Ein inkrementeller Ansatz zur Generierung informativer 3D-Animationen

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

vorgelegt von

Andreas Martin Butz

Saarbrücken
27. Juni 1997

Kurzzusammenfassung

Die vorliegende Arbeit beschreibt einen neuen Ansatz zur inkrementellen Generierung von 3D-Animationen. Die Skripte (Drehbücher) der Animationen haben eine hierarchische Struktur, die durch eine Skriptgrammatik beschrieben wird. Die Expansion der Grammatikregeln wird durch den Generierungskontext gesteuert, der im Verlauf der Generierung modifiziert wird. Aufgrund der in diesem Kontext enthaltenen Informationen ist es möglich, die generierten Animationen an bestimmte stilistische Vorgaben (Generierungsparameter) und die zur Verfügung stehende Rechenzeit und die Darstellungsmöglichkeiten anzupassen (Ressourcenadaptivität). Die Skripte werden außerdem Schritt für Schritt erzeugt und die Darstellung der Animation beginnt bereits während der Generierung des Skriptes (Inkrementalität). Das Verfahren wird anhand des implementierten Systems CATHI evaluiert.

Short abstract

This dissertation describes a new approach to the incremental generation of 3D animations. The animation scripts have a hierarchical structure which is described by a script grammar. The expansion of the grammar rules is guided by a generation context, which is also modified over time. The information contained in the context allows the animation to be adapted to stylistic constraints (generation parameters) and to the specific limitations of computing and graphics capacity (resource adaptiveness). The scripts are generated step by step and the presentation of the animation starts while later parts of the script are still being generated (incrementality). The approach is evaluated as implemented in the system CATHI.

Zusammenfassung

Die vorliegende Arbeit beschreibt einen neuen Ansatz zur inkrementellen Generierung informativer 3D-Animationen im Kontext intelligenter Multimedia-Präsentationssysteme. Ausgangspunkt für die Generierung sind die mit der Animation verbundenen kommunikativen Ziele innerhalb eines koordinierten Multimediadokumentes sowie eine vorgegebene 3D-Modellwelt. Der Vorgang der Animationsgenerierung gliedert sich in drei Teilschritte: die Erzeugung eines Animationsskriptes, die Bildberechnung und die Darstellung der Animation am Bildschirm. Mit dem beschriebenen Ansatz ist es erstmals möglich, alle drei Teilschritte zeitlich überlappend auszuführen. Das Verfahren arbeitet also vollständig inkrementell, was die Zeitspanne vom Beginn der Generierung bis zum Beginn der Darstellung erheblich verkürzt.

Die erzeugten Animationsskripte haben eine hierarchische Struktur, die durch die Regeln einer Skriptgrammatik beschrieben wird. Die Expansion der Regeln dieser Grammatik wird durch einen Kontext gesteuert, der eine Beschreibung des aktuellen Zustandes der Modellwelt enthält und während der Skriptgenerierung laufend aktualisiert wird. Hierdurch ist es möglich, gestalterische Entscheidungen in Abhängigkeit von der aktuellen Kameraposition oder Objektpositionen und -eigenschaften in der Grammatik zu kodieren. Der Kontext enthält außerdem in Form von Generierungsparametern Informationen über stilistische Vorgaben sowie über die graphischen Fähigkeiten des Ausgabemediums wie Farbe oder Transparenz. Hierdurch kann die Verwendung bestimmter gestalterischer Mittel in der Animation gesteuert werden und die verfügbare Graphikleistung der zur Darstellung verwendeten Maschine optimal ausgenutzt werden.

Bei der Expansion der Regeln wird die Skriptstruktur in chronologischer Reihenfolge und in die Tiefe aufgebaut. So können in sich abgeschlossene Teilskripte, die Inkremente, bereits zur Bildberechnung weitergereicht werden, während spätere Skriptteile noch generiert werden. Die Bildberechnung und Darstellung kann somit schon nach der Generierung des ersten Inkrementes beginnen. Nach Abschluß jedes einzelnen Skriptteils werden Informationen über die zur Generierung benötigte Zeit sowie über die in der Darstellung bereits verstrichene Zeit festgehalten, woraus sich ableiten läßt, wie viel Zeit für die Generierung des nächsten Skriptteils zur Verfügung steht. In Abhängigkeit von dieser Information kann in der Skriptgrammatik zwischen einfacheren oder komplexeren Expansionen des nächsten Skriptteiles ausgewählt werden, so daß sich ein Schritthalten des Generierungsprozesses mit der schon laufenden Animation in den meisten Fällen erreichen läßt. Das Verfahren verhält sich also in zweifacher Weise ressourcenadaptiv, indem neben den graphischen Fähigkeiten des Ausgabemediums auch die Rechenleistung der zur Generierung verwendeten Maschine berücksichtigt wird.

Der vorgestellte Ansatz wurde in dem System CATHI implementiert und anhand dreier verschiedenartiger Domänen überprüft. Da die Implementation auf einer breiten Palette von Rechnern lauffähig ist, konnte neben der Funktion des Verfahrens auch sein Verhalten unter verschiedensten Bedingungen evaluiert werden. Die in der Arbeit gezeigten Generierungsbeispiele dokumentieren die Skriptgenerierung bei unterschiedlichen Beschränkungen der Rechenleistung sowie der graphischen Fähigkeiten bei der Ausgabe. Die generierten Animationsskripte können durch zwei verschiedene Graphiksysteme dargestellt werden. Eines dieser Systeme stellt die Ani-

mationen mit begrenzten graphischen Mitteln in Echtzeit dar, während das andere auch aufwendigere Effekte wie Scheinwerfer oder Tiefenschärfe unterstützt, allerdings derzeit nicht in Echtzeit.

Die in den Animationen verwendeten gestalterischen Mittel sind in der Skriptgrammatik kodiert. Sie beschränken sich nicht auf die filmtechnischen Mittel der Kameraführung, des Schnittes und der Beleuchtung, sondern beziehen für die Computergraphik typische Stilmittel wie Farb- und Transparenzeffekte, Metagraphik sowie die gezielte Verwendung unterschiedlicher Detaillierungsgrade mit ein. Somit kann die Ausdrucksstärke der erzeugten Animationen gegenüber anderen, rein filmtechnisch motivierten Ansätzen wesentlich erhöht werden. Im Ausblick der Arbeit werden schließlich Erweiterungen und andere Einsatzgebiete vorgeschlagen und die notwendigen Veränderungen diskutiert.

Danksagung

Zum Gelingen dieser Arbeit haben einige Menschen beigetragen, die ich an dieser Stelle nennen möchte. Zunächst bedanke ich mich bei meinem Doktorvater Wolfgang Wahlster. Er schlug das sehr interessante Thema vor unterstützte mich massiv bei dessen Bearbeitung in wissenschaftlichen wie in praktischen Fragen. Bei den Mitarbeitern am DFKI fand ich außerdem jederzeit ein offenes Ohr und fruchtbare Diskussionen. Meinem Mitstudenten Antonio Krüger verdanke ich die in der Arbeit verwendeten Modellabstraktionen sowie viele Anregungen. Teile des Systems CATHI wurden im Rahmen eines Fortgeschrittenenpraktikums von Jens Haase, Matthias Hüther und Uwe Kern implementiert. Zwei der drei Modellwelten wurden von Harry H. Chang modelliert. Für Korrekturen und Vorschläge zum Manuskript dieser Arbeit danke ich Elisabeth André, Gerhart und Hannelore Butz, Silvia Eichhorn-Jung, Tatjana Klajić, Antonio Krüger, Susanne Van Mulken und Stephan Oepen.

Meine finanzielle Grundlage in den vergangenen zwei Jahren bildete ein Stipendium der Landesgraduiertenförderung des Saarlandes sowie die zusätzliche Unterstützung durch meine Eltern. Meinen seelischen Rückhalt finde ich in Tatjana Klajić, die mir außerdem stets eine interessierte Diskussionspartnerin ist. Mein Zweitgutachter Bernhard Nebel erklärte sich recht kurzfristig dazu bereit, dieses mit nicht wenig Aufwand verbundene Amt zu übernehmen und ihm gehört – wie allen anderen an der Endphase meiner Promotion beteiligten Personen – großer Dank für die Einhaltung des recht eng bemessenen Zeitplanes.

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Prolog

B: Ja, hallo, ich bin's. Ich wollte fragen, ob du Lust hast, morgen auf einen Kaffee vorbeizukommen.

B: ...Nein, wieso Anlaß? Einfach so. Mir geht da so eine Idee im Kopf herum und du hast doch Ahnung vom Filmemachen...

B: ...Nein, ich will keinen Film drehen. Aber sowas ähnliches. ... Gut. Also um Vier. Bis dann! (legt auf und wählt neu)

B: (wartet etwas, rollt die Augen) Hallo, hier ist B. Ich wollte Dich für morgen zum Kaffee einladen. Ruf mich doch bitte zurück, ob das bei Dir klappt.

Kapitel 1

Einleitung

1.1 3D-Animation: Begriffsklärung und Abgrenzung

Wenn man das Wort *3D-Animation* in seine Teile zerlegt, so besteht es aus *3D* – der Abkürzung für dreidimensional – und *Animation*. *Animieren* bedeutet im eigentlichen Sinne 'beleben' und kann so für jede Art bewegter Darstellung verwendet werden. Der Zusatz *3D* schränkt die Bedeutung auf die bewegte Darstellung dreidimensionaler Objekte und Szenerien ein, was im Gegensatz zu *2D*-Animationen wie z. B. Zeichentrick steht.

Im Zusammenhang mit Computern hat der Begriff dann eine weitgehend eingegrenzte und klar definierte Bedeutung, nämlich die bewegte Darstellung dreidimensionaler Körper auf einem zweidimensionalen Bildschirm. Mittels einer virtuellen Kamera werden Punkte des dreidimensionalen Raumes auf die zweidimensionale Bildfläche projiziert und erzeugen dort ein Bild, das dem Betrachter in der Regel eine Rekonstruktion ihrer Lage im dreidimensionalen Raum ermöglicht.

Diese Projektion kann auf verschiedene Arten ausgeführt werden und stellt die Modelle durch Linienzüge oder schattierte Flächen im Bild dar. Sie kann der physikalischen Realität mehr oder weniger genau entsprechen, je nachdem, wie genau beispielsweise der Strahlengang des Lichtes in der Kamera modelliert wird, oder wie gut die Verfahren sind, durch die die Oberflächeneigenschaften der Modelle nachgebildet werden.

Um aus derart berechneten Bildern Animationen zu erhalten, müssen mehrere Bilder berechnet und hinreichend schnell nacheinander dargestellt werden, so daß beim Betrachter der Eindruck einer Bewegung entsteht. Dieser Bewegungseindruck kann dadurch hervorgerufen werden, daß sich die Kamera im Raum bewegt, oder daß die dargestellten Objekte ihre Position, Orientierung oder andere Eigenschaften wie Farbe, Form oder Transparenz verändern. Sofern das Projektionsverfahren mit einem Lichtmodell arbeitet, kann auch eine Veränderung der Beleuchtung einen Bewegungseindruck hervorrufen.

1.2 3D-Animationen im multimedialen Kontext

Animationen können mit anderen Darstellungsformen wie Text oder statischen Graphiken auf einem Computerbildschirm zu *Multimedia*präsentationen oder -dokumenten kombiniert werden. Nimmt man die Möglichkeit einer akustischen Ausgabe hinzu, so können auch Geräusche, Musik oder Sprache hinzukommen. Solche Präsentationen haben die Aufgabe, Information zu übermitteln. Verschiedene Bestandteile der Präsentation übermitteln dabei jeweils Teile der Gesamtinformation. Einerseits können sich diese Informationsteile überschneiden, andererseits wird auch durch die zeitliche und räumliche Anordnung der Präsentationsteile Information vermittelt. Multimediale Präsentationen können Information wesentlich besser verständlich machen als Präsentationen, die auf ein Medium beschränkt sind, da jedes Medium zur Darstellung bestimmter Informationsarten besser geeignet ist. In Texten ist es beispielsweise recht einfach, kausale Zusammenhänge zu vermitteln, während räumliche Information vorteilhafter in einer Graphik oder Animation dargestellt werden kann.

Multimediadokumente werden herkömmlicherweise von menschlichen Autoren gestaltet. Auch ihre einzelnen Bestandteile wie Texte, Graphiken, Video oder Animationen sind meist das Produkt menschlicher Autoren, auch wenn sie mit Hilfe des Rechners erzeugt wurden. Um beispielsweise eine 3D-Animation zu erzeugen, muß ihr Autor deren gesamten Ablauf festlegen. Dies umfaßt die Positionen und Bewegungen der Kamera und der dargestellten Objekte in der Zeit sowie die Beleuchtung der Szenerie. Die Erstellung optisch ansprechender Animationen ist mit einem recht hohen gestalterischen Aufwand verbunden, abgesehen vom Rechenaufwand für die Erzeugung der Einzelbilder.

Um den Aufwand für die Erstellung multimedialer Präsentationen zu reduzieren, wird zunehmend deren automatische Erstellung durch den Rechner erforscht. Sogenannte Intelligente Multimedia-Präsentations- (IMP-) Systeme, wie sie beispielsweise in [André, 95, André et al., 96, Feiner, 85] beschrieben werden, generieren die Bestandteile eines Dokumentes automatisch und setzen daraus eine Präsentation zusammen. Da jede neu erzeugte Präsentation im Idealfall aus genau aufeinander abgestimmten Teilen besteht, müssen diese Teile entweder in großer Vielfalt zur Verfügung stehen oder jeweils genau passend erzeugt werden. Eine komplette Neuerzeugung der Präsentationsteile ist hierbei zweifellos der flexiblere, wenn auch aufwendigere Ansatz.

Im Falle der 3D-Animationen bedeutet eine komplette Erzeugung zunächst die Berechnung passender Kamerapositionen, Kamerafahrten und Lichter, sowie deren zeitliche Koordination mit eventuell zu zeigenden Objektbewegungen. All diese Informationen müssen in einem Animationsskript zusammengefaßt und anschließend in Bilder umgesetzt und am Bildschirm dargestellt werden.

1.3 Zielsetzung und Anspruch der Arbeit

In der vorliegenden Arbeit möchte ich einen neuen Ansatz zur automatischen Generierung von 3D-Animationen entwickeln. Dieser Ansatz zeichnet sich einerseits durch eine sehr hohe Generierungsgeschwindigkeit und eine inkrementelle Konzeption aus, die es auch mit einfachen Rechnern ermöglicht, Präsentationen innerhalb weniger

Sekunden zu erzeugen, was für interaktive Präsentationen ein sehr großer Gewinn ist. Andererseits bietet der Ansatz genügend Ausdrucksstärke und Flexibilität zur Erzeugung optisch ansprechender Animationen. Mit dem System CATHI werde ich eine Implementation meines Ansatzes vorstellen und seine Generierungsergebnisse an Beispielen erläutern.

Schwerpunkt der Arbeit ist die Erzeugung der Animationsskripte, die eine vollständige geometrische Spezifikation der Animationen enthalten. Die Darstellung der Animationen am Bildschirm erfolgt dann mit Hilfe vorhandener Bilderzeugungssysteme (Renderer) und Standard-Software. Ziel der Arbeit ist also nicht die Konzeption eines neuen Bilderzeugungssystems, das möglichst viele graphische Effekte darstellen kann, sondern der gezielte Einsatz der gegebenen graphischen Mittel vorhandener Systeme. Die generierten Animationen arbeiten gezielt mit den filmtechnischen Mitteln der Kameraführung, der Beleuchtung und des Schnitts, nutzen aber auch Computergraphikspezifische Stilmittel wie Transparenz, Farbeffekte oder Metagraphik. Der Einsatz all dieser Ausdrucksmittel hängt von den graphischen Möglichkeiten des Ausgabesystems sowie von der Parametrisierung des Skriptgenerators ab.

Wie die Formulierung eines *guten* Textes oder das Erstellen einer *guten* Graphik ist auch die Gestaltung einer *guten* Animation ein kreativer Akt und als solcher natürlich im Rechner nicht nachvollziehbar. Trotzdem lassen sich bestimmte Aspekte der Gestaltung in Regeln und Gesetze fassen, die natürlich von menschlichen Gestaltern beliebig durchbrochen werden können. Bei Texten ist dies beispielsweise die Grammatik und die Rhetorik, für technische Abbildungen gibt es sogar DIN-Normen [DIN, 76, DIN, 82], die die Gestaltung einer (im Sinne dieser Normen) *richtigen* Zeichnung beschreiben. Solche Regelwerke können ein Gerüst liefern, nach dem es möglich ist, eine *formal richtige* Gestaltung vorzunehmen (Siehe auch Abschnitt 2.3). Dieses Gerüst dient auch als Grundlage einer Implementation und erlaubt es, Systeme zu entwickeln, die selbsttätig *formal richtige* Texte oder Graphiken erstellen. Wie *gut* diese Präsentationen sind, hängt vom einzelnen System ab.

Die beiden Worte *gut* und *richtig* haben, sofern sie kursiv gesetzt sind, in dieser Arbeit eine spezielle Bedeutung. Eine *richtige* Präsentation ist eine Präsentation, deren Gestaltung bestimmten formalen Anforderungen genügt, und die ihren Zweck erfüllt. Eine *gute* Präsentation ist darüberhinaus eine Präsentation, die ihren Zweck auf besonders effiziente oder elegante Art erfüllt. Eine *gute* Präsentation muß also zumindest immer *richtig* gestaltet sein, während nicht jede *richtige* Präsentation auch *gut* ist. In diesem Sinne ist es Ziel der vorliegenden Arbeit, einen Ansatz zur Generierung bildsprachlich *richtiger* 3D-Animationen zu beschreiben, die bestimmte Informationen möglichst *gut* vermitteln. Einen künstlerischen Anspruch verbinde ich mit den so gestalteten Animationen ausdrücklich nicht. Eine formalere Beschreibung der Wirksamkeit intentionsbasierter Präsentationen findet sich beispielsweise in [Graf, 96] auf Seite 28.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit folgt in ihrem Aufbau der Vorgehensweise bei der Bearbeitung des gestellten Themas. Zunächst wird die Gestaltung von 3D-Animationen aus

verschiedenen Blickrichtungen grundsätzlich beschrieben. Kapitel 2 beleuchtet diesen Prozeß aus der Sicht der Filmtechnik, aus computergraphischer Sicht und mit Blick auf den Informationsgehalt und die Struktur der erzeugten Animationen. Hierbei werden die gestalterischen, technischen und terminologischen Grundlagen zum Verständnis der Arbeit gelegt. In Kapitel 3 werden zunächst einige andere Ansätze zur Lösung ähnlicher Probleme diskutiert, danach in Kapitel 4 die offenen Probleme umrissen und die Grundkonzeption eines neuen Ansatzes vorgestellt. Kapitel 5 beschreibt dann die entwickelten Verfahren zur Durchführung der geometrischen Berechnungen in der Modellwelt, während Kapitel 6 den von mir entwickelten Ansatz zur inkrementellen Skriptgenerierung im Detail erläutert. In Kapitel 7 werden einige praktische Aspekte des implementierten Systems CATHI diskutiert, wobei Abschnitt 7.4 Strategien zur technischen Umsetzung der generierten Skripte in Animationen beschreibt und deren Anwendbarkeit in verschiedenen Situationen diskutiert. Nachdem in Kapitel 8 ein Generierungsbeispiel ausführlich beschrieben wurde, werde ich in Kapitel 9 auf denkbare Anwendungs- und Erweiterungsmöglichkeiten des Ansatzes sowie des Systems CATHI eingehen. In den Anhängen schließlich werden technische Details aufgeführt. Dies sind die Typen elementarer Skriptsequenzen (Anhang A), die Regeln der Skriptgrammatik (Anhang B) sowie weitere Animationen als Bildfolgen (Anhang C).

Erste Szene

Auf dem Tisch stehen die Reste von Kaffee und Kuchen. Die drei sind in ein erregtes Gespräch über den Sinn oder Unsinn der Idee vertieft.

A: ...So ein Quatsch! Ein Drehbuch von einem Computer schreiben lassen! Wenn dabei so etwas herauskommt wie bei deinen computergeschriebenen Gedichten letztens, dann Danke!

B: Das kannst du nicht vergleichen. Ein Gedicht ist was anderes. Ein normaler Text, der einfach irgendwas erzählt oder erklärt läßt sich schon automatisch generieren.

A: Also ich sehe nicht, wie das gehen soll. So ein Drehbuch zu schreiben, weißt du überhaupt selbst, wie man das macht? Da wirst du Dir ganz schön die Zähne ausbeißen.

B: (grinst) Na, das lasse ich mir doch gern von dir erklären, wenn du das so gut kannst... Wozu hat man schließlich Freunde?

C: (nachdenklich) ...Aber mit dem Drehbuch ist es ja noch nicht getan! Weißt du überhaupt, was man für eine 3D-Animation alles eingeben muß? Kamerapositionen, Keyframes, und du mußt deine Szenen auch ausleuchten. Beim besten Willen, sowas automatisch zu machen ist doch Unsinn. Wo sollen denn die ganzen Parameter herkommen?

B: Also ich denke mir, daß man das ganz ähnlich machen könnte wie mit einem Text. Überlegt doch mal... Eine Sprache hat eine Grammatik, mit der sich die syntaktisch richtigen Sätze der Sprache beschreiben lassen. Für die Wörter greift man dann auf ein Lexikon zu. Habt Ihr schon einmal das Wort "Bildsprache" gehört?

A: Aha, Bildsprache. Und davon willst du jetzt eine Grammatik aufstellen? Daß ich nicht lache... (wendet sich ab)

B: Bevor wir uns jetzt streiten schlage ich folgendes vor: Du erzählst uns in groben Zügen, was du über's Filmemachen weißt, (schaut zu C) du erzählst uns, was man für eine 3D-Animation braucht, und ich erkläre euch die Sache mit der Struktur und Bildsprache. Und dann schauen wir, wie wir das alles unter einen Hut bekommen.

Kapitel 2

Gestaltung und Aufbau von 3D-Animationen

Grundlegend für die Konzeption der vorliegenden Arbeit ist die Überlegung, daß sich die Gestaltung von 3D-Animationen aus mehreren völlig verschiedenen Blickrichtungen betrachten läßt. Jede dieser Betrachtungsweisen ist dabei für die Erklärung bestimmter Teilaspekte hilfreich, ähnlich der Beschreibung des Phänomens Licht als Welle oder Teilchen in der Physik. Keine der Beschreibungen kann für sich allein genommen alle Aspekte adäquat fassen. Durch die Wahl des passenden Beschreibungsmodells kann jedoch jedes Phänomen für sich genommen erklärt werden. Diese Überlegung spiegelt sich direkt im Aufbau des folgenden Kapitels wieder. Führt man beispielsweise die in der Einleitung diskutierte Metapher der virtuellen Kamera logisch weiter, so wird aus der Gestaltung einer Animation das Drehen eines Films. Diese Sichtweise ist sehr gut geeignet, um bestimmte Vorgehensweisen bei der Kameraführung oder Beleuchtung der Modellwelten zu erklären und sie liefert die notwendige Terminologie zur Beschreibung bestimmter Filmtechniken.

Zur Beschreibung der konkreten Darstellung von Animationen am Rechner ist die Sichtweise der Computergraphik am besten geeignet. Sie beschäftigt sich mit der Darstellung dreidimensionaler Objekte, ihrer Eigenschaften und ihrer Bewegungen und beschreibt die dabei auftretenden geometrischen und berechnungstechnischen Phänomene. Die Computergraphik bietet für viele Probleme vereinfachte Lösungen oder Näherungslösungen, eine etablierte Terminologie und effiziente Verfahren. Betrachtet man eine Animation unter den Aspekten ihres formalen Aufbaus in Analogie zur Sprache, so gelangt man schließlich zur strukturellen Sicht, die beispielsweise Syntax, Semantik oder kommunikationstheoretische Strukturierungsprinzipien einer Animation beschreibt. Dieser Ansatz ermöglicht, das Zusammenwirken einer Animation mit anderen Präsentationsteilen im multimedialen Kontext zu beschreiben. Aufgrund struktureller Analysen kann der informative Inhalt einer Animation formal beschrieben werden.

Nach der Einführung der drei Sichtweisen mit ihrer zugehörigen Terminologie in den folgenden Abschnitten werde ich in den weiteren Kapiteln – je nach Bedarf – zwischen diesen Sichtweisen wechseln und jeweils die Position einnehmen, die ein bestimmtes Phänomen am besten erklärt. Diese Vorgehensweise verdeutlicht sehr gut die Grundidee, einen Ansatz zu entwickeln, der alle drei Betrachtungsweisen zur Lösung der gestellten Aufgabe ausnutzt.

2.1 Filmtechnische Sicht

Die Realisierung eines Filmes läuft typischerweise in mehreren Schritten ab. Hierbei arbeiten meist mehrere Menschen zusammen, die jeweils für Teilaspekte des Filmes verantwortlich sind.

2.1.1 Filmidee

Am Anfang eines wie auch immer gearteten Filmes steht (wie am Anfang eines Textes oder Bildes) eine Idee. Bei Spielfilmen ist dies die Handlungsidee, bei journalistischen Berichten die Information, die vermittelt werden soll. Diese Idee sollte sich (laut [Schult & Buchholz, 93]) stets in wenigen einfachen Sätzen formulieren lassen und den Inhalt des geplanten Filmes kurz und prägnant beschreiben. Auch in [Field, 91] wird angehenden Drehbuchautoren deshalb nahegelegt, ihre Idee in wenigen kurzen Sätzen darzulegen:

> Reduzieren Sie Ihre Idee auf eine Figur und die Handlung. Schreiben Sie nicht mehr als drei oder vier Sätze. ([Field, 91], S.23)

Eine solche Formulierung der Filmidee enthält keine Angaben über die konkrete filmtechnische Umsetzung. Diese soll ausdrücklich erst später ausgearbeitet werden, da sie im Anfangsstadium einer prägnanten, klaren Darstellung der Idee abträglich wäre.

2.1.2 Treatment, Filmplan, Storyboard und Drehbuch

Nachdem die Handlungsidee eines Filmes feststeht, wird ihre filmtechnische Umsetzung in mehreren Stufen immer weiter ausgearbeitet. Eine detaillierte Beschreibung dieses Prozesses findet sich in Lehrbüchern wie [Field, 91, Schult & Buchholz, 93]. Zunächst wird ein sogenanntes *Treatment* geschrieben, eine genauere Ausarbeitung der Ideenskizze, die Angaben über den Spannungsverlauf und die einzelnen Darstellungsschritte des Films enthält. Der *Filmplan* präzisiert die Angaben des Treatments um genauere Angaben zu Dauer, Drehort und filmtechnischer Umsetzung bestimmter Inhalte. Das *Storyboard* schließlich legt die einzelnen Kameraeinstellungen des Filmes fest und enthält meist Skizzen der geplanten Bilder. Im *Drehbuch* ist dann der gesamte Ablauf des Filmes auf hohem Detailniveau spezifiziert. Technische Details, die wichtig für die Gesamtwirkung sind, werden genau beschrieben, während andere, unwichtigere, der Entscheidungsfreiheit des Regisseurs überlassen sind.

2.1.3 Regie

Der Regisseur koordiniert nun vor Ort das Drehen der Filmteile. Er füllt Spezifikationslücken des Drehbuches auf oder modifiziert sogar die dort gemachten Angaben nach Maßgabe der jeweiligen Situation. Die Hauptaufgabe des Regisseurs ist es, die Arbeit von Kameramann, Beleuchter und Schauspielern zu koordinieren. Dies setzt voraus, daß er über all diese Bereiche ein grundlegendes Wissen hat und in der Lage ist, die notwendigen Handlungen seiner Teammitglieder genau genug zu beschreiben.

Eine weitere Aufgabe der Regie ist es, die Konsistenz, beziehungsweise Kohärenz der einzelnen Aufnahmen untereinander zu sichern. Schauspieler müssen an verschiedenen Drehtagen einer Szene exakt die gleiche Bekleidung tragen, Hemden und Jacken müssen gleich weit geknöpft sein, und bei einer Filmhandlung im Mittelalter darf natürlich keiner der Schauspieler eine Armbanduhr tragen. Diese Aufgaben sind auch meist an Regieassistenten delegiert.

2.1.4 Kamera

Die Verantwortung des Kameramanns besteht darin, die Vorgaben des Drehbuchs und des Regisseurs in Bilder umzusetzen. Er entscheidet letztendlich über die exakten Kamerapositionen und Einstellungen sowie über Kamerafahrten und ähnliches. Eine *Kamerafahrt* ist die Veränderung der Kameraposition während des Drehens. Hierbei bedeutet Fahrt nicht zwangsläufig eine Bewegung auf Rollen, sondern ganz allgemein eine Bewegung. Eine *Kranfahrt* beispielsweise wirkt eher wie ein Flug auf einer (fast) beliebigen Bahn im Raum. Ein *Schwenk* ist eine Drehung der Kamera, in der Regel um ihre vertikale Achse bei gleichbleibender Kameraposition. Dies entspricht bei *subjektiver*, das heißt dem Blick eines Betrachters folgender Kamera einem Drehen des Kopfes.

Ein *Zoom* ist eine Veränderung der Objektivbrennweite der Kamera. Hierdurch verändert sich der Bildwinkel des Objektivs und es entsteht ein Eindruck, der einer Kamerafahrt auf die Szenerie zu (*Zufahrt*) oder davon weg (*Rückfahrt*) gleicht. In Wirklichkeit verändert sich jedoch beim Zoomen die Kameraperspektive nicht, sondern lediglich der Bildausschnitt, während bei einer Kamerafahrt die Perspektive wechselt. Zooms sind zwar technisch einfacher umzusetzen als Fahrten, erwecken jedoch teilweise durch die fehlende Perspektivenveränderung einen unnatürlichen Eindruck.

Durch *Fokussieren* der Kamera bei weit offener Blende verlagert sich die *Schärfeebene* des Bildes. Da ein Kameraobjektiv nicht in der Lage ist, alle Objekte der abgebildeten Szenerie gleich scharf darzustellen, sondern nur die, die in einer bestimmten Ebene, der *Schärfeebene* liegen, können Personen oder Gegenstände optisch hervorgehoben werden, indem ihr Hintergrund in der Unschärfe verschwimmt.

Abgesehen von diesen Veränderungen der Kameraeinstellungen oder Kameraposition ist es auch Aufgabe des Kameramannes, für einen gelungenen Aufbau der Bilder zu sorgen. So müssen für den Film wichtige Gegenstände oder Personen der Intention entsprechend im Bild positioniert werden. Durch die Wahl einer bestimmten Objektivbrennweite können die Größenverhältnisse der dargestellten Objekte bewußt unterstrichen oder verfälscht werden. Durch die Wahl der Blende können Gegenstände oder Schauspieler von ihrem Hintergrund losgelöst werden. Eine systematische Aufstellung solcher Kamerapositionen und -einstellungen findet sich beispielsweise in [Arijon, 76].

2.1.5 Licht

Die Beleuchtung eines Filmes sollte (nach [Schult & Buchholz, 93, Dunker, 93]) verschiedenen Kriterien genügen. Sie sollte *Räumlichkeit darstellen*, *Strukturen hervorheben*, *Atmosphäre schaffen* und außerdem bestimmte Anforderungen an die

Kontinuität erfüllen. Für die Beleuchtung von Filmszenen gelten mit gewissen Einschränkungen ähnliche Regeln wie für die Beleuchtung photographischer Aufnahmen. Eine Beleuchtung, die für ein Portrait geeignet ist, paßt meistens auch für eine entsprechende Filmeinstellung. Das Darstellen der Räumlichkeit läßt sich am ehesten durch den bewußten Einsatz der Schatten erreichen, was auch für die Betonung der Strukturen gilt. Welche Atmosphäre eine Beleuchtung erzeugt, hängt von ihrer prinzipiellen Zusammensetzung ab. So kann beispielsweise eine sehr neutrale Beleuchtung aus einer Nachbildung des Tageslichtes bestehen, also aus einer weit entfernten Hauptlichtquelle von oben oder seitlich oben (Sonne) und einer diffusen Ausleuchtung von allen Seiten, die die Schatten aufhellt, ähnlich dem vom Himmel reflektierten Licht in der freien Natur.

2.1.6 Schauspieler und Handlung

Die Schauspieler im Spielfilm oder die gezeigten Personen, Tiere oder Gegenstände im Dokumentarfilm sind der Ausgangspunkt der gezeigten Handlungen. Nach ihnen und ihren Aktionen richten sich Kameraführung, Beleuchtung und Schnitt. Bei einem Spielfilm sind die Handlungen dieser Personen im Drehbuch festgelegt und werden auf Anweisung des Regisseurs ausgeführt, während im Dokumentarfilm lediglich Personen, Dinge und Handlungen gezeigt werden, die auch ohne den Film in der Natur so existieren oder ablaufen würden.

Das Drehbuch beschreibt zwar in beiden Fällen die Darstellung der Handlung durch Kamera und Beleuchtung, jedoch müssen die beiden genannten Arten von Handlung grundsätzlich unterschieden werden. Die Aktionen der Schauspieler eines Spielfilmes werden im Drehbuch festgelegt und dadurch bestimmt, während die Handlung eines Dokumentarfilmes durch die reale Welt festgelegt ist und das Drehbuch sie nur festhält, um Kamera und Licht darauf abzustimmen und die Handlung so dramaturgisch aufzubereiten.

2.1.7 Schnitt und Filmstruktur

Ein Film ist – bedingt durch seine Ezeugungsprozeß – inhaltlich und physikalisch in bestimmter Weise gegliedert. Zunächst enthält ein Film in der Regel mehrere *Sequenzen.* Sequenzen sind Handlungsteile, die inhaltlich oder chronologisch zusammenhängen. Diese Sequenzen bestehen jeweils aus einer oder mehreren *Szenen.* Eine Szene ist ein Teil eines Filmes, der an einem bestimmten Ort spielt, also nicht nur inhaltlich, sondern auch räumlich begrenzt ist. Szenen sind wiederum aus einer oder mehreren *Einstellungen* zusammengesetzt. Eine Einstellung bezeichnet ein Stück Film, das zusammenhängend gefilmt wurde.

Diese Filmstücke müssen physikalisch zum Gesamtfilm zusammengesetzt werden, und das geschieht nach Abschluß der Dreharbeiten beim Schneiden. Zwischen den Einstellungen liegen bestimmte Arten von Übergängen, die mit *Schnitt* oder *Blende* bezeichnet werden. Ein Schnitt bedeutet das direkte Aneinanderkleben zweier Filmstücke, was beim fertigen Film den Eindruck eines Sprungs der Kamera zu einer neuen Position erweckt. Wird zwischen die beiden Filmstücke ein Stück klarer oder schwarzer Film eingefügt, so spricht man von einem *Blitzer* oder *Schwarzbild.* Dieser Effekt wird vom Zuschauer nicht bewußt wahrgenommen, vermittelt aber

unterbewußt, daß nach dem Schnitt etwas Neues im Film beginnt. Eine Blende ist das langsame Aus- oder Einblenden des Bildes, also der allmähliche Übergang zu einem weißen oder schwarzen Bild oder eines Bildes ins nächste.

2.2 Computergraphische Sicht

Nachdem im vorangegangenen Abschnitt die Terminologie und der Handlungsablauf beim Erstellen eines realen Filmes eingeführt wurden, soll nun beschrieben werden, wie eine 3D-Computeranimation herkömmlicherweise erzeugt wird, welche Informationen verarbeitet und welche Konzepte und Verfahren benutzt werden. Viele dieser Verfahren sind in den gängigen Standardwerken zur Computergraphik, wie [Watt & Watt, 92] und [Foley et al., 96] detailliert beschrieben.

2.2.1 Modelldaten

Ausgangspunkt einer dreidimensionalen Computergraphik sind die rechnerinternen Beschreibungen der darzustellenden Gegenstände, die sogenannten *Modelldaten.* Diese geben die Form, Farbe, Transparenz, Oberfläche und weitere Eigenschaften der betreffenden Gegenstände an. Zur Beschreibung der Form dreidimensionaler Objekte gibt es prinzipiell verschiedene Ansätze.

2.2.1.1 Constructive Solid Geometry

Komplexere geometrische Körper können durch die logische Verknüpfung geometrischer Primitive wie Quader, Zylinder und Kugel zusammengesetzt werden. Dieses als *CSG* bezeichnete Verfahren erlaubt die Addition, Subtraktion oder die exklusive Oder-Verknüpfung einzelner Primitive zu neuen Objekten. Obwohl es einige Vorteile bei der Bildberechnung besitzt, ist dieses Verfahren doch für die Modellierung in der wirklichen Welt auftretender Gegenstände oft unzureichend. CSG-Modelle sind nicht nur in der Lage, die Form und Oberfläche der modellierten Gegenstände, sondern auch ihr Volumen sehr gut zu beschreiben.

2.2.1.2 Boundary Representation

Die nächstkomplexere Methode, dreidimensionale Körper zu beschreiben, ist die Modellierung durch umgebende Flächen, *Boundary Representation* oder *B-rep* genannt. Ein Würfel kann so beispielsweise aus sechs ebenen Flächen modelliert werden. Je nach der Komplexität der zur Beschreibung verwendeten Flächenarten unterscheidet man flache, elementare und erweiterte B-reps. Modelle, die im CSG-Verfahren modelliert wurden, lassen sich in die B-rep Darstellung übertragen, jedoch ist das Volumen eines Körpers aus einem allgemeinen B-rep-Modell nicht mehr so einfach zu berechnen, es sei denn, das Modell genügt gewissen Geschlossenheitsbedingungen. B-rep ist derzeit die am weitesten verbreitete Modellierungsart für CAD-Systeme.

2.2.1.3 Polygonmodelle

Polygonmodelle bestehen aus einer Menge von Punkten im dreidimensionalen Raum und einer Menge von planaren Kantenzügen oder *Polygonen* zwischen diesen Punkten. Diese Polygone können beliebig im Raum angeordnet sein und damit beliebige Formen realer Gegenstände umschreiben. Rundungen der Gegenstände werden dabei jedoch in mehr oder weniger große gerade Stücke unterteilt, da die Polygone selbst jeweils flach sind. Ein Polygonmodell ist also nicht in der Lage, runde Formen zu beschreiben. Trotzdem können die Polygone beim Schattieren so eingefärbt werden, daß sie gewölbt aussehen (Siehe Abschnitt 2.2.4).

Volumina können mit Polygonmodellen (im Gegensatz zu CSG) nicht korrekt modelliert werden, da nicht von vornherein gewährleistet ist, daß ein Polygonnetz einen Raum vollständig umschließt. Sieht man von diesem Mangel ab, so bieten Polygonmodelle jedoch eine sehr vielseitige und einfache Form der Repräsentation, da aus ihnen auch verschiedene Darstellungsformen, wie z. B. Drahtgitterdarstellungen oder schattierte Flächen abgeleitet werden können.

2.2.1.4 Parametrische Oberflächen

Nimmt man die Punkte im dreidimensionalen Raum nicht als Eckpunkte für Polygone, sondern als Stütz- oder Kontrollpunkte eines Netzes interpolierter Kurven, so erhält man parametrische Oberflächenbeschreibungen. Benutzt man zur Interpolation beispielsweise Bezier-Kurven, so heißen die entstehenden Flächenstücke *Bezierpatches*. Mit Hilfe solcher parametrischer Darstellungen lassen sich nahezu beliebige Formen beschreiben und die modellierten Rundungen werden im Gegensatz zu Polygonmodellen nicht von vorneherein in gerade Stücke zerbrochen. Außerdem verformt sich das Modell bei der Verschiebung eines Kontrollpunktes mehr oder weniger weich und elastisch im Gegensatz zu Polygonmodellen.

Je nach Art der Interpolationsfunktion unterscheidet man verschiedene Typen parametrischer Oberflächenbeschreibungen, die jedoch allesamt die oben genannten Eigenschaften haben. Parametrische Oberflächen sind vor allem zur Erstellung hochwertiger Graphiken geeignet, bei denen es auf eine genaue Approximation der Form der Gegenstände ankommt.

2.2.1.5 Objekthierarchie

Oft liegen nun die Modelldaten nicht einfach als eine Menge einzelner Objekte vor, sondern die Objekte sind nach funktionalen oder inhaltlichen Kriterien gruppiert. Bei Modellen technischer Geräte ist es beispielsweise oft sinnvoll, Baugruppen, die funktional oder zumindest mechanisch zusammenhängen, auch in der Modellwelt zu gruppieren. Dadurch ist es beispielsweise leichter, die gesamte Baugruppe mit einem Befehl zu bewegen oder ihre Eigenschaften zu verändern. Diese Gruppierung kann in mehreren Stufen erfolgen und ergibt eine Objekthierarchie, die nach dem Prädikat *'ist Teil von'* oft auch als *Teil-von-Hierarchie* bezeichnet wird.

2.2.1.6 Abstraktionen

Zur Erzeugung von Animationen ist nicht immer das detaillierteste Modell das beste. Ist ein Objekt im Bild sehr klein dargestellt oder für die Bildaussage nicht wichtig, so wäre es Verschwendung, trotzdem seine voll detaillierte Beschreibung zur Bilderzeugung zu verwenden. Zu diesem Zweck nimmt man oft vereinfachte Modelle, die entweder von Hand oder automatisch aus den detaillierten Modellen abgeleitet werden. Die automatische Ableitung solcher Modelle wird beispielsweise in [Krüger, 98, Krüger, 95, Feiner, 85] beschrieben. Welcher Grad der Abstraktion gewählt werden kann, ist in [Butz & Krüger, 97] untersucht, und wie die abstrahierten Modelle geschickt für Animationen eingesetzt werden können, wird in [Butz & Krüger, 96] ausgeführt. Bestehende Systeme, die mehrere Detaillierungsgrade der Modelldaten verarbeiten, sind beispielsweise VRML-Browser [SGI, 96] und einige CAD-Systeme [Rossignac & Borrel, 93].

2.2.1.7 Materialeigenschaften

Außer ihrer Form werden für die Darstellung der 3D-Modelle noch andere Informationen benötigt. Im einfachsten Falle ist dies die Farbe der darzustellenden Linien oder Flächen und ihre Transparenz. Viele Graphiksysteme verarbeiten aber komplexere *Materialbeschreibungen*. Im Falle der Graphikbibliothek OpenGL[1] sind dies beispielsweise die Reflexionskoeffizienten und -farben für ambiente, diffuse und Glanzreflektion. Das RenderMan[2] Interface ([Upstill, 90]) sieht sogar eine prozedurale Beschreibung von Materialeigenschaften in einer eigenen Beschreibungssprache vor. Diese prozeduralen Beschreibungen, die sogenannten *Shader*, können die Farbverteilung auf einer Materialoberfläche beschreiben, ihre Form verändern (*bump mapping*, siehe Abbildung 2.3 auf Seite 20), sie können atmosphärische Effekte wie Nebel oder Wolken beschreiben und das Verhalten von Lichtquellen angeben. In welcher Form Materialeigenschaften angegeben werden, ist jeweils vom verwendeten Graphiksystem abhängig.

2.2.2 Bewegungsbeschreibungen

Um die in der Animation auftretenden Objekte bewegen zu können, braucht ein Animationssystem geeignete Beschreibungen der Objektbewegungen. Auch hierzu gibt es unterschiedlich komplexe Ansätze.

2.2.2.1 Elementare Bewegungen und Trajektorien

Im einfachsten Falle handelt es sich bei den Bewegungen um elementare *Rotationen* oder *Translationen*. Diese geben die Positions- oder Orientierungsveränderung eines Objektes in der Zeit an. Einfache Bewegungen, wie beispielsweise ein fahrendes Auto mit sich drehenden Rädern sind damit sehr leicht zu beschreiben. Die nächstkomplexere Darstellung von Bewegungen ist die Beschreibung der zugehörigen *Trajektorien* durch interpolierte Kurven, wie zum Beispiel *splines*. Gibt man eine Trajektorie für

[1] OpenGL ist ein eingetragenes Warenzeichen der Firma Silicon Graphics
[2] RenderMan ist ein eingetragenes Warenzeichen der Firma Pixar

die Position eines Objektes im Raum an und zwei für seine Orientierung, so lassen sich damit nahezu beliebige Bewegungsbahnen im Raum beschreiben.

2.2.2.2 Hierarchische Bewegungsbeschreibungen

Zur Modellierung noch komplexerer Bewegungsabläufe an mechanisch verbundenen Teilen werden schließlich hierarchische Bewegungsbeschreibungen benötigt, die die Bewegungen der Objekte in Abhängigkeit von ihrem mechanischen Zusammenbau beschreiben. Die Hierarchie der Bewegungen kann sich beispielsweise nach der Hierarchie der Objekte (Abschnitt 2.2.1.5) richten. Solche Formalismen ermöglichen es aber auch, zusammengesetzte Objekte durch das Bewegen eines Teils zu animieren, wie z.B. den Arm einer Gliederpuppe durch das Heben der Hand. Nützlich sind diese Modelle vor allem zur physikalisch richtigen Modellierung technischer Apparaturen, bei der Simulation physikalischer Vorgänge oder zur Ansteuerung virtueller Agenten auf sehr hoher Ebene ([Badler et al., 90, Badler et al., 93], siehe auch Abschnitt 3.4).

2.2.2.3 Veränderung anderer Objekteigenschaften

Neben ihrer Position und Orientierung können Objekte auch ihre in Abschnitt 2.2.1.7 beschriebenen Eigenschaften verändern. Da diese Veränderung letztendlich nur eine Manipulation der Parameter ist, stellt dies mathematisch keinen Unterschied zur Veränderung der Position dar. Statt der Parameter x, y und z für die Position werden die Parameter r, g und b für die Farbe oder Transparenz manipuliert. Dies kann mit den gleichen mathematischen Mitteln erfolgen, also durch lineare Interpolation oder andere Interpolationsfunktionen und eine solche Veränderung von Objekteigenschaften kann auf die gleiche Art und Weise beschrieben werden wie Objektbewegungen.

2.2.3 Lichtquellen

Will man dreidimensionale Modelle von Gegenständen in einer Weise darstellen, die deren Aussehen in der Natur nahekommt, so ist es hilfreich, die physikalischen Phänomene Licht, Reflexion und Absorption im Rechner zu modellieren. Damit ein Gegenstand Licht reflektieren kann, müssen also in die Bildberechnung eine oder mehrere Lichtquellen miteinbezogen werden. Wir unterscheiden dabei prinzipiell mehrere unterschiedliche Arten der Lichtmodellierung.

2.2.3.1 Ambientes Licht

Das sogenannte *ambiente* Licht ist eine Lichtart, die so in der Natur nicht vorkommt. Es beschreibt eine überall und aus allen Richtungen vorhandene Strahlung, die auf den dargestellten Körpern je nach deren Oberflächeneigenschaften eine gleichmäßige Helligkeit hervorruft. Dieses Licht ist mit wenig Aufwand zu berechnen und vor allem dazu nützlich, auf einfache Art und Weise Schattenpartien aufzuhellen. Ein starker Einsatz der ambienten Lichtquelle führt jedoch in aller Regel zu unrealistisch wirkenden Darstellungen.

2.2.3.2 Entferntes Licht

Eine typische entfernte Lichtquelle in der Natur ist die Sonne. Ihr Licht trifft in nahezu parallelen Strahlen auf die Gegenstände, wirft relativ scharf umgrenzte Schatten und nimmt in seiner Intensität über irdische Entfernungen fast nicht ab. Diese nahezu konstante Helligkeit rührt daher, daß der Abstand zweier Gegenstände auf der Erde im Verhältnis zu ihrem Abstand zur Sonne immer sehr klein ist. Da die Lichtintensität mit dem Quadrat der Entfernung von der Lichtquelle abnimmt, bedeutet dies eine sehr kleine Schwankung der Helligkeit bei diesen verhältnismäßig kleinen Abständen.

Das in der Computergraphik verwendete mathematische Modell idealisiert die Situation nun dahingehend, daß die Strahlen genau parallel laufen, die Lichtquelle unendlich weit entfernt und von der Größe Null und die Lichtintensität an allen Stellen des Raumes konstant ist. Trotzdem können entfernte Lichtquellen einen sehr realistischen Lichteindruck erzeugen, da wir an ihre Eigenschaften vom Sonnenlicht her gewöhnt sind.

2.2.3.3 Punktlichter

Punktförmige Lichtquellen in endlicher Entfernung von den beleuchteten Gegenständen können am ehesten mit den künstlichen Lichtquellen unserer Umgebung verglichen werden. Eine Glühbirne hat (bei mittlerem Abstand) ein verhältnismäßig kleines Volumen. Die Intensität des von ihr ausgesandten Lichts nimmt (wie bei jeder Strahlung) mit dem Quadrat der Entfernung ab, und da wir es nun mit relativ kleinen Entfernungen zu tun haben, ist diese Abnahme in der Regel auch sichtbar. Ein Gegenstand, der doppelt so weit entfernt von der Lichtquelle ist, wie ein anderer, bekommt also nur ein Viertel des Lichts ab. Das mathematische Modell vereinfacht die Realität wieder derart, daß die Lichtquelle keine Ausdehnung hat. Obwohl diese Lichtart von der Berechnung her aufwendiger ist, ist ihr Einsatz zur Erzeugung realistischer Bilder manchmal unumgänglich, wenn beispielsweise künstliche Lichtquellen in kurzer Entfernung dargestellt werden sollen.

2.2.3.4 Flächenlichter

Beseitigt man nun die letzte mathematische Vereinfachung und ordnet den Lichtquellen eine räumliche Ausdehnung zu, so erhält man sogenannte *Flächenlichter*. Diese Lichtquellen entsprechen den eben genannten Glühbirnen physikalisch exakter und haben einige grundlegende Eigenschaften, die für ihren naturnahen Lichteindruck verantwortlich sind. Flächenlichtquellen werfen beispielsweise weich umrandete Schatten, wie es auch jede reale (also natürliche oder künstliche) Lichtquelle (mehr oder weniger stark) tut. Dadurch, daß bestimmte Positionen im Raum nur von einem Teil der Lichtquelle, die ja jetzt eine Ausdehnung hat, beleuchtet werden, liegen sie im sogenannten *Halbschatten* und sind deshalb nur teilweise beleuchtet. Außerdem erzeugen Flächenlichter auf reflektierenden Materialien Reflexionen größerer Ausdehnung, sogenannte *Reflexstreifen*, die in der Photographie gerne zur optischen Hervorhebung solcher Oberflächen eingesetzt werden. Das ideale photographische Äquivalent zum Flächenlicht ist die Lichtwanne oder der Reflexschirm im Portraitstudio.

2.2.3.5 Scheinwerfer

Eine Lichtquelle, die zu gestalterischen Zwecken gerne benutzt wird, ist der Scheinwerfer. Das mathematische Modell des Scheinwerfers in der Computergraphik beschreibt die Position der Lichtquelle, ihre Leuchtrichtung und den Winkel, der voll vom Licht erfaßt wird. Außerhalb des so gebildeten Lichtkegels nimmt die Leuchtstärke kontinuierlich ab und durch einen zweiten Winkel wird beschrieben, wann sie die Stärke Null erreicht. Die Lichtintensität innerhalb des Kegels nimmt wie die der Punktlichtquelle mit dem Quadrat der Entfernung von der Lichtquelle ab. Scheinwerfer heben die von ihnen erfaßten Objekte optisch hervor und können so gezielt unter gestalterischen Aspekten eingesetzt werden.

2.2.3.6 Schatten

Die physikalisch richtige Berechnung des Schattenwurfes bei den eben beschriebenen Lichtquellen ist je nach dem eingesetzten Bilderzeugungsverfahren recht komplex. Um den Rechenaufwand zu reduzieren, kann man deshalb bei den meisten Graphiksystemen bestimmen, welche Lichtquellen im erzeugten Bild Schatten werfen sollen und welche nicht. Da in der Natur meistens eine Lichtquelle dominiert und die Schatten der anderen Lichtquellen kaum wahrnehmbar sind, kommt man auch in der Computergraphik meistens damit aus, nur die Schatten der stärksten Lichtquelle zu berechnen, ohne dabei zu sehr an Realismus zu verlieren (Siehe hierzu auch [Brugger, 93]).

2.2.4 Bilderzeugung

Zur Erzeugung der Bilder aus den beschriebenen Daten gibt es verschiedene Verfahren, die bestimmte Grundtatsachen gemeinsam haben. Alle Verfahren projizieren Punkte des dreidimensionalen Raumes in eine zweidimensionale Bildebene. Dies geschieht mittels einer virtuellen Kamera, oder mathematisch gesehen in einer entsprechenden Matrix. Je nach Art der Projektion werden dabei entferntere Gegenstände kleiner abgebildet, wie es auch eine Fotokamera oder das menschliche Auge tun (*Zentralprojektion* oder *perspektivische* Projektion), oder sie behalten ihre Größe bei (*Parallelprojektion*). Während die Zentralprojektion also ein naturgetreueres Bild der dreidimensionalen Szenerie liefert, ist die Parallelprojektion eher vorzuziehen, wenn es um die genaue Wiedergabe der Größenverhältnisse geht. Im Zusammenhang mit der Erzeugung realistischer Animationen wollen wir deshalb im folgenden eine perspektivische Projektion voraussetzen. Nach der Projektion in die Bildebene wird nun ein Pixelbild berechnet. Dies kann auf verschiedene Arten erfolgen.

2.2.4.1 Drahtgitterdarstellung

Eine Drahtgitterdarstellung stellt die Kanten eines Polygonmodells als Linien im Bild dar. Die Polygone des Modells ergeben also auch wieder Polygone auf dem Bildschirm, jedoch durch die Projektionsmatrix verformt. Um eine realistische Darstellung zu erzielen, muß außerdem ermittelt werden, welche der Polygone mit einer natürlichen Kamera sichtbar wären und welche durch andere verdeckt sind. Dies erfolgt bei einer Drahtgitterdarstellung meist durch eine Sortierung der Polygone nach

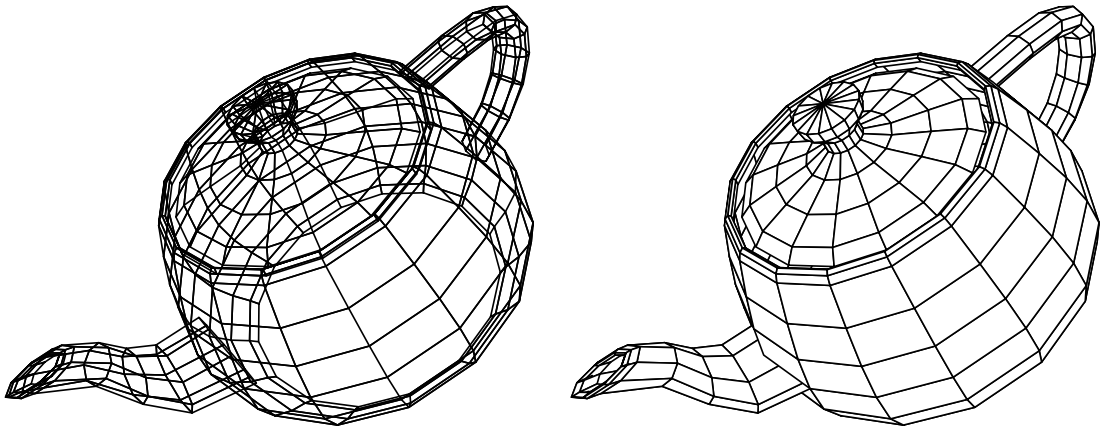


Abbildung 2.1: Drahtgitterdarstellung und Berechnung verdeckter Polygone

ihrer Tiefe im Raum (von der Kamera aus gesehen). Dieses Verfahren liefert zwar in manchen Situationen unrichtige Ausgaben, ist aber dafür schnell zu berechnen. Manche Systeme, die Drahtgittermodelle darstellen, können die dargestellten Linien zusätzlich auch unterschiedlich einfärben.

2.2.4.2 Schattierte Darstellung

Einen wesentlich realistischeren Eindruck erhält man bereits, wenn die Polygone des Modells je nach ihrer Lage zum Licht, nach ihrer Eigenfarbe und ihrem Reflexionsverhalten als farbige Flächen dargestellt werden. Wird jedem Polygon genau eine Farbe zugeordnet, so spricht man von *flacher* Schattierung (englisch *flat* oder *constant shading*). Die Entscheidung über die Sichtbarkeit der Polygone kann dabei entweder auf Polygonebene erfolgen wie bei der Drahtgitterdarstellung beschrieben, oder auch auf Pixelebene mit dem sogenannten *Z-buffer* Verfahren (für eine Beschreibung siehe [Watt & Watt, 92], S.22).

Der Z-buffer-Algorithmus benötigt einen zusätzlichen Speicher in Größe des erzeugten Bildes, läßt sich aber sehr effizient ausführen, so daß eine relative schnelle Bildberechnung möglich ist. Manche Graphiksysteme greifen sogar auf einen in der Rechnerhardware, meist in der Graphikkarte implementierten Z-buffer zurück und entlasten damit den Hauptprozessor von der Bilderzeugung. Beispiele hierfür sind bestimmte Graphikeinheiten der Firmen Silicon Graphics und SUN, aber auch die immer weiter verbreiteten preiswerten 3D-beschleunigten PC-Graphikkarten. Nimmt man zur Bildberechnung noch Information über die *Flächennormalen* der Polygone hinzu, und zwar jeweils in deren Eckpunkten, so läßt sich diese Normale und damit die Orientierung der Oberfläche eines Polygons über seine gesamte Fläche interpolieren und wir erhalten statt der Kanten weiche Übergänge zwischen den einzelnen Flächen. Zur Berechnung dieser Interpolation gibt es zwei gängige Verfahren, das 1971 vorgestellte und nach seinem Erfinder benannte *Gouraudshading* und das 1975 daraus weiterentwickelte etwas aufwendigere *Phongshading*, ebenfalls nach seinem Autor benannt. Diese Verfahren liefern in Verbindung mit Polygonmodellen bereits optisch recht ansprechende Ergebnisse und lassen sich zudem meist sehr schnell oder sogar in Hardware berechnen.

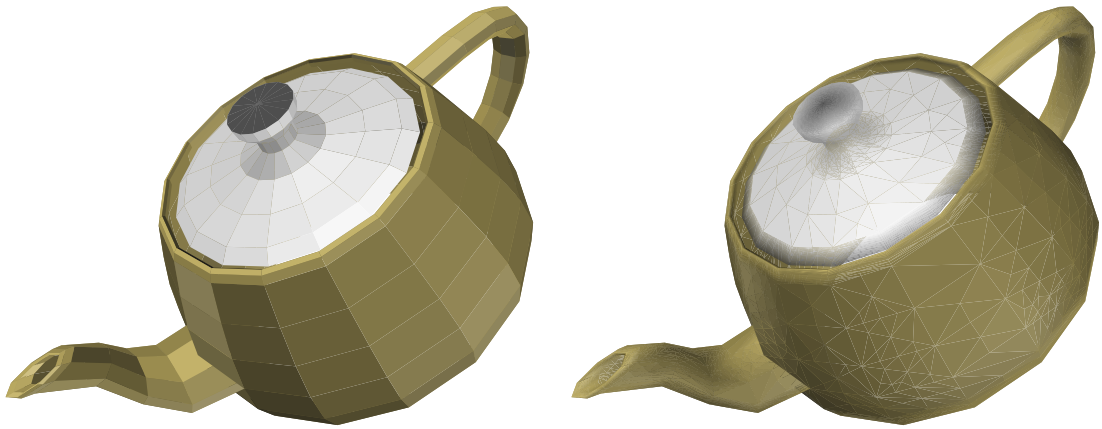


Abbildung 2.2: Flat shading und Phong shading

2.2.4.3 Raytracing

Eine noch genauere Nachbildung der virtuellen Kamera ist in einem um 1980 eingeführten Verfahren, dem sogenannten *Raytracing* realisiert. Die Grundidee ist die, einen Blickstrahl vom Auge des Betrachters aus durch jedes Pixel des Bildes in die dreidimensionale Modellwelt zu senden und zu verfolgen, auf welches Objekt er als erstes trifft, beziehungsweise wohin er von dort aus reflektiert wird. Von den Punkten des Objektes aus, auf die der Blickstrahl trifft, werden dann Strahlen in Richtung der Lichtquellen berechnet, deren Verfolgung die Information liefert, wie stark der Objektpunkt von dieser Lichtquelle beschienen wird. Die Funktionsweise ist in Abbildung 2.4 links schematisch dargestellt. Die Stärken des Raytracing-Verfahrens

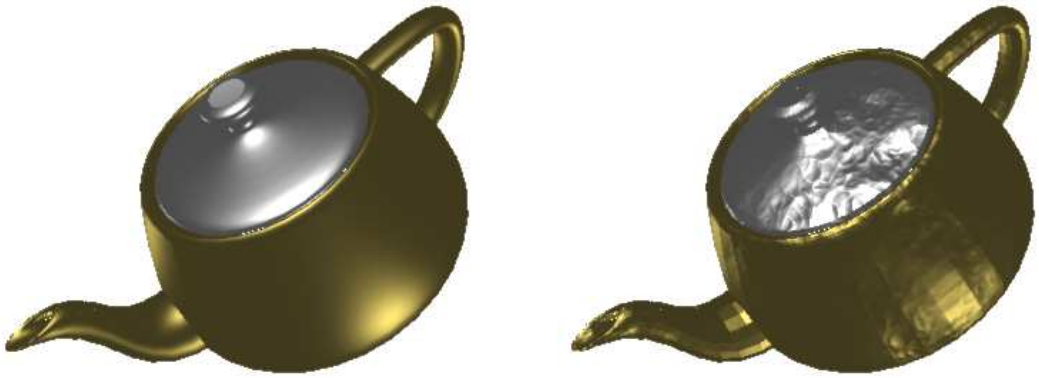


Abbildung 2.3: Raytracing und bump mapping

liegen in der Darstellung von Spiegelreflexionen, Lichtbrechungen und Schatten, es ist jedoch mit einem wesentlich höheren Rechenaufwand verbunden als die schattierte Darstellung, was es zur Darstellung von Animationen in Echtzeit meist ungeeignet macht.

2.2.4.4 Radiosityverfahren

Von einer anderen Grundidee geht das 1984 entwickelte *Radiosityverfahren* aus. Es beschreibt die Energieverteilung innerhalb einer Szenerie, in der die Lichtquellen eine bestimmte Strahlungsenergie aussenden. Die dargestellten Objekte bekommen je nach ihrer Entfernung von den Strahlungsquellen unterschiedlich viel Strahlung ab und senden somit je nach ihrem eigenen Reflektionsverhalten selbst wieder Strahlung aus. In vielen Iterationen über Oberflächenstücke der Objekte wird allmählich ein Gleichgewicht der Energieverhältnisse in der Szenerie berechnet, das die Helligkeiten und Farben der Objekte beschreibt. Diese Energieverteilung im Raum ist von der Position der Kamera unabhängig und braucht daher für eine statische Szenerie mit bewegter Kamera nur einmal berechnet zu werden. Die Funktionsweise ist in Abbildung 2.4 rechts schematisch dargestellt.

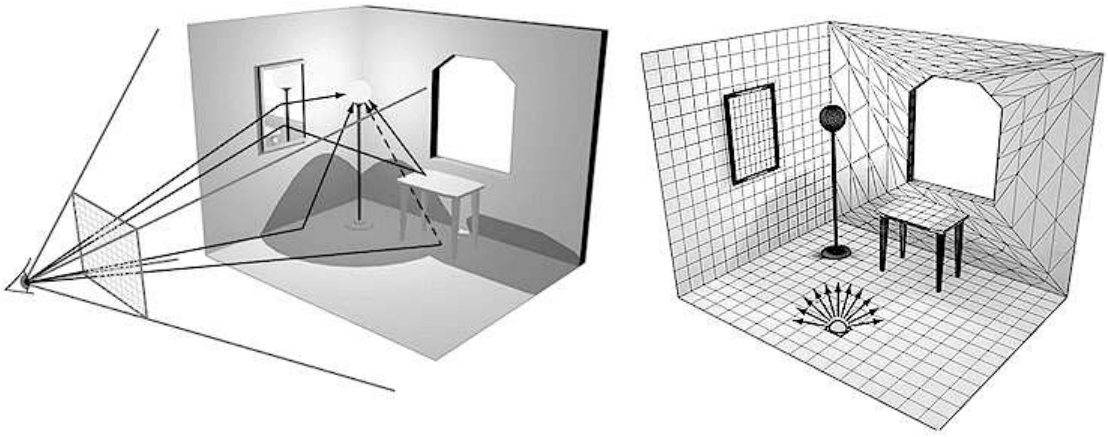


Abbildung 2.4: Funktionsweise des Raytracing und Radiosity Verfahrens

Das Radiosity-Verfahren ist physikalisch motiviert und sehr gut dazu geeignet, diffuse Reflexionen darzustellen. So wirkt beispielsweise ein weißer Gegenstand vor einer roten Wand nicht mehr weiß, sondern mehr oder weniger rosa, da zu dem Licht, das ihn direkt trifft, das von der roten Wand reflektierte rote Licht mit in die Berechnung eingeht. Ein Phänomen, das mit dem Verfahren sehr schlecht modelliert werden kann, sind Glanzreflektionen und Spiegelungen.

