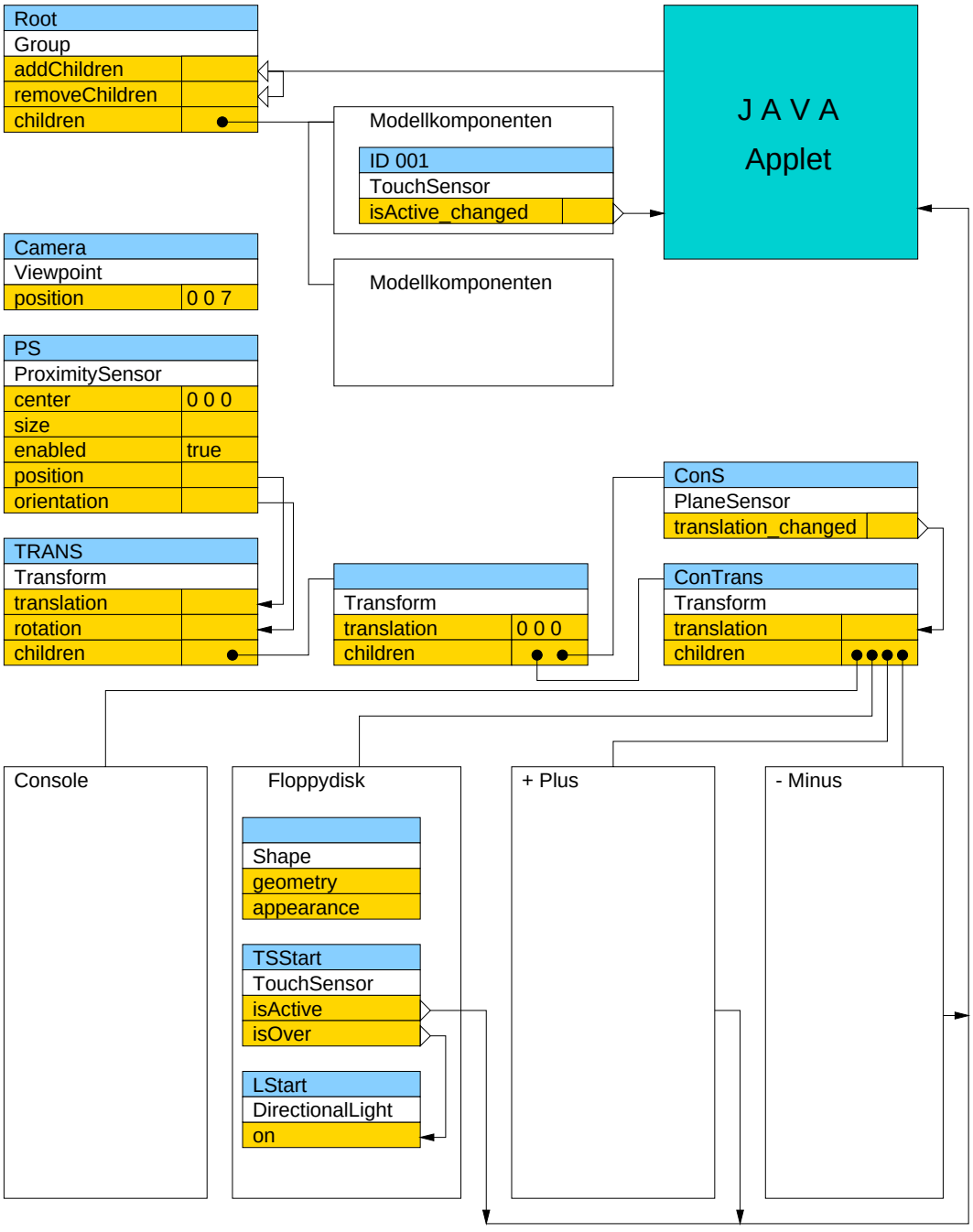


Abbildung 5.1: VRML Szenegraph mit Konsole

5.2 Statisches VAI mit VRML-Vorberechnung

Dieses Kapitel beschreibt die Implementierung der Offline-Version des VRML Abstraction Interface. Abschnitt 5.2.1 beschreibt den Ablauf der Vorberechnungsphase. In Abschnitt 5.2.2 wird der Einsatz von Javascript Programmcode in VRML behandelt. Dort wird in Abschnitt 5.2.2.1 gezeigt, wie dieser Code verwendet wird, um die Funktionen der Konsole zu realisieren. Im Abschnitt 5.2.2.2 wird erklärt, wie im VRML-Szenegraphen zwischen den verschiedenen Abstraktionsgraden umgeschaltet wird.

5.2.1 Vorberechnung der VAI-erweiterten VRML-Welt

Die Abbildung 5.2 stellt den Ablauf der Vorberechnungsphase von VAI dar und umfasst den Datenfluss von der Eingabe VRML-Welt bis zur Ausgabe der interaktiven, VAI-erweiterten Welt. Das Listing zeigt parallel dazu den Programmcode der *main(..)* Methode. Eine Darstellung der verwendeten Klassen mit ihren Attributen und Methoden findet sich im Anhang B.1.

Alle Datenstrukturen und Methoden, die in der Abbildung schematisch dargestellt sind, werden von der Klasse *VRMLgraph* zusammengefasst. So wird in der *main(..)* Methode nur ein einziges Objekt instanziert (Listing Zeile 16), auf dem nacheinander alle im Diagramm dargestellten Methoden aufgerufen werden.

Die Quelle des Datenflusses bildet eine beliebige VRML-Datei, die z.B. aus dem Internet heruntergeladen werden kann. Die VRML-Datei enthält einen als ASCII-Text kodierten Szenegraphen, so daß ein Parser benötigt wird, um den Graphen einzulesen und auf Java-Objekte abzubilden. Dazu wird das Java Klassenpaket *iicm.vrml.pw*[2] verwendet (Zeile 2). Es ist Bestandteil des freien Open-Source VRML-Browsers *VRwave* [Ins00].

Der Szenegraph enthält typischerweise unterschiedlichste Knotentypen, die von VRML zur Beschreibung der Szene benötigt werden. In der Abbildung sind die verschiedenen Knotentypen zur Veranschaulichung grob klassifiziert und durch verschiedene Farben symbolisiert worden. So sind Geometrieknoten schwarz, Gruppenknoten weiß und sonstige Knoten wie Sensoren, Scripte oder Viewpoints grau dargestellt.

Zur Abstraktion mit dem System ARP müssen die Geometrieinformationen zusammen mit ihrer Hierarchie und Funktionalität extrahiert werden. Dieser Vorgang wird in Abschnitt 3.2 im Detail beschrieben. In der Implementierung wird ein Traverser (Methode *traverse()*, Zeile 20) eingesetzt, um den Graphen von der Wurzel aus rekursiv zu durchlaufen und einen neuen Baum aus einfachen Knoten der Klasse *ANode* aufzubauen. Dieser im Programmcode als SSG[3] bezeichnete Baum wird in der *vrmlgraph* Instanz unter dem Attribut *aroot* gespeichert und enthält nur Geometriedaten. Zur Analyse der Funktionalität dieser Geometrieknoten werden die Routenverbindungen im Szenegraph betrachtet. Dazu wird die Methode *route()* aufgerufen (Zeile 19). Der vereinfachte Baum wird durch die Methode *writeARP()* (Zeile 22) als geklammerte Liste in LISP Syntax ausgegeben. Die Listenelemente referenzieren Dateien, die die Geometrie im *OFF* Format enthalten, das in Abschnitt 2.3.1 vorgestellt wurde. Die Klasse *PolygonSet* implementiert die erforderlichen Konvertierungsmethoden, um VRML- Geometrien und Primitive im *OFF* Format auszugeben. In der Abbildung ist dieser SSG Baum links unter dem Szenegraphen dargestellt, Knoten mit Geometrie erscheinen schwarz. Weiße Knoten ohne Geometrie entstehen aus VRML-Gruppierungsknoten.

Jetzt kann ARP gestartet werden und bekommt als Eingabe die Objekthierarchie in Listenform. ARP erzeugt zuerst Abstraktionen von allen Geometrien. Danach berechnet ARP entlang der Hierarchie (von unten nach oben) Verschmelzungen der Objekte. Der mit den erzeugten Verschmelzungen angereicherte Baum wird wieder als Liste ausgegeben. In Anhang B.2 sind als Beispiel die Bäume der Celiafish-Welt auszugsweise als Liste abgebildet.

[2] Copyright (c) 1996,97 Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Austria.

[3] Stripped Scene Graph

Nun wird erneut VAI aufgerufen und bis Zeile 23 ausgeführt. Über die Methode *readARP()* wird der neue Baum eingelesen und im Speicher auf Objekte der Klasse *ANode* abgebildet, wobei die Wurzel im Attribut *a2root* gespeichert wird. Im ersten Durchlauf war das Lesen der Datei noch erfolglos, so daß das Programm in Zeile 25 terminiert. Jetzt wird das Programm in Zeile 26 fortgesetzt.

Der angereicherte Baum ist in der Abbildung rechts unten dargestellt. Die vorher leeren Gruppierungsknoten enthalten nun Verschmelzungen ihrer Nachfolger und sind schwarz dargestellt. Die Methode *switchify()* (Zeile 27) sucht nun alle diese Gruppierungsknoten heraus und speichert ihre Namen in einer Liste zur späteren Verwendung durch Javascript. Aus dem modifizierten und angereicherten SSG muß nun wieder der vollständige VRML-Szenegraph rekonstruiert werden. Diese Transformation ist in der Methode *graphToVRML()* (Zeile 28) implementiert. Dabei werden die vorher mit *switchify* identifizierten Knoten durch *Switch* Knoten erweitert, die zwischen Originaldarstellung (also den Nachfolgern) und der abstrakten Verschmelzung umschalten.

Die von ARP generierten und vom SSG referenzierten Abstraktionen werden mithilfe der Methode *createIndexedFaceSet_grey()* der Klasse *PolygonSet* eingelesen. Die Methode erzeugt einen VRML *Switch* Knoten, der die Abstraktion auf Generierungsebene ermöglicht, indem auf ein Modell in einheitlichem unschattierten Grau umgeschaltet werden kann. Dieser *Switch* Knoten ermöglicht auch das Abschalten und Ausblenden der Abstraktion. Das Schalten wird später während der Navigation interaktiv durch die Konsole ausgelöst. Dazu wird die Konsole an ein zentrales Javascript-Programm gekoppelt, das die Knoten entsprechend schaltet.

Der mit Schaltern und Abstraktionen auf Modell- und Generierungsebene angereicherte VRML-Szenegraph kann nun ausgegeben werden. Die Methode *writeVRML()* (Zeile 29) generiert aus dem aus Java-Objekten bestehenden Baum eine Datei im *VRML 2.0* Format. Dabei wird rekursiv von jedem Knoten die *write()* Methode aufgerufen. Diese ist in den *pw* Klassen implementiert und gibt die Felder des Knotens der VRML-Syntax entsprechend als String aus.

Die Methode *scriptify()* (Zeile 30) generiert ein Javascript-Programm, dessen Funktionen durch Ereignisse von der Konsole aufgerufen werden und das über die Schalter des Szenegraphen die Abstraktionsgrade entsprechend der Benutzeranforderung auswählt. Dieses Programm wird als *Script* Knoten an die VRML-Welt angehängt. Als letzter Schritt wird die Steuerkonsole in die VRML-Datei kopiert. Die fertige VRML-Welt kann nun in einem Browser geöffnet werden.

```
 1 package afc.vain;
 2 import iicm.vrml.pw.*;
 3 import java.io.*;
 4 import java.util.*;
 5
 6 class Vain {
 7
 8  public static void main(String args[]) {
 9
10    if (args.length < 1)
11    {
12      System.out.println ("usage: Vain vrml.wrl");
13      return;
14    }
15
16    VRMLgraph vrmlgraph=new VRMLgraph();
17    vrmlgraph.parse(args[0]);
18    if (vrmlgraph.root==null) return;
19    vrmlgraph.route();
20    vrmlgraph.traverse();
```

```
21
22    vrmlgraph.writeARP( vrmlgraph.aroot,
                          vrmlgraph.arpname );
23    vrmlgraph.readARP( vrmlgraph.a2root,
                         vrmlgraph.absname );
24
25    if (vrmlgraph.a2root!=null) { //world.abs exists:
26      vrmlgraph.a2root.inheritFunc(false, false, false);
27      vrmlgraph.switchify();
28      vrmlgraph.graphToVRML();
29      vrmlgraph.writeVRML( vrmlgraph.abstVrml,
                             vrmlgraph.worldpath+"abstract.wrl" );
30      vrmlgraph.scriptify( vrmlgraph.worldpath+"abstract.wrl" );
31    } else {
32      System.out.println("\n\nno .abs file found.
         make it with ARP from .arp file.");
33    }
34  }
35 }
```

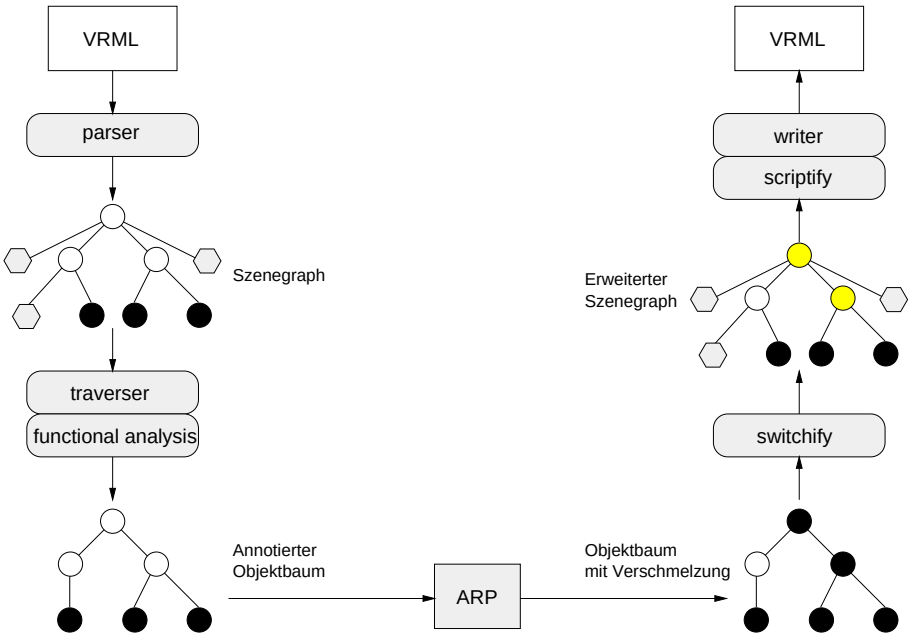


Abbildung 5.2: Datenstrukturen in V.A.I.

5.2.2 Verwendung von Javascript-Code in VRML

Die von Menü und Konsole gebotenen Interaktionsmöglichkeiten übersteigen den Rahmen dessen, was mit einfachem Routen von elementaren Ereignissen in VRML machbar ist. Zu ihrer Realisierung wird zusätzliche Logik benötigt, die in der Javascript Programmiersprache formuliert und über das VRML-Konzept der Script-Knoten in die Szene eingebunden wird. Script-Knoten enthalten Variablen, die auch auf andere Knoten zeigen können, sowie Funktionen, die von Ereignissen aufgerufen werden. Das Ereignis beinhaltet einen Parameter wie z.B. eine Koordinate oder Zahl, der von der Funktion ausgewertet werden kann. Im folgenden sollen zuerst der zur Konsole gehörende Programmcode und danach die für das Verschmelzen zuständige Logik des Objektmenüs beschrieben werden. Daraus wird auch ersichtlich, wie sämtliche Abstraktionen in die VRML-Welt integriert sind.