2.2.5 Animationsskripte

Um eine 3D-Animation zu beschreiben muß ihr gesamter Ablauf, das heißt die Positionen und Orientierungen der Kamera und der Objekte sowie deren andere Eigenschaften zu jedem Zeitpunkt eindeutig beschrieben werden. Dies für jedes einzelne Bild zu tun ist einerseits mit sehr viel Arbeit verbunden und verbraucht andererseits viel Speicherplatz. Um die für eine Animationsbeschreibung notwendige Datenmenge und den Arbeitsaufwand einzuschränken gibt es verschiedene Verfahren.

2.2.5.1 Mathematische Beschreibung vs. Keyframing

Durch Angabe einer beliebigen mathematischen Beschreibung der gesteuerten Parameter in Abhängigkeit von der Zeit ist deren Wert zu jedem Zeitpunkt genau spe-

zifiziert beziehungsweise berechenbar. Ein schwingendes Pendel kann beispielsweise durch eine Sinusfunktion für seine Orientierung beschrieben werden, ein rollendes Rad durch zwei lineare Funktionen für seine Position und seine Orientierung, und zur Modellierung beliebiger Bewegungen können Splinefunktionen angegeben werden.

Es stellt sich heraus, daß die meisten Bewegungen durch lineare oder Spline-Interpolation beschrieben werden können. Eine naheliegende weitere Vereinfachung der Beschreibung ist also das sogenannte *Keyframing*. Die Position (Orientierung, Farbe, ...) von Objekten, Lichtern und Kamera wird zu bestimmten Zeitpunkten festgelegt und dazwischen linear oder mittels einer Splinefunktion interpoliert. Die spezifizierten Zeitpunkte heißen *Keyframes* und die so aufgebauten Skripte beschreiben mit einem geringen Datenaufwand jeden Zeitpunkt der Animation eindeutig.

2.2.5.2 Flache vs. hierarchische Skripte

Zur Beschreibung der Animation ist es unerheblich, auf welche Art die Steuerbeschreibungen der einzelnen Parameter im Skript organisiert oder sortiert sind, solange ihre zeitliche Zuordnung eindeutig ist. Für den (menschlichen) Autor solcher Skripte ist es jedoch wichtig, bestimmte Parameter nach inhaltlichen Gesichtspunkten zu gruppieren. So ist es beispielsweise leichter, ein Skript zu entwerfen oder zu verstehen, wenn die einzelnen Bewegungen chronologisch aufeinanderfolgend oder nach Objekten sortiert angeordnet sind. Diese Gruppierung kann in mehreren Ebenen erfolgen, indem beispielsweise bei der Bewegung einer Objektgruppe zunächst alle Parameter jedes einzelnen Objektes gruppiert sind, dann all diese Objektbeschreibungen zu einer Beschreibung der Objektgruppe, die schließlich mit den Kamera- und Lichtbeschreibungen im Skript koordiniert ist. Genauso können Bewegungen nach chronologischen Gesichtspunkten in Gruppen paralleler, zeitlich überlappender oder aufeinanderfolgender (sequenzieller) Beschreibungen zusammengefaßt werden. Auch dies kann wieder über mehrere Ebenen hinweg erfolgen und verleiht dem Skript eine interne *hierarchische* Struktur, die sein Verständnis, seine Darstellung und (wie ich in Kapitel 6 zeigen werde) auch seine automatische Generierung vereinfacht.

2.2.6 Geometrische Vereinfachungen

Will man Vorgänge im dreidimensionalen Raum simulieren oder Animationsskripte automatisch generieren, so müssen bestimmte Informationen aufgrund der Modelldaten berechnet werden. Eine übliche Vorgehensweise in dieser Situation ist es, nicht die komplette geometrische Beschreibung der dargestellten Objekte als Berechnungsgrundlage zu verwenden, sondern starke Vereinfachungen davon. Eine mathematisch sehr starke Vereinfachung eines Objektes ist beispielsweise die Beschreibung seines umgebenden Quaders durch Angabe eines Punktes und dreier Vektoren oder seiner umschreibenden Kugel durch Angabe des Mittelpunktes und des Radius (Abbildung 2.5). Aus der umschreibenden Kugel läßt sich mit einer einzigen Vergleichsoperation die Gefahr einer Kollision zweier Objekte ableiten, die umschreibenden Quader sind bei den in der 3D-Graphik häufig auftretenden rechten Winkeln in der Regel die besseren Approximationen und eigen sich gut dazu, Verdeckung und Inklusion näherungsweise zu berechnen. Obwohl diese Verfahren keine mathematisch völlig exakten Ergebnisse für die ursprünglichen Objekte liefern, reichen sie doch meistens

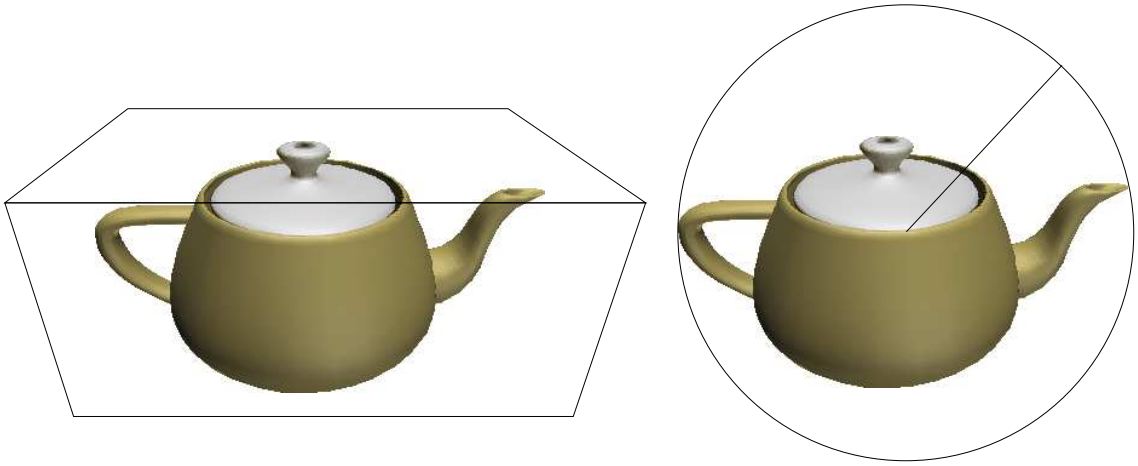


Abbildung 2.5: Umgebender Quader und umgebende Kugel

aus, um bestimmte Entscheidungen zu treffen und die Schnelligkeit ihrer Berechnung überwiegt ihre verbleibende Unsicherheit bei weitem.

2.2.7 Die Schritte der Bilderzeugung

Ein Beschreibungsmodell für den gesamten Bilderzeugungsprozeß in der Computergraphik ist die in [Watt & Watt, 92, Foley et al., 96] beschriebene *Rendering Pipeline* oder *Bilderzeugungskette*. Aus den Modelldaten wird in mehreren Schritten das Bild erzeugt und jedem dieser Schritte lassen sich bestimmte Daten und Prozesse zuordnen. Dieses Beschreibungsmodell möchte ich hier einführen, um die später beschriebenen Verfahren und Informationen selbst einzelnen Schritten der Rendering Pipeline zuordnen zu können. Am Anfang der in Abbildung 2.6 (nach [Foley et al., 96]) gezeigten Pipeline stehen die eingangs beschriebenen Modelldaten, ihre Gruppierungen zu Objektgruppen und die in der Szenerie vorhandenen beweglichen Lichtquellen. Diese Daten werden im ersten Schritt in eine Weltkoordinatendarstellung transformiert, in der die (möglicherweise bewegten) Objekte und Lichter sich an fest bestimmten Positionen im Raum befinden. Diese statische 3D-Welt wird nun unter Berücksichtigung der Kameraposition und -Einstellung in eine Bildebene projiziert. Bei der Projektion werden auch die sichtbaren Polygone ermittelt und damit festgelegt, welche Objekte im Bild sichtbar sind. Aus der zweidimensionalen Projektion wird schließlich das Bild berechnet, das nur noch aus verschieden gefärbten Pixeln besteht. Diese Bildberechnung bezieht möglicherweise Angaben über die Kamerablende und Verschlußzeit mit ein, um Tiefenschärfe und Bewegungsunschärfe korrekt darzustellen.

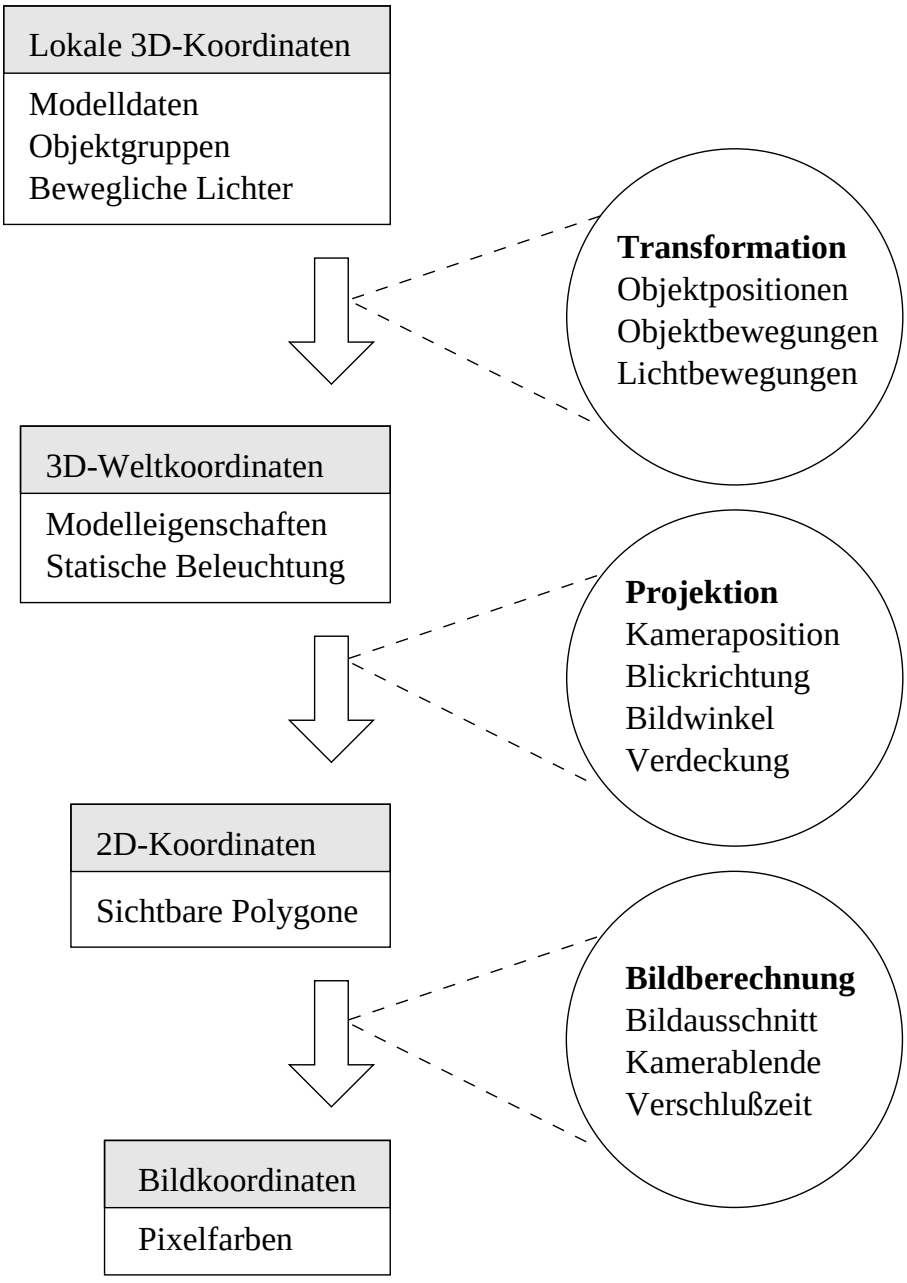


Abbildung 2.6: Die Bilderzeugungskette

2.3 Strukturelle Sicht

Nachdem sich die ersten beiden Abschnitte dieses Kapitels vor allem mit der technischen Erzeugung von Filmen beziehungsweise Animationen befaßten, soll nun die Struktur und der kommunikative Inhalt der Animationen beschrieben werden.

2.3.1 Syntax, Semantik und Pragmatik

In Anlehnung an die Analyse eines Textes oder einer Bildfolge (zu letzterem siehe [Bandyopadhyay, 90, Rist, 95, André, 95]) läßt sich eine Animation beziehungsweise das sie beschreibende Drehbuch bis zu einem gewissen Grad syntaktisch und semantisch beschreiben. Die Kohärenz einer Animation auf der syntaktischen Beschreibungsebene sichert dabei ihre Wohlgeformtheit. Die semantische Beschreibung stellt eine Verbindung zwischen den Oberflächenelementen und deren Bedeutung her und die Pragmatik beschreibt die Verwendungsweise bestimmter bildsprachlicher Elemente. Während sich die Syntax und Semantik relativ einfacher Bildsprachen formal vollständig fassen läßt (Eine grundlegende Arbeit hierzu ist [Mackinlay, 86]), ist dies für die komplexe Bildsprache eines Filmes oder einer 3D-Animation nur begrenzt möglich. Im folgenden möchte ich diese Beschreibungen daher eher informal oder durch Beispiele vornehmen.

2.3.1.1 Syntaktische Beschreibung

Die syntaktische Beschreibung einer Animation nennt – soweit möglich – ihre Oberflächenelemente und deren Relationen zueinander. Die Oberflächen- oder syntaktischen Elemente einer 3D-Animation sind zunächst die dargestellten Polygonmodelle sowie die zusätzlich eingeführten metagraphischen Objekte (Pfeile, Beschriftungen), aber auch die Beleuchtungsanordnung und die Kamerapositionen und -einstellungen. Zu den syntaktischen Elementen zählen im einzelnen:

- dargestellte Objekte der Modellwelt,
- dargestellte metagraphische Objekte,
- Materialeigenschaften der dargestellten Objekte,
- Detaillierungsgrade der dargestellten Objekte,
- verwendete Kamerapositionen,
- verwendete Brennweiten und Blenden,
- verwendete Beleuchtungsanordnungen.

Zwischen diesen Elementen gibt es bestimmte Möglichkeiten der Verknüpfung. Manche dieser Verknüpfungen beschränken sich auf syntaktische Elemente innerhalb eines einzelnen Bildes, andere umfassen den zeitlichen Ablauf einer Animation, also Beziehungen zwischen verschiedenen Bildern. Beispiele für solche Beziehungen sind:

- Kontinuität (zwischen verschiedenen Bildern),
- Konsistenz (innerhalb eines Bildes),
- Kontrast (innerhalb eines Bildes oder zwischen verschiedenen Bildern),
- Serialität (zwischen verschiedenen Bildern).

Zur *Kontinuität* gehört hierbei, daß innerhalb einer Einstellung oder Szene (im filmtechnischen Sinne, siehe. Abschnitt 2.1.7) nicht plötzlich Objekte verschwinden, das Licht ausfällt oder die Kamera unmotiviert zu einer anderen Position springt. *Konsistenz* bedeutet beispielsweise, gleichartige Objekte mit gleichem Detaillierungsgrad darzustellen, während *Kontrast* beschreibt, daß verschiedenartige Objekte verschieden dargestellt werden. *Serialität* bezieht sich schließlich auf den richtigen zeitlichen Ablauf von Bewegungsstadien oder Handlungsfolgen. Zur Verdeutlichung sollen im folgenden einige syntaktische Beziehungen einer einfachen wohlgeformten Bildfolge (Abbildung 2.7) aufgezählt werden. Von links oben nach rechts unten wird in dieser

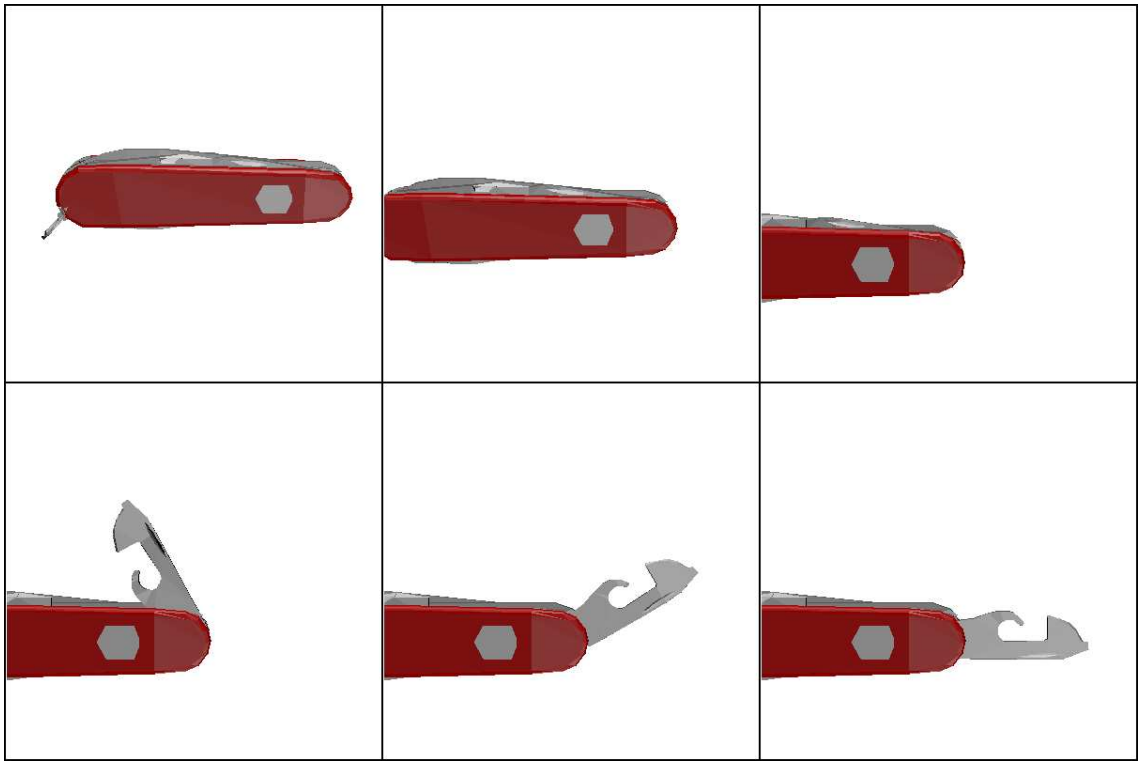


Abbildung 2.7: Bildfolge aus einer syntaktisch wohlgeformten Animation

Bildfolge dargestellt, wie der Dosenöffner eines Schweizer Taschenmessers geöffnet wird. Es existieren folgende syntaktische Beziehungen:

- Konsistenz der Detaillierungsgrade aller dargestellten Hintergrundobjekte (alle außer dem Dosenöffner) innerhalb jedes Bildes
- Kontrast der Detaillierungsgrade zwischen dem Vordergrundobjekt (Dosenöffner) und den Hintergrundobjekten (in Abbildung 2.7 nicht sehr auffallend)

- Kontinuität der Detaillierungsgrade und Materialeigenschaften aller dargestellten Objekte während der gesamten Bildfolge
- Kontinuität der Beleuchtung und Kameraführung während der gesamten Bildfolge
- Serialität der Bewegungsstadien des Dosenöffners

Unter Kontinuität der Kameraführung ist hierbei nicht zwangsläufig eine feste Kameraposition zu verstehen, sondern vielmehr eine Abfolge von Kamerafahrten und Schnitten, die sich harmonisch aneinanderreihen und keine unmotivierten großen Sprünge machen. Die Erfüllung solcher Beziehungen sichert insgesamt die syntaktische Wohlgeformtheit einer Animation bezüglich der verwendeten Bildsprache. Die

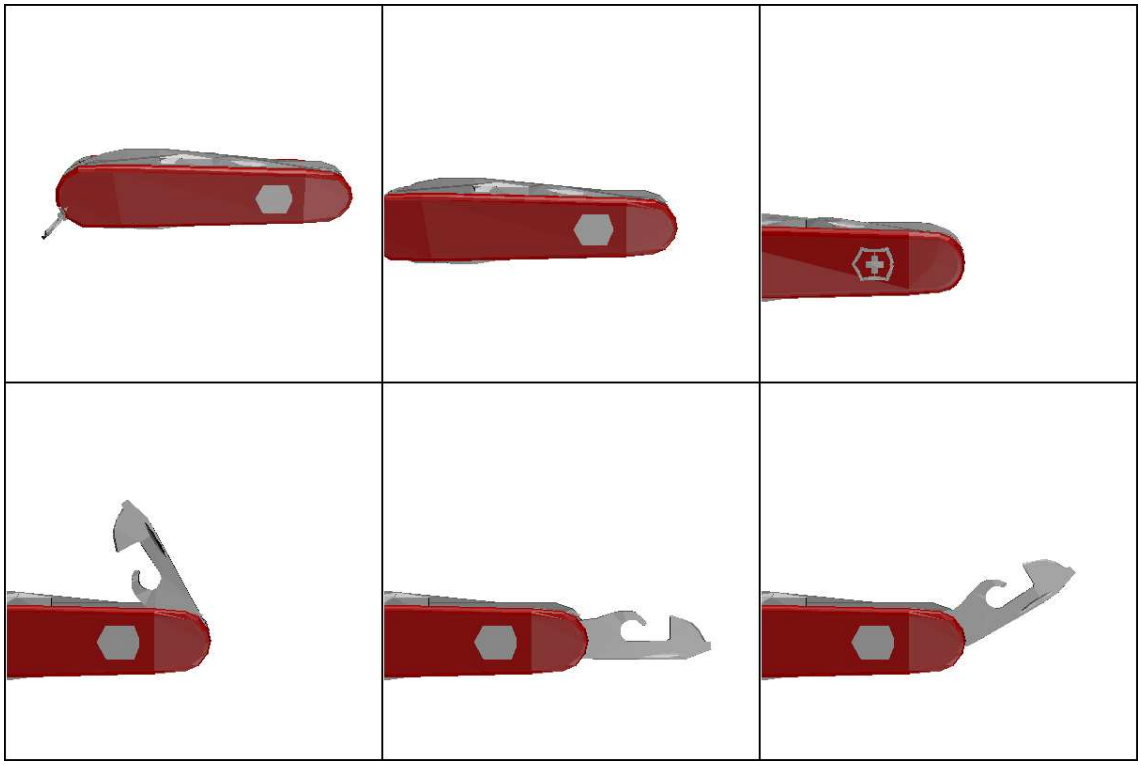


Abbildung 2.8: Bildfolge aus einer syntaktisch nicht wohlgeformten Animation

Bildfolge 2.8 zeigt als Gegenbeispiel eine Animation, in der zwei der oben genannten Relationen verletzt sind. Die Serialität der Bewegungsstadien ist in den letzten drei Bildern gestört (zumindest bezüglich der einfachen Bewegung *öffnen*) und das dritte Bild verletzt die Kontinuität der Detaillierungsgrade (sichtbar am Logo des Schweizer Messers). Eine solche Animation ist syntaktisch nicht wohlgeformt, ähnlich einem Satz, der eine falsche Wortreihenfolge und nicht zueinander passende Deklinationsformen verwendet.

2.3.1.2 Semantische Beschreibung

Im Gegensatz zur syntaktischen Beschreibungsebene beschäftigt sich die semantische Ebene mit der Bedeutung der dargestellten Oberflächenelemente und der

daraus syntaktisch gebildeten komplexeren Konstrukte. Den dargestellten Oberflächenelementen oder Konstrukten wird über eine sogenannte *Enkodierrelation* eine bestimmte Bedeutung zugeordnet, das heißt die Oberflächenelemente oder Konstrukte *enkodieren* bestimmte Informationen. Diese Notation wurde von [Mackinlay, 86] eingeführt und eine Semantik läßt sich für die bei ihm behandelten Liniendiagramme formal vollständig spezifizieren. Für die vorliegende Situation komplexer 3D-Animationen ist eine solche vollständige formale Beschreibung jedoch mit den derzeit zur Verfügung stehenden Beschreibungsmitteln zu umfangreich.

Immerhin läßt sich für bestimmte Arten syntaktischer Elemente der Animation eine Semantik angeben. So enkodieren die Polygonmodelle der dargestellten Objekte offensichtlich die entsprechenden Weltobjekte. Eine wesentliche Stärke des Mediums Animation (und anderer dynamischer Medien) ist die Fähigkeit, neben den drei Raumdimensionen auch die Dimension Zeit sehr direkt darstellen zu können. So enkodiert die zeitliche Anordnung (Reihenfolge, Dauer) von Aktionen in einer Animation deren Reihenfolge und Dauer in der Wirklichkeit (Siehe auch [Feiner et al., 93]). Im Gegensatz dazu ist es sehr schwierig, beispielsweise die *Semantik* einer Kamerafahrt oder eines Zooms anzugeben. Solche Techniken lassen sich eher unter dem Gesichtspunkt ihrer Verwendung (*Pragmatik*) fassen.

2.3.1.3 Pragmatische Beschreibung

Eine Kamerafahrt kann beispielsweise eingesetzt werden, um ein Objekt ins Bild zu bringen oder es bei einer Bewegung zu verfolgen. Ein Scheinwerfer hat den Zweck, das Objekt, auf das er gerichtet ist, optisch hervorzuheben. Die gleiche Wirkung wird erreicht durch eine passende Wahl der Abstraktionsgrade, einen metagraphischen Zeigepfeil oder eine rhythmische Farbveränderung (blinken) des Objektes. Auch Schnitte innerhalb einer Animation haben einen bestimmten Zweck, und zwar entweder die Überbrückung großer Distanzen, für die eine Kamerafahrt nicht mehr geeignet ist oder die Verdeutlichung des Umstandes, daß eine neue Handlung beginnt oder ein Zeitraum übersprungen wurde.

2.3.2 Kommunikationstheoretische Strukturierung

Während die Beschreibung einer Animation mit den klassischen Mitteln der Syntax, Semantik und Pragmatik eher schwierig ist, erscheint ihre Beschreibung auf der Ebene ihres kommunikativen Gehaltes und Aufbaus wesentlich angemessener. Eine Animation läßt sich nach ihren kommunikativen Zielen (*intentionale Struktur*), nach den rhetorischen Beziehungen ihrer einzelnen Teile (*rhetorische Struktur*) und nach der Aufmerksamkeit des Betrachters im Verlauf der Animation (*attentionale Struktur*) gliedern. Diese Strukturierungsprinzipien unterteilen eine Animation in Abschnitte, die größtenteils auch mit den filmtechnischen Einheiten *Sequenz*, *Szene* und *Einstellung* zusammenfallen.

2.3.2.1 Rhetorische Struktur

Texte, die ein kommunikatives Ziel verfolgen, tun dies meist mit Hilfe bestimmter rhetorischer oder argumentativer Techniken. Bei der Analyse solcher Texte läßt sich

eine rhetorische Struktur angeben, die deren logischen und argumentativen Aufbau beschreibt. In [Mann & Thompson, 87b, Mann & Thompson, 87a] wird die *Rhetorical structure theory* (RST) eingeführt, die eine solche Beschreibung auf einer formalen Ebene ermöglicht. Längere Texte werden hierzu in Untereinheiten zergliedert, die ihrerseits wiederum aus Untereinheiten bestehen können. Zwischen diesen Einheiten werden rhetorische Relationen wie Kontrast, Elaboration oder Hintergrundbildung definiert. Es ergibt sich eine hierarchische Gliederung des Textes in Form einer baumähnlichen Struktur mit Relationen zwischen den Knoten und Blättern. In Anlehnung an diese Formalisierung läßt sich nun eine rhetorische Struktur von Animationen beschreiben. Die Elemente dieser rhetorischen Struktur sind dabei Teile der Animation, die sich über einen bestimmten Zeitraum erstrecken und eine bestimmte rhetorische Funktion erfüllen. Beispiele hierfür sind

- das Zeigen eines Objektes im Bild,
- die Hervorhebung eines Objektes (durch Farbe, Detaillierungsgrad oder Metagraphik),
- das Zeigen einer Aktion mit einem Objekt,
- das Zeigen einer Objekteigenschaft.

Die zugehörigen Relationen lassen sich in Anlehnung an die in ([Mann & Thompson, 87b, Mann & Thompson, 87a]) beschriebenen Relationen festlegen. So bildet das Zeigen eines Objektes im Bild beispielsweise den notwendigen Hintergrund zum Zeigen einer Objektbewegung. Es liegt nahe, diese Art der Beschreibung auf die strukturell vorhandenen Untereinheiten hierarchischer Animationsskripte anzuwenden. Relationen, die sich aus [Mann & Thompson, 87b] direkt auf die Beschreibung von Animationen übertragen lassen, sind beispielsweise

- Hintergrund (background)
- Ausarbeitung (elaboration)
- Wiederholung (restatement)
- Sequenz (sequence)
- Kontrast (contrast)
- Ergebnis (volitional/non-volitional result)
- Grund (volitional/non-volitional cause)

Es lassen sich jedoch nicht alle RST-Relationen auf die Animation übertragen. Gegenbeispiele sind das Zugeständnis (concession) und die Interpretation, die bildlich nicht darstellbar sind. Mit den oben genannten Relationen läßt sich die in Abbildung 2.7 gezeigte Animation nun wie folgt beschreiben (Abbildung 2.9 links): Vom ersten bis zum dritten Bild findet eine *Elaboration* statt, die den Ort, an dem sich der Dosenöffner befindet, zunächst in seinem gesamten Umfeld (*Position 1*) und dann in Nahaufnahme (*Position 2*) zeigt. Diese ersten drei Bilder ergeben die *Einstellung 1*

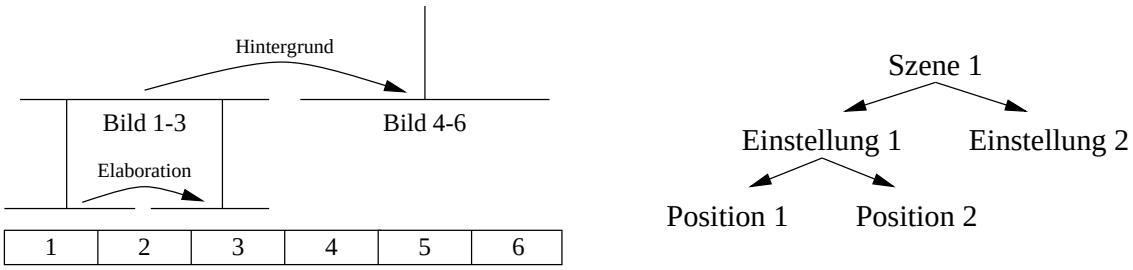


Abbildung 2.9: Rhetorische und filmtechnische Struktur der Animation 2.7

(In Wirklichkeit handelt es sich bei einer Animation dabei um einige hundert Bilder, während Abbildung 2.7 nur einzelne davon zeigt). Die restlichen drei Bilder zeigen die Aktion *Öffnen* des Objekts *Dosenöffner* und bilden damit die *Einstellung 2*. Hierbei schafft Einstellung 1 den *Hintergrund*, der zum Verständnis der Einstellung 2 notwendig ist, nämlich die Information, an welcher Stelle sich der Dosenöffner überhaupt befindet. Außerdem führt Einstellung 1 zu einer passenden Kameraposition, die das Zeigen der gesamten Bewegung in Einstellung 2 ermöglicht. Auf diese Art lassen sich auch umfangreichere Animationen in rhetorische Einheiten unterschiedlicher Größe zerlegen, die ihrerseits wieder aus Untereinheiten bestehen können. Diese Einheiten sind untereinander durch rhetorische Relationen verbunden und ergeben zusammen eine rhetorische Struktur der Animation, die in den allermeisten Fällen vom Aufbau her mit ihrer filmtechnischen Struktur übereinstimmt.

2.3.2.2 Intentionale Struktur

Eine Grundidee bei der automatischen Generierung von Präsentationen ist, daß diese Präsentationen bestimmte *kommunikative Ziele* verfolgen, und zwar die Ziele, bestimmte Informationseinheiten zu präsentieren. Diese *Präsentationsziele* sind wie die präsentierte Information hierarchisch strukturiert und zerfallen jeweils entweder in einfachere Unterziele oder können direkt durch bestimmte Präsentationsteile umgesetzt werden (Siehe auch [Rist, 95]).

Die intentionale Struktur einer Animation ist gegeben durch die Präsentationsziele und Unterziele, die mit der Animation und ihren Untereinheiten wie Sequenzen, Szenen und Einstellungen verfolgt werden (Siehe auch [André, 95]). Ziel der Animation in Abbildung 2.7 ist es beispielsweise, dem Zuschauer zu zeigen, *wie* der Dosenöffner des gezeigten Schweizer Taschenmessers geöffnet wird. Hierzu muß zunächst klar sein, *wo* sich der Dosenöffner überhaupt am Taschenmesser befindet. Das Ziel zerfällt also in die zwei Unterziele *"Zeige den Ort, an dem sich der Dosenöffner befindet"* und *"Zeige den Vorgang des Öffnens"*.

Die Elemente der intentionalen Struktur lassen sich den Elementen der filmtechnischen Struktur direkt zuordnen. Diese wiederum können direkt in Form rhetorischer Einheiten beschrieben werden (Abbildung 2.9), so daß sich unter den drei Gesichtspunkten eine übereinstimmende Strukturierung ergibt. Abbildung 2.10 zeigt die intentionale und die filmtechnische Struktur der Animation 2.7.

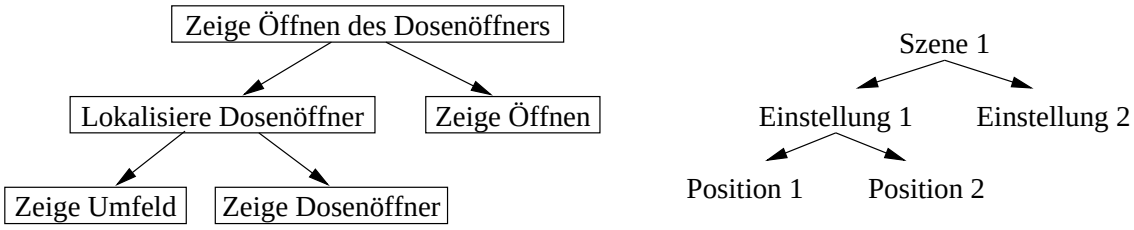


Abbildung 2.10: Intentionale und filmtechnische Struktur der Animation 2.7

2.3.2.3 Attentionale Struktur

Eine sehr nützliche Betrachtungsweise einer animierten Präsentation ist außerdem ihre *attentionale* oder *Fokusstruktur*. Die attentionale Struktur einer Animation beschreibt den Verlauf des *visuellen Fokus*, also den Wechsel der Objekte oder Bildteile, auf die sich die Aufmerksamkeit eines Betrachters richtet. Innerhalb eines einzelnen Bildes oder einer Bildfolge lassen sich beispielsweise *Fokusobjekte* und *Hintergrundobjekte* nach ihrer Rolle für die Bildaussage unterscheiden. In [Rist, 95] wird der Versuch einer solchen Strukturierung für statische Graphiken beschrieben und erweitert man das Verständnis einer Graphik um die Dimension Zeit, so läßt sich auch eine attentionale Struktur animierter Präsentationen angeben (Abbildung 2.11).

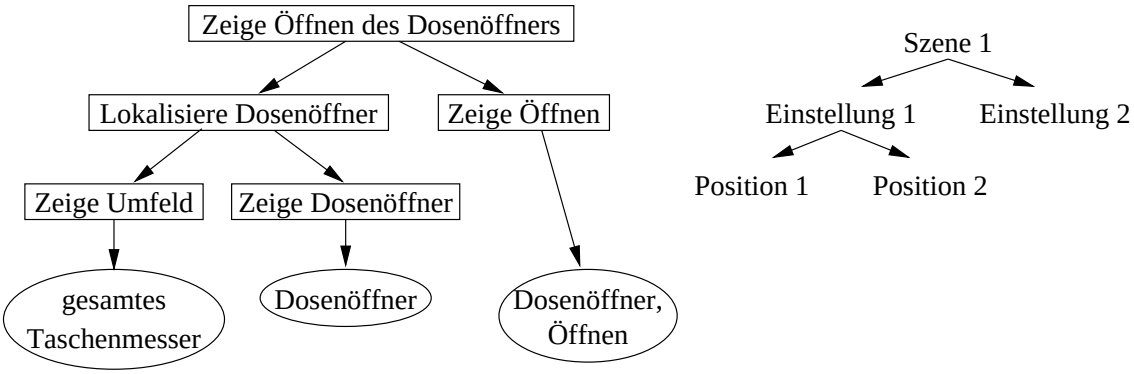


Abbildung 2.11: Attentionale und filmtechnische Struktur der Animation 2.7

Die attentionale Struktur einer Animation läßt sich aus ihrer intentionalen (und damit auch aus ihrer filmtechnischen) Struktur ableiten. Man ordnet hierzu jedem elementaren Abschnitt der intentionalen Struktur, also jedem nicht weiter untergliederten Präsentationsziel eine Beschreibung des visuellen Fokus zu, die für den zugehörigen Animationsteil die bildwichtigen Objekte oder Aktionen nennt. Am Beispiel der Animation aus Bild 2.7 sind dies die folgenden Beschreibungen: Bild 1 fokussiert das gesamte Taschenmesser, während im Verlauf zu Bild 3 die Aufmerksamkeit des Betrachters auf den Dosenöffner gelenkt wird. Während des Zeigens der eigentlichen Aktion in den Bildern 4-6 (Einstellung 2) verweilt der Fokus dann auf dem Objekt *Dosenöffner* sowie der Aktion *Öffnen*.

Der Fokusverlauf ergibt sich somit aus der intentionalen Struktur der Animation und bei einer automatischen Generierung des Drehbuches kann diese Information berücksichtigt werden. Die Steuerung des visuellen Fokus ist eine wichtige Aufgabe bei der Gestaltung des Drehbuches und man kann sie durch verschiedene film- oder animationstechnische Mittel unterstützen. Das Blinken eines Objektes, seine

Annotation durch metagraphische Elemente oder eine passende Wahl der Abstraktionsgrade (Siehe Abschnitt 4.2) sind effektive Mittel, die Wirkung einer Animation zu unterstützen und ihre Wahrnehmung durch den Betrachter im Sinne der Präsentationsziele zu lenken.

Zweite Szene

C: Ok, jetzt haben wir alle unseren Teil erklärt, aber wie geht's weiter?

B: Na, es haben sich ja schon andere Leute mit dem Problem befaßt. Schauen wir doch mal, was die herausgefunden haben. (geht zum Regal und nimmt einen Ordner heraus)

C: Gute Idee, aber sag mal, wenn das schon andere gemacht haben, wieso willst du das denn dann nochmal aufrollen?

B: (hört garnicht zu, blättert in seinem Ordner) ...Da! Mindestens Neun Leute, die es schonmal probiert haben!

A: (grinst) Aha, und du bist der zehnte, bei dem es dann endlich klappen soll...

B: Nein, das hat bei denen schon geklappt. Die haben halt alle so ihre Schwachpunkte, aber prinzipiell haben sie gezeigt, daß man es tun kann.

C: Laß mal sehen! (blättert durch den Ordner)

Die drei vertiefen sich in den Ordner und verbringen die nächste Zeit mit dem Durchblättern der darin abgehefteten Artikel.

Kapitel 3

Bisherige Arbeiten zur automatischen Animationsgenerierung

Nachdem die Gestaltung einer 3D-Animation nun unter verschiedenen Aspekten betrachtet wurde, möchte ich zunächst einige Arbeiten und Systeme beschreiben, die sich mit diesem Problem oder mit ähnlichen Aufgabenstellungen beschäftigen. Zunächst soll das System BETTY kurz vorgestellt werden, dessen Generierungsverfahren ein direkter Vorläufer des in dieser Arbeit vorgestellten Ansatzes ist.

3.1 BETTY

In dem in [Butz, 94a, Butz, 94b] vorgestellten System BETTY sind bereits Grundideen der vorliegenden Arbeit verwirklicht. Die Struktur der Animationsskripte entspricht der in Abschnitt 6.3 beschriebenen und das Generierungsverfahren arbeitet ähnlich dem des Systems CATHI, jedoch nicht inkrementell.

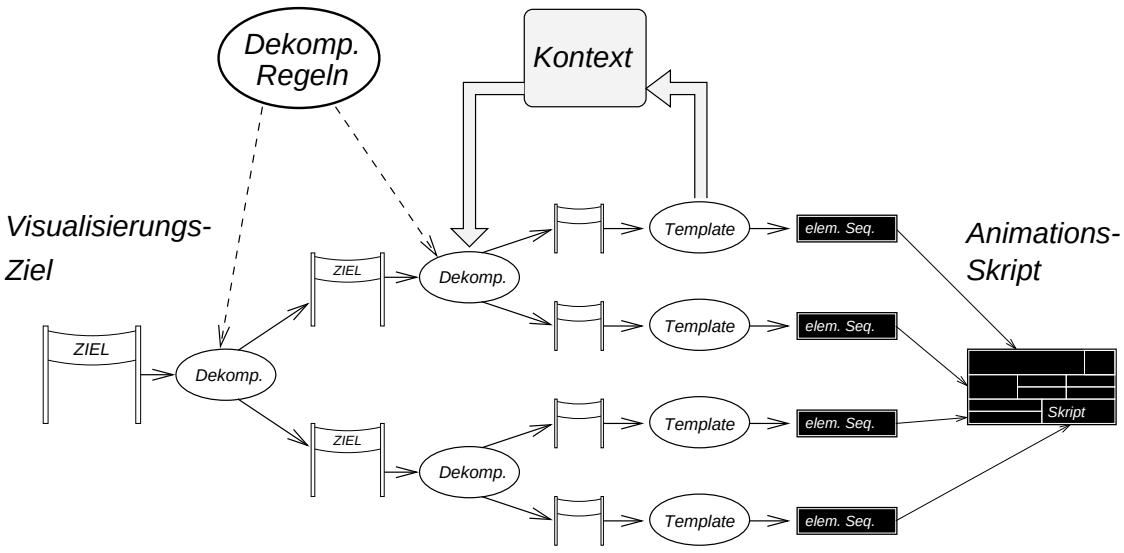


Abbildung 3.1: Dekomposition der Präsentationsziele in BETTY

Wie in Abbildung 3.1 gezeigt, werden auch in BETTY die vorgegebenen Präsentationsziele Schritt für Schritt in einfachere Unterziele dekomponiert und anschließend in elementare Skriptsequenzen übersetzt. Die Generierung des Skripts muß jedoch vollständig abgeschlossen sein, bevor die Bildberechnung beginnt.

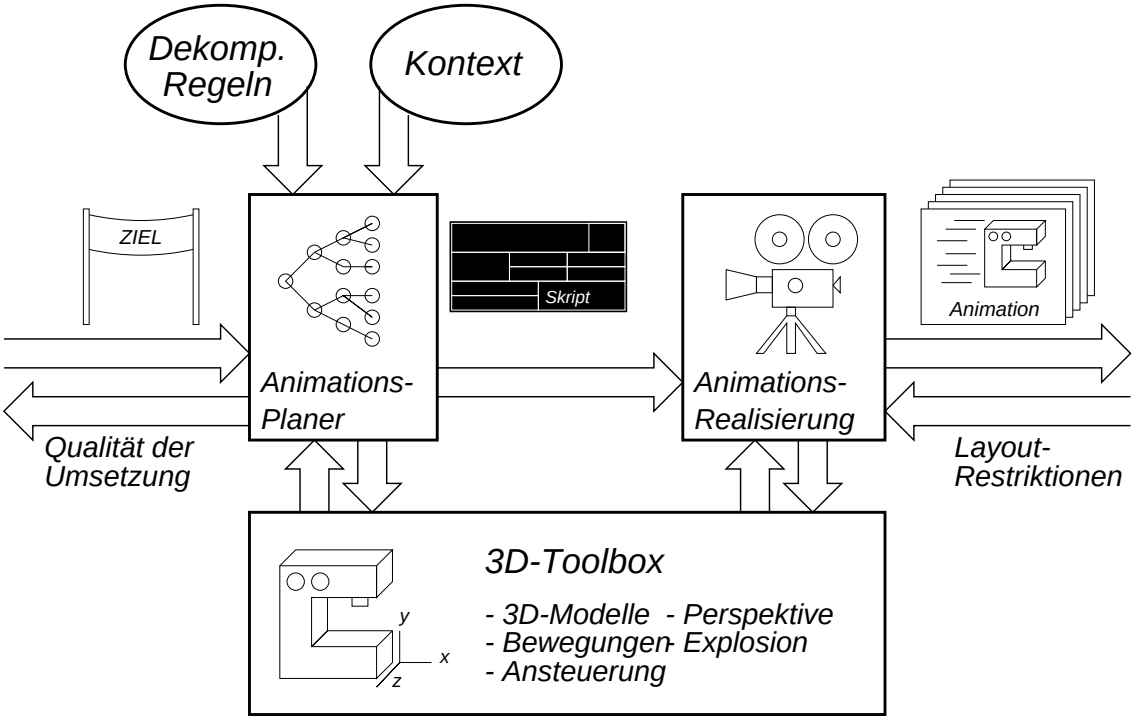


Abbildung 3.2: Aufbau des Systems BETTY

Bedingt durch das damals verwendete Ausgabesystem S-Dynamics befaßte sich die Arbeit lediglich mit Drahtgitterdarstellungen, die keiner Beleuchtung bedurften. In Kapitel 8 der Arbeit [Butz, 94b] werden als denkbare und sinnvolle Erweiterungen die Einbeziehung der Lichtführung in den Planungsprozeß sowie der gezielte Einsatz von Farbeffekten, Metagraphik und Abstraktionen genannt. Die damals angestellten Überlegungen sind in mehr oder weniger großem Umfang in die vorliegende Arbeit eingeflossen. Insbesondere der Einsatz abstrahierter Modelle erwies sich dabei als großer Gewinn für die Systemperformanz und die Qualität der erzeugten Animationen. Letztendlich wurde das System BETTY jedoch nie vollständig softwaretechnisch in das übergeordnete Projekt WIP integriert, da die benötigten Rechenzeiten den Rahmen eines interaktiven Systems bei weitem überschritten. Im Grunde genommen sind die in BETTY verwendeten Verfahren jedoch Vorläufer der in dieser Arbeit geschilderten und neben einer Verfeinerung der Methoden (Inkrementalität, verbesserte geometrische Berechnungen, wesentlich effizientere Umsetzung des Generierungsverfahrens) hat auch der Übergang zur nächsten Rechnergeneration ihre technische Umsetzung begünstigt. Abbildung 3.2 zeigt die Architektur des Systems BETTY. Weitere Details des Systems möchte ich hier nicht ausführen, da diejenigen Ideen, die sich bewährt haben, in der vorliegenden Arbeit bereits an anderer Stelle beschrieben sind.

3.2 Zoom Illustrator

Das an der Universität Magdeburg entwickelte System *Zoom Illustrator* [Preim et al., 96] hat einen inhaltlich sehr engen Bezug zu den Arbeiten an CATHI beziehungsweise BETTY. Zielsetzung des Systems ist es, komplexe medizinische Modelle zu visualisieren. Medizinstudenten soll dabei am Bildschirm die Möglichkeit gegeben werden, beispielsweise Teile des menschlichen Körpers zu erforschen. Um dies zu erreichen, werden die Körperteile aus verschiedenen Richtungen dargestellt, gedreht, zerlegt, teilweise durchsichtig gemacht und mit erklärenden Texten annotiert. Das System plant also außer einfachen Kamerafahrten (rund um das Modell, näher heran bzw. weiter weg) auch die Präsentation am Bildschirm mit dem zugehörigen Text.

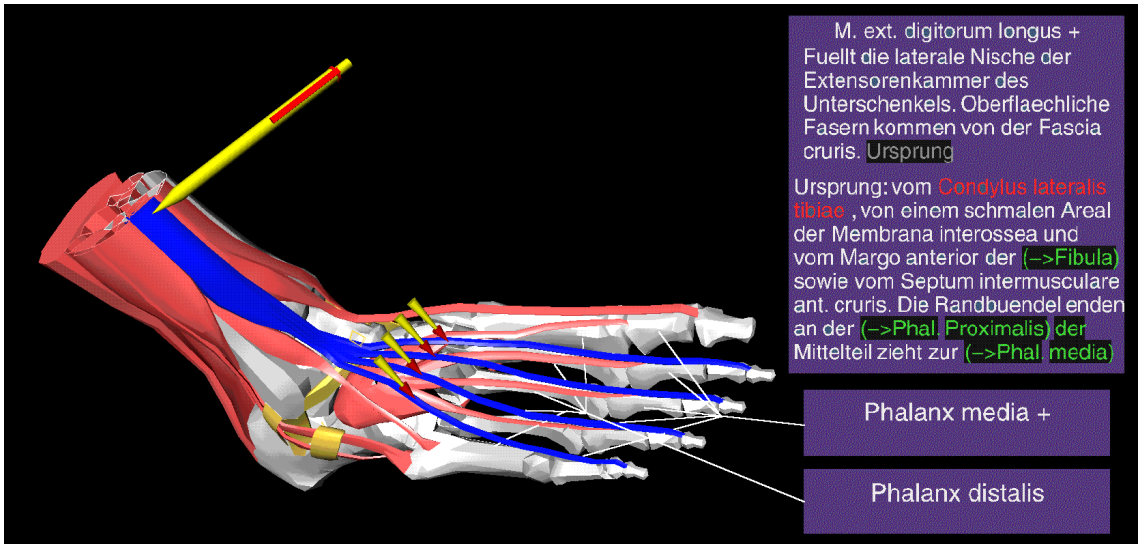


Abbildung 3.3: Beispiel einer Präsentation des Zoom Illustrators

Außer Text-Annotationen ist das System aber auch in der Lage, metagraphische Zeigeobjekte einzusetzen (Abbildung 3.3, mit freundlicher Genehmigung des Autors Bernhard Preim). So kann der Verlauf von Muskelsträngen oder Sehnen beispielsweise mit einem Pfeil oder dem Modell eines Kugelschreibers nachgezeichnet werden. An Verzweigungsstellen des Muskels teilt sich hierbei auch das Zeigegerät in mehrere kleinere Geräte auf. Diese Techniken beruhen auf umfangreichen Berechnungen am Polygonmodell, da eine Näherungslösung bei derart komplexen und sich durchdringenden Objekten mit zu vielen Fehlern behaftet wäre.

Die Autoren benutzen außerdem den Begriff *Zoom* in einem anderen als dem klassischen filmtechnischen Sinn (Siehe Abschnitt 2.1.4). Ein Zoom auf ein Objekt ist in der hier angewendeten Terminologie eine Darstellung, die dieses Objekt vergrößert und alle umgebenden Objekte so verformt und verkleinert, daß die Zusammensetzung der Modellwelt und die räumlichen Verhältnisse außer dem Größenverhältnis insgesamt erhalten bleiben. Dieses Verfahren ist jedoch eigentlich eher den metagraphischen Techniken zuzuordnen, da es ähnlich einer Explosionszeichnung die Modellwelt auf eine nicht natürliche und nur durch die Präsentation motivierte Weise verändert. Die Beleuchtung der Modelle des Zoom-Illustrators ist bei der Modellierung vorgegeben und wird nicht zu Visualisierungszwecken eingesetzt. Die auftretenden Kamerafahrten beschränken sich auf Rundflüge um das jeweilige Hauptobjekt

sowie Zu- und Rückfahrten. Diese werden jedoch automatisch generiert, je nach dem, welches Objekt der Betrachter gerade sehen möchte. Insgesamt ist somit das System Zoom Illustrator eine interessante Umsetzung einer ähnlichen Aufgabe unter einer anderen Sichtweise und mit unterschiedlichen Schwerpunkten. Einige der in [Butz, 94a, Butz, 94b] sowie in früheren Arbeiten am DFKI veröffentlichten Ideen sind direkt oder indirekt in den Zoom Illustrator eingeflossen, jedoch sind viele Details unterschiedlich gelöst und genauer auf den jeweiligen Verwendungszweck abgestimmt.

3.3 ESPLANADE

Ein System, das sich mit fast der gleichen Aufgabenstellung befaßt, wie die vorliegende Arbeit, ist das in [Karp & Feiner, 90, Karp & Feiner, 93] vorgestellte an der Columbia University New York entwickelte System ESPLANADE (Expert System for PLANning Animation, Design, and Editing). Es handelt sich dabei um einen Skriptgenerator für 3D-Animationen, der auf einem hierarchischen Planungsverfahren basiert (Abbildung 3.4, aus [Karp & Feiner, 93]).

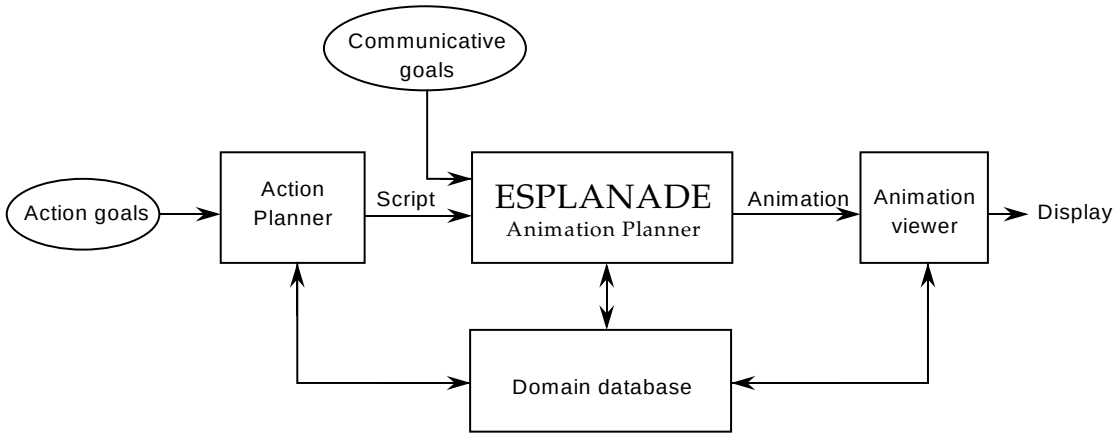


Abbildung 3.4: Funktionsweise des Systems ESPLANADE

3.3.1 Eingaben des Systems

Eingabe des Systems ist eine Reihe von Aktionen, die visualisiert werden sollen, und die von einem *action planner* schon vorher zu einer Art Skript zusammengefaßt wurden. Außerdem wird eine Reihe *kommunikativer Ziele* spezifiziert, die die Animation erfüllen soll. Solche Ziele sind (laut [Karp & Feiner, 93]): 'Zeige die Aktion *a*' oder 'Zeige, daß Aktion *b* durch Aktion *a* bewirkt wurde'. Das System ist also dazu ausgelegt, Aktionen und deren kausale Zusammenhänge in der Modellwelt zu visualisieren.

3.3.2 Planungsverfahren

Der Autor Peter Karp lehnt sich an die Grundidee der Filmstruktur, die in Abschnitt 2.1.7 eingeführt wurde, an und unterteilt die Skriptplanung in die Planung auf *Film-*

ebene, *Sequenzebene*, *Szenenebene* und *Einstellungsebene*. Auf diesen Ebenen generiert ein operatorbasiertes Planungsverfahren ähnlich dem ABSTRIPS System eine immer genauer werdende Spezifikation des Animationsskriptes. Die Dekomposition des Skripts in kleinere Einheiten erfolgt nach filmtechnisch motivierten Heuristiken. Diese filmtechnische Motivation ist in [Karp & Feiner, 90] weiter ausgeführt und geht auf Grundregeln der Filmgestaltung und Kameraführung ein. In einer anschließenden *Sequenziellen Phase* wird aus der filmtechnischen Beschreibung der einzelnen Einstellungen eine computergraphische Beschreibung der Animationsframes erzeugt.

3.3.3 Eigenschaften des Ansatzes

Das System berücksichtigt außer der Kameraführung selbst auch die Präsentation der Animation am Bildschirm, indem die Animationsskripte das sogenannte *viewport layout* angeben. So ist es möglich, während der Animation das Bild einer zweiten Kamera in einem zweiten Fenster einzublenden und darin beispielsweise Details einer Aktion zu zeigen. Mit diesem Verfahren wird auch die Blende beim Schnitt simuliert, und zwar indem das Bild zweier Kameras ineinander überführt wird. Die Wahl der Kamerapositionen selbst wird jedoch auf Defaultpositionen zurückgeführt, die in der Modellwelt vorgegeben sind. Auch die Lichtführung wird offensichtlich fest vorgegeben, da das Problem in den zu ESPLANADE veröffentlichten Arbeiten nicht angesprochen wird.

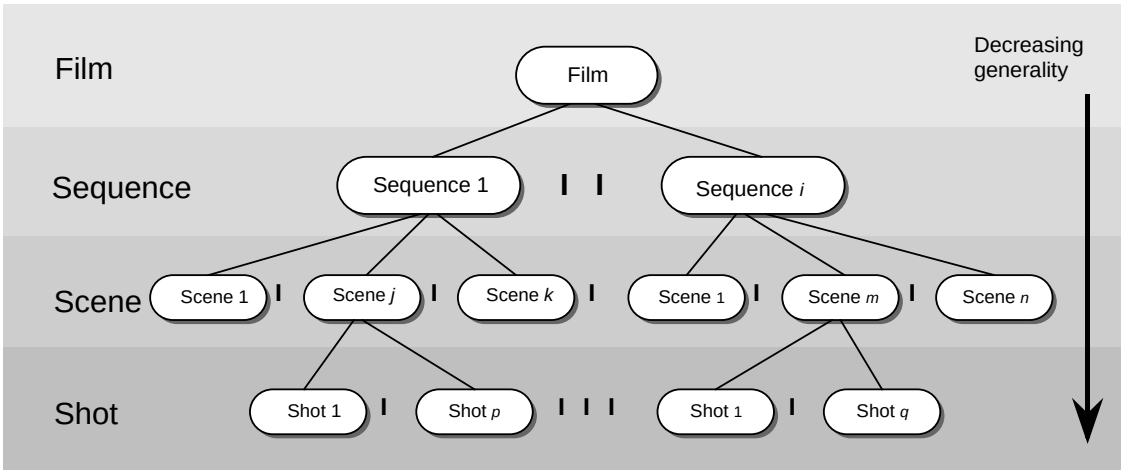


Abbildung 3.5: Skriptplanung in ESPLANADE

Eine auffallende Eigenschaft des Ansatzes ist die strikte Trennung der filmtechnisch motivierten Abstraktionsebenen. Der Planungsprozeß legt zunächst alle Sequenzen des Films, danach alle Szenen jeder Sequenz und erst zum Schluß alle Einstellungen jeder Szene fest (Siehe Abbildung 3.5, aus [Karp & Feiner, 93]). Dieses Verfahren ermöglicht einen kompletten *look-ahead* und *look-back* innerhalb jeder Planungsebene, jedoch nicht darüber hinaus. Dies ermöglicht einerseits Planungsentscheidungen, die durch später zu planende Filmteile motiviert sind, andererseits aber nicht solche, die durch andere (insbesondere tiefere) Planungsebenen zu begründen wären. Es erscheint fraglich, ob eine so enge Anlehnung an die Filmstruktur zur Generierung eines Animationsskriptes sinnvoll ist, da die Struktur des Films ja durch seinen Herstellungsprozeß bestimmt ist, der bei der Animation völlig anders abläuft.

Nachdem in der beschriebenen Art und Weise eine komplette Spezifikation des Animationsskriptes geplant ist, kann die zugehörige Animation in Echtzeit durch ein vorhandenes Graphiksystem abgespielt werden. Eine vorgeplante Animation läßt sich also kurzfristig abspielen, die Planung selbst jedoch ist nicht in Echtzeit möglich. Dabei benötigt die Planung der in [Karp & Feiner, 93] gezeigten Beispielanimation laut Angaben des Autors nicht wesentlich länger als ihre Präsentation, durch die Struktur des Planungsprozesses muß sie jedoch ähnlich wie in BETTY vollständig abgeschlossen sein bevor die Ausgabe beginnt.

3.4 Jack und seine Anwendungen

Während ESPLANADE sich mit der filmtechnischen und dramaturgischen Gestaltung einer 3D-Animation beschäftigt, liegt der Schwerpunkt des in [Badler et al., 93] vorgestellten Systems Jack gewissermaßen auf der Modellierung virtueller Schauspieler (Siehe Abbildung 3.6). Durch die aufwendige Modellierung der Bewegungsabläufe beim Gehen, Greifen und anderen Handlungen ist es möglich, den synthetischen Agenten Jack, dargestellt durch das dreidimensionale Modell eines Menschen, auf sehr hoher Ebene anzusteuern. In [Zeltzer, 90] werden verschiedene Abstraktionsebenen einer solchen Ansteuerung formalisiert. Auf diesem System aufbauend gibt es mehrere Projekte, die sich mit der gezielten Ansteuerung der Bewegungen befassen. Ich möchte kurz diejenigen nennen, deren Thema eine automatische Generierung von Bewegungsabläufen ist.

3.4.1 Bewegungsplanung und Interpolation von Positionen

In [Badler et al., 94] stellen Bindiganavale und Wei eine Methode vor, um den virtuellen Agenten aus einer Körperposition in eine andere zu bewegen. Hierzu werden in einem endlichen Automaten alle anatomisch sinnvollen Übergänge von Körperpositionen codiert und dann daraus der kürzeste Übergangsweg berechnet. Treten Kollisionen auf, so werden die Körperhaltungen durch gleichwertige ersetzt, die kollisionsfrei sind. Der Ansatz wurde erfolgreich eingesetzt, um den Agenten zwischen den Körperhaltungen *stehend* und *kniend*, jeweils mit oder ohne einen Gegenstand in der Hand, wechseln zu lassen. Außerdem wurde untersucht, welchen Einfluß die Faktoren Kraft, Geschwindigkeit und Ermüdung auf die Bewegungen einer virtuellen menschlichen Figur haben. Diese Faktoren beeinflussen die Bewegungsabläufe genauso wie Kinematik und Schwerkraft, lassen sie dabei aber noch natürlicher erscheinen.

3.4.2 Animation aus Instruktionen

Das System AnimNL ([Badler et al., 91, Webber et al., 93]) extrahiert aus natürlichsprachlichen Eingaben in der Form von Folgen kurzer Anweisungen deren Bedeutung und leitet daraus Ziele für den Agenten ab. Aufgrund dieser Ziele führt der Agent in der virtuellen Welt Handlungen aus, um sie zu erfüllen. Der Schwerpunkt der Arbeit liegt dabei auf der Konstruktion einer sinnvollen Folge von Handlungen um die gegebenen Anweisungen zu erfüllen. In der genannten Veröffentlichung wird auf

Abbildung 3.6: Der virtuelle Schauspieler JACK

eine Gestaltung der Kamera- oder Lichtführung nicht näher eingegangen. Man muß daher annehmen, daß sie im System festgelegt ist oder von Hand nachgeführt wird.