5.2.2.1 Implementierung der Konsole

Das Abstraktionssystem ARP erzeugt entlang der Objekthierarchie alle möglichen Abstraktionen, indem Teile paarweise verschmolzen werden. Diese Abstraktionen werden der original VRML-Welt hinzugefügt. Dabei kommen im Szenegraphen hierarchisch geschachtelte Switch-Knoten zum Einsatz, die zwischen Verschmelzung und Originalgeometrie umschalten. Eine Schachtelungsebene des Szenegraphen ist in Abbildung 5.3 dargestellt. Dieser Abschnitt beschreibt, wie die Konsole die Switch-Knoten ansteuert.

Speziell für die Konsole enthält die VRML-Welt den Script-Knoten *CONTROLscript*, die zu den Ereignissen gehörenden Funktionen sind in der externen Datei *controlscript.js* in Javascript implementiert. Ein Teil der darin enthaltenen Funktionen ist zur Realisierung der Benutzeroberfläche notwendig. Dazu gehört der Leuchteffekt, der in der zweiten Ebene der Konsole anzeigt, welche der drei Funktionalitäten gerade aktiviert sind. Der aktuelle Zustand wird in der Variablen *mode* gespeichert und bei jedem Anklicken umgeschaltet.

Weitere Funktionen sind für die Auswahl der Kugeln der dritten Ebene der Konsole zuständig, welche die Abstraktion auf Generierungsebene kontrollieren. Dabei wird je nach Kugel die Funktion *color()*, *grey()* bzw. *none()* aufgerufen, die von einer kleinen auf eine große Kugel umschaltet und die Funktion *set(value)* des Scriptknotens *switchrender* aufruft. Dieses Script sorgt für das Umschalten der Darstellungsweise aller abstrakten Objekte der Szene. Jede Abstraktion wird sowohl als farbige als auch graue Geometrie in die VRML-Szene eingebunden. Übergeordnete Switch-Knoten schalten zwischen beiden Alternativen um oder beide aus. In Abbildung 5.3 ist dieser Switchknoten entsprechend der Implementierung als *IFS* bezeichnet. Das Script enthält Zeiger auf alle diese Switchknoten, die Funktion schaltet sie entsprechend dem Parameter *mode* um.

Analog dazu sind die Funktionen der obersten Ebene der Konsole implementiert, nur daß hier der als *SWIn* bezeichnete Switch-Knoten zwischen Abstraktion und originalen Nachfolgern umschaltet, wie es in Abbildung 5.3 dargestellt ist. Die **+** und **-** Ikonen verwenden wieder die gleiche Funktion set(value), um alle Switches auf Original bzw. Abstraktion zu schalten. Die **F** Ikone führt die Funktion modefy(value) aus. In ihr ist ein logischer Test implementiert, der bei jedem Objekt prüft, ob mindestens eine der in der zweiten Ebene eingestellten Funktionalitäten gegeben ist. Die Funktionen set() und modefy() sind in der Datei *switchallscript.js* zu finden.

5.2.2.2 Implementierung der Auswahl von Abstraktionsgraden im Szenegraph

Im vorangegangenen Abschnitt wurde erklärt, wie über die Konsole funktionale Sichten abgerufen werden können. Das Objektmenü hingegen ermöglicht es dem Betrachter, interaktiv beliebige Objektteile auszuwählen und zu verschmelzen. Dazu sind die Touch-Sensoren im Szenegraphen vorgesehen, die in der Abbildung 5.3 als *SWInA* und *SWInB* bezeichnet werden. Diese Sensoren melden sowohl das Berühren als auch das Anklicken der Geometrie an das Objektmenü. Der Sensor A bezieht sich auf die Abstraktion, während

Sensor B auf die Originalgeometrie reagiert. Abhängig vom Switch SWI ist entweder die Abstraktion (farbig oder grau) sichtbar und Sensor A aktiv, oder der original Teilgraph ist sichtbar und Sensor B reagiert auf Berührungen. Beide Sensoren melden die Position der Berührung an das zum Switch gehörende Script und rufen dabei die Funktionen hitPointA(value) bzw. hitPointB(value) auf. Diese iFunktionen rufen die zum Objektmenü-Script gehörenden Funktionen *pos(), activate()* und *callback()* auf (Implementierung in *consolescript.js*). Diese positionieren das Objektmenü und aktivieren die **+** und **-** Ikonen, soweit sie sinnvoll sind. Auf der höchsten Abstraktionsebene erscheint nur die **+** Ikone, an den original Geometrien nur die **-** Ikone zum Verschmelzen. Zusätzlich merkt sich das Script des Objektmenüs, von welchem Switch-Knoten und Sensor (A oder B) der Aufruf kam. Wird nun **+** oder **-** Ikone angeklickt, wird die Funktion *clickPlus()* bzw. *clickMinus()* des Scriptes aufgerufen, das zu dem Switch gehört, der das Menü aktiviert hat. Der Programmcode ist als Beispiel für Javascript in Abbildung B.3 im Anhang aufgelistet.

Wird die Funktion *clickPlus* von der Abstraktion (Sensor A) aufgerufen, schaltet sie den Switch *SWIn* auf den original Szenegraphen um. Ein **+** Aufruf vom Original kann nicht vorkommen, da die betreffende Ikone nicht aktiviert wird. Die Funktion *clickMinus* reagiert auf einen Aufruf von der Abstraktion (Sensor A), indem sie den übergeordneten Vater-Switchknoten auf die Abstraktion umschaltet und damit die angeklickte Abstraktion mit ihren Nachbarn verschmilzt. Ein Aufruf der *clickMinus* Funktion durch das Original(Sensor B) schaltet den Switch auf die Abstraktion um, wobei der original Szenegraph ausgeblendet wird.

Zum Schluß seien noch zwei technische Details erwähnt. Zu jedem Switch wird ein eigener Scriptknoten erzeugt, um Parameter wie Zeiger auf Vaterknoten zu initialisieren. Der dazugehörende Programmcode ist in einer Datei namens *switchscript.js* gespeichert und wird von allen Scriptknoten gemeinsam verwendet. Die ursprüngliche Funktionalität der Szene soll durch die zusätzlichen Sensoren für Konsole und Objektmenü nicht behindert werden. Dabei können original Sensoren die A- und B-Sensoren überlagern. So sind zusätzlich Routen von original Sensoren zu den Switch-Scripten notwendig, um deren Funktion zu übernehmen.

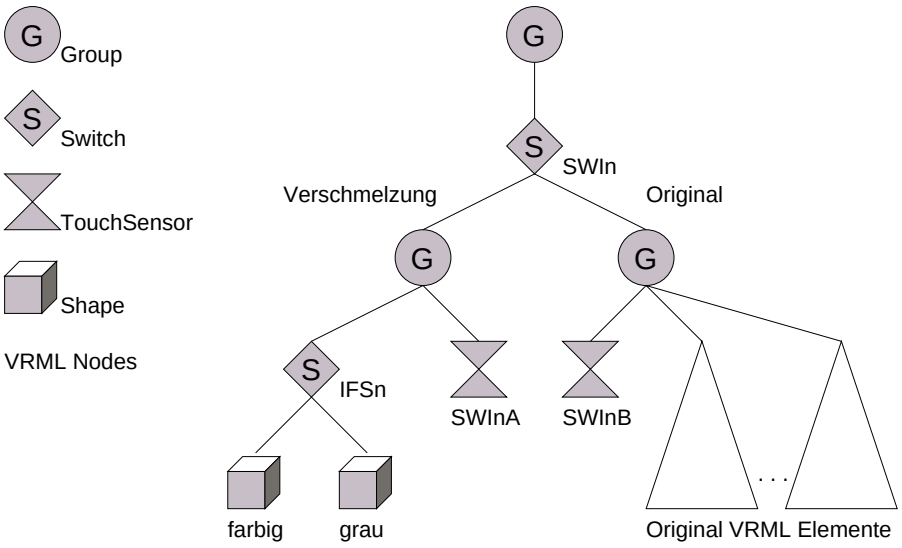


Abbildung 5.3: Szenegraph erweitert um Switch- und TouchSensor-Knoten

5.3 Primitiverkennung

Der Algorithmus zur Primitiverkennung wurde vollständig in ANSI C [KR88] programmiert. Dabei wurden der GNU C-Compiler und ein LINUX Betriebssystem verwendet. Als Hardware stand ein PC mit Intel Pentium II CPU und 128 MB Hauptspeicher zur Verfügung, der mit 333 Mhz getaktet ist. Das Programm wurde auch erfolgreich unter Solaris compiliert.

Die Programmstruktur lässt sich am besten mit einem Datenflussdiagramm entsprechend der SA[4] Technik [DeM78] darstellen. Dabei werden Datenquellen und Senken in einem Context Diagram dargestellt. Die Funktionen werden dann in Datenflussdiagrammen hierarchisch verfeinert. Das sogenannte Data Dictionary beschreibt die Verfeinerung der Datenflüsse. Die in den Diagrammen dargestellten, nicht mehr weiter verfeinerbaren Funktionen werden in sogenannten Minispecs durch Pseudocode beschrieben. Diese finden sich in Anhang A.1. Zusätzlich zu den SA Diagrammen werden die verwendeten Datenstrukturen in Anhang A.2 aufgeführt.

Abbildung 5.4 zeigt das Kontext-Diagramm, das dazugehörige Data Dictionary ist in Abbildung 5.5 dargestellt. Auf der linken Seite befindet sich die Datenquelle, dabei handelt es sich um ein Polygonmodell, repräsentiert im *OFF* Dateiformat (siehe Abschnitt 2.3.1). In der Mitte ist das Programm auf Ebene der *main(..)* Funktion symbolisiert. Die Ausgabe besteht aus einer qualitativein Beschreibung der erkannten Primitive. Dabei werden die Profile der Rotationskörper in eigene Dateien im *VECT* Format geschrieben. Dieses Dateiformat ist mit dem *OFF* Format verwandt und beschreibt 2D Polygonzüge durch eine Liste von Koordinaten. Alle Polygone der erkannten Primitive werden aus dem Eingabemodell gefiltert und wieder als Polygonmodell im *OFF* Format zurückgeliefert.

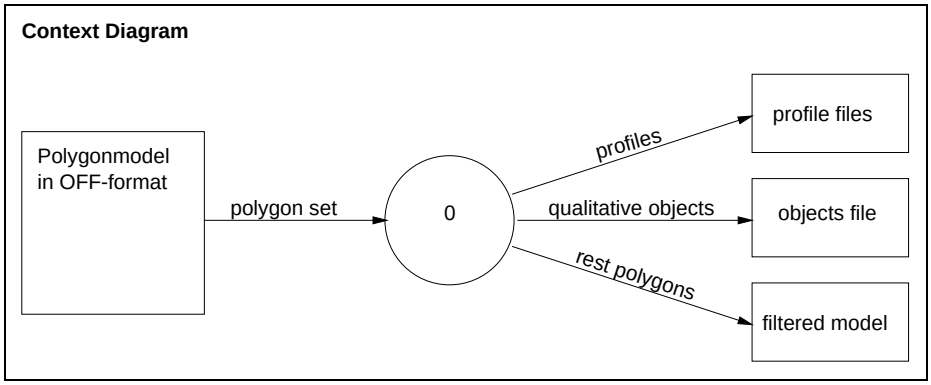


Abbildung 5.4: Context Diagram

Data Dictionary

```
polygonset := polygons + coordinates
qualitative objects := cuboids + lathes
lathes := cycle + cone + cylinder + frustum +lathe + extrusion
```

Abbildung 5.5: Das Data Dictionary zum Context Diagram

[4] Structured Analysis

Das in Abbildung 5.6 dargestellte DFD 0 verfeinert das Kontext Diagramm in die beiden Funktionen zur Quadererkennung und Rotationskörpererkennung. Die Quadersuche erfolgt zuerst, wobei alle erkannten Quader entfernt werden. Die Ausgabedaten *qualitative objects* verfeinern sich in *cuboids* und *lathes*. Es werden die Datenspeicher beider Funktionen eingeführt. Dabei werden in *Edges* die Kanten der Polygone und in *Corners* die gefundenen Ecken gespeichert. Zur Erkennung von Rotationskörpern werden in *Angles* alle Winkel zwischen Kanten und in *Cycles* gefundene Ringe gespeichert. Der Speicher *Objects* gruppiert verbundene Ringe zu Objekten. Pfeile zeigen an, in welcher Richtung die Speicher verändert werden. Die Funktionen .1 und .2 werden im folgenden durch DFD 1 (Abbildung 5.7) und DFD 1.3 (Abbildung 5.8) sowie DFD 2 (Abbildung 5.9) verfeinert dargestellt.

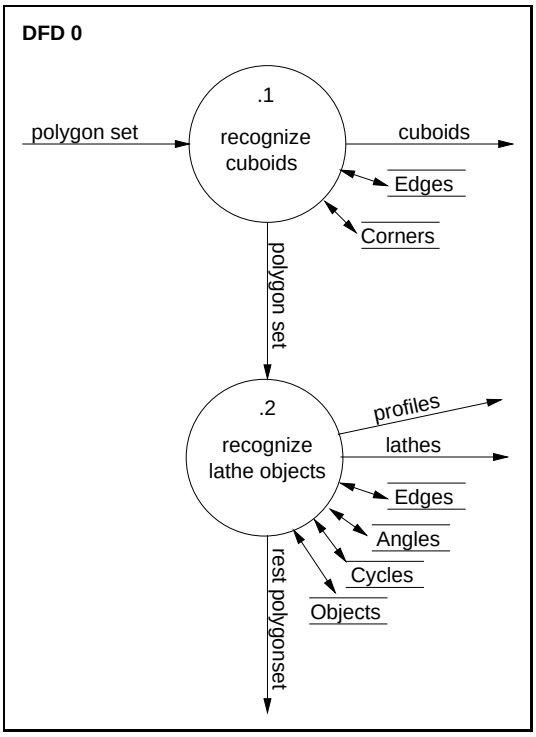


Abbildung 5.6: DFD 0

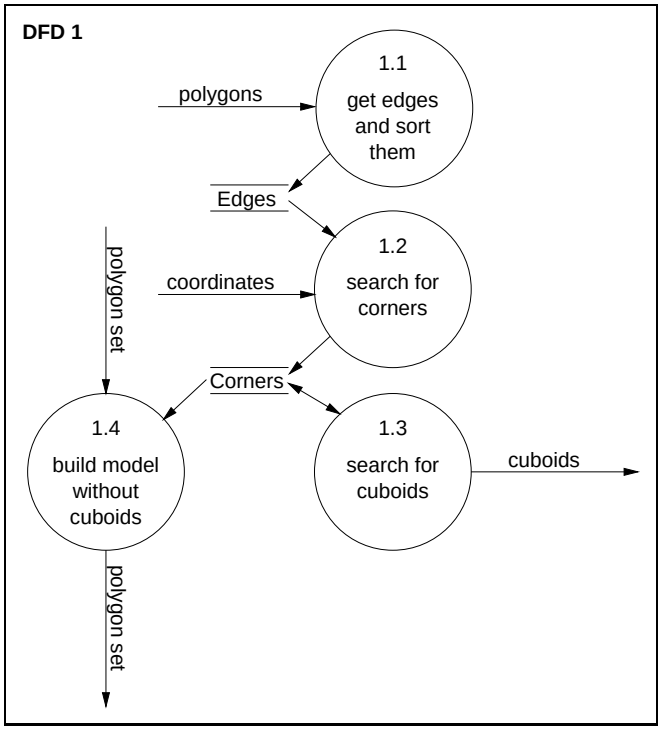


Abbildung 5.7: DFD 1

In Abbildung 5.7 wird die Funktion der Quadererkennung detailliert dargestellt. Die Eingabedaten *polygonset* teilen sich in *polygons* und *coordinates* auf. Die Vorgehensweise entspricht dem Entwurf in Abschnitt 3.3.1. Die Funktionen 1.1 und 1.2 bereiten die Daten für die Suche nach Quadern vor. Dazu werden in Funktion 1.1 die Kanten des Eingabemodells geordnet und indexiert. In Funktion 1.2 werden alle Ecken gesucht und gespeichert. Dazu werden an jedem Knoten Kombinationen von je drei Kanten geprüft. In Funktion 1.3 findet auf Basis der Ecken die eigentliche Suche nach Quadern statt, die in DFD 1.3 nocheinmal verfeinert wird. In Funktion 1.4 wird als Ausgabe eine Kopie des Eingabemodells ohne die erkannten Quader erstellt, so daß deren Polygone bei der Suche nach Rotationskörpern nicht erneut betrachtet werden müssen. Die Funktionen 1.1, 1.2 und 1.4 werden in Anhang A.1 durch die MiniSpecs 1.1, 1.2 und 1.4 in Pseudocode beschrieben.