Die genannten Projekte sind hauptsächlich mit der intelligenten Steuerung des Agenten befaßt. Vorgänge in der virtuellen Welt werden simuliert und Kollisionen und Konflikte erkannt. Dies sind für die Animation in virtuellen Welten wichtige Gesichtspunkte, die jedoch nur in Bezug auf den Agenten selbst beleuchtet werden. Daß eine Kollisionsdetektion für die virtuelle Kamera, bzw. eine Detektion *optischer Kollisionen*, also der Verdeckung wichtiger Objekte, für eine gelungene Animation ebenso wichtig sind, wird in keinem der Systeme behandelt.

Zur Beleuchtung der Animationen wird in der angeführten Literatur ebenfalls nichts gesagt, jedoch gibt es neuere Arbeiten ([Nimeroff et al., 95]) am gleichen Institut, die sich mit der schnellen Berechnung einer globalen Beleuchtung der Szenerie befassen. Diese Ansätze fallen jedoch eher unter die Entwicklung neuer Renderverfahren, da sie sich mit anderen Formen der Lichtverteilung befassen statt mit der Position der Lichtquellen, und sie sollen deshalb hier nicht näher besprochen werden. Die Animation eines virtuellen Agenten stellt ein interessantes Anwendungsgebiet für eine automatische Kameraführung oder filmtechnische Gestaltung dar, da der Benutzer eines solchen Systems in der Regel vollständig mit der Steuerung des

Agenten beschäftigt ist und nicht mit der Kameraführung belastet werden möchte. Dieses Problem wird in den nächsten beiden besprochenen Arbeiten sowie in der in 3.8 beschriebenen angegangen.

3.5 Camdroid

Während die in Abschnitt 3.4 beschriebenen Systeme die Steuerung virtueller Schauspieler zum Ziel hatten, wird in [Drucker & Zeltzer, 95] eine Methode zur Formalisierung und Steuerung der Kameraführung in virtuellen Welten vorgestellt. Wie in [Zeltzer, 90] ist es das Ziel der Arbeit, Aktionen auf einer sehr hohen Abstraktionsebene, dem *task level* zu spezifizieren, die dann vom System eigenständig umgesetzt werden. Ein solcher Kamera-*task* ist beispielsweise das Filmen einer Unterhaltung zweier virtueller Akteure (Siehe Abbildung 3.8, aus [Drucker & Zeltzer, 95]). Die Autoren implementieren filmtechnisches Wissen aus einem Lehrbuch für Filmemacher [Arijon, 76] in Form von untereinander verbundenen *Kameramodulen* (Abbildung 3.7, aus [Drucker & Zeltzer, 95]). Diese Module sind in sich abgeschlossene Ein-

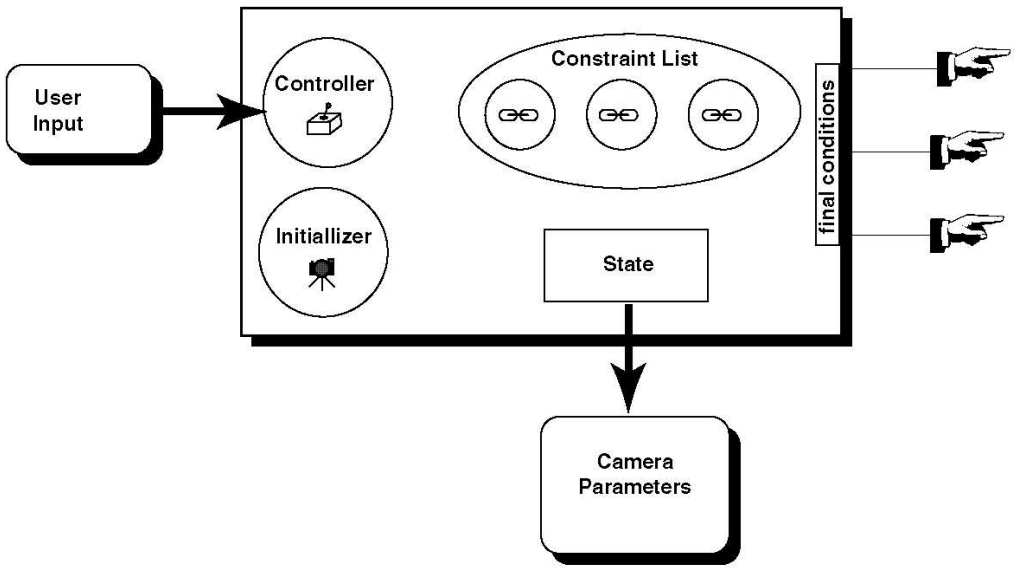


Abbildung 3.7: Kameramodul im System Camdroid

heiten, die eine spezialisierte Aufgabe erfüllen wie zum Beispiel das Filmen eines Sprechers aus einer bestimmten Perspektive. Um diese Aufgabe zu lösen, enthalten die Module eine Constraintmenge, aus der alle Kameraeinstellungen für eine gegebene Situation abgeleitet werden. Mehrere solcher Kameramodule sind durch einen Graphen mit entsprechenden Übergangsbedingungen zu einer Art endlichem Automaten verknüpft und übernehmen abwechselnd die Kontrolle der virtuellen Kamera. Im genannten Beispiel einer Konversation zwischen zwei Akteuren besteht der Graph aus zwei Kameramodulen zum Filmen beider Darsteller, die jeweils dann aktiviert werden, wenn eine Seite das Wort ergreift.

Nimmt man schließlich mehrere solcher Zustandsgraphen als verschiedene *Verhaltensweisen* und gibt dem System die Möglichkeit, je nach dem Geschehen in seiner Umgebung zwischen verschiedenen Verhaltensweisen auszuwählen, so erhält

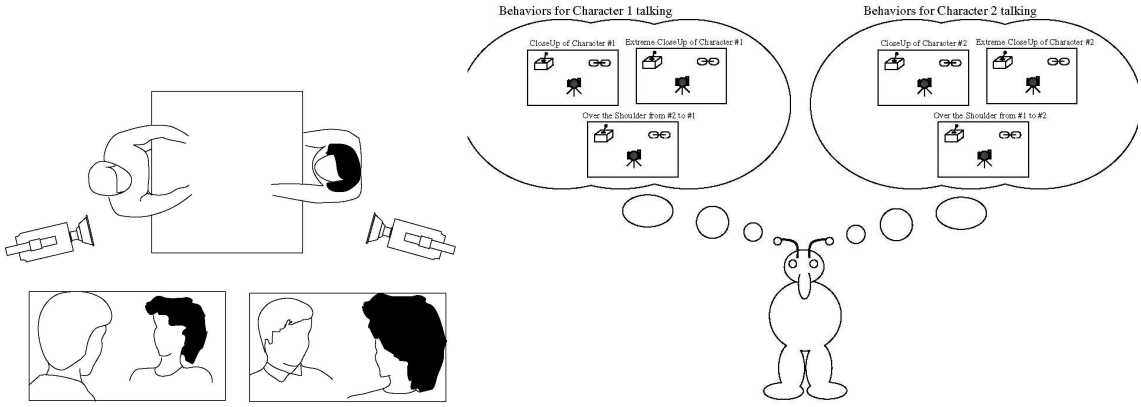


Abbildung 3.8: Filmen einer Konversation durch Camdroid

man den in [Drucker & Zeltzer, 95] beschriebenen virtuellen Kameramann *CamDroid*. Dieser Kameramann ist neben der Darstellung einer einfachen Unterhaltung auch in der Lage, ein so komplexes Geschehen wie ein Football-Spiel zu filmen. Die entstehenden Animationen entsprechen den enkodierten filmtechnischen Regeln und stellen das zu zeigende Geschehen ansprechend dar, sie beschränken sich dabei jedoch ausschließlich auf die klassischen filmtechnischen Mittel der Kameraführung und des Schnitts. Die Beleuchtung der Szenerien wird im genannten Artikel nicht angesprochen und Techniken, die für das Medium Computergraphik spezifisch sind, wie beispielsweise Abstraktion, Opazitätseffekte und Metagraphik, werden völlig außer acht gelassen. Außerdem beschränkt sich CamDroid auf eine passive Rolle und greift in die dargestellte Handlung selbst nicht ein, ähnlich dem Drehen eines Dokumentarfilmes. Diese Eigenschaft ist zwar zwingend notwendig für den Einsatz in virtuellen Welten, in denen das Geschehen durch Benutzer oder Simulationen bestimmt wird, man verschenkt jedoch gewissermaßen die Möglichkeit, wie ein Spielfilmregisseur in die Handlung einzugreifen und bestimmte Aktionen in der Modellwelt auf die Kameraführung abzustimmen.

3.6 DCCL und "The Virtual Cinematographer"

Eine weitere Formalisierung des gleichen Ausgangswissens wird in [Christianson et al., 96, He et al., 96] vorgeschlagen. Die Autoren formalisieren die Regeln des gleichen Lehrbuches [Arijon, 76] und benutzen sogar eine ähnliche Terminologie wie [Drucker & Zeltzer, 95]. Elementare Kamerapositionen werden von *Kameramodulen* ausgewählt und diese Module sind in Zustandsgraphen zusammengesetzt. Insgesamt ist die Formalisierung in [He et al., 96] jedoch auf einer etwas höheren Abstraktionsebene angesetzt. Die Kamerapositionen werden nicht aus der Menge aller möglichen Positionen mittels *constraint solving* ausgewählt, sondern aus einer kleinen Menge als sinnvoll bekannter Positionen. Die Zustandsgraphen hingegen sind hierarchisch aufgebaut und erlauben eine Formalisierung komplexerer Verhaltensweisen (Siehe Abbildung 3.9, aus [He et al., 96]). Mit Hilfe dieser hierarchisch gegliederten Beschreibung des filmtechnischen Wissens wird nun wieder ein Animationsskript erzeugt, das auf unterster Ebene Kamerapositionen für die verschiedenen Einstellungen beschreibt (Abbildung 3.10, aus [He et al., 96]).

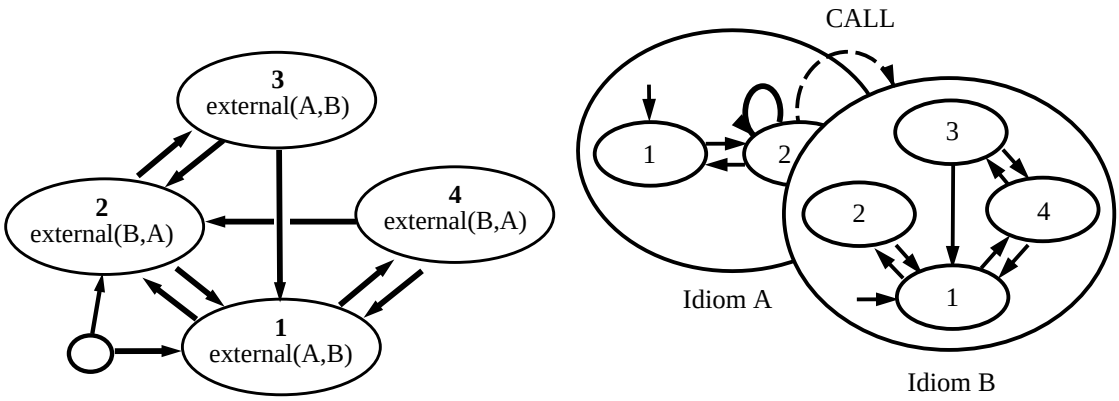


Abbildung 3.9: Einfacher und hierarchischer Zustandsgraph im System VC

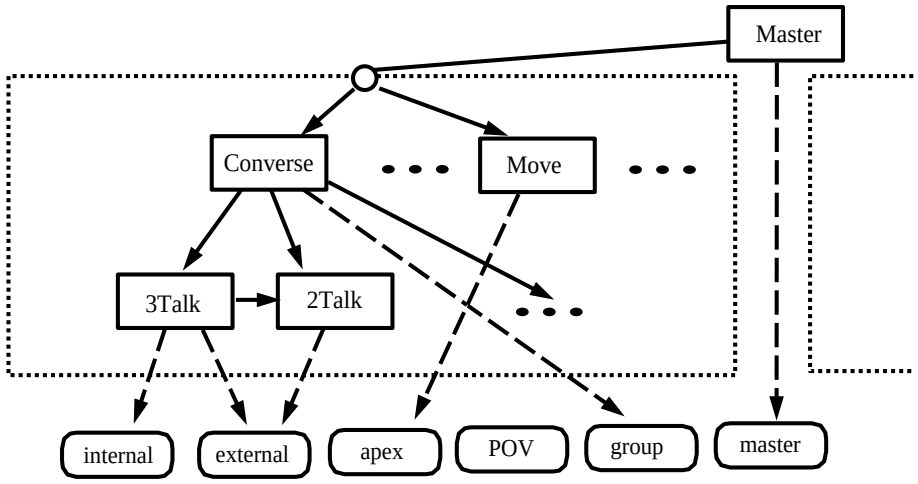


Abbildung 3.10: Skriptplanung des Virtual Cinematographer

Das in [He et al., 96] beschriebene System steuert die Kamera einer Simulationsanwendung und reagiert auf deren Ausgaben durch die Wahl passender Einstellungen (Abbildung 3.11, aus [He et al., 96]). Ein solches Verfahren eignet sich sehr gut zur Kameraführung in virtuellen Welten.

In [Christianson et al., 96] wird zusätzlich die Terminologie der filmtechnischen *Idiome* und *Fragmente* eingeführt und eine deklarative Beschreibungssprache für filmtechnisches Wissen vorgeschlagen. Fragmente sind demnach Untereinheiten einer Einstellung, Idiome Untereinheiten einer Szene, die aber immer noch mehrere Einstellungen enthalten. Diese Formalisierung spiegelt die Erkenntnis wieder, daß eine starre Unterteilung in die vier Ebenen Film, Sequenz, Szene und Einstellung nicht immer ausreichend ist. Sie legt außerdem ein Verständnis der Filmsprache nahe, das dem der natürlichen Sprache gleicht. Idiome sind in beiden Fällen fest konstruierte, häufig und gern verwendete und leicht verständliche Einheiten, die einen bestimmten Sachverhalt ausdrücken.

Ausgehend von einer Deklaration des Filmwissens in DCCL (Declarative Camera Control Language) wird nun eine Filmstruktur abgeleitet, die hierarchisch aufgebaut ist. Für jede Szene des so generierten Films werden jedoch verschiedene Umsetzungsmöglichkeiten generiert und auch bis auf Bildebene berechnet. Die

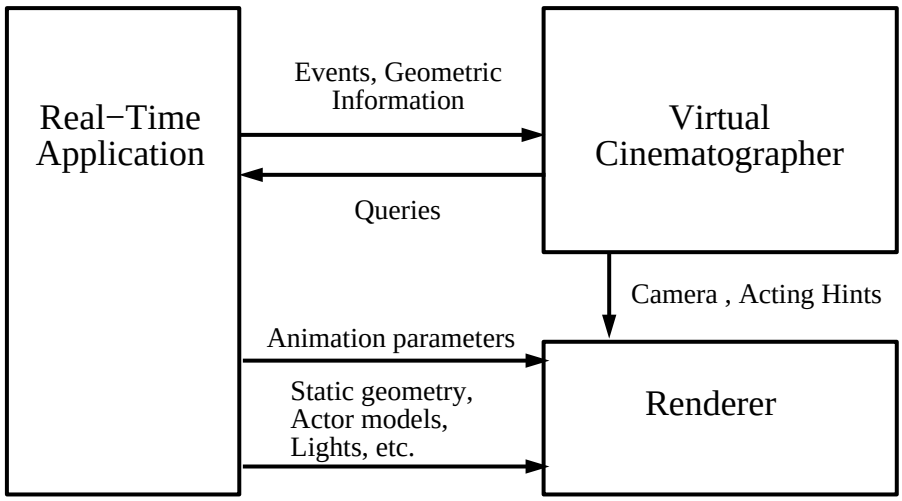


Abbildung 3.11: Einsatzweise des Virtual Cinematographer

so erzeugten Alternativen werden anschließend auf ihre visuelle Qualität überprüft. Die optisch gelungenste Umsetzung liefert schließlich den fertigen Film.

Als Motivation für ein solch komplexes Verfahren geben die Autoren an, daß sich die optische Wirkung einer Einstellung erst am erzeugten Bild beurteilen lasse. Wie dies geschieht, bleibt jedoch unklar. Immerhin lassen sich durch das beschriebene Verfahren filmtechnische Fehler wie das Durchkreuzen der Bewegungsachse verhindern, jedoch bleibt fraglich, ob dies nicht effektiver vor der eigentlichen Bildberechnung behandelt werden könnte.

3.7 RAPID

Im Zusammenhang mit der automatischen Erklärung biologischer Modelle stellen [Bares & Lester, 97b] das System RAPID vor. RAPID erzeugt informative Animationen zur Erklärung biologischer Vorgänge anhand von Polygonmodellen pflanzlicher Strukturen. Es ist somit das Projekt, das dem in der vorliegenden Arbeit beschriebenen System CATHI von der Aufgabenstellung her am nächsten steht. In beiden Fällen werden Animationen erzeugt, die bestimmte Vorgänge und Sachverhalte darstellen und dadurch Informationen vermitteln sollen. Die Funktionsweise RAPIDs ist in Abbildung 3.12 (aus [Bares & Lester, 97b]) dargestellt.

3.7.1 Eingaben des Systems

Eingabe des Systems ist eine Reihe von Anfragen der Art *"Erkläre Funktion X"*, die nacheinander von einer Person gestellt werden, die sich am Bildschirm einen bestimmten Sachverhalt erklären lassen will. Um eine Animation zu erzeugen, die die gestellte Frage beantwortet, greift RAPID auf eine Sammlung pädagogisch motivierter Präsentationsstrategien sowie auf eine Wissensbasis mit Domänenwissen zurück. Dieses Domänenwissen umfaßt die für die Domäne wichtigen Konzepte, jeweils annotiert durch die zugehörigen 3D-Modelle, Bewegungsbeschreibungen, relative Wichtigkeiten und erklärenden Äußerungen. Je nach Kontext werden sogar verschiedene

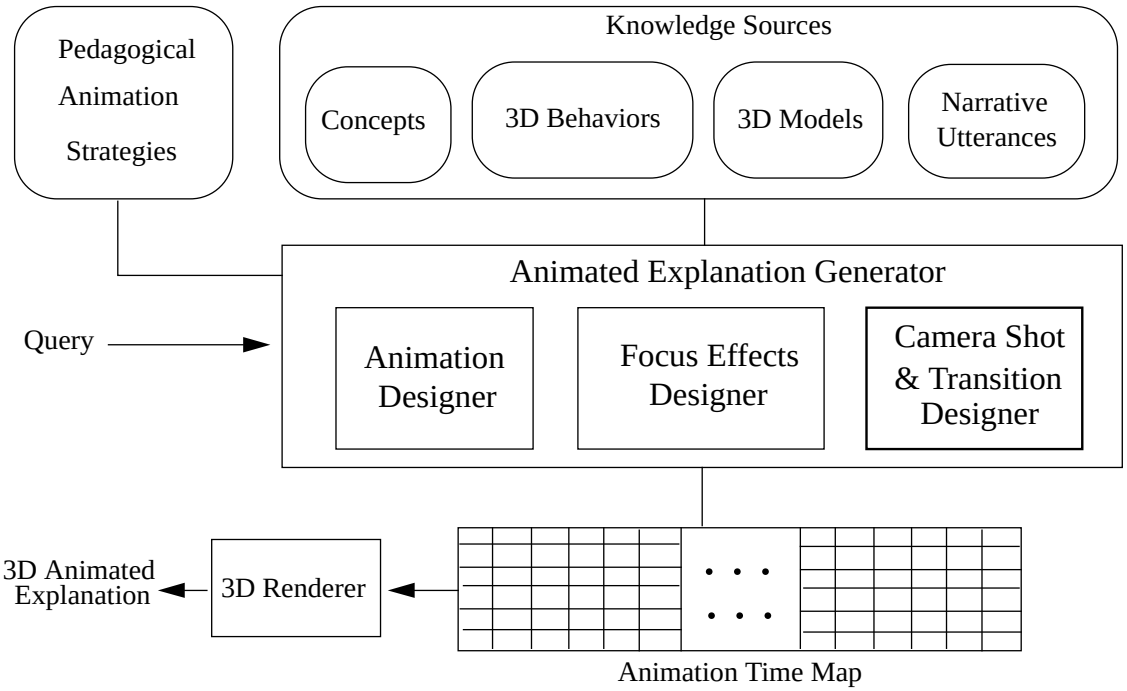


Abbildung 3.12: Arbeitsweise des Systems RAPID

Polygonmodelle des gleichen Weltobjektes verwendet, beispielsweise detailreiche und detailarme Varianten oder vorberechnete Aufrißdarstellungen.

3.7.2 Gestaltung der Animationen

Ausgehend von den aus der Wissensbasis ausgewählten Konzepten und ihren Annotationen wird nun von einer Designkomponente eine attentionale Struktur (Siehe auch Abschnitt 2.3.2.3) der Animation festgelegt, und zwar in Form einer Folge von *Fokuspunkten*, die den Verlauf des inhaltlichen Fokus festlegen. Die Konzepte werden daraufhin in einem vorläufigen Zeitplan der Animation angeordnet, wobei auch berücksichtigt wird, ob ein Konzept bereits vorher verwendet oder eingeführt wurde. Dieser vorläufige Zeitplan wird nun erweitert um graphische Techniken zur Objektfokussierung (Metagraphik, strategische Pausen, Kamerapositionen in Schlüsselszenen) und in einer anschließenden dritten Phase mit den fehlenden Kamerafahrten zwischen den Schlüsselpositionen aufgefüllt. Dabei werden zur Auswahl zwischen Kamerafahrten und Schnitten meist syntaktische Kriterien wie Entfernung und Drehwinkel der Kamera eingesetzt.

3.7.3 Eigenschaften des Verfahrens

Eine solche Vorgehensweise stellt zunächst einmal sicher, daß die attentionale Struktur der Präsentation gut umgesetzt wird. Die filmtechnische Umsetzung eines bestimmten Inhaltes hängt dabei stark von den geometrischen Gegebenheiten der Modellwelt ab und berücksichtigt nur lokale Kriterien des jeweiligen Abschnittes der Animation. Eine globalere Gestaltung, etwa im Sinne der in [Christianson et al., 96] verwendeten Idiome, findet nicht statt und die Kameraführung wird jeweils nur

von Schlüsselszene zu Schlüsselszene geplant. Der Aspekt der Kontinuität in der Kameraführung bleibt jedoch trotzdem nicht unberücksichtigt, da das System bemüht ist, unnötige Schnitte zu vermeiden und wo immer möglich durch eine Kamerafahrt zu ersetzen. Wie die Kamerapositionen konkret gewählt werden, wird in [Bares & Lester, 97b] nicht näher erläutert, die gezeigten Beispiele lassen aber auf die Verwendung eines ähnlichen Verfahrens wie in [Bares & Lester, 97a] schließen (Siehe hierzu Abschnitt 3.8). Die Planung des Animationsskriptes muß auch beim System RAPID zu Beginn der Ausgabe vollständig abgeschlossen sein. Zu jeder eingegebenen Anfrage wird zunächst ein komplettes Animationsskript gestaltet und erst dann mit der Ausgabe begonnen. Immerhin verläuft die Gestaltung so schnell, daß nach Angabe der Autoren schon auf einem PC (Pentium 133, 32MB) innerhalb einer Sekunde mit der Ausgabe begonnen werden kann, was in diesem Zusammenhang den Begriff *Echtzeit* rechtfertigt.

3.8 UCAM

Die Autoren des eben diskutierten Systems RAPID stellen in [Bares & Lester, 97a] ein weiteres System zur automatischen Kameraführung, diesmal jedoch in dynamischen virtuellen Welten vor. Das System UCAM unterstützt Akteure in einer virtuellen Welt beim Erfüllen bestimmter Aufgaben, indem es ihnen die Kameraführung abnimmt und sie automatisiert. Der Stil dieser Kameraführung läßt sich durch ein *kinematographisches Benutzermodell* vorgeben, in dem festgehalten wird, ob der Benutzer längere oder kürzere Einstellungen bevorzugt, ob er eher dramatische oder eher unspektakuläre Kamerapositionen mag, und ob er in bestimmten Situationen lieber mit einem Schnitt oder mit einer ausgedehnten Kamerafahrt konfrontiert werden möchte.

Kernpunkt der Kameraführung ist die automatische Auswahl immer neuer Kamerapositionen, die je nach gewünschter Einstellungslänge seltener oder häufiger erfolgt. Die Kamerapositionen werden stets in Relation zu einem Zielobjekt gewählt

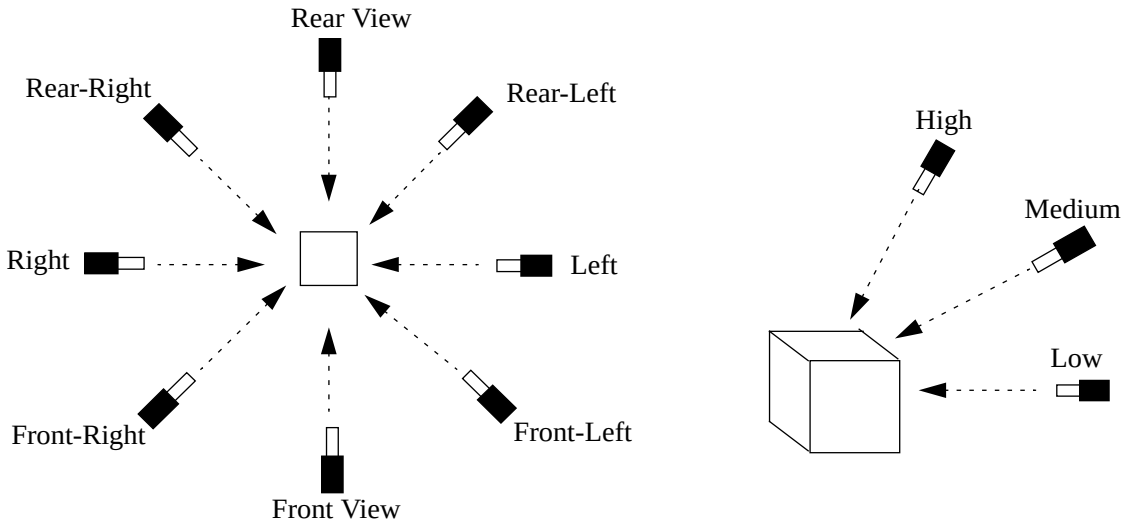


Abbildung 3.13: Wahl der Kameraposition in UCAM

und zeigen dies aus einer von acht Richtungen horizontal und aus einer von drei Höhen vertikal (Abbildung 3.13, aus [Bares & Lester, 97a]). Bewegt sich das Objekt, so bewegt sich auch die Kamera mit, um es zu verfolgen. Neue Betrachtungsrichtungen werden zufällig aus einer der sich ergebenden 24 Möglichkeiten ausgewählt, jedoch sind die Wahrscheinlichkeiten entsprechend den Benutzerpräferenzen für dramatische Kamerapositionierung verteilt. Beim Übergang zur nächsten Kameraposition wird ebenfalls je nach Benutzermodell und geometrischen Gegebenheiten entschieden, ob dies durch einen Schnitt oder eine Kamerafahrt geschehen soll.

Da ein solches Verfahren naturgemäß keine global sinnvolle Abfolge von Kameraeinstellungen generieren kann, sind die erzeugten Animationen zwar visuell mehr oder weniger ansprechend, ein übergreifendes Konzept im Sinne eines Drehbuches existiert jedoch nicht. Dies ist auch nicht ohne weiteres möglich, da die Veränderungen in der Modellwelt nicht vorher bekannt sind, das System jedoch darauf reagieren muß. Da keine Gestaltung auf höherer Ebene erfolgt, traten zunächst filmtechnische Fehler wie zum Beispiel das Kreuzen der Bewegungsachse auf. Dies wurde dadurch umgangen, daß die Auswahl einer neuen Kameraposition auf den Halbraum der gegenwärtigen Kameraposition bezüglich der Bewegungsachse beschränkt wird. Das System generiert auf einem handelsüblichen PC (Pentium 133, 32MB) bis zu 8 Bilder pro Sekunde, wobei von Bild zu Bild die Kameraposition nach den beschriebenen Methoden verändert wird und keine Vorlaufzeit zur Skriptgenerierung benötigt wird. Es fällt in diesem Zusammenhang also in die Kategorie der Echtzeitsysteme.

3.9 Player

In [Kurlander & Ling, 95] wird ein Verfahren vorgestellt, das sich nicht direkt mit der Generierung animierter 3D-Graphiken beschäftigt. Trotzdem enthält es eine sehr interessante Idee, die für eine Weiterentwicklung des in dieser Arbeit beschriebenen Ansatzes relevant ist und soll deshalb hier kurz beschrieben werden. Das von Kurlander und Ling beschriebene System Player ist Bestandteil eines *User Interface Management Systems* (UIMS) und generiert Skripte für Animationen einer Benutzeroberfläche. Als Anwendungsbeispiel wird die Steuerung des animierten Vogels Peedy beschrieben, der auf dem Bildschirm umherfliegt, sitzt, schläft oder redet. Abhängig von Benutzereingaben soll dieser Vogel bestimmte Aktionen am Bildschirm ausführen, deren Steuerskript je nach Ausgangssituation unterschiedlich aussehen muß.

Statt nun Animationsskripte für alle denkbaren Situationen manuell vorzugeben wird ein operatorbasierter Planer eingesetzt, um aus allen denkbaren Ausgangszuständen Pläne zum Erreichen eines spezifizierten Endzustandes zu erreichen. Das Problem, das sich dabei stellt, ist die Komplexität des Planungsverfahrens. Soll der Vogel innnerhalb der Benutzerschnittstelle in Echtzeit auf Benutzereingaben reagieren, so läßt sich eine operatorbasierte Planung mit Suchraum und Backtracking nicht durchführen. Um dieses Problem zu lösen wird der eigentliche Planungsprozeß in das UIMS vorgezogen. Der Autor der Benutzerschnittstelle, die den Vogel Peedy später animieren soll, spezifiziert in einer Beschreibungssprache die möglichen Handlungen des Vogels mit den jeweiligen Vor- und Nachbedingungen. Das Planungsverfahren erzeugt daraus nun einen Zustandsgraphen mit einem Animationsskript für den

Übergang aus jedem denkbaren Zustand der Benutzerschnittstelle in jeden anderen. Dieser Zustandsgraph enthält also alle potentiell möglichen Animationen. Die Anzahl der Zustände wird gering gehalten durch die Zusammenfassung ähnlicher Zustände zu einer Hierarchie aus Zustandsklassen. Das System erzeugt kompilierbaren C++-code, der bei der Kompilierung der Benutzerschnittstelle miteingebunden wird und dort die möglichen Animationen Peedys beschreibt. So kann die Deklarativität eines operatorbasierten Planungsverfahrens ausgenutzt werden, ohne dadurch einen Performanzverlust in Kauf nehmen zu müssen. In Kapitel 9 werde ich auf diese Idee kurz zurückkommen.

3.10 Zusammenfassung

Aus dem weiten Spektrum und der Aktualität der besprochenen Systeme ist zu ersehen, daß die automatische Generierung animierter Präsentationen ein wachsendes Forschungsgebiet darstellt. Die genannten Arbeiten sind dabei nur diejenigen, die einen engen Bezug zur vorliegenden Arbeit aufweisen. Die Systeme decken zusammengenommen viele Aspekte der automatischen Animationsgenerierung ab, jedoch fehlt eine umfassende Integration aller für eine gelungene 3D-Animation wichtigen Aspekte in einem einzigen Ansatz. Die Berechnung von Kamerapositionen wird vor allem im Bereich virtueller Welten untersucht, während die Systeme, die sich um eine gelungene filmtechnische Gestaltung bemühen, meist auf Defaultpositionen zurückgreifen. Die Verfahren, die auf klassische Filmtechniken zurückgreifen, lassen meist die für das Medium Computergraphik spezifischen Techniken außer acht, während die Systeme, die sich beispielsweise metagraphischer Techniken bedienen, nur ein eingeschränktes filmtechnisches Repertoire bieten. Schließlich wird auch in keinem der besprochenen Systeme eine filmtechnisch motivierte Skriptplanung mit der Fähigkeit kombiniert, Animationen unter Zeitdruck zu generieren. In Kapitel 4 werde ich auf diese Defizite näher eingehen und einen Ansatz vorschlagen, der alle genannten Aspekte berücksichtigt. Die folgende Tabelle zeigt die Leistungsmerkmale der vorgestellten Systeme nochmals im Überblick. Hierbei bedeutet "Filmt." die Verwendung filmtechnischen Wissens, "Meta" die Verwendung metagraphischer Techniken wie Pfeile und Explosionsdarstellungen, "LOD" die Verwendung verschiedener Abstraktionsgrade (levels of detail) und "DOF" die Verwendung von Tiefenschärfe (depth of field). Die letzte Zeile zeigt unter dem Systemnamen CATHI die Merkmale des in dieser Arbeit entwickelten Ansatzes.

System	inkr.	Echtzeit	Filmt.	Kamera	Schnitt	Licht	Meta	LOD	DOF
BETTY	nein	nein	ja	ja	ja	nein	nein	nein	nein
Zoom Ill.	teil	ja	ja	ja	nein	nein	ja	nein	nein
Esplanade	teil	nein	ja	default	ja	nein	nein	nein	nein
Camdroid	teil	ja	ja	ja	ja	nein	nein	nein	nein
Virt. Cin.	teil	ja	ja	default	ja	nein	nein	nein	nein
RAPID	teil	ja	(ja)	(ja)	ja	nein	ja	(ja)	nein
UCAM	teil	ja	nein	ja	(ja)	nein	nein	nein	nein
CATHI	voll	ja	ja	ja	ja	ja	ja	ja	ja

Dritte Szene

Nach einiger Zeit ist der Ordner durchgearbeitet. Die Artikel liegen auf dem Tisch verstreut mit rot unterstrichenen und umrahmten Stellen. Auf einem Notizblock sind Ideen vermerkt.

C: ...Na gut, und was willst du nun besser machen als die?

B: Also zuerst mal generieren die alle ihr gesamtes Drehbuch fertig, bevor sie es weitergeben zum Rendern.

A: (fällt B ins Wort) Na, das ist doch klar. Ich würde ja auch nie ein unfertiges Drehbuch aus der Hand geben.

B: Auch wenn du weißt, daß die ersten Kapitel schon fertig sind und dein Produzent dir Druck macht weil er mit dem Drehen anfangen will?

A: Nun gut, wenn du das so sagst...

B: (tippt auf den Notizblock) Außerdem steht da nur die Hälfte drin, in diesen Drehbüchern. (zu C) Du hast ja vorhin allerhand aufgezählt, was man in einer Animation so machen kann. Da hab ich sofort ein paar Ideen bekommen.

C: Na dann bin ich aber mal gespannt...

A: (überfliegt nochmals seinen Block) Also, ich habe da sogar schon eine Idee, wie wir das vom Prinzip her angehen könnten. Ich erkläre euch das mal kurz...

Kapitel 4

Ein neuer Ansatz zur automatischen Animationsgenerierung

Nachdem im vorangegangenen Kapitel einige verwandte Arbeiten besprochen wurden, möchte ich in diesem Kapitel zunächst aufzeigen, welche bei der Animationsplanung auftretenden Probleme bisher weitgehend unbeachtet bleiben. Anschließend sollen der Umfang und die Elemente der in der vorliegenden Arbeit verwendeten Bildsprache sowie die Grundkonzeption des von mir vorgestellten Ansatzes erläutert werden.

4.1 Defizite der bisherigen Ansätze

Einige der in meiner Arbeit behandelten Probleme sind jeweils für sich genommen schon in einem oder mehreren der in Kapitel 3 beschriebenen Systeme untersucht worden. Hierbei fällt jedoch auf, daß die folgenden Aspekte in der bis zum aktuellen Zeitpunkt veröffentlichten Literatur fast gänzlich fehlen:

4.1.1 Inkrementelle Generierung unter Zeitdruck

Die zur Planung von Drehbüchern eingesetzten Verfahren lassen großenteils den Aspekt der Generierungszeit außer Acht. (Dies trifft nicht für die genannten Echtzeitsysteme zu. Allerdings kann dort meist nicht von der Erzeugung eines echten Drehbuches gesprochen werden.) Die Systeme erzeugen zunächst ein vollständiges Drehbuch und beginnen erst dann mit der Bildberechnung und Ausgabe der Animation. Der Zeitablauf einer solchen *nichtinkrementellen* Animationsgenerierung ist in Abbildung 4.1 dargestellt. In einigen der zitierten Arbeiten sind sogar Tabellen enthalten, in denen die Generierungszeiten für verschiedene Animationen angegeben sind. Diese Generierungszeiten bewegen sich meist im Bereich einer oder mehrerer Minuten, je nach Komplexität des Verfahrens und Rechenleistung der verwendeten Maschine. Ein nichtinkrementelles Generierungsverfahren wurde beispielsweise im beschriebenen System BETTY verwendet. Besonders interessant ist, daß die angegebenen Generierungszeiten in fast allen Fällen in der gleichen Größenordnung

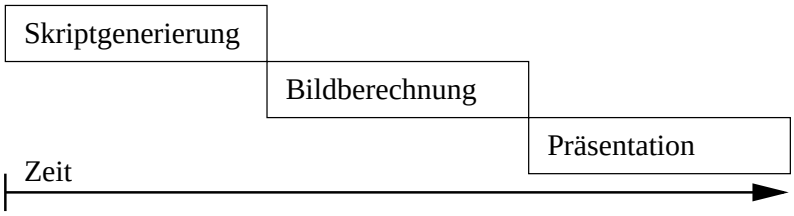


Abbildung 4.1: Nichtinkrementelle Animationsgenerierung

liegen wie die Dauer der zugehörigen Animationen. Durch ein geschicktes Aufteilen der Rechenleistung und ein Überschneiden von Generierungs-, Bildberechnungs- und Präsentationszeit wären fast alle beschriebenen Systeme dazu in der Lage, innerhalb weniger Sekunden mit einer Ausgabe der Animation zu beginnen, ohne dabei auf komplexe Generierungsstrategien zu verzichten. Teilinkrementelle Verfahren (Abbildung 4.2 links) überlagern immerhin zwei der drei ablaufenden Prozesse, typischerweise die Bildberechnung und -ausgabe. Solche Verfahren werden beispielsweise in den Systemen Zoom Illustrator, ESPLANADE, Camdroid, Virtual Cinematographer und RAPID verwendet.

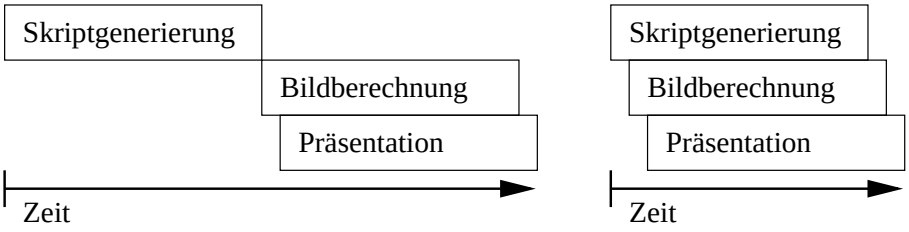


Abbildung 4.2: Teilinkrementelle und Inkrementelle Animationsgenerierung

Sobald ein System aber in der Lage ist, auch den Prozeß der Skriptgenerierung zeitlich mit den beiden nachfolgenden Prozessen zu überlagern (Abbildung 4.2 rechts), verringert sich die Verzögerung vom Start der Generierung bis zum Beginn der Präsentation nochmals drastisch. (Die verschiedenen Formen der Inkrementalität werden unter anderem in [Finkler, 96] ausführlich diskutiert.) Das im vorangehenden Kapitel diskutierte System UCAM ist ein solches vollständig inkrementell arbeitendes System bezüglich seiner Ausgabe. Allerdings generiert es immer nur lokal begründete Kameraeinstellungen und nicht ein global strukturiertes Drehbuch.

Ein vollständig inkrementell arbeitendes Skriptgenerierungssystem ist nur mit einem inkrementellen Ansatz zur Skriptgenerierung möglich, wie er in dieser Arbeit vorgestellt wird. Das Kräftedreieck aus Rechenleistung, Rechenzeit und Komplexität des Generierungsverfahrens wird oft als unumstößlich hingenommen, obwohl es sich durch einen inkrementellen und ressourcenadaptiven Generierungsansatz erheblich dehnen läßt. Das Problem der Skriptgenerierung unter beschränkten Ressourcen ist in die vorliegende Arbeit auf verschiedene Arten eingeflossen, und im Verlauf des Kapitels 6 werde ich die gefundenen Lösungen näher erläutern.

4.1.2 Gestalterische Funktion des Lichtes

Ein weiteres Problem, das in der vorliegenden Literatur weitgehend ignoriert wird, ist die gezielte Beleuchtung von Animationssequenzen. Während eine akzeptable Grundausleuchtung in vielen Fällen durch feste Vorgaben erreicht werden kann, wird die gestalterische Funktion des Lichtes völlig außer Acht gelassen. Einige Systeme verlangen eine Beleuchtung als Teil der Modellwelt oder der sie erzeugenden Simulation, andere geben eine für alle Modellwelten gültige Standardbeleuchtung vor. Keines der vorgestellten Systeme arbeitet jedoch explizit mit Lichteffekten als Teil der Bildsprache. Dies ist wohl teilweise der Tatsache zuzuschreiben, daß viele Beleuchtungseffekte in der Filmsprache nur unbewußt wahrgenommen werden und daher für den Laien sehr schwer zu erkennen sind. Da aber die überwiegende Mehrzahl der Systeme zur automatischen Animationsgenerierung von filmtechnischen Laien – Informatikern nämlich – konzipiert werden, stehen meist die Aspekte im Vordergrund, die sich auch leicht analysieren lassen: Kameraführung und Schnitt. Die Einschränkung auf diese beiden Stilmittel bedeutet jedoch, ein erhebliches Potential an Ausdrucksmöglichkeiten zu verschenken.

4.1.3 Ausnutzung des Mediums Computergraphik

Ein System, das die Ausdrucksstärke des Mediums 3D-Animation optimal ausnutzen soll, muß aber nicht nur die Kameraführung und die Beleuchtung gezielt in den Gestaltungsprozeß miteinbeziehen. Die Computergraphik bietet weitere Stil- und Ausdrucksmittel, die im klassischen Film so nicht vorzufinden sind. Ein Beispiel hierfür ist der Einsatz metagraphischer Techniken wie Abstraktion, Explosionsdarstellung oder die Verwendung von Zeige- und Bewegungspfeilen, ein anderes der Einsatz von Transparenz oder Farbeffekten. Eine Ausnahme hiervon bildet bis zu einem gewissen Grad das System Zoom Illustrator, das sowohl Transparenzeffekte als auch Metagraphik zur Präsentation einsetzt. Alle anderen besprochenen filmtechnisch motivierten Ansätze lassen jedoch diese Techniken völlig außer Acht. Die Bildsprache der Computeranimation ist jedoch in mancher Hinsicht umfangreicher als die des Filmes, und die zusätzlichen Ausdrucksmittel nicht zu verwenden bedeutet ein Verschenken gestalterischer Ressourcen.

4.2 Umfang der verwendeten Bildsprache

Der Umfang der in der vorliegenden Arbeit verwendeten Bildsprache überschreitet den aller beschriebenen Systeme. Durch eine Einbeziehung der Beleuchtung, des Detaillierungsgrades, der Materialeigenschaften und metagraphischer Techniken in die Gestaltung wird eine sehr hohe Ausdrucksstärke der erzeugten Animationen erreicht. Bevor sich nun eine Grammatik dieser Bildsprache formulieren läßt, müssen zunächst ihre Elemente erkannt und klassifiziert werden. Hierzu möchte ich die in meiner Arbeit verwendeten bildsprachlichen Mittel in verschiedene Bereiche aufteilen und kurz beschreiben.

4.2.1 Kameraführung

Die Kameraführung ist der Aspekt, der dem Betrachter eines Films oder einer Animation außer den dargestellten Dingen, Personen und Handlungen am ehesten bewußt wird. Allein durch die Wahl der Kameraposition wird beispielsweise bestimmt, wie viel einer Szenerie gezeigt wird und in welchem räumlichen Verhältnis die gezeigten Dinge stehen. In der klassischen Filmtechnik unterscheidet man hierzu fünf Entfernungsklassen von der *Totale* bis zur *Nahaufnahme*, die sich dadurch unterscheiden, wie viel von den gezeigten Schauspielern zu sehen ist.

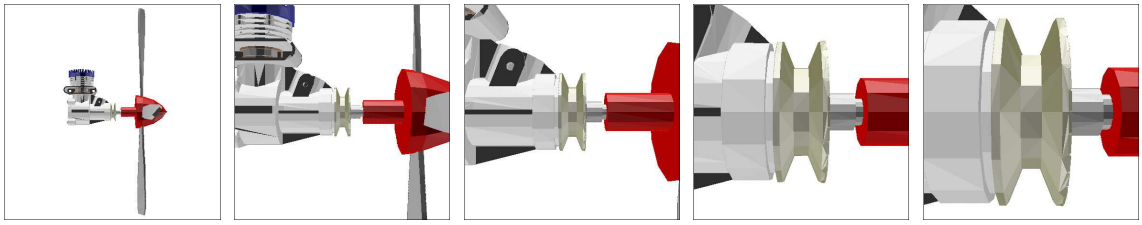


Abbildung 4.3: Totale, Halbtotale, Mittelnahe, Halbnahe und Nahaufnahme

Auf die Computergraphik lassen sich diese Klassen nur abgewandelt übertragen, da wir es dabei in der Regel nicht mit Schauspielern, sondern mit Modellen beliebiger Objekte zu tun haben (Siehe Abbildung 4.3). Trotzdem kann auch hier der Übergang von einer *Totalen* (Objekt mit seinem gesamten Umfeld) oder *Halbtotalen* (Objekt mit einem Teil seines Umfelds) zur *Nahaufnahme* in Form einer *Zufahrt* ausgenutzt werden, um Objekte in ihr räumliches Umfeld einzuordnen. Ein ähnlicher Effekt wird durch eine *Rückfahrt* erzeugt, nur daß hierbei der visuelle Fokus am Ende der Einstellung nicht auf dem fraglichen Objekt, sondern auf seinem Umfeld liegt.

Kamerafahrten können dazu eingesetzt werden, Objektbewegungen zu verfolgen, wobei das Objekt einerseits jederzeit ausreichend groß sichtbar ist, andererseits aber nie das Bildfeld verläßt, wie es bei einer festen Kameraposition der Fall wäre. Eine andere Einsatzmöglichkeit für Kamerafahrten ist es, Objekte aus verschiedenen Richtungen zu zeigen und damit ihre räumliche Anordnung beziehungsweise ihre Lage im Umfeld zu unterstreichen. Das *Zoomen* (Verändern der Brennweite des Kameraobjektivs) ist eine Technik, die beim Drehen echter Filme eine Kamerazufahrt ersetzen kann (Abbildung 4.4 Mitte). Da es technisch viel einfacher umzusetzen ist, können so Einstellungen gedreht werden, die als Kamerafahrt nicht umsetzbar wären, weil sich beispielsweise Hindernisse auf dem Weg der Kamera befinden. Da sich jedoch beim zoomen nur der Bildausschnitt verändert, während die Perspektive (bedingt durch die feste Kameraposition) gleich bleibt, entsteht ein unnatürlicher Bildeindruck, der in der Regel unerwünscht ist. Dieser Eindruck ist deshalb unnatürlich, weil sich die Brennweite des menschlichen Auges im Gegensatz zur Kamera nicht verändern läßt. Will man als Mensch also einen Gegenstand größer sehen, so ist man gezwungen, sich näher heran zu bewegen. Dabei ändert sich automatisch die Betrachtungsperspektive. Da die geschilderten technischen Hindernisse in der Computergraphik aber nicht auftreten, ist es hier in den allermeisten Fällen sinnvoller, statt eines Zooms eine Kamerazufahrt einzusetzen (Abbildung 4.4 unten).

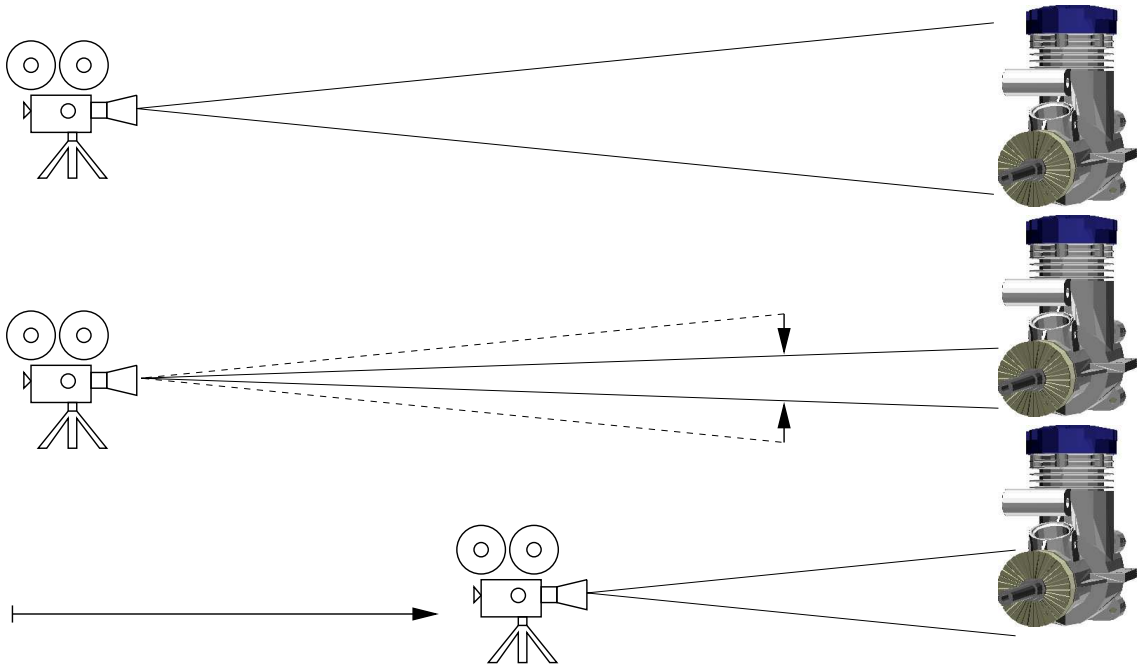


Abbildung 4.4: Kamerazoom und Kamerazufahrt

4.2.2 Schnitt

Während der Schnitt beim Drehen eines Filmes eine technische Notwendigkeit ist, könnte man ihn theoretisch in der Computergraphik völlig außer Acht lassen. Eine Animation, die aus einer einzigen Einstellung bestünde, würde jedoch einen seltsamen Eindruck hinterlassen, da wir es durch unsere tägliche Konfrontation mit gefilmtem Material gewohnt sind, die Sprache des Schnittes unbewußt mitaufzunehmen. Ein Schnitt kann versteckt und unauffällig sein, beispielsweise um während einer Konversation die beiden Gesprächspartner aus verschiedenen Winkeln zu zeigen, oder um eine langen Zufahrt durch Auslassen eines Stückes in der Mitte zu verkürzen. Ein Schnitt kann aber auch ein deutlich wahrnehmbares Element der Bildsprache sein, das dem Betrachter vermittelt, daß in der anschließenden Einstellung eine neue Person, Handlung oder ein neuer Ort gezeigt wird. Solche Schnitte haben eine wichtige Funktion innerhalb der Filmsprache und sie tragen wesentlich zur visuellen Verständlichkeit eines Films oder einer Animation bei.

4.2.3 Beleuchtung

Bei Photographen und Malern läßt sich das bewußte Arbeiten mit Licht recht gut beobachten, da man als Betrachter genügend Zeit hat, das statische Bild zu analysieren. Im Medium Film oder Computeranimation ist dies nicht mehr so einfach, da sich die Lichtverhältnisse oft ändern, noch bevor sie überhaupt bewußt wahrgenommen wurden. Die eingesetzten Lichteffekte erzeugen ihre Wirkung unterbewußt, und zusammen mit der Kameraführung und dem Schnitt bilden sie eine gestalterische Einheit, die maßgeblich an der visuellen Qualität des Ergebnisses beteiligt ist. Solche Beleuchtungseffekte sind beispielsweise das Hervorheben bestimmter Bildbereiche durch eine besondere Ausleuchtung oder das Erzeugen von *Lichtstimmungen* durch

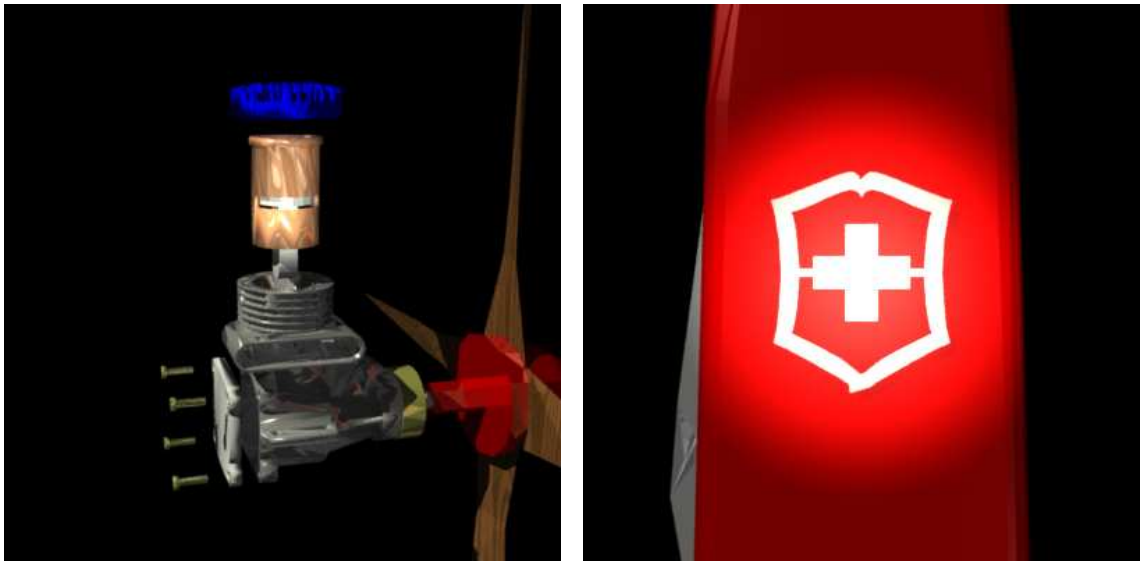


Abbildung 4.5: Hervorhebung von Objekten durch die Beleuchtung

eine bewußte Beeinflussung der *Farbtemperatur*. So treten Gegenstände, die in einem sonnenbeschienenen Bereich am Fenster oder im Lichtkegel eines Scheinwerfers stehen, optisch in den Vordergrund, während andere im Halbdunkel des umgebenden Raumes nahezu verschwinden (Siehe Abbildung 4.5). Eine solche vom Betrachter nicht bewußt wahrgenommene Lenkung des visuellen Fokus ist wesentlich effektiver als die indirekte Lenkung der Aufmerksamkeit durch Pfeile oder Annotationen, da keine zusätzlichen Zeichen wahrgenommen und interpretiert werden müssen.

4.2.4 Gezielter Einsatz von Abstraktion

Das gleiche Argument gilt für die Lenkung des visuellen Fokus durch den gezielten Einsatz verschiedener Detaillierungsgrade der verwendeten Modelle. In [Butz & Krüger, 96, Butz & Krüger, 97] werden die formalen Hintergründe dieses Effektes beschrieben. Werden in einer Szenerie fast alle Objekte mit relativ wenig Detail gezeigt während ein einzelnes Objekt sehr detailliert erscheint, so bleibt der Blick des Betrachters automatisch an diesem detaillierten Objekt hängen. Wechselt man außerdem noch den Abstraktionsgrad vor laufender Kamera, so kann man damit den Blick eines Betrachters sehr effektiv lenken ohne den Umweg über metagraphische Mittel gehen zu müssen (Abbildung 4.6).

Dieses bildsprachliche Mittel ist eine spezielle Eigenschaft des Mediums Computergraphik, da es den Umgang mit mehreren Abstraktionsgraden eines Modells voraussetzt. Im Film kann es höchstens durch Weichzeichnung oder Tiefenschärfe simuliert werden. Während der Einsatz mehrerer Detaillierungsgrade der verwendeten Modelle in Computerspielen, Simulationen und virtuellen Welten durchaus gebräuchlich ist, ist ihr *gezielter* Einsatz sehr selten. Der verwendete Detaillierungsgrad eines Modells wird meist einfach durch seine Entfernung zur Kamera bestimmt. Objekte, die weit entfernt sind, werden abstrahiert dargestellt, während nahe Objekte in vollem Detailumfang gerendert werden, auch wenn sie den Betrachter eigentlich gar nicht interessieren. Dies bedeutet einerseits einen erhöhten Rechenaufwand für die detaillierten Modelle, andererseits aber auch, daß ein bildsprachliches Mittel

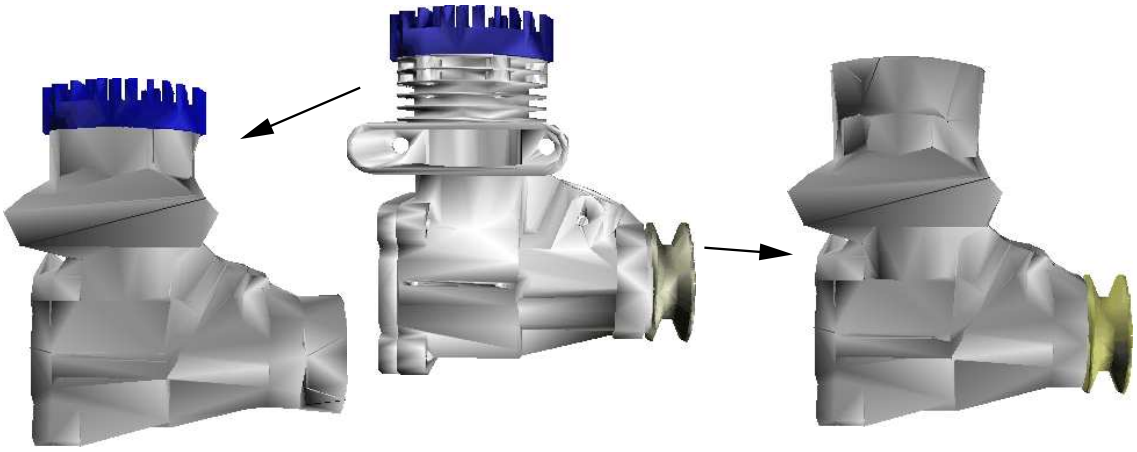


Abbildung 4.6: Steuerung des visuellen Fokus durch Abstraktion

überhaupt nicht beziehungsweise willkürlich eingesetzt wird, was die visuelle Qualität der Animation teilweise sogar reduziert.

4.2.5 Transparenz und Farbe

Eine weitere Eigenschaft des Mediums Computergraphik ist es, die Materialeigenschaften der dargestellten Gegenstände beliebig verändern zu können. Was im Film unmöglich ist, wird in der Computeranimation zum gebräuchlichen Mittel: Störende Gegenstände, die ein Zielobjekt verdecken, werden einfach durchsichtig gemacht, wodurch sie optisch zwar noch vorhanden sind, den Blick des Betrachters auf das Zielobjekt aber fast nicht mehr beeinträchtigen. Die Visualisierung technischer Geräte wird durch die Einbeziehung dieses bildsprachlichen Elementes aus der technischen Graphik wesentlich bereichert und die erzeugten Animationssequenzen haben einen sehr hohen informativen Gehalt, da sie außer der Freistellung verdeckter Objekte auch in der Lage sind, deren enges räumliches Umfeld beizubehalten (Abbildung 4.7). Außerdem ist es sehr einfach, Objekte farblich hervorzuheben. Indem die Ei-

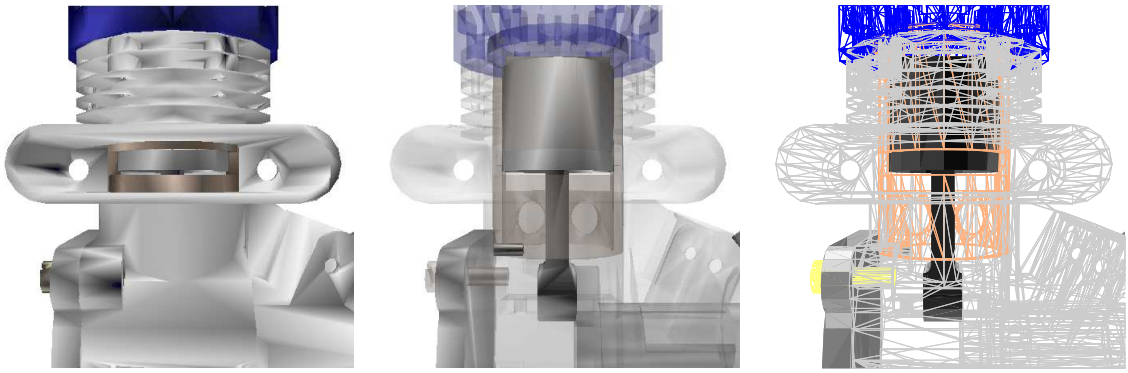


Abbildung 4.7: Transparente Objekte zur Beseitigung von Verdeckungen

genfarbe eines dargestellten Objektes rhythmisch verändert wird, kann man das Objekt in der Animation blinken lassen und somit die Aufmerksamkeit explizit darauf lenken (Abbildung 4.8). Diese Technik ist nicht so subtil wie die geschilderten

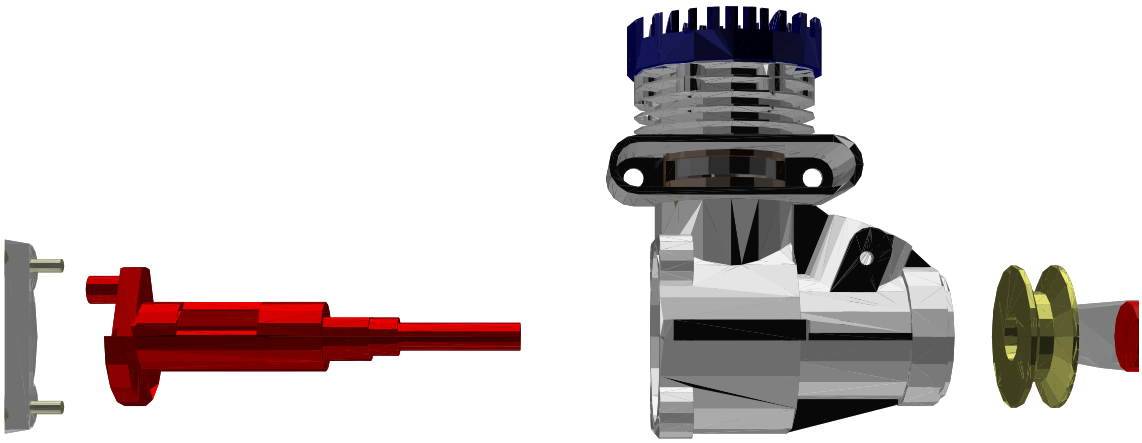


Abbildung 4.8: Farbliche Hervorhebung eines Objektes

Licht- und Abstraktionseffekte, sie kommt aber immer noch ohne zusätzliche Bildelemente wie Pfeile oder Beschriftungen aus. Sorgfalt ist geboten bei der Wahl der Farbe, in der ein Objekt blinkt. Ein rotes Aufleuchten ist insbesondere bei einem von Natur aus roten Objekt sinnlos. Soll ein Objekt hervorgehoben werden, so läßt man es zwischen seiner Eigenfarbe und deren Komplementärfarbe wechseln. Nach [Itten, 87] bildet jede Farbe auf natürliche Weise mit ihrer eigenen Komplementärfarbe einen starken Kontrast, so daß eine Signalwirkung sichergestellt ist. Ausnahmen bilden Farben, die im Farbraum recht nahe bei einem mittleren Grau liegen. Deren Komplementärfarbe ergibt wiederum eine Farbe nahe dem mittleren Grau, so daß die Kontrastwirkung nicht mehr zustandekommt. Ab einer bestimmten Nähe einer Farbe zum mittleren Grau wird daher zur Hervorhebung die Farbe Rot verwendet, die an sich schon mit einer starken Signalwirkung verbunden ist.

4.2.6 Metagraphik

Metagraphische Elemente der Bildsprache im näheren Sinne sind alle zusätzlich zur dargestellten Modellwelt ins Bild eingeführten Bildelemente. Die gebräuchlichsten Beispiele sind verschiedene Arten von Pfeilen (Bewegungs- oder Zeigepfeile, Abbildung 4.9) und Textannotationen. Diese haben die Funktion, die Aufmerksamkeit des Betrachters auf bestimmte Bildbereiche zu lenken (Zeigepfeile), Bewegungen von Objekten anzudeuten (Bewegungspfeile) oder Objekte in der Modellwelt mit zusätzlichen Informationen zu versehen (Annotation).

Im weiteren Sinne zählen zu den metagraphischen Elementen der Bildsprache auch illustrative Techniken aus der technischen Dokumentation. Der Aufbau eines zusammengesetzten Objektes kann beispielsweise in einer Graphik sehr effektiv durch eine Explosionszeichnung dargestellt werden. In einer Computeranimation kann man zusätzlich hierzu das Objekt noch allmählich explodieren lassen, was dazu führt, daß sein räumlicher Aufbau sehr plastisch dargestellt wird.

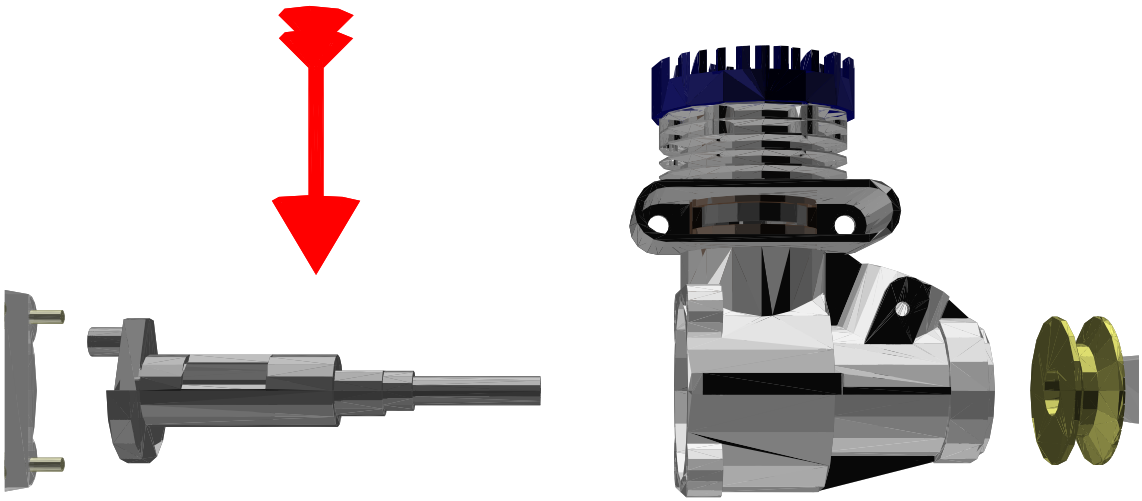


Abbildung 4.9: Zeigepfeil zur Hervorhebung eines Objektes

4.2.7 Selektive Schärfe

Das Objektiv einer Filmkamera bildet nie alle gezeigten Gegenstände wirklich scharf ab, sondern immer nur einen bestimmten Bereich des Raumes. Je nach gewählter Blendenöffnung ist dieser Bereich, den man auch *Schärfentiefe* nennt, unterschiedlich groß. Objekte innerhalb der Schärfentiefe werden scharf abgebildet, Objekte davor und dahinter mehr oder weniger unscharf. In Film und Photographie wird dieser Effekt ausgenutzt, um Personen oder Gegenstände visuell von ihrem Umfeld oder Hintergrund zu lösen (Abbildung 4.10).

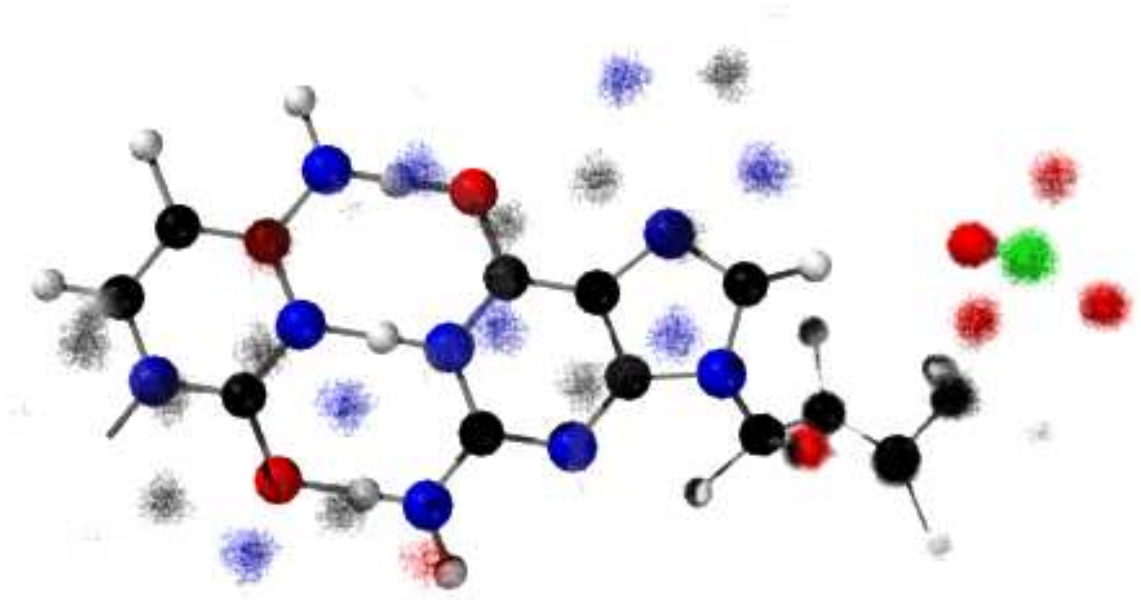


Abbildung 4.10: Herauslösen einer Struktur durch selektive Schärfe

Liegen zwei Objekte in der Modellwelt nun entlang der optischen Achse in unterschiedlichem Abstand zur Kamera, so ist das Verlagern der Schärfeebene ein weiteres effektives Mittel zur Lenkung des visuellen Fokus. Da der geschilderte optische Effekt aber nur mit hochwertigen (und daher langsamen) Rendering-Verfahren

dargestellt werden kann, beschränkt sich seine Anwendbarkeit auf die Erzeugung von Animationen ohne Zeitdruck.

4.3 Grundkonzeption des Generierungsansatzes

Aus den geschilderten Vorüberlegungen sowohl zur allgemeinen Gestaltung von Animationen (Kapitel 2) als auch zu den Defiziten der existierenden Verfahren ergab sich der vorliegende Ansatz zur inkrementellen Skriptgenerierung. Bevor in den folgenden Kapiteln der Ansatz in all seinen Einzelheiten vorgestellt wird, möchte ich hier einen Überblick über seine Grundkonzeption geben.

4.3.1 Hybrider Aufbau

Zunächst wurde in Abschnitt 2.2 gezeigt, daß zur Beschreibung einer Animation sehr viele geometrische Spezifikationen notwendig sind, die sich durch Vektorberechnungen im dreidimensionalen Raum finden lassen. Solche Spezifikationen sind beispielsweise Kamerapositionen und -orientierungen, Objektbewegungen und Verdeckungen, aber auch die Beleuchtung und die Positionierung metagraphischer Elemente. Dieser Problemkreis legt einen Ansatz auf einer recht mathematischen beziehungsweise prozeduralen Ebene nahe.

Auf der anderen Seite haben wir in Abschnitt 2.3 gesehen, daß sowohl die in einer informativen Animationssequenz enthaltene Information als auch ihr rhetorischer Aufbau hierarchisch strukturiert sind. Berücksichtigt man zusätzlich, daß ein herkömmlich produzierter Film eine ähnliche hierarchische Struktur aufweist (Siehe Abschnitt 2.1) und sich die hierarchische Beschreibung von Animationsskripten bereits an anderer Stelle bewährt hat (Siehe Abschnitt 2.2), so drängt sich ein Formalismus auf, der in der Lage ist, ein solches hierarchische Skript zu erzeugen. Solche Formalismen findet man beispielsweise in der Form regulärer, kontextfreier, kontextsensitiver oder allgemeiner Chomsky-Grammatiken.

Um diese beiden sehr verschiedenen Ideen zur Lösung des Problems miteinander zu verbinden, schlage ich einen hybriden Ansatz vor, der die Stärken beider Teillösungen ausnutzt. Die erforderliche hierarchische Struktur wird in meinem Ansatz durch eine kontextgesteuerte Expansion der Produktionsregeln einer Skriptgrammatik erzeugt, die notwendigen geometrischen Berechnungen sind in Form kleiner Programmteile realisiert, die an den entsprechenden Stellen der Expansion benutzt werden, um die erforderlichen geometrischen Spezifikationen in die erzeugte Skriptstruktur einzusetzen. Diese Vorgehensweise erlaubt einerseits die elegante Definition einer Skriptgrammatik durch eine Menge von terminalen und nichtterminalen Produktionsregeln, andererseits können beliebig komplexe mathematische Berechnungen in den entsprechenden Prozeduren ausgelagert werden und stören nicht die Übersichtlichkeit bei der Definition der Grammatik.

4.3.2 Skriptgrammatik

Die Skriptgrammatik gibt für alle vorkommenden Typen von Präsentationszielen (Siehe Abschnitt 2.3) eine Dekomposition in Unterziele an. Diese Dekomposition ist

abhängig vom Kontext und kann in verschiedenen Situationen verschiedene Mengen von Unterzielen liefern. Die Skriptgrammatik ist also kontextsensitiv. Die einfachsten der Präsentationsziele werden schließlich direkt in film- beziehungsweise animationstechnische Einheiten übersetzt, wodurch eine filmtechnische Struktur entsteht, die mit der intentionalen Struktur übereinstimmt. Da zu jedem Präsentationsziel genau eine Dekompositionsregel existiert, entsteht kein Suchraum, sondern genau eine (je nach Situation unterschiedliche) Dekomposition.

4.3.3 Inkrementalität und Adaptivität

Erzeugt man das Animationsskript in chronologischer Reihenfolge, so lassen sich abgeschlossene Einheiten finden, die nach ihrer vollständigen Expansion bereits dargestellt werden können, während die nachfolgenden Skriptteile erst generiert werden. Aus diesem Grunde geschieht die Expansion des Animationsskriptes zuerst in die Tiefe (*Tiefenexpansion*). Sobald auf diese Art ein abgeschlossener Skriptteil expandiert ist, wird er zur Ausgabe weitergereicht und so kann die Präsentation bereits nach der Expansion des ersten Skriptteils beginnen. Das Generierungsverfahren arbeitet also vollinkrementell und verkürzt dadurch die Verzögerung von der Eingabe des Präsentationszieles bis zum Beginn der Präsentation erheblich. Repräsentiert man außerdem im Kontext der Skriptgrammatik die Beschränkungen des Ausgabemediums und die aktuellen Zeiten im Generierungs- und Präsentationsprozeß, so können diese Faktoren ebenfalls die Expansion der Grammatikregeln steuern. Eine bestimmte Dekompositionsregel kann in Abhängigkeit vom Ausgabemedium oder in Abhängigkeit von der verfügbaren Generierungszeit verschiedene Mengen von Unterzielen liefern, und die Generierung wird dadurch auf doppelte Weise ressourcenadaptiv.

4.3.4 Geometrische Berechnungen

Die Berechnung der notwendigen geometrischen Daten wie Kamerapositionen, Bildwinkel, Objektpositionen und Beleuchtung wird prozedural beschrieben. Hierdurch besteht die Möglichkeit, beliebig komplexe Berechnungen mit Iteration oder Rekursion im vollen Sprachumfang einer Programmiersprache auszudrücken, ohne dazu umständlich den Formalismus der Skriptgrammatik zu erweitern. Diese Vorgehensweise ermöglicht einerseits die effiziente Ausnutzung der zur Verfügung stehenden Rechenleistung und trägt andererseits dazu bei, in der Skriptgrammatik von den mathematischen Details der Berechnungen zu abstrahieren und die zugehörigen Prozeduren als *black box* zu behandeln. Diese Sichtweise läßt sich auch aus der Überlegung motivieren, daß ein Filmregisseur nie alle Kameraparameter bis ins Detail angibt, sondern dies dem Kameramann überläßt.

Vierte Szene

C: Schön. Du bist ja wenigstens ein bißchen von deinem Grammatiktrip runtergekommen. Alles kann man vielleicht mit dem einen Formalismus doch nicht machen. Das mit den Berechnungen hört sich an wie die Lexikonzugriffe.

A: Also mich hast du immer noch nicht überzeugt. Deine externen Berechnungen in allen Ehren, aber was ist mit dem Kameramann und dem Beleuchter? Wie willst du die Kamera und die Lichter denn automatisch setzen?

B: Na, C hat doch vorhin erzählt, wie man sich das Leben in einer 3D-Welt einfach macht. Man nimmt ein paar Punkte, die zu einem Objekt den umschreibenden Quader bilden. Wenn du dann noch weißt wo vorne und hinten ist, kannst du wunderbare Kamera- und Lichtpositionen berechnen. Wenn wir den zugehörigen Algorithmus dann in eine schöne schwarze Kiste verpacken und Kameramann nennen...

A: Und der Beleuchter? Wie soll der Rechner denn sehen, ob die Beleuchtung stimmt?

B: Das macht eine Kamera doch auch selbst! Jede einfache Video- oder Fotokamera regelt die Helligkeit so, daß das Bild richtig belichtet wird, und genau so werden wir das auch machen...

Die Diskussion taucht nach und nach ab in technischen Details wie Vektorrechnung und Grauwerte. Nach einer Weile sind auch diese Fragen geklärt.

Kapitel 5

Verfahren zur Berechnung geometrischer Spezifikationen

Um eine Animation geometrisch vollständig zu spezifizieren, muß das Skript konkrete geometrische Angaben enthalten, und zwar in Form von Matrizen, Vektoren, Punkten und absoluten Werten. Die Kameraposition beispielsweise wird durch eine Matrix angegeben, eine Kamerafahrt durch ihren Bewegungsvektor, ein Schwenk durch seine Rotationsachse und den Rotationwinkel. Die konkreten Werte müssen aus der gegebenen Modellwelt berechnet werden, wozu in der vorliegenden Arbeit einige Verfahren entwickelt wurden. Diese Verfahren sowie die ihnen zugrunde liegenden Daten möchte ich im folgenden kurz beschreiben.

5.1 Verwendete Modelldaten

Die zur Animationsgenerierung benötigten Modelldaten können beliebige 3D-Modelle der Domänenobjekte sein. Einzige Bedingung ist, daß das zur Ausgabe benutzte Graphiksystem aus ihnen Bilder erzeugen kann und daß für den Skriptgenerierungsprozeß bestimmte geometrische Vereinfachungen und strukturelle Beschreibungen der Modelle zur Verfügung stehen. Diese Vereinfachungen können entweder aus den Modellen selbst abgeleitet werden oder müssen getrennt modelliert werden.

5.1.1 Geometrische Vereinfachungen

Grundlegende Information für die in diesem Kapitel beschriebenen geometrischen Berechnungen sind die umschreibenden Quader der Objekte, wie in Bild 2.5 auf Seite 23 gezeigt. Diese Quader werden bei Objektbewegungen mitbewegt und liefern eine einfache Näherung für den Raum, den ein Objekt einnimmt. Eine solche Näherungslösung ist immer ein Kompromiß zwischen der Genauigkeit des Ergebnisses und seiner Berechnungsgeschwindigkeit. In der Praxis hat sich jedoch gezeigt, daß die aufgrund der umschreibenden Quader berechneten Kamerapositionen und Verdeckungsinformationen zur Generierung ansprechender 3D-Animationen bei weitem ausreichen. Die Repräsentation der Objekte durch umschreibende Quader sowie bevorzugte Betrachtungsrichtungen lehnt sich an die in [Schneider, 95] vorgestellten P-Quader an.

5.1.2 Bevorzugte Betrachtungsrichtungen

Um eine intelligente Auswahl der Kameraposition zu ermöglichen, muß für die Modelle die grundlegende Information zur Verfügung stehen, welche ihrer Seiten die Vorderseite ist, welche die Oberseite, und wo links und rechts ist. Diese Information dient später dazu, eine Kameraposition zu berechnen, die das Modell beispielsweise von vorne oder von vorne leicht oben und etwas links zeigt. Die Information wird in Form dreier Vektoren modelliert, die sich mit dem Objekt und seinem umschreibenden Quader bewegen und so jeweils angeben, in welche Richtung die Vorder- (Ober-, rechte) Seite des Objektes gerade zeigt. Die Hinter-, Unter- und linke Seite ergeben

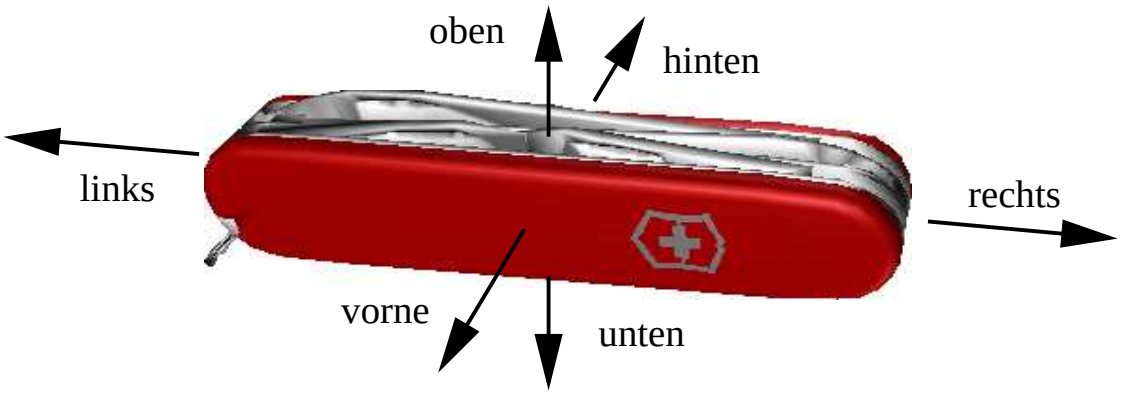


Abbildung 5.1: Bevorzugte Betrachtungsrichtungen

sich als Umkehrvektoren daraus automatisch. Die bevorzugten Betrachtungsrichtungen *vorne* und *oben* fallen oft, jedoch nicht immer mit den entsprechenden intrinsischen Objektseiten (Für einen Überblick hierzu siehe [Maass, 96]) zusammen, während rechts und links im Vergleich dazu oft vertauscht sind. Die Festlegung der bevorzugten Betrachtungsrichtungen bestimmt lediglich die Seiten, von denen das Objekt später in der Animation gezeigt wird und erhebt ansonsten keinen Anspruch auf eine tiefere Semantik.

5.1.3 Materialbeschreibung

Um einen gezielten Einsatz von Farb- und Transparenzeffekten planen zu können, wird eine Beschreibung der Materialeigenschaften der verwendeten Objekte benötigt. Ist ein Objekt beispielsweise schon transparent modelliert, so ergibt es keinen Sinn, es als Effekt durchsichtig zu machen. Aus der Objektfarbe können außerdem zu ihr kontrastierende Farben berechnet werden, um ein Objekt beispielsweise auffällig blinken zu lassen. Die in meinem Ansatz verwendete Materialbeschreibung lehnt sich an die Materialbeschreibungen der Graphikbibliothek OpenGL an und beschreibt als grundlegende Information die Farbe, Transparenz und Reflexionseigenschaften der Objektoberfläche, was zum gezielten Einsatz von Farb- und Transparenzeffekten völlig ausreicht.

5.1.4 Teil-von-Hierarchie

Eine wichtige strukturelle Beschreibung der Modellwelt ist die Zusammenfassung der Objekte zu Objektgruppen. Diese Gruppen können gemeinsam bewegt oder in ihren Eigenschaften (Farbe, Transparenz, Abstraktionsgrad) verändert werden. Außerdem sind für die Objektgruppen die gleichen geometrischen Vereinfachungen und eine Beschreibung der bevorzugten Betrachtungsrichtung vorhanden. Während der umschreibende Quader einer Objektgruppe leicht aus den umschreibenden Quadern der Einzelobjekte zu berechnen ist, müssen die Betrachtungsrichtungen wieder explizit angegeben werden, da die Vorderseite einer Baugruppe nicht unbedingt mit der Vorderseite aller ihrer Bestandteile zusammenfällt. Die Gruppierung der Objekte erfolgt in mehreren Ebenen, so daß sich ein Baum aus Objektgruppen ergibt, die sogenannte *Teil-von-Hierarchie*. Wurzel des Baumes ist die Objektgruppe *Welt* (Siehe auch Abbildung 8.2 auf Seite 118).

5.1.5 Zusammenbauinformation

Eine weitere wichtige Information, die zur Berechnung der Explosionsdarstellungen benötigt wird, ist die Zusammenbauinformation der Modellteile. Durch die Beschreibung, welches Objekt mit welchem anderen mechanisch verbunden ist und in welche Richtung es von ihm getrennt werden kann, ist es möglich, die für eine Explosionsdarstellung benötigten Objektbewegungen zu berechnen. Die Zusammenbauinformation ist grundsätzlich verschieden von der Teil-von-Hierarchie, da funktional zusammengehörige Objekte (Objektgruppen) durchaus mechanisch getrennt sein können und umgekehrt die Teile verschiedener Objektgruppen mechanisch verbunden sein können. Die Bedeutung und Form solcher Zusammenbauinformationen wird in [Li, 96] eingehend diskutiert.

5.1.6 Bewegungsbeschreibungen

Zur Modellierung der Objektbewegungen habe ich mich in der vorliegenden Arbeit auf die Bewegungsbeschreibung einzelner Objekte beschränkt. Mechanische Zusammenhänge sind bezüglich der Bewegungen nicht modelliert, da dies eine Berücksichtigung der Kinematik des Modells im Graphiksystem erfordern würde, was die Trennung zwischen Animationsplanung und Animationsrealisierung aufhebt. Die Bewegungsbeschreibungen der Objekte müssen in einer Form vorliegen, aus der die Objektposition und -orientierung zu jedem Zeitpunkt der Bewegung ableitbar sind. Für Translationen, Rotationen und Splines ist diese Berechnung sehr einfach, für implizite Objektbewegungen durch Kinematik und inverse Kinematik des Modells müßte eine Simulation der Bewegung schon während der Generierung erfolgen.

5.1.7 Berechnung der Abstraktionen

Ein System, das Animationsskripte unter kommunikativen Aspekten generiert, produziert schon bei der Skriptplanung alle notwendigen Informationen, um den Abstraktionsgrad der verwendeten Modelle gezielt zu steuern. Dies wird in [Butz &

Krüger, 96, Butz & Krüger, 97] ausführlich erläutert. Um die berechneten Abstraktionsgrade auch tatsächlich zur Bilderzeugung zu verwenden, ist es eigentlich unerläßlich, die zugehörigen Modelle dynamisch zu berechnen. In der Praxis kommt man jedoch meist mit einigen vorher berechneten Abstraktionsgraden für jedes Modell aus, die dann beliebig kombiniert werden können. Die eigentlichen Verfahren zur Ableitung abstrahierter Polygonmodelle werden in [Krüger, 98] ausführlich behandelt und ich möchte sie hier nur kurz anreißen. Grundlage ist ein Algorithmus aus dem Bereich des CAD, vorgestellt in [Rossignac & Borrel, 93], der den Raum entlang der drei Achsen in Untereinheiten, sogenannte *Cluster* unterteilt. Innerhalb jedes Clusters werden nun alle Eckpunkte des Polygonmodells zu einem einzigen zusammengefaßt, was die Anzahl der Eckpunkte und damit automatisch auch die Anzahl der Polygone reduziert. Durch die Größe der Cluster kann die Dichte der

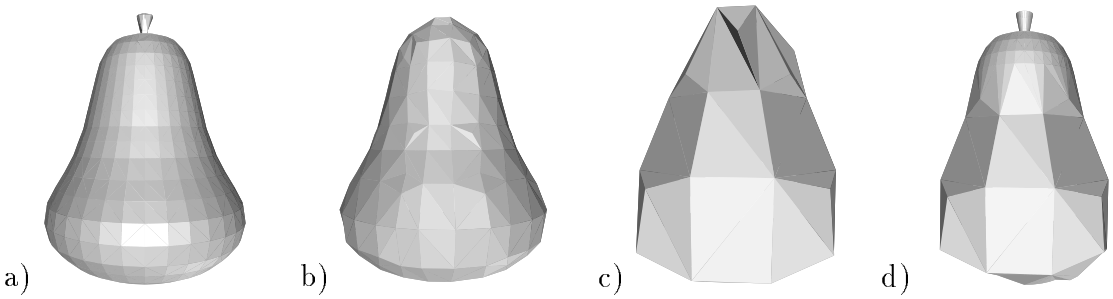


Abbildung 5.2: Verschiedene Abstraktionsgrade eines Polygonmodells

Punkte im Modell und somit der Abstraktionsgrad reguliert werden. Durch eine unterschiedliche Gewichtung der einzelnen Knoten innerhalb jedes Clusters kann eine Verbesserung des Abstraktionsergebnisses erreicht werden und durch eine unterschiedliche Clustergröße in verschiedenen Teilen des Raumes oder die Fixierung bestimmter Punkte (Abbildung 5.2 d) können Teile eines Modells einen höheren Detaillierungsgrad behalten als andere. Eine genauere Beschreibung der Erweiterungen des Verfahrens und der Kriterien zur Bestimmung der Clustergröße findet sich in [Butz & Krüger, 97] sowie [Krüger, 98].

5.2 Perspektivenwahl

Die Grundvoraussetzung, um ein Objekt im Bild zu zeigen, ist eine passende Kameraposition. Während diese Position im Skript durch eine Matrix spezifiziert wird, möchte man sie im Generierungsprozeß auf einer höheren Abstraktionsebene behandeln. Eine Kameraposition läßt sich beispielsweise auch dadurch charakterisieren, welches Objekt sie im Bildmittelpunkt zeigt, wie stark dieses Objekt das Bild ausfüllt, und von welcher Seite es gezeigt wird. Diese Art der Spezifikation wurde in [Schneider, 95] vorgeschlagen und eignet sich sehr gut für den vorliegenden Fall.

5.2.1 Blickrichtung

Benutzen wir die in Abbildung 5.1 gezeigten bevorzugten Betrachtungsrichtungen und außerdem beliebige gewichtete Kombinationen dieser Richtungen (als Listen der

Schlüsselwörter *oben*, *vorne* u.s.w.), so kann aus dieser symbolischen Beschreibung durch Linearkombination der zugehörigen Vektoren direkt ein Blickvektor berechnet werden. Ein Beispiel ist in Abbildung 5.3 gezeigt. In der gleichen Art wird die

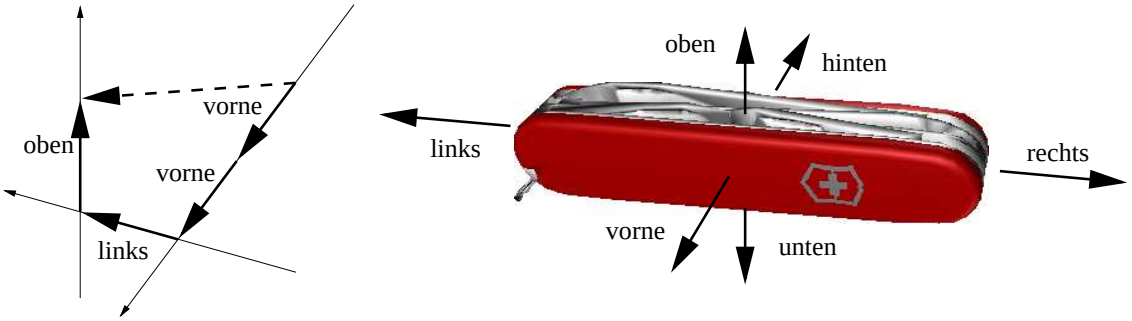


Abbildung 5.3: Berechnung der Blickrichtung *(vorne, vorne, links, oben)*

Orientierung der Kamera um die Längsachse festgelegt. Die *oben*-Richtung der Kamera liegt normalerweise in der Ebene aus *oben*-Richtung der Welt und optischer Achse und ist zur optischen Achse senkrecht. Prinzipiell können aber auch hier beliebige Richtungen angegeben werden, was zu einem mehr oder weniger schrägen Kamerabild führt.

5.2.2 Bildwinkel und Abstand

Nachdem unter gestalterischen Gesichtspunkten der Bildwinkel festgelegt wurde, wird die Kamera nun auf dem berechneten Blickvektor vor oder zurück verschoben, bis das Zielobjekt den vorgesehenen Raum im erzeugten Bild einnimmt. Diese Bewegung läßt sich in einem Schritt mit Hilfe des Strahlensatzes berechnen (Abbildung 5.4). Nehmen wir an, der Bildwinkel w sei fest vorgegeben, und das Objekt solle den

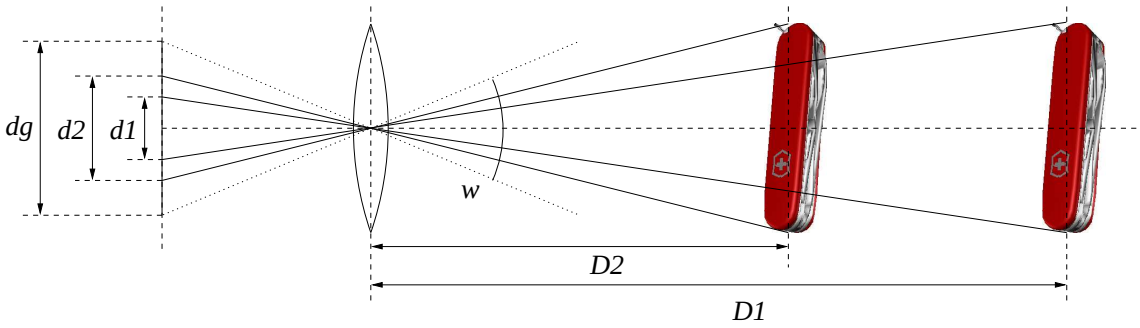


Abbildung 5.4: Berechnung des Betrachtungsabstandes

Anteil x der Bildbreite d_g (beispielsweise $x = 0.7$ für 70 Prozent) einnehmen. Bei der aktuellen Kameraentfernung D_1 stellen wir fest, daß die Projektion den Raum d_1 einnimmt, so ergibt sich die gewünschte neue Entfernungseinstellung D_2 wie folgt:

$$d_2 = x * d_g, \quad \frac{D_2}{D_1} = \frac{d_1}{d_2}, \quad \text{also} \quad D_2 = D_1 * \frac{d_2}{d_1} = x * D_1 * \frac{d_g}{d_1}$$

Somit erhalten wir mit D_2, die gewünschte Kameraentfernung und können mit Hilfe der Objektposition und des Blickvektors die Kameraposition sofort berechnen. Diese

Berechnungen müssen nicht auf den vollständigen Modelldaten ausgeführt werden, sondern es genügt, die in Abschnitt 5.1.1 genannten umschreibenden Quader zu benutzen.

5.2.3 Verdeckungen

Um bei der Berechnung der Kameraposition festzustellen, welche Objekte der virtuellen Welt das Zielobjekt möglicherweise verdecken, kann man folgende effiziente Näherungslösung verwenden (Siehe Abbildung 5.5): Zunächst scheiden alle Objekte aus, deren umschreibende Quader komplett hinter dem des betrachteten Objekts liegen, also in Bild 5.5 die Objekte 1 und 2. Von den Objekten, die komplett davor liegen, verursachen diejenigen, deren umschreibender Quader die von der Kamera aufgespannten Blickpyramide schneidet, wahrscheinlich eine Verdeckung (Objekt 5), während die außerhalb der Blickpyramide liegenden Objekte (Objekt 6) ebenfalls ausscheiden.

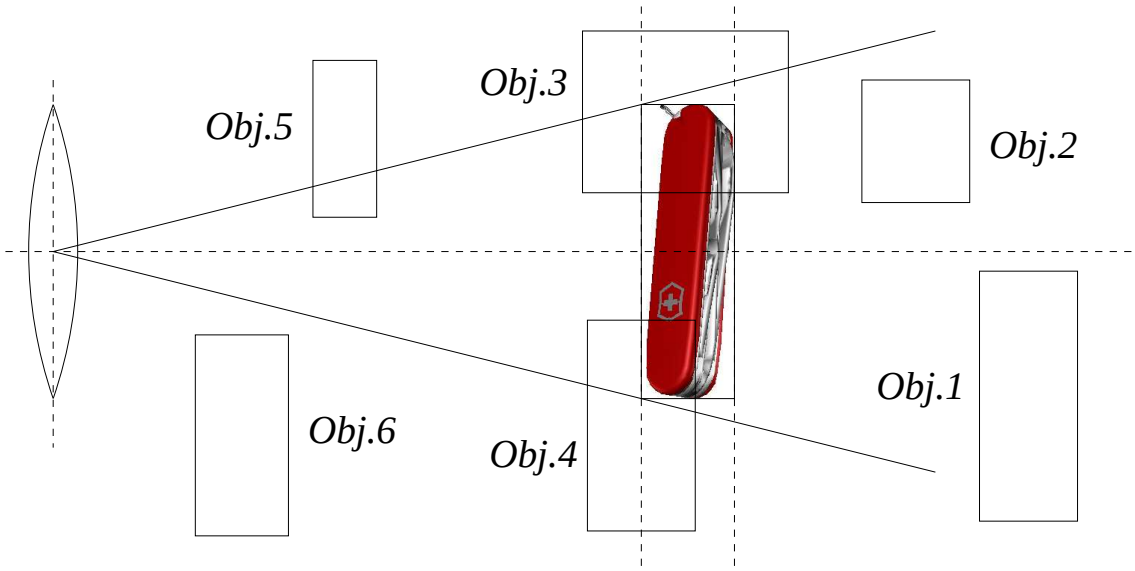


Abbildung 5.5: Berechnung verdeckter Objekte

Nach dieser Selektion bleiben also nur die Objekte übrig, deren umschreibender Quader sich mit dem des betrachteten Objektes in Richtung des Blickvektors überschneidet. Findet diese Überschneidung in allen drei Raumdimensionen statt (was in Bild 5.5 leider nicht darstellbar ist), so liegt mit ziemlicher Sicherheit eine Verdeckung vor. Findet sie nur in zwei Dimensionen statt, so verdecken sich die Objekte meist nicht. Um eine völlig sichere Entscheidung unter den verbliebenen Objekten zu treffen, müßte man die kompletten Modelldaten im Berechnungsprozeß berücksichtigen. Es hat sich jedoch gezeigt, daß die genannte Näherungslösung in der Praxis völlig ausreicht und schlimmstenfalls ein Objekt als störend eingestuft wird, das in Wirklichkeit gar keine Verdeckung hervorruft.

5.3 Beleuchtung

Will man eine 3D-Animation vollständig automatisch generieren, so muß man auch automatisch eine angemessene Beleuchtung der Szenerie bestimmen. Zur Positionierung der Lichtquellen bietet es sich dabei an, auf etablierte Lichtanordnungen aus Malerei, Photographie und Computergraphik zurückzugreifen.

5.3.1 Grundanordnungen

Ein gängiges Verfahren in der Computergraphik ist es, eine Lichtquelle genau an der Position der Kamera oder direkt dahinter zu positionieren. Diese Anordnung stellt sicher, daß es keine Schattenzonen im Bild gibt, da die Lichtstrahlen ja den gleichen Weg nehmen wie die Blickstrahlen der Kamera und deshalb jede sichtbare Fläche auch erreichen. In Bild 5.6 ist dies einmal mit einer Punktlichtquelle (1*a*) und einmal mit gerichtetem Licht (1*b*) dargestellt. Unter gestalterischen Aspekten ist diese Ausleuchtung allerdings meist nicht sehr *gut*. Sie erhellt zwar die gesamte Szenerie, betont aber weder deren Räumlichkeit (Siehe Abschnitt 2.1.5), noch hebt sie Strukturen hervor. Mit nur geringfügig mehr Aufwand läßt sich beispielsweise eine in der Photographie verwendete Beleuchtungsanordnung verwirklichen, die eben diese Nachteile umgeht.

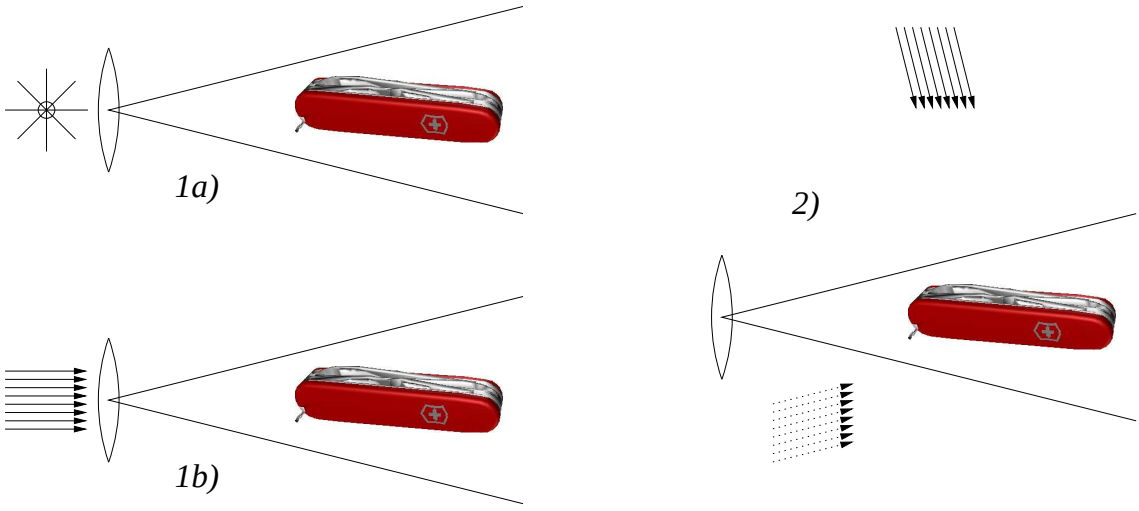


Abbildung 5.6: Grundanordnungen für Lichtquellen

Abbildung 5.6, Anordnung 2 zeigt eine Beleuchtungsanordnung, die sich an dem in der freien Natur vorhandenen Licht orientiert. Dort ist die Sonne die einzige echte Lichtquelle. Sie entspricht im computergraphischen Modell einer entfernten, gerichteten Lichtquelle von neutraler oder warmer Farbabstimmung, die meist von schräg oben kommt. Die entstehenden harten Schatten werden durch das vom umgebenden Himmel oder den Wolken reflektierten Licht aufgehellt, das meistens eine eher bläuliche, kühle Farbabstimmung hat.

Diese Situation ist außerdem die gestalterische Basis vieler photographischer Lichtanordnungen. Eine Grundregel ist es, auch dort nur eine Hauptlichtquelle zu benutzen und die entstehenden harten Schatten durch diffuse Reflektoren oder schwache sekundäre Lichtquellen aufzuhellen. Als Faustregel für die Helligkeit sekundärer

Lichtquellen setzt man fest, daß ihre Stärke insgesamt nicht die Hälfte der Leuchtstärke des Hauptlichtes übersteigen darf (Siehe auch [Dunker, 93]). Es gilt als gestalterischer Fehler, wenn im Bild mehrfache Schatten eines Gegenstandes oder einer Person zu sehen sind.

5.3.2 Helligkeitssteuerung

Nachdem sich so eine Anordnung der Lichtquellen ergibt, muß ihre Helligkeit den bestehenden Verhältnissen angepaßt werden. Auch hier möchte ich wieder einige Analogien aus der Photographie zu Hilfe nehmen. In [Adams, 82] wird das sogenannte Zonensystem beschrieben, eine Methode, die es Photographen ermöglicht, den Belichtungsspielraum des verwendeten Filmmaterials möglichst gut auszunutzen. Es wird dort geschildert, wie man ein Negativ belichtet und entwickelt, damit alle Stellen des Bildes von den hellsten Lichtern bis zu den dunkelsten Schatten noch Zeichnung aufweisen. Die Helligkeit wird dabei durch Blende und Verschlußzeit geregelt, der *Kontrastumfang* durch die Filmentwicklung.

Übertragen auf die Computergraphik bedeutet das, die Beleuchtung der Szenerie so auszulegen, daß die hellsten Stellen des Bildes noch nicht vollständig weiß sind, während die dunkelsten Stellen nicht mehr völlig schwarz sein dürfen. Das klingt zunächst nach einer sehr groben und unangemessenen Vereinfachung, wird jedoch relativiert durch die Tatsache, daß selbst die besten derzeit erhältlichen Kameras mit automatischer Belichtungssteuerung nichts anderes tun. Ein Photokamera regelt dabei die Grundbelichtung durch Blende und Verschlußzeit und gleicht einen zu großen Kontrastumfang durch eine dicht bei der Kamera positionierte schwächere Lichtquelle, den sogenannten Aufhellblitz aus.

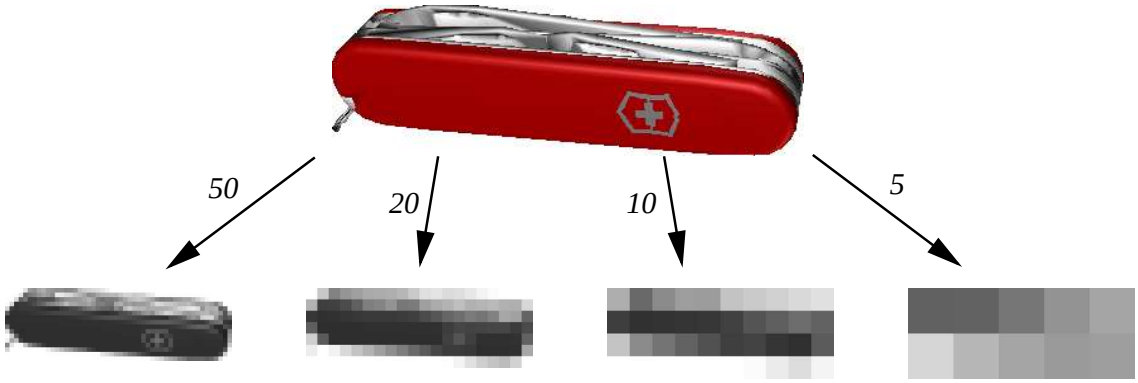


Abbildung 5.7: Belichtungsmessung mit verschiedenen Auflösungen

Bei der in Abbildung 5.6, 2) beschriebenen Lichtanordnung läßt sich genau dieses Verfahren ebenfalls anwenden. Nachdem das Hauptlicht mit einer Standardhelligkeit positioniert wurde, wird ein Bild damit erzeugt, dessen Helligkeit über Quadrate fester Größe gemittelt (Abbildung 5.7) und daraus die hellste und die dunkelste Stelle berechnet. Das Hauptlicht wird nun so geregelt, daß das hellste Quadrat einen hellen Grauwert annimmt, aber nicht weiß wird. Damit ist die Grundhelligkeit bereits recht gut getroffen. Sind im Bild nun sehr dunkle Schatten vorhanden, das heißt, das dunkelste Quadrat schwarz, so wird ein Hilfslicht hinzugenommen um die Schatten aufzuhellen. Dieses Hilfslicht wird so geregelt, daß die dunkelste Stelle im Mittel ein

sehr dunkles Grau annimmt. Das Hilfslicht darf jedoch nicht heller als die Hälfte des Hauptlichtes werden. Der Algorithmus zur Berechnung einer Grundausleuchtung der Modellwelt sieht wie folgt aus:

```
proc Berechne-Grundausleuchtung
    Setze Hauptlicht von schräg oben mit Default-Helligkeit
    Führe Belichtungsmessung durch

    Solange die hellste Stelle im Bild zu hell oder zu dunkel ist
            Korrigiere Hauptlicht entsprechend
            Führe Belichtungsmessung durch

    Setze Hilfslicht von schräg unten mit halber Helligkeit des Hauptlichts
    Führe Belichtungsmessung durch

    Solange die dunkelste Stelle im Bild zu hell ist
            Korrigiere Hilfslicht entsprechend
            Führe Belichtungsmessung durch
```

Die Regelung der Lichtquellen ist meist in drei bis vier Iterationszyklen abgeschlossen und die so berechnete Ausleuchtung beleuchtet die Szenerie in der Regel sehr gut, da sichergestellt ist, daß Objekte in allen Bildteilen sichtbar sind. Die Auflösung der Belichtungsmessung bestimmt dabei einerseits deren Qualität, andererseits aber auch den Berechnungsaufwand. In der Praxis hat sich eine Auflösung von $10 * 10$ Feldern als guter und schnell berechenbarer Kompromiß herausgestellt, was die Auflösung der bisher bei Photokameras gebräuchlichen Matrixmessungen (5 bis 16 Felder) bei weitem überschreitet. Kurz nach der Entwicklung des eben beschriebenen Verfahrens kam eine Kamera mit einer Auflösung von 1005 Feldern bei der Belichtungsmessung auf den Markt ([Nikon, 96]), die jedoch die ermittelten Helligkeitswerte in einem neuronalen Netz verarbeitet, das mit über 30.000 Beispielbildern trainiert wurde.

5.3.3 Scheinwerfer

Um Gegenstände optisch hervorzuheben, kann man sie beispielsweise mit einem Scheinwerfer beleuchten (Abbildung 5.8). Hierzu muß die Hauptbeleuchtung etwas zurückgenommen werden und der Kegel des Scheinwerfers so berechnet werden, daß er genau das Objekt erfaßt. Eine einfache Strategie ist es dabei, den Scheinwerfer in Kameranähe zu positionieren, da meistens bereits sichergestellt ist, daß das Objekt von dort aus nicht verdeckt wird. Zur Berechnung der Lichtposition, der Richtung und des Leuchtwinkels kann man die oben zu Kameraposition und Bildwinkel angestellten Überlegungen direkt übertragen, nur daß in diesem Falle die Position vorgegeben ist und der Winkel angepaßt werden muß.

5.4 Explosion

Zur Berechnung der Explosionszeichnungen habe ich auf das in [Schneider, 95, Schneider & Krüger, 93] für statische Graphiken vorgestellte Verfahren zurückge-

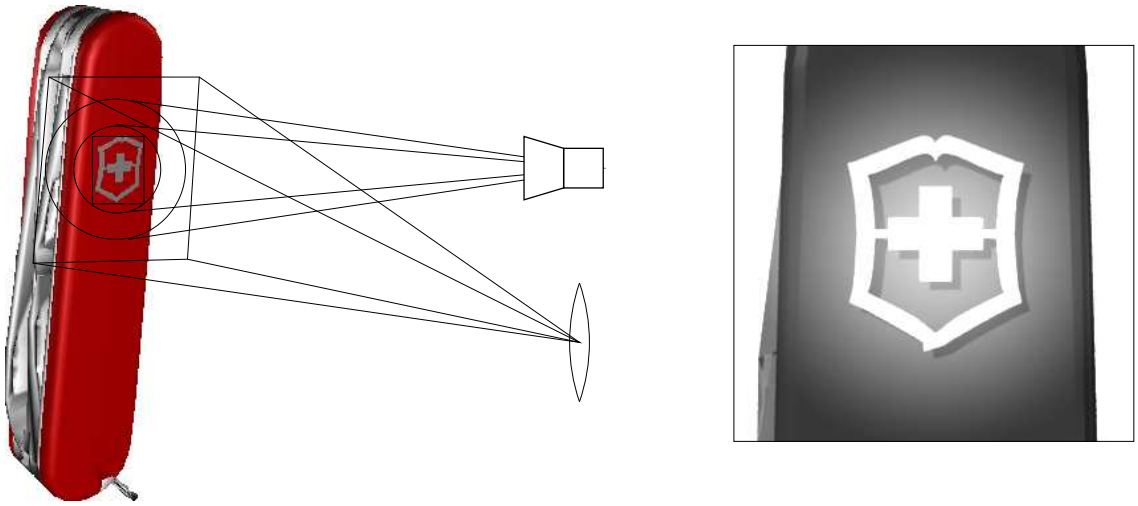


Abbildung 5.8: Positionierung eines Scheinwerfers

griffen. Dieses Verfahren wurde in [Li, 96] weiterentwickelt, ist jedoch in diesem Umfang zur Anwendung in Animationen und unter starker Zeitbeschränkung nicht mehr ohne weiteres einsetzbar. Die Grundideen aus [Schneider, 95] ließen sich hingegen direkt auf die animierte Explosionsdarstellung erweitern und die erzeugten Explosionen sind, obwohl nicht *perfekt*, doch in den allermeisten Fällen ziemlich *gut*.

5.4.1 Explosionsrichtung

Die Richtung, in die ein Objekt bewegt werden muß, um es räumlich von anderen Objekten zu trennen, ergibt sich aus den in Abschnitt 5.1.5 genannten Zusammenbauinformationen. Wenn ein Objekt bewegt wird, so müssen natürlich alle anderen Objekte, die mechanisch mit ihm verbunden sind, mit ihm bewegt werden. Es wird unterschieden zwischen den Vorgängen *Separieren*, *Isolieren* und *Explodieren*. Ein Objekt zu separieren, heißt, es mechanisch von dem Teil, an das es angebaut ist, zu trennen. Ein Teil zu isolieren heißt, das Objekt selbst und alle daran angebauten Teile zu separieren. Explodieren schließlich bezieht sich auf Objektgruppen und bedeutet, alle Teilobjekte der Gruppe zu isolieren.

5.4.2 Explosionsweg

Außer der Richtung, in die sich ein Objekt bewegt, ist auch wichtig, wie weit es sich bewegt. Bei statischen Graphiken kann dieser Weg unter Berücksichtigung der Kameraperspektive berechnet werden, indem das Objekt so weit verschoben wird, bis es in der zweidimensionalen Projektion deutlich getrennt ist. Bei der Generierung von Animationen mit bewegter Kamera ist dieses Verfahren nicht übertragbar, da für die Trennung in der Projektion alle möglichen Kamerapositionen berücksichtigt werden müßten. Um eine kombinatorische Explosion der Berechnung zu verhindern, kommt auch hier eine Näherungslösung zum Einsatz.

Die Objekte werden zunächst so weit in Explosionsrichtung verschoben, daß ihre umschreibenden Quader räumlich getrennt sind. Anschließend werden sie noch um einen weiteren Betrag verschoben, der linear sowohl von ihrer Größe als auch von

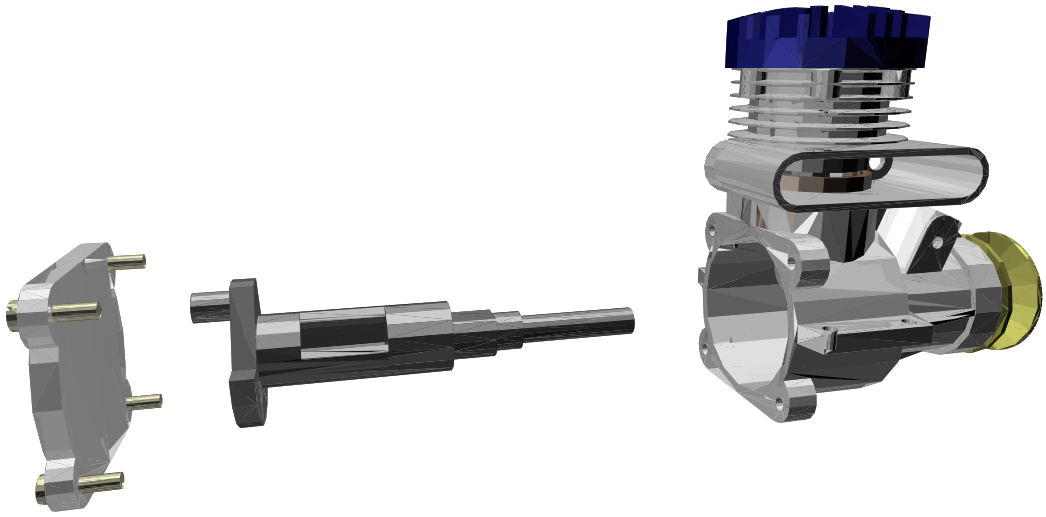


Abbildung 5.9: Beispiel einer Explosionsdarstellung

der Größe des Objektes abhängt, an das sie angebaut sind. Dieses Verfahren führt beispielsweise zu der in Abbildung 5.9 gezeigten Explosionsdarstellung. Das Ziel in diesem Beispiel war es, die Achse des Motors zu isolieren. Hierzu wurde der hintere Deckel samt seiner vier Befestigungsschrauben entfernt und die Achse vollständig aus dem Motorblock herausgezogen.

5.5 Metagraphik

Metagraphische Bildelemente sind Elemente, die nicht zur Graphik selbst gehören, sondern sie modifizieren. Pfeile und Text- oder Bildannotationen sind Beispiele hierfür. Prinzipiell gibt es verschiedene Stellen der Bilderzeugungskette (Siehe Abbildung 2.6 auf Seite 24), an denen solche metagraphischen Elemente eingeführt werden können. Pfeile können beispielsweise in Form räumlicher Objekte in der Modellwelt plaziert werden und gemeinsam mit den anderen Objekten die gesamte *Rendering Pipeline* durchlaufen, sie können aber auch als zweidimensionale Objekte in das fertig berechnete Bild eingeblendet werden.

5.5.1 Einfügen in die Rendering Pipeline

Für die zweite Lösung spricht, daß ein Objekt, das nachher nur ein zweidimensionales Bildelement erzeugt, prinzipiell auch nur als solches modelliert werden muß und daß durch ein Einfügen am Ende der Bilderzeugungskette Rechenaufwand gespart werden kann. Bei näherer Betrachtung gibt es jedoch noch mehr Argumente, die für die erste Lösung sprechen. Um am Ende der Bilderzeugungskette Objekte im Bild zu plazieren, muß eine direkte Beziehung zwischen den 3D-Objekten der Modellwelt und den farbigen Pixeln auf dem Bildschirm herstellbar sein. Diese Beziehung geht jedoch bei den meisten Bilderzeugungssystemen verloren. Alleine anhand des erzeugten Pixelbildes ist es nicht mehr möglich, metagraphische Elemente einzufügen. Außerdem müßte die zeitliche Koordination des Systems bis zum Ende der Bilder-

zeugungskette ausgedehnt werden. Um diese Probleme zu umgehen ist es sinnvoll, die metagraphischen Objekte von Anfang an in die Bildberechnung mit einzubeziehen. In der Modellwelt läßt sich sehr leicht berechnen, ob ein Pfeil beispielsweise mit Objekten kollidiert oder aufgrund seiner Richtung und Nähe dem richtigen Objekt zugeordnet werden kann. Außerdem erhält man so die Möglichkeit, die erzeugten Animationsskripte mit verschiedenen Graphiksystemen umzusetzen, da man über die eigentliche Bilderzeugungskette keinerlei Annahmen zu machen braucht.

Eine weitere Überlegung spricht dafür, den metagraphischen Objekten auch eine räumliche Gestalt zu geben. Würde man einen Pfeil beispielsweise als zweidimensionales Objekt modellieren, so müßte immer sichergestellt sein, daß dessen Flächennormale parallel zur optischen Achse orientiert ist, da sonst schlimmstenfalls nur noch eine Linie zu sehen wäre. Diese Bedingung ist während einer durch Keyframes beschriebenen Kamerafahrt aber sehr schwer einzuhalten. Gibt man dem Pfeil hingegen eine dreidimensionale Gestalt, beispielsweise als drehrundes Objekt (Abbildung 5.10), so behält er seine Pfeilform aus einer wesentlich größeren Anzahl von Betrachtungsrichtungen.

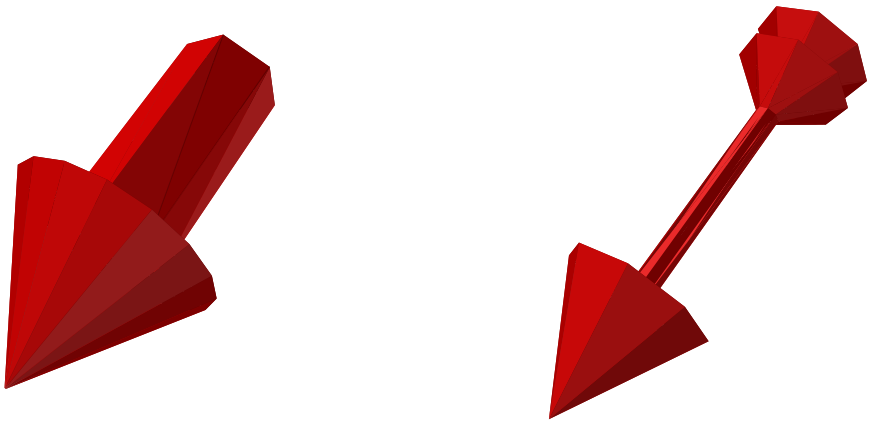


Abbildung 5.10: Dreidimensionale Zeigepfeile

5.5.2 Positionierung eines Zeigepfeils

Zur Positionierung metagraphischer Zeigepfeile gibt es einige sehr einfache Regeln, die in den allermeisten Fällen eine gelungene Plazierung liefern. Zunächst ist es stets sinnvoll, Pfeile senkrecht zur optischen Achse zu orientieren, um zu gewährleisten, daß sie in ihrer vollen Länge und ohne räumliche Verzerrung dargestellt werden. Da sich im Verlauf einer Kamerabewegung diese optische Achse verändert, ist es nicht einfach, die genannte Bedingung für alle Zwischenpositionen zu gewährleisten. Betrachtet man aber nur die Kameraposition zu Ende einer Kamerafahrt, so ist sichergestellt, daß zumindest gegen Ende einer Einstellung der Pfeil in seiner vollen Größe zu sehen ist. Diese einfache Regel schränkt den Suchraum zur Positionierung des Pfeils bereits auf eine Klasse von Ebenen ein, und zwar auf die Ebenen, die senkrecht zu dieser optischen Achse stehen. Nimmt man ferner an, daß der Pfeil stets auf den Mittelpunkt des Objektes zeigen soll, so ist der Suchraum eindeutig auf eine bestimmte Ebene begrenzt. Die nächsten beiden Regeln sind, daß der Pfeil möglichst nahe am Zielobjekt sein sollte, und daß er sich räumlich mit keinem Ob-

jekt der Modellwelt überschneiden darf. Betrachtet man, daß Zeigepfeile sehr oft horizontal oder vertikal im Bild angeordnet sind, seltener auch diagonal, so bleiben innerhalb der genannten Ebene 8 Richtungen übrig, aus denen der Pfeil auf das Zielobjekt zeigen kann. Probiert man diese Richtungen nun in wachsendem Abstand zum Objekt und testet die potentiellen Pfeilpositionen auf Kollisionen mit Objekten der Modellwelt, so braucht man nur die erste kollisionsfreie Position zu nehmen, da sie mit Sicherheit näher am Objekt liegt, als die später zu testenden.

Die Reihenfolge innerhalb eines Abstandes ergibt sich aus einer Beobachtung über die "Leserichtung" von Bildern. Beim Betrachten eines Bildes wandert unser Blick, sofern er nicht anders gelenkt wird, von links oben nach rechts unten, entsprechend dem Lesen eines Textes. Somit liegt es nahe, die Pfeile bevorzugt entlang dieser Richtungen anzuordnen. Treten hierbei Kollisionen auf, so werden die verbleibenden Richtungen getestet. Abbildung 5.11 zeigt, welche Positionen in Frage kommen (schwarz dargestellt) und welche als erste alle gestellten Bedingungen erfüllt (rot dargestellt). Der zugehörige Algorithmus lautet wie folgt:

```
proc Positioniere-Zeigepfeil
    Wähle Pfeillänge l in Abhängigkeit von der Objektgröße
    Für Distanz = (l, 1.5 * l, 2 * l, 3 * l)
        Für Richtung = (oben, links, rechts, unten
                        oben links, unten links, oben rechts, unten rechts)
            Berechne Pfeilposition
            Teste auf Kollisionen
            Falls kollisionsfrei
                liefere diese Position zurück
    Pfeil ist nicht positionierbar
```

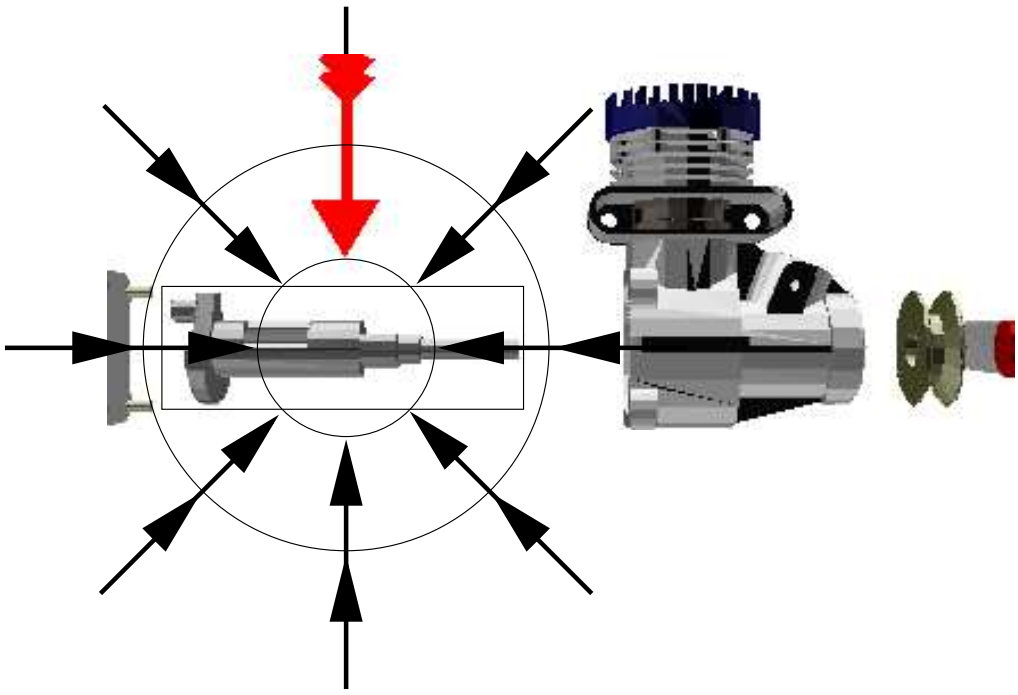


Abbildung 5.11: Positionierung eines Zeigepfeils

Fünfte Szene

Draußen ist es mittlerweile dunkel geworden, aber die drei sind so begeistert bei der Sache, daß das keinem auffällt.

C: Sieh an, da kann man ja doch mehr machen als ich gedacht hätte. Und das alles automatisch...

A: Naja, sieht wirklich aus, als ob du den Kameramann und den Beleuchter automatisiert hättest. Und außerdem hast du noch ein paar neue Berufe erfunden: den Abstrahierer, den Transparierer und den Metagraphiker.

B: (lehnt sich genüßlich zurück) Aber das war ja erst der Anfang. Warte erstmal meinen Drehbuchautor ab. Der beschäftigt einen Rekursierer, einen Inkrementierer und einen Ressourcenadapteur.

Kapitel 6

Inkrementelle Skriptgenerierung

In diesem Kapitel soll der von mir entwickelte Ansatz zur inkrementellen Skriptgenerierung vorgestellt werden. Nach einer Beschreibung aller zur Generierung benötigten geometrischen Daten und Berechnungen im vorangegangenen Kapitel werde ich nun den Generierungsprozeß selbst schildern und seine Besonderheiten aufzeigen.

6.1 Präsentationsziele

Wie in Abschnitt 1.2 beschrieben, können 3D-Animationen im Kontext eines IMP-Systems dazu eingesetzt werden, bestimmte Informationen innerhalb eines koordinierten Multimediadokumentes zu vermitteln. Ein Dokumentplaner generiert dabei die Struktur des Gesamtdokumentes und die Verteilung einzelner Informationsstücke auf die Teile der Präsentation. Ziel jedes Präsentationsteiles ist es also, die ihm zugeteilte Information zu vermitteln. Wir bezeichnen dieses Ziel im folgenden als *Präsentationsziel*.