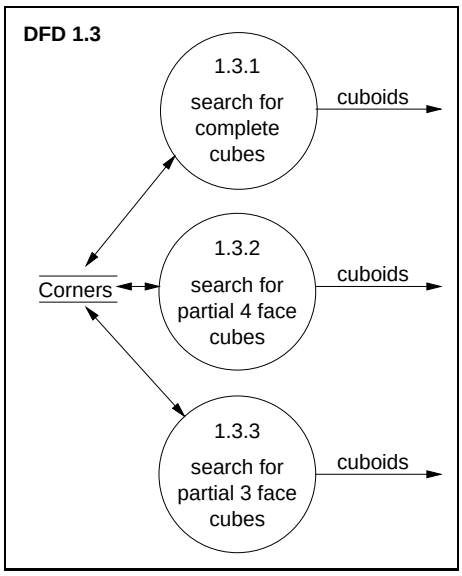


Abbildung 5.8: DFD 1.3

Abbildung 5.8 zeigt das DFD 1.3, in dem die Suche nach Quadern in drei leicht modifizierten Varianten ausgeführt wird. In Funktion 1.3.1 wird nur nach vollständigen Quadern mit 6 Seitenflächen und 8 Ecken gesucht. Diese werden symbolisch als *cuboids* ausgegeben und die verwendeten Ecken aus dem *Corners* Speicher gelöscht. Nun wird nach Teilquadern gesucht, wie sie in Abb. 3.3 auf Seite 35 dargestellt sind. Dabei wird in Funktion 1.3.2 zuerst nach 4 Seiten und anschliessend in Funktion 1.3.3 nach 3 Seiten gesucht. Durch diese Reihenfolge wird verhindert, daß ein vollständiger Quader als zwei 3-seitige Teilquader klassifiziert wird.

Nach der besprochenen Quadererkennung in Funktion .1 des DFD 0 folgt in Funktion .2 die Suche nach Rotationskörpern. Sie wird im DFD 2 in Abbildung 5.9 dargestellt und verfeinert. Die Suche entspricht dem Entwurf aus Abschnitt 3.3.2 und beginnt ähnlich der Quadererkennung. Die Funktion 2.1 gleicht der Funktion 1.1. Anschliessend werden in Funktion 2.2 die Winkel zwischen Kantenpaaren bestimmt und im Speicher *Angles* abgelegt. Funktion 2.3 enthält den Kern der Suche. Es werden gleiche zusammenhängende Winkel in einer Ebene gesucht, die einen n-eckigen Zyklus bilden. Die Zyklen werden als Indexmenge der beteiligten Knoten in *Cycles* gespeichert. In Funktion 2.4 werden die Verbindungen zwischen Zyklen geprüft. Miteinander verbundene Zyklen werden als Objekt identifiziert und in *Objects* gespeichert. Diese Objekte werden in Funktion 2.5 als Geometrische Körper klassifiziert und symbolisch ausgegeben. Als letzter Schritt werden in Funktion 2.6 die erkannten Objekte aus dem Eingabemodell gefiltert, indem eine Kopie ohne die in Zyklen verwendeten Knoten erzeugt wird. Die Funktionen 2.1 bis 2.5 werden in Anhang A.1 durch ihre entsprechenden MiniSpecs in Pseudocode beschrieben.

5.3.1 Probleme der Implementierung durch Gleitkommazahlen

Ein generelles Problem bei der Erkennung von Primitiven entsteht durch die begrenzte Präzision von Gleitkommazahlen. Zum einen sind die Koordinaten des Polygonmodells bereits gerundet und dadurch leicht zueinander verschoben, so daß die Winkel in den Ringen eines Zylinders keineswegs wirklich gleich sind. Zum anderen multiplizieren sich diese Rundungsfehler bei der Bestimmung von Winkeln und Normalenvektoren, so daß nur mittels Toleranzen beim Vergleich überhaupt Ringe gefunden werden können. Ist der Toleranzbereich zu eng, bricht die Suche nach Ringen zu früh ab, ist er zu groß, verläuft die

Suche unkontrolliert und kehrt nie zum Anfangsknoten zurück. Die gleiche Problematik gilt natürlich auch für die Suche nach Quadern. Für ein vollständiges Erkennen von Primitiven sind daher mehrere Durchläufe mit verschiedenen Toleranzen notwendig.

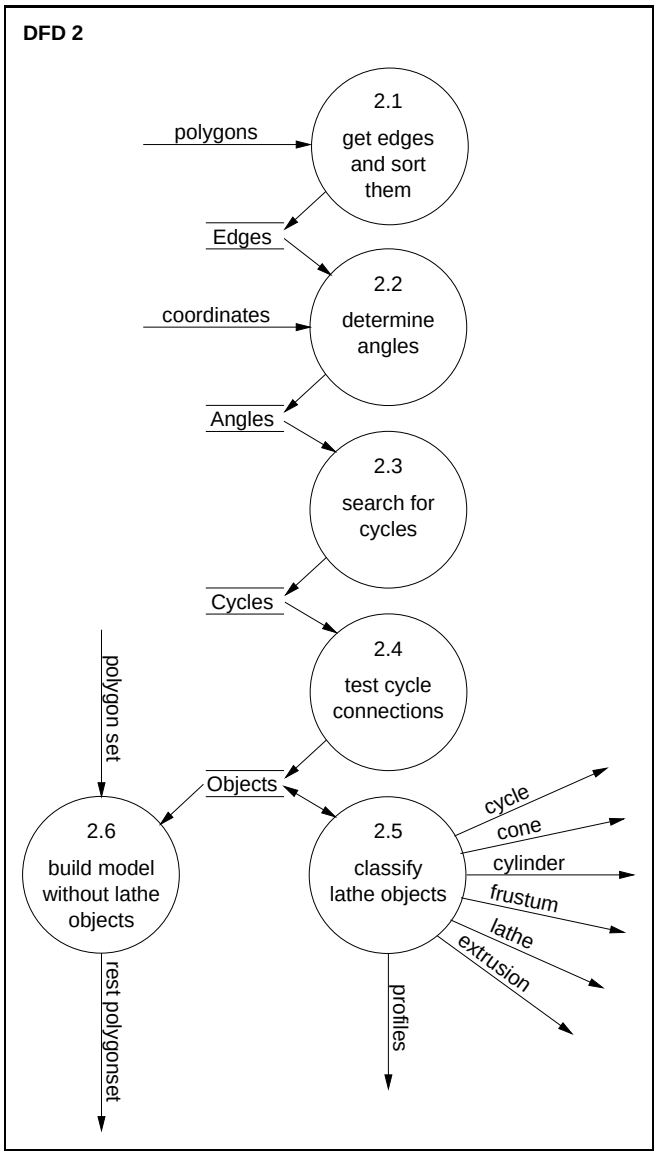


Abbildung 5.9: DFD 2

Kapitel 6

Anwendungsbeispiele

In diesem Kapitel sollen Anwendungsbeispiele für die Primitiverkennung und beide Varianten des VAI-Systems präsentiert werden. Die Beispiele der Primitiverkennung in Abschnitt 6.1 sollen die Funktionsweise des Verfahrens veranschaulichen und gleichzeitig die Anwendbarkeit auf komplexe Objekte demonstrieren. In Abschnitt 6.2 werden Beispiele zur dynamischen VAI-Version, in Abschnitt 6.3 Beispiele zur statischen VAI-Version vorgestellt. Sie sollen anhand von Bildsequenzen zeigen, wie das VAI- Hilfssystem die Navigation in VRML-Welten erleichtern kann.

6.1 Beispiele zur Primitiverkennung

Die folgenden Abschnitte zeigen Beispiele für die Anwendung der Primitiverkennung. Das erste Beispiel ist ein einfacher Rotationskörper, im zweiten Beispiel kommt die Quadererkennung hinzu.

6.1.1 Beispiel Goblet

Da das Programm noch nicht in das VAI-System integriert ist, wird es direkt von der Unixshell aus gestartet. Das einzulesende Polygonmodell wird dabei als Parameter übergeben. Im ersten Anwendungsbeispiel handelt es sich um einen Becher, der in Abbildung 6.1(a) dargestellt ist. Dieser Becher ist ein typischer Vertreter der Klasse der Rotationskörper. Diese Körper entstehen durch die Rotation eines Profils um eine Achse. Um solche Körper mit polygonbasierten Verfahren der Computergraphik zu verarbeiten, müssen sie durch Polygone approximiert werden. Dabei wird ein Kompromiss zwischen einer möglichst runden Form, die theoretisch für jeden Bildpunkt ein Polygon benötigt, und einem realisierbaren Datenvolumen geschlossen.

Das Profil des Goblet besteht aus 52 Stützpunkten, die durch Geraden verbunden sind. Es wurde 10 mal um die Längsachse gedreht und verbunden, so daß das Modell aus 520 Polygonen besteht.

Die Implementierung des in Abschnitt 3.3.2 im Detail beschriebenen Algorithmus sucht und erkennt nun im ersten Schritt die 10-Ecke. Dabei werden zu jedem 10-Eck sein Radius, Mittelpunkt und Lot berechnet und gespeichert. Ab jetzt werden die 10-Ecke als geometrisch perfekte Ringe interpretiert. Die Ringe werden nun auf ihren Zusammenhang hin überprüft und entsprechend sortiert. Daraus resultiert als Ergebnis ein zweidimensionaler offener Polygonzug, dessen Punkte Radius und Höhe der Ringe bezüglich einer gemeinsamen Achse repräsentieren. Dieser Polygonzug ist eine Abstraktion des Rotationskörpers auf Datenstrukturebene, da er das Modell viel kompakter und exakter repräsentiert. Während das Originalmodell eine runde Form durch diskrete Flächen approximiert hat, repräsentiert das Profil einen perfekten Rotationskörper.

Das Programm kann den Polygonzug nun in verschiedenen Formaten in eine Datei ausgeben. Für Abbildung 6.1(b) wurde eine OOGL VECT-Datei erzeugt, die mit dem Programm Geomview [Geo96a] dargestellt werden kann. Von dort aus wurde es als Postscript Datei exportiert und in Latex eingebunden.

Da das Profil den Becher mathematisch exakter als die Annäherung durch Polygone beschreibt, liegt es nahe, diese Beschreibung anstelle von Polygonen zur Visualisierung zu verwenden.

Zu diesem Zweck wurde als alternatives Ausgabeformat die Szenenbeschreibungssprache des frei erhältlichen Raytracers *PovRay* [PT99] implementiert. Diese Software ist in der Lage, nicht nur Polygone, sondern auch Geometrische Primitive anhand ihrer Parameter wie Radien und Orientierung darzustellen. Es entstehen keine Fehler durch eine Approximation mittels endlich vieler Polygone, dafür ist der Zeitaufwand zur Bilderzeugung um Grössenordnungen höher als bei polygonbasierten Verfahren. In PoyRay wird der Becher als sogenanntes Lathe[1] Objekt mit kubischer Spline-Interpolation repräsentiert. Die Interpolation glättet das Profil, so daß die Oberfläche des Rotationskörpers keine Unstetigkeiten an den Stützpunkten aufweist. So werden realistische Spiegelungen möglich, wie sie in Abbildung 6.1(c) zu sehen sind.

6.1.2 Beispiel Capitol

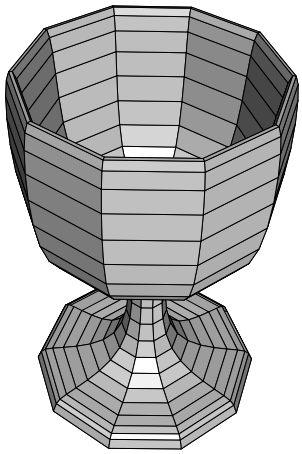
In diesem Anwendungsbeispiel sollen neben Rotationskörpern auch Quader gefunden werden. Als Demonstrationsobjekt wird ein recht detailliertes Polygonmodell des Capitols in Washington verwendet, das aus einer 3D-Modellbibliothek stammt [DCM98a]. Die Gebäudearchitektur basiert im wesentlichen auf regelmässigen geometrischen Primitiven, so daß auch das Polygonmodell aus diesen Grundelementen aufgebaut wurde. Somit eignete es sich hervorragend als Testobjekt während der Implementierung des Programms.

Nach dem Aufruf des Programms erfolgt zuerst die Quaderanalyse. Der zugrundeliegende Algorithmus wurde bereits in Abschnitt 3.3.1 beschrieben. Es werden zuerst vollständige, dann auch Teilquader gesucht. Gefundene Quader werden aus dem Polygonmodell herausgefiltert und in eine eigene Datei ausgegeben, dabei werden Teilquader vervollständigt. Das Ergebnis der Suche ist in Abbildung 6.3(b) zu sehen. Zur Veranschaulichung wurden die Quader zufällig eingefärbt, um benachbarte Flächen unterscheiden zu können. Bei genauer Betrachtung der Quader fallen einige Unregelmässigkeiten auf. Es fehlen die oberen Treppenstufen. Das liegt daran, daß die oberen Stufen nur durch ihre zwei sichtbaren Aussenseiten modelliert wurden. Als Teilquader werden jedoch nur Objekte mit mindestens 3 Flächen akzeptiert. Die unteren Stufen sind als vollständige Quader modelliert worden. Ein weiterer Modellierungsfehler findet sich rechts der Treppe des Haupteingangs. Auch dort fehlt im Originalmodell eine sichtbare Fläche. Derartige Fehler stecken in den meisten Polygonmodellen. Sie fallen jedoch in der Regel nicht auf, da die Polygone ohnehin nie in typischen Perspektiven sichtbar sind oder im Gesamtbild nur wenige Pixel ausmachen und vom Betrachter nicht wahrgenommen werden. Erst eine Analyse der Struktur macht solche Fehler deutlich.

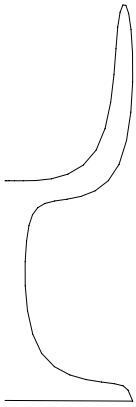
Die zu Quadern gehörenden Polygone werden aus dem Modell herausgefiltert und damit der Suchraum für Zylinder und Rotationskörper stark verkleinert. Der Suchvorgang selbst erfolgt wie schon im ersten Beispiel beschrieben. Genau wie die Suche nach Quadern ist auch die Suche nach Rotationskörpern als Filter implementiert. Die gefundenen Zylinder und Profile werden in eine eigene Datei ausgegeben und sind in Abbildung 6.3(c) dargestellt. Die übrigen Polygone werden getrennt davon ausgegeben und sind in Abbildung 6.4(a) dargestellt. Die Abbildungen 6.3(b) und 6.3(c) dienen jedoch nur der Veranschaulichung des Filtervorgangs. Das eigentlich interessante Ergebnis der Suche ist die qualitative Beschreibung der gefundenen Primitive.

Wie schon in Abschnit 6.1.1 erwähnt, können die Primitive beispielsweise in der Syntax

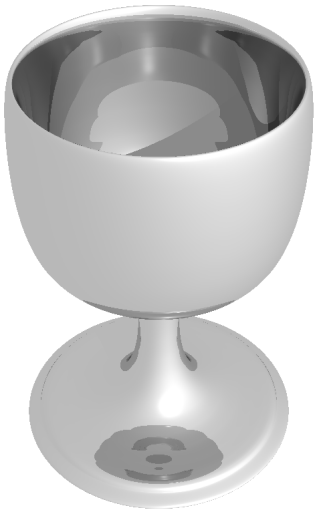
[1] Englisch für Drehbank



(a) Polygonmodell goblet.off



(b) Erkanntes Profil



(c) Darstellung mit Raytracer Povray

Abbildung 6.1: Erkennen und Rendern des Eingabemodells goblet.off, Modellquelle: Geomview[Geo96a]

```
box {
    <1.150000,-0.058000,-0.280000>,
    <1.280000,0.000000,-0.650000>
}
box {
    <0.860000,-0.058000,-0.750000>,
    <0.350000,0.000000,-0.970000>
}
cylinder {
    <0.671299,0.005000,-1.000887>,
    <0.671299,0.095000,-1.000887>,
    0.003999
}
cylinder {
    <0.681299,0.005000,-1.000887>,
    <0.681299,0.095000,-1.000887>,
    0.003999
}
```

(a) Beschreibung der Quader und Zylinder

```
lathe {
    8,
    <0.000000,0.001607>
    <0.034865,0.000000>
    <0.072891,-0.010679>
    <0.098540,-0.026498>
    <0.114999,-0.048393>
    <0.098540,-0.026498>
    <0.072891,-0.010679>
    <0.034865,0.000000>
    rotate <0,0,0>
    translate <1.005,0.163593,-0.55>
}
```

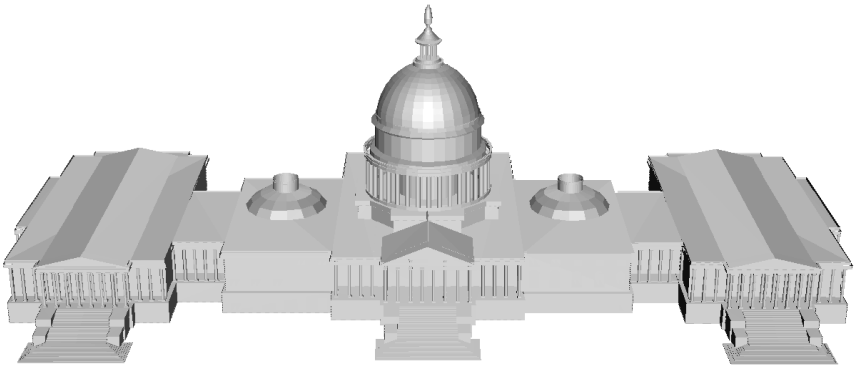
(b) Beschreibung einer Kuppel des Capitols

Abbildung 6.2: Für *PovRay* erzeugte Szenenbeschreibung

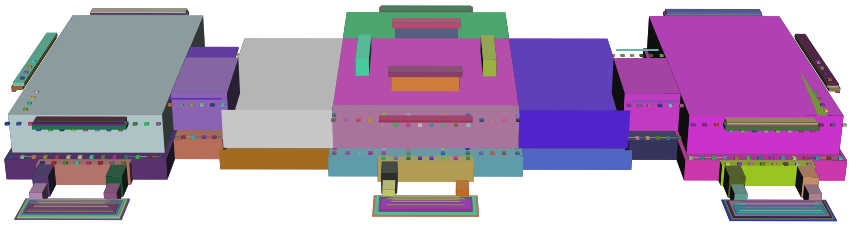
von PovRay ausgegeben werden, wie in Abbildung 6.2(a) zu sehen. Quader werden durch zwei gegenüberliegende Eckpunkte repräsentiert. Zylinder werden über die Endpunkte ihrer Achse und ihren Radius definiert. Analog können auch Kegel dargestellt werden. Für jeden komplexen Rotationskörper wird das Profil in einer eigenen Datei gespeichert. Zum Capitol gehören 4 Profile: Die Haupt- und Seitenkuppeln und die Spitze. Ein Beispiel ist in Listing 6.2(b) zu sehen. Das Profil wird durch Punktkoordinaten spezifiziert und in die richtige Postition gebracht und gedreht.

Ein mit Povray berechnetes Bild des Capitols ist in Abbildung 6.4(b) zu sehen, erkannte Primitive sind mit einer Marmortextur dargestellt.

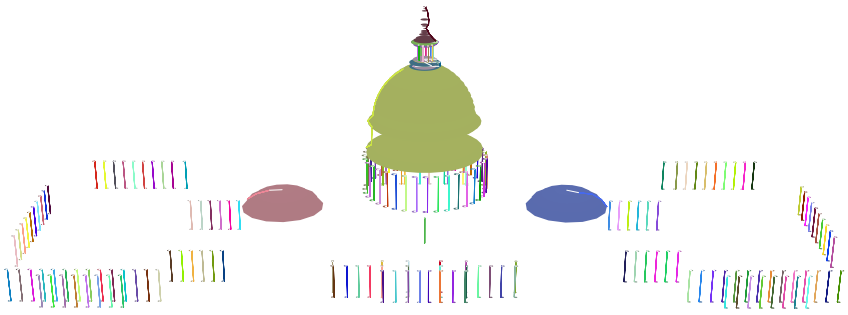
Grundlage der Suche nach Primitiven sind Kriterien wie Rechte Winkel, gleiche Winkel und Längen sowie parallele Vektoren. Die Berechnung dieser Maße erfolgt aus den Punktkoordinaten der Polygone nach einfachen Formeln der linearen Algebra. Jedoch ist die Genauigkeit der Koordinaten durch ihre interne Repräsentation als Gleitkommazahlen begrenzt, und mit jedem Rechenschritt summieren sich die Rundungsfehler. Daher führt zum Beispiel ein einfacher Vergleich zwischen zwei berechneten Vektoren, die mathematisch gesehen exakt gleich sind, in der Praxis nur selten zum wahren Ergebnis. Dieses Problem ließe sich mit exakter Arithmetik oder der Berechnung von oberen und unteren Schranken lösen, der Implementierungs- und Rechenaufwand steigt dann jedoch beträchtlich. Als praktikable Lösung wird bei jedem Vergleich eine gewisse Fehlertoleranz akzeptiert, die in der Größenordnung der Rundungsfehler liegen sollte. Ist sie zu klein gewählt, werden Primitive nicht als solche erkannt. Ist sie zu groß, verläuft die Suche in die falsche Richtung und bricht ab. Je nach Polygonmodell und Maßstab muß mit den Toleranzen experimentiert werden, um möglichst viele Primitive zu finden. Dazu kann die Toleranzgrenze für Längen und Winkel über Parameter beim Programmaufruf eingestellt werden. Eine Automatisierung wäre denkbar, indem die Toleranzen kontinuierlich verändert werden, um die Menge der erkannten Primitive zu maximieren.



(a) Polygonmodell capitol.off

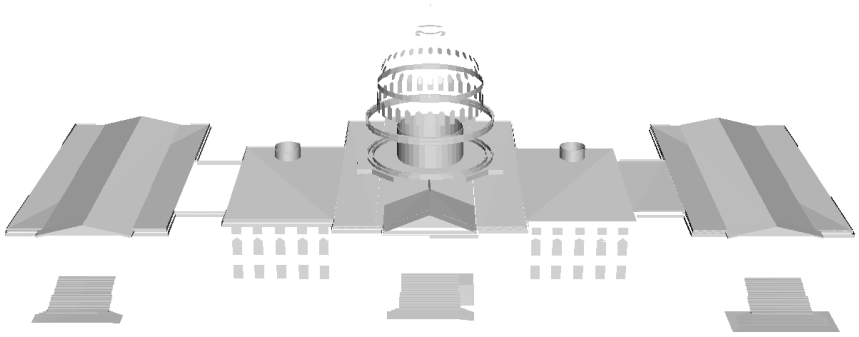


(b) Erkannte Quader

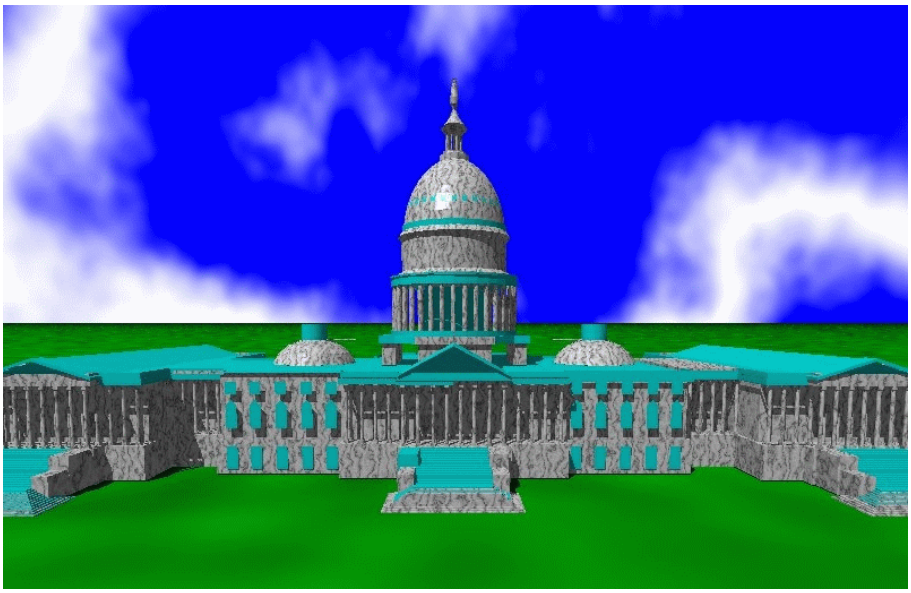


(c) Erkannte Rotationskörper

Abbildung 6.3: Zerlegung des Capitol-Polygonmodells [DCM98a] in Primitive



(a) Übrige Polygone



(b) Erkannte Primitive sind mit Textur versehen

Abbildung 6.4: Darstellung mit Povray

6.2 Navigation mit dynamischer VAI Variante

In diesem Abschnitt gibt es zwei Anwendungsbeispiele zur dynamischen Version des VRML Abstraction Interface. Das erste Modell ist das Capitol, das zweite ein Videorecorder. Anhand des komplexen Modells des Capitols soll gezeigt werden, wie durch Auswahl von Abstraktionsgraden die zur Visualisierung benötigten Rechenressourcen geschont und die Navigation beschleunigt werden kann. Das Modell des Videorecorders soll verdeutlichen, wie die Aufmerksamkeit des Betrachters mittels Abstraktion auf bestimmte Tastengruppen fokussiert werden kann.

6.2.1 Beispiel Capitol

Die Abbildung 6.5 zeigt die typische Umgebung des Interface. In dem HTML-Browser *Netscape Navigator* wird eine Seite geöffnet, die sowohl die Initiale VRML-Welt mit der Konsole als auch das JAVA-Applet *V.A.I.* enthält. Die VRML-Welt wird durch das Plugin *SGI Cosmoplayer* dargestellt. Über die Javascript- Schnittstelle des Browsers erhält das Applet Zugang zum Plugin. So kann der *Cosmoplayer* über die EAI-Schnittstelle veranlasst werden, Objekte wie das Capitol [DCM98a] zu laden. Umgekehrt wird das Applet über alle Zeigeraktionen in der VRML-Szene informiert, um auf die Ikonen der Konsole zu reagieren.

Die Navigation beginnt auf der abstraktesten Ebene, in der das gesamte Gebäude als Einheit dargestellt ist. Siehe dazu die 2. Ebene in Abbildung 6.6. In dieser Ansicht ist die der Renderingpipeline zugrundeliegende Datenmenge so gering wie möglich. So wird bei der Navigation die höchste Bildschirmrate erzielt, und der Benutzer kann sich problemlos ohne Verzögerungen um das Capitol herum bewegen. Dieser Ansatz ist besonders geeignet für ein Szenario, in dem der Benutzer bereits weiss, daß das Capitol Gegenstand der Navigation ist und eine ungefähre Vorstellung von seinem Ziel hat. Andernfalls wäre es sinnvoller, statt auf dem höchsten Abstraktionsgrad mit der Originaldarstellung zu beginnen. Diese entspricht der obersten Ebene in Abbildung 6.6.

Der Benutzer hat nun die Möglichkeit, mit Hilfe der Konsole oder dem Menü des Applets schrittweise Gebäudeteile zu verfeinern. Das entsprechende Segment wird mit der Maus selektiert, anschließend die **+** Ikone gedrückt. Nun wird auf die nächste Abstraktionsebene geschaltet und der gewählte Gebäudeabschnitt feiner gegliedert. Auf diese Weise gelangt man letztlich zur Originalansicht bestimmter Gebäudeteile, während andere Abschnitte als abstrakte Einheit dargestellt werden. Die **-** Ikone führt vom Detail zurück zur Abstraktion. Die Vorgehensweise ist analog zur Verfeinerung.

6.2.2 Beispiel Videorecorder

Dieses Beispiel bezieht sich auf das Szenario einer Bedienungsanleitung für einen Videorecorder [Krü00]. Dort werden mithilfe des Abstraktionssystems ARP automatisch Ansichten des Videorecorders erzeugt, in denen die Aufmerksamkeit des Betrachters mittels graphischer Abstraktion gezielt auf wichtige Tasten gelenkt wird. In einer interaktiven Umgebung wie VRML könnte der Benutzer mittels einer Konsole den vorgegebenen Abstraktionsgrad verändern und frei navigieren.