Das System $\mathcal{PPP}$ ([André et al., 96]), in dessen Rahmen diese Arbeit angefertigt wurde, ist ein solches IMP-System. Der von Elisabeth André entwickelte und in [André, 95] beschriebene Dokumentplaner erzeugt die Struktur eines koordinierten Dokumentes, das Text, Graphik, Animation, Sprachausgabe und Zeigegesten eines animierten Interfaceagenten enthält. Koordiniert heißt, daß einzelne Teile des Dokumentes Querverweise und Anaphern enthalten können und die referenzierte Information durch andere Präsentationsteile vermittelt wird. Eine Sprachausgabe kann also beispielsweise den Verlauf der gezeigten Animation kommentieren und die gezeigten Vorgänge näher erklären.

Der Planer steuert für jedes zu generierende Ausgabemedium eine Generierungskaskade an, die prinzipiell aus einem Gestaltungs- und einem Realisierungsteil besteht. Die Graphikgenerierung beispielsweise besteht aus einer Komponente, die für die konsistente und aussagekräftige Gestaltung der Graphik zuständig ist, und einer Realisierungskomponente, die aus dieser Spezifikation ein Rasterbild erzeugt (Für eine nähere Beschreibung siehe [Rist, 95]). Analog dazu unterteilt sich die Erzeugung der animierten Darstellungen in eine Gestaltungsphase (Skriptgenerierung) und eine Realisierungsphase (Umsetzung und Rendering, siehe Kapitel 7). Eine wesentliche Eingabe der Gestaltungskomponente sind die vom Dokumentplaner spezifizierten *Präsentationsziele*. Die in meinem Ansatz verwendeten Präsentationsziele bestehen

aus einem Schlüsselwort für die Klasse des Präsentationszieles und mehreren Argumenten, die es instanziieren. Beispiele für solche Präsentationsziele sind:

1. (localize-object :object "shaft")
2. (show-action :object "cover" :action "open")
3. (show-group-parts :group "shaft-group")
4. (show-relation :object-1 "shaft" :object-2 "arm")

Im Klartext bedeuten diese Ziele: 1.) Zeige dem Betrachter, wo sich Objekt "shaft" befindet, 2.) Zeige dem Betrachter die Aktion "open" des Objektes "cover", 3.) Zeige die Bestandteile der Baugruppe "shaft-group" und 4.) Zeige dem Benutzer die (räumliche) Beziehung zwischen den Objekten "shaft" und "arm". Damit sind außerdem auch schon die vier grundlegenden Klassen von Präsentationszielen beschrieben, die das in Kapitel 7 beschriebene System CATHI verarbeitet.

6.2 Generierungsparameter

Eine weitere Information, die vom übergeordneten IMP-System zur Verfügung gestellt wird, sind die sogenannten *Generierungsparameter*. Dies sind Angaben, die die Gestaltung der Animation, den Umfang der im Skript spezifizierten Informationen und die Ausführung des Generierungsprozesses selbst beeinflussen. Aufgrund eines Benutzermodells kann der Präsentationsplaner so beispielsweise vorschreiben, daß keine metagraphischen Effekte oder Explosionsdarstellungen benutzt werden sollen, da der angenommene Betrachter eventuell Probleme beim Verständnis dieser Techniken haben könnte oder sie einfach nicht mag. Die gestalterischen Parameter sind im einzelnen:

Parameter	Typ
Dauer der Animation	Zeit in Sek.
Benutze Farbeffekte	Ja/Nein
Benutze Transparenzeffekte	Ja/Nein
Benutze Beleuchtungseffekte	Ja/Nein
Benutze selektive Schärfe	Ja/Nein
Benutze Abstraktionen	Ja/Nein
Benutze Explosionen	Ja/Nein
Benutze Metagraphik	Ja/Nein

Darüberhinaus gibt es eine Reihe von Parametern, die durch das Ausgabemedium vorgegeben sind. Sie beschreiben seine graphischen Fähigkeiten, also ob beispielsweise der Bildwinkel der Kamera steuerbar ist, ob Farbe überhaupt dargestellt werden kann oder auf welche Lichtarten zur Beleuchtung der Szenerie zurückgegriffen werden kann. Diese Parameter sind im einzelnen:

Parameter	Typ
Steuere Bildwinkel der Kamera	Ja/Nein
Steuere Entfernungseinstellung der Kamera	Ja/Nein
Steuere Objektivblende der Kamera	Ja/Nein
Steuere Transparenz der Objekte	Ja/Nein
Steuere Farbe der Objekte	Ja/Nein
Steuere Abstraktionsgrad der Objekte	Ja/Nein
Erzeuge metagraphische Objekte	Ja/Nein
Setze ambientes Licht ein	Ja/Nein
Setze gerichtete Lichtquellen ein	Ja/Nein
Setze Punktlichtquellen ein	Ja/Nein
Setze Scheinwerfer ein	Ja/Nein

Außerdem gibt es Parameter, die den Ablauf des Generierungsprozesses selbst beeinflussen. Sie geben an, ob die Generierung inkrementell erfolgen soll oder nicht, und ob das Auftreten von Zeitbeschränkungen beachtet werden soll:

Parameter	Typ
Inkrementelle Generierung	Ja/Nein
Zeitadaptive Generierung	Ja/Nein

Offensichtlich sind die Generierungsparameter nicht voneinander unabhängig, denn Farbeffekte können beispielsweise nur dann eingesetzt werden, wenn das Ausgabemedium eine Ansteuerung der Objektfarben überhaupt erlaubt. Ist eine Steuerung der Farben möglich, so heißt das aber auch nicht, daß in jedem Fall Farbeffekte eingesetzt werden müssen. Diesen Zusammenhängen wird durch eine passende Formulierung der kontextabhängigen Regeln der Skriptgrammatik Rechnung getragen.

6.3 Struktur der Animationsskripte

Die in meinem Ansatz erzeugten Animationsskripte haben eine hierarchische Struktur (Siehe auch Abschnitt 2.2.5.2). Ein Skript entspricht dabei einem Baum, an dessen Blättern sich elementare Animationsanweisungen, im folgenden als *elementare Skriptsequenzen* bezeichnet, für Objekte, Kamera oder Lichter befinden. Die Verzweigungen oder inneren Knoten des Baumes sind die sogenannten *nichtelementaren Skriptsequenzen*, und sie gruppieren ihre Untersequenzen auf zwei mögliche Arten.

6.3.1 Elementare Skriptsequenzen

Die elementaren Skriptsequenzen geben jeweils eine elementare Bewegung oder Eigenschaftsveränderung der Objekte, Lichter oder Kamera an und gelten für ein bestimmtes Zeitintervall. Beispiele für elementare Sequenzen sind:

1. (translate-object 0.0 0.0 0.0 :object “shaft” :startposition (-7.21 -100 0.55) :endposition (-7.21 0 0.55))

2. (rotate-camera 0.0 2.0 2.0 :axis ((0.0 0.0 0.0) (0.69 0.71 0.08)) :angle 0.341)

3. (adjust-distantlight 0.0 2.0 2.0 :name "sunlight" :from-color (0 0 0) :to-color (0.41 0.41 0.39))

Die jeweils ersten drei Parameter sind dabei Zeitangaben, für die die Anweisung gelten soll, nämlich ihre Startzeit, Endzeit und Dauer. Diese Angaben sind zwar redundant, fallen aber im später beschriebenen Generierungsprozeß als Nebenprodukt ab und können den Umsetzungsprozeß durch die Einsparung ihrer erneuten Berechnung beschleunigen. Eine komplette Auflistung der Typen verwendeter elementarer Skriptsequenzen mit ihren Parametern ist in Anhang A gegeben. Einen Sonderfall bildet die *leere Sequenz*. Diese Sequenz hat in der Animation keine Auswirkung und wird gegebenenfalls aus einem nicht erfüllbaren Unterziel generiert.

6.3.2 Nichtelementare Skriptsequenzen

Die elementaren Skriptsequenzen sind zu Gruppen zeitlich oder logisch zusammengehöriger Sequenzen, den sogenannten *nichtelementaren Skriptsequenzen* zusammengefaßt. Es gibt nun zwei grundlegende Arten nichtelementarer Skriptsequenzen. *Parallele* Sequenzen gruppieren Untersequenzen, die zeitlich parallel oder überlappend ablaufen, und *sequenzielle* Gruppierungen fassen Untersequenzen zusammen, die strikt nacheinander ablaufen. Die Unterscheidung dieser beiden Sequenzarten ist zwar für die Umsetzung in Bilder nicht zwingend nötig, da sich die zeitliche Anordnung ohnehin aus den Zeitangaben der elementaren Sequenzen ergibt, aber sie erleichtert das Verständnis der generierten Skripte und ihre Darstellung. Außerdem wird die Information, ob mehrere Sequenzen parallel (das heißt auch möglicherweise überlappend) verlaufen, beim später beschriebenen Generierungsprozeß quasi umsonst mitgeneriert.

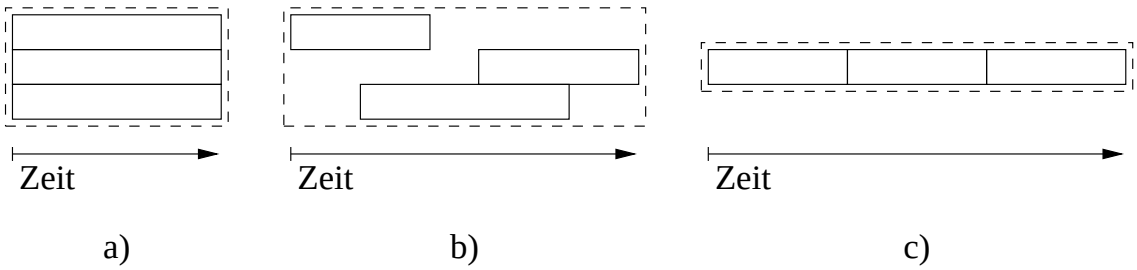


Abbildung 6.1: Anordnungsmöglichkeiten für Untersequenzen

Um dies graphisch zu verdeutlichen, sind in Abbildung 6.1 die verschiedenen zeitlichen Anordnungen dargestellt. Hierbei sind die Anordnungen a) und b) parallele Gruppierungen, wobei die Zeit auf der horizontalen Achse verläuft und Diagramm a) die Anordnung ist, die man mit der ursprünglichen Bedeutung des Wortes parallel verbinden würde. In Diagramm c) sind die Untersequenzen sequenziell angeordnet und deshalb strikt nacheinander gültig.

6.4 Das Kontextkonzept

Grundbestandteil meines inkrementellen Ansatzes ist eine Beschreibung der gegenwärtigen Situation im Generierungsprozeß, der sogenannte *Generierungskontext*. Dieser Kontext ist eine Art schwarzes Brett, das eine Beschreibung des Systemzustandes unter bestimmten Gesichtspunkten enthält. Manche der in ihm enthaltenen Informationen verändern sich während des Generierungsprozesses, während andere unverändert bleiben.

6.4.1 Unveränderliche Information

Die im Kontext beschriebene unveränderliche Information besteht im wesentlichen aus den Generierungsparametern, die in Abschnitt 6.2 genannt wurden. Diese nehmen Einfluß auf die Regeln der in Abschnitt 6.5 beschriebenen Skriptgrammatik und bestimmen somit die Gestaltung der Animation. Der Kontext enthält außerdem Angaben über die Rechenleistung der Maschine, auf der der Generierungsprozeß ausgeführt wird und über die graphischen Möglichkeiten der Maschine, auf der die Animation abgespielt wird. Diese Informationen ermöglichen den korrekten Umgang mit Ressourcenbeschränkungen (Abschnitt 6.7).

6.4.2 Veränderliche Information

Im Kontext beschrieben sind außerdem Informationen über den Zustand der virtuellen Welt, die jeweils auf den neusten Stand gebracht werden, sobald sich diese Welt im Generierungsprozeß verändert. Solche Informationen sind

1. der Zustand der virtuellen Kamera (Position, Einstellungen),
2. der Zustand der Objekte (Position, Eigenschaften),
3. Verdeckungen von der gegenwärtigen Kameraposition aus,
4. der Zustand der Lichter (Position, Helligkeit) und
5. die gegenwärtigen Zeiten in Skript, Generierung und Präsentation.

Wird ein Skriptteil generiert, der die virtuelle Kamera verschiebt, so wird ihre Position auch im Kontext verändert. Die vorhergehende Position wird dabei bis zur nächsten Änderung ebenfalls noch im Kontext gehalten und ermöglicht dem Planungsprozeß eine begrenzte Sicht auf die zurückliegende Animation. Das gleiche geschieht für Objekte und Lichter. Bei der Berechnung neuer Kamerapositionen (Abschnitt 5.2) wird die dabei ermittelte Verdeckung von Objekten aktualisiert.

Schließlich wird immer dann, wenn ein fertig generierter Skriptteil zur Präsentation weitergereicht wird (Abschnitt 6.6.2), die Information im Kontext gespeichert, wann dieser Teil der Präsentation gestartet wurde und bis wann er läuft. Die Summe der im Kontext gespeicherten Informationen wird im Generierungsprozeß dazu genutzt, gestalterische Entscheidungen zu treffen und Werte für die Keyframes des Skriptes zu spezifizieren.

6.5 Die Skriptgrammatik

Das eigentliche gestalterische Wissen zur Generierung der Animationen ist in einer kontextsensitiven Grammatik codiert. Diese Grammatik spezifiziert den syntaktischen Aufbau der Animationsskripte und die Funktion ihrer Produktionsregeln wird durch den eben beschriebenen Kontext gesteuert. Die Ableitungsbäume der Grammatik entsprechen dabei strukturell genau den erzeugten hierarchischen Skripten.

6.5.1 Animationsziele und Unterziele

Ausgangspunkt der Skriptgenerierung sind die in Abschnitt 6.1 erläuterten Präsentationsziele der Animation. Der grundlegende Gedanke meines Ansatzes ist es, diese Ziele mit Hilfe einer Grammatik in immer kleinere Teilziele zu zerlegen, die sich am Ende direkt in elementare Animationsanweisungen übersetzen lassen. Da auf der niedrigsten Ebene (Kamera- und Objektbewegungen) jedoch der Name *Präsentationsziel* mit Sicherheit nicht mehr gerechtfertigt ist, möchte ich die in der Grammatik verarbeiteten Ziele mit *Animationsziel* bezeichnen. Im Unterschied zu den Präsentationszielen enthalten die Animationsziele außerdem eine Zeitspezifikation, die auf unterster Ebene zur Spezifikation der elementaren Sequenzen benötigt wird. Ein Beispiel:

Präsentationsziel: `(localize-object :object 'shaft')`
Animationsziel: `(localize-object :object 'shaft' :duration 10)`

Das ursprüngliche Präsentationsziel wird also um den Generierungsparameter Zeit (Siehe Abschnitt 6.2) ergänzt und die Regeln der Grammatik geben an, wie das so entstandene Animationsziel in kleinere und einfachere Ziele zu unterteilen ist.

6.5.2 Nichtterminale Dekompositionsregeln

Die *nichtterminalen* Dekompositionsregeln der Skriptgrammatik spezifizieren einerseits, in welche Unterziele ein Animationsziel unterteilt wird, und andererseits, wie diese zeitlich angeordnet sind. (Um ein Gefühl für die Funktionsweise dieser Regeln zu bekommen, lohnt es sich, während des Studiums dieses und der folgenden Abschnitte immer auch parallel in Anhang B die Regeln des implementierten Systems CATHI mitzulesen.) Die Deklaration einer nichtterminalen Dekompositionsregel hat die folgende Form:

`(defrule` <**Name**> (<**Argumentliste**>) <**Regelkörper**>)

Die Regel wird eindeutig durch ihren Namen bezeichnet, und jedem Typ von Animationsziel ist genau eine gleichnamige Dekompositionsregel zugeordnet. Die Argumentliste der Dekompositionsregel entspricht der Argumentliste des Animationsziels. Bei der Deklaration können auch Defaultwerte für die Argumente festgelegt werden, so daß es nicht immer nötig ist, alle deklarierten Argumente auch anzugeben. Zu den im Regelkörper spezifizierten Unterzielen muß es ebenfalls wieder (nichtterminale oder terminale) Regeln geben.

6.5.2.1 Zeitspezifikationen

Bei der Deklaration jeder Dekompositionsregel sind implizit die drei Argumente *start-time*, *end-time* und *duration* mitdeklariert, deren Werte innerhalb des Regelkörpers zur Verfügung stehen. Bei der Dekomposition eines Animationsziels müssen diese drei Parameter nur so weit spezifiziert werden, bis die Angabe eindeutig ist. Wird nur die Dauer (*duration*) für ein Unterziel angegeben, so wird die gegenwärtige Zeit im Generierungsprozeß als Startzeit angenommen und die Endzeit konsistent ergänzt. Wird nur die Startzeit spezifiziert, so wird als Endzeit die Endzeit des gerade expandierten Ziels angenommen. Auf diese Art werden aus einer oder zwei Zeitspezifikationen immer die fehlende(n) konsistent aufgefüllt.

6.5.2.2 Die Konstrukte *parallel*, *sequential* und *incremental*

Innerhalb des Regelkörpers wird nun die eigentliche Dekomposition des Zieles in Unterziele festgelegt. Hierzu gibt es die drei Konstrukte *parallel*, *sequential* und *incremental*. Das Konstrukt *parallel* ordnet die mit seiner Hilfe spezifizierten Unterziele zeitlich parallel oder überlappend an (Siehe auch Abbildung 6.1). Es hat die Form

`(parallel` <**Ziel 1**> ... <**Ziel n**>`)`.

Das Konstrukt *sequential* ordnet die mit seiner Hilfe spezifizierten Unterziele statt dessen zeitlich aufeinanderfolgend an. Es hat die Form

`(sequential` <**Ziel 1**> ... <**Ziel n**>`)`.

Das Konstrukt *incremental* schließlich ordnet seine Unterziele in der gleichen Art an wie *sequential*, jedoch spezifiziert es zusätzlich, daß jeder der generierten Unterbäume ein in sich abgeschlossener Skriptteil ist und, nachdem er generiert ist, sofort zur Ausgabe weitergeleitet werden kann. Das Konstrukt hat die Form

`(incremental` <**Ziel 1**> ... <**Ziel n**>`)`.

Die genannten Konstrukte können außerdem rekursiv beliebig geschachtelt werden und so mehrere Dekompositionen im Körper einer einzigen Dekompositionsregel spezifizieren. Ein einfaches Beispiel einer nichtterminalen Regel ist:

```
(defrule steady-shot ()
  (parallel (keep-viewangle :duration duration)
            (keep-camera :duration duration)))
```

6.5.3 Terminale Regeln

An den Blättern des Ableitungsbaumes werden die sogenannten *terminalen* Regeln angewendet. Sie übersetzen ein Animationsziel direkt in eine elementare Skriptsequenz (Siehe Abschnitt 6.3). Terminale Regeln werden deklariert in der Form

`(defrule` <**Name**> `(`<**Argumentliste**>`)` <**elementare Skriptsequenz**>`)`.

Die spezifizierte elementare Skriptsequenz wird mit Werten aus der Argumentliste aufgefüllt. Ein Beispiel einer solchen Regel ist:

```
(defrule keep-camera (transform)
  `(:keep-camera ,start-time ,end-time ,duration
                 :transform ,transform))
```

Man sieht hier, daß sowohl der explizit spezifizierte Parameter *transform* als auch die implizit deklarierten Zeitspezifikationen in die Spezifikation der elementaren Skriptsequenz eingehen.

6.5.4 Kontextabhängige Regeln

Bis zu diesem Punkt ist die Dekomposition der Animationsziele in ihre Unterziele noch allein durch ihre Struktur bestimmt. Um nun eine flexiblere Generierung mit der Skriptgrammatik zu ermöglichen, können Dekompositionsregeln formuliert werden, die in Abhängigkeit vom Zustand des Kontextes (Abschnitt 6.4) verschiedene Dekompositionen liefern. Dies wird erreicht durch das Zulassen bedingter Dekompositionen der Form

`(if` <**Bedingung**> <**Ziel 1**> <**Ziel 2**>`)`
`(when` <**Bedingung**> <**Ziel**>`)`
`(unless` <**Bedingung**> <**Ziel**>`)`
`(cond (`<**Bedingung 1**> <**Ziel 1**>`)...(`<**Bedingung n**> <**Ziel n**>`))`

Die Bedingungen können entweder Anfragen an den Zustand des Kontextes sein oder direkt das Ergebnis der Berechnungen in externen Prozeduren. Die Konstrukte liefern je nach Auswertung ihrer Bedingungen das aus einem der spezifizierten Unterziele generierte Teilskript oder aber die leere Sequenz.

6.5.5 Einbindung geometrischer Berechnungen

Um den Generierungskontext zu verändern, werden während der Generierung an manchen Stellen die in Kapitel 5 beschriebenen geometrischen Berechnungen durchgeführt. Die entsprechenden Aufrufe können beliebig zwischen den Unterzielen verteilt werden, da sie von den Konstrukten *parallel*, *sequential* und *incremental* nicht zu Teilbäumen sondern zur leeren Sequenz expandiert werden und somit nicht in die generierten Skripte eingehen. Beispiele solcher geometrischer Berechnungen sind:

`(look-at` <**object**>`)`	(siehe Abschnitt 5.2)
`(set-up-basic-lights)`	(siehe Abschnitt 5.3)
`(set-spotlight-on` <**object**>`)`	(siehe Abschnitt 5.3)
`(explode-object` <**object**>`)`	(siehe Abschnitt 5.4)
`(set-arrow-on` <**object**>`)`	(siehe Abschnitt 5.5)

Diese Berechnungen beeinflussen den Generierungskontext indem sie die Einstellungen der virtuellen Kamera verändern, Lichter berechnen oder verändern und im Kontext eintragen oder metagraphische Symbole in die Szenerie einfügen.

6.5.6 Eine einfache Beispielableitung

Um die in den vorangegangenen Abschnitten definierten Konzepte und Konstrukte etwas deutlicher zu machen, möchte ich in Abbildung 6.2 eine Ableitung und danach die dazugehörige Grammatik angeben. Die Abbildung zeigt die Zerlegung des

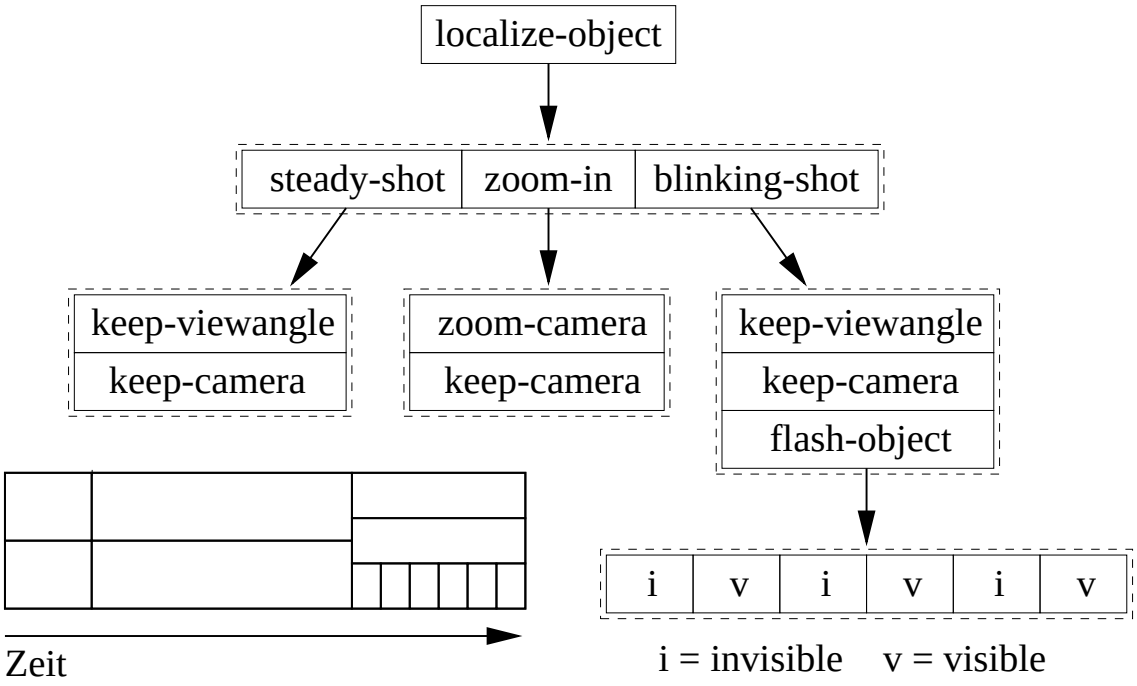


Abbildung 6.2: Beispiel einer einfachen Ableitung

ursprünglichen Animationsziels *localize-object* in drei sequenziell angeordnete Unterziele, die jeweils wieder in zwei oder drei parallel angeordnete Unterziele dekomponiert werden. Hierbei werden noch keine kontextabhängigen Regeln verwendet. Die Definition der zugehörigen Grammatikregeln lautet wie folgt:

```
(defrule localize-object ()
  (incremental (look-at :world)
               (steady-shot :duration (* 0.2 duration))
               (look-at object)
               (zoom-in :duration (* 0.5 duration))
               (blinking-shot :duration (* 0.3 duration))))

(defrule steady-shot ()
  (parallel (keep-viewangle :duration duration)
            (keep-camera :duration duration)))

(defrule zoom-in ()
  (parallel (zoom-camera :duration duration)
            (keep-camera :duration duration)))

(defrule blinking-shot (object)
  (parallel (keep-viewangle :duration duration)
            (keep-camera :duration duration)
            (flash-object :object object :duration duration)))
```

```
(defrule flash-object (object)
  (sequential (invisible :object object)
              (visible :object object)
              (invisible :object object)
              (visible :object object)
              (invisible :object object)
              (visible :object object)))

(defrule keep-camera ((transform *current-transform*))
  `(:keep-camera ,start-time ,end-time ,duration :transform ,transform))

(defrule keep-viewangle ((angle *current-viewangle*))
  `(:keep-viewangle ,start-time ,end-time ,duration :viewangle ,angle))

(defrule zoom-camera ((start-angle *last-viewangle*)
                      (end-angle *current-viewangle*))
  `(:keep-camera ,start-time ,end-time ,duration
                 :start-angle ,start-angle
                 :end-angle ,end-angle))

(defrule invisible (object)
  `(:visibility :level 0 :object ,object))

(defrule visible (object)
  `(:visibility :level 1 :object ,object))
```

Dieses Beispiel ist natürlich gegenüber der wirklichen Skriptgrammatik sehr stark vereinfacht, doch die grundsätzliche Funktionsweise der Grammatik ist erkennbar. Die Regel *localize-object* benutzt beispielsweise das Konstrukt *incremental*, da die generierten Teilbäume im Skript in sich zeitlich abgeschlossene Einheiten bilden. Im Gegensatz dazu beschreibt flash-object keine Skriptteile, die direkt weitergegeben werden können, da parallel zu ihnen die Sequenzen zur Kamerabeschreibung verlaufen. Ein ausführlich kommentiertes Generierungsbeispiel mit Regeln der Skriptgrammatik aus dem System CATHI wird in Kapitel 8 beschrieben.

6.5.7 Gestaltung und Aufbau einer Skriptgrammatik

In den vorangegangenen Abschnitten wurden die einzelnen Bausteine der Skriptgrammatik beschrieben. Um ansprechende Animationen aus einer solchen Grammatik generieren zu können, sind bei ihrer Formulierung einige grundsätzliche Dinge zu beachten.

6.5.7.1 Domänenunabhängigkeit

Zunächst sollte die Grammatik in fast allen Fällen domänenunabhängig sein. Das heißt, in einer gut geschriebenen Grammatik dürfen natürlich keine fest vorgegebenen Objektnamen vorkommen, weder in Kontextanfragen, noch in den spezifizierten Unterzielen einer Regel. Eine Ausnahme bildet das Objekt *Welt*, das in jeder Modellwelt definiert ist (Siehe Abschnitt 5.1.4) und die Objektgruppe aus allen Objekten

der Modellwelt darstellt. Das Objekt *Welt* ist beispielsweise dazu geeignet, eine Kameraposition zu Beginn der Animation festzulegen, die die gesamte Szenerie aus einer passenden Perspektive zeigt.

In einigen Sonderfällen, beispielsweise bei der Generierung von Animationen für ein ganz eng eingegrenztes Gebiet (Siehe Abschnitt 9.2), kann es wohl auch sinnvoll sein, eine domänenabhängige Skriptgrammatik zu formulieren, die dann viel gezielter spezielle Visualisierungstechniken in dieser Domäne einsetzen kann, doch im allgemeinen ist es vorteilhafter, dies nicht zu tun und somit mit einer einzigen Grammatik in verschiedenen Domänen gute Animationen generieren zu können.

Die Grammatik des implementierten Systems CATHI beispielsweise liefert gleichermaßen gute Ergebnisse für drei grundverschiedene Domänen. Das System generiert Animationen für ein Schweizer Messer mit vielen beweglichen Teilen und komplexen Verdeckungsproblemen, für einen Modellmotor mit komplett modellierter Zusammenbauinformation und ineinander geschachtelten Teilobjekten und für das Kugelmodell eines DNA-Moleküls, das an sich schon die Visualisierung eines Gedankenmodells ist und wiederum andere Probleme an die Kameraführung stellt.

6.5.7.2 Starteinstellung und Kontinuität

Am Anfang der Animation sollte eine Starteinstellung stehen, die dem Betrachter einen Überblick über die Modellwelt gestattet. Diese Starteinstellung kann beispielsweise als erste Einstellung der Dekompositionsregeln oberster Ebene (also der Regeln, die die allgemeinsten Animationsziele dekomponieren) explizit festgelegt werden. Somit beginnt jede Animation, die aus einem solchen Animationsziel generiert wird, mit der Starteinstellung. Werden mehrere Ziele nacheinander spezifiziert, so erscheint die Starteinstellung aber unsinnigerweise auch wieder zu Beginn jedes neuen Ziels. Geht man davon aus, daß in der Regel Animationen generiert werden, die mehrere Animationsziele nacheinander visualisieren, so ist es vorteilhafter, zu Beginn jeder Dekompositionsregel oberster Ebene eine Entscheidung zu treffen, die beispielsweise abhängig von der gegenwärtigen und der als nächstes benötigten Kameraposition die Animation mit einem Schnitt, einer Kamerafahrt oder einem Zoom fortsetzt. Als Sonderfall dieser Entscheidung kann dann überprüft werden, ob die aktuelle Skriptzeit gleich Null ist, was bedeutet, daß wir uns am Anfang einer Animation befinden. In diesem Falle wäre es dann sinnvoll, die genannte Starteinstellung zu generieren.

6.5.7.3 Stil und Bildsprache

Durch die Formulierung der Grammatik wird festgelegt, welche gestalterischen Techniken in der Animation eingesetzt werden. Die verwendete Bildsprache sollte in den meisten Fällen nüchtern, deutlich und direkt sein. Ebensogut ist es aber auch möglich, eine Bildsprache in der Grammatik festzulegen, die mit Effekten überladen ist, eine unruhige Kameraführung beschreibt und hektische Schnitte und Kameraschwenks aneinanderreiht. Die Gestaltung der generierten Animationen ist deshalb immer nur so gut wie das ästhetische Empfinden des Grammatikautors. Insbesondere läßt sich durch einen Wechsel der Grammatik die gleiche Information mit anderen bildsprachlichen Mitteln visualisieren und so die Animation gezielt auf bestimmte

Zielgruppen anpassen. Das System CATHI macht von diesem Verfahren ebenfalls Gebrauch, indem es zwei verschiedene Grammatiken verwendet, eine für nüchterne technisch informative Animationen und eine bunte hektische an der Bildsprache des Fernsehsenders MTV orientierte Grammatik.

6.5.7.4 Segmentierung

Oft ist es sinnvoll, die Grammatik in verschiedene Segmente zu gliedern, und zwar nach inhaltlichen, gestalterischen oder organisatorischen Kriterien. Zwei verschiedene Grammatiken können sich durchaus überschneiden und zur Umsetzung einfacherer Animationsziele die gleichen Regeln benutzen. Die Verwendung bildsprachlicher Mittel wird meist auf den oberen Dekompositionsebenen entschieden und so genügt es, nur die dafür zuständigen Teile der Grammatik auszutauschen. Eine andere Möglichkeit, die Grammatik zu segmentieren, ist, die Rollen der einzelnen Mitglieder eines Filmteams auf verschiedene Grammatiksegmente zu verteilen. Eine solche Grammatik besteht dann aus Segmenten für die Beleuchtung, die Kameraführung, für Objektbewegungen und für die übergeordnete Koordination (Regie). Durch Austausch eines Teammitglieds (also Grammatiksegments) kann so gezielt ein Teil der Gesamtgestaltung verändert werden ohne die anderen Teile zu beeinflussen.

6.6 Der Generierungsprozeß

Bei der zurückliegenden Beschreibung der Bestandteile der Skriptgrammatik und ihrer Verwendung wurde bisher keine Aussage über den konkreten Generierungsprozeß gemacht. Während die Regeln der Grammatik eine bestimmte Bildsprache zur Umsetzung der Animationsziele deklarativ festlegen, muß es möglich sein, aus diesen Deklarationen mit Hilfe eines Algorithmus auch voll spezifizierte Animationsskripte zu generieren. Einen solchen Algorithmus möchte ich nun beschreiben.

6.6.1 Rekursion

Eine Grundidee, die sich durch den gesamten vorgestellten Ansatz zieht, ist die Idee der *Rekursion*. Aus rekursiv strukturierten Modellwelten und einer (per se rekursiven) Grammatik sollen rekursiv strukturierte Animationsskripte generiert werden. Was liegt also näher, als einen rekursiven Generierungsalgorithmus vorzuschlagen? Die Rekursion läuft dabei zuerst in die Tiefe (depth-first). Der Algorithmus zur Generierung eines Animationsskriptes teilt sich in verschiedene Prozeduren auf, die sich gegenseitig rekursiv aufrufen.

```
proc Expandiere-animationsziel (ziel)
    Falls ziel kein gültiges Animationsziel ist
        Gib die leere Sequenz zurück;
    Andernfalls
        Finde die zu ziel gleichnamige Dekompositionsregel R;
        Werte den Regelkörper von R aus;
        Setze aktuelle Zeit auf Endzeit der von R produzierten Skriptsequenz;
        liefere die von R produzierte Skriptsequenz zurück;
```

Der Regelkörper einer *terminalen* Regel liefert dabei eine elementare Skriptsequenz zurück. Der Regelkörper einer *nichtterminalen* Regel kann die (eventuell rekursiv geschachtelten) Konstrukte *parallel*, *sequential* oder *incremental* enthalten. Die Begriffe *Startzeit* und *Endzeit* innerhalb der Algorithmen beziehen sich auf die entsprechenden Argumente der Regel, in deren Körper die Konstrukte ausgewertet werden.

proc sequential (*Liste*)
    Für jedes *Unterziel* in *Liste*
        Expandiere-animationsziel (*Unterziel*);
    Gib die Liste der expandierten Unterziele zurück;

proc parallel (*Liste*)
    Für jedes *Unterziel* in *Liste*
        Expandiere-animationsziel (*Unterziel*);
        Setze aktuelle Zeit auf *Startzeit*;
    Gib die Liste der expandierten Unterziele zurück;

6.6.2 Inkrementalität

Bis zu diesem Punkt ist die Ableitung des Skriptbaumes noch vollständig rekursiv und der gesamte Skriptbaum müßte expandiert werden, bevor das Ergebnis feststeht. Da aber bestimmte Teilbäume des generierten Skripts in sich zeitlich abgeschlossene Einheiten bilden, können diese zu gegebenem Zeitpunkt vom Gesamtbaum abgetrennt und zur Ausgabe weitergereicht werden. Hierzu dient das Konstrukt *incremental*, das durch folgende Prozedur ausgewertet wird:

proc incremental (*Liste*)
    Für jedes *Unterziel* in *Liste*
        Expandiere-animationsziel (*Unterziel*);
        Gib das expandierte Unterziel zur Ausgabe weiter;
    Gib die leere Sequenz zurück;

Da dieses Konstrukt die leere Sequenz zurückgibt, werden die von ihm generierten Skriptteile nicht auf einer höheren Ebene (durch ein weiteres *incremental*-Konstrukt) nochmals ausgegeben. Der Effekt dieses Abzweigens generierter Teilbäume ist, daß am Ende des Generierungsprozesses möglicherweise ein leeres Skript generiert wurde, da alle in sich abgeschlossenen Teilbäume schon unterwegs an die Ausgabe weitergegeben wurden. Ein abgeschlossener Skriptteil, der in dieser Art weitergereicht wird, heißt *Inkrement*.

6.6.3 Look-ahead und Look-back

In einem Lehrbuch über das Schreiben eines Drehbuchs fand ich das folgende Zitat über die Weltsicht eines Drehbuchautors während des Schreibens:

> ([Field, 91], S.24-25) Ein Drehbuch zu schreiben ist wie eine Bergbesteigung. Während Sie klettern, können Sie nichts anderes sehen als

> den Felsen vor und unmittelbar über Ihnen. Sie können weder sehen, woher Sie gekommen sind, noch wohin Sie gehen. Das gleiche gilt für den Prozeß des Drehbuchschreibens. Sie können nichts anderes sehen als die Seite, die Sie gerade schreiben, und die Seiten, die Sie schon geschrieben haben. Darüberhinaus können Sie nichts sehen.

Die bei der Auswertung eines Regelkörpers durchgeführten geometrischen Berechnungen und die Expansion elementarer Skriptsequenzen verändern den Kontext im Verlaufe des Generierungsprozesses. Es ist immer nur der aktuelle Zustand der Welt, sowie der Zustand vor der letzten Veränderung jedes Wertes im Generierungsprozeß sichtbar. Sobald Generierungsentscheidungen von einem in der Zukunft oder weiter in der Vergangenheit liegenden Geschehen abhängen, muß dieses Geschehen in der aktuellen Welt simuliert werden.

Um beispielsweise ein bewegtes Objekt bei statischer Kamera von Anfang bis Ende der Bewegung nicht aus der Kamera zu verlieren, muß eine Kameraposition berechnet werden, die sowohl die Start- als auch die Endposition des Objektes im Bild zeigt. Dies wird dadurch erreicht, daß die Bewegung des Objektes vorweggenommen wird, bevor die eigentliche Bewegungssequenz generiert wird. Das Objekt wird in seine Endposition gebracht, diese festgehalten, dann wird das Objekt in die Startposition zurückgebracht und eine Kameraposition berechnet, die es in beiden Positionen gleichzeitig sichtbar machen würde. Dies ist offensichtlich bei einfachen Rotationen und Translationen eine passende Kameraposition für die gesamte Bewegung. Nachdem die Kamera nun auf diese Position gefahren wurde, kann die Generierung des Skriptteiles zur Objektbewegung stattfinden.

Obwohl dieses Verfahren etwas eigenwillig scheint und einiges Nachdenken bei der Formulierung der Grammatik erfordert, ist es doch die einzige Möglichkeit, im Generierungsprozeß eine Entscheidung aufgrund eines Geschehens *nach* der aktuellen Skriptzeit zu ermöglichen. Durch den Druck, generierte Skriptteile sofort zur Präsentation weiterreichen zu müssen, ist eine wirkliche Berücksichtigung zukünftiger Skriptteile nicht möglich, da diese ja dazu erst generiert werden müßten. Zum Ausgleich erhält man jedoch die Fähigkeit, schon nach der Generierung des ersten Inkrementes mit der Ausgabe der Animation beginnen zu können. Dieser Zeitgewinn ist der eigentliche Zweck und zugleich der größte Vorteil eines inkrementellen Generierungsverfahrens.

6.7 Behandlung von Ressourcenbeschränkungen

Bei der automatischen Generierung von Präsentationen gibt es zwei verschiedene Arten verwendeter Ressourcen, die beschränkt sein können. Dies sind einerseits die technischen Ressourcen der Maschine, die die Präsentation erzeugt oder darstellt und andererseits die kognitiven Ressourcen des potentiellen Konsumenten der Präsentation. Die kognitiven Ressourcen des Betrachters einer Animation können durch die Wahl der Generierungsparameter geschont werden, beispielsweise durch den Verzicht auf komplizierte Darstellungsformen wie Explosionsdarstellungen oder durch den gezielten Einsatz abstrahierter Modelle.

Für den Generierungsprozeß selbst ist hingegen eine Betrachtung der technischen Ressourcen wichtig. Der in dieser Arbeit vorgestellte Ansatz zur Skriptgenerierung

sieht neben der inkrementellen Generierung auch die Berücksichtigung beschränkter technischer Ressourcen vor. Die technischen Ressourcen, die den Generierungsprozeß beschränken, sind einerseits die zur Skripterzeugung zur Verfügung stehende Rechenzeit beziehungsweise Rechenleistung und andererseits die graphischen Darstellungsmöglichkeiten des Ausgabemediums.

6.7.1 Zeitbeschränkungen

Die Beschränkung der Generierungszeit wird nicht in einem völlig strengen Sinne gehandhabt. Wie in Abschnitt 6.4 geschildert, enthält der Generierungskontext die Information, wann das letzte Inkrement zur Präsentation weitergegeben wurde und bis wann es voraussichtlich abläuft. Da beim Generierungsprozeß jederzeit die aktuelle Systemzeit feststellbar ist, läßt sich auch jederzeit feststellen, wie viel Generierungszeit noch zur Verfügung steht bis das nächste Inkrement fertig sein sollte. Sobald die zur Skriptgenerierung verfügbare Rechenzeit beziehungsweise Rechenleistung zu knapp wird, kann der durch die Grammatik vorgegebene ursprüngliche Zeitplan der Animation nicht mehr eingehalten werden. Die Generierung einzelner Inkremente dauert dabei länger als die Darstellung des jeweils vorangegangenen, was sich darin äußert, daß die laufende Animation an manchen Stellen stockt und insgesamt länger dauert. Die Grundidee zur Beseitigung dieses Effekts ist, an günstigen

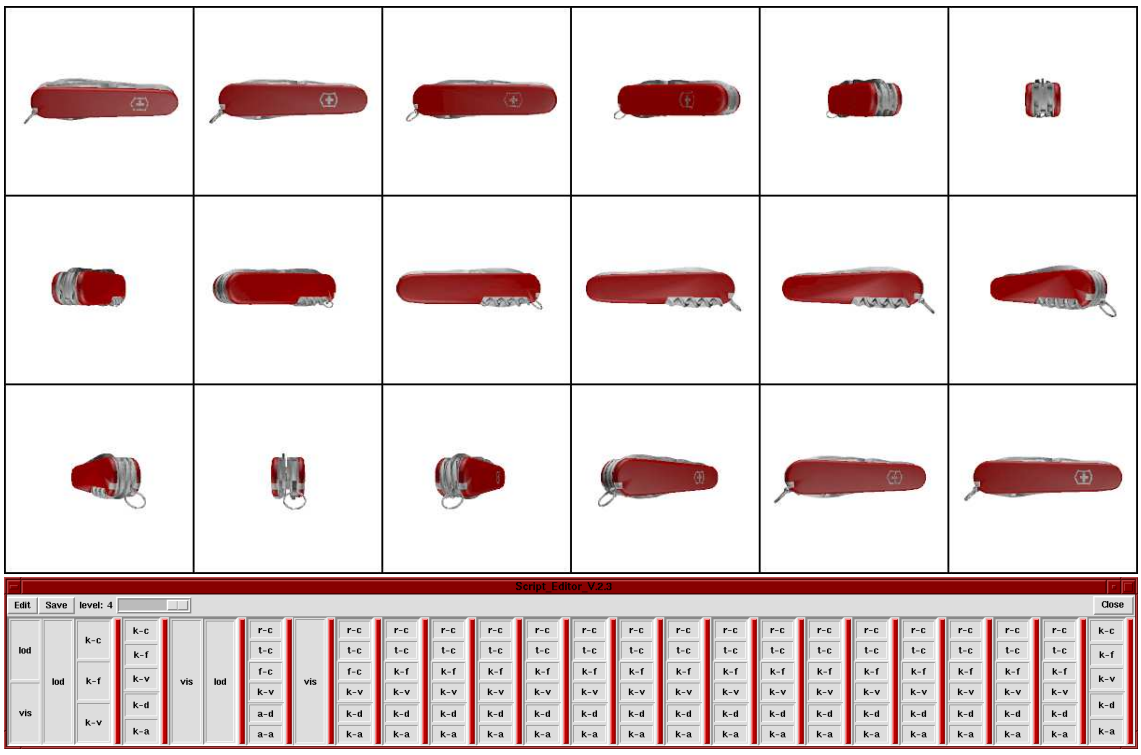


Abbildung 6.3: Rundflug um ein Objekt in 16 Schritten

(und möglichst vielen) Stellen der Grammatik diese Zeitinformation abzufragen und (natürlich unter Berücksichtigung der Systemgeschwindigkeit) abzuschätzen, ob die verbleibende Zeit zur Generierung aufwendiger Skriptteile ausreicht, oder ob statt dessen eine einfachere Dekomposition des gegenwärtigen Animationsziels gewählt werden soll. Am Beispiel eines Kamerarundfluges um ein Objekt möchte ich diese

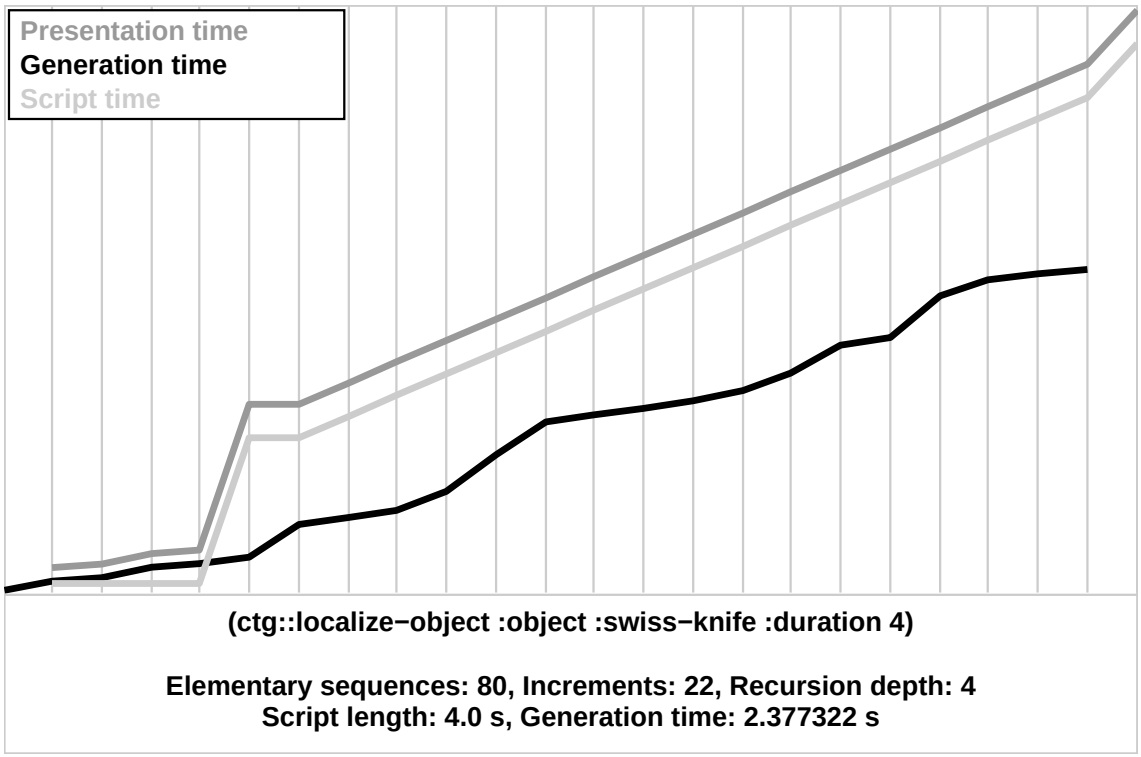


Abbildung 6.4: Timing bei ausreichender Generierungszeit

Strategie verdeutlichen. Abbildung 6.3 zeigt den Rundflug in Form von 18 Keyframes sowie das zugehörige Skript im Skripteditor des Systems CATHI (Kapitel 7).

Der zugehörige Zeitablauf der Generierung und Präsentation ist in Abbildung 6.4 dargestellt. Die vertikalen Striche des Diagrammes geben die Inkrementgrenzen entlang der horizontalen Achse an, während die verstreichende Zeit auf der vertikalen Achse abgetragen ist. Die hellste der drei Diagrammlinien gibt den geplanten Zeitverlauf der Animation wieder, die schwarze Linie den Zeitverlauf der Skriptgenerierung. Da ein Inkrement nicht gezeigt werden kann, bevor es generiert ist, bewegt sich die resultierende Präsentationszeit immer oberhalb dieser beiden Zeiten. Im Idealfall sollten Skript- und Präsentationszeit immer parallel verlaufen, und zwar im Abstand der Zeit, die die Generierung des ersten Inkrementes einnimmt. In Abbildung 6.4 kann diese Bedingung fast eingehalten werden. Das einzige Problem bilden die ersten vier Inkremente, in denen der Ausgangszustand der Modellwelt hergestellt wird, und die alle die Dauer 0.0 Sekunden haben. Verringert man nun die Dauer der Animation und somit die zur Verfügung stehende Generierungszeit, so verläuft die Generierungszeitkurve im Verhältnis zur Skriptzeit steiler (Abbildung 6.5). Der Zeitplan der Animation wird gestört, da die Generierung einzelner Inkremente zu lange dauert.

Die Lösung dieses Problems besteht darin, den Kamerarundflug nicht wie in Abbildung 6.3 gezeigt in 16 Teilstücke zu zerlegen, sondern je nach Zeitknappheit in 8 oder 4 Teilabschnitte. In der resultierenden Animation läuft der Kameraflug nun weniger rund, da die Kameraposition zwischen den Stützpositionen immer nur linear interpoliert wird, jedoch läßt sich das wesentlich einfachere Skript (Abbildung 6.6) schneller generieren und der eigentliche Zeitplan der Animation kann wieder besser eingehalten werden (Abbildung 6.7).

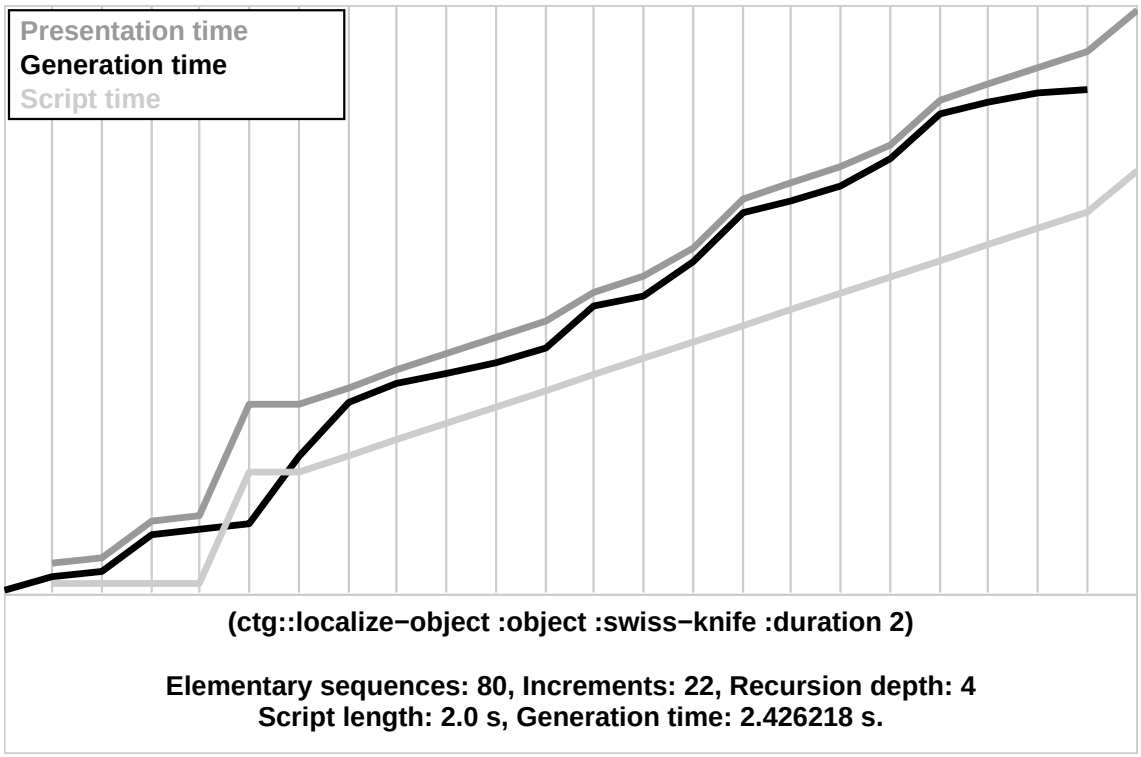


Abbildung 6.5: Timing bei mangelnder Generierungszeit

Dieses Verfahren ist nicht strikt in dem Sinne, daß die vorgefundene Zeitbeschränkung mit Sicherheit eingehalten würde. Die Generierung des aktuellen Inkrements kann immer noch länger dauern als die verbleibende Zeit, damit überhaupt ein sinnvolles Inkrement erzeugt werden kann. Wird eine solche Zeitbeschränkung vom Generierungsprozeß nicht eingehalten, so ist das Resultat wie eingangs beschrieben ein Stocken der Animationsausgabe an einer Inkrementgrenze. Das Stocken wird vom Betrachter als Standbild wahrgenommen und ein sehr kurzes Standbild wird oft garnicht als fehlerhaft registriert. Da das beschriebene Verfahren die größeren Abweichungen aber in der Regel beseitigt, scheint es vertretbar, die Zeitbeschränkung nicht völlig streng zu handhaben und die verbleibenden Timingfehler zu tolerieren.

6.7.2 Begrenzungen des Ausgabemediums

Eine harte Beschränkung des Generierungsprozesses sind hingegen die graphischen Darstellungsmöglichkeiten des Ausgabemediums. Soll die Animation auf einem Graustufen- oder Schwarzweißmonitor ablaufen, so wäre es natürlich völlig unsinnig, im Skript mit Farbeffekten zu arbeiten. Genauso können Lichteffekte und selektive Schärfe nur eingesetzt werden, wenn das System, das die Bilder der Animation berechnet, die entsprechenden Steueranweisungen auch sinnvoll umsetzen und darstellen kann. Die Berücksichtigung dieser Beschränkungen findet durch normale kontextabhängige Dekompositionsregeln statt. Der Kontext enthält Information über die Verfügbarkeit der verschiedenen Steuerparameter im Ausgabesystem und durch Anfragen an den Kontext können die entsprechenden Regeln eine Auswahl bildsprachlicher Mittel treffen. Die Berücksichtigung der Ausgabebeschränkungen ist (bei richtiger Formulierung der Grammatik) strikt in dem Sinne, daß die nicht

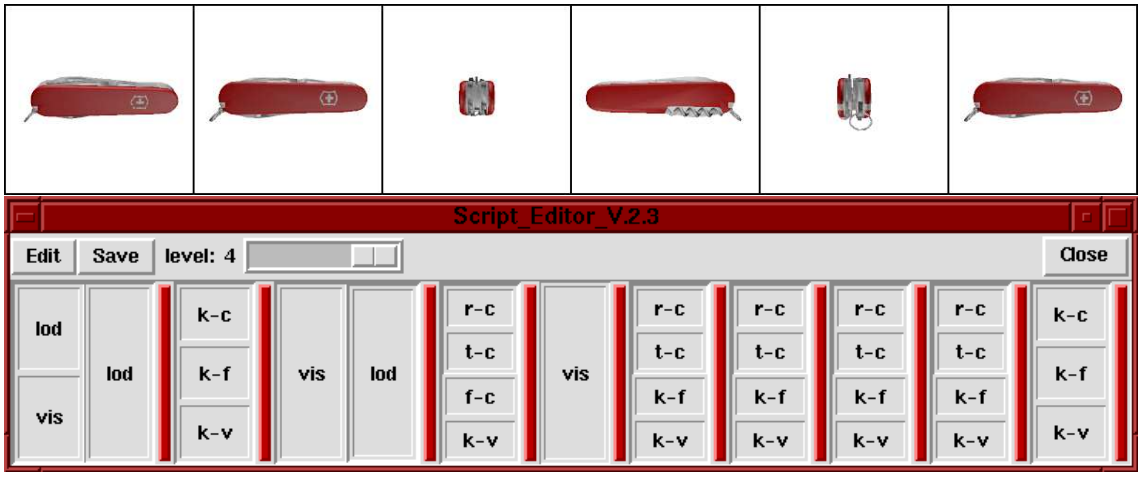


Abbildung 6.6: Rundflug um ein Objekt in 4 Schritten

zur Verfügung stehenden Parameter auch nicht benutzt und nicht im Skript spezifiziert werden. Das Resultat ist im schlimmsten Fall eine Animation, die sich außer der Bildsprache durch Kameraführung und Schnitt keiner gestalterischen Mittel bedient. Die Abbildung 7.6 auf Seite 113 zeigt, wie das Durchsichtigmachen von Objekten bei verschiedenen Ausgabequalitäten umgesetzt wird.

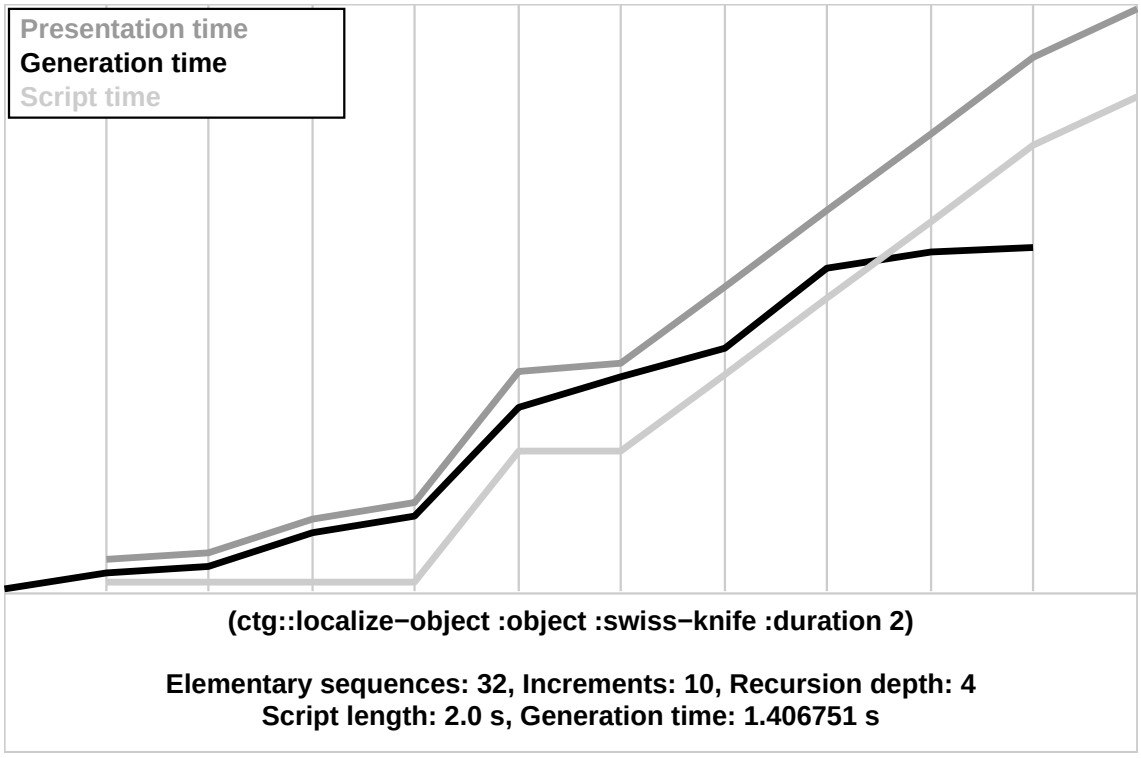


Abbildung 6.7: Korrektur des Timings bei mangelnder Generierungszeit

6.8 Komplexität des Ansatzes

Ausgehend von den vorgegebenen Präsentationszielen werden mit Hilfe der Skriptgrammatik Animationsskripte einer bestimmten Länge generiert. Die Länge dieser Skripte hängt von der Formulierung der Grammatik sowie von der Generierungssituation (Kontext) ab. Sie ist außerdem in der beschriebenen Art von den zur Verfügung stehenden Ressourcen abhängig. Diese Ressourcenabhängigkeit soll bei der folgenden Komplexitätsbetrachtung vorerst unberücksichtigt bleiben, da sie in jedem Falle nur zur Verbesserung des Laufzeitverhaltens beitragen kann.

Eingaben des Generierungsverfahrens sind die Präsentationsziele $P_1 \ldots P_n$ sowie die Regelmenge $\mathcal{R}$. Betrachtet man die Komplexität des Verfahrens in Abhängigkeit von der Mächtigkeit r der Regelmenge $\mathcal{R}$, so muß man zunächst einige Grundannahmen über die Regeln treffen. Zunächst muß sichergestellt sein, daß die Grammatik zyklenfrei ist. Hierzu definiert man über der Regelmenge $\mathcal{R}$ eine Relation $\prec$ und ihre transitive Hülle $\prec^*$ wie folgt (Animationsziele und gleichnamige Regeln werden hier der Einfachheit halber synonym verwendet): Seien $R_1, R_2 \in \mathcal{R}$

$$R_1 \prec R_2 \quad \Leftrightarrow \quad R_1 \text{ wird im Rumpf von } R_2 \text{ verwendet.} \tag{6.1}$$

$$R_1 \prec^* R_2 \quad \Leftrightarrow \quad \exists R_i, \ldots, R_k \in \mathcal{R} : R_1 \prec R_i \ldots \prec R_k \prec R_2 \tag{6.2}$$

Um Endlosrekursionen zu vermeiden möchte man sicherstellen, daß keine Dekompositionsregel ihre eigene Anwendung auf einer tieferen Rekursionsebene bewirkt. Eine Skriptgrammatik ist also zyklenfrei, wenn die Relation $\prec^*$ irreflexiv und asymmetrisch über $\mathcal{R}$ ist, das heißt

$$\forall R_i \in \mathcal{R} \quad : \quad R_i \not\prec^* R_i \tag{6.3}$$

$$R_1 \prec^* R_2 \quad \Rightarrow \quad R_2 \not\prec^* R_1 \tag{6.4}$$

Mit diesem Kriterium ist sichergestellt, daß bei der rekursiven Anwendung der Dekompositionsregeln keine Endlosrekursion auftreten kann, da jeder Ast des Ableitungsbaumes nur so viele Knoten enthalten kann wie die Regelmenge $\mathcal{R}$ Regeln. Dies läßt sich einfach durch Widerspruch zeigen. Seien

$$| \mathcal{R} | \quad = \quad r \text{ die Anzahl der Regeln} \tag{6.5}$$

$$R_1, \ldots, R_k \quad \in \quad \mathcal{R} \tag{6.6}$$

$$A \quad = \quad R_1 \prec \ldots \prec R_k \text{ ein Ast eines Ableitungsbaumes in } \mathcal{R} \tag{6.7}$$

$$k \quad > \quad r \tag{6.8}$$

Dann müßte sich wegen 6.6 und 6.8 mindestens ein R_i in A wiederholen, das heißt

$$\exists R_i \in \mathcal{R} \quad : \quad R_1 \prec \ldots \prec R_i \prec \ldots \prec R_i \prec \ldots \prec R_k \tag{6.9}$$

was im Widerspruch zu 6.3 und 6.2 steht. Eine weitere Annahme über die Eigenschaften der Skriptgrammatik ist ihr maximaler Verzweigungsfaktor b. Er ist definiert als die maximale Anzahl der in einer Dekompositionsregel spezifizierten Unterziele und somit auf jeden Fall endlich, sofern die Grammatikdefinition endlich ist. Macht man diese Annahmen über die Skriptgrammatik (Zyklenfreiheit, endlicher maximaler Verzweigungsfaktor), so läßt sich aufgrund der Anzahl der Grammatikregeln eine Obergrenze für die Länge der Animationsskripte angeben. Da die maximale Astlänge

des Ableitungsbaumes die Mächtigkeit der Regelmenge r wegen 6.3 und 6.4 nicht übersteigen kann, ergibt sich mit dem Verzweigungsfaktor b eine maximale Blattanzahl des Ableitungsbaumes von b^r. Dies ist somit die maximale Skriptlänge für ein Präsentationsziel. Nimmt man an, daß die Anwendung einer Dekompositionsregel in konstanter Zeit t_d geschieht, so wird für die maximal b^r terminalen Regeln maximal die Zeit $t_d * b^r$ benötigt. Die Zeit für die Anwendung der nichtterminalen Dekompositionsregeln läßt sich durch die maximale Anzahl der nichtterminalen Knoten des Ableitungsbaumes abschätzen zu $t_d * \sum_{i=0}^{r-1} b^i$, was wiederum für $b^r > r$ kleiner ist als $t_d * b^r$. Somit ist die maximale Laufzeit zur Erzeugung des Animationsskript für ein einzelnes Präsentationsziel

$$t \leq 2 * t_d * b^r \tag{6.10}$$

Eine Laufzeitbetrachtung des Generierungsverfahrens in Abhängigkeit von der Anzahl der Präsentationsziele ergibt trivialerweise ein lineares Laufzeitverhalten, da aus jedem einzelnen Ziel nacheinander das zugehörige Skript in der gezeigten maximalen Zeit generiert wird. Das Ergebnis für das Laufzeitverhalten des Verfahrens in Abhängigkeit von der Mächtigkeit r der Regelmenge ist auf den ersten Blick jedoch entmutigend. Es relativiert sich aber durch die folgenden Überlegungen:

Eine filmtechnisch sinnvolle Grammatik ist in ihrer Struktur sehr weit von den eben angenommenen *worst-case*-Grammatiken entfernt. Segmentiert man die Grammatik zusätzlich (Siehe Abschnitt 6.5.7.4), so kann man über eine Betrachtung der Regelanzahlen in den einzelnen Segmenten eine wesentlich bessere Abschätzung für die maximale Tiefe des Ableitungsbaumes angeben. Die Anzahl der Hierarchieebenen der entstehenden Ableitungsbäume in der prototypischen Implementation des Ansatzes (Kapitel 7) überschreitet nie den Wert von fünf, obwohl die Grammatik siebenundneunzig Regeln umfaßt. Die generierten Skripte haben eine Länge von bis zu fünfhundert elementaren Skriptsequenzen, so daß der mittlere Verzweigungsfaktor der Grammatik unter vier liegt. Die auftretenden Werte sind also sehr weit von den angegebenen Maximalwerten entfernt und es zeigt sich in der Praxis, daß das Verfahren sehr verträgliche Laufzeiten aufweist. Die Generierung der Skripte zu den in Anhang C gezeigten Animationen benötigt auf einem handelsüblichen Personal Computer nur wenige Sekunden. Die sehr hohe theoretische Komplexität des Ansatzes macht ihn also nicht unbenutzbar.