Die Bildsequenz 6.7 zeigt das Gerät in verschiedenen Abstraktionsgraden. Das erste Bild 6.7(a) zeigt den Anfangszustand. Dort sind alle Bedienungselemente mit dem Gehäuse verschmolzen. Das Gerät wird durch Mausklick selektiert und die **+** Ikone der Konsole gedrückt. So werden die einzelnen Tastengruppen sichtbar, wie in Bild 6.7(b) gezeigt. Nun wird die Haupttastengruppe (blau hervorgehoben) selektiert und in Abbildung 6.7(c) detailliert dargestellt. Diese Darstellung könnte beispielsweise dem Benutzer vorgegeben werden, um die Wiedergabe-Taste zu lokalisieren.

Abbildung 6.7(d) zeigt zum Vergleich das Originalmodell ohne Abstraktion.

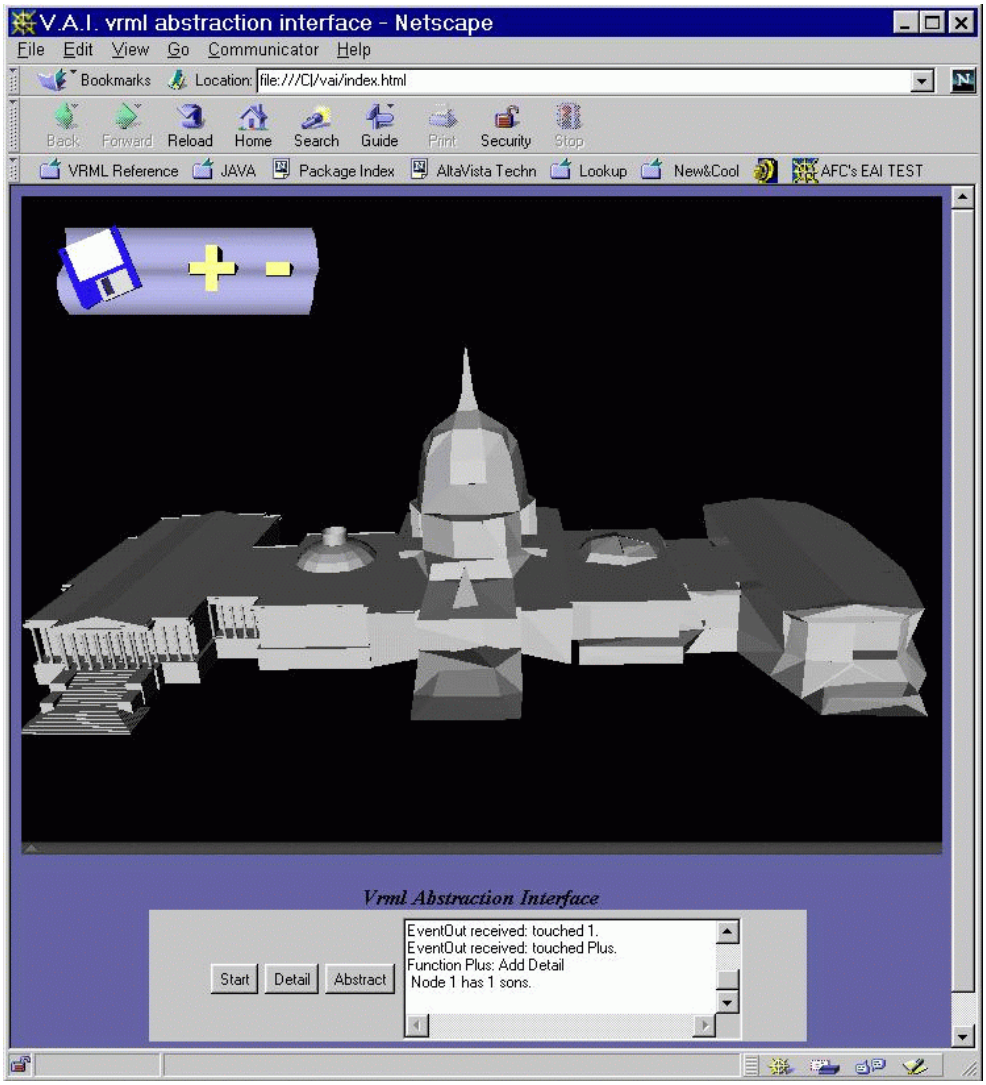


Abbildung 6.5: HTML-Browser mit VRML-Plugin und JAVA-Applet

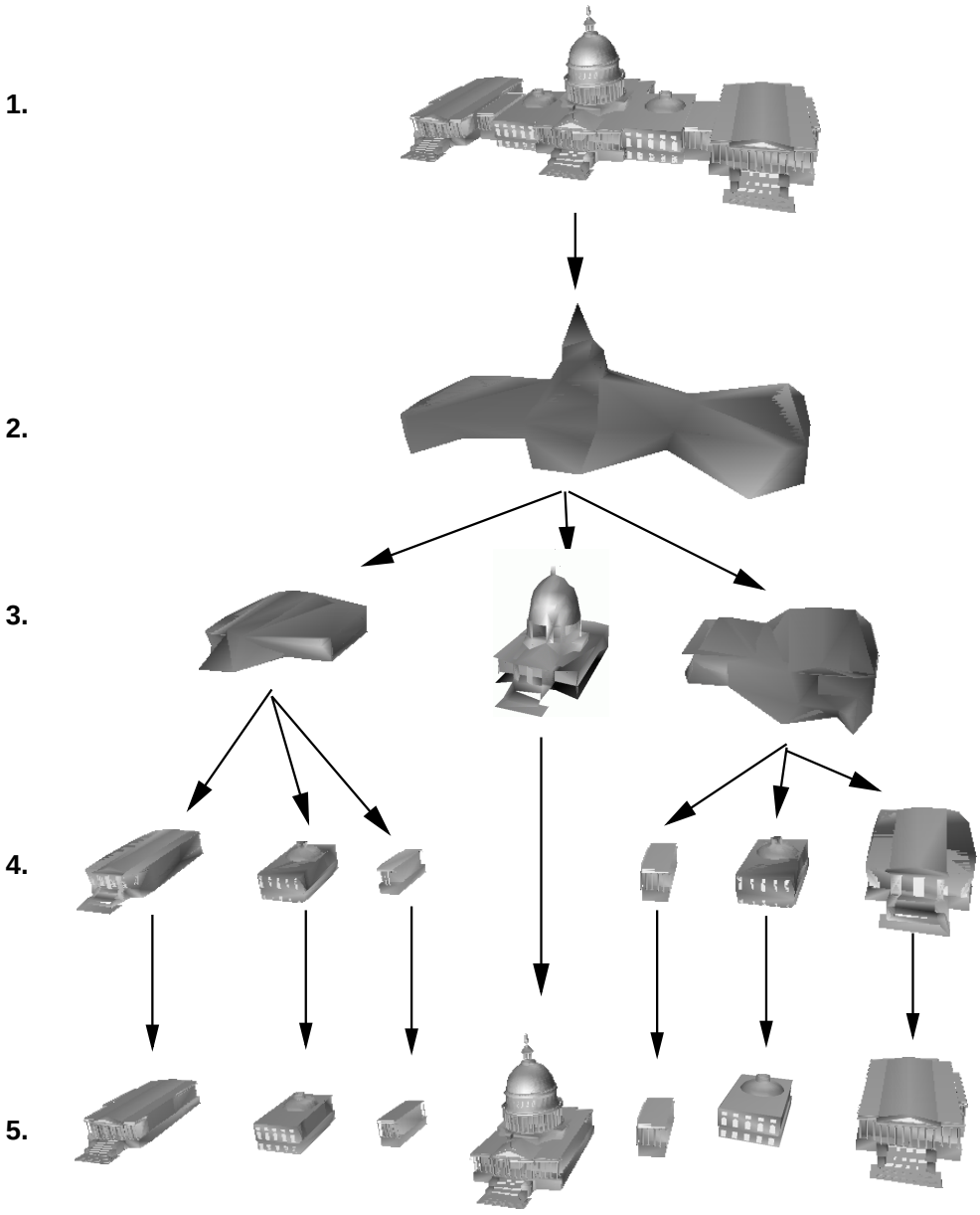
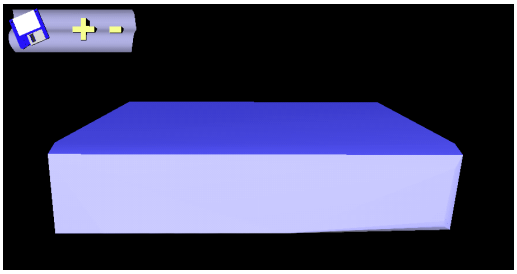
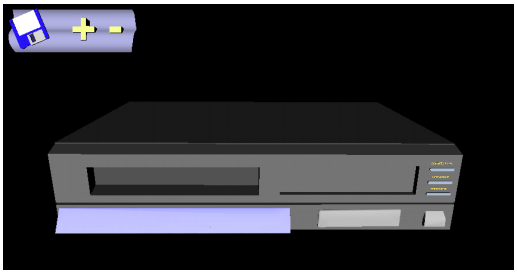


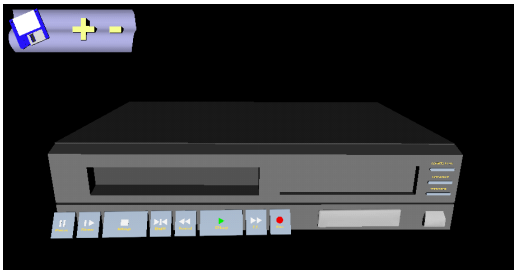
Abbildung 6.6: verschiedene Abstraktionsgrade gemäß der Objekthierarchie des Capitol-Modells [DCM98a]



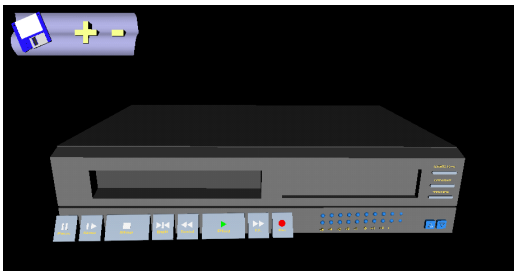
(a) Anfang



(b) Hinzufügen von Details



(c) Detaildarstellung der Haupttastengruppe



(d) Original

Abbildung 6.7: Videorecorder-Ansichten in V.A.I., Modell Quelle: [DCM98b]

6.3 Arbeiten mit statischer VAI Variante

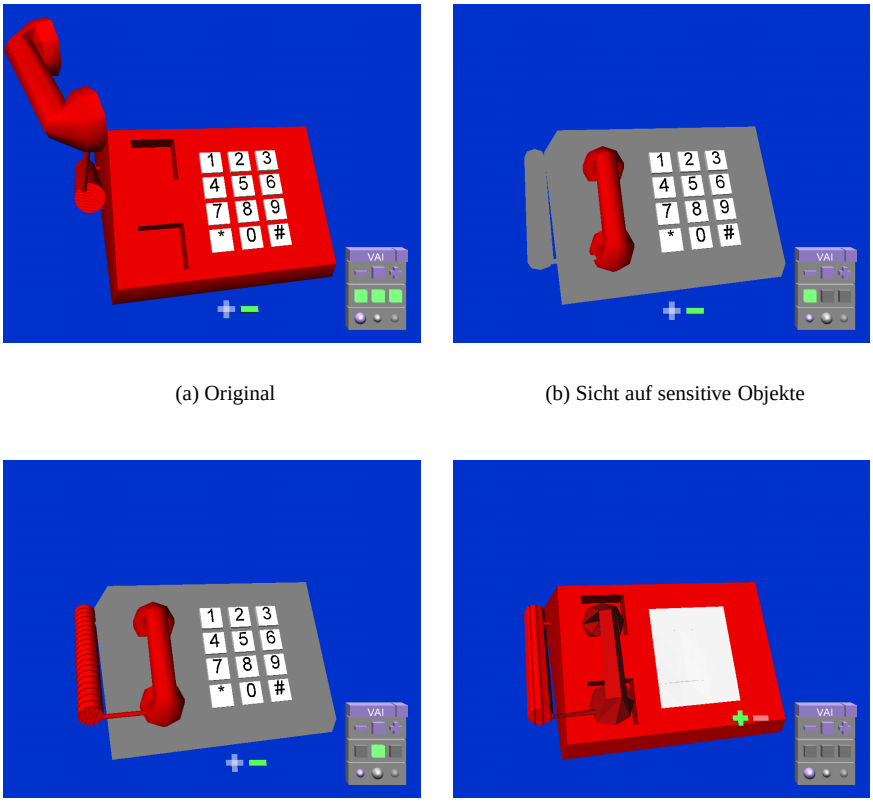
Die statische Variante des VRML Abstraction Interface wurde in Java realisiert. Bisher ist noch keine graphische Benutzeroberfläche zur Systemsteuerung implementiert worden, daher müssen VAI und ARP von Hand aufgerufen werden. Eine Automatisierung der Vorgänge durch eine Batchdatei wäre technisch durchaus möglich.

Das Programm wird von der Unixshell aus gestartet, dabei wird der Name der zu abstrahierenden VRML-Welt als Parameter übergeben. Jetzt werden der Szenegraph geparst und die Strukturinformation für das Abstraktionssystem ARP als Datei in Form einer geklammerten Liste herausgeschrieben. Nun wird ARP im LISP-Interpreter gestartet, wobei Parameter für die Verschmelzungsoperation übergeben werden können. Diese geben Grenzwerte für Überschneidung und Farbunterschied der zu verschmelzenden Elemente vor. ARP erzeugt alle Abstraktionen und erlaubten Verschmelzungsoperationen. Im Anschluß wird erneut VAI gestartet, diesmal findet es die von ARP gelieferten Dateien vor und erzeugt aus ihnen eine VRML-Welt, die alle Abstraktionen und die Konsole zur Steuerung enthält.

Es folgen drei Beispiele (Telephon, Büro, Celiafish), die von VAI nach der oben beschriebenen Vorgehensweise automatisch erzeugt worden sind. Lediglich einige Details wurden von Hand optimiert. Zum einen konnte das Ergebnis der Verschmelzungsoperation durch geschicktes Umgruppieren der Geometrieelemente verbessert werden. Zum anderen musste die Konsole entsprechend dem Maßstab des jeweiligen Modells skaliert werden. Dies ist nicht ohne weiteres durch ein festes Verhältnis zur Bounding Box der Szene zu realisieren. Im Fall des Telephons ändert sich die Bounding Box der Szene dramatisch, falls es sich in einem Gebäude befindet. Die optimale Größe der Konsole bleibt jedoch gleich. Das Problem würde sich nicht ergeben, wenn alle VRML Modelle, wie empfohlen, in einem der Realität entsprechenden Maßstab modelliert wären. Dann würde die Konsole einfach der Körpergröße des Avatars angepasst.

6.3.1 Beispiel Telephon

Das erste Anwendungsbeispiel für das VRML Abstraction Interface ist ein in VRML modelliertes Telephon mit Hörer und Tastenblock, das in 6.8(a) abgebildet ist und aus der VRML-Repository [Con01] Sammlung von Beispielmodellen entnommen ist. Das Modell bietet dem Betrachter verschiedene Möglichkeiten zur Interaktion. Der Hörer kann durch Anklicken abgenommen und wieder aufgelegt werden. Die Zifferntasten können ebenfalls durch Anklicken gedrückt werden. Diese Funktionalität ist im Modell mit den typischen VRML Konzepten durch Routen von Sensoren zu Animationsknoten realisiert. Im folgenden Beispiel wird gezeigt, wie diese Funktionen mit Hilfe graphischer Abstraktion visualisiert werden können. Dazu wird die gewünschte funktionale Sicht über die Konsole abgerufen. Für eine detaillierte Ansicht der Konsole siehe Abbildung 4.8. Zuerst sollen die Sensorfunktionen gezeigt werden. Dazu aktiviert man im mittleren Abschnitt der Konsole die linke Ikone und deaktiviert die übrigen beiden. Da die schlichte Geometrie des Apparates kaum Angriffsfläche zur Abstraktion auf der Modellebene bietet, soll zusätzlich auf der Generierungsebene durch Graufärbung abstrahiert werden. Dazu wird die mittlere der drei Ikonen im unteren Abschnitt der Konsole ausgewählt. Nun wird die so konfigurierte funktionale Sicht über die mittlere Ikone des oberen Abschnitts aktiviert. Das Ergebnis ist in Abbildung 6.8(b) zu sehen. Der Hörer und die Tasten heben sich deutlich von dem Apparat und Kabel ab. Dabei sind auch in dem abstrakten VRML-Modell noch alle Interaktionsmöglichkeiten vorhanden. Im zweiten Schritt sollen die animierten Elemente des Modells hervorgehoben werden. Dazu wird im mittleren Abschnitt der Konsole die linke Ikone deaktiviert und die mittlere Ikone aktiviert. Die daraus resultierende funktionale Sicht ist in Abbildung 6.8(c) dargestellt. Der Unterschied zwischen beiden Sichten liegt im Kabel, das den Hörer mit dem Apparat verbindet. Das Kabel kann zwar nicht angeklickt werden, wird aber in der Animationssequenz des Hörers mitbewegt.



(a) Original

(b) Sicht auf sensitive Objekte

(c) Sicht auf animierte Objekte

(d) Abstraktion

Abbildung 6.8: VRML-Modell phone.wrl, Quelle: [Con01]

Über die **-** Ikone links im oberen Abschnitt der Konsole kann das gesamte Telephon abstrahiert werden, wie in Abbildung 6.8(d) gezeigt. Dabei wird die Geometrie des Hörers abstrahiert, und die Zifferntasten werden zu einem Block verschmolzen. In der Abstraktion geht mit der Identität auch die Funktion der einzelnen Tasten verloren; sie können nicht mehr angeklickt und gedrückt werden.

6.3.2 Beispiel Büro

Das Büro-Modell, das ebenfalls aus den VRML-Repository [Con01] Beispielen stammt, bietet ein weiteres Muster für funktionale Sichten. Es handelt sich dabei um eine typische Virtual Reality Welt, in welcher der Betrachter als Avatar navigieren und interagieren kann. Das hier in Abbildung 6.9 gezeigte Modell ist mit nur einem Raum sehr übersichtlich gehalten und bietet einfache Funktionen. Die Schreibtischlampen können über den Lichtschalter neben der Tür ein- und ausgeschaltet werden (vergleiche Abbildungen 6.9 und 6.10). Der PC auf dem mittleren Schreibtisch stellt Informationen zur Verfügung. Die Geometrie des PC wurde dazu mit einem Hyperlink auf ein HTML-Dokument verknüpft. Klickt der Betrachter auf den PC, öffnet der VRML-Browser die entsprechende Webseite. Diese Funktionalität kann man der Szene nicht unmittelbar ansehen; der Betrachter kann sie nur durch Ausprobieren herausfinden. Deshalb soll die Funktionalität nun mithilfe von graphischer Abstraktion visualisiert werden. Dazu wird in der Konsole über die **F** Ikone eine funktionale Sicht auf die Szene aktiviert. Um diese besonders deutlich zu machen, soll auf der Generierungsebene zusätzlich mittels Ausblenden abstrahiert werden. Dazu wird die rechte, transparent dargestellte Kugel in der unteren Ebene der Konsole gewählt. Über die mittlere Ebene der Konsole kann nun auf verschiedene Funktionalitäten fokussiert werden. In Abbildung 6.11(a) wird der Fokus auf sensitive Objekte gerichtet, im Büro ist dies der Lichtschalter. Setzt man nun den Fokus auf die gesteuerten Objekte, werden in Abbildung 6.11(b) die beiden Lampen sichtbar. Zuletzt sollen in Abbildung 6.11(c) Hyperlinks hervorgehoben werden, dabei wird der Computer fokussiert. Für die Navigation ist die Abstraktion mittels Ausblenden natürlich kaum geeignet, da man sprichwörtlich im Dunkeln, oder genauer im leeren Raum, steht. Daher soll die Abstraktion auf Generierungsebene zurückgenommen werden. Die Abbildungen 6.12(a) und 6.12(b) entsprechen den Abbildungen 6.11(b) und 6.11(c), setzen zum Hervorheben jedoch nur eine Abstraktion auf Modellebene ein. So kann sich der Betrachter uneingeschränkt in der Szene bewegen und weiterhin auch solche Objekte manipulieren, die nicht im Fokus der funktionalen Sicht liegen. In Abbildung 6.11(d) ist kein Fokus gesetzt, sowohl der PC als auch die Schreibtischlampen sind abstrakt dargestellt.

Abbildung 6.9: VRML-Modell office.wrl, Quelle: [Con01]

Abbildung 6.10: Schreibtischlampen eingeschaltet

(a) Sicht auf Sensitive Objekte: Lichtschalter

(b) Sicht auf gesteuerte Objekte: Lampen

(c) Sicht auf Hyperlinks: Computer

(d) Abstraktion

Abbildung 6.11: Verschiedene Funktionale Sichten mit Abstraktion auf Generierungsebene

(a) Fokus auf gesteuerte Objekte: Lampen

(b) Fokus auf Hyperlinks: Computer

Abbildung 6.12: Funktionale Sichten des Büros mit Fokussierung durch Abstraktion auf Modellebene

6.3.3 Beispiel Celiafish

Das dritte Anwendungsbeispiel ist das VRML-Modell Celiafish [You97], das von dem Künstler C. Scott Young, alias *The VRML Knight*, zur Demonstration der Fähigkeiten von VRML geschaffen wurde. Die Schwanz- und Seitenflossen sind animiert, die Rippen erzeugen Klänge beim Anklicken. Das unveränderte Modell ist in Abbildung 6.13 dargestellt. In diesem Beispiel soll anhand einer Bildsequenz gezeigt werden, wie über das Objektmenü Bestandteile des Modells interaktiv miteinander verschmolzen werden können. (Dieser Vorgang wird auch in Abschnitt 4.3.2.2 beschrieben.) Bewegt der Benutzer den Mauszeiger über das VAI-erweiterte Modell, erscheint am jeweiligen Berührungspunkt sowohl eine **+** als auch eine **-** Ikone, wie in Abbildung 6.14(a) über dem Kopf des Celiafish zu sehen. Dabei ist im Original keine höhere Detaillierung vorhanden und daher die **+** Ikone deaktiviert und transparent dargestellt. Durch Mausklick auf die **-** Ikone wird nun die zuletzt berührte Rippe mit der benachbarten Rippe verschmolzen. Das Ergebnis dieser Operation ist in Abbildung 6.14(b) zu sehen. Wird nun die verschmolzene Rippe berührt, erscheinen dort wieder die beiden Ikonen. Nun ist auch die **+** Ikone aktiviert, über sie kann die Verschmelzung rückgängig gemacht werden. Im Beispiel soll jedoch weiter verschmolzen werden, wie in Abbildung 6.15(a) zu sehen ist. Dazu sind alle Möglichkeiten im Rahmen der Vorberechnung durch das Abstraktionssystem ARP bestimmt worden. Dabei wurde nach den Kriterien Nachbarschaft und Farbähnlichkeit ausgewählt. Im Fallbeispiel Celiafish werden die inzwischen vollständig verbundenen Rippen mit dem Kopf verschmolzen, das Resultat ist in Abbildung 6.15(b) dargestellt. In einem weiteren Schritt wird die dem Betrachter zugewandte Flosse verschmolzen. Dabei wird die Originalgeometrie durch die Abstraktion ersetzt und als Konsequenz stoppt die Animation, das in Abbildung 6.15(c) gezeigte Modell erstarrt nun teilweise. Die übrigen Flossen bewegen sich jedoch noch. In den folgenden Abbildungen 6.15(d) und 6.15(e) wird die Schwanzflosse in mehreren Stufen verschmolzen, bis sie letztendlich mit der Rückenflosse verwächst. Nun ist in der durch paarweises Verschmelzen entstandenen Baumstruktur die Wurzel erreicht und das Beispiel mit Abbildung 6.15(f) beendet. Über die **+** Ikone lassen sich die Blätter des Baumes jederzeit wieder expandieren und alle Stadien in umgekehrter Reihenfolge bis hin zum Original durchlaufen.

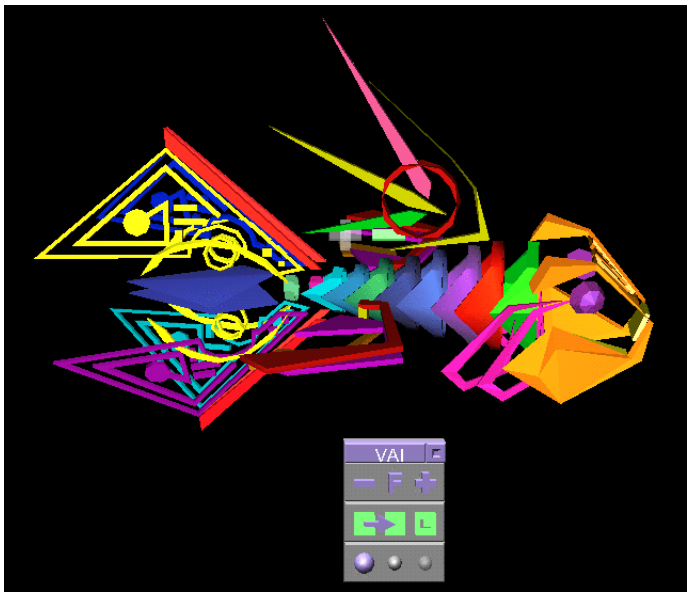
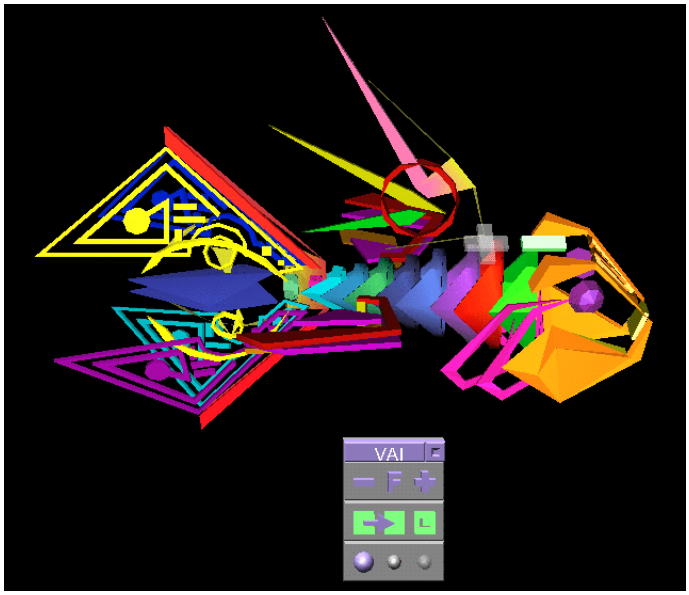
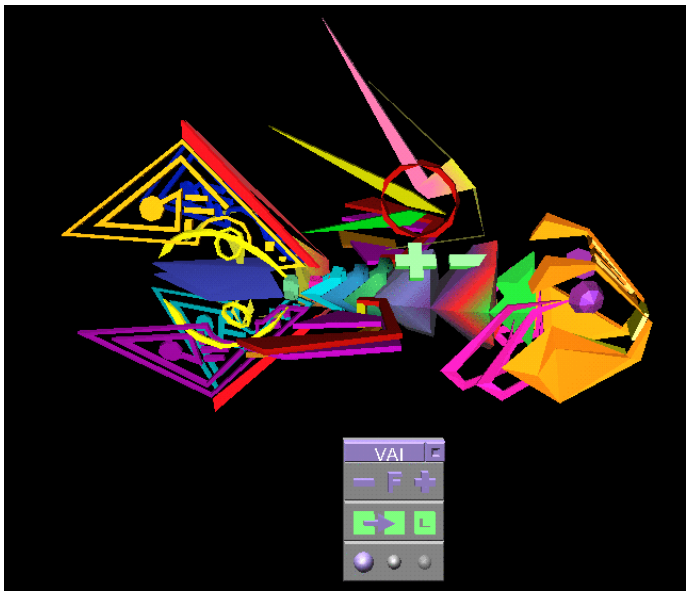


Abbildung 6.13: VRML-Modell Celiafish, Quelle: [You97]

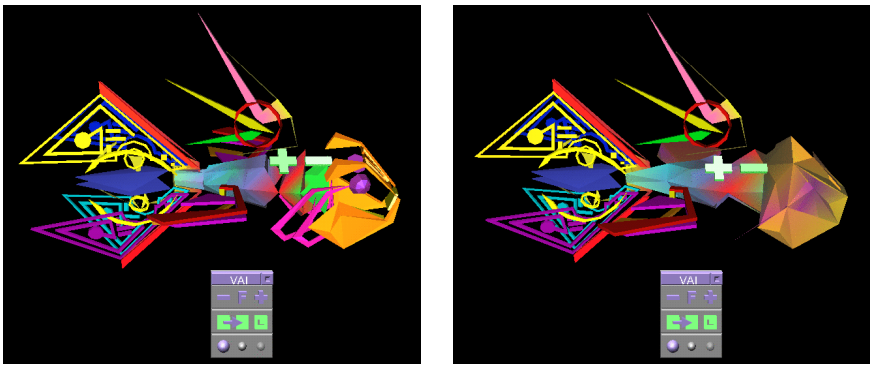


(a) Verschmelzen von Rippen



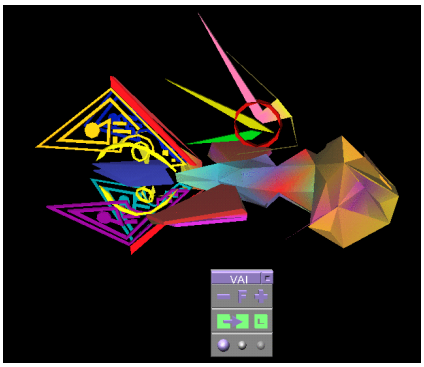
(b) Rippen fortgesetzt

Abbildung 6.14: Interaktives Verschmelzen von Elementen im Modell Celiafish

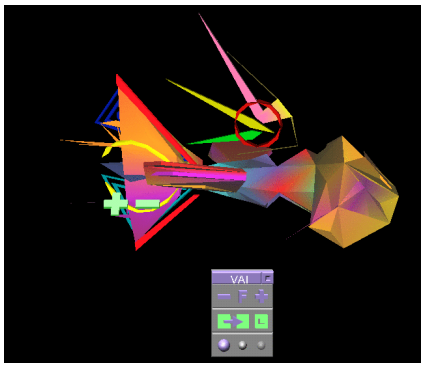


(a) Rippen fortgesetzt

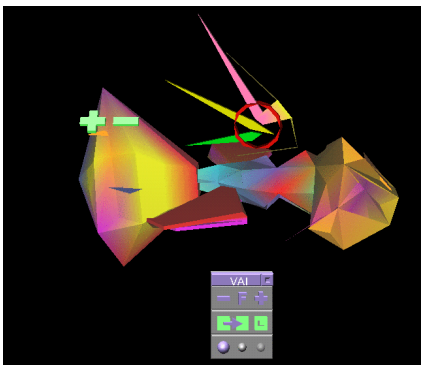
(b) Kopf



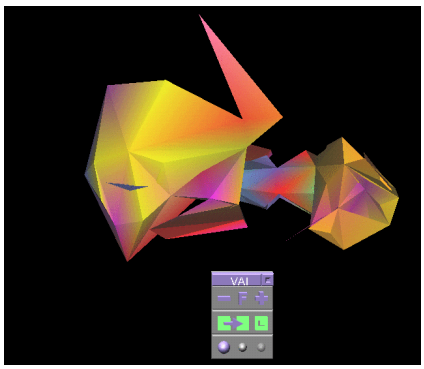
(c) Rechte Flosse



(d) Schwanzflosse halb



(e) Schwanzflosse voll



(f) Rückenflosse

Abbildung 6.15: Interaktives Verschmelzen von Elementen im Modell Celiafish. Quelle: [You97]

Kapitel 7

Zusammenfassung und Ausblick

Heute wird der Betrachter von VRML [VRM97] Welten bei der Navigation vor verschiedene Probleme gestellt. Der Wunsch nach einer realitätsnahen Darstellung fordert auf der einen Seite hohe Rechenleistung zur Bildgenerierung. Auf der anderen Seite wird der Betrachter durch die Darstellung zuvieler Details von seinem Navigationsziel abgelenkt. Desweiteren führt gerade ein Hauptvorteil von VRML, die einfache Interaktion mit Objekten, häufig zu einem Problem bei der Navigation. Dies liegt daran, daß die Interaktionsmöglichkeiten für den Betrachter nicht klar erkennbar sind, sondern nur durch Probieren herausgefunden werden können.