Sechste Szene

Mittlerweile ist es tiefe Nacht. Das Kaffeegeschirr ist einer Flasche Wein gewichen, die anfangs kritische Stimmung ist vollständig in Arbeitseifer umgeschlagen und kurz darauf beschließen die drei, die vielen Ideen doch gleich in die Praxis umzusetzen.

A: Also ich bin dabei. So ein paar grundlegende Regeln für deine Grammatik lassen sich bestimmt finden.

C: Gut. Ich kümmere mich um die Berechnungen. Gib mir ein bißchen Zeit, um die Modelle zu besorgen. Aber den Grammatikkram macht ihr...

B: Na klar. Dann brauchen wir nur noch jemanden, der uns die Bilder erzeugt...

C: ... Besorge ich ...

B: ... und jemanden, der unsere Drehbücher schön darstellt ...

A: Meinetwegen!

Die folgende Zeit ist angefüllt mit eifrigem Programmieren, ausprobieren, ändern und dem Ringen mit technischen Problemen.

Kapitel 7

Technische Umsetzung des Ansatzes im System CATHI

Ausgehend von dem in den vorangegangenen Kapiteln beschriebenen Ansatz wurde das System CATHI implementiert. CATHI steht für "Computer Animation Tool for Help and Information systems" und stellt die Animationskomponente des Systems $\mathcal{PPP}$ [André et al., 96, Rist et al., 97] dar. Der Präsentationsplaner des Systems $\mathcal{PPP}$ (beschrieben in [André, 95]) generiert koordinierte Multimediapräsentationen und verteilt bestimmte Präsentationsziele auf die verschiedenen Teile der Präsentation. Seine Ausgaben bilden also als Eingaben des Systems CATHI die in Abschnitt 6.5.1 eingeführten Animationsziele. Von diesen Zielen ausgehend generiert CATHI zunächst ein Drehbuch der Animation und gibt dieses dann inkrementell an einen Renderer weiter, der die Animation am Bildschirm zeigt. Abbildung 7.1

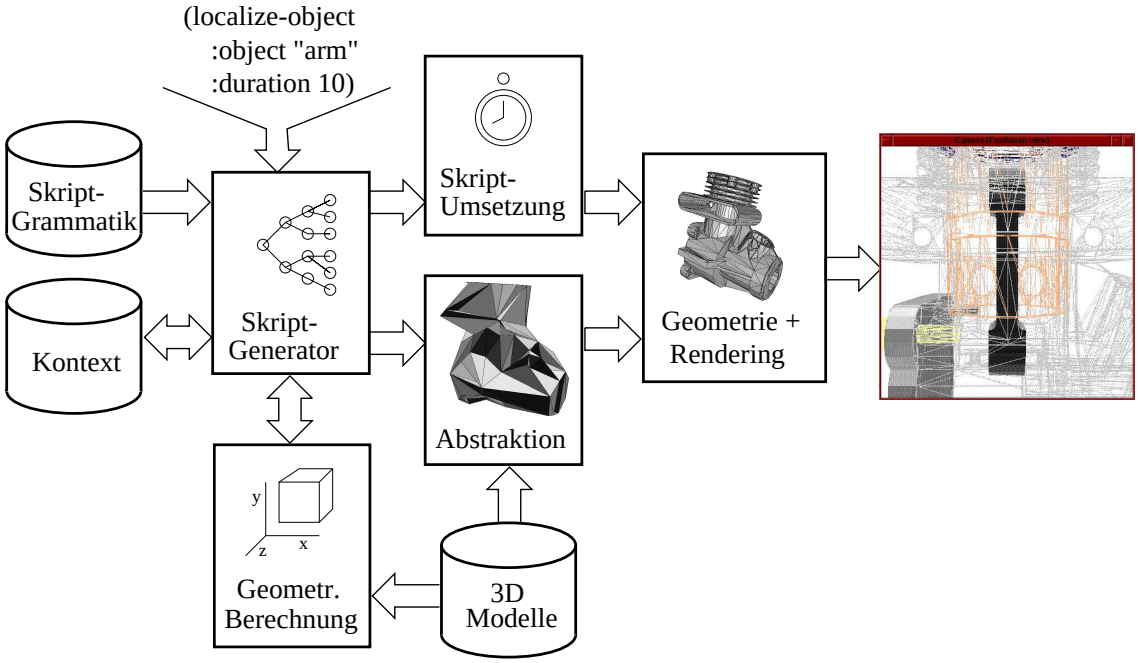


Abbildung 7.1: Architektur des Systems CATHI

zeigt den Aufbau des implementierten Systems mit den in den vorangegangenen Kapiteln beschriebenen Komponenten Skriptgrammatik, Kontext und Skriptgene-

rator sowie den Komponenten zur Durchführung der geometrischen Berechnungen, der Abstraktionen und zur Umsetzung der Animationsskripte in die Rendererspezifische Kommandosprache. In diesem Kapitel möchte ich die Implementation des Systems entlang der Bilderzeugungskette (Abbildung 2.6), also von den Modelldaten bis zur fertigen Animation beschreiben.

7.1 Beschaffung der Modelldaten

Um eine bestimmte Domäne bildlich darzustellen werden dreidimensionale Modelle der Domänenobjekte benötigt. Grundsätzlich steht man dabei vor der Wahl, Modelle im benötigten Zielformat selbst zu entwerfen, oder aber auf die große Anzahl im Internet vorhandener Modelle zurückzugreifen, die dann möglicherweise in einem anderen Format vorliegen als benötigt. Die insgesamt sinnvollere Alternative hierbei war, Konvertierungsroutinen für möglichst viele Modellformate zu schreiben und somit aus dem großen Vorrat vorhandener Modelle auswählen zu können.

7.1.1 Konvertierung anderer Formate

Im Rahmen meiner Arbeit habe ich deshalb eine Reihe von Konvertierungsprogrammen geschrieben, die es erlauben, Polygonmodelle der Dateiformate 3D-Studio, 3D-Studio-ASCII, DXF, AOFF, TPOLY und RAW in das von mir benötigte OOGL-Format zu konvertieren. Insbesondere das Format DXF ist ein sehr häufig verwendetes Format, da es von den meisten kommerziellen CAD-Systemen gelesen und geschrieben werden kann. Außerdem finden sich im Internet viele Modelle im Format des Animationssystems 3D-Studio, das im Heimbereich sehr verbreitet ist. Die genannten Formate enthalten Sammlungen von Polygonmodellen, die jeweils aus einer Liste von Knoten und einer Liste von Dreiecken zwischen diesen Knoten aufgebaut sind. Dieses Format wird auch von gängigen Graphikbibliotheken wie OpenGL effizient unterstützt und ist deshalb bei der schnellen Generierung von Animationen zu einer Art Standard geworden (Siehe auch Abschnitt 2.2.1.3).

7.1.2 Nachbearbeitung

Um die beschriebenen geometrischen Berechnungen zu beschleunigen, werden von den konvertierten Modellen zu Beginn jeweils geometrische Vereinfachungen in Form umschreibender Quader berechnet. Diese Quader werden dann mit den Modellen gemeinsam abgespeichert und brauchen nicht bei jedem Systemlauf neu berechnet zu werden. Außerdem werden die zur Perspektivenwahl (Abschnitt 5.2) benötigten bevorzugten Betrachtungsrichtungen manuell festgelegt, die Objekthierarchie modelliert und die Modelle zur Wiedergabe mit einem hochwertigen Renderer (RenderMan) mit den entsprechenden Materialbeschreibungen versehen.

7.2 Generierungsverfahren

Ein wichtiger Punkt bei der Implementation des Systems war, das Generierungsverfahren auf eine Art zu implementieren, die die Vorzüge des Ansatzes (Inkrementalität, Ressourcenadaptivität) unterstützt und gleichzeitig die Portabilität des Systems auf verschiedene Architekturen gewährleistet, damit die Leistungsfähigkeit des zugrundeliegenden Ansatzes auf verschiedenen Plattformen getestet werden kann. Zur Implementation des Generierungsverfahrens wurde LISP gewählt, da diese Sprache auf Systemebene schon viele Dienste bereitstellt, die in einer anderen Programmiersprache neu programmiert werden müßten.

7.2.1 Behandlung der Modelldaten

Jedes LISP-System enthält eine relativ mächtige Funktion zum Einlesen von Dateien oder Benutzereingaben, den LISP-Reader. Dieser Reader kann dazu benutzt werden, aus beliebigen Dateien bestimmte Informationen zu extrahieren. Im System CATHI werden auf diese Art die geometrischen Vereinfachungen, Bewegungsbeschreibungen, Materialbeschreibungen und Beschreibungen der Objekthierarchie aus den Modelldateien gewonnen. Die selben Informationen werden vom verwendeten Graphiksystem GeomView als Kommentare ignoriert, so daß für alle beteiligten Systeme eine einzige Modelldatei ausreicht.

7.2.2 Umsetzung des Generierungsverfahrens

Auch das Generierungsverfahren ist recht tief in die Programmiersprache LISP eingebettet. Durch die dort verfügbare gleiche Darstellung von Daten und Programmcode ist es möglich, eingelesene Daten direkt in ausführbare Programmteile zu verwandeln.

7.2.2.1 Grammatik

Die Dekompositionsregeln der Skriptgrammatik werden bereits beim Einlesen in eine interne prozedurale Form umgewandelt, so daß für jede Dekompositionsregel in Wirklichkeit eine entsprechende LISP-Prozedur definiert wird. Diese Prozeduren können mit dem Compiler des LISP-Systems in Maschinencode übersetzt werden und ihre Verarbeitung läßt sich hierdurch nochmals erheblich beschleunigen. Außerdem bietet jedes LISP-System an sich eine effiziente Verwaltung für LISP-Funktionen, so daß die Verwaltung der Regelmenge bereits auf der Systemebene stattfindet. Ein solches Verfahren hat neben einer Reduktion der Codemenge den Effekt, daß unnötige Programmschichten zwischen dem Grammatikformalismus und der Maschinenebene vermieden werden.

7.2.2.2 Kontextanfragen und geometrische Berechnungen

Der Generierungskontext ist durch eine Menge von LISP-Funktionen und -Variablen repräsentiert, die vom Generierungsprozeß aufgerufen beziehungsweise ausgelesen

und verändert werden können. Geometrische Berechnungen werden durch LISP-Funktionen ausgeführt, die direkt von der internen prozeduralen Form der Dekompositionsregeln aufgerufen werden. Somit ist die Erweiterung der zur Verfügung stehenden Berechnungsfunktionen einfach durch die Definition neuer LISP-Funktionen zu berwerkstelligen, was eine sehr leichte Erweiterung des Systems ermöglicht (Siehe auch Kapitel 9).

7.3 Benutzeroberfläche

In seiner eigentlichen Funktion als Animationsgenerator des Systems $\mathcal{PPP}$ benötigt CATHI natürlich keine Benutzeroberfläche, da eine direkte Interaktion mit dem Benutzer ja nicht stattfindet. Während der Entwicklung des Systems und insbesondere der Skriptgrammatik war es jedoch wichtig, interaktiv Animationsziele und Generierungsparameter vorgeben zu können.

7.3.1 Parameter und Ziele

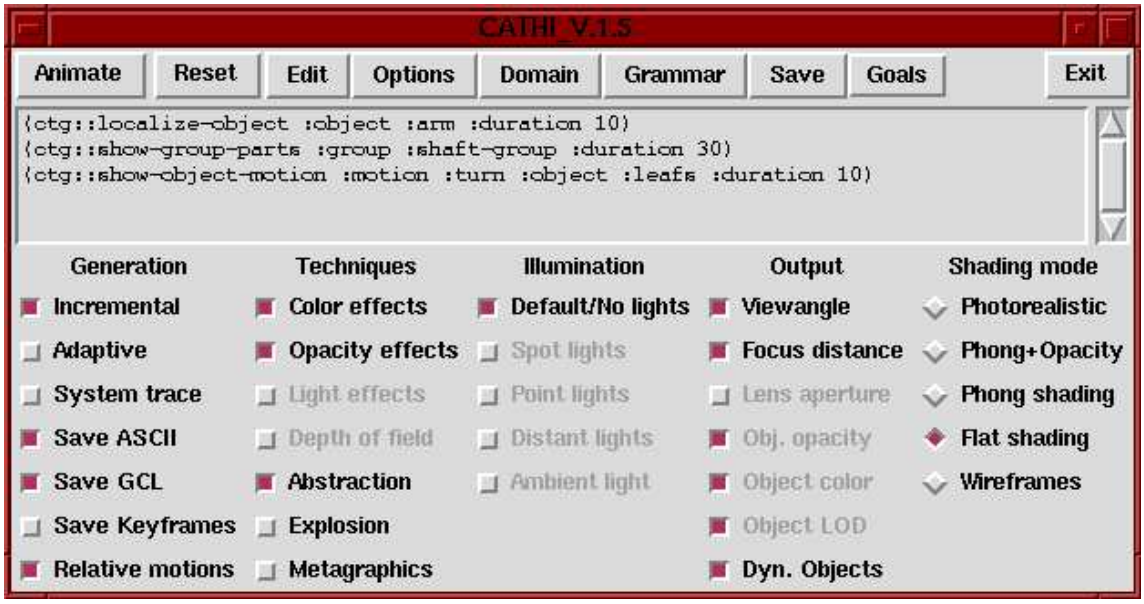


Abbildung 7.2: Benutzeroberfläche des Systems CATHI

Das in Abbildung 7.2 gezeigte (in TCL/TK implementierte) Interface erlaubt die Vorgabe der Generierungsparameter aus Abschnitt 6.2, das Einstellen bestimmter Parameter des Generierungsprozesses sowie die Formulierung von Animationszielen in einem Textfeld. Zur Auswertung der Generierungsergebnisse steht außerdem ein Skripteditor sowie eine Funktion zur Erstellung von Timing-Diagrammen zur Verfügung (Siehe Abschnitt 7.5.1).

7.3.2 Skripteditor

Der Skripteditor lehnt sich an die Darstellung der Animationsskripte im System S-Dynamics an. Auf der horizontalen Achse verläuft die Zeit, während zeitlich parallele

Skriptteile auf der vertikalen Achse untereinander angeordnet sind (Abbildung 7.3). Die senkrechten roten Balken stellen die Grenzen zwischen einzelnen Inkrementen

Abbildung 7.3: Skripteditor des Systems CATHI

des Skriptgenerators dar. Gruppen von Untersequenzen können als einzelnes Feld mit der Beschriftung *Parallel* beziehungsweise *Sequential* oder als entsprechende Anordnung der Untersequenzen dargestellt werden. Mit der Maus können solche Gruppen geöffnet oder geschlossen werden oder am Regler (Abbildung 7.3 links oben) generell eine Hierarchieebene eingestellt werden, bis zu der das hierarchische Skript geöffnet dargestellt wird. Diese Art der Darstellung erlaubt eine sehr schnelle Beurteilung des Generierungsergebnisses auf der Ebene seiner hierarchischen Struktur.

7.4 Umsetzung der Skripte in Animationen

Um aus den erzeugten Skripten Animationen zu berechnen, muß die plattformunabhängige Skriptsprache in Anweisungen für einen spezifischen Renderer übersetzt werden. Hierzu stehen in CATHI zwei verschiedene Möglichkeiten zur Verfügung.

7.4.1 Umsetzung in Echtzeit

Zur Umsetzung der Animationen in Echtzeit habe ich das als Shareware erhältliche Graphiksystem Geomview ([Phillips, 96]) verwendet. Geomview stellt Polygonmodelle im OOGL-Format am Bildschirm dar. Es läuft auf verschiedenen Plattformen und nutzt jeweils die vorhandene Hardware optimal aus. Je nach verwendetem Rechner können dabei Drahtgitterdarstellungen, flach schattierte, phong-schattierte oder texturierte Polygone dargestellt werden und auf manchen Plattformen ist auch die Opazität der dargestellten Polygone steuerbar. Somit konnte die Ressourcenadaptivität meines Generierungsansatzes bezüglich des Ausgabemediums direkt überprüft werden. Ein Beispiel für die verschiedenen Ausgabequalitäten zeigt Abbildung 7.6 auf Seite 113.

Zur Darstellung in Echtzeit wird ein Inkrement, sobald es fertig generiert ist, in die Geomview-Kommandosprache GCL übersetzt und an das Graphiksystem ausgegeben, das daraufhin selbständig die zugehörigen Bilder berechnet und anzeigt. Da die Prozesse der Skriptgenerierung und der Bildberechnung beziehungsweise -ausgabe auf einem Multi-Tasking-System parallel ablaufen, können sich die Generierungs- und die Präsentationszeit problemlos überlappen. Die zugehörigen Prozesse können sogar auf verschiedenen Maschinen laufen, da es in der Regel günstiger ist, die Bildberechnung und -ausgabe direkt auf dem Rechner vorzunehmen, mit dem der benutzte Bildschirm verbunden ist.

7.4.2 Umsetzung ohne Zeitbeschränkung

Um graphisch anspruchsvollere Stilmittel wie Scheinwerfer und Tiefenschärfe einsetzen zu können, ist ein leistungsfähigeres Bilderzeugungssystem notwendig. Die RenderMan-Graphikbeschreibungssprache ([Upstill, 90]) bot sich hierbei als offener Standard an, der plattformübergreifend verfügbar ist. Zur Bildberechnung aus RenderMan-Dateien gibt es verschiedene Programmpakete, von denen ich das Sharewareprodukt "Blue Moon Rendering Tools" (BMRT) von Larry Gritz [Gritz, 97] verwende. Die Animationsskripte werden also zunächst in die Beschreibung einzelner Bilder im RenderMan-Format übersetzt, aus denen anschließend die Bilddateien erstellt werden. Da die Berechnung eines einzelnen Bildes hierbei zwischen wenigen Sekunden und 1-2 Stunden liegt, ist eine Ausgabe in Echtzeit natürlich nicht mehr möglich. Aus den Einzelbildern können anschließend MPEG- oder GIF-Animationen erzeugt werden.

7.4.3 Verteilte Bilderzeugung

Um die große Menge an Bildern berechnen zu können, die für eine flüssige Animation benötigt werden, wurden die Bildberechnungsprozesse auf verschiedene Maschinen verteilt. Die hierzu verfügbaren knapp siebzig SUN-Workstations und -server des DFKI bilden bezüglich ihrer Rechenleistung eine sehr heterogene Renderfarm, die ihrerseits interessante Probleme aufwirft. Zur Verteilung der Prozesse verwendete ich nacheinander verschiedene Strategien, die ich hier kurz erläutern möchte, da sie einen interessanten Teil der Implementation bildeten.

7.4.3.1 Zentralistische Strategie

Der erste Ansatz bestand darin, jedem freien oder nicht voll ausgelasteten Rechner reihum einen Prozeß zuzuteilen bis alle Bilder berechnet waren. Da die Abarbeitung der Rechnerliste aber schon mehr als eine Minute in Anspruch nahm, wurden die schnellen Server, deren Bildberechnung schon nach wenigen Sekunden abgeschlossen war, viel zu selten mit neuen Prozessen versorgt. Eine geringfügige Verbesserung war dadurch erreichbar, daß die als schnell bekannten Rechner öfter abgefragt wurden. Das prinzipielle Problem bestand jedoch darin, daß der entstehende Verwaltungsaufwand für eine einzelne Maschine zu groß wurde.

7.4.3.2 Anarchistische Strategie

Der zweite Lösungsansatz bestand also darin, nicht nur die Bildberechnung selbst, sondern auch die Verwaltung der Bildberechnungsprozesse gleichmäßig auf alle Maschinen zu verteilen. Auf der Steuermaschine wurde also zunächst eine Liste mit zu erledigenden Prozessen erstellt und dann auf jeder Maschine im Netz ein Programm gestartet, das sich selbst aus dieser Liste einen Prozeß zuteilte, ihn als bearbeitet markierte, abarbeitete und nach erfolgreicher Erledigung aus der Liste strich. Kam aus irgendeinem Grund keine gültige Bilddatei zustande, so wurde der Prozeß als unbearbeitet in die Liste zurückgeschrieben und das Programm beendete sich auf der entsprechenden Maschine, da vermutlich ein Fehler vorlag. Durch die Tatsache,

daß alle beteiligten Maschinen sich selbst um die Zuteilung neuer Prozesse kümmerten, konnte der Durchsatz der gesamten Renderfarm und insbesondere der schnellen Maschinen deutlich erhöht werden, jedoch trat jetzt ein anderes Problem zutage: Wurde einer der letzten Prozesse von einer sehr langsamen Maschine übernommen, so wartete am Ende die gesamte Renderfarm auf diese langsame Maschine und im Extremfall wurde durch dieses Warten die Gesamtdauer mehr als verdoppelt.

7.4.3.3 Darwinistische Strategie

Um das geschilderte Problem zu beheben fragt bei der letzten entwickelten Verwaltungsstrategie jeder Rechner zu Beginn des Verfahrens seine eigene CPU-Leistung ab. Wie beim zweiten Verfahren teilen sich die Maschinen selbständig Prozesse aus der zentralen Liste zu, sofern ihre Systemlast eine bestimmte Grenze nicht überschreitet und markieren in der Liste zusätzlich, ob der Prozeß von einer schnellen oder einer langsamen Maschine bearbeitet wird. Die Abfrage der Systemlast wurde aus Rücksicht auf die an den Rechnern beschäftigten Mitarbeiter eingebaut, so daß nur wirklich freie Rechnerresourcen zum Rendern benutzt werden.

Sind schließlich keine neuen Prozesse mehr zu vergeben, so beendet sich das Programm auf den Maschinen, die sich selbst als langsam eingestuft haben. Auf den schnellen Maschinen hingegen beginnt eine zweite Phase, in der sie nacheinander die Bildberechnungsprozesse, die gerade von langsamen Maschinen bearbeitet werden, übernehmen und die Prozesse auf der entsprechenden langsamen Maschine beenden. Hierdurch sind zu Ende des Verfahrens nur noch die schnellen Maschinen an der Bilderzeugung beteiligt und das oben geschilderte Warteproblem entfällt. Gegenüber der zentralen Verteilung der Bildberechnungsprozesse konnte eine Geschwindigkeitssteigerung um den Faktor 3 erreicht werden, und die Auslastung der beteiligten Rechner lag im Durchschnitt bei über neunzig Prozent. Je nach Komplexität der Modellwelt und der verwendeten graphischen Mittel dauert das Rendern einer Animation mit 250 Bildern (10 Sekunden bei 25 Bildern pro Sekunde) in einer Auflösung von 300x200 Pixel zwischen 10 Minuten und wenigen Stunden. Bei den in Anhang C gezeigten Animationen sind jeweils die Berechnungszeiten vermerkt.

7.5 Verhalten in verschiedenen Umgebungen

Wie in Abschnitt 7.4.1 beschrieben laufen die Prozesse der Skriptgenerierung und der Bildberechnung voneinander unabhängig entweder auf der gleichen oder auf unterschiedlichen Maschinen. Je nach der für die einzelnen Prozesse zur Verfügung stehenden Rechenleistung ergeben sich verschiedene Verhaltensweisen des Systems durch die Beachtung der Ressource Rechenzeit. Außerdem hat das zur Verfügung stehende Ausgabemedium Einfluß auf die Qualität der erzeugten Bilder.

7.5.1 Planung und Timing

Die in Abschnitt 6.7.1 beschriebenen Effekte können entschärft werden, indem man die Skriptgenerierung und die Bildberechnung und Präsentation auf verschiedene Maschinen verteilt. Abbildung 7.4 zeigt das Systemtiming für den Fall, daß beide

Prozesse auf der gleichen Maschine, in diesem Falle einem Pentium 100 unter Linux laufen. Die dünnen vertikalen Striche des Diagrammes geben die Inkrementgrenzen entlang der horizontalen Achse an, während die verstreichende Zeit auf der vertikalen Achse abgetragen ist. Die hellste der drei Diagrammlinien gibt den geplanten Zeitverlauf der Animation wieder, die schwarze Linie den Zeitverlauf der Skriptgenerierung. Die zugehörige Animation sowie ihr Skript ist auf Seite 97 in Abbildung 6.3 gezeigt.

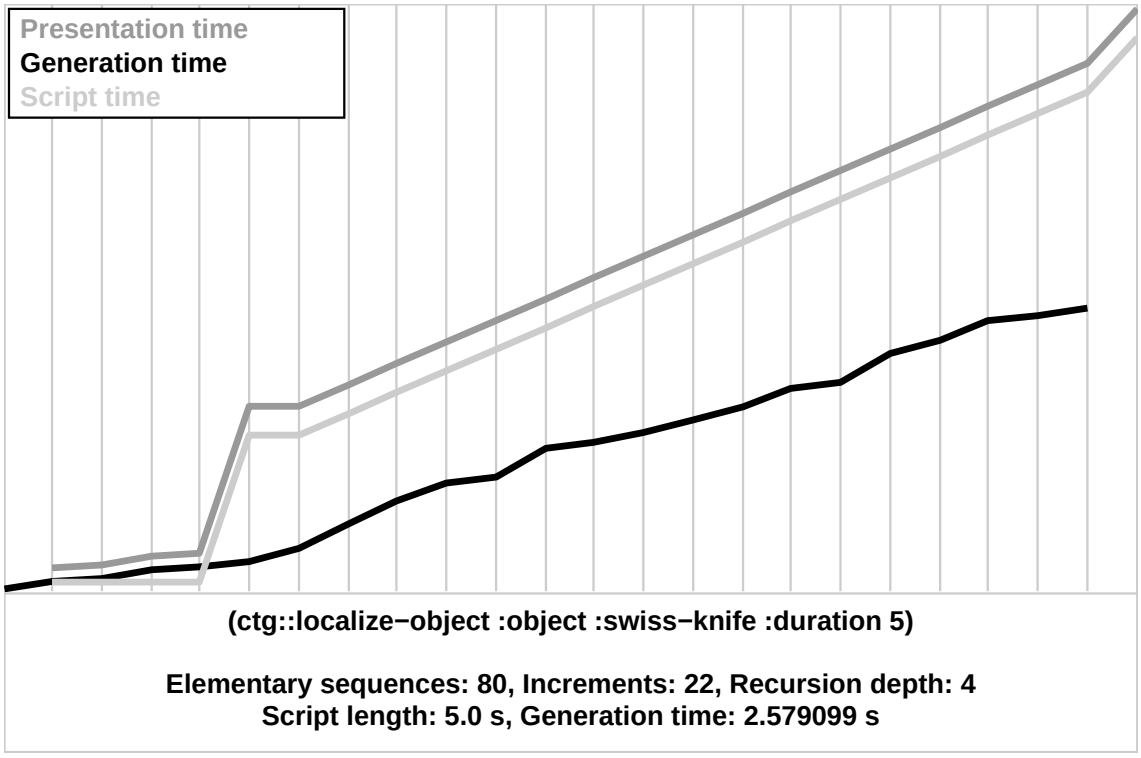


Abbildung 7.4: Timing bei Generierung und Präsentation auf einer Maschine

Verteilt man die beiden Prozesse nun auf verschiedene Maschinen, so verdoppelt sich die zur Skriptgenerierung und zur Bildberechnung verfügbare Rechenleistung jeweils. Das Resultat ist eine flüssigere Animation mit einer höheren Bildfrequenz sowie eine flachere Kurve für die Generierungszeit im Timingdiagramm. Abbildung 7.5 zeigt das Timing für die Verteilung der Prozesse auf zwei gleiche Maschinen.

7.5.2 Ausgabequalität

Bei wechselnden Darstellungsmöglichkeiten des Ausgabemediums müssen bestimmte Techniken graphisch unterschiedlich umgesetzt werden. Kann ein Renderer transparente Objekte darstellen, so können Verdeckungen direkt durch eine Steuerung der Objekttransparenz beseitigt werden (Abbildung 7.6 links). Auf anderen Rechnerarchitekturen als der SGI ist das verwendete Graphiksystem Geomview jedoch nicht in der Lage, die Transparenz korrekt darzustellen. In diesem Falle liefert die Drahtgitterdarstellung einen brauchbaren Ersatz (Abbildung 7.6 Mitte). Soll die Animation schließlich auf einem sehr langsamen Rechner mit Graustufenbildschirm dargestellt werden, so kann auch bei ausschließlicher Verwendung der Drahtgitterdarstellung ein ähnlicher Effekt erreicht werden, indem die störenden Objekte in einer Grau-

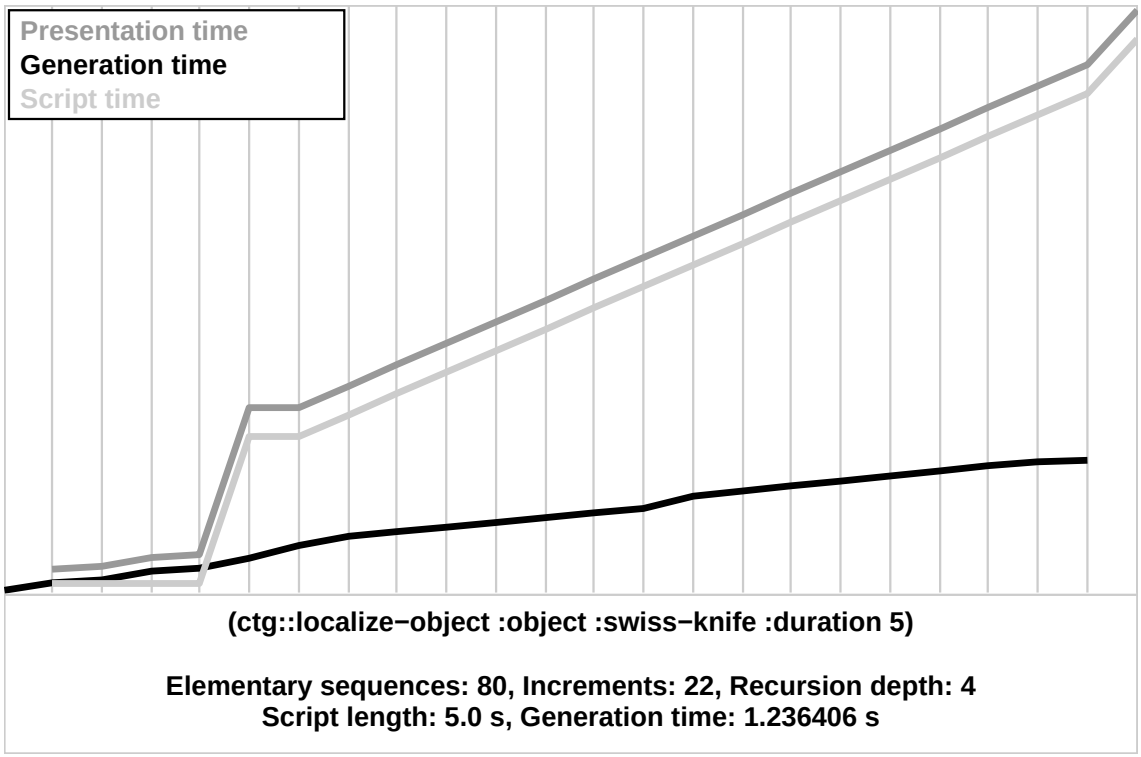


Abbildung 7.5: Timing bei Generierung und Präsentation auf zwei Maschinen

stufe gezeichnet werden, die sich weniger stark vom Hintergrund abhebt, bei einem weißen Hintergrund also zum Beispiel in einem hellen Grau (Abbildung 7.6 rechts).

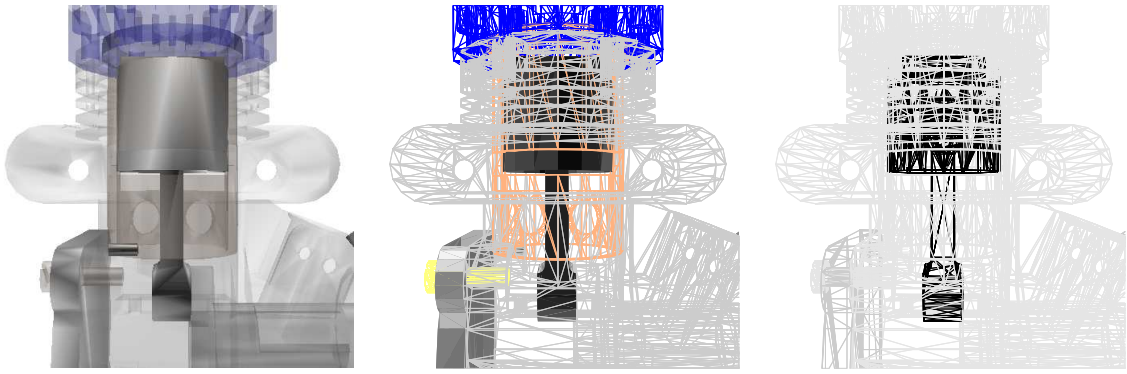


Abbildung 7.6: Verschiedene Ausgabequalitäten des Systems CATHI

Siebte Szene

Einige Zeit nach dem letzten Treffen sitzen die drei wieder zusammen. B hat seinen Laptop mitgebracht. Auf dem Bildschirm stehen fünf große Buchstaben: "CATHI"

A: So, dann laß mal sehen, was du aus unseren Ideen gemacht hast! Ich hoffe, es gibt was zu sehen...

B: (drückt eine Taste, auf dem Bildschirm bewegt sich kurz darauf etwas) Das ist unser System. Und es ist alles drin von unserer Sitzung neulich. (A und C schauen gebannt auf den Bildschirm, auf dem eine 3D-Animation abläuft.)

C: Also was ich immer noch nicht ganz verstehe: Wie funktioniert das jetzt genau mit deiner Grammatik? Kannst du das nicht mal vorrechnen?

A: Oh je, aber bitte so, daß ich es auch verstehe...

B: Ich werd's versuchen...

Kapitel 8

Ein ausführliches Generierungsbeispiel

Um das Zusammenspiel der in den Kapiteln 5 und 6 beschriebenen Verfahren und Komponenten zu zeigen soll nun ein ausführliches Generierungsbeispiel gegeben werden. Hierbei wird die Erzeugung eines einfachen Animationsskriptes sowie die dazu notwendigen Berechnungen Schritt für Schritt beschrieben. Die zugrundeliegende Modellwelt ist ein einzylindriger Benzinmotor, wie er für Modellflugzeuge verwendet wird. Das Polygonmodell besteht aus vierzehn Einzelobjekten, die hierarchisch zu sieben Objektgruppen zusammengefaßt sind (Siehe Abbildung 8.1 und 8.2). Zu

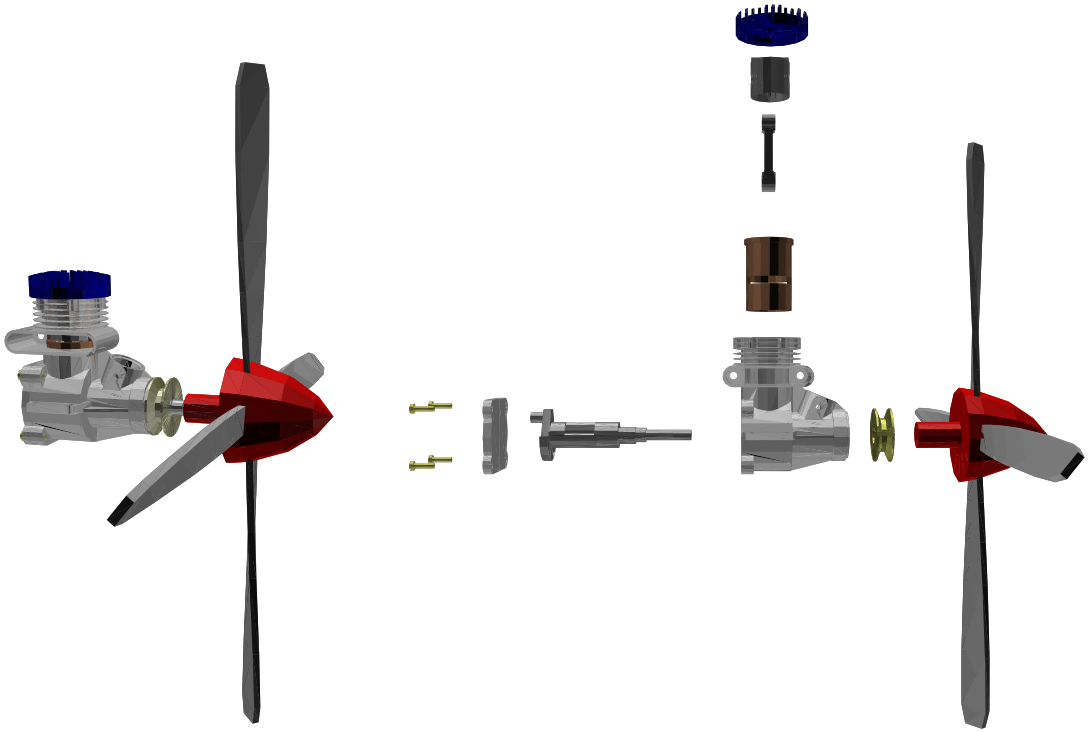


Abbildung 8.1: Bestandteile des Modellmotors

Beginn der Generierung des Skriptes wird die aktuelle Skriptzeit `*current-time*` auf den Wert 0.0 gesetzt. Außerdem sind die folgenden Generierungsparameter im Kontext abgelegt:

Parameter	Wert
Steuere Bildwinkel der Kamera	Nein
Steuere Entfernungseinstellung der Kamera	Nein
Steuere Objektivblende der Kamera	Nein
Steuere Transparenz der Objekte	Ja
Steuere Farbe der Objekte	Ja
Steuere Abstraktionsgrad der Objekte	Ja
Erzeuge metagraphische Objekte	Nein
Inkrementelle Generierung	Ja
Zeitadaptive Generierung	Nein
Benutze Farbeffekte	Ja
Benutze Transparenzeffekte	Ja
Benutze Beleuchtungseffekte	Nein
Benutze selektive Schärfe	Nein
Benutze Abstraktionen	Ja
Benutze Explosionen	Nein
Benutze Metagraphik	Nein

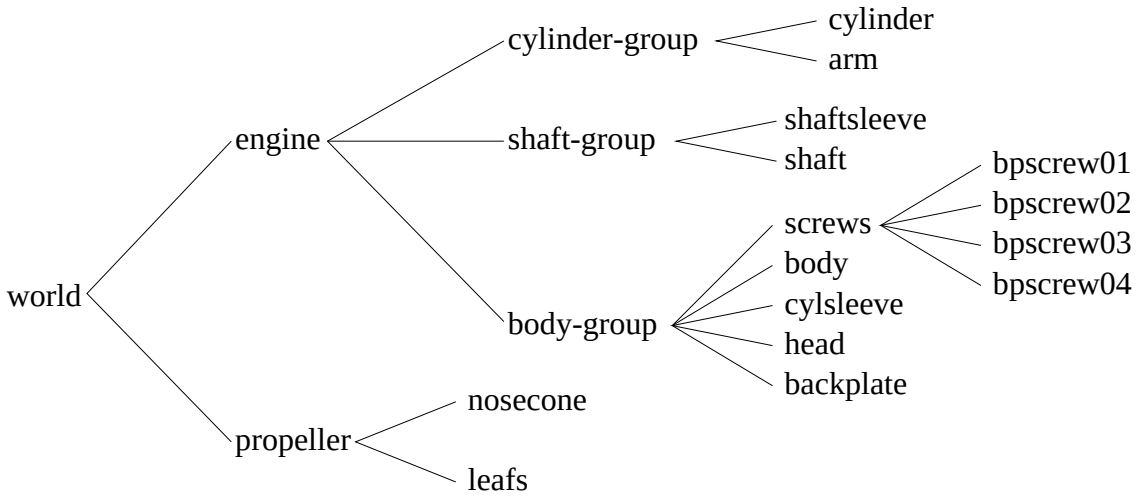


Abbildung 8.2: Objekthierarchie des Modellmotors

Ziel der Animation soll es sein, dem Betrachter zu vermitteln, an welcher Stelle des Motors sich die Baugruppe `"cylinder-group"` befindet. Ausgangspunkt der Generierung ist also das folgende Animationsziel:

```
(localize-object :object "cylinder-group" :duration 10)
```

Die Skriptgrammatik ist die in Anhang B aufgelistete Grammatik des Systems CATHI. In dieser Grammatik lautet die Dekompositionsregel für das angegebene Animationsziel:

```
(defrule localize-object (object)
  (incremental
   (start-shot :object object :duration 0.0)
   (cond ((same (parent-object object) :world)
          (localize-toplevel-object
             :object object :duration (* 0.9 duration)))
         ((and (feature :useexplosion) (isolatable object)
               (time-enough (+ 0.5 (* 0.1 (number-of-parts object)))))
          (localize-object-isolated
             :object object :duration (* 0.9 duration)))
         ((or (and (feature :invisible) (feature :useinvisible))
              (and (feature :color) (feature :usecolor)))
          (localize-regular-object
             :object object :duration (* 0.9 duration)))
         (t
          (go-from-here-to-object
             :object object :duration (* 0.9 duration))))
   (steady-shot :duration (* 0.1 duration))))
```

Es müssen also inkrementell nacheinander die Skriptteile zu den Unterzielen `(start-shot ...)`, `(cond ...)` und `(steady-shot ...)` generiert werden. Das erste dieser drei Unterziele wird durch die Belegung der Variable `object` mit dem Wert `"cylinder-group"` instanziiert zu dem Animationsziel

```
(start-shot :object "cylinder-group" :duration 0.0)
```

Das Konzept einer Starteinstellung wurde in Abschnitt 6.5.7.2 diskutiert. In der verwendeten Grammatik existiert hierzu die folgende Dekompositionsregel:

```
(defrule start-shot (object)
  (if (zerop *current-time*)
      (incremental
       (sequential
        (reset-world-state :duration 0)
        (look-at "world" :side '(:front :front :front :up))
        (setq *current-lights* nil)
        (when (feature :useabstract)
          (focus-with-abstraction :object object :duration 0))
        (when (feature :light) (steady-shot :duration 0)))
       (when (feature :light) (set-up-basic-lights))
       (steady-shot :duration duration))
    (sequential
     (when (feature :useinvisible)
       (visibility :level 1 :duration 0 :objects object))
     (steady-shot :duration duration))))
```

Die Bedingung `(zerop *current-time*)` in der ersten Zeile ist erfüllt, da diese Zeit ja zu Beginn der Generierung auf `0.0` gesetzt wurde. Es wird also der erste Teil des zugehörigen `(if ...)` Ausdruckes generiert. Dessen erste inkrementelle Untersequenz besteht wiederum aus sequenziell angeordneten Unterzielen. Das erste dieser Unterziele lautet `(reset-world-state :duration 0)`. Hierzu existiert die Dekompositionsregel

```
(defrule reset-world-state ()
  (parallel
   (level-of-detail
    :level 1 :duration duration :objects (all-scene-objects))
   (visibility
    :level 1 :duration duration :objects (all-scene-objects))
   (let ((objects (exploded-objects)))
     (undo-explosion)
     (update-positions :objects objects :duration duration))))
```

Die ersten beiden Unterziele dieser Dekomposition haben nun terminale Dekompositionsregeln, die folgendermaßen definiert sind:

```
(defrule level-of-detail ((level 1) (objects (all-scene-objects)))
  (when (and objects (feature :abstract))
        `(:level-of-detail ,start-time ,end-time ,duration
            :level ,level :objects ,(base-objects objects))))

(defrule visibility ((level 1)
                     (objects (objects-between-camera-and-aimpoint)))
  (when (and objects (feature :invisible))
        `(:visibility ,start-time ,end-time ,duration
            :level ,level :objects ,(base-objects objects))))
```

Das dritte Unterziel (`let ...`) ergibt die leere Sequenz, da zu Beginn dieser Generierung die Modellwelt noch nicht verändert wurde und somit keine explodierten Objekte existieren, die an ihre ursprüngliche Position zurückversetzt werden müßten. Insgesamt wird auf dieser Ebene also eine Gruppe von zwei parallel angeordneten elementaren Skriptsequenzen erzeugt:

```
(:PARALLEL
  (:LEVEL-OF-DETAIL 0.0 0.0 0.0
    :LEVEL 1
    :OBJECTS
     ("BACKPLATE" "BODY" "CYLINDER" "ARM" "CYLSLEEVE" "SHAFTSLEEVE"
      "HEAD" "LEAFS" "NOSECONE" "SHAFT" "BPSCREW04" "BPSCREW02"
      "BPSCREW03" "BPSCREW01"))
   (:VISIBILITY 0.0 0.0 0.0
     :LEVEL 1
     :OBJECTS
      ("BACKPLATE" "BODY" "CYLINDER" "ARM" "CYLSLEEVE" "SHAFTSLEEVE"
       "HEAD" "LEAFS" "NOSECONE" "SHAFT" "BPSCREW04" "BPSCREW02"
       "BPSCREW03" "BPSCREW01")))
```

Dieser erste Skriptteil wird vorerst noch zurückgehalten, da das übergeordnete Konstrukt ein (`sequential ...`) ist und somit alle seine Unterziele dekomponiert, bevor es den dadurch entstehenden Skriptteil zurückliefert. Bis zu diesem Zeitpunkt steht die aktuelle Skriptzeit `*current-time*` noch immer auf 0.0 und es wurde bisher auch noch keine Kameraposition berechnet. Dies geschieht im nächsten Ausdruck

```
(look-at "world" :side '(:front :front :front :up))
```

Hierdurch wird die geometrische Berechnung zur Auswahl der Kameraposition gestartet, und zwar für eine Position, die das Objekt `"world"`, also die gesamte Modellwelt aus der Richtung *(vorne, vorne, vorne, oben)* zeigt (Siehe Abschnitt 5.2 beziehungsweise Abbildung 8.4 links auf Seite 124). Die geometrische Berechnung setzt die aktuellen Kameraeinstellungen im Kontext auf die passenden Werte und liefert das leere Unterziel zurück. Der folgende Ausdruck `(setq *current-lights* nil)` löscht alle im Kontext eventuell noch vorhandenen Lichter. Da der Generierungsparameter "Benutze Abstraktionen" mit "ja" spezifiziert ist, generiert das anschließende Konstrukt

```
(when (feature :useabstract)
  (focus-with-abstraction :object object :duration 0))
```

durch die Belegung der Variable `object` mit dem Wert `"cylinder-group"` das Unterziel `(focus-with-abstraction :object "cylinder-group" :duration 0)`, dessen Dekompositionsregel folgendermaßen lautet:

```
(defrule focus-with-abstraction (object)
  (when (feature :useabstract)
    (parallel
     (level-of-detail :objects (all-objects-but object)
                      :level 0 :duration duration)
     (level-of-detail :objects object
                      :level 1 :duration duration))))
```

Es werden also die folgenden elementaren Skriptsequenzen generiert:

```
(:PARALLEL
  (:LEVEL-OF-DETAIL 0.0 0.0 0.0 :LEVEL 0 :OBJECTS
    ("BACKPLATE" "BODY" "CYLSLEEVE" "SHAFTSLEEVE" "HEAD" "LEAFS"
     "NOSECONE" "SHAFT" "BPSCREW04" "BPSCREW02" "BPSCREW03" "BPSCREW01"))
  (:LEVEL-OF-DETAIL 0.0 0.0 0.0 :LEVEL 1 :OBJECTS
    ("CYLINDER" "ARM")))
```

Damit ist das Konstrukt `(sequential ...)` vollständig abgearbeitet und der Generierungsprozeß kehrt zur nächsthöheren Rekursionsebene zurück, die durch ein `(incremental ...)` gruppiert ist. Die bisher generierten Skriptteile werden also zum Graphiksystem weitergereicht und durch die elementare Skriptsequenz

```
(:INCREMENT :SCRIPTTIME 0.0 :GENERATIONTIME 0.086854)
```

Abbildung 8.3: Graphische Darstellung des Beispielskripts im Skripteditor

abgeschlossen. Es sind bisher 0.086854 Sekunden verstrichen und die Ausgabe der Animation am Bildschirm beginnt. Am Bildschirm werden alle Objekte vollständig sichtbar und die Objekte der Gruppe `"cylinder-group"` voll detailliert dargestellt. Die nicht zu dieser Gruppe gehörenden Objekte erhalten den niedrigsten Detaillierungsgrad 0. Die Darstellung des Inkrementes reicht in Abbildung 8.3 bis zur ersten roten Markierung. Die verbleibenden Unterziele dieser Ebene sind

```
(when (feature :light) (set-up-basic-lights))
(steady-shot :duration 0))
```

und sie werden zum nächsten Skriptteil dekomponiert und zum Graphiksystem weitergegeben:

```
(:KEEP-CAMERA 0.0 0.0 :TRANSFORM (0.0 ... ))
(:INCREMENT :SCRIPTTIME 0.0 :GENERATIONTIME 0.102542)
```

Nach 0.1 Sekunden wird hiermit die Kameraposition im Graphiksystem spezifiziert. Somit ist die Ausgangsposition für die erste Kameraeinstellung festgelegt und die anschließende Kamerafahrt kann beginnen. Innerhalb der obersten Dekompositionsregel `localize-object` lautet das nächste Unterziel

```
(cond ((same (parent-object object) "world")
       (localize-toplevel-object
          :object object :duration (* 0.9 duration)))
      ((and (feature :useexplosion) (isolatable object)
            (time-enough (+ 0.5 (* 0.1 (number-of-parts object)))))
       (localize-object-isolated
          :object object :duration (* 0.9 duration)))
      ((or (and (feature :invisible) (feature :useinvisible))
           (and (feature :color) (feature :usecolor)))
       (localize-regular-object
          :object object :duration (* 0.9 duration)))
      (t
       (go-from-here-to-object
          :object object :duration (* 0.9 duration))))
```

Dieser Ausdruck stellt eine Verzweigung über verschiedenen Bedingungen dar und liefert als Ergebnis das Unterziel, dessen Bedingung zuerst erfüllt ist. Der erste Test `(same (parent-object object) "world")` überprüft, ob sich das Objekt `"cylinder-group"` in der Objekthierarchie (Abbildung 8.2) direkt unter dem Objekt `"world"` befindet. Da dies nicht der Fall ist, ist die Bedingung nicht erfüllt und die zweite Bedingung

```
(and (feature :useexplosion) (isolatable object)
     (time-enough (+ 0.5 (* 0.1 (number-of-parts object)))))
```

wird überprüft. Die im Kontext abgelegten Generierungsparameter werden durch Konstrukte der Form `(feature ...)` abgefragt. Da schon der erste Zweig der *und*-Verknüpfung nicht erfüllt ist (Generierungsparameter "Benutze Explosionen"), ist auch diese Bedingung nicht erfüllt und der dritte Fall wird überprüft:

```
(or (and (feature :invisible) (feature :useinvisible))
    (and (feature :color) (feature :usecolor)))
```

Da sowohl der Generierungsparameter "Benutze Farbeffekte" als auch der Parameter "Steuere Farbe der Objekte" mit "ja" spezifiziert sind, ist der erste Teil dieser *oder*-Verknüpfung und somit die gesamte Bedingung erfüllt. Ergebnis der gesamten Konstruktion `(cond ...)` ist also das Unterziel

```
(localize-regular-object :object "cylinder-group" :duration 9)
```

Für dieses Animationsziel existiert die folgende Dekompositionsregel:

```
(defrule localize-regular-object (object)
  (incremental
   (go-from-here-to-object :object object :duration (* 0.7 duration))
   (flash-objects-steady-shot :objects object
                              :duration (* 0.3 duration))))
```

Als nächstes wird also das Unterziel

```
(go-from-here-to-object :object "cylinder-group" :duration 6.3)
```

generiert, dessen zugehörige Dekompositionsregel wie folgt definiert ist:

```
(defrule go-from-here-to-object (object (fill 0.9) (arrow t))
  (cond ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 1.0)
              (< (gx::angle-between-vectors
                  (object-direction object :up)
                  *current-upvector*) 2.0))
         (zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))
        ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 2.0)
              (< (gx::angle-between-vectors
                  (object-direction object :up)
                  *current-upvector*) 2.0))
         (turn-and-zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))
        (t
         (cut-and-zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))))
```

Die Abarbeitung dieser kontextabhängigen Dekompositionsregel liefert im ersten Ast ihrer Verzweigung das Unterziel

```
(zoom-from-here-to-object
  :object "cylinder-group" :fill 0.9 :arrow t :duration duration))
```

mit der zugehörigen Dekompositionsregel

```
(defrule zoom-from-here-to-object (object (fill 0.9) (arrow t))
  (incremental
   (sequential
    (look-at object :side :front :fill fill)
    (remove-disturbing-objects :object object :duration 0)
    (focus-by-all-means :object object :duration 0 :arrow arrow))
   (update-camera-and-lights :duration duration)
   (remove-arrow :object object :duration 0)))
```

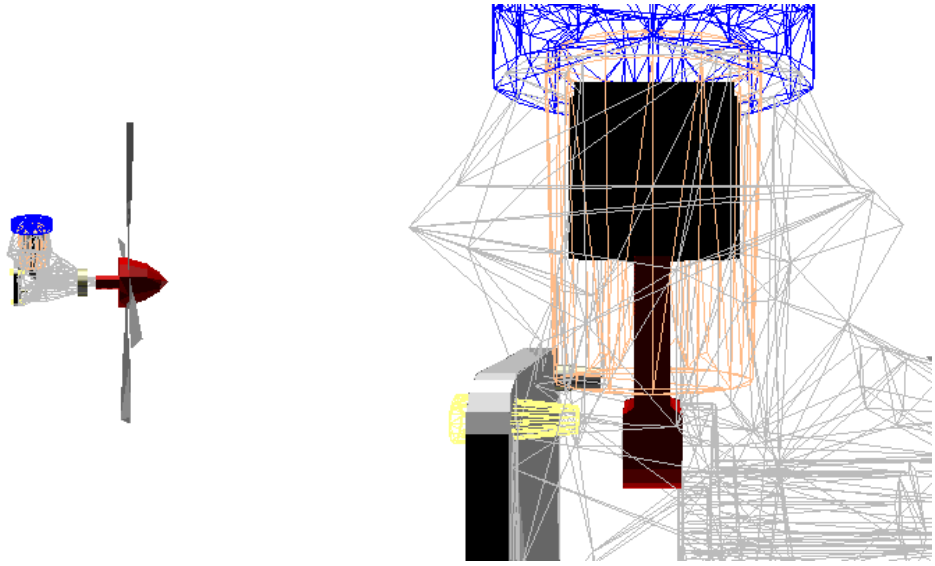


Abbildung 8.4: Start- und Zielposition der Kamerafahrt

Die Abarbeitung dieser Dekompositionsregel sollte aus der vorangegangenen Beschreibung klar geworden sein. Es wird zunächst eine neue Kameraposition berechnet, und zwar die Zielposition der Kamerafahrt (Abbildung 8.4 rechts). Während dieser geometrischen Berechnung wird die *aktuelle* Kameraposition zur *vorhergehenden* und an ihre Stelle werden die neu berechneten Werte gesetzt. Dies ermöglicht bei der nachfolgenden Dekomposition des Unterziels

```
(update-camera-and-lights :duration duration)
```

die Generierung der entsprechenden Kamerafahrt von der alten zur neuen Kameraposition, da beide im Kontext enthalten sind. Vorher wird jedoch durch die Dekompositionsregeln

```
(defrule remove-disturbing-objects (object)
  (when (feature :useinvisible)
    (parallel
     (visibility
      :level (if (feature :uselight) 0 0.2)
      :duration duration :objects
      (union (objects-between-camera-and-aimpoint)
             (all-objects-roughly-including object)
             :test #'equal))
     (visibility :level 1 :duration duration
                 :objects object))))

(defrule focus-by-all-means (object (arrow t))
  (parallel
   (focus-with-abstraction :object object :duration duration)
   (when arrow (focus-with-arrow :object object :duration duration))
   (focus-with-spotlight :object object :duration duration)))
```

sichergestellt, daß das Zielobjekt der Kamerafahrt vollständig sichtbar und die es verdeckenden Objekte durchsichtig sind. Außerdem werden die Fokusobjekte, also

die Bestandteile der Objektgruppe "cylinder-group" voll detailliert dargestellt, während der Detaillierungsgrad der übrigen Objekte auf Null gesetzt wird. Das nächste Inkrement lautet somit

```
(:SEQUENTIAL
  (:PARALLEL
    (:VISIBILITY 0.0 0.0 0.0 :LEVEL 0.2 :OBJECTS
       ("BPSCREW01" "BPSCREW04" "SHAFT" "HEAD" "CYLSLEEVE" "BODY"))
    (:VISIBILITY 0.0 0.0 :LEVEL 1 :OBJECTS
       ("CYLINDER" "ARM")))
  (:PARALLEL
    (:PARALLEL
      (:LEVEL-OF-DETAIL 0.0 0.0 0.0 :LEVEL 0 :OBJECTS
        ("BACKPLATE" "BODY" "CYLSLEEVE" "SHAFTSLEEVE" "HEAD" "LEAFS"
         "NOSECONE" "SHAFT" "BPSCREW04" "BPSCREW02" "BPSCREW03"
         "BPSCREW01"))
      (:LEVEL-OF-DETAIL 0.0 0.0 0.0 :LEVEL 1 :OBJECTS
        ("CYLINDER" "ARM")))))
(:INCREMENT :SCRIPTTIME 0.0 :GENERATIONTIME 0.290375)
```

Nach insgesamt 0.29 Sekunden werden somit am Bildschirm die verdeckenden Objekte durchsichtig gemacht und die Generierung der anschließenden Kamerafahrt durch die Dekompositionsregel

```
(defrule update-camera-and-lights  ((lights *current-lights*))
  (parallel
   (update-camera :duration duration)
   (update-lights :duration duration :lights lights)))
```

beginnt. Da keine Lichter verwendet werden, lautet das resultierende Inkrement

```
(:PARALLEL
  (:PARALLEL
    (:PARALLEL
      (:ROTATE-CAMERA 0.0 6.3 6.3
        :AXIS ((0.0 0.0 0.0) (0.0s0 -0.99999s0 0.0s0)) :ANGLE 0.3217163)
      (:TRANSLATE-CAMERA 0.0 6.3 6.3
        :FROM-POSITION (-52.808s0 -4.707s0 10.2656s0)
        :TO-POSITION (-8.4628s0 -0.8625s0 -4.6763s0)))))
(:INCREMENT :SCRIPTTIME 6.2999997 :GENERATIONTIME 0.660428)
```

Die dreifache Schachtelung des Konstruktes (:PARALLEL ...) kommt durch die dazwischenliegenden weiteren Dekompositionsebenen zustande, die eine Veränderung weiterer Kameraparameter und einer eventuell vorhandene Beleuchtung gruppieren. Die Schachtelung ist in dieser Situation zwar überflüssig, sie wird jedoch notwendig, sobald parallel zur Kamerafahrt noch Lichter gesteuert oder der Bildwinkel verändert werden soll. Somit sind 6.3 Sekunden des Skriptes generiert und während die Kamerafahrt am Bildschirm abläuft, wird das anschließende Blinken der Objektgruppe generiert. Das zugehörige Animationsziel und seine Dekompositionsregel lauten:

```
(flash-objects-steady-shot :objects "cylinder-group" :duration 0.27)
```

```
(defrule flash-objects (objects color times)
  (unless times (setq times (max (round duration) 3)))
  (parallel
    (loop for o in (base-objects objects) collect
          (flash-object :object o :color color
                        :times times :duration duration))))
```

In einer Schleife über die Bestandteile der Baugruppe `"cylinder-group"` werden die einzelnen Blinksequenzen erzeugt, die zusammen mit der Sequenz zum Festhalten der Kameraposition das nächste Inkrement ergeben:

```
(:PARALLEL
  (:PARALLEL
    (:SEQUENTIAL
      (:SEQUENTIAL
        (:COLOR 6.3 6.75 0.45 :COLOR (0 0 0) :OBJECTS ("CYLINDER"))
        (:COLOR 6.75 7.2 0.45 :COLOR (1 1 1) :OBJECTS ("CYLINDER")))
  ...

      (:SEQUENTIAL
        (:COLOR 8.1 8.55 0.45 :COLOR (1 0 0) :OBJECTS ("ARM"))
        (:COLOR 8.55 9.0 0.45 :COLOR (0.8 0.8 0.8) :OBJECTS ("ARM")))))
  (:PARALLEL
    (:PARALLEL
      (:KEEP-CAMERA 6.3 9.0 2.7 :TRANSFORM (-4.45 ... )))))
(:INCREMENT :SCRIPTTIME 9.0 :GENERATIONTIME 0.935345)
```

Nach insgesamt 0.93 Sekunden ist die Sequenz vollständig generiert und kann zum Graphiksystem weitergegeben werden. Das verbleibende Animationsziel

```
(steady-shot :duration 1.0)
```

ergibt ein einsekundiges Standbild als letztes Inkrement. Der Zeitablauf des gesamten Generierungsprozesses ist in Abbildung 8.5 nochmals graphisch dargestellt und Abbildung 8.3 auf Seite 121 zeigt das erzeugte Animationsskript im Skripteditor des Systems CATHI (Kapitel 7).

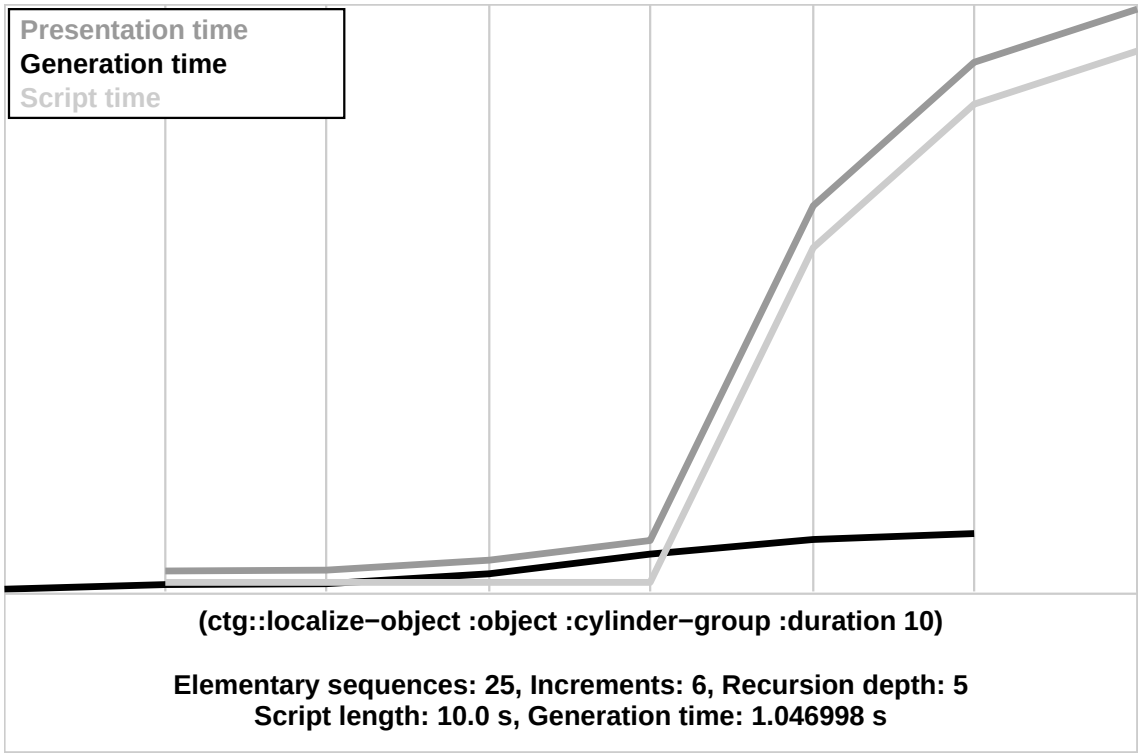


Abbildung 8.5: Zeitablauf bei der Generierung des Beispielskripts

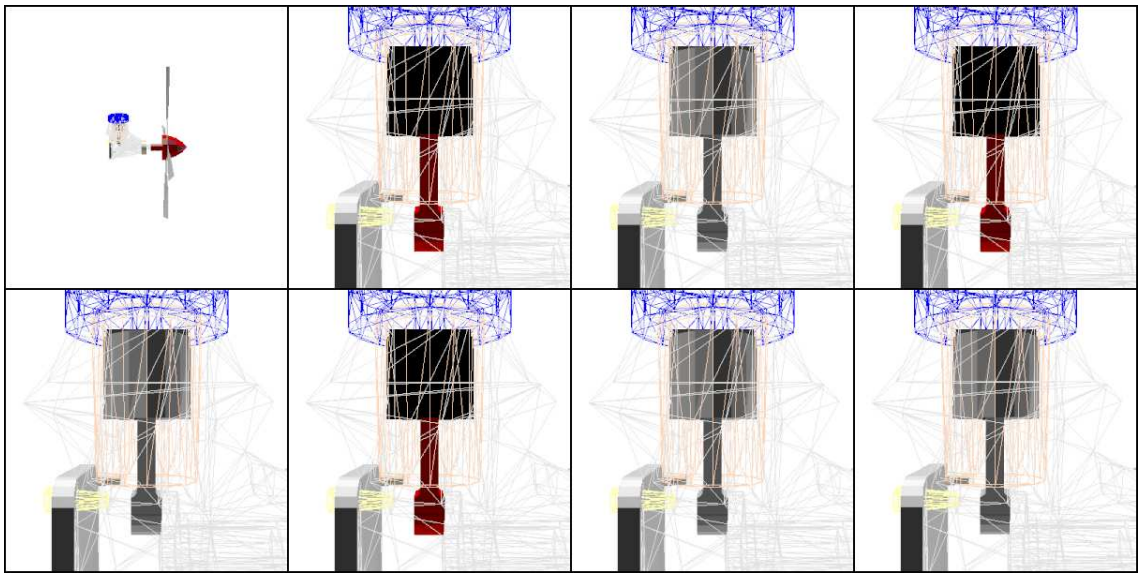


Abbildung 8.6: Keyframes der generierten Animation

Achte Szene

Nachdem B seine Musterrechnung beendet hat schweigen alle beeindruckt. Weder A noch C wollen zugeben, daß sie schon in der Hälfte den Faden verloren haben und eigentlich nur noch aus Höflichkeit weiter zugehört haben.

A: Tja, echt beeindruckend. So arbeitet also ein automatischer Skriptgenerator. Der hat ja fast noch mehr Kleinarbeit zu leisten als ein richtiger Drehbuchautor.

B: Richtig. Aber er hat ja auch alle Aufgaben des Filmteams am Hals: Drehbuchautor, Kameramann, Abstraktor, Metagraphiker...

C: Immerhin kommen dabei wirklich brauchbare Animationen raus. Und je länger ich darüber nachdenke, desto mehr fällt mir ein, was man mit so einem System noch alles machen könnte.

A: Man müßte halt mal noch ein paar andere Grammatiken entwerfen. Vielleicht ginge das ja auch eleganter, als sie einfach einzutippen.

B: Tja, und eigentlich sollte man das alles noch mal sauber und wissenschaftlich aufschreiben und veröffentlichen, damit auch andere etwas davon haben.

A: ...Was, aufschreiben? Veröffentlichen? Bist du verrückt?

C: Das Tüfteln hat ja noch Spaß gemacht, aber aufschreiben... Können wir das nicht irgendeinem Studenten auf's Auge drücken?

Kapitel 9

Ergebnisse und Ausblick

9.1 Wissenschaftlicher Beitrag der Arbeit

In der vorliegenden Arbeit wurde ein neuer Ansatz zur automatischen Generierung von 3D-Animationen vorgestellt. Dieser Ansatz hat mehrere Eigenschaften, die ihn von den bisherigen Arbeiten zur automatischen Animationsgenerierung wesentlich unterscheiden. Diese Eigenschaften sind:

- **Inkrementelle Skriptgenerierung**
  Das in Kapitel 6 vorgestellte Generierungsverfahren reicht einzelne in sich abgeschlossene Skriptteile zur Präsentation weiter, sobald sie generiert sind. Hierdurch kann die Verzögerung vom Beginn der Skriptgenerierung bis zum Beginn der Ausgabe drastisch verkürzt werden. Das Kräftedreieck aus Rechenleistung, Rechenzeit und Qualität des Ergebnisses wird hierdurch gestreckt.

- **Umfassende Nutzung des Mediums 3D-Animation**
  In der vorliegenden Arbeit wurden zum ersten Mal sämtliche Aspekte des Mediums 3D-Animation in den automatischen Gestaltungsprozeß einbezogen. Neben der Kameraführung und dem Schnitt, die in mehreren der in Kapitel 3 beschriebenen Systeme geplant werden, werden in der vorliegenden Arbeit sämtliche in einer Animation zu steuernden Parameter in die Planung einbezogen. Diese Parameter sind die Beleuchtung, Objektfarben, Transparenz, Metagraphik und die Detaillierungsgrade der verwendeten Modelle. Hierdurch wird eine wesentlich bessere Nutzung der Ausdrucksstärke des Mediums bei gleichzeitiger Reduktion des Aufwandes bei der Bilderzeugung erreicht.

- **Ressourcenadaptivität**
  Um in möglichst vielen verschiedenen Situationen passende 3D-Animationen erzeugen zu können, berücksichtigt der beschriebene Ansatz bei der Gestaltung der Animationen die zur Verfügung stehenden Ressourcen. Bildsprachliche Mittel, die in einem konkreten Fall nicht darstellbar sind, wie zum Beispiel Farbeffekte auf einem Schwarzweißbildschirm, werden nicht eingesetzt und soweit möglich durch andere Mittel ersetzt. Außerdem überwacht das Generierungsverfahren seine eigene Laufzeit und ist somit in der Lage, bei knapper Generierungszeit einfachere Skripte zu generieren.

- **Einbeziehung verschiedener Sichtweisen**
  Obwohl die Computergraphik und die Künstliche Intelligenz lediglich verschiedene Richtungen des Faches Informatik bilden, unterscheiden sich die jeweils verwendeten Methoden doch gewaltig. Während die möglichst allgemeine deklarative Repräsentation von Information und die formale Ableitbarkeit neuer Information daraus eine Maxime der KI darstellt, werden viele Problemlösungen der Computergraphik durch spezialisierte und hocheffiziente Verfahren algorithmisch beschrieben. In der vorliegenden Arbeit wurde der Versuch unternommen, durch eine hybride Konzeption des Generierungsverfahrens die Stärken beider Arbeitsweisen miteinander zu kombinieren. Außerdem wurde die Arbeit durch die Benutzung von Ideen und Metaphern aus dem Bereich der Filmproduktion und Photographie wesentlich bereichert.

- **Praktische Überprüfung des Ansatzes**
  Mit dem System CATHI wurde eine prototypische Implementation des Ansatzes vorgestellt und seine Generierungsergebnisse an Beispielen aus drei verschiedenartigen Domänen erläutert. Das System CATHI zeichnet sich durch eine effiziente Implementation sowie eine hohe Portabilität aus, wodurch die Eigenschaften des Generierungsansatzes unter den verschiedensten Bedingungen verifiziert werden konnten. So läuft das System auf Laptops, PCs und Graphikworkstations verschiedener Leistungsklassen und generiert dort jeweils Animationen, die die vorhandenen Ressourcen bestmöglich ausnutzen. Hierdurch konnte ein praktischer Nachweis für die Ressourcenadaptivität des Systems erbracht werden.

Aus Sicht der anwendungsorientierten Grundlagenforschung soll die vorliegende Arbeit eine Grundlage für weitere Forschungen bilden und gleichzeitig einen nachweislich gangbaren Weg für künftige Anwendungen aufzeigen.

9.2 Erweiterungen und weitere Anwendungen

Der in der vorliegenden Arbeit vorgestellte Ansatz zur inkrementellen Skriptgenerierung wurde im Kontext eines intelligenten Multimedia-Präsentationssystems (IMP-System) entwickelt. Durch seine Grundeigenschaften wie Inkrementalität, Ressourcenadaptivität und hohe Generierungsgeschwindigkeit ist er jedoch auch für andere Anwendungsgebiete geeignet. Zudem gibt es verschiedene Möglichkeiten, den Ansatz sinnvoll für diese anderen Anwendungsgebiete zu erweitern.

Die automatische Generierung informativer 3D-Animationen ist überall dort einsetzbar, wo es um die intelligente Darstellung räumlicher Objekte und Information geht. Solche Situationen sind beispielsweise die Erklärung beliebiger, als 3D-Modell vorliegender Domänen beim computerunterstützten Lernen, die Darstellung räumlicher Information bei der Fehlerdiagnose in komplexen Maschinen oder industriellen Anlagen sowie die Visualisierung geplanter Bauvorhaben für Architekten.

9.2.1 Computerunterstütztes Lernen

Wie in Abschnitt 3.2 beim System Zoom Illustrator beschrieben, können automatisch generierte Animationen dazu eingesetzt werden, bestimmte Lerninhalte am Bildschirm plastisch zu vermitteln. Die dort gezeigte Domäne medizinischer Modelle wäre auch mit dem vorliegenden Ansatz zu behandeln und ein Schritt in diese Richtung wurde bereits mit der Wahl eines Molekülmodells als Beispieldomäne des Systems CATHI unternommen (Abbildung 9.1 und Beispielanimationen in Anhang C). In diesem Zusammenhang wäre es sicherlich wünschenswert, die Präsen-

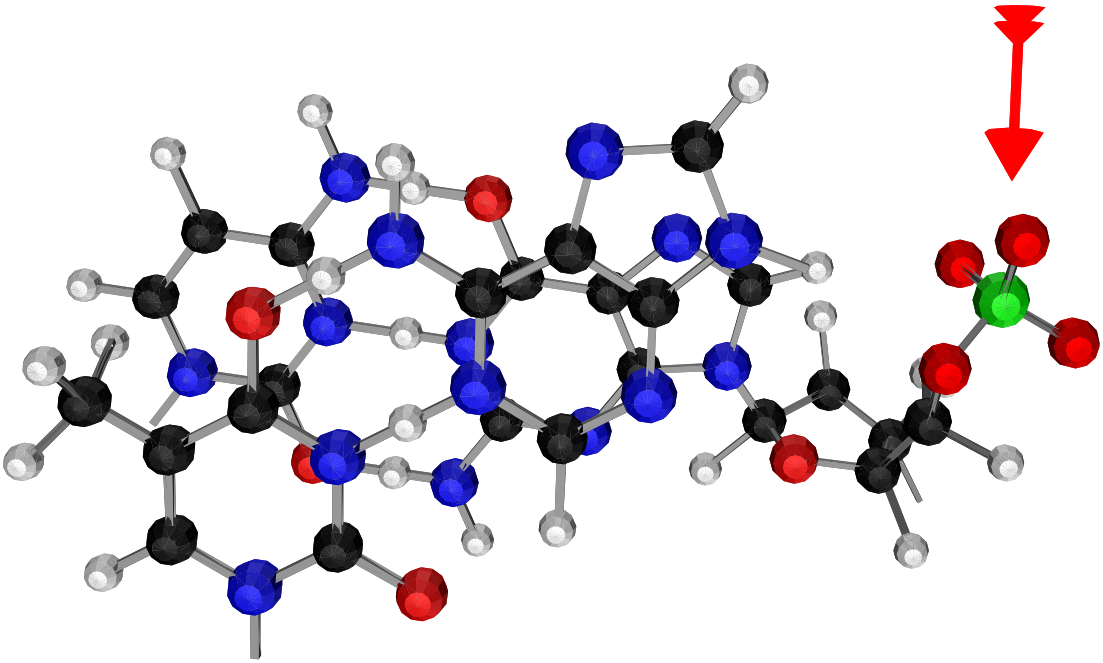


Abbildung 9.1: Visualisierung eines chemischen Modells

tationen bis zu einem gewissen Grad interaktiv zu machen. Die Verzahnung zwischen Graphiksystem und Skriptplanung müßte hierzu enger ausgelegt werden und der Skriptgenerator müßte in der Lage sein, Interaktionen des Benutzers aus dem Graphiksystem abzufragen und darauf zu reagieren. So könnte der Betrachter beispielsweise Objekte direkt im Bild mit der Maus selektieren und Anfragen dazu stellen. Der Skriptgenerator könnte diesen Anfragen neue Präsentationsziele entnehmen und die entsprechenden Animationen direkt generieren. Eine andere denkbare Form der Interaktivität besteht einfach darin, dem Betrachter die Möglichkeit zu geben, die Kameraposition selbst zu manipulieren und dann mit einer neuen Animationssequenz von dieser Kameraposition aus zu starten, statt eine neue Position vorzugeben. Auch hierzu wäre es lediglich nötig, vor der ersten Einstellung der neuen Sequenz den Weltzustand des Graphiksystems abzufragen und diesen im Kontext als aktuellen Weltzustand einzutragen. Dies kann beispielsweise in der in Abschnitt 6.5.7.2 beschriebenen Starteinstellung geschehen. In der gegenwärtigen Implementation des Ansatzes sind die beschriebenen Formen der Interaktivität nicht vorgesehen, um die klare Trennung zwischen Skriptgenerierung und Bilderzeugung beizubehalten. Nur so kann problemlos gewährleistet werden, daß das gleiche Animationsskript mit verschiedenen Bilderzeugungsverfahren (Siehe Abschnitt 7.4) umgesetzt werden kann.

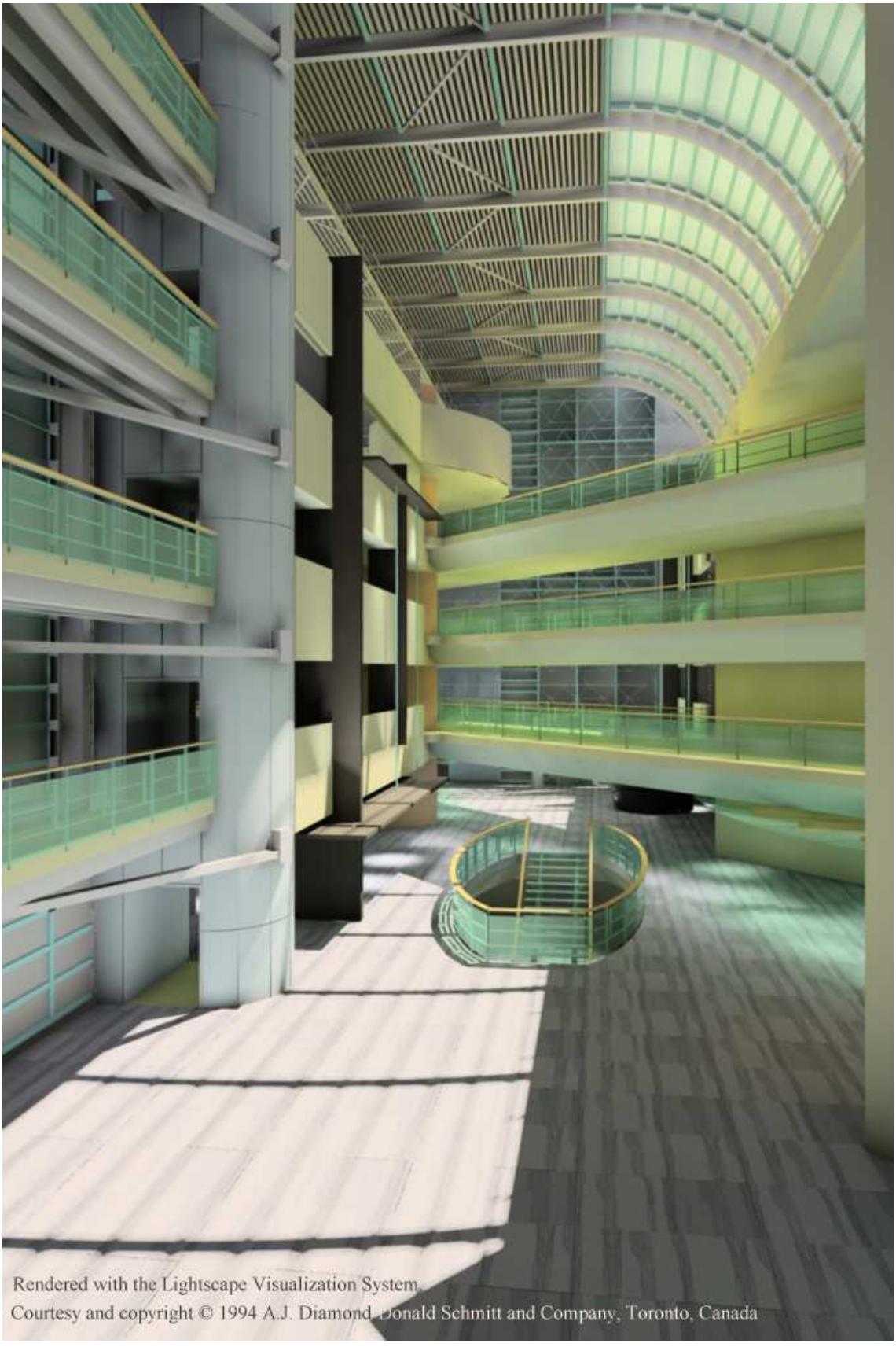


Abbildung 9.2: Hochwertige Visualisierung eines Architekturprojektes

9.2.2 Architekturvisualisierung

Ein weiterer sehr interessanter Einsatzbereich automatisch generierter 3D-Animationen ist die Visualisierung von Architekturprojekten. Gegenwärtig werden Gebäude zwar oft schon am Rechner konstruiert, so daß die gesamten Modelldaten bereits vorhanden sind, jedoch wird die Visualisierung und Präsentation des geplanten Bauwerkes in diesem Vorgang oft vernachlässigt. Da der Architekt meist auch ohne eine Computerpräsentation eine recht klare Vorstellung von seinem Entwurf hat, ist die Präsentation eher für die Kunden oder Investoren eines solchen Projektes interessant. Hier besteht ein großer Bedarf an optisch hochwertigen Präsentationen, der gegenwärtig meist durch Drittfirmen abgedeckt wird, da das manuelle Erstellen einer solchen Animation ein sehr aufwendiger Prozeß ist. Abbildung 9.2 auf Seite 134 zeigt ein Bild aus einer Architekturvisualisierung (mit freundlicher Genehmigung der Firma Lightscape). Der Autor einer solchen Visualisierung muß sämtliche Kamerapositionen manuell vorgeben, er muß das Modell ausleuchten und einen filmtechnisch sinnvollen Ablauf der Animation gestalten. Genau diese Aufgaben könnten aber in vielen Fällen weitgehend automatisch erledigt werden. Ein Skriptgenerator nach dem in der vorliegenden Arbeit beschriebenen Verfahren könnte so zumindest einen Rohentwurf der Visualisierung vorgeben, der dann interaktiv verbessert und verfeinert wird. Zur Behandlung von Architekturdomänen müßte jedoch zumindest eine wichtige Erweiterung vorgenommen werden. Die Erkennung von Kollisionen und deren Umgehung ist gerade bei Rundgängen in virtuellen Gebäuden ein sehr wichtiges Problem. Die entsprechenden geometrischen Berechnungen müßten bei der Planung von Kamerafahrten durchgeführt werden und das Verfahren zur Positionierung der Kamera (Siehe Abschnitt 5.2) müßte Kollisionen beachten und die Kameraposition entsprechend anpassen statt die störenden Objekte zu beseitigen oder durchsichtig zu machen.

9.3 Grammatikerstellung

In den bisherigen Kapiteln wurde davon ausgegangen, daß die Skriptgrammatik bei der Implementation des Systems wie ein Stück Programmcode mit einem Texteditor erstellt wird. Diese Vorgehensweise erfordert einige Gewöhnung und ist nur in einem ständigen Zyklus aus Veränderung der Grammatik und Generierung von Testanimationen sinnvoll zu bewältigen.

9.3.1 Grammatikeditor

Zur Erstellung der Skriptgrammatik wäre deshalb in manchen Fällen ein graphisches Werkzeug wünschenswert, das diesen Prozeß benutzerfreundlicher gestaltet. Man könnte beispielsweise in Anlehnung an die Skriptdarstellung im Skripteditor des Systems CATHI einen Grammatikeditor entwerfen, der einzelne Dekompositionsregeln als Anordnung der von ihnen spezifizierten Unterziele darstellt. Durch direkte Manipulation auf einem Bildschirmfenster könnten so Typen von Animationszielen definiert und zu neuen Typen zusammengesetzt werden. Bei dieser Art der Erstellung wäre es beispielsweise ohne weiteres möglich, auf die Gefahr von Endlosrekursionen in der Regelmenge hinzuweisen oder sicherzustellen, daß zu je-

dem spezifizierten Unterziel auch wieder eine Dekompositionsregel existiert. Dieser Gedanke wurde bei der Implementation des Systems CATHI nicht weiter verfolgt, da die Skriptgrammatik gewissermaßen mit den Fähigkeiten des Systems wuchs, um diese auszutesten und so mit Abschluß der Implementation bereits eine vollständige Grammatik vorhanden war. Ein Werkzeug zur leichteren Erstellung von Grammatiken ist jedoch sinnvoll, sobald weitere Grammatiken für spezialisierte Einsatzgebiete entworfen werden sollen.

9.3.2 Grammatikgenerierung

Eine letzte Idee schließlich ist, die Spezifikation der Skriptgrammatik nicht manuell vorzunehmen, sondern sie durch ein constraint- oder operatorbasiertes Planungsverfahren zu erzeugen. Das Problem der Spezifikation filmtechnischen Wissens verlagert sich hierbei auf die Definition passender Planoperatoren oder Constraints, was je nach Situation leichter sein kann als die direkte Angabe der Dekompositionsregeln. Ein solches Verfahren könnte mit wesentlich höherem Planungsaufwand zunächst einen prototypischen Plan der möglichen Animationen berechnen, der für die verwendeten Kamerapositionen beispielsweise Variablen vorsieht und somit eine allgemeine Beschreibung des Animationsablaufes darstellt, die mit verschiedenen konkreten Werten instanziiert werden kann. Eine solche Darstellung könnte dann direkt in Dekompositionsregeln überführt werden, die wiederum eine schnelle inkrementelle Generierung der konkreten Skripte ermöglicht.

Diese Idee ist eigentlich nur eine fortgeschrittene Spekulation und müßte wesentlich tiefer untersucht werden, bevor überhaupt mit Sicherheit gesagt werden kann, wie weit sie für die vorliegende Aufgabenstellung umsetzbar ist. Immerhin arbeitet das in Abschnitt 3.9 beschriebene System Player mit einem ähnlichen Verfahren. Dort wird aus einer Spezifikation möglicher Aktionen mit Vor- und Nachbedingungen ein Zustandsgraph generiert, in dem für den Übergang von jedem Zustand zu jedem anderen ein Animationsskript enthalten ist. Die Probleme bei der Generierung von 3D-Animationen sind zwar teilweise verschieden von den bei der Animation eines Interfaceagenten auftretenden, aber die in [Kurlander & Ling, 95] geschilderte Idee verdient zumindest eine nähere Untersuchung.

Epilog

Auf einem Schreibtisch liegt ein Notizblock mit Skizzen, Ideen und rot markierten Textstellen. Auf den Notizblock ist ein Scheinwerfer gerichtet, während die anderen Gegenstände seltsam vereinfacht im Halbdunkel verschwinden. Über dem Block erscheint ein roter Pfeil. Die Kamera setzt sich in Bewegung und fährt auf den Block zu, bis dieser das Bild fast ausfüllt.
Nachdem die Kamera kurze Zeit auf dem Block verharrt hat, fährt sie etwas zurück, der Pfeil verschwindet und der Scheinwerfer wandert zu einem Stapel bedruckten Papiers neben dem Notizblock. Nachdem die Kamera nahe genug herangefahren ist, kann man die Schrift auf dem zuoberst liegenden Blatt lesen:

Ein inkrementeller Ansatz zur Generierung informativer 3D-Animationen

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

vorgelegt von

Andreas Martin Butz

Vorläufige Version

. . .

Anhang A

Typen elementarer Skriptsequenzen

Grundsätzlich sind elementare Skriptsequenzen eine Liste aus einem Schlüsselwort für den Sequenztyp, drei Zeitangaben und einer Reihe von Paaren aus Schlüsselwort und Wert:

(Sequenztyp Startzeit Endzeit Dauer { Schlüsselwort Wert }*)

Der Sequenztyp ist ein Schlüsselwort aus einer der unten angegebenen Tabellen, Start- und Endzeit sowie Dauer sind Zeiten in Sekunden (Real). Nach diesen vier Angaben kommen weitere Parameter, die je nach Sequenztyp verschieden sind. Alle Parameter sind Paare aus Schlüsselwort und Wert. Die Reihenfolge kann beliebig sein. Die auftretenden Parameter und Argumenttypen sind mit kurzen Beschreibungen in den folgenden Tabellen aufgeführt.

Sequenztyp	Parameter	Wert	Beschreibung
keep-camera	transform	4x4-Matrix	abs. Kameratransformation
move-camera	from-transform	4x4-Matrix	Startmatrix der Bewegung
	to-transform	4x4-Matrix	Endmatrix
rotate-camera	axis	2 3D-Punkte	Rotationsachse der Bewegung
	angle	Winkel 0-360	Rotationswinkel
translate-camera	from-position	3D-Punkt	Startposition der Translation
	to-position	3D-Punkt	Endposition der Translation
keep-viewangle	viewangle	Winkel 0-180	Bildwinkel der Kamera
zoom-camera	from-viewangle	Winkel 0-180	Anfangsbildwinkel
	to-viewangle	Winkel 0-180	Endbildwinkel
keep-focus	focus	Real-Zahl ≥ 0	Entfernungseinstellung
focus-camera	from-focus	Real-Zahl ≥ 0	Starteinstellung
	to-focus	Real-Zahl ≥ 0	Endeinstellung
keep-f-stop	f-stop	Real-Zahl > 0	Blendenzahl des Objektivs
change-f-stop	from-f-stop	Real-Zahl > 0	Startblende des Objektivs
	to-f-stop	Real-Zahl > 0	Endblende des Objektivs

Die genannten 4x4-Matrizen geben Transformationen in homogenen Koordinaten an, die Blendenzahl ist physikalisch gesehen das Verhältnis aus Objektivbrennweite und Durchmesser der genutzten Eintrittsöffnung. Ihr sinnvoll einsetzbarer Bereich liegt etwa zwischen 1 und 32.

Sequenztyp	Parameter	Wert	Beschreibung
keep-ambientlight	color	RGB-Farbe	ambiente Lichtfarbe
adjust-ambientlight	from-color	RGB-Farbe	Anfangsfarbe
	to-color	RGB-Farbe	Endfarbe
keep-distantlight	name	String	Name des Lichtes
	color	RGB-Farbe	Farbe
	position	3D-Punkt	Lichtrichtung
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf
adjust-distantlight	name	String	Name des Lichtes
	from-color	RGB-Farbe	Anfangsfarbe
	to-color	RGB-Farbe	Endfarbe
	from-position	3D-Punkt	Anfangsrichtung
	to-position	3D-Punkt	Endrichtung
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf
keep-pointlight	name	String	Name des Lichtes
	color	RGB-Farbe	Farbe
	position	3D-Punkt	Position
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf
adjust-pointlight	name	String	Name des Lichtes
	from-color	RGB-Farbe	Anfangsfarbe
	to-color	RGB-Farbe	Endfarbe
	from-position	3D-Punkt	Anfangsposition
	to-position	3D-Punkt	Endposition
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf
keep-spotlight	name	String	Name des Lichtes
	color	RGB-Farbe	Farbe
	position	3D-Punkt	Position
	aimpoint	3D-Punkt	Zielpunkt
	coneangle	Winkel 0-180	Kern des Leuchtkegels
	conedeltaangle	Winkel 0-180	Randbereich
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf
adjust-spotlight	name	String	Name des Lichtes
	from-color	RGB-Farbe	Anfangsfarbe
	to-color	RGB-Farbe	Endfarbe
	from-position	3D-Punkt	Anfangsposition
	to-position	3D-Punkt	Endposition
	from-aimpoint	3D-Punkt	Anfangszielpunkt
	to-aimpoint	3D-Punkt	Endzielpunkt
	from-coneangle	Winkel 0-180	Anfangswert Kern
	to-coneangle	Winkel 0-180	Endwert Kern
	from-conedeltaangle	Winkel 0-180	Anfangswert Randber.
	to-conedeltaangle	Winkel 0-180	Endwert Randbereich
	refframe	camera/world	Referenzsystem
	shadows	Ja/Nein	Schattenwurf

Sequenztyp	Parameter	Wert	Beschreibung
rotate-object	object	String	Name eines 3D-Objekts
	axis	Punkt+Vektor	Rotationsachse
	startangle	Winkel 0-360	Startwinkel der Rotation
	endangle	Winkel 0-360	Endwinkel der Rotation
translate-object	object	String	Name eines 3D-Objekts
	startposition	3D-Punkt	Startposition der Translation
	endposition	3D-Punkt	Endposition der Translation
visibility	level	Wert 0-1	0 = unsichtbar, 1 = sichtbar
	object	String	Name eines 3D-Objekts
level-of-detail	level	Wert 0-1	0 = abstrahiert, 1 = detailliert
	object	String	Name eines 3D-Objekts
color	color	RGB-Farbe	Farbe eines Objekts
	object	String	Name eines 3D-Objekts
create-arrow	name	String	Name des Pfeils
	color	RGB-Farbe	Farbe des Pfeils
	aimpoint	3D-Punkt	Zielpunkt des Pfeils
	startpoint	3D-Punkt	Anfangspunkt
destroy-arrow	name	String	Name des Pfeils
increment	scripttime	Zeit in Sek.	Skriptzeit an dieser Stelle
	generationtime	Zeit in Sek.	Gen.zeit an dieser Stelle
nil	-	-	die leere Sequenz

Der Sequenztyp *increment* stellt einen Sonderfall dar. Er beschreibt keine Veränderung in der 3D-Welt, sondern gibt an, daß das Skript bis zu dieser Stelle eine abgeschlossene Einheit enthält, mit deren Ausgabe begonnen werden kann. Die leere Sequenz hat keine Auswirkung für die Animation und wird ignoriert.

Anhang B

Auflistung der Skriptgrammatik

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The leaves of the script tree: rules that return elementary sequences
;;; This is the 'lexicon' of the generator, while the 'syntax' of the scripts
;;; is defined by the set of non-elementary rules.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Visibility, Color and Level of detail
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; invisible means transparent if possible or removed otherwise
(defrule visibility ((level 1) (objects (objects-between-camera-and-aimpoint)))
  (when (and objects (feature :invisible))
        `(:visibility ,start-time ,end-time ,duration
                      :level ,level :objects ,(base-objects objects))))

;;; color (1 1 1) = white, (0 0 0) = black
(defrule color ((color '(1 1 1)) (objects nil))
  (when (and objects (feature :color))
        `(:color ,start-time ,end-time ,duration
                 :color ,color :objects ,(base-objects objects))))

;;; level of detail means: 1.0 = full detail, 0.0 = maximum abstraction
(defrule level-of-detail ((level 1) (objects (all-scene-objects)))
  (when (and objects (feature :abstract))
        `(:level-of-detail ,start-time ,end-time ,duration
                           :level ,level :objects ,(base-objects objects))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Camera control
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule keep-camera ((transform *current-camera-transform*))
  `(:keep-camera ,start-time ,end-time ,duration :transform ,transform))

(defrule keep-viewangle ((viewangle *current-viewangle*))
  (when (feature :viewangle)
    `(:keep-viewangle ,start-time ,end-time ,duration :viewangle ,viewangle)))

;;; focus in its optical meaning...
(defrule keep-focus ((focus *current-focus*))
  (when (feature :focus)
    `(:keep-focus ,start-time ,end-time ,duration :focus ,focus)))
```

```
(defrule keep-f-stop ((f-stop *current-f-stop*))
  (when (feature :f-stop)
    `(:keep-f-stop ,start-time ,end-time ,duration :f-stop ,f-stop)))

;;; the old move-camera works with geomview, but not with RenderMan..
(defrule move-camera-with-matrix
  ((from-transform *old-camera-transform*)
   (to-transform *current-camera-transform*))
  (setq *current-camera-transform* to-transform)
  `(:move-camera ,start-time ,end-time ,duration
                 :from-transform ,from-transform
                 :to-transform   ,to-transform ))

(defrule move-camera-with-rot-trans
  ((from-transform *old-camera-transform*)
   (to-transform *current-camera-transform*))
  (setq *current-camera-transform* to-transform)
  (let* ((m0 from-transform)
         (frompos (gx::multiply-4x4-matrix-vector m0 (gx::make-3d-vector) nil))
         (m0r (gx::matrix-rotation m0))
         (m0ri (gx::transpose-4x4-matrix m0r nil))
         (m0t (gx::matrix-translation m0))
         (m1 to-transform)
         (topos (gx::multiply-4x4-matrix-vector m1 (gx::make-3d-vector) nil))
         (m1r (gx::matrix-rotation m1))
         (m1t (gx::matrix-translation m1))
         (dr (gx::multiply-4x4-matrices m1r m0ri nil))
         (ax (gx::matrix-rotation-axis dr))
         (axis (gx::make-axis :point1 (gx::make-3d-point)
                              :point2 ax))
         (p1 (gx::make-3d-vector))
         (p2 (gx::orthogonal-vector ax nil))
         (p3 (gx::multiply-4x4-matrix-vector dr p2 nil))
         (w  (gx::angle-between-vectors p2 p3))
         helpvector)

;;; get the local translation
    (setq helpvector (gx::multiply-4x4-matrix-vector m0t p1 nil))
    (setq p1 (gx::multiply-4x4-matrix-vector m1t p1 p1))
    (setq p1 (gx::vector-sub p1 helpvector p1))

    (if (or (zerop w) (gx::zero-vector-p ax))
        `(:translate-camera ,start-time ,end-time ,duration
                            :from-position ,frompos
                            :to-position ,topos)
      `(:parallel
        (:rotate-camera ,start-time ,end-time ,duration
                        :axis ,axis
                        :angle ,w)
        (:translate-camera ,start-time ,end-time ,duration
                            :from-position ,frompos
                            :to-position ,topos)))))
```

```
(defrule move-camera
    ((from-transform *old-camera-transform*)
     (to-transform *current-camera-transform*))
  (if (feature :genrel)
      (move-camera-with-rot-trans
       :from-transform from-transform
       :to-transform to-transform
       :duration duration)
    (move-camera-with-matrix
     :from-transform from-transform
     :to-transform to-transform
     :duration duration)))

(defrule zoom-camera
    ((from-viewangle *old-viewangle*) (to-viewangle *current-viewangle*))
  (setq *current-viewangle* to-viewangle)
  (when (feature :viewangle)
    (if (same from-viewangle to-viewangle)
        (keep-viewangle :duration duration)
      `(:zoom-camera ,start-time ,end-time ,duration
                     :from-viewangle ,from-viewangle
                     :to-viewangle   ,to-viewangle))))

;;; focus in its optical meaning...
(defrule focus-camera
    ((from-focus *old-focus*) (to-focus *current-focus*))
  (setq *current-focus* to-focus)
  (when (feature :focus)
    (if (same from-focus to-focus)
        (keep-focus :duration duration)
      `(:focus-camera ,start-time ,end-time ,duration
                      :from-focus ,from-focus
                      :to-focus   ,to-focus))))

(defrule change-f-stop
  ((from-f-stop *old-f-stop*) (to-f-stop *current-f-stop*))
  (setq *current-f-stop* to-f-stop)
  (when (feature :f-stop)
        (if (same from-f-stop to-f-stop)
            (keep-f-stop :duration duration)
          `(:change-f-stop ,start-time ,end-time ,duration
                           :from-f-stop ,from-f-stop
                           :to-f-stop   ,to-f-stop))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; light control
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule keep-light (light)
  (when (feature :light)
        (case (type-of light)
              ((gx::ambient-light)
               (when (feature :ambient-light)
                     `(:keep-ambientlight
                       ,start-time ,end-time ,duration
                       :name ,(gx::object-id (gx::object-name light))
                       :color,(gx::light-color light))))
              ((gx::distant-light)
               (when (feature :distant-light)
                     `(:keep-distantlight
                       ,start-time ,end-time ,duration
                       :name     ,(gx::object-id (gx::object-name light))
                       :color    ,(gx::light-color    light)
                       :position ,(gx::light-position light)
                       :refframe ,(gx::light-refframe light)
                       :shadows  ,(gx::light-shadows  light))))
              ((gx::point-light)
               (when (feature :point-light)
                     `(:keep-pointlight
                       ,start-time ,end-time ,duration
                       :name     ,(gx::object-id (gx::object-name light))
                       :color    ,(gx::light-color    light)
                       :position ,(gx::light-position light)
                       :refframe ,(gx::light-refframe light)
                       :shadows  ,(gx::light-shadows  light))))
              ((gx::spot-light)
               (when (feature :spot-light)
                     `(:keep-spotlight
                       ,start-time ,end-time ,duration
                       :name     ,(gx::object-id (gx::object-name light))
                       :color    ,(gx::light-color    light)
                       :position ,(gx::light-position light)
                       :aimpoint ,(gx::light-aimpoint light)
                       :coneangle ,(+ (gx::light-cone    light)
                                      (gx::light-falloff light))
                       :conedeltaangle ,(gx::light-cone  light)
                       :refframe ,(gx::light-refframe light)
                       :shadows  ,(gx::light-shadows  light))))
              ((gx::area-light)
               (when (feature :area-light)
                     `(:keep-arealight
                       ,start-time ,end-time ,duration
                       :name     ,(gx::object-id (gx::object-name light))
                       :color    ,(gx::light-color    light)
                       :geometry ,(gx::object-name (gx::light-geometry light))
                       :refframe ,(gx::light-refframe light)
                       :shadows  ,(gx::light-shadows  light)))))))

(defrule adjust-light (light)
  (when (feature :light)
        (case (type-of light)
              ((gx::ambient-light)
               (when (feature :ambient-light)
                     (prog1
                         `(:adjust-ambientlight
```

```
              ,start-time ,end-time ,duration
              :name     ,(gx::object-id (gx::object-name light))
              :from-color  ,(gx::light-old-color light)
              :to-color    ,(gx::light-color light))
          (setf (gx::light-old-color light)
                (gx::light-color light)))))
((gx::distant-light)
 (when (feature :distant-light)
       (prog1
           `(:adjust-distantlight
             ,start-time ,end-time ,duration
             :name     ,(gx::object-id (gx::object-name light))
             :from-color  ,(gx::light-old-color light)
             :to-color    ,(gx::light-color light)
             :from-position ,(gx::light-old-position light)
             :to-position   ,(gx::light-position light)
             :refframe ,(gx::light-refframe light)
             :shadows  ,(gx::light-shadows  light))
         (setf (gx::light-old-position light)
               (gx::light-position light))
         (setf (gx::light-old-color light)
               (gx::light-color light)))))
((gx::point-light)
 (when (feature :point-light)
       (prog1
           `(:adjust-pointlight
             ,start-time ,end-time ,duration
             :name     ,(gx::object-id (gx::object-name light))
             :from-color  ,(gx::light-old-color light)
             :to-color    ,(gx::light-color light)
             :from-position ,(gx::light-old-position light)
             :to-position   ,(gx::light-position light)
             :refframe ,(gx::light-refframe light)
             :shadows  ,(gx::light-shadows  light))
         (setf (gx::light-old-position light)
               (gx::light-position light))
         (setf (gx::light-old-color light)
               (gx::light-color light)))))
((gx::spot-light)
 (when (feature :spot-light)
       (prog1
           `(:adjust-spotlight
             ,start-time ,end-time ,duration
             :name     ,(gx::object-id (gx::object-name light))
             :from-color  ,(gx::light-old-color light)
             :to-color    ,(gx::light-color light)
             :from-position ,(gx::light-old-position light)
             :to-position   ,(gx::light-position light)
             :from-aimpoint ,(gx::light-old-aimpoint light)
             :to-aimpoint   ,(gx::light-aimpoint light)
             :from-coneangle ,(+ (gx::light-old-cone    light)
                                 (gx::light-old-falloff light))
             :to-coneangle ,(+ (gx::light-cone    light)
                               (gx::light-falloff light))
             :from-conedeltaangle ,(gx::light-old-cone  light)
             :to-conedeltaangle   ,(gx::light-cone  light)
             :refframe ,(gx::light-refframe light)
             :shadows  ,(gx::light-shadows  light))
```

```
                    (setf (gx::light-old-position light)
                          (gx::light-position light))
                    (setf (gx::light-old-aimpoint light)
                          (gx::light-aimpoint light))
                    (setf (gx::light-old-cone light)
                          (gx::light-cone light))
                    (setf (gx::light-old-falloff light)
                          (gx::light-falloff light))
                    (setf (gx::light-old-color light)
                          (gx::light-color light)))))
            ((gx::area-light)
             (when (feature :area-light)
                   (prog1
                       `(:adjust-arealight
                         ,start-time ,end-time ,duration
                         :name     ,(gx::object-id (gx::object-name light))
                         :from-color  ,(gx::light-old-color light)
                         :to-color    ,(gx::light-color light)
                         :geometry ,(gx::object-name
                                     (gx::light-geometry light))
                         :refframe ,(gx::light-refframe light)
                         :shadows  ,(gx::light-shadows  light))
                       (setf (gx::light-old-color light)
                             (gx::light-color light))))))))

(defrule keep-lights ((lights *current-lights*))
  (when (and lights (feature :light))
    (parallel
      (loop for l in lights collect
            (keep-light :light l :duration duration)))))

(defrule adjust-lights ((lights *current-lights*))
  (when (and lights (feature :light))
    (parallel
      (loop for l in lights collect
            (adjust-light :light l :duration duration)))))


(defrule create-arrow (arrow)
  (when (and arrow (feature :meta))
    `(:create-arrow ,start-time ,end-time ,duration
                    :name ,(gx::object-name arrow)
                    :color ,(gx::arrow-color arrow)
                    :aimpoint ,(gx::arrow-aimpoint arrow)
                    :startpoint ,(gx::arrow-startpoint arrow))))

(defrule destroy-arrow (arrow)
  (when (and arrow (feature :meta) (member arrow *current-metagraphics*))
    (setq *current-metagraphics*
      (remove arrow *current-metagraphics*))
    `(:destroy-arrow ,start-time ,end-time ,duration
                    :name ,(gx::object-name arrow))))

(defrule destroy-spot (spot)
  (when (and spot (member spot *current-lights*))
    (setq *current-lights*
      (remove spot *current-lights*))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Object motions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; move-object needs an object and a motion description and applies the
;;; specified part of the motion in the given time

(defrule move-object (object motion (from 0.0) (to 1.0))
  (let ((motion (gx::motion-for-object motion object)))
    (cond ((typep motion 'gx::object-rotation)
           (let ((fromangle (+ (gx::rotation-startangle motion)
                               (* from (- (gx::rotation-endangle motion)
                                          (gx::rotation-startangle motion)))))
                 (toangle (+ (gx::rotation-startangle motion)
                             (* to (- (gx::rotation-endangle motion)
                                      (gx::rotation-startangle motion))))))
             `(:rotate-object ,start-time ,end-time ,duration
                              :object ,(gx::object-id object)
                              :axis ,(gx::rotation-axis motion)
                              :startangle ,fromangle
                              :endangle ,toangle)))
          ((typep motion 'gx::object-translation)
           (let ((frompos (gx::vector-add
                           (gx::translation-startvector motion)
                           (gx::scalar-mult
                            from (gx::vector-sub
                                  (gx::translation-endvector motion)
                                  (gx::translation-startvector motion)
                                  nil)
                            nil)
                           nil))
                 (topos (gx::vector-add
                         (gx::translation-startvector motion)
                         (gx::scalar-mult
                          to (gx::vector-sub
                              (gx::translation-endvector motion)
                              (gx::translation-startvector motion)
                              nil)
                          nil)
                         nil)))
             `(:translate-object ,start-time ,end-time ,duration
                                 :object ,(gx::object-id object)
                                 :startposition ,frompos
                                 :endposition ,topos)))
          (t nil))))

(defrule update-position (object)
  (let ((position (gx::copy-coordinates (gx::object-center object) nil)))
    `(:translate-object ,start-time ,end-time ,duration
                        :object ,(gx::object-id object)
                        :startposition ,position
                        :endposition ,position)))

(defrule update-positions ((objects (all-scene-objects)))
  (parallel
    (loop for o in objects collect
          (update-position :object o :duration duration))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Object explosions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule explosion-script (explosion)
  (when explosion
        (let* ((movements (gx::combine-explosion explosion))
               (objects (mapcar #'car movements))
               startpos endpos)
          (setq startpos (loop for o in objects collect
                               (list o (gx::copy-coordinates
                                        (gx::object-center o) nil))))
          (gx::do-explosion :explosion movements)
          (setq endpos (loop for o in objects collect
                             (list o (gx::copy-coordinates
                                      (gx::object-center o) nil))))
          (parallel
                (loop for o in objects collect
                      `(:translate-object ,start-time ,end-time ,duration
                                          :object ,(gx::object-id o)
                                          :startposition
                                          ,(second (assoc o startpos))
                                          :endposition
                                          ,(second (assoc o endpos))))))))

(defrule separate-object (object)
  (explosion-script :explosion (gx::separate-object object)
                    :duration duration))

(defrule isolate-object (object)
  (explosion-script :explosion (gx::isolate-object object)
                    :duration duration))

(defrule explode-object (object)
  (explosion-script :explosion (gx::explode-object object)
                    :duration duration))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The nodes of the script tree: rules that return sets of subsequences.
;;; This is the 'grammar' of the generator.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some basics:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule steady-camera
  ((transform *current-camera-transform*)
   (focus     *current-focus*)
   (f-stop    *current-f-stop*)
   (viewangle *current-viewangle*))
  (parallel
   (keep-camera :duration duration :transform transform)
   (keep-focus     :duration duration :focus focus)
   (keep-f-stop     :duration duration :f-stop f-stop)
   (keep-viewangle :duration duration :viewangle viewangle)))

(defrule update-camera ()
  (parallel
   (move-camera :duration duration)
   (focus-camera :duration duration)
   (change-f-stop :duration duration)
   (zoom-camera :duration duration)))

(defrule steady-lights ((lights *current-lights*))
  (keep-lights :duration duration :lights lights))

(defrule update-lights ((lights *current-lights*))
  (adjust-lights :duration duration :lights lights))

(defrule steady-shot ((lights *current-lights*))
  (parallel
   (steady-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule update-camera-steady-lights ((lights *current-lights*))
  (parallel
   (update-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule update-lights-steady-camera ((lights *current-lights*))
  (parallel
   (steady-camera :duration duration)
   (update-lights :duration duration :lights lights)))

(defrule update-camera-and-lights  ((lights *current-lights*))
  (parallel
   (update-camera :duration duration)
   (update-lights :duration duration :lights lights)))

(defrule move-object-steady-shot
  (object motion (from 0.0) (to 1.0) (lights *current-lights*))
  (parallel
   (move-object :object object :motion motion :duration duration
                :from from :to to)
```

```
   (apply-motion object motion 0.0)
   (steady-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule move-object-update-camera-and-lights
  (object motion (from 0.0) (to 1.0) (lights *current-lights*))
  (parallel
   (move-object :object object :motion motion :duration duration
                :from from :to to)
   (update-camera :duration duration)
   (update-lights :duration duration :lights lights)))

(defrule move-object-following
  (object motion (from 0.0) (to 1.0) (fill 0.7) (lights *current-lights*))
  (apply-motion object motion from)
  (look-at object
            :direction (viewing-direction-for-object-motion motion object)
            :upvector *current-upvector*
            :fill fill)
  (apply-motion object motion to)
  (look-at object
            :direction (viewing-direction-for-object-motion motion object)
            :upvector *current-upvector*
            :fill fill)
  (parallel
   (move-object :object object :motion motion :duration duration
                :from from :to to)
   (update-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule move-object-overtaking
  (object motion (from 0.0) (to 1.0) (fill 0.7) (lights *current-lights*))
  (apply-motion object motion to)
  (look-at object
            :direction (viewing-direction-for-object-motion motion object)
            :upvector *current-upvector*
            :fill fill)
  (parallel
   (move-object :object object :motion motion :duration duration
                :from from :to to)
   (update-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule visibility-steady-shot
  ((level 1.0) (objects (objects-between-camera-and-aimpoint))
   (lights *current-lights*))
  (parallel
   (visibility :level level :objects objects :duration duration)
   (steady-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule visibility-update-camera-and-lights
  ((level 1.0) (objects (objects-between-camera-and-aimpoint))
   (lights *current-lights*))
  (parallel
   (visibility :level level :objects objects :duration duration)
   (update-camera :duration duration)
   (update-lights :duration duration :lights lights)))
```

```
(defrule color-steady-shot
  ((color '(1 1 1)) (objects (objects-between-camera-and-aimpoint))
   (lights *current-lights*))
  (parallel
   (color :color color :objects objects :duration duration)
   (steady-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule level-of-detail-steady-shot
  ((level 1.0) (objects (objects-between-camera-and-aimpoint))
   (lights *current-lights*))
  (parallel
   (level-of-detail :level level :objects objects :duration duration)
   (steady-camera :duration duration)
   (steady-lights :duration duration :lights lights)))

(defrule flash-object-once (object color)
  (cond ((and (feature :usecolor) (feature :color))
         (let ((co (gx::object-color object))
               (cc (or color
                       (contrasting-color (gx::object-color object)))))
           (sequential
            (color :objects object :color cc :duration (/ duration 2))
            (color :objects object :color co :duration (/ duration 2)))))
        ((and (feature :useinvisible) (feature :invisible))
         (sequential
          (visibility :objects object :level 0 :duration (/ duration 2))
          (visibility :objects object :level 1 :duration (/ duration 2))))
        (t (steady-shot :duration duration))))

(defrule flash-object (object color times)
  (sequential2
   (loop for i from 1 to times collect
         (flash-object-once :object object :color color
                            :duration (/ duration times)))))

(defrule flash-objects (objects color times)
  (unless times (setq times (max (round duration) 3)))
  (parallel
        (loop for o in (base-objects objects) collect
                  (flash-object :object o :color color
                                :times times :duration duration))))

(defrule flash-objects-steady-shot (objects color times)
  (parallel
   (flash-objects :objects objects :color color
                  :times times :duration duration)
   (steady-shot :duration duration)))

(defrule restore-color (objects)
  (parallel
        (loop for o in (base-objects objects) collect
                  (color :objects o :color (gx::object-color o)
                         :duration duration))))
```

```
(defrule restore-color-steady-shot (objects)
  (parallel
   (restore-color :objects objects :duration duration)
   (steady-shot :duration duration)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some rules to initialize the world state:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule reset-world-state ()
  (parallel
   (level-of-detail
    :level 1 :duration duration :objects (all-scene-objects))
   (visibility
    :level 1 :duration duration :objects (all-scene-objects))
   (let ((objects (gx::exploded-objects)))
     (gx::undo-explosion)
     (update-positions :objects objects :duration duration))))

(defrule init-scene ()
  (setq *current-lights* nil)
  (steady-shot :duration duration))

(defrule init-lights ()
  (and (feature :light) (zerop *current-time*) (set-up-basic-lights)
       (keep-lights :duration duration)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some rules to guide the visual focus
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; focus in its attentional meaning...
(defrule focus-with-abstraction (object)
  (when (feature :useabstract)
    (parallel
     (level-of-detail :objects (all-objects-but object)
                      :level 0 :duration duration)
     (level-of-detail :objects object
                      :level 1 :duration duration))))

;;; focus in its attentional meaning...
(defrule focus-with-spotlight (object)
  (when (and (feature :uselight) (feature :spot-light))
    (progn (dim-lights 0.7)
           (set-spotlight-on object)
           (unless (zerop duration)
             (update-lights-steady-camera :duration duration)))))

;;; focus in its attentional meaning...
(defrule focus-with-arrow (object)
  (when (and (feature :usemeta) (feature :meta))
    (create-arrow :arrow (set-arrow-on object) :duration duration)))

(defrule focus-with-abstraction-and-arrow (object)
  (parallel
   (focus-with-abstraction :object object :duration duration)
   (focus-with-arrow :object object :duration duration)))
```

```
(defrule focus-by-all-means (object (arrow t))
  (parallel
   (focus-with-abstraction :object object :duration duration)
   (when arrow (focus-with-arrow :object object :duration duration))
   (focus-with-spotlight :object object :duration duration)))

(defrule remove-arrow (object)
  (when (feature :meta)
    (destroy-arrow :arrow (arrow-for-object object) :duration duration)))

(defrule remove-spot (object)
  (sequential
   (dim-lights 1.4)
   (destroy-spot :spot (spotlight-for-object object) :duration duration)
   (unless (zerop duration)
     (update-lights-steady-camera :duration duration))))

(defrule unfocus-by-all-means (object)
  (parallel
   (level-of-detail :objects (all-objects-but object)
                    :level 1 :duration duration)
   (remove-arrow :object object :duration duration)
   (remove-spot :object object :duration duration)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some rules to deal with disturbing objects
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule remove-disturbing-objects (object)
  (when (feature :useinvisible)
    (parallel
     (visibility
      :level (if (feature :uselight) 0 0.2)
      :duration duration :objects
      (union (objects-between-camera-and-aimpoint)
             (all-objects-roughly-including object)
             :test #'equal))
     (visibility :level 1 :duration duration
                 :objects object))))

(defrule remove-all-other-objects (object)
  (when (feature :useinvisible)
    (parallel
     (visibility
      :level (if (feature :uselight) 0 0.2)
      :duration duration :objects (all-objects-but object))
     (visibility :level 1 :duration duration :objects object))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The start shot for each sequence:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule start-shot (object)
  (if (zerop *current-time*)
      (incremental
       (sequential
        (reset-world-state :duration 0)
        (look-at :world :side '(:front :front :front :up))
```

```
        (setq *current-lights* nil)
        (when (feature :useabstract)
          (focus-with-abstraction :object object :duration 0))
        (when (feature :light) (steady-shot :duration 0)))
       (when (feature :light) (set-up-basic-lights))
       (steady-shot :duration duration))
    (sequential
     (when (feature :useinvisible)
       (visibility :level 1 :duration 0 :objects object))
     (steady-shot :duration duration))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Introduce an object:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule zoom-from-here-to-object (object (fill 0.9) (arrow t))
  (incremental
   (sequential
    (look-at object :side :front :fill fill)
    (remove-disturbing-objects :object object :duration 0)
    (focus-by-all-means :object object :duration 0 :arrow arrow))
   (update-camera-and-lights :duration duration)
   (remove-arrow :object object :duration 0)))

(defrule turn-and-zoom-from-here-to-object (object (fill 0.9) (arrow t))
  (incremental
   (look-at (toplevel-object object)
            :direction *current-direction*
            :upvector *current-upvector*)
   (update-camera-steady-lights :duration (* 0.2 duration))
   (look-at object :side :front :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.4 duration))
   (zoom-from-here-to-object
    :object object :fill fill :arrow arrow :duration (* 0.4 duration))))

(defrule cut-and-zoom-from-here-to-object (object (fill 0.9) (arrow t))
  (incremental
   (look-at (toplevel-object object)
            :direction (object-direction object :front)
            :upvector (object-direction object :up) )
   (steady-shot :duration (* 0.3 duration))
   (zoom-from-here-to-object
    :object object :fill fill :arrow arrow :duration (* 0.7 duration))))

(defrule go-from-here-to-object (object (fill 0.9) (arrow t))
  (cond ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 1.0)
              (< (gx::angle-between-vectors
                  (object-direction object :up)
                  *current-upvector*) 2.0))
         (zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))
        ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 2.0)
              (< (gx::angle-between-vectors
                  (object-direction object :up)
```

```
                *current-upvector*) 2.0))
         (turn-and-zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))
        (t
         (cut-and-zoom-from-here-to-object
          :object object :fill fill :arrow arrow :duration duration))))

(defrule fly-around-object-16 (object)
  (incremental
   (look-at object :side '(:right :front :front :front))
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:front :right) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:front :right :right :right) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side :right :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))

   (look-at object :side '(:back :right :right :right) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:right :back) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:right :back :back :back) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side :back :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))

   (look-at object :side '(:back :back :back :left) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:back :left) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:back :left :left :left) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side :left :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))

   (look-at object :side '(:left :left :left :front) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:left :front) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side '(:left :front :front :front) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))
   (look-at object :side :front :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.0625 duration))))

(defrule fly-around-object-8 (object)
  (incremental
   (look-at object :side '(:front :right) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side :right :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side '(:right :back) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side :back :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side '(:back :left) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side :left :distance *current-focus*)
```

```
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side '(:left :front) :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))
   (look-at object :side :front :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.125 duration))))

(defrule fly-around-object-4 (object)
  (incremental
   (look-at object :side :right :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.25 duration))
   (look-at object :side :back :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.25 duration))
   (look-at object :side :left  :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.25 duration))
   (look-at object :side :front :distance *current-focus*)
   (update-camera-steady-lights :duration (* 0.25 duration))))

(defrule fly-around-object (object)
  (cond ((time-enough 2)
         (fly-around-object-16 :object object :duration duration))
        ((time-enough 1)
         (fly-around-object-8 :object object :duration duration))
        (t
         (fly-around-object-4 :object object :duration duration))))

(defrule localize-toplevel-object (object)
  (incremental
   (go-from-here-to-object :object object :duration (* 0.3 duration))
   (remove-all-other-objects :object object :duration 0)
   (fly-around-object :object object :duration (* 0.7 duration))))

(defrule localize-regular-object (object)
  (incremental
   (go-from-here-to-object :object object :duration (* 0.7 duration))
   (flash-objects-steady-shot :objects object
                              :duration (* 0.3 duration))))

(defrule localize-object (object)
  (incremental
   (start-shot :object object :duration (* 0.0 duration))
   (cond ((same (parent-object object) :world)
          (localize-toplevel-object :object object :duration (* 0.9 duration)))
         ((and (feature :useexplosion) (gx::isolatable object)
               (time-enough (+ 0.5 (* 0.1 (number-of-parts object)))))
          (localize-object-isolated :object object :duration (* 0.9 duration)))
         ((or (and (feature :invisible) (feature :useinvisible))
              (and (feature :color) (feature :usecolor)))
          (localize-regular-object :object object :duration (* 0.9 duration)))
         (t
          (go-from-here-to-object :object object :duration (* 0.9 duration))))
   (steady-shot :duration (* 0.1 duration))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; show the exploded view of object groups
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule separate-object-steady-shot (object)
  (incremental
   (parallel
    (separate-object :object object :duration duration)
    (steady-shot :duration duration))))

(defrule isolate-object-steady-shot (object)
  (incremental
   (parallel
    (isolate-object :object object :duration duration)
    (steady-shot :duration duration))))

(defrule explode-object-steady-shot (object)
  (incremental
   (parallel
    (explode-object :object object :duration duration)
    (steady-shot :duration duration))))

(defrule isolate-object-update-camera (object)
  (incremental
   (parallel
    (isolate-object :object object :duration duration)
    (look-at object)
    (update-camera-steady-lights :duration duration))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; localization using explosion
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule localize-object-isolated (object)
  (if (or (feature :invisible) (feature :color))
      (incremental
       (focus-by-all-means :object object :duration 0 :arrow nil)
       (isolate-object-update-camera :object object
                                     :duration (* 0.7 duration))
       (flash-objects-steady-shot :objects object ;:color '(0 1 0)
                                  :duration (* 0.3 duration)))
    (incremental
     (focus-by-all-means :object object :duration 0 :arrow nil)
     (isolate-object-update-camera :object object
                                  :duration (* 1.0 duration)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Show object motions:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule zoom-from-here-to-object-motion (object motion)
  (incremental
   (let (d1 d2 d3
         (tr *current-camera-transform*)
         (dir *current-direction*)
         (dist *current-focus*)
         (up  *current-upvector*))
     (apply-motion object motion 0.5)
```

```
    (look-at object :direction dir :upvector up :fill 0.5)
    (setq d1 *current-focus*)
    (apply-motion object motion 1.0)
    (look-at object :direction dir :upvector up :fill 0.5)
    (setq d2 *current-focus*)
    (apply-motion object motion 0.0)
    (look-at object :direction dir :upvector up :fill 0.5)
    (setq d3 *current-focus*)
    (apply-motion object motion 0.5)
    (look-at object :upvector up
             :distance (max d1 d2 d3))
    (apply-motion object motion 0.0)
    (setq *old-camera-transform* tr)
    (setq *old-focus* dist)
    (setq *old-direction* dir)
    (setq *old-upvector* up)
    nil)
   (parallel
    (focus-with-abstraction :object object :duration 0)
    (focus-with-arrow :object object :duration 0))
   (update-camera-steady-lights :duration duration)
   (remove-arrow :object object :duration 0)))

(defrule turn-and-zoom-from-here-to-object-motion (object motion)
  (incremental
   (look-at (list object :world) :side :front
            :upvector (object-direction :world :up))
   (update-camera-steady-lights :duration (* 0.5 duration))
   (zoom-from-here-to-object-motion
    :object object :motion motion :duration (* 0.5 duration))))