Ausgehend von diesen Schwierigkeiten wurde in dieser Arbeit das Hilfssystem VAI entwickelt.

7.1 Kernergebnisse der Arbeit

An dieser Stelle sollen die erbrachten Leistungen und Ergebnisse dieser Diplomarbeit kurz zusammengefaßt und auf den Punkt gebracht werden.

- Das System verwendet erstmalig die Technik der **graphischen Abstraktion**, die im Prozess der Bilderzeugung sowohl auf der Modell- als auch auf der Generierungsebene eingesetzt wird.
- Der Betrachter der Welt bekommt durch VAI eine Konsole zur Verfügung gestellt, mit der er **interaktiv den Abstraktionsgrad der einzelnen Objekte direkt manipulieren kann**.
- Ergänzend sind **funktionale Sichten** auf die Welt abrufbar, um interaktive Elemente und Hyperlinks graphisch besonders hervorzuheben.
- Die Arbeit beschreibt **Architektur und Implementation** zweier Varianten des VAI-Systems und demonstriert dessen Möglichkeiten anhand von **Anwendungsbeispielen**.
- **Ein Verfahren zur Primitiverkennung in Polygonmodellen** wurde entwickelt, um die Qualität der Abstraktionsergebnisse zu verbessern. So lassen sich hierarchische Strukturen finden, anhand derer sinnvolle Verschmelzungsoperationen erfolgen können.

7.2 Konzeptionelle Erweiterungen

Im folgenden werden einige Ansätze besprochen, die Techniken der Abstraktion über die Möglichkeiten von VRML 2.0 hinaus auszudehnen und auf andere Szenarien zu übertragen.

- **Lupen-Metapher.** Statt den Detaillierungsgrad eines Objektes über Ikonen zu manipulieren, könnte ein Instrument ähnlich einer Lupe verwendet werden. Es wird vom Betrachter über ein abstraktes Modell gezogen. Dabei erscheint die beobachtete Stelle nicht vergrössert, sondern voll detailliert.
- **Abstraktion auf Generierungs- und Bildebene.** Derzeit wird von VRML kein Eingriff in die Generierungs- und Bildebene ermöglicht. Zukünftige Systeme könnten dies jedoch ändern und den Einsatz von den in Abschnitt 2.4 vorgestellten Verfahren wie Skizzenrendering und Bitmapfilter zulassen.
- **Java3D.** Praktisch ist es nicht möglich, Java 3D-Programme ähnlich VRML- Welten zu parsen und zu manipulieren. Das liegt vor allem daran, daß Java als Programmiersprache viel mächtiger als das einfache VRML-Ereignismodell ist. Es ist jedoch durchaus denkbar, die für VAI entwickelten Abstraktionstechniken den Entwicklern von Java-Anwendungen in Form von Klassen verfügbar zu machen.

7.3 Implementatorische Erweiterungen

Das entwickelte System kann bereits allgemeine VRML-Welten verarbeiten und sinnvoll eingesetzt werden. Es sind jedoch noch einige Ansätze zur Weiterentwicklung gegeben, da sie im Rahmen dieser Arbeit nicht umgesetzt werden konnten. Diese Punkte sollen im folgenden Abschnitt kurz aufgeführt werden.

7.3.1 VAI

- **Eine graphische Benutzeroberfläche für die statische VAI-Variante**

  Ein GUI könnte den Programmablauf automatisieren und die Einstellung von Parametern zur Abstraktion erleichtern.

- **Bessere Berücksichtigung von Farben und Texturen bei Abstraktion**

  Bisher werden Texturen zwar auf die Abstraktion übertragen, jedoch nicht selbst durch Bitmapfilter abstrahiert. Bei Texturen oder Farben, die an einzelne Polygone gebunden sind, kommt es naturgemäß beim Verschmelzen von Polygonen durch den Filteralgorithmus von Rossignac zu Problemen.

- **Einsatz von besseren Abstraktionstechniken bei hohen Abstraktionsgraden**

  Bei hohen Abstraktionsgraden werden die Modelle bei Verwendung des Rossignac-Filteralgorithmus sehr kantig. Hier können Verfahren zur Glättung von Polygonmodellen angewendet werden wie z.B. Wavelets [SDS96].

7.3.2 Primitiverkennung

- **Integration der Primitiverkennung in VAI**

  Zum Einsatz der Primitiverkennung in VAI muß es möglich sein, Muster von regelmässigen Anordnungen gleicher Primitive zu erkennen und diese dann im Szenegraphen zu gruppieren. Nur so kann das Abstraktionsergebnis durch Verfeinerung der Hierarchie verbessert werden.

- **Beachtung von Flächen bei Quadererkennung statt wie bisher nur von Kanten**

  Bisher wird nur mit einem Kantenmodell gearbeitet, dabei werden eventuell offene Seitenflächen nicht bemerkt. Nachdem ein Quader gefunden wird, muß überprüft werden, ob die durch Kanten beschriebenen Seitenflächen durch Polygone ausgefüllt sind. Dieser Test ist nicht trivial, da die Kanten segmentiert sein können und die Fläche so aus beliebig vielen Polygonen bestehen kann.

- **Beachtung von Material der Flächen wie Farbe oder Textur**

  Materialeigenschaften können den Suchraum weiter beschränken und die Zuordnung von Flächen zu verschiedenen Körpern erleichtern. Dazu kann die Eingabemenge der Polygone in Klassen gleicher Materialeigenschaften partitioniert werden, anschliessend wird in jeder Partition eine eigene Suche durchgeführt. Dieser Vorgang könnte der jetzigen Implementierung vorgeschaltet werden, zum Beispiel durch ein Unix-Script.

- **Intelligente Anpassung der Fehlertoleranzen**

  Durch die begrenzte Genauigkeit von Gleitkommazahlen ergeben sich Rundungsfehler, die sich aufsummieren. Zum Vergleich von Winkeln und Längen müssen daher Toleranzen eingeräumt werden. Diese können entweder numerisch abgeschätzt werden oder interativ angepasst werden, um möglichst viele Geometrien zu erkennen.

Anhang A

Primitiverkennung

A.1 Pseudocode

Mini-Spec 1.1

```
// get edges of polygons
foreach polygon p(v1,..,vi) do {
  Edges:=Edges + (v1,v2) + .. + (vi,v1);
}
quicksort(Edges);
```

Mini-Spec 1.2

```
// extract all Edges connected to a Vertex

foreach Vertex-index v do {
  C= { j | Exists (v,j) in Edges }

  foreach i in C do {
    a⃗ = V[i] - V[v];
    new edgevec(i, a⃗);  // store edge vector
  }
// test all combinations of 3 edges for corner
  foreach triple (i,j,k) with j<k<l in C^3 do {
    a⃗ = V[i]  b⃗ = V[j]  c⃗ = V[k]
    if (1 − e  <  |a⃗ b⃗ c⃗| / (|a⃗||b⃗||c⃗|)  <  1 + e) then
      new corner(v, i,j,k, e1,e2,e3) // store corner with
                                     // vertex indices and
                                     // edgevecs indices
  }
}
// normalize edgevecs

foreach e in edgevecs do {
  e.a=e.a / |a|;  // now all edge-vectors have length 1
}
quicksort (edgevecs);
replace all parallel edgevecs by unique one but remember direction;
```

```
substitute edge-index-references accordingly, set direction;

// now all parallel edges have common index and direction
```

Mini-Spec 1.3.1

```
// check 8 corner cuboid
foreach c0 in corners do {
  v0=co.v0;
  oct=co.oct;
  //find a corner connected to edge n of corner with
  //matching orientation (flipped edge n direction)
  c1=getCorner(c0,1,oct xor %001);
  c2=getCorner(c0,2,oct xor %010);
  c3=getCorner(c0,3,oct xor %100);

  etc..
}
```

Mini-Spec 1.4 *filter recognized primitives*

```
V':={v ∈ V | v is no vertex in an recognized primitive };
F':={(v1, .. , vn) ∈ F | v1, .. , vn ∈ V' };
return F';
```

Mini-Spec 2.1 *get edges and sort them*

```
// get edges of polygons
foreach polygon p(v1,..,vi) do {
  Edges:=Edges + (v1,v2) + .. + (vi,v1);
}

quicksort(Edges);
```

Mini-Spec 2.2 *determine angles*

```
// compute all edge-pairs angles and normals
foreach edge (j,i) in Edges do {
  foreach edge (i,k) with j<k in Edges do {
    a = V[j] - V[i];
    b = V[k] - V[i];
    if (|a| != |b|) continue;
    if (|a| = 0) continue;
    α = acos(a · b / |a|²);
    if (α < 100) continue;
    u = a × b;
    new angle(i,j,k,u,α,|a|);
  }
}
```

Mini-Spec 2.3 *search for cycles*

```
foreach angle(i,k,j,u⃗,α,|a⃗|) in Angles do {
  i_start=i;
  clear anglebuffer;
  while (k!=i_start) {
    if exists angle(k,m,n,u⃗,α,|a⃗|) with (n==i OR m==i) do {
      if (m==i) then l=n; else l=m;
      store angle to anglebuffer;
      i=k; k=l;
    } else break;
  }
  if (k==i_start) new cycle(anglebuffer); // found a cycle
  foreach angle in anglebuffer {
    remove angle from Angles;
  }
}

foreach cycle c do {
  let v⃗_i be the cycle's coordinates
  center c⃗ = 1/n · (v⃗_0 + .. + v⃗_n)
  radius r = |c⃗ − v⃗_0|
  normal n⃗ = (v⃗_0 − v⃗_{n/2}) × (v⃗_{n/4} − v⃗_{3n/4})
}
```

Mini-Spec 2.4 *test cycle connections*

```
// Array VC[] maps each Vertex to a cycle using it or NULL
// Get all possible connected cycles C or points V
foreach cycle c in Cycles do {
  // Set C_i contains all Cycle indices connected to i-th vertex of cycle c
  // Set V_i contains all Vertex indices connected to i-th vertex of cycle c
  C_i:={ VC[j] | ∃ edge (cycle.v[i],j) VC[j]!=NULL}
  V_i:={    j  | ∃ edge (cycle.v[i],j) VC[j]==NULL}
  // now verify all possible connections
  C = C_1 ∩ C_2 ∩ .. ∩ C_n
  V = V_1 ∩ V_2 ∩ .. ∩ V_n
  foreach i in C do new cyclecon(c,i);
  foreach v in V do new cyclecone(c,i);
}
// now give connected cycles an unique object id
// Array OC[] assigns each cycle an object id
objectid=0; // inital id
foreach cycle c in Cycles do {
  if(OC[c]==NULL) {
    OC[c]=objectid;
    push c; // put cycle c on stack
    while (stack not empty) do {
      j=pop; // pull a cycle j from stack
      foreach cycle l connected to cycle j do { // use cyclecon data for this
        if(OC[l]==NULL) { OC[l]=objectid; push l; }
      }
```

```
    }
  }
}
```

Mini-Spec 2.5 *classify objects*

```
foreach objectid do {
  let c_i be all n cycles with objectid;
  let r_i be i-th cycle radius and c⃗_i i-th cycle center;
  let m be the number of connected points p⃗_i;
  o=new object();
  o.cycle1=c_1;
  if (n>1) then o.cycle2=c_2;
  if (all cycles normals are colinear) then axis=TRUE; else axis=FALSE;
  if (n==1 && m==0) then o.type=CYCLE; o.r=c_1.r;
  if (n==1 && m==1) then o.type=CONE;  o.r=c_1.r; o.h=|c⃗_1 − p⃗_1|;
  if (n==2) then {
    if (c_1.r==c_2.r) then o.type=CYLINDER; o.r=c_1.r; o.h=|c⃗_1 − c⃗_2|;
    else  o.type=FRUSTUM;  o.r1=c_1.r; o.r2=c_2.r;o.h=|c⃗_1 − c⃗_2|;
  }
  if (n>2 && axis=TRUE) then o.type=ROT; save profile;
  if (n>2 && axis=FALSE) then o.type=ANY; save polygonmodel;
}
```

A.2 Datenstrukturen

Data structure definitions for cuboids in C language syntax

```
typedef struct corner {
  unsigned int v0,v1,v2,v3; /* vertice indices     */
  unsigned int e1,e2,e3;    /* edge id's           */
  unsigned int octant;      /* directions of edges */
  unsigned int used;        /* true=part of cuboid */
} Corner;

typedef struct edgevec {
  unsigned int id;  /* edge id   */
  double x,y,z;     /* vector    */
  int inv;          /* direction */
} Edgevec;

typedef struct connect {
  int v1;   /* vertex            */
  int v2;   /* vertex            */
  int e;    /* edge              */
  int inv;  /* direction of edge */
} Connect;
```