(defrule cut-and-zoom-from-here-to-object-motion (object motion)
  (incremental
   (look-at (toplevel-object object)
            :direction (object-direction object :front)
            :upvector (object-direction :world :up))
   (steady-shot :duration (* 0.3 duration))
   (zoom-from-here-to-object-motion
    :object object :motion motion :duration (* 0.7 duration))))

(defrule go-from-here-to-object-motion (object motion)
  (cond ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 1.0)
              (< (gx::angle-between-vectors
                  (object-direction :world :up)
                  *current-upvector*) 2.0))
         (zoom-from-here-to-object-motion
          :object object :motion motion :duration duration))
        ((and (< (gx::angle-between-vectors
                  (object-direction object '(:front :front :up))
                  *current-direction*) 2.0)
              (< (gx::angle-between-vectors
                  (object-direction :world :up)
                  *current-upvector*) 2.0))
         (turn-and-zoom-from-here-to-object-motion
          :object object :motion motion :duration duration))
```

```
        (t
         (cut-and-zoom-from-here-to-object-motion
          :object object :motion motion :duration duration))))

(defrule show-object-motion-1 (object motion)
  (incremental
   (apply-motion object motion 0.0)
   (start-shot :object object :duration (* 0.0 duration))
   (move-object-steady-shot :object object :motion motion
                            :to 0.0 :duration 0)
   (go-from-here-to-object-motion :object object :motion motion
                                  :duration (* 0.5 duration))
   (move-object-steady-shot :object object :motion motion
                            :duration (* 0.5 duration))
   (apply-motion object motion 1)))

(defrule show-object-motion-2 (object motion)
  (incremental
   (apply-motion object motion 0.0)
   (start-shot :object object :duration (* 0.0 duration))
   (move-object-steady-shot :object object :motion motion
                            :to 0.0 :duration 0)
   (look-at object
            :direction (viewing-direction-for-object-motion motion object)
            :upvector *current-upvector*
            :fill 0.5)
   (focus-with-abstraction-and-arrow :object object :duration 0)
   (update-camera-steady-lights :duration (* 0.5 duration))
   (remove-arrow :object object :duration 0)
   (move-object-steady-shot :object object :motion motion
                            :duration (* 0.5 duration))
   (apply-motion object motion 1)))

(defrule show-object-motion-f (object motion)
  (incremental
   (apply-motion object motion 0.0)
   (start-shot :object object :duration 0.0)
   (move-object-steady-shot :object object :motion motion
                            :to 0.0 :duration 0)
   (apply-motion object motion 0.1)
   (look-at object
            :direction (viewing-direction-for-object-motion motion object)
            :upvector *current-upvector*
            :fill 0.7)
   (focus-with-abstraction-and-arrow :object object :duration 0)
   (update-camera-steady-lights :duration (* 0.3 duration))
   (remove-arrow :object object :duration 0)
   (move-object-steady-shot :object object :motion motion
                            :duration (* 0.2 duration)
                            :from 0.0 :to 0.3)
   (move-object-overtaking :object object :motion motion
                           :duration (* 0.3 duration)
                           :from 0.3 :to 0.8 :fill 0.7)
   (move-object-following :object object :motion motion
                          :duration (* 0.2 duration)
                          :from 0.8 :to 1.0 :fill 0.7)
   (apply-motion object motion 1)))
```

```
(defrule show-object-motion (object motion)
  (cond ((translationp motion object)
         (show-object-motion-f
          :object object
          :motion motion
          :duration duration))
        ((> (gx::angle-between-vectors
             (viewing-direction-for-object-motion motion object)
             (object-direction object :front)) 1.0)
         (show-object-motion-2
          :object object
          :motion motion
          :duration duration))
        (t
         (show-object-motion-1
          :object object
          :motion motion
          :duration duration))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; show the spatial relation between two objects
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule focus-dof-from-object-to-object (object1 object2 (f-stop 1.0))
  (setq *old-f-stop* 32)
  (setq *current-f-stop* f-stop)
  (look-at (list object1 object2) :side :front)
  (look-at (list object1 object2) :side :front)
  (setq *current-focus* (distance-from-camera-to-object object1))
  (setq *old-focus* *current-focus*)
  (incremental
   (sequential (reset-world-state :duration 0)
               (init-scene :duration 0))
   (init-lights :duration 0)
   (if (feature :useabstract)
       (level-of-detail-steady-shot
        :objects (all-objects-but (list object1 object2))
        :level 0 :duration 0))
   (update-camera-steady-lights :duration (* 0.2 duration))
   (and (setq *old-focus* *current-focus*)
        (setq *current-focus* (distance-from-camera-to-object object2))
        (setq *old-f-stop* f-stop)
        nil)
   (update-camera-steady-lights :duration (* 0.6 duration))
   (steady-shot :duration (* 0.2 duration))))

(defrule shine-from-object-to-object (object1 object2 (spotlight nil))
  (incremental
   (reset-world-state :duration 0)
   (look-at (list object1 object2) :side :front)
   (init-scene :duration 0)
   (init-lights :duration 0)
   (progn (dim-lights 0.7)
          (setq spotlight (set-spotlight-on object1 :use-light spotlight))
          nil)
   (if (feature :useabstract)
       (level-of-detail-steady-shot
        :objects (all-objects-but (list object1 object2))
```

```
        :level 0 :duration 0))
   (look-at (list object1 object2) :side :front)
   (when (feature :useinvisible)
         (sequential
          (visibility :level 0 :duration 0 :objects
                      (union (objects-between-camera-and-aimpoint)
                             (all-objects-roughly-including object1)
                             :test #'equal))
          (visibility :level 1 :duration 0
                      :objects (list object1 object2))))
   (update-lights-steady-camera :duration (* 0.3 duration))
   (when (feature :useinvisible)
         (sequential
          (visibility :level 0 :duration 0 :objects
                      (union (objects-between-camera-and-aimpoint)
                             (all-objects-roughly-including object2)
                             :test #'equal))
          (visibility :level 1 :duration 0
                      :objects (list object1 object2))))
   (progn (set-spotlight-on object2 :use-light spotlight) nil)
   (update-lights-steady-camera :duration (* 0.7 duration))
   (steady-shot :duration (* 0 duration))))

(defrule move-from-object-to-object (object1 object2)
  (incremental
   (look-at (list object1 object2) :side :front)
   (start-shot :object (list object1 object2)
               :duration (* 0.0 duration))
   (go-from-here-to-object
    :object object1 :fill 0.8 :duration (* 0.2 duration))
   (if (or (feature :usecolor) (feature :useinvisible))
       (flash-objects-steady-shot :objects object1
                                  :duration (* 0.1 duration))
     (steady-shot :duration (* 0.1 duration)))
   (go-from-here-to-object
    :object (list object1 object2) :arrow nil :duration (* 0.2 duration))
   (go-from-here-to-object
    :object object2 :fill 0.8 :duration (* 0.2 duration))
   (if (or (feature :usecolor) (feature :useinvisible))
       (flash-objects-steady-shot
        :objects object2 :duration (* 0.1 duration))
     (steady-shot :duration (* 0.1 duration)))
   (go-from-here-to-object
    :object (list object1 object2) :arrow nil :duration (* 0.2 duration))))

(defrule show-spatial-relation (object1 object2)
  (cond ((and (feature :uselight) (feature :spot-light))
         (shine-from-object-to-object
          :object1 object1 :object2 object2 :duration duration))
        ((and (feature :usedof) (feature :f-stop)
              (feature :focus) (feature :viewangle))
         (focus-dof-from-object-to-object
          :object1 object1 :object2 object2 :duration duration))
        (t
         (move-from-object-to-object
          :object1 object1 :object2 object2 :duration duration))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Show the construction of an object group
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule zoom-out-to-object (object (fill 0.9) (arrow t))
  (incremental
   (sequential
    (look-at object :direction *current-direction* :fill fill)
    (if (and arrow (feature :usemeta) (feature :meta)
             (setq arrow (set-arrow-on object)))
        (create-arrow :arrow arrow :duration 0)
      (setq arrow nil)))
   (update-camera-steady-lights :duration duration)
   (if arrow (destroy-arrow :arrow arrow :duration 0))))

(defrule localize-and-move-around (group)
  (if (groupp group)
      (let* ((parts (gx::object-or-group-elementary-parts group))
             (number (+ (length parts) 2)))
        (incremental
         (localize-object :object group :duration (/ duration number))
         (sequential2
          (loop for part in parts collect
                (incremental
                 (go-from-here-to-object
                  :object part :duration (/ (* 0.4 duration) number))
                 (flash-object-once
                  :object part :color '(1 0 0)
                  :duration (/ (* 0.2 duration) number))
                 (zoom-out-to-object
                  :object (parent-object part)
                  :arrow nil :duration (/ (* 0.4 duration) number))
                 )))
         (localize-object :object group :duration (/ duration number))))
    (localize-object :object group :duration duration)))

(defrule shine-from-here-to-object (object (spotlight nil))
  (set-spotlight-on object :use-light spotlight)
  (sequential
   (update-lights-steady-camera :duration (* 0.6 duration))
   (steady-shot :duration (* 0.4 duration))))

(defrule shine-around-object-group (group spotlight)
  (let* ((parts (gx::object-or-group-elementary-parts group))
         (number (length parts)))
    (sequential2
     (loop for part in parts collect
           (shine-from-here-to-object
            :object part
            :spotlight spotlight
            :duration (/ duration number))))))

(defrule localize-and-shine-around (object (spotlight nil))
  (incremental
   (localize-object :object object :duration (* 0.2 duration))
   (progn (dim-lights 0.7)
          (setq spotlight (set-spotlight-on object))
          (steady-lights :duration 0))
```

```
   (update-lights-steady-camera :duration (* 0.1 duration))
   (shine-around-object-group
    :group object :spotlight spotlight
    :duration (* 0.7 duration))))

(defrule show-group-parts (group)
  (if (and (feature :uselight) (feature :spot-light))
      (localize-and-shine-around :group group :duration duration)
    (localize-and-move-around :group group :duration duration)))
```

Anhang C

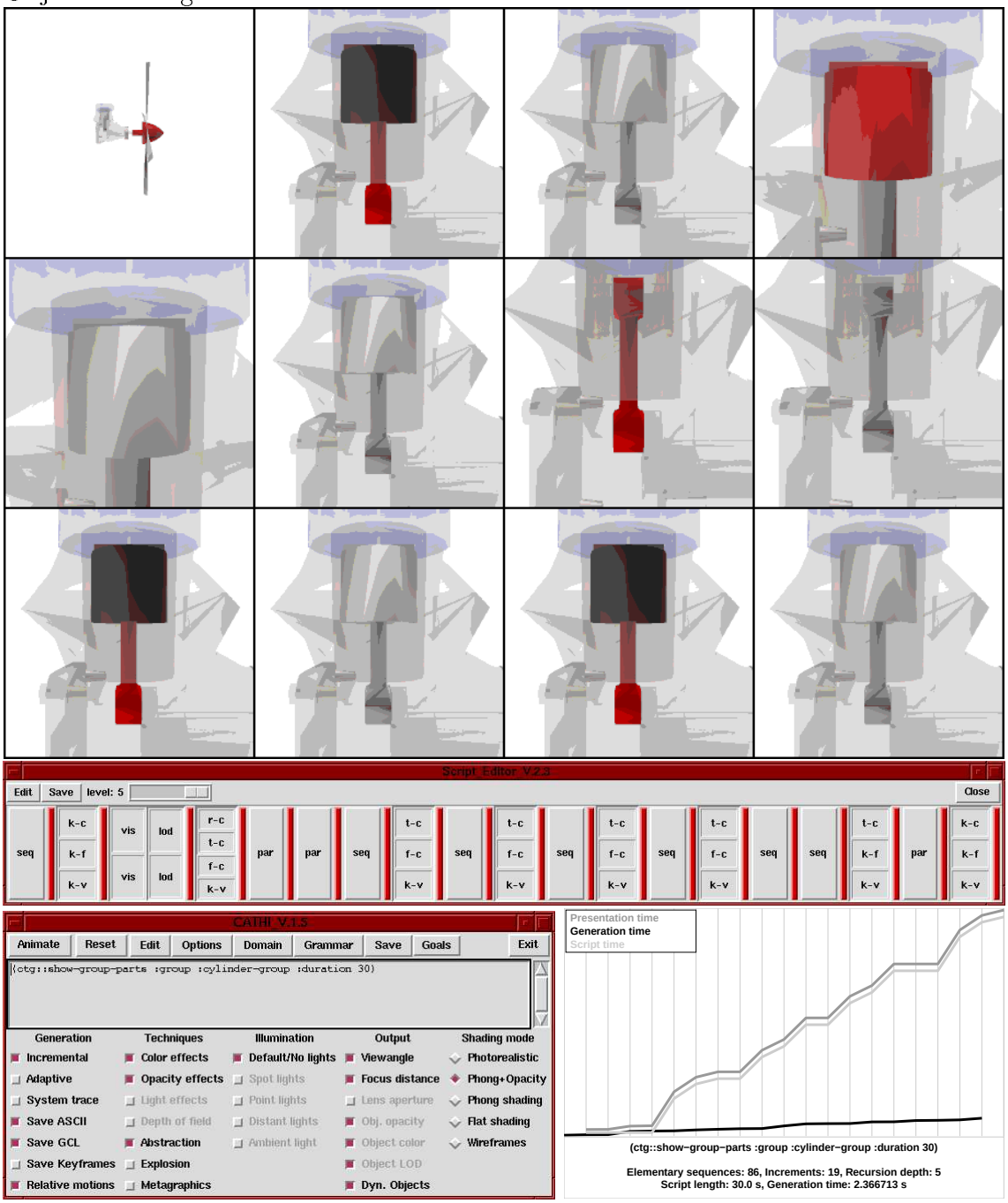
Weitere Generierungsbeispiele des Systems CATHI

Die Bildfolgen in diesem Anhang wurden mit dem System CATHI generiert. Sie enthalten jeweils von links oben nach rechts unten Keyframes der zugehörigen Animation. Diese Art der Darstellung kann den Bewegungseindruck bei Kamerafahrten, bewegten Lichtern und Objektbewegungen allerdings nicht vollständig vermitteln. Einige Beispielanimationen sind deshalb im WWW unter der Adresse

`http://www.dfki.uni-sb.de/~butz/work/examples.html`

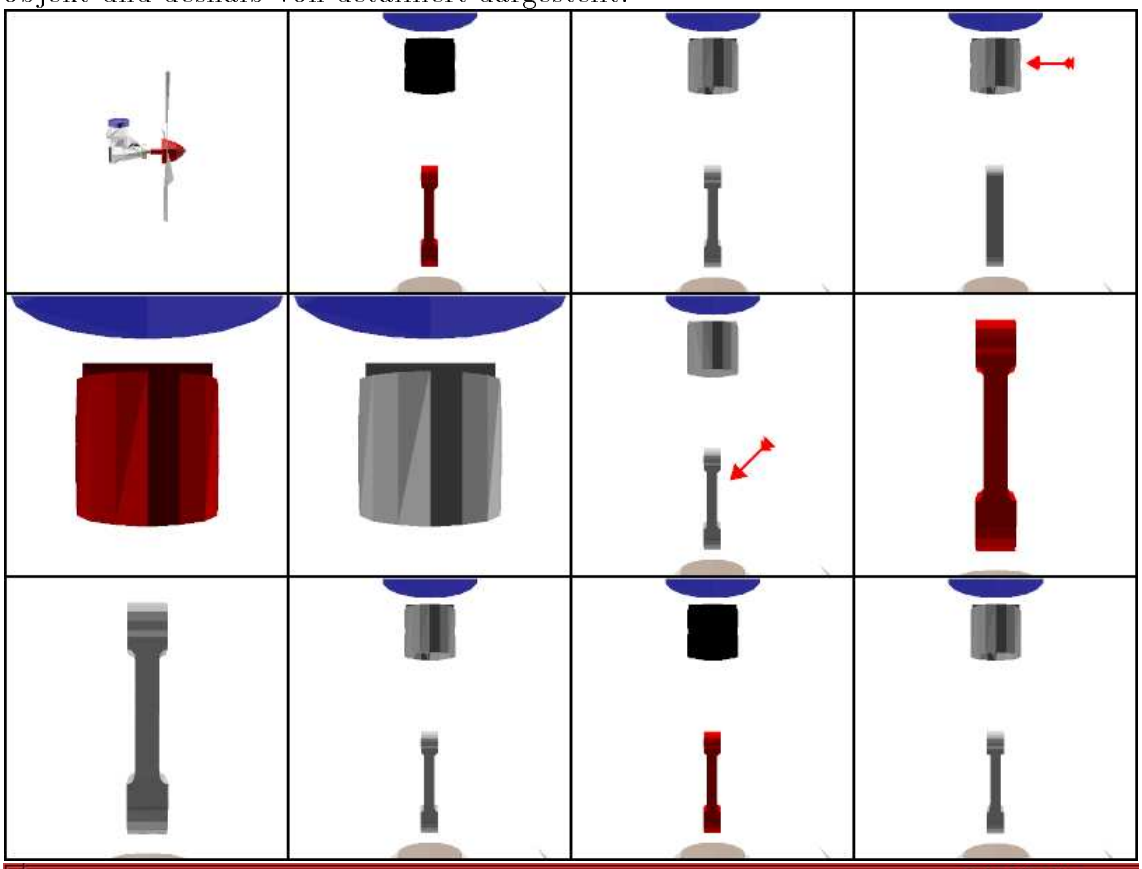
als MPEG-Dateien verfügbar. Zusätzlich zu den Bildfolgen sind auf den nächsten Seiten die eingestellten Generierungsparameter, die Skripte oder Teile davon, sowie ein Timingdiagramm der Generierung abgebildet.

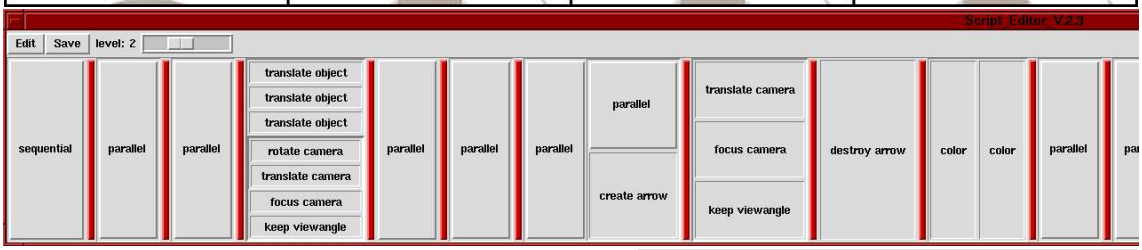
Die erste Animation zeigt die Bestandteile der Baugruppe `cylinder-group` (Siehe Kapitel 8) eines Modellmotors. Da aufgrund der eingestellten Generierungsparameter keine Explosionsdarstellung verwendet werden kann, werden zuerst die verdeckenden Teile durchsichtig gemacht und die Kamera bringt die Baugruppe formatfüllend ins Bild. Zunächst wird die gesamte Baugruppe farblich hervorgehoben, anschließend fährt die Kamera auf die einzelnen Bestandteile zu, die dann ebenfalls ihre Farbe kurz verändern. Zu Ende der Animation kommt wieder die gesamte Gruppe ins Bild und wird nochmals farblich hervorgehoben, um sie von den umgebenden Objekten abzugrenzen.

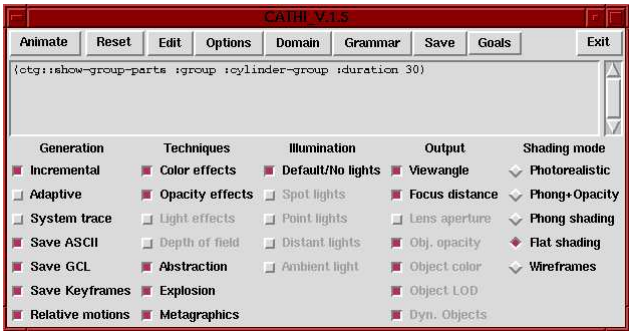


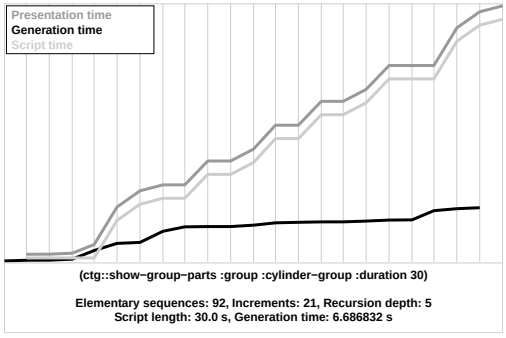
Die nächste Bildfolge wurde aus dem gleichen Animationsziel, jedoch mit anderen Generierungsparametern und für die Darstellung mit einer anderen Ausgabequalität generiert. Zusätzlich zu der farblichen Hervorhebung der Objekte werden metagra-

phische Zeigepfeile eingesetzt und zur Beseitigung von Verdeckungen wird eine Explosionsdarstellung erzeugt. Vom dritten zum vierten Bild ist deutlich zu sehen, wie sich der Abstraktionsgrad des Pleuels verändert, da er vorher als Bestandteil der Baugruppe im Fokus war, sobald der Fokus zum Zylinder selbst wechselt, aber zum Hintergrundobjekt wird. Im siebten Bild wird der Pleuel wieder selbst zum Fokusobjekt und deshalb voll detailliert dargestellt.

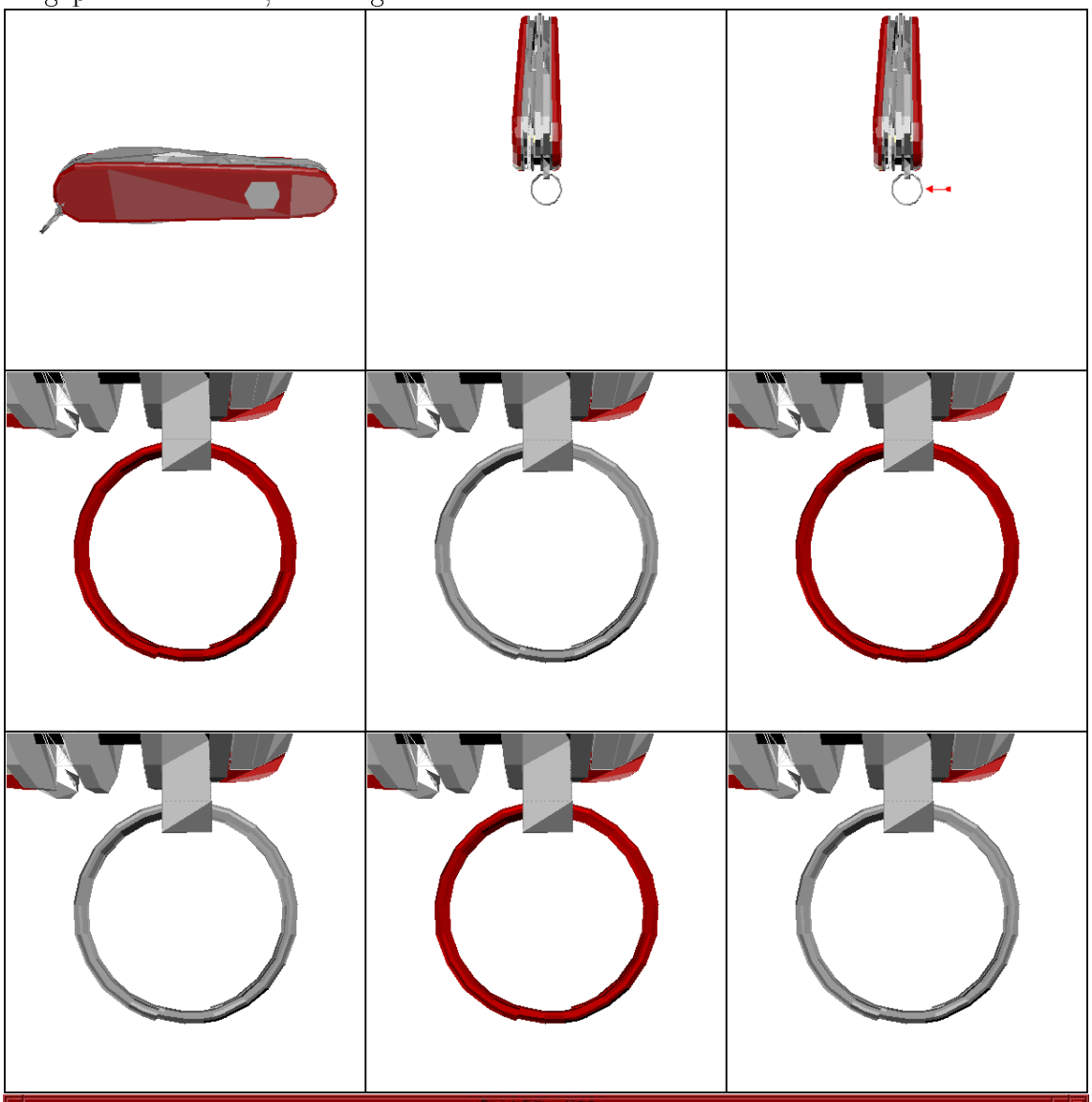


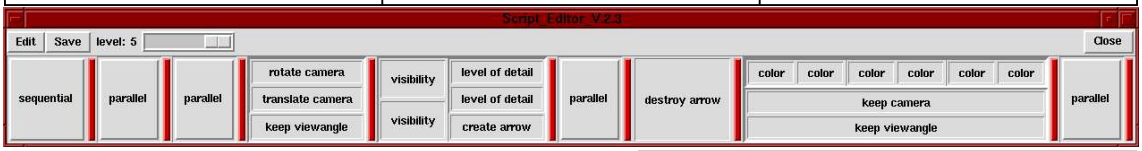


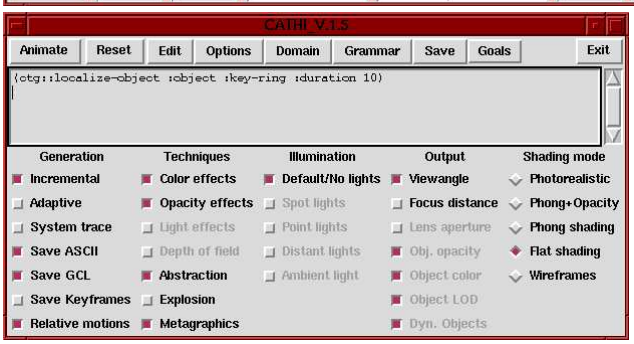


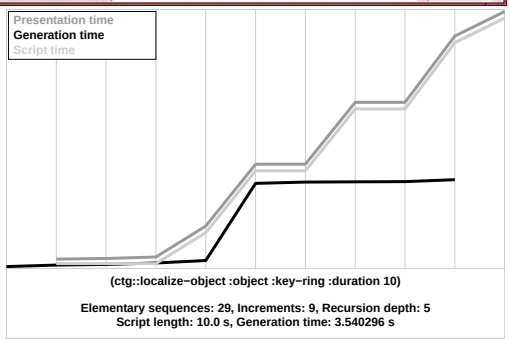


Im dritten Beispiel soll gezeigt werden, wo sich der Schlüsselring eines Schweizer Messers befindet. Hierzu muß die Kamera zunächst auf eine andere Seite der Modellwelt fahren. Anschließend wird der Schüsselring durch einen Pfeil hervorgehoben, die Kamera fährt auf ihn zu und er blinkt dreimal. Die deutliche Stufe in der Generierungszeitkurve im fünften Inkrement kommt durch die Positionierung des Zeigepfeils zustande, die vergleichsweise rechenintensiv ist.

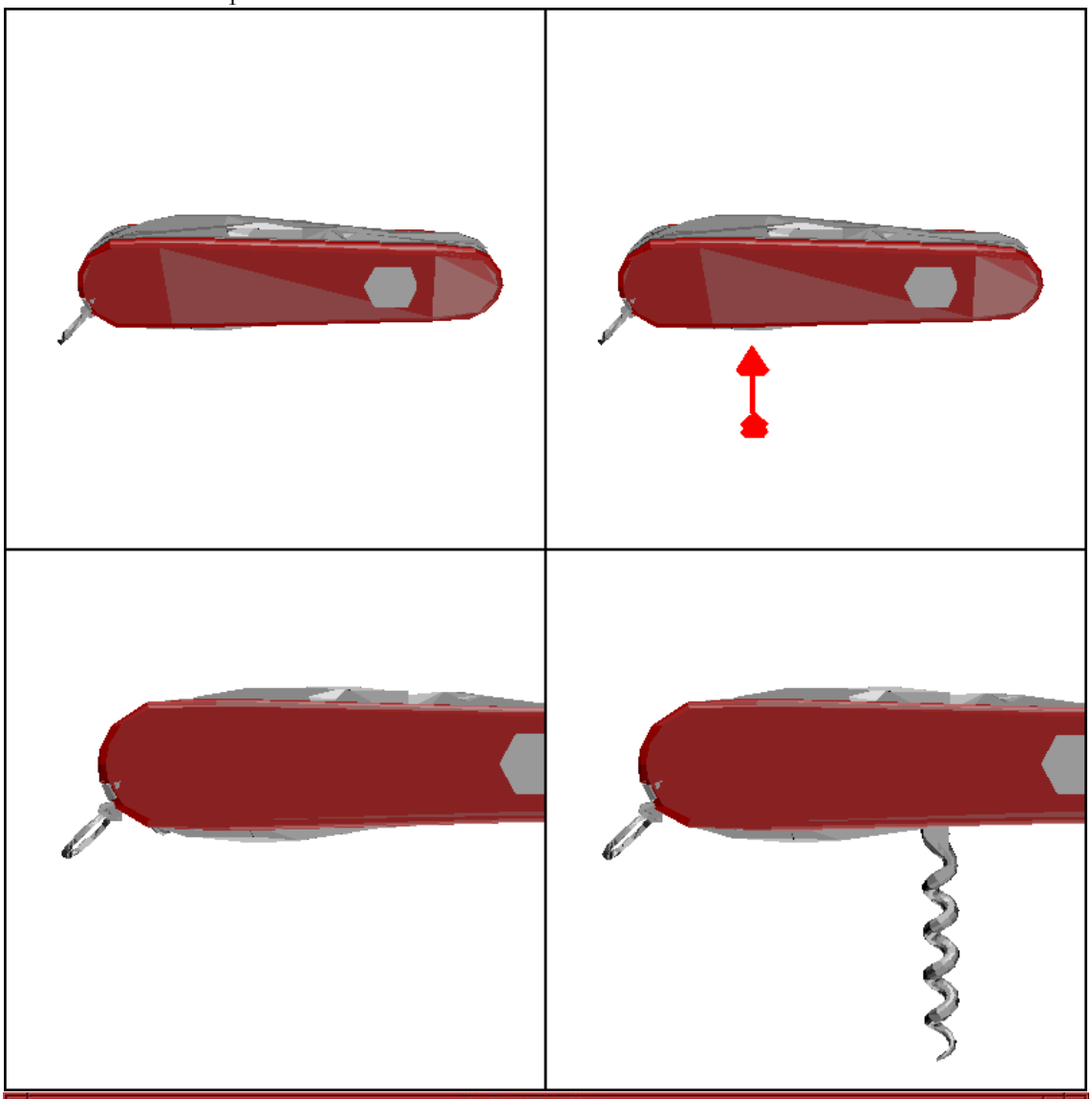


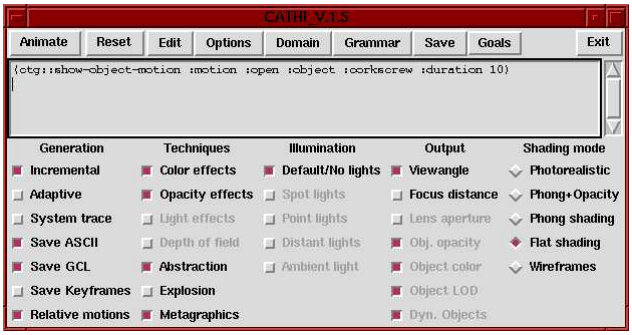


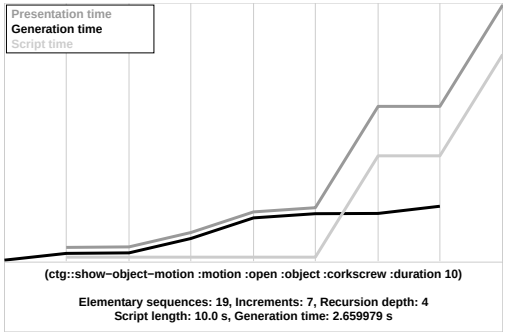


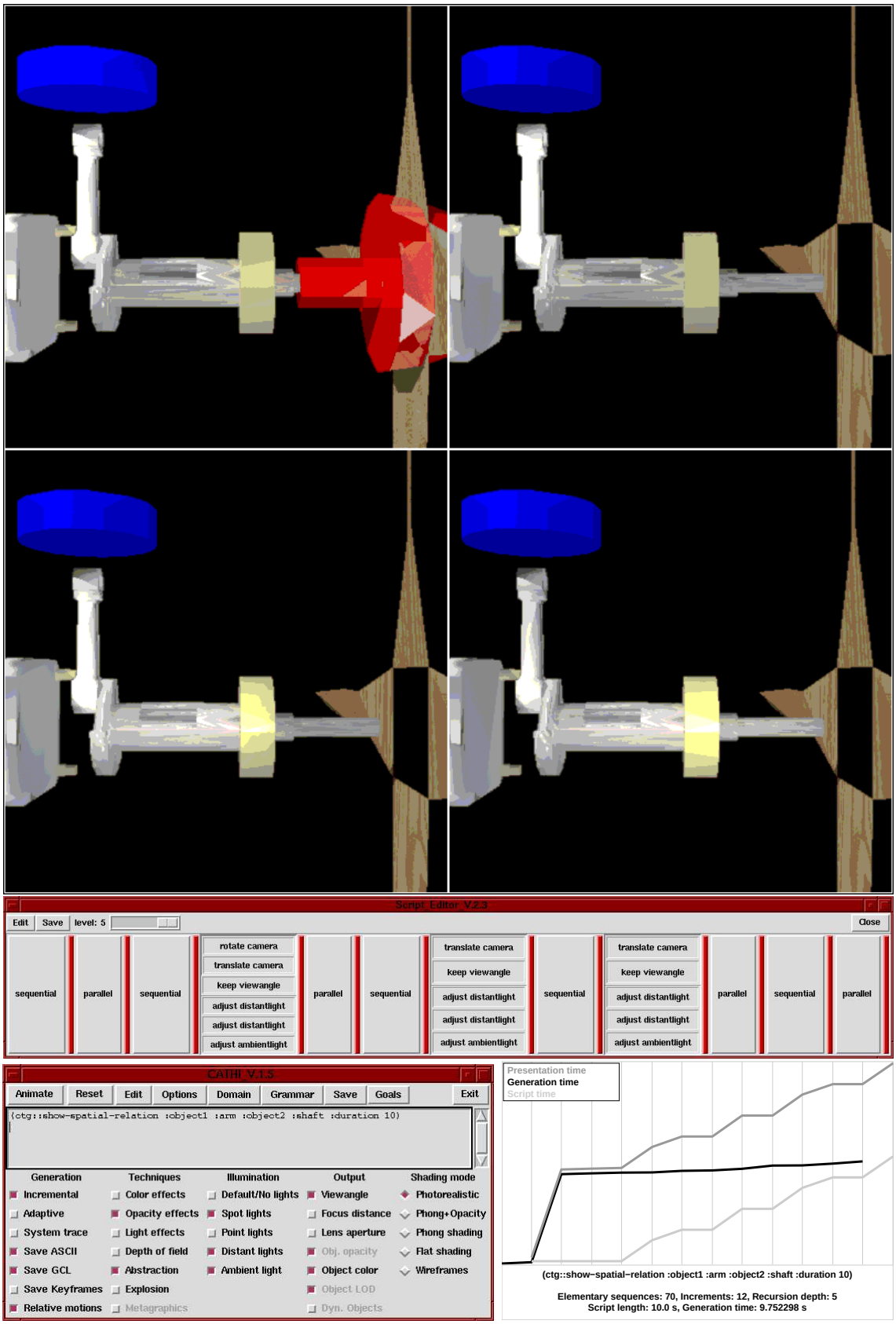


Das Zeigen einer Objektbewegung erfordert eine Kameraposition, bei der das Objekt während seiner gesamten Bewegung möglichst groß zu sehen ist, jedoch den Bildausschnitt nicht verläßt. Dies kann beispielsweise eine Kamerafahrt sein, die das Objekt verfolgt. Bei einfachen Rotationen wie in diesem Beispiel genügt jedoch eine statische Kameraposition.





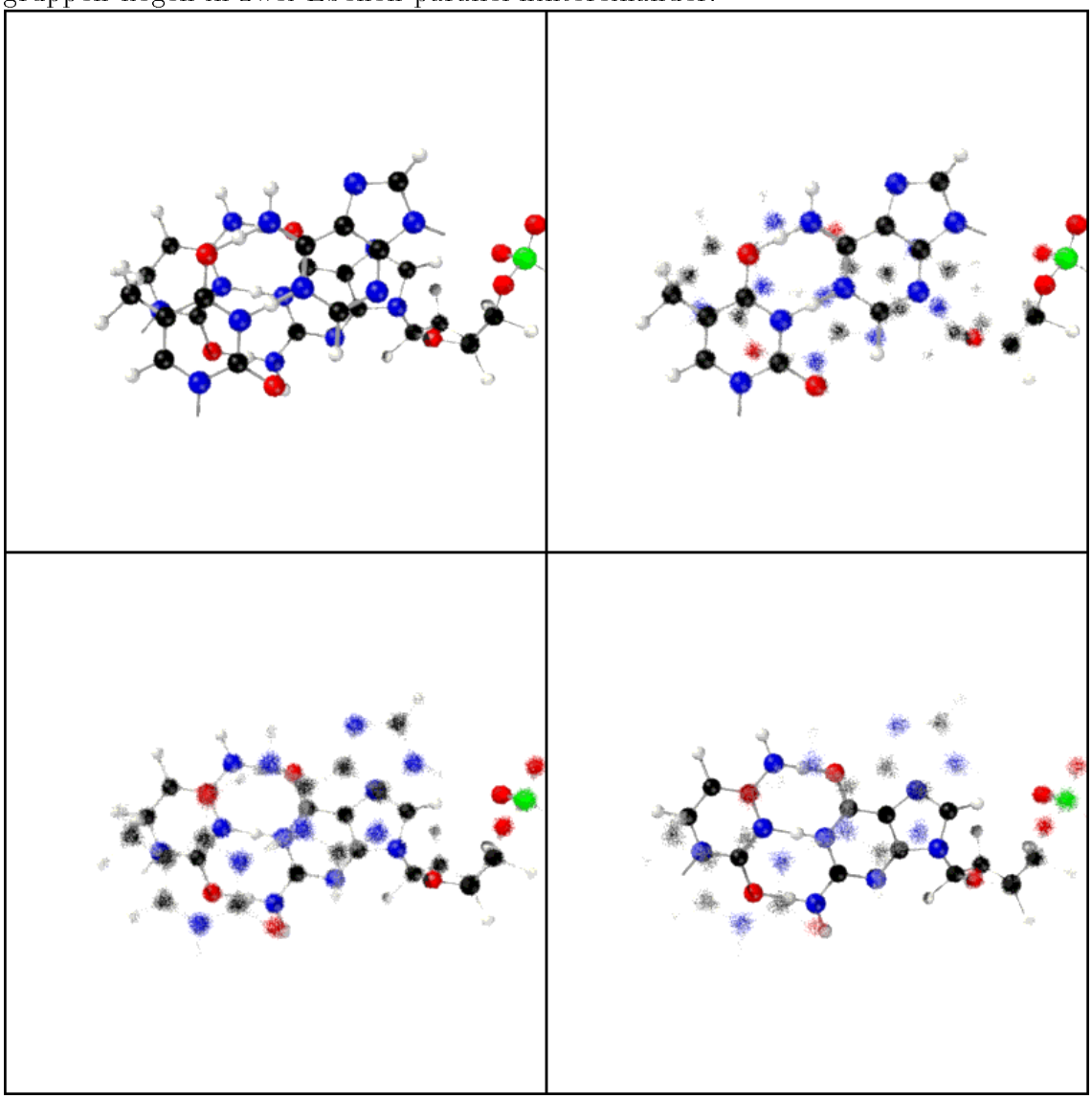




Durch den Einsatz bewegter Spotlichter kann der visuelle Fokus gelenkt werden, ohne daß sich etwas anderes in der Modellwelt verändert. Zunächst wird die Gesamtbeleuchtung zurückgenommen und ein Scheinwerfer auf das erste Objekt (Pleuel) ge-

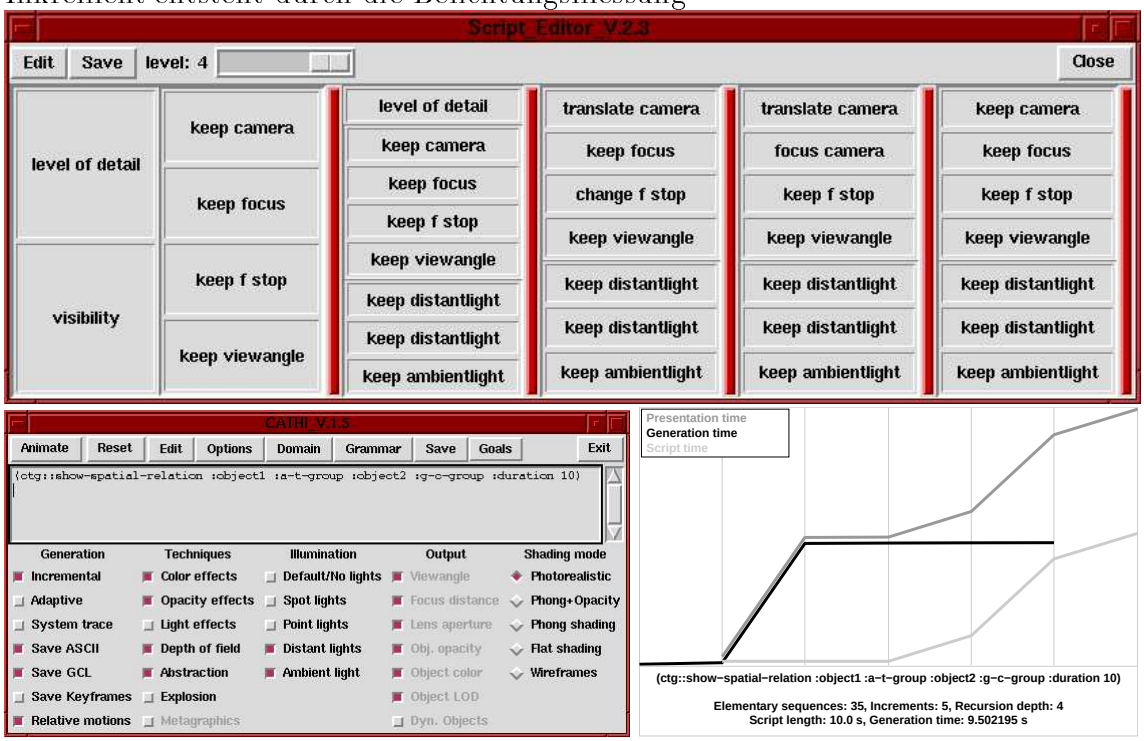
richtet. Dieser Scheinwerfer wandert anschließend zum zweiten Objekt (Achse) und vermittelt hierdurch deren Relation zueinander. Im Timingdiagramm ist zu ersehen, daß die Regelung der Beleuchtung (zweites Inkrement) die bei weitem aufwendigste Berechnung ist.

Ein anderer Effekt, der nur mit hohem computergraphischem Aufwand darstellbar ist, ist das Verlagern der Schärfeebene. Voraussetzung für seine Anwendung ist, daß die Objekte, die in verschiedenen Schärfebereichen liegen sollen, ungleiche Entfernung von der Kamera haben. Im folgenden Beispiel wird die räumliche Lage zweier Teilstrukturen eines DNA-Moleküls zueinander gezeigt. Die beiden Atomgruppen liegen in zwei Ebenen parallel hintereinander.

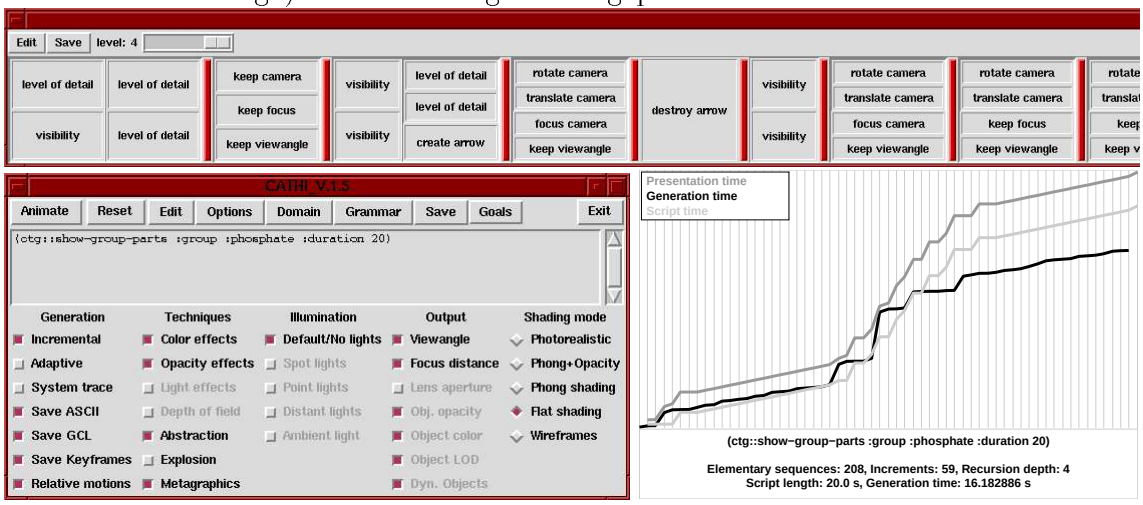


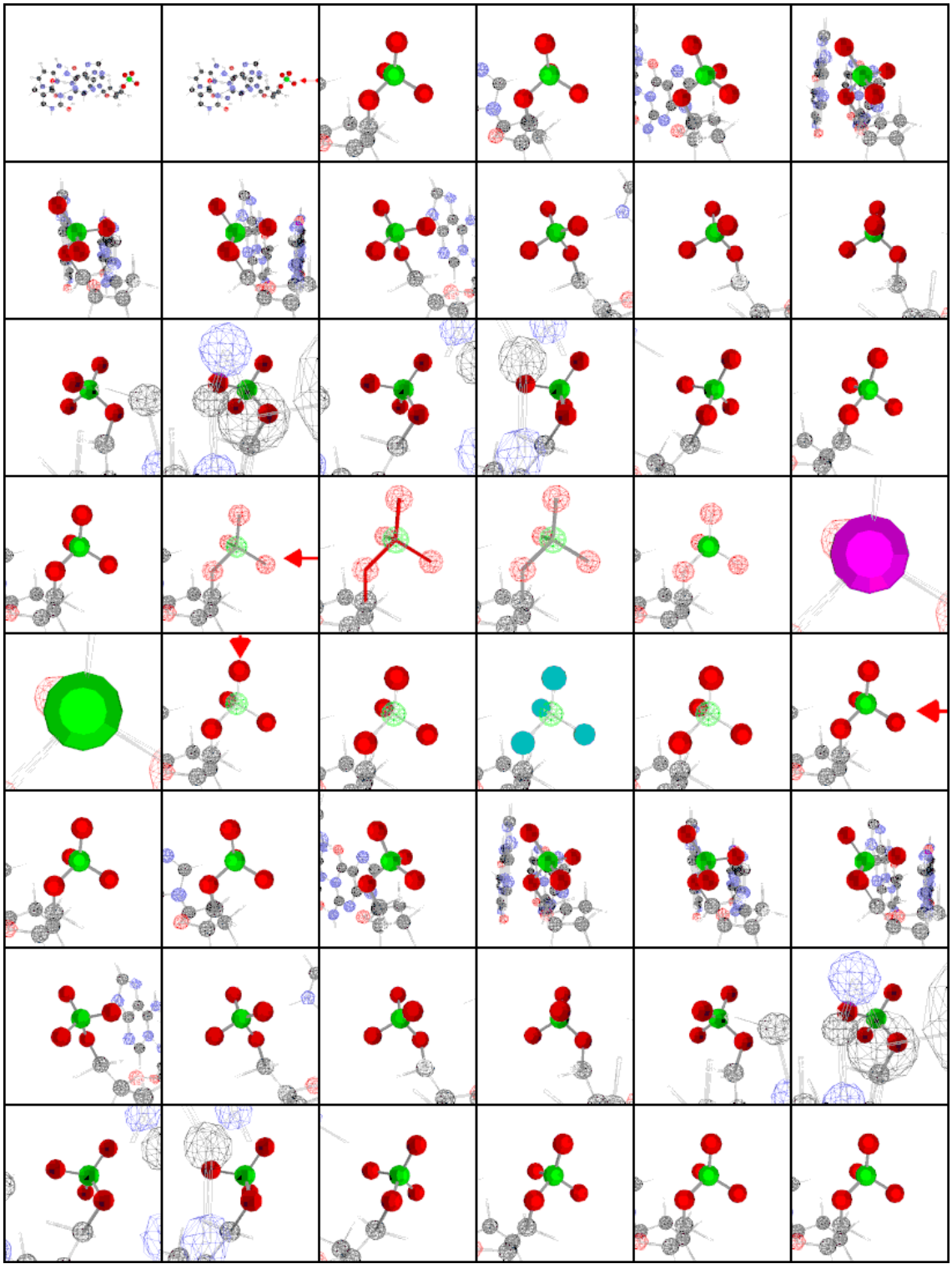
Zunächst wird die Kamerablende weit geschlossen, wodurch die gesamte Modellwelt praktisch vollständig scharf abgebildet wird (erstes Bild). Die eigentliche Schärfeebene liegt jedoch in der vorderen Atomgruppe. Mit dem allmählichen Öffnen der Kamerablende werden die vor und hinter dieser Ebene liegenden Objekte immer unschärfer, bis die vordere Atomkette schließlich völlig aus ihrem Umfeld herausgelöst ist (zweites Bild). Anschließend wird die Schärfeebene bei offener Blende auf die hintere Atomgruppe verlagert, was dazu führt, daß nach einer Phase vollständiger Unschärfe (drittes Bild) schließlich die hintere Atomgruppe scharf dargestellt

wird. Das Timing bei der Generierung dieser Animation ist eigentlich uninteressant, da sich dieser Effekt (derzeit) nicht in Echtzeit darstellen läßt. Die Stufe im zweiten Inkrement entsteht durch die Belichtungsmessung



Das letzte Beispiel schließlich zeigt die Bestandteile der Atomgruppe `phosphate` des DNA-Moleküls. Da diese Gruppe in der Objekthierarchie direkt unter `world` eingetragen ist, erfolgt ihre Lokalisation durch einen Rundflug um die Gruppe. Störende Objekte werden hierbei durchsichtig gemacht. Anschließend werden durch Zeigepfeile und farbliche Hervorhebung die einzelnen Bestandteile fokussiert und die Animation mit einem nochmaligen Rundflug um die gesamte Gruppe abgeschlossen. Die Stufen in der Generierungszeit im Timingdiagramm entstehen durch die (unterschiedlich schwierige) Positionierung der Zeigepfeile.





Literaturverzeichnis

[Adams, 82] Ansel **Adams**. *Das Negativ.* München: Christian Verlag, 1982.

[André et al., 96] Elisabeth **André**, Jochen **Müller**, and Thomas **Rist**. *WIP/PPP: Automatic Generation of Personalized Multimedia Presentations.* ACM Multimedia, pp. 407–408, 1996.

[André, 95] Elisabeth **André**. *Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen*, Band 108: DISKI - Dissertationen zur KI. St. Augustin: infix, 1995.

[Arijon, 76] Daniel **Arijon**. *Grammar of the Film Language.* Samuel French Trade, 7623 Sunset Blvd., Hollywood, CA 90046: Silman-James Press, 1976.

[Badler et al., 90] Norman I. **Badler**, Brian **Barsky**, and David **Zeltzer**. *Making Them Move: Mechanics, Control and Animation of Articulated Figures.* San Mateo: Morgan Kaufmann, 1990.

[Badler et al., 91] Norman I. **Badler**, Bonnie L. **Webber**, J. **Kalita**, and J. **Esakov**. *Animation from Instructions.* In: Norman Badler, Brian Barsky, and David Zeltzer (eds.), Making Them Move: Mechanics, Control and Animation of Articulated Figures, pp. 51–93. San Mateo: Morgan Kaufmann, 1991.

[Badler et al., 93] Norman I. **Badler**, Cary B. **Phillips**, and Bonny Lynn **Webber**. *Simulating Humans. Computer Graphics Animation and Control.* Oxford University Press, 1993.

[Badler et al., 94] Norman I. **Badler**, Ramamani **Bindiganavale**, John P. **Granieri**, Susanna **Wei**, and Xinmin **Zhao**. *Posture interpolation with collision avoidance.* In: Proceedings of Computer Animation '94, Geneva, Switzerland. IEEE Computer society press, 1994.

[Bandyopadhyay, 90] Som **Bandyopadhyay**. *Towards an Understanding of Coherence in Multimodal Discourse.* Technical Memo TM-90-01, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken, 1990.

[Bares & Lester, 97a] William H. **Bares** and James C. **Lester**. *Cinematographic User Models for Automated Realtime Camera Control in Dynamic 3D Environments.* In: Anthony Jameson, Cecile Paris, and Carlo Tasso (eds.), User modeling: Proceedings of the Sixth International Conference, UM97, Vienna, New York, 1997. Springer.

[Bares & Lester, 97b] William H. **Bares** and James C. **Lester**. *Realtime Generation of Customized 3D Animated Explanations for Knowledge-Based Learning Environments.* In: Proceedings of AAAI '97, 1997.

[Brugger, 93] Ralf **Brugger**. *Professionelle Bildgestaltung in der 3D-Computergrafik: Grundlagen und Prinzipien für eine ausdrucksstarke Computervisualisierung.* Bonn: Addison-Wesley (Deutschland), 1993.

[Butz & Krüger, 96] Andreas **Butz** and Antonio **Krüger**. *Lean Modeling - The intelligent use of geometrical abstraction in 3D animations.* In: Wolfgang Wahlster (ed.), Proceedings of ECAI 96, pp. 246–250, Chichester, New York, 1996. John Wiley & Sons,.

[Butz & Krüger, 97] Andreas **Butz** und Antonio **Krüger**. *Zur Auswahl von Abstraktionsgraden.* In: Oliver Deussen und Peter Lorenz (Hrsg.), Simulation und Animation '97, pp. 147–158. SCS, 1997.

[Butz, 94a] Andreas **Butz**. BETTY*: Planning and Generating Animations for the Visualization of Movements and Spatial Relations.* In: Proceedings of the 2nd International Workshop on Advanced Visual Interfaces AVI '94, pp. 53–58. ACM Press, 1994.

[Butz, 94b] Andreas **Butz**. BETTY*: Planung und Generierung von Animationssequenzen zur Visualisierung von Bewegungen und räumlichen Zusammenhängen.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1994.

[Christianson et al., 96] David B. **Christianson**, Sean E. **Anderson**, Li **wei He**, David H. **Salesin**, Daniel S. **Weld**, and Michael F. **Cohen**. *Declarative Camera Control for Automatic Cinematography.* In: Proceedings of AAAI '96, pp. 148–155, 1996.

[DIN, 76] **DIN**. *DIN 6776-1: Technische Zeichnungen; Beschriftung, Schriftzeichen.* Berlin, Wien, Zürich: Beuth Verlag, 1976.

[DIN, 82] **DIN**. *DIN 6774-4: Technische Zeichnungen; Ausführungsregeln; Gezeichnete Vorlagen für Druckzwecke.* Berlin, Wien, Zürich: Beuth Verlag, 1982.

[Drucker & Zeltzer, 95] Steven **Drucker** and David **Zeltzer**. *CamDroid: A System for Implementing Intelligent Camera Control.* In: SIGGRAPH Symposium on Interactive 3D graphics, pp. 139–144, 1995.

[Dunker, 93] Achim **Dunker**. *Die chinesische Sonne scheint immer von unten, Licht- und Schattengestaltung im Film.* München: TR-Verlagsunion, 1993.

[Feiner et al., 93] Steven K. **Feiner**, D.J. **Litman**, Kathleen R. **McKeown**, and R.J. **Passonneau**. *Towards Coordinated Temporal Multimedia Presentations.* In: M. Maybury (ed.), Intelligent Multimedia Interfaces, pp. 139–147. AAAI Press, 1993.

[Feiner, 85] Steven **Feiner**. *APEX: An Experiment in the Automated Creation of Pictorial Explanations.* IEEE Computer Graphics and Applications, 5(11):117–123, 1985.

[Field, 91] Syd **Field**. *Das Handbuch zum Drehbuch.* Frankfurt am Main: Zweitausendeins Verlag, 1991.

[Finkler, 96] Wolfgang **Finkler**. *Automatische Selbstkorrektur bei der inkrementellen Generierung gesprochener Sprache unter Realzeitbedingungen.* DISKI - Dissertationen zur KI. St. Augustin: infix, 1996.

[Foley et al., 96] James D. **Foley**, Andries **van Dam**, John F. **Hughes**, and Steven K. **Feiner**. *Computer Graphics: Principles and Practice. 2nd Edition.* Reading, Massachusetts: Addison-Wesley, 1996.

[Graf, 96] Winfried H. **Graf**. *Intentionsgesteuertes Layout-Design multimedialer Präsentationen mit Constraints.* DISKI – Dissertationen zur Künstlichen Intelligenz. Sankt Augustin: infix, 1996. Im Druck.

[Gritz, 97] Larry **Gritz**. *Blue Moon Rendering Tools Home Page.* Im WWW unter der Adresse http://www.seas.gwu.edu/student/gritz/bmrt.html: , 1997.

[He et al., 96] Li-wei **He**, Michael F. **Cohen**, and David H. **Salesin**. *The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing.* In: Proceedings of SIGGRAPH '96, pp. 217–224, 1996.

[Itten, 87] Johannes **Itten**. *Kunst der Farbe.* Ravensburg: Ravensburger Buchverlag Otto Maier, 1987.

[Karp & Feiner, 90] Peter **Karp** and Steven **Feiner**. *Issues in the Automated Generation of Animated Presentations.* In: Graphics Interface '90, pp. 39–48, 1990.

[Karp & Feiner, 93] Peter **Karp** and Steven **Feiner**. *Automated Presentation Planning of Animation Using Task Decomposition with Heuristic Reasoning.* In: Graphics Interface '93, pp. 17–21, 1993.

[Krüger, 95] Antonio **Krüger**. *PROXIMA: Ein System zur Generierung graphischer Abstraktionen.* Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken, 1995.

[Krüger, 98] Antonio **Krüger**. *Graphische Abstraktion – Durchblick statt Durcheinander.* Dissertation, Universität des Saarlandes, Saarbrücken, Saarbrücken, 1998. in Vorbereitung.

[Kurlander & Ling, 95] David **Kurlander** and Daniel T. **Ling**. *Planning-Based Control of Interface Animation.* In: Proceedings of CHI'95 (Denver, May 7-11), pp. 472–479, 1995.

[Li, 96] Ming **Li**. *Ein System zur automatischen Generierung von Explosionszeichnungen.* Dissertation, Universität des Saarlandes, Saarbrücken, 1996.

[Maass, 96] Wolfgang **Maass**. *Von visuellen Daten zu inkrementellen Wegbeschreibungen in dreidimensionalen Umgebungen: Das Modell eines kognitiven Agenten.* Dissertation, Universität des Saarlandes, Saarbrücken, 1996.

[Mackinlay, 86] Jock D. **Mackinlay**. *Automatic Design of Graphical Presentations.* PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1986.

[Mann & Thompson, 87a] W. C. **Mann** and S. A. **Thompson**. *Rhetorical Structure Theory: A Theory of Text Organization.* Report ISI/RS-87-190, Univ. of Southern California, Marina del Rey, CA, 1987.

[Mann & Thompson, 87b] W. C. **Mann** and S. A. **Thompson**. *Rhetorical Structure Theory: Description and Construction of Text Structures.* In: G. Kempen (ed.), Natural Language Generation: New Results in Artificial Intelligence, Psychology, and Linguistics, pp. 85–95. Dordrecht: Martinus Nijhoff Publishers, 1987.

[Nikon, 96] **Nikon** (ed.). *F5, Imported from the Future.* 1300 Walt Whitman Road, Melville, N.Y. 11747-3064, U.S.A.: Nikon Inc., 1996.

[Nimeroff et al., 95] Jeffry **Nimeroff**, Julie **Dorsey**, and Holly **Rushmeier**. *A Framework for Global Illumination in Animated Environments.* In: 6th Annual Eurographics Workshop on Rendering, Dublin Ireland, 1995.

[Phillips, 96] Mark **Phillips**. *Geomview Manual.* Zusammen mit der Software unter der Adresse http://www.geom.umn.edu/, 1996.

[Preim et al., 96] Bernhard **Preim**, Alf **Ritter**, und Gunnar **Steinicke**. *Gestaltung von Animationen zur Erklärung komplexer 3D-Modelle.* In: Proceedings Simulation und Animation für Planung, Bildung und Präsentation, Magdeburg, 29.2.–1.3. 1996, pp. 255–266, 1996.

[Rist et al., 97] T. **Rist**, E. **André**, and J. **Müller**. *Adding Animated Presentation Agents to the Interface.* In: Proceedings of the 1997 International Conference on Intelligent User Interfaces, pp. 79–86, Orlando, Florida, 1997.

[Rist, 95] Thomas **Rist**. *Wissensbasierte Verfahren für den automatischen Entwurf von Gebrauchsgraphik in der technischen Dokumentation.* DISKI - Dissertationen zur KI. St. Augustin: infix, 1995.

[Rossignac & Borrel, 93] Jarek **Rossignac** and Paul **Borrel**. *Multi-Resolution 3D Approximations for Rendering Complex Scenes.* Technical Report RC 17697, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, 1993.

[Schneider & Krüger, 93] Georg **Schneider** und Antonio **Krüger**. *TOPAS: Eine Werkbank zur Erzeugung von 3D-Illustrationen.* In: A. Iwainsky (Hrsg.), Workshop auf der Wartburg: Computergrafik und automatisierte Layoutsynthese, pp. 115–133. Gesellschaft für Informatik (GI), 1993.

[Schneider, 95] Georg **Schneider**. *Eine Werkbank zur Erzeugung von 3D-Illustrationen.* Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, Prof. Dr. Wolfgang Wahlster, 1995.

[Schult & Buchholz, 93] Gerhard **Schult** und Axel **Buchholz** (Hrsg.). *Fernsehjournalismus, Ein Handbuch für Ausbildung und Praxis.* München, Leipzig: List, 1993.

[SGI, 96] **SGI**. *VRML 2.0 Specification by Silicon Graphics Inc.* Im WWW unter der Adresse http://vrml.sgi.com/moving-worlds/index.html, 1996.

[Upstill, 90] Steve **Upstill**. *The RenderMan companion.* Reading, Massachusetts: Addison-Wesley, 1990.

[Watt & Watt, 92] Alan **Watt** and Mark **Watt**. *Advanced Animation and Rendering Techniques, Theory and Practice.* Reading, Massachusetts: Addison-Wesley, 1992.

[Webber et al., 93] Bonnie **Webber**, Norman **Badler**, F. **Baldwin**, W. **Becket**, B. Di **Eugenio**, C. **Geib**, M. **Jung**, L. **Levison**, M. **Moore**, and M. **White**. *Doing what you're told: Following task instructions in changing but hospitable environments.* In: Y. Wilks and N. Okada (eds.), Language and Vision across the Pacific., 1993.

[Zeltzer, 90] David **Zeltzer**. *Task-level Graphical Simulation: Abstraction, Representation, and Control.* In: Norman Badler, Brian Barsky, and David Zeltzer (eds.), Making Them Move: Mechanics, Control and Animation of Articulated Figures, pp. 3–33. San Mateo: Morgan Kaufmann, 1990.

Stichwortverzeichnis

Seitenzahlen in Fettdruck verweisen auf Definitionen.