Data structure definitions for lathes in C language syntax

```
typedef struct angle {
  unsigned int i,j,k; /* edges (i,j) and (i,k) */
  double n1,n2,n3; /* e1 x e2 */
  double a; /* angle */
  double l; /* length of e1=e2 */
  char used; /* has been used in a cycle */
} Angle;

typedef struct cycle {
  unsigned int n;
  unsigned int v[MAXCYCLE];
  Vertex center; /* center */
  double radius; /* radius */
  Vertex normal; /* normalenvektor */
  unsigned int p; /* point-id in 2d-conversion */
} Cycle;

typedef struct cyclecon {
  int i,j; /* Connection from Cycle i to j */
} Cyclecon;

typedef struct objectcyc {
  int obj, i; /* object obj consists of cycle i */
} Objectcyc;


typedef struct object {
  int cycles, cones;
  int hasaxis; /* common axis true/false */
  Vertex axis; /* axis */
  Vertex base; /* base point */
  Vertex cap; /* base point */
  double r1,r2,h;
  int type; /*CYLINDER/FRUSTUM/CONE/ROT/ANY */
  Cycle* cycle1; /* reference to cycle 1 of cylinder/cone */
  Cycle* cycle2; /* reference to cycle 2 of cylinder */
  int c1,c2,c3; /* references to relevant cones for cyl/cone */
  char filename[32];
} Object;
```

Anhang B

VAI

B.1 Java Klassendiagramme

Die Offline-VAI Java Klassen mit ihren relevanten Attributen und Methoden

Vain	
public static void	main(String args[])

VRMLgraph	
GroupNode	root
ANode	aroot
ANode	a2root
GroupNode	abstVrml
public void	parse(String filename)
public void	traverse()
public void	graphToVRML()
public void	writeVRML(GroupNode root, String filename)
public void	readARP(ANode root, String filename)
public void	writeARP(ANode root, String filename)
public void	switchify()
public void	scriptify(String filename)
public void	route()

ANode	
public String	name
public Node	node
public ANode	parent
public Vector	VRMLnodes
public boolean	func_from
public boolean	func_to
public boolean	func_anchor
public boolean	switchnode
public TransMatrix	T
public PolygonSet	polygons
public PolygonSet	abstraction
public String	switchName
public void	printARP(PrintStream os)
public node	makeVRML()
public int	switchify()

VaiTraverser extends Traverser	
VRMLgraph	vrmlgraph
ANode	aroot
ANode	anode
void	traverse(GroupNode root)
void	tGroupNode(GroupNode group)
void	aVRML(Node n)
void	aSensor(Sensor sensor)
void	aPoly(Shape shape)

Scriptify	
static void	writeScripts(
Hashtable switchref,Hashtable switchmode)	

Polygonset	
String	filename
Appearance	appearance
Material	material
public	Polygonset()
IndexedFaceSet	createIndexedFaceSet()
Switch	createIndexedFaceSet_grey()
void	extrusionToOff()
void	ifsToOff()
void	boxToOff()
void	coneToOff()
void	cylinderToOff()
void	sphereToOff()

TransMatrix	
float[][]	A
float[][]	I
static float[]	crossProduct(float[] x,float[] y)
static float[]	length(float[] x)
static float[]	normalize(float[] x)
static float	angle(float[] x, float[] y)
static void	translation(float[] t)
static void	translation_inv(float[] t)
static void	scale(float[] t)
static float[]	transform(float[] x)
static float[]	transform_inv(float[] x)
static void	rotation(float[] r)
static void	multiply(float[][] B)
static void	makeInverse()

B.2 Datenaustausch mit ARP

Abbildung B.2 zeigt die Eingabedatei von ARP, als Beispiel wurde der Celiafish gewählt. Sie enthält einen als geklammerte Liste formatierten Baum von VRML Polygonmodellen im OFF Format. Abbildung B.2 zeigt die Ausgabedatei von ARP, die an VAI zurückgegeben wird und Abstraktionen und Verschmelzungen enthält. Die Abstraktionen sind am Pfad *tmp/ARP/AFM.*. zu erkennen, die Verschmelzungen ergeben neue Knoten mit Namen *Art*.

```
(("ROOT" NIL "" )(
  (("TRANS" NIL "" )(
    (("ribs" NIL "" )(
      (("rib1" NIL "" )(
        (("SHAPE" "vrml/celiafish/temp/SHAPE.off" "" ) NIL)
      ))
      (("rib3" NIL "" )(
        (("SHAPE1" "vrml/celiafish/temp/SHAPE1.off" "" ) NIL)
      ))
      (("rib5" NIL "" )(
        (("SHAPE2" "vrml/celiafish/temp/SHAPE2.off" "" ) NIL)
      ))
      (("rib6" NIL "" )(
        (("SHAPE3" "vrml/celiafish/temp/SHAPE3.off" "" ) NIL)
      ))
      (("bodyrib04" NIL "" )(
        (("SHAPE4" "vrml/celiafish/temp/SHAPE4.off" "" ) NIL)
      ))
      (("bodyrib05" NIL "" )(
        (("SHAPE5" "vrml/celiafish/temp/SHAPE5.off" "" ) NIL)
      ))
      (("rib2" NIL "" )(
        (("SHAPE6" "vrml/celiafish/temp/SHAPE6.off" "" ) NIL)
      ))
      (("rib4" NIL "" )(
        (("SHAPE7" "vrml/celiafish/temp/SHAPE7.off" "" ) NIL)
      ))
      (("bodyrib08" NIL "" )(
        (("SHAPE8" "vrml/celiafish/temp/SHAPE8.off" "" ) NIL)
      ))
      (("bodyrib09" NIL "" )(
        (("SHAPE9" "vrml/celiafish/temp/SHAPE9.off" "" ) NIL)
      ))
    ))
    (("front" NIL "" )(
      (("face" NIL "" )(
        (("SHAPE10" "vrml/celiafish/temp/SHAPE10.off" "" ) NIL)
      ))
      (("face01" NIL "" )(
        (("SHAPE11" "vrml/celiafish/temp/SHAPE11.off" "" ) NIL)
      ))
...
```

Abbildung B.1: Datei celiafish.arp (gekürzt)

```
(("ROOT" NIL)
 ((("TRANS" "tmp/ARP/AFM-3823-5565")
   ((("Art" "tmp/ARP/AFM-3823-5561")
     ((("LEFT_SIDE_FIN" "tmp/ARP/AFM-3823-5464")
         ((("Art" "tmp/ARP/AFM-3823-5460")
             ((("sidefintop" NIL)
                 ((("SHAPE33" "tmp/ARP/AFM-3823-5458") NIL))
              )
               (("sidefinbot" NIL)
                 ((("SHAPE34" "tmp/ARP/AFM-3823-5457") NIL))
          )  ) )
           (("Art" "tmp/ARP/AFM-3823-5462")
             ((("sidefinmid" NIL)
                 ((("SHAPE35" "tmp/ARP/AFM-3823-5456") NIL))
              )
               (("sidetoptob" NIL)
                 ((("SHAPE36" "tmp/ARP/AFM-3823-5455") NIL))
      )  ) ) ) ) )
       (("Art" "tmp/ARP/AFM-3823-5557")
         ((("ribs" "tmp/ARP/AFM-3823-5555")
           ((("Art" "tmp/ARP/AFM-3823-5551")
              ((("Art" "tmp/ARP/AFM-3823-5539")
                  ((("rib1" NIL)
                    ((("SHAPE" "tmp/ARP/AFM-3823-5537") NIL))
                   )
                    (("bodyrib05" NIL)
                    ((("SHAPE5" "tmp/ARP/AFM-3823-5532") NIL))
               )  ) )
                (("Art" "tmp/ARP/AFM-3823-5547")
                  ((("rib4" NIL)
                    ((("SHAPE7" "tmp/ARP/AFM-3823-5530") NIL))
                   )
                    (("Art" "tmp/ARP/AFM-3823-5541")
                      ((("rib3" NIL)
                       ((("SHAPE1" "tmp/ARP/AFM-3823-5536") NIL))
                       )
                       (("rib2" NIL)
                       ((("SHAPE6" "tmp/ARP/AFM-3823-5531") NIL))
           ) ) ) ) ) ) )
            (("Art" "tmp/ARP/AFM-3823-5553")
              ((("Art" "tmp/ARP/AFM-3823-5543")
                 ((("rib5" NIL)
                   ((("SHAPE2" "tmp/ARP/AFM-3823-5535") NIL))
                  )
                   (("rib6" NIL)
                   ((("SHAPE3" "tmp/ARP/AFM-3823-5534") NIL))
                )  ) )
...
```

Abbildung B.2: Datei celiafish.abs (gekürzt)

B.3 Beispiel Javascript

```
javascript:
    function hitPointA(value){
      consolescript.set_pos = value;
      consolescript.callback = scriptnode;
      consolescript.activate = activate; // + or +-
      memory=0; // A
    }
    function hitPointB(value){
      consolescript.set_pos = value;
      consolescript.callback = scriptnode;
      consolescript.activate = 2; // only -
      memory=1; // B
    }
    function clickPlus(value){
      if (memory==0) {  // A+
        switchnode.set_whichChoice = 1;
      }
      if (memory==1) {  // B+
        // impossible case
      }
      consolescript.activate = 0; // hide +-
    }
    function clickMinus(value){
      if (memory==0) {  // A-
        parentswitchnode.set_whichChoice = 0;
      }
      if (memory==1) {  // B-
        switchnode.set_whichChoice = 0;
      }
      consolescript.activate = 0; // hide +-
    }
```

Abbildung B.3: switchscript.js

Literaturverzeichnis

[3dm00] *3D Studio MAX*. URL: http://www.3dmax.com, 2000.

[ama98] *Amapi 3D*. URL: http://www.tgs.com, 1998.

[Aut95] Autorenkollektiv. *AP5/C User's Guide*. Asus Inc., 1995.

[Aut97] Autorenkollektiv. *Elsa Gloria Synergy: Handbuch*. Elsa GmbH, Aachen, 1997.

[Aut99] Autorenkollektiv. *External Authoring Interface*. EAI Working Group, URL: http://www.web3d.org/WorkingGroups/vrml-eai/, 1999.

[CB97] Rikk Carey and Gavin Bell. *The Annotated VRML 2.0 Reference Book*. Addison-Wesley Publishing Company, URL: http://www.vrml.org/fs_specifications.htm, 1997.

[Con01] Web3D Consortium. *VRML Repository*. URL: http://www.web3d.org, 2001.

[DCM98a] Anonymer Autor 3D-Cafe Modellbibliothek. *Capitol*. URL: http://www.3DCafe.com/architecture/landmarks/, 1998.

[DCM98b] Anonymer Autor 3D-Cafe Modellbibliothek. *Videorecorder*. URL: http://www.3DCafe.com/, 1998.

[DeM78] Tom DeMarco. *Structured Analysis and System Specification*. 1978.

[Fei85] Steven Feiner. Apex: An experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications*, 5(11):117–123, 1985.

[FvDFH96] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. The system programming series. Addison-Wesley, second edition, 1996.

[fvw96] *FVWM Virtual Dektop Window Manager for X Window System*. URL: http://www.fvwm.org, 1996.

[Geo96a] The Geometry Center, University of Minnesota, URL: http://www.geomview.org/docs/. *Geomview*, 1996.

[Geo96b] The Geometry Center, University of Minnesota, URL: http://www.geomview.org/docs/oogltour.html. *OOGL Geom File Formats*, 1996.

[Gra00] Parallel Graphics. *Computertable*. URL: http://www.parallelgraphics.com/showroom/, 2000.

[HHS98] Ralf Helbing, Knut Hartmann, and Thomas Strothotte. Dynamic visual emphasis in interactive technical documentation. In Thomas Rist, Hsg., *Workshop: AI and Graphics for the interfaces of the future*, Seiten 82–90. held in conjunction with ECAI'98 in Brighton, 1998.

[Hop96] Hugues Hoppe. Progressive meshes. In *Computer Graphics (SIGGRAPH 96 Conference Proceedings)*, New Orleans, 1996.

[Ins00] Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Austria., URL: http://www.iicm.edu/vrwave. *VRwave*, 2000.

[JSGB00] Joy, Steele, Gosling, and Bracha. *The Java Language Specification*. Addison-Wesley, URL: http://www.java.sun.com/, 2nd edition, 2000.

[Kos94] Stephen Michael Kosslyn. *Elements of graph design*, Seiten 1–13. W.H. Freeman and Company, USA, 1994.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language, Second Edition, ANSI C*. Prentice-Hall International, 1988.

[Krü00] Antonio Krüger. *Automatische Abstraktion in 3D-Graphiken*. infix, 2000.

[Krü95] Antonio Krüger. Proxima: Ein System zur Generierung graphischer Abstraktionen. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, Prof. Dr. W. Wahlster, 1995.

[KS97] Tatjana Klajic and Christoph Stahl. Ein System zur graphischen Abstraktion von Polygonmodellen. Fortgeschrittenenpraktikum an der Universität des Saarlandes, 1997.

[Net00] Netscape Communications, Netscape World Headquarters, 466 Ellis Street , Mountain View, CA 94043-4042. *Netscape Navigator*, 2000.

[New01] NewTek, URL: http://www.newtek.com. *Lightwave*, 2001.

[Pla00] Planet9. *Virtual Tokyo*. URL: http://www.planet9.com/earth/tokyo/, 2000.

[Pre98] Bernhard Preim. *Interaktive Illustrationen und Animationen zur Erkl ärung komplexer räumlicher Zusammenhänge*, Seiten 70–74. VDI Fortschrittberichte, 1998.

[PT99] POV-Team. *Persistance Of Vision*. URL: http://www.povray.org, 1999.

[RB92] Jarek R. Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. Technical report, IBM Research Division, T.J. Watson Research Centre, Yorktown Heights, NY 10598, 1992.

[RN95] Stuart Russel and Peter Norvig, Hsg. *Artificial Intelligence - A Modern Approach*, Seiten 734–749. Prentice Hall, Inc., New Jersey, 1995.

[SC92] Paul S. Strauß and Rikk Carey. An object-oriented 3d-graphics toolkit. In Edwin E. Catmull, Hsg., *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, Seiten 341–352, Chicago, Illinois, 1992. Addison Wesley.

[Sch99] Stefan Schlechtweg. *Interaktives wissenschaftliches Illustrieren von Texten*. Shaker Verlag, 1999.

[SDS96] Eric J. Stollnitz, Tony D. Derose, and David H. Salesin. *Wavelets For Computer Graphics*, Seiten 109–123. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[Sed90] Robert Sedgewick. *Algorithmen in C*, Seiten 145–157. Addison-Wesley Publishing Company, Inc., 1990.

[Shn92] Ben Shniderman. *Designing the User Interface*, Seiten 181–205, 277–299. Addison-Wesley Publishing Company, 1992.

[Shr99] Dave Shreiner, Hsg. *OpenGL Reference Manual*. Addison-Wesley, URL: http://www.opengl.org, 3rd edition, 1999.

[Sil98] Silicon Graphics and Cosmo Software, URL: http://www.cai.com/cosmo/. *Cosmoplayer*, 1998.

[SK00] Peter Mattis Spencer Kimball. *GNU Image Manipulation Program*. GNU, URL: http://www.gimp.org, 2000.

[SRD00] Sowizral, Rushforth, and Deering. *Java 3D API Specification*. Addison-Wesley, URL: http://www.j3d.org/books/, 2000.

[Umb98] Scott E. Umbaugh. *Computer Vision and Image Processing*. Prentice Hall, 1998.

[vD91] Andries van Dam. Electronic books and interactive illustrations - transcript of a talk. In *Interactive Learning Through Visualization*, Seiten 9–24, 1991.

[VRM97] VRML97. *The Web 3D Consortium* . URL: http://www.vrml.org/fs_specifications.htm, 1997.

[Wal75] D. Waltz. *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[You97] C. Scott Young. *Celiafish*. The VRML Knight, 1997